დისკრეტული სტრუქტურები
(ალგორითმული მიდგომა)

ალექსანდრე გამყრელიძე

სარჩევი

შესავალი

ჩვენს ყოველდღიურ ცხოვრებაში ალგორითმები უაღრესად დიდ როლს თამაშობენ ისე, რომ ჩვენ ამას ვერც კი ვამჩნევთ. უფრო მეტიც, ბევრმა არც კი იცის, თუ რა არის ალგორითმი. არა და ალგორითმები ყოველ ფეხის ნაბიჯზე გხვდება, ამ სიტყვის პირდაპირი მნიშვნელობით – ადამიანის სიარული გარკვეული თვალსაზრისით ალგორითმია: მარცხენა ფეხი გადადგი წინ, ტანი გადახარე ოდნავ წინ, მარჯვენა ფეხი გადაგი წინ და ეს პროცესი თავიდან გაიმეორე მანამ, სანამ სიარულის შეწყვეტა მოგინდება. სხვათა შორის, ეს ცენტრალური ალგორითმია რობოტოტექნიკაში და დღეისათვის ბოლომდე გადაჭრილი არ არის – როგორც აღმოჩნდა, ასეთი ერთი შეხედვით მარტივი ალგორითმის რეალიზაცია ძალიან რთულია.

მეორე მაგალითად ფშავური ხინკლის გაკეთების ალგორითმი შეიძლება მოვიყვანოთ:

<u>მონაცემები:</u>

ხორცი, ხახვი, რეჰანი, ქონდარი, წითელი წიწაკა, პილპილი, მარილი, ფქვილი

<u>ალგორითმის მუშაობის შედეგი:</u> ფშავური ხინკალი

<u>ალგორითმის მუშაობის აღწერა:</u>

ალგორითმი „ფშავური ხინკალი"

მონაცემები: ხორცი, ხახვი, რეჰანი, ქონდარი, წითელი წიწაკა, პილპილი, მარილი, ფქვილი, წყალი

1. გააკეთე ბულიონი: ძვლები ჩაყარე ქვაბში, დაასხი იმდენი წყალი, რომ დაიფაროს და ნელ ცეცხლზე ადუღე. როცა გასინჯავ და უკვე წყალ-წყლა აღარ იქნება, გადმოდგი და გაატარე წვრილ ბადეში, რომ ძვლების ნარჩენები არ შეჰყვეს. ამის შემდეგ გააცივე და გვერდზე გადადგი.

2. ხორცი, წიწაკა, ხახვი, რეჰანი და ქონდარი ცალ-ცალკე წვრილდ აკეპე.

3. ხორცს დაასხი მარილითა და წიწაკით გაზავებული ნელ-თბილი ბულიონი და აზილე. შემდეგ კიდევ დაასხი და აზილე. ეს პროცედურა გაიმეორე მანამ, სანამ ბულიონს არ შეიწოვს და თავზე კიდევ ცოტა არ დადგება.

4. არსებულ ფარშს შეურიე დარჩენილი ხახვი, წიწაკა, პილპილი და მწვანილი (გემოვნებით).

5. შემდეგ აიღე ზუსტად იმდენივე ბულიონი, რამდენიც დაჭირდა ხორცს და შეურიე მარილი ისე, რომ სიმლაშე საკმაოდ ეტყობოდეს. ამ ბულიონით მოზილე საკმაოდ მაგარი ცომი.

6. დაადგი ბევრი მარილწყალი ძალიან მაღალ ცეცხლზე.

7. ცომიდან ჩამოჭერი მოგრძო ნაჭერი, თოკივით დაამრგვალე და დაჭერი პატარა ნაჭრებად. ეს ნაჭრები ცალ-ცალკე გააბრტყელე თხელ, მრგვალ დისკებად. კოვზით აიღე ფარში, ცომის დისკებზე დადე და გაახვიე.

8. შემდეგ ჩაყარე მდუღარე, მარილიან წყალში და დაახლოებით 10 - 12 წთ. ხარშე.

ალგორითმი დასრულებულია

ზემოთ მოყვანილ ხინკლის ალგორითმში შემდეგი რამ არის გასათვალისწინებელი: „ცომის მოზელვის" პროცესი თავის მხრივ ალგორითმია, რომელიც პერიოდულად უნდა გაგრძელდეს მანამ, სანამ ცომი სასურველ კონსისტენციას არ მიაღწევს (ასეთივე რამ შეიძლება ითქვას ხორცის აკეპვისა და ხინკლის გახვევის პროცედურებზეც). ესე იგი, აქ ჩართულია კიდევ შემოწმების მექანიზმი: თუ კონსისტენცია კარგია, მაშინ ალგორითმი დაასრულე. თუ არა, იგივე გაიმეორე.

ზოგადად, ალგორითმი რაიმე ამოცანის გადაჭრის გზაა, მაგრამ ამ გადაჭრისას უნდა გავითვალისწინოთ შემდეგი სამი პუნქტი:

1. ალგორითმი უნდა შედგებოდეს ერთი ან რამოდენიმე ბიჯისაგან;
2. როცა ალგორითმი ერთი ბიჯის შესრულებას დაასრულებს, იგი შემდგომი ბიჯის შესრულებაზე უნდა გადავიდეს;
3. ბიჯები შეიძლება პერიოდულად გამეორდეს, მაგრამ საერთო ჯამში ყოველი ალგორითმის ბიჯების საერთო რაოდენობა სასრული უნდა იყოს – ალგორითმი როდესღაც უნდა გაჩერდეს.

ალგორითმებში მნიშვნელოვანია ორი ასპექტი:

1. სისწორე – ეს ალგორითმი მართლა იმას აკეთებს, რაც მოეთხოვება?
2. სისწრაფე – რამდენ ბიჯს ანდომებს ალგორითმი დაწყებიდან დამთავრებამდე?

ჩვენს ირგვლივ ძალიან ბევრი ამოცანა არსებობს: ხინკლის მოხარშვიდან დაწყებული და კოსმოსში რაკეტების გაგზავნით დამთავრებული. ბუნებრივად წამოიჭრა შეკითხვა: შეიძლება თუ არა ყველა ამოცანა ალგორითმულად გადაიჭრას (ანუ შეიძლება თუ არა ყველა ამოცანისათვის დავწეროთ ალგორითმი, რომელიც მას ამოხსნის)? როგორც აღმოჩნდა, არსებობს ისეთ ამოცანათა სიმრავლე, რომლებსაც ალგორითმულად ვერ ამოვხსნით. უფრო მეტიც – გაცილებით მეტია ისეთი ამოცანები, რომლებსაც ალგორითმულად ვერ ამოვხსნით, ვიდრე ისეთები, რომლებსაც შეიძლება მოვუგონოთ ალგორითმი. ეს კი იმას ნიშნავს, რომ ადამიანის ცხოვრებაში გაცილებით მეტი რამ არის ისეთი, რომელსაც კომპიუტერი ვერ ამოხსნის, ვიდრე ისეთი, რომელსაც „ხელოვნური ინტელექტი" დაძლევს.

როგორც აღმოჩნდა, ალგორითმულად ამოხსნად ამოცანებს შორისაც არსებობს ისეთი ამოცანები, რომელთა დღეისათვის ცნობილი ალგორითმებით ამოხსნაც ძალიან დიდ დროს მოითხოვს, ანუ უმეტეს შემთხვევებში ჩვენს ხელთ არსებული უძლიერესი გამომთვლელი მანქანებით ასობით ათას წელს მოანდომებდა – ბიჯების რაოდენობა ძალიან სწრაფად იზრდება. მაგრამ მთავარი აქ ისაა, რომ არ არის ცნობილი, შეიძლება თუ არა ასეთი ამოცანებისათვის დაიწეროს ისეთი ალგორითმი, რომელიც უფრო სწრაფი იქნებოდა.

როდესაც წამოიჭრება ახალი ამოცანა, პირველ რიგში უნდა დავადგინოთ, შეიძლება თუ არა მისი ალგორითმულად ამოხსნა. თუ არ შეიძლება, მაშინ უნდა დავადგინოთ, როგორ შევცვალოთ ამ ამოცანის პირობები ისე, რომ იგი ამოხსნადი გახდეს და, ამავდროულად, რაც შეიძლება ახლოს იყოს ამ დასმულ ამოცანასთან.

თუ ამოცანა ამოხსნადია, უნდა დავადგინოთ, შეიძლება თუ არა მისი სწრაფად ამოხსნა? თუ არ შეიძლება, მაშინ უნდა დავადგინოთ, როგორ შევცვალოთ ამ ამოცანის პირობები ისე, რომ იგი ამოხსნადი გახდეს და, ამავდროულად, რაც შეიძლება ახლოს იყოს ამ დასმულ ამოცანასთან (ევრისტიკების შექმნა) ან ისეთი სწრაფი ალგორითმი შევქმნათ, რომელიც ზუსტად იმავე მონაცემებზე და პირობებში ზუსტ პასუხთან მიახლოვებულ პასუხს მოგვცემს (მიახლოებითი ალგორითმები).

მაგრამ თუ სწრაფი ალგორითმის შექმნა შესაძლებელია, როგორ შევქმნათ ოპტიმალური ალგორითმი, ანუ ისეთი, რომ მასზე სწრაფი ალგორითმი არ არსებობდეს?

ამ საკითხების გარკვევაში გვეხმარება თეორიული ინფორმატიკის ერთ-ერთი განხრა – ალგორითმების თეორია, რომლის შესავალსაც ჩვენ აქ განვიხილავთ.

თავი 1

ალგორითმების მარტივი მაგალითები

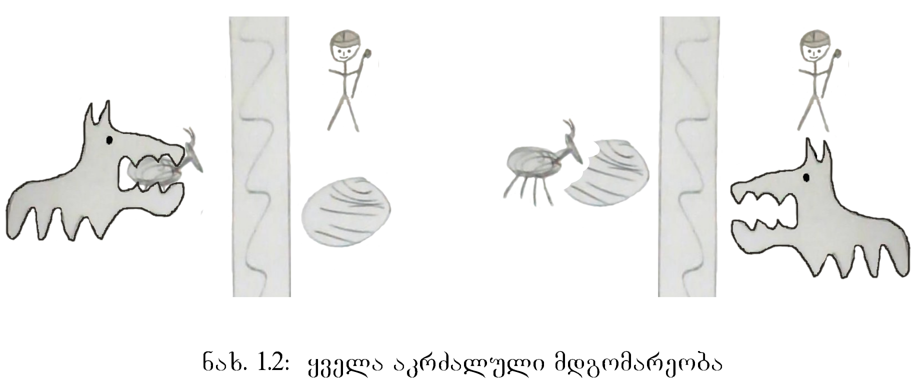
1.1 მგელი, თხა და კომბოსტო

განვიხილოთ ბევრისათვის კარგად ცნობილი ამოცანა მგლის, თხისა და კომბოსტოს შესახებ:

მდინარის ერთ ნაპირზე იმყოფებიან ადამიანი, მგელი, თხა და კომბოსტო (ნახ. 1.1). ადამიანს აქვს ნავი, რომელშიც ეტევა მხოლოდ იგი და ერთი რომელიმე სხვა მგზავრი: მგელი, თხა ან კომბოსტო.

ნახ. 1.1: ამოცანის საწყისი და საბოლოო პირობები

სანამ ადამიანი სხვა ცხოველებთან ერთადაა ნაპირზე, ისინი კარგად იქცევიან და ერთმანეთს არ დაერევიან. მაგრამ საკმარისია მან მარტო დატოვოს ერთ ნაპირზე თხა და მგელი, რომ ეს უკანასკნელი თხას ეტაკება. თვით თხა კი მარტო დარჩენილ კომბოსტოს შეჭამს.
თუ მგელი კომბოსტოთი დარჩება ერთ ნაპირზე მარტო, არაფერი არ მოხდება.



ნახ. 1.2: ყველა აკრძალული მდგომარეობა

ამოცანა მდგომარეობს შემდეგში: დაწერეთ ალგორითმი, რომლის მეშვეობითაც ადამიანი თავისი ნავით სამივეს გადაიყვანს მეორე ნაპირზე.

პირველ რიგში უნდა ჩამოვაყალიბოთ ამოცანა: მოცემულობა, საბოლოო შედეგი და ალგორითმის მსვლელობისას დადებული შეზღუდვები.

მოცემულია: მდინარე და მის ერთ ნაპირზე მყოფი ნავი, ადამიანი, მგელი, თხა და კომბოსტო (ნახ. 1.1 მარცხნივ).

შედეგი: ეს ყველა მეორე ნაპირზე ერთად მყოფი (ნახ. 1.1 მარჯვნივ).

შეზღუდვა: ცხოველები გადაჰყავს ადამიანს ორ ადგილიანი ნავით (პირველი შეზღუდვა – ნავში უნდა იჯდეს ადამიანი, რომელსაც მხოლოდ ერთი ადგილი რჩება თავისუფალი და, აქედან გამომდინარე, მეორე ნაპირზე ერთ ჯერზე შეუძლია გადაიყვანოს ან მხოლოდ მგელი, ან მხოლოდ თხა, ან მხოლოდ კომბოსტო). მგლისა და კუდღლის მარტო დატოვება არ შეიძლება, ასევე არ შეიძლება თხისა და კომბოსტოს მარტო დატოვება (მეორე და მესამე შეზღუდვა).

ამ ამოცანის ამოსახსნელად შეიძლება გამოვიყენოთ შემდეგი ალგორითმი, რომლის ყოველი ბიჯის წარმოდგენილია ნახ. 1.3-ში (დავუშვათ, რომ დასაწყისში ყველა მდინარის მარცხენა ნაპირზეა და ბოლოს მარჯვენა ნაპირზე უნდა იყოს):

ალგორითმი 1.1: „მგელი, თხა და კომბოსტო"
მონაცემები: მდინარე და მის მარცხენა ნაპირზე ადამიანი, ნავი, მგელი, თხა და კომბოსტო;

მარჯვენა ნაპირზე გადაიყვანე თხა ;
დაბრუნდი მარცხენა ნაპირზე ;
მარჯვენა ნაპირზე გადაიყვანე მგელი ;
მარცხენა ნაპირზე გადაიყვანე თხა ;
მარჯვენა ნაპირზე გადაიტანე კომბოსტო ;
დაბრუნდი მარცხენა ნაპირზე ;
მარჯვენა ნაპირზე გადაიყვანე თხა .
ალგორითმი დასრულებულია

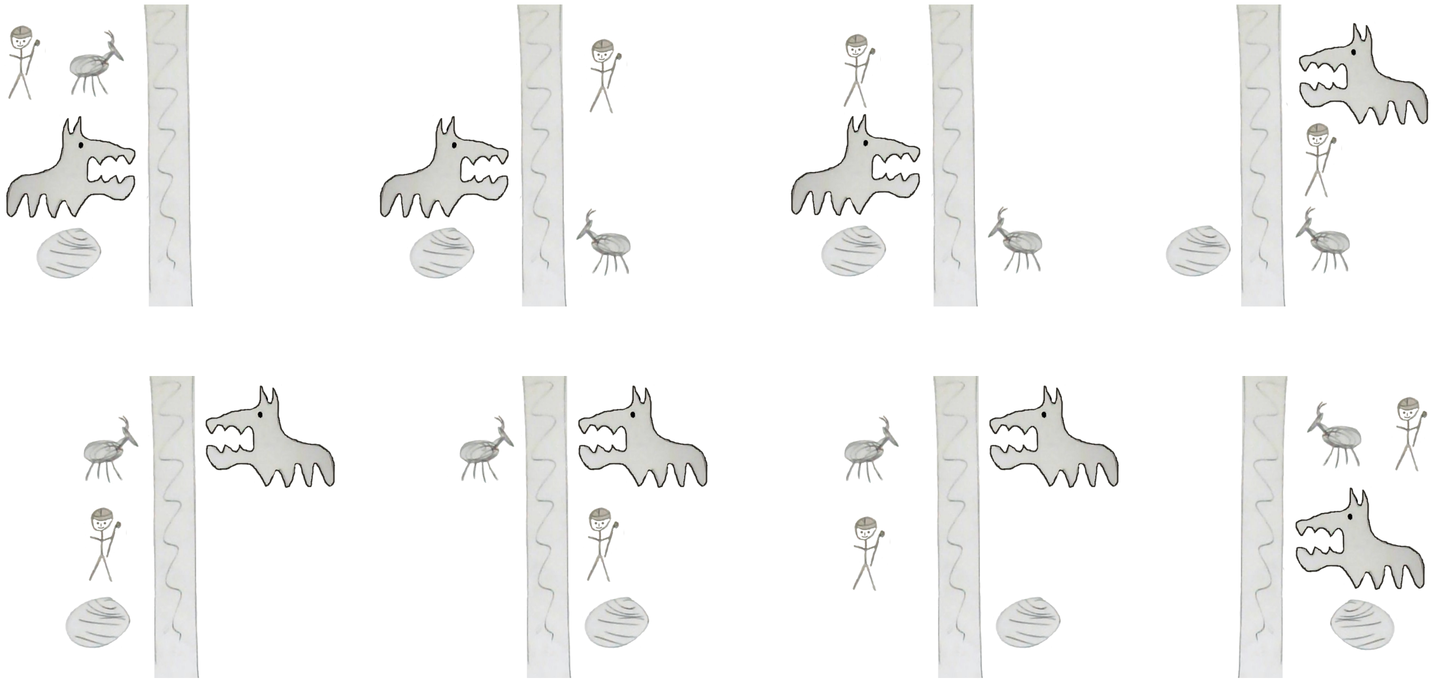
პირველ რიგში უნდა დავამტკიცოთ ამ ალგორითმის სისწორე: რომ მისი საწყისი მონაცემებით გაშვებისას სასურველი შედეგი მიიღება და რომ ამ ალგორითმის მსვლელობისას ამოცანის არც ერთი პირობა არ ირღვევა (არ ხდება ისეთი რამ, რაც ზემოთ ჩამოთვლილ შეზღუდვებს დაარღვევდა).

სავარჯიშო 1.1: დაამტკიცეთ ამ ალგორითმის სისწორე (აჩვენეთ, რომ შედეგად მიიღება ის, რაც მოეთხოვება და არც ერთი ბიჯის ჩატარების შემდეგ ამოცანის პირობა თავისი შეზღუდვებით არ ირღვევა).

შემდეგ უნდა გამოვითვალოთ მისი სისწრაფე, ანუ რამდენ ბიჯს ანდომებს იგი დასაწყისიდან გაჩერებამდე. ცხადია, რომ აქ უნდა განვსაზღვროთ, თუ რას ნიშნავს „ერთი ბიჯი". ჩვენს შემთხვევაში ეს მდინარის გადაკვეთა შეიძლება იყოს, ანუ ალგორითმი იმდენ ბიჯს საჭიროებს, რამდენჯერაც გადალახავს ადამიანი მდინარეს (არ აქვს მნიშვნელობა იმას, რამდენად დატვირთულია ნავი).

სავარჯიშო 1.2: დაითვალეთ ამ ალგორითმის ბიჯების რაოდენობა.

როგორც წესი, ყოველდღიური ამოცანის დასმისას დიდი ინფორმაცია არ არის მნიშვნელოვანი. მაგალითად, არ არის საინტერესო, თუ რა ფორმისა ან სიგანისაა მდინარე, რა ფერისაა ნავი და ა.შ. ჩვენ გვაინტერესებს მხოლოდ ის ინფორმაცია, რომელიც ამოცანის პირობისთვისაა მნიშვნელოვანი. მაგალითად ის, რომ ერთ ჯერზე მხოლოდ ორი მგზავრი ეტევა ნავში და ერთ-ერთი მგზავრი აუცილებლად ადამიანია. თუ ჩვენ მარცხენა ნაპირს დავარქმევთ A, ხოლო მარჯვენას კი B, ეს ორი ნაპირი შეგვიძლია გამოვსახოთ ორი სიმრავლით, რომელსაც აგრეთვე სიმრავლე A და სიმრავლე B ეწოდება. ყოველ ცხოველს შევუსაბამებთ ერთ ასოს – ადამიანი ⇒ ა, მგელი ⇒ მ, თხა ⇒ თ და კომბოსტო ⇒ კ (ნახ. 1.4).



ნახ. 1.3: ალგორითმის თითოეული ბიჯი

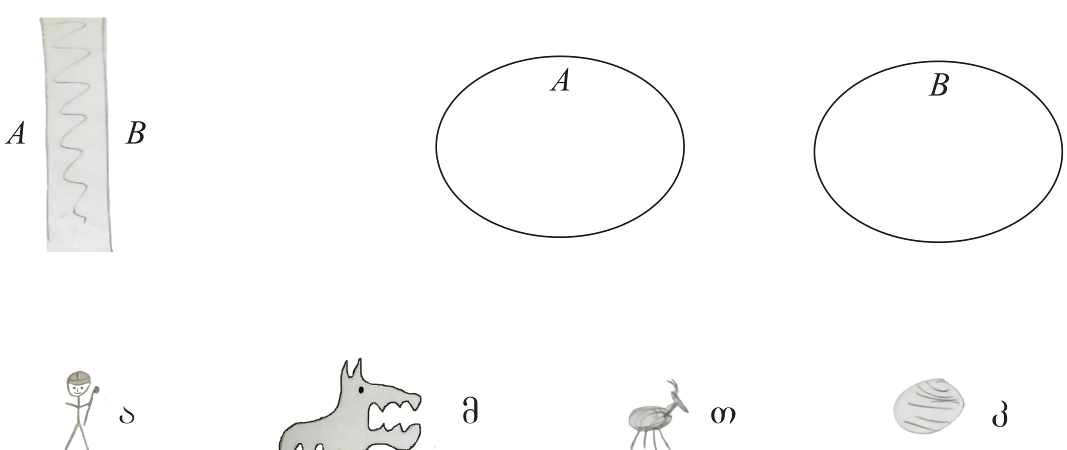
მაშინ საწყისი და საბოლოო პირობები შემდეგნაირი იქნება $A = \{$ მ,თ,კ,ა$\}, B = \emptyset$ და, შესაბამისად, $A = \emptyset, B = \{$ მ,თ,კ,ა$\}$ (ნახ. 1.5). მათემატიკურ ენაზე კი დასმული ამოცანის პირობა ასე შეიძლება ჩამოყალიბდეს:

მოცემულია: ორი სიმრავლე $A = \{$ა, მ, თ, კ$\}$ და $B = \emptyset$.

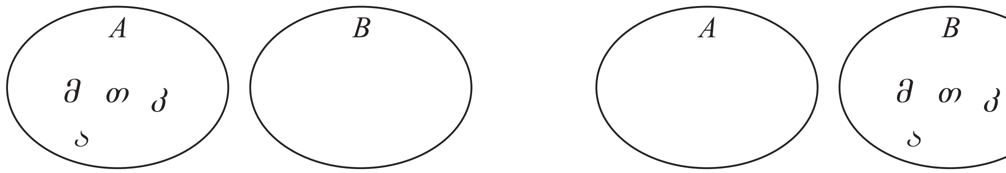
შედეგი: $A = \emptyset$ და $B = \{$ა, მ, თ, კ$\}$.

შეზღუდვა: ყოველ ჯერზე იმ სიმრავლიდან, რომელიც შეიცავს ასოს „ა”, მეორე სიმრავლეში უნდა გადავიტანოთ ეს ასო და კიდევ ერთი ან ნული ასო. ის სიმრავლე, რომელიც **არ** შეიცავს ასოს „ა”, **არ** უნდა შეიცავდეს ერთად ასოებს $\{$მ, თ$\}$ და $\{$თ, კ$\}$.

ამოცანის პირობა ოდნავ გამარტივდება, თუ ასოების ნაცვლად გარკვეულ რიცხვებს ავიღებთ: ადამიანი $\Rightarrow$ 10,



ნახ. 1.4: ნახატების ფორმალიზაცია



ნახ. 1.5: ფორმალიზაციის შედეგად მიღებული საწყისი და საბოლოო პირობა

მგელი ⇒ 1, თხა ⇒ 2 და კომბოსტო ⇒ 3. მაშინ თხა და კომბოსტო ან თხისა და მგლის ერთ ნაპირზე ყოფნა იმას ნიშნავს, რომ შესაბამისი სიმრავლის ელემენტების ჯამი კენტია, ხოლო ის ფაქტი, რომ ადამიანი რომელიღაცა ნაპირზე არ იმყოფება, იმას ნიშნავს, რომ შესაბამისი ელემენტების ჯამი ნაკლებია 10-ზე.

სავარჯიშო 1.3: ზემოთ ნახსენები ამოცანა ჩამოაყალიბეთ რიცხვებისათვის.

სავარჯიშო 1.4: წინა სავარჯიშოში ჩამოყალიბებული ამოცანისათვის დაწერეთ ალგორითმი და მისი ყოველი ბიჯისათვის შესაბამისი სიმრავლეები ჩამოწერეთ.

სავარჯიშო 1.5: დაამტკიცეთ წინა სავარჯიშოში დაწერილი ალგორითმის სისწორე და დაითვალეთ მისი ბიჯების რაოდენობა.
სავარჯიშო 1.6: განიხილეთ შემდეგი ალგორითმი:

ალგორითმი 1.2: „მგელი, თხა და კომბოსტო” (სწრაფი ვერსია)
მონაცემები: მდინარე და მის მარცხენა ნაპირზე ადამიანი, ნავი, მგელი, თხა და კომბოსტო;

მარჯვენა ნაპირზე გადაიყვანე თხა ;
დაბრუნდი მარცხენა ნაპირზე ;
მარჯვენა ნაპირზე გადაიყვანე მგელი ;
დაბრუნდი მარცხენა ნაპირზე ;
მარჯვენა ნაპირზე გადაიტანე კომბოსტო.
ალგორითმი დასრულებულია

მივიღებთ თუ არა ამ ალგორითმის მუშაობის შემდეგ იმ შედეგს, რომელიც ამოცანაშია მოთხოვნილი? არის თუ არა ეს ყველაზე სწრაფი ალგორითმი იმ ალგორითმთა შორის, რომელიც ამ ამოცანას ხსნის?

შენიშვნა: აქამდე ჩვენ განვიხილავდით შემთხვევას, როდესაც დასაწყისში ყველა მარცხენა ნაპირზე დგას. ზუსტად იგივე მსჯელობის ჩატარება შეიძლება იმ შემთხვევისათვის, როდესაც ყველა მარჯვენა ნაპირზე დგას. ამ შემთხვევისათვის ალგორითმი ანალოგიური იქნება. არც იმას აქვს მნიშვნელობა, თუ რა თანმიმდევრობით ჩამოვთვლით ცხოველებს მოცემულობაში. ეს ყოველთვის ასე არაა, როგორც შემდეგი მარტივი მაგალითი გვიჩვენებს:

მოცემულია ორი რიცხვი. გამოითვალეთ $\dfrac{\text{პირველი რიცხვი}}{\text{მეორე რიცხვი}}$.

ცხადია, რომ აქ გადამწყვეტი მნიშვნელობა აქვს ამოცანის პირობაში მოცემული რიცხვების თანმიმდევრობას.

სავარჯიშო 1.7: განვიხილოთ n მთელი რიცხვის ზრდადობით დალაგების ამოცანა. რა არის ამ ამოცანაში მოცემული? რა უნდა იყოს მისი საბოლოო შედეგი?

სავარჯიშო 1.8: მოიყვანეთ შემდეგი ამოცანის ალგორითმი: მოცემული 10 ცალი მთელი რიცხვისათვის დაითვალეთ კენტ რიცხვთა ჯამი. მინიშნება: ყოველ ბიჯზე უნდა შევამოწმოთ, არის თუ არა მოცემული რიცხვი კენტი. რამდენ ბიჯს მოითხოვს ასეთი ლგორითმი? რიცხვის კენტობის შემოწმება და მიმატების ოპერაცია თითო-თითო ბიჯად ჩათვალეთ.
რა არის ამ ამოცანის მონაცემი? რა არის შედეგი? როგორია პირობაზე დადებული შეზღუდვა?

1.2 მოკლე დასკვნა

პირველ თავში ჩვენ განვიხილეთ მარტივი ამოცანების ამოხსნის გზები, რის მაგალითზეც ვაჩვენეთ ამოცანის დასმისა და მისი ალგორითმების, ანუ ამოცანის გადასაჭრელად საჭირო ბიჯების ჩამონათვალის შედგენის გზა. ამოცანის ჩამოსაყალიბებლად საჭიროა მისი მონაცემებისა და შედეგის მკაფიოდ ჩამოთვლა (რა არის საწყისი ვითარება და რა უნდა მივიღოთ შედეგად), ასევე ის შეზღუდვები, რომლებიც უნდა გავითვალისწინოთ ამოცანის გადაჭრის (ანუ ალგოროთმის ჩატარების) დროს და რომელთა დარღვევა დაუშვებელია.
ალგორითმის შედგენის შემდეგ უნდა დავამტკიცოთ მისი სისწორე: რომ ყოველ დასაშვებ საწყის მონაცემზე შესაბამისი სწორი პასუხი მიიღება და რომ ალგორითმის არც ერთი ბიჯის შემდეგ შეზღუდვა არ ირღვევა.
ბოლოს - ალგორითმის შეფასებისთვის - უნდა გამოვითვალოთ მისი ბიჯების რაოდენობა, რომ დავადგინოთ, თუ რამდენად სწრაფია ჩვენი მეთოდი. აღსანიშნავია, რომ ყოველ ასეთ დროს უნდა განისაზღვროს, თუ რას ნიშნავს „ერთი ბიჯი": ერთსა და იმავე ალგორითმში ეს სხვადასხვა რამ შეიძლება იყოს, თუმცა ერთი ბიჯი რეალურად საინტერესო მარტივი მოქმედება უნდა იყოს. ასე, მაგალითად,ცხოველების გადაყვანის ალგორითმში ერთ ბიჯად შეიძლებოდა ადამიანის მიერ ნავის ნიჩბის მოსმის რაოდენობა აგვეღო, მაგრამ ეს უინტერესოდ მარტივი ოპერაცია იქნებოდა და ამოცანის არსის შეფასებაში არ გამოგვადგებოდა.

დასასრულს მოვიყვანოთ კიდევ ერთი

ამოცანა: ორი დიდი ხნის უნახავი მათემატიკოსი ერთმანეთს ხვდება. ერთი ეუბნება: მე სამი შვილი მყავს. ერთ რამეს გეტყვი და თუ გამოიცნობ მათ ასაკს: მათი ასაკის ნამრავლია 36.
მეორე ეუბნება: ვერ გამოვიცნობ, დამატებით სხვა პირობა მჭირდება. პირველი ეტყვის: მათი ასაკის ჯამი შენს წინ მდებარე სახლის ფანჯრების რაოდენობის ტოლია. მეორე შეხედავს სახლს და ეტყვის: ერთი დამატებითი პირობა კიდევ მჭირდება.
პირველი ეტყვის: უფროსს ლურჯი თვალები აქვს. ამით მეორე სამივეს ასაკს გამოიცნობს.

შეკითხვა: რამდენი წლის არიან შვილები?

თავი 2

ამოცანათა რეკურსიული და იტერაციული აღწერა

2.1 ამოცანა ნავების შესახებ

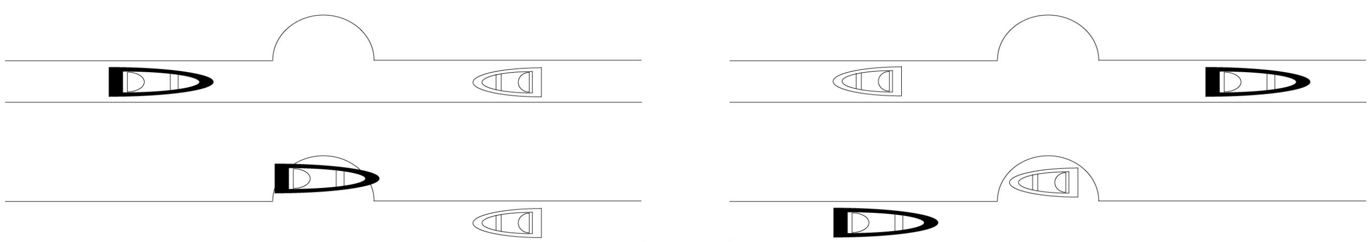
განვიხილოთ შემდეგი ამოცანა:

მოცემულია: ვიწრო მდინარე პატარა ყურეთი. მდინარეში ყურეს მარცხნივ გრძელი შავი ნავი და მარჯვნივ - მოკლე თეთრი ნავი.

შედეგი: მდინარეში ყურეს მარცხნივ მოკლე თეთრი ნავი და მარჯვნივ – გრძელი შავი ნავი (ნავებმა ერთმანეთს გვერდი უნდა აუქციონ).

შეზღუდვა: მდინარე იმდენად ვიწროა, რომ სიგანეში მხოლოდ ერთი ნავი ეტევა. ყურეში ეტევა მხოლოდ თეთრი ნავი. შავი ნავი ყურეში არ ეტევა.

ნახ. 2.1-ში გრაფიკულადაა ნაჩვენები ამოცანის მონაცემი, შედეგი და შეზღუდვები.



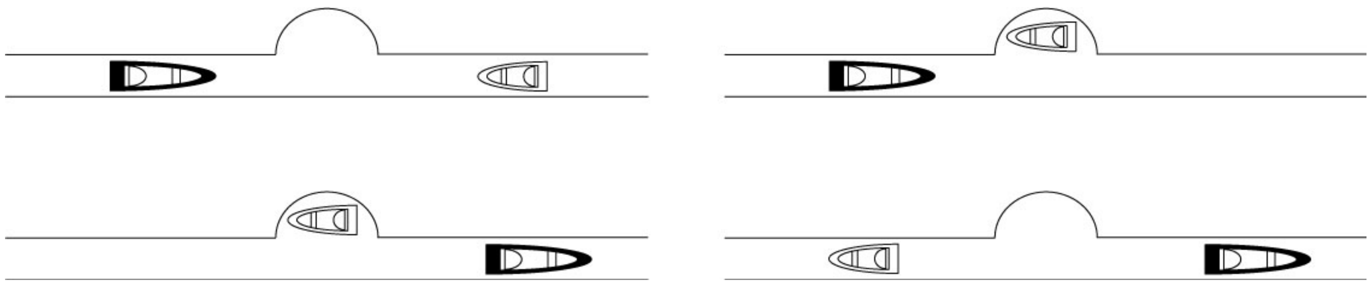
ნახ. 2.1:

იმისათვის, რომ ერთმა თეთრმა ნავმა შავს გვერდი აუქციოს, საჭიროა შემდეგი ალგორითმის ჩატარება:

ალგორითმი 2.1: „ერთი ნავის გაყვანა"
მონაცემები: ვიწრო მდინარე პატარა ყურეთი,
ყურეს მარცხნივ გრძელი შავი ნავი და მარჯვნივ - მოკლე თეთრი ნავი

1: თეთრი ნავი შევიდეს ყურეში;
2: შავმა ნავმე გაიაროს;
3: თეთრი ნავი გამოვიდეს ყურედან.
ალგორითმი დასრულებულია

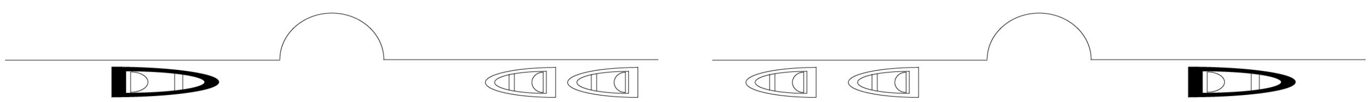
ადვილი საჩვენებელია, რომ ეს ალგორითმი ამოცანის საბოლოო შედეგს მოგვცემს და მისი არც ერთი ბიჯი ამოცანის შეზღუდვებს არ ეწინააღმდეგება (ნახ. 2.2).



ნახ. 2.2:

ზემოთ მოყვანილი ალგორითმი აღვნიშნოთ როგორც A_1. ესე იგი, თუ ზემოთ მოყვანილ საწყის პირობაზე ჩავატარებთ ალგორითმს A_1, შედეგად მივიღებთ ზემოთვე მოყვანილ საბოლოო შედეგს.

ახლა კი განვიხილოთ ისეთი შემთხვევა, როდესაც ყურეს მარჯვნივ არა ერთი, არამედ ორი ნავია განთავსებული. ნახ. 2.3-ში გრაფიკულადაა ნაჩვენები ამ ამოცანის მონაცემი და შედეგი. ამ ამოცანას ჩვენ ვუწოდებთ „ორი ნავი".



ნახ. 2.3:

თუ პირველ რიგში ჩავატარებთ იგივე სამ ბიჯს, რაც ალგორითმში A_1, მივიღებთ ისეთ სიტუაციას, როგორიც ნაჩვენებია ნახ. 2.4-ში (მარცხნივ). შემდეგ, თუ შავი ნავი წავა უკან ყურეს მარცხნივ, შეიქმნება ისეთივე სიტუაცია, როგორც წინა ამოცანაში (ნახ. 2.4 მარჯვნივ)

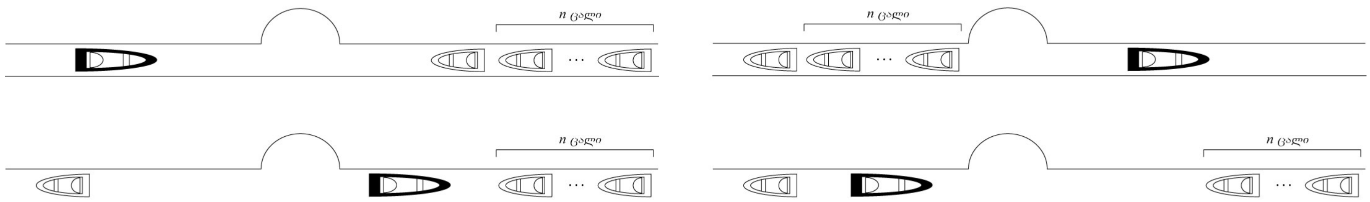
ნახ. 2.4:

შავი ნავის უკან გასვლის პროცესი აღვნიშნოთ როგორც U. ესე იგი, თუ საწყისი მდგომარეობაა ისეთი, როგორც ნახ. 2.3-ში მარცხნივ და ჯერ ჩავატარებთ ალგორითმს A_1, მივიღებთ ისეთ ვითარებას, როგორც ნახ. 2.4-ში მარცხნივ. თუ შემდეგ კიდევ ჩავატარებთ ალგორითმს U, მივიღებთ ისეთ ვითარებას, როგორც ნახ. 2.4-ში მარჯვნივ. აღსანიშნავია, რომ ამ შემთხვევაში შეიქმნა ისეთივე ვითარება, როგორც ამოცანაში „ერთი ნავი". ეს კი იმას ნიშნავს, რომ თუ გამოვიყენებთ ალგორითმს A_1, საბოლოო მდგომარეობას მივაღწევთ.
ასე რომ, ალგორითმი A_2, რომელიც ამოცანას „ორი ნავი" ხსნის, შემდეგნაირად შეიძლება ჩაიწეროს: $A_2 = A_1, U, A_1$ (ჯერ ჩაატარე ალგორითმი A_1, შემდეგ ალგორითმი U და ბოლოს ისევ ალგორითმი A_1.

ახლა კი დავუშვათ, რომ ალგორითმი A_n n თეთრი ნავის გვერდის აქცევას ახერხებს (ნახ. 2.5). აქამდე ჩვენ განვიხილეთ, თუ როგორია A_n, თუ $n = 1$, ან $n = 2$.

თუ განვიხილავთ $n + 1$ ნავის გვერდის აქცევის ამოცანას ისეთი საწყისი და საბოლოო მდგომარეობებით, რომლებიც ნაჩვენებია ნახ. 2.6-ში (ზემოთ) და ჩავატარებთ ალგორითმებს A_1, U, მივიღებთ ისეთ სიტუაციას, რომელიც გვქონდა n ნავის გვერდის აქცევის ამოცანაში (ნახ. 2.6 ქვემოთ).

ნახ. 2.5:



ნახ. 2.6:

ეს კი იმას ნიშნავს, რომ A_n ალგორითმის გამოყენების შემდეგ მიიღება საბოლოო მდგომარეობა (ნახ. 2.6 ზემოთ მარჯვნივ).

საბოლოოდ მივიღებთ შემდეგ ჩანაწერს: $A_{n+1} = A_1, U, A_n$. თუ ვიცით, როგორია ალგორითმი A_1, ადვილი გამოსათვლელია ალგორითმი A_2 (აქ $n+1 = 2$ და $n = 1$): ჯერ ჩავატარებთ ალგორითმს A_1, შემდეგ U და შემდეგ ისევე A_1. A_3 ალგორითმის ჩასატარებლად ჯერ უნდა ჩავატაროთ A_1, შემდეგ U და შემდეგ A_2. ასე ნაბიჯ-ნაბიჯ შეიძლება გამოვითვალოთ A_n ნებისმიერი ნატურალური n რიცხვისათვის: $A_n = A_1, U, A_{n-1} = A_1, U, A_1, U, A_{n-2} = A_1, U, A_1, U, A_1, U, A_{n-2} = A_1, U, A_1, U, \cdots, A_1$ (n-ჯერ).

<u>სავარჯიშო 2.1:</u> რისი ტოლია A_7? (მაგ.: $A_3 = A_1, U, A_2 = A_1, U, A_1, U, A_1$)

მნიშვნელოვანია ის ფაქტი, რომ ალგორითმი A_n იყენებს „თავის თავს", მხოლოდ უფრო დაბალი პარამეტრით (მაგ. $A_2 = A_1, U, A_1$; $A_7 = A_1, U, A_6$ და ა.შ.)
იმ შემთხვევაში, როდესაც ალგორითმი თავის თავს იყენებს, მას „რეკურსიული" ეწოდება. ესე იგი, $A_n = A_1, U, A_{n-1}$ ალგორითმის ეს ჩანაწერი რეკურსიულია.
აღსანიშნავია ისიც, რომ ნებისმიერი რეკურსიული ალგორითმი შეიძლება არარეკურსიული სახითაც ჩაიწეროს (განიხილეთ წინა სავარჯიშოს მაგალითი).

რეკურსიული ალგორითმების გარდა არსებობს ე. წ. „იტერაციული" ალგორითმებიც, რომელშიც ერთი და იგივე ოპერაცია (ან უფრო ხშირად ოპერაციათა მიმდევრობა) რამოდენიმეჯერ მეორდება.
ნავების ამოცანის ასეთი ალგორითმი შეიძლება შემდეგნაირად ჩაიწეროს:

ალგორითმი 2.2: ნავების გაყვანა (არარეკურსიული - იტერაციული - ვერსია)
მონაცემი: n (ნავების რაოდენობა)

```
გაიმეორე n-ჯერ:
{
  თეთრი ნავი შევიდეს ყურეში;
  შავმა ნავმა გაიაროს;
  თეთრი ნავი გამოვიდეს ყურედან;
  შავი ნავი წავიდეს უკან
}
```
ალგორითმი დასრულებულია

იტერაციული ალგორითმის იმ ნაწილს, რომელიც რამოდენიმეჯერ მეორდება, ***ციკლი***, კოლო ციკლში მოქცეულ ბრძანებათა მიმდევრობას კი ***ციკლის ტანი*** ეწოდება.

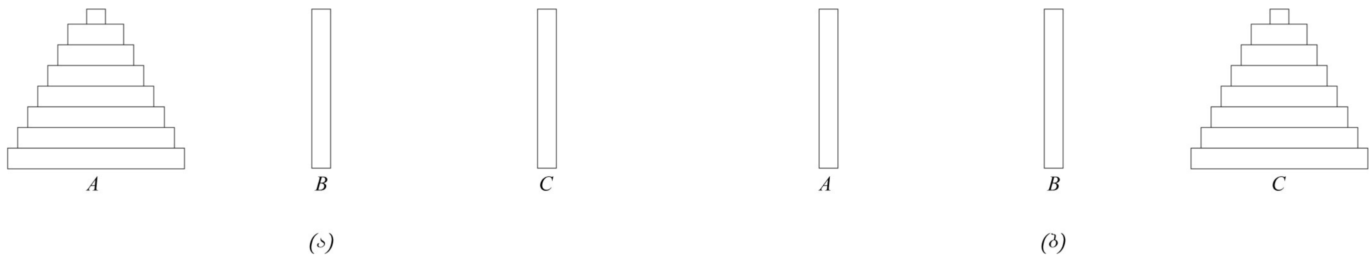
<u>სავარჯიშო 2.2:</u> დაამტკიცეთ, რომ ამ ალგორითმის ბიჯების რაოდენობაა $4n$.

<u>სავარჯიშო 2.3:</u> როგორ უნდა შეიცვალოს ალგორითმი, რომ მისი ბიჯების რაოდენობა გახდეს $4n-1$?

2.2 ჰანოის კოშკების ამოცანა

1883 წელს ფრანგმა მათემატიკოსმა ედუარდ ლუკასმა დასვა შემდეგი ამოცანა:

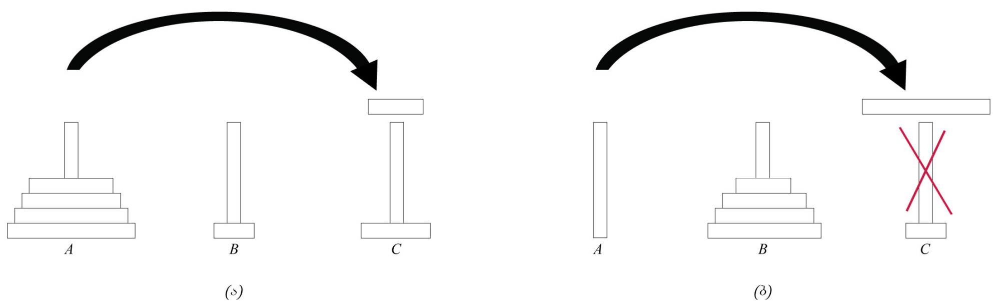
მოცემულია: სამი ძელი A, B, C. A ძელზე ჩამოცმულია სხვადასხვა ზომის n რგოლი ისე, რომ დიდ რგოლს უფრო პატარა ადევს – შექმნილია პირამიდა (ნახ. 2.7 (ა)).



ნახ. 2.7: ჰანოის კოშკების ამოცანის საწყისი და საბოლოო მდგომარეობები

შედეგი: A ძელზე აგებული პირამიდა C ძელზე (ნახ. 2.7 (ბ)).

შეზღუდვა: თითო ჯერზე ერთი ძელიდან მეორეზე უნდა გადავიტანოთ ერთი და მხოლოდ ერთი რგოლი, რომელიც ყველაზე მაღლა დევს. ამავე დროს არ შეიძლება პატარა ზომის რგოლზე დიდი ზომის რგოლის დადება.



ნახ. 2.8: დასაშვები *(ა)* და აკრძალული *(ბ)* სვლები

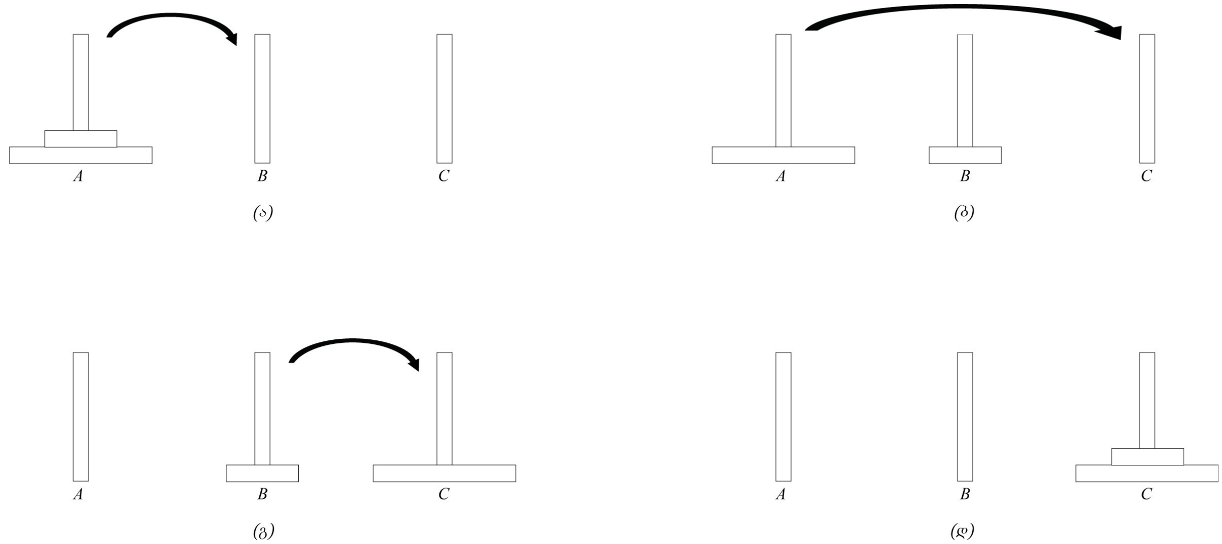
დავუშვათ, მოცემულია ერთ რგოლიანი პირამიდა. ცხადია, რომ მისი ერთი ძელიდან მეორეზე გადასატანად საკმარისია ერთი მოქმედება. თუ ეს ერთი რგოლი A ძელიდან C ძელზე გადაგვაქვს, ამ პროცედურას ვუწოდებთ $A_1^{A,C}$.

იმისათვის, რომ ორ რგოლიანი პირამიდა A ძელიდან C ძელზე გადავიტანოთ, საჭიროა შემდეგი მოქმედებების ჩატარება:

1. A ძელიდან ზედა რგოლი გადაიტანე B ძელზე (ჩაატარე $A_1^{A,B}$, ნახ. 2.9 *(ბ)*);
2. A ძელიდან ზედა რგოლი გადაიტანე C ძელზე (ჩაატარე $A_1^{A,C}$, ნახ. 2.9 *(გ)*);
3. B ძელიდან ზედა რგოლი გადაიტანე C ძელზე (ჩაატარე $A_1^{B,C}$, ნახ. 2.9 *(დ)*).

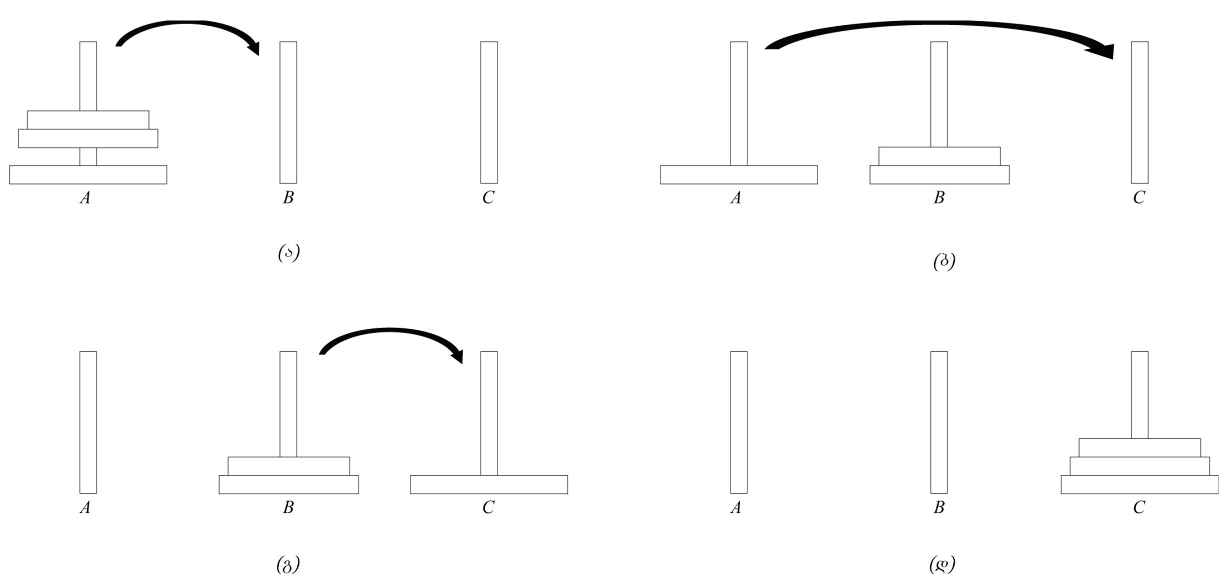
ორ რგოლიანი პირამიდის A ძელიდან C ძელზე გადატანის ალგორითმი (ანუ ზემოთ მოყვანილი სამ ბიჯიანი პროცესი) აღვნიშნოთ როგორც $A_2^{A,C}$.

ზოგადად, n რგოლის ერთი ძელიდან მეორეზე გადატანის ალგორითმი შემდეგნაირად შეიძლება აღინიშნოს: $A_n^{X_1,X_2}$. აქ $n \in \mathbb{N}$, $X_1, X_2 \in \{A, B, C\}$ და $X_1 \neq X_2$. ამრიგად, $A_{13}^{C,A}$ ნიშნავს ალგორითმს, რომელიც C ძელზე



ნახ. 2.9: ორ რგოლიანი პირამიდის გადატანისათვის საჭირო ოპერაციები

აწყობილ 13 რგოლიან პირამიდას A ძელზე გადაიტანს, ხოლო $A_{108}^{B,A}$ კი იმ ალგორითმს, რომელიც B ძელზე აწყობილ 108 რგოლიან პირამიდას A ძელზე გადაიტანს.
თუ ვიცით, როგორ გადავიტანოთ ორ რგოლიანი პირამიდა ერთი ძელიდან მეორეზე, ადვილად შევადგენთ ალგორითმს $A_3^{A,C}$:
სამ რგოლიანი პირამიდა განვიხილოთ, როგორც ქვედა დიდ რგოლზე დადგმული ორ რგოლიანი პირამიდა (ნახ. 2.12 *(ა)*).



ნახ. 2.10: სამ რგოლიანი პირამიდის გადატანისათვის საჭირო ოპერაციები

ამრიგად, $A_2^{A,B}$ ალგორითმით შეიძლება ზედა ორ რგოლიანი პირამიდის გადატანა B ძელზე (ნახ. 2.12 *(ბ)*), შემდეგ $A_1^{A,C}$ ალგორითმით ქვედა რგოლი გადაგვაქვს A ძელიდან C ძელზე (ნახ. 2.12 *(გ)*) და ბოლოს ისევე $A_2^{B,C}$ ალგორითმით ორ რგოლიანი პირამიდა გადაგვაქვს B ძელიდან C ძელზე (ნახ. 2.12 *(დ)*).

ეს ალგორითმი რეკურსიულად შემდეგნაირად შეიძლება ჩაიწეროს: $A_3^{A,B} = [A_2^{A,B}, A_1^{A,C}, A_2^{B,C}]$ (ჯერ შეასრულე $A_2^{A,B}$, შემდეგ $A_1^{A,C}$ და ამის შემდეგ $A_2^{B,C}$.

აღსანიშნავია, რომ $A_2^{A,B}$ და $A_2^{B,C}$ თვითონ რამოდენიმე ბიჯისაგან შედგება: $A_2^{A,B} = [A_1^{A,C}, A_1^{A,B}, A_2^{C,B}]$ და $A_2^{B,C} = [A_1^{B,A}, A_1^{B,C}, A_2^{A,C}]$.

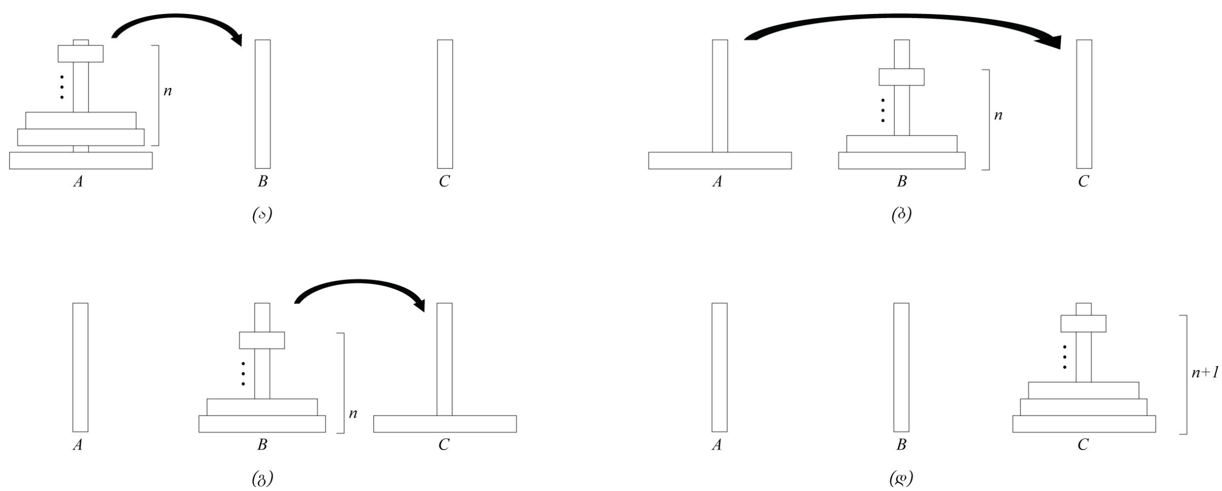
სავარჯიშო 2.4: რეკურსიულად ჩაწერეთ $A_3^{B,C}$, $A_3^{C,A}$, $A_3^{A,B}$, $A_3^{B,A}$ და $A_3^{C,B}$ (იხ. ზემოთ მოყვანილი ანალოგიური ჩანაწერი $A_3^{A,C}$).

თუ ვიცით, როგორ გადავიტანოთ 3 რგოლიანი პირამიდა ერთი ძელიდან მეორეზე, რეკურსიულად შეიძლება $A_4^{X_1,X_2}$ ალგორითმის დადგენა. მაგ., $A_4^{A,C} = [A_3^{A,B}, A_1^{A,C}, A_3^{B,C}]$.

სავარჯიშო 2.5: რეკურსიულად ჩაწერეთ $A_4^{B,C}$, $A_4^{C,A}$, $A_4^{A,B}$, $A_4^{B,A}$ და $A_4^{C,B}$ (იხ. ზემოთ მოყვანილი ანალოგიური ჩანაწერი $A_3^{A,C}$).

თუ ვიცით, როგორია n რგოლიანი პირამიდის ერთი ძელიდან მეორეზე გადატანის ალგორითმი $A_n^{X_1,X_2}$, ადვილად შევადგენთ $n+1$ რგოლიანი ალგორითმის გადატანის ალგორითმს $A_{n+1}^{X_1,X_2}$ (შესაბამისი მოქმედებები ნაჩვენებია ნახ. 3.1 -ში):

$$A_{n+1}^{X_1,X_2} = [A_n^{X_1,X_3}, A_1^{X_1,X_2}, A_n^{X_3,X_2}], \quad X_1 \neq X_2 \neq X_3, \quad X_1, X_2, X_3 \in \{A, B, C\}.$$



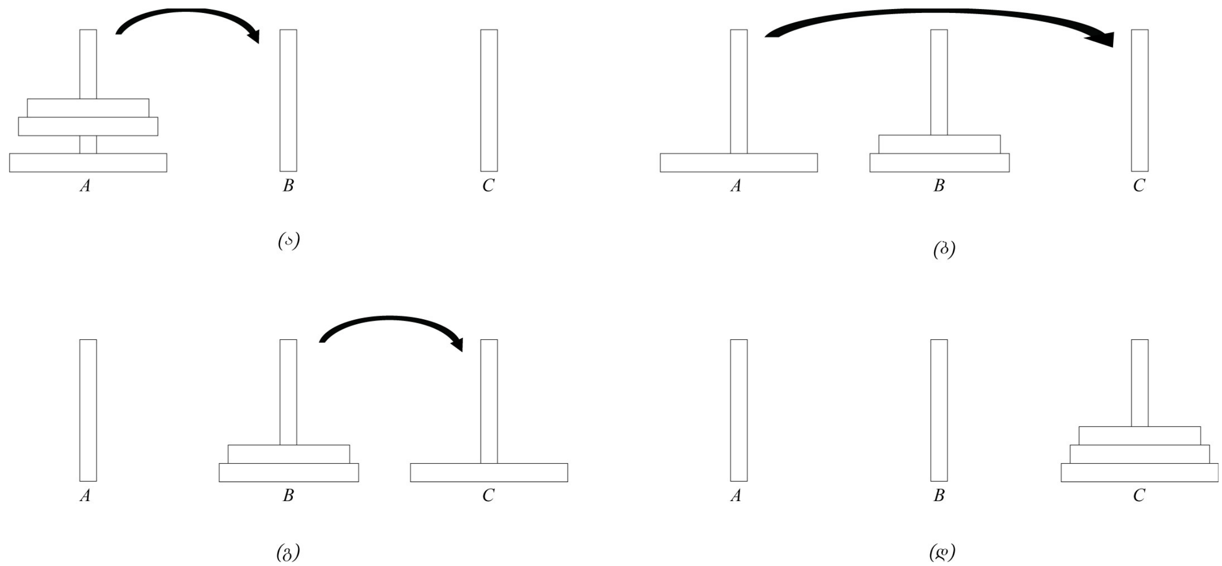
ნახ. 2.11: $n+1$ რგოლიანი პირამიდის გადატანისათვის საჭირო ოპერაციები

როგორც ყველა წინა მაგალითში, აქაც n ცალი რგოლის გადატანა ერთდროულადაა ნაჩვენები იმის და მიუხედავად, რომ $A_n^{X_1,X_2}$ რამოდენიმე ბიჯისაგან შედგება.

სავარჯიშო 2.6: რას აღნიშნავს შემდეგი ჩანაწერები: $A_7^{B,C}$, $A_{12}^{C,B}$, $A_4^{B,C}$?

ადვილი შესამოწმებელია, რომ $A_1^{X_1,X_2}$ ალგორითმის შესრულებისას ამოცანის პირობა არ ირღვევა. თუ განვიხილავთ $A_2^{A,C}$ ალგორითმის რეკურსიულ ჩანაწერს, დავინახავთ, რომ პირველ რიგში უნდა შევასრულოთ ალგორითმი $A_1^{A,B}$. ადვილი სანახავია, რომ ამ ალგორითმის შესრულებისასაც პირობა არ ირღვევა. შემდეგ უნდა შევასრულოთ $A_1^{A,C}$. რადგან C ძელზე რგოლი არ დევს, მასზე A ძელიდან რგოლის გადატანა შესაძლებელია (პირობა არ დაირღვევა) და C ძელზე ყველაზე დიდი რგოლი იდება. ბოლოს უნდა ჩავატაროთ $A_1^{B,C}$. ეს შესაძლებელია, რადგან C ძელზე ყველაზე დიდი რგოლი დევს.
ანალოგიური მსჯელობით შეიძლება დავამტკიცოთ, რომ თუ $A_3^{A,C}$ ალგორითმს ჩავწერთ ისე, როგორც ზემოთ განვიხილეთ და მას თანმიმდევრულად შევასრულებთ, ამოცანის პირობა არ ირღვევა: პირველ რიგში უნდა შესრულდეს $A_2^{A,B}$ (ნახ. 2.12 *(ა)*). ეს შესაძლებელია, რადგან B და C ძელები ცარიელია და A ძელზე ქვემოთ ყველაზე დიდი რგოლი დევს, რომელზეც პირობის თანახმად სხვა ნებისმიერი რგოლის დადება შეიძლება. ასე რომ, ამ ოპერაციების შესრულების დროს ამოცანის პირობა არ დაირღვევა. შედეგად მივიღებთ A ძელზე ერთ ყველაზე დიდ რგოლს და B ძელზე კი ორ რგოლიან პირამიდას (ნახ. 2.12 *(ბ)*). შემდეგ უნდა ჩავატაროთ $A_1^{A,C}$. ესეც არ

არღვევს ამოცანის პირობას, რადგან ამ მომენტისათვის C ძელი ცარიელია. შედეგად მივიღებთ C ძელზე ერთ ყველაზე დიდ რგოლს და B ძელზე კი ორ რგოლიან პირამიდას, ხოლო A ძელი კი ცარიელი იქნება (ნახ. 2.12 *(გ)*). ბოლოს უნდა შევასრულოთ $A_2^{B,C}$. ესეც შესაძლებელია, რადგან A ძელი ცარიელია და C ძელზე ყველაზე დიდი რგოლი დევს, რომელზედაც ყველა დანარჩენი რგოლის დადება შეიძლება. ამ ოპერაციების ჩატარების შედეგად ამოცანის საბოლოო შედეგს მივიღებთ (ნახ. 2.12 *(დ)*).



ნახ. 2.12: სამ რგოლიანი პირამიდის გადატანისათვის საჭირო ოპერაციები

სავარჯიშო 2.7: დავუშვათ, მოცემულია შემდეგი ჩანაწერი: $A_3^{A,C} = [A_1^{A,B}, A_2^{A,C}, A_1^{B,C}]$. სიტყვიერად ახსენით, რა ოპერაციები უნდა შესრულდეს ამ ჩანაწერის შესაბამისად. ირღვევა თუ არა ამ ალგორითმის შესრულებისას ჰანოის კოშკების ამოცანის პირობა?

ახლა კი განვიხილოთ ჰანოის კოშკების იტერაციული ალგორითმი:

ალგორითმი 2.3: ჰანოის კოშკები (არარეკურსიული - იტერაციული - ვერსია)
მონაცემი: n (ზომის კლებადობის მიხედვით დალაგებული რგოლების რაოდენობა, სადაც რგოლების პირამიდა A ძელზეა აგებული, B და C ძელი კი ცარიელია)

გაიმეორე მანამ, სანამ n რგოლიანი პირამიდა არ იქნება გადატანილი A ძელისგან განსხვავებულ ძელზე:
{
  მინიმალური ზომის რგოლი გადაიტანე ერთი პოზიციით მარჯვნივ (ან C ძელიდან A ძელზე);
  არამინიმალური ზომის რგოლი გადაიტანე შეძლებისამებრ (აქ მხოლოდ ერთი ვარიანტი იარსებებს)
}
ალგორითმი დასრულებულია

უნდა აღინიშნოს, რომ იმის მიუხედავად, რომ წარმოდგენილი იტერაციული ალგორითმი მარტივად აღიწერება, რეკურსიული ალგორითმისაგან განსხვავებით მისი სისწორის მტკიცება საკმაოდ რთულია.
ასევე რთულია ამ ალგორითმის ბიჯების რაოდენობის დათვლა, რადგან არც ისე ცხადია, როდის შეწყდება ციკლი (როდის გადავა მთელი პირამიდა A ძელიდან რაიმე სხვა ძელზე).
ამ საკითხებს ჩვენ დაწვრილებით შემდეგ თავში განვიხილავთ.

სავარჯიშო 2.8: რომელ ძელზე გადაიტანს ზემოთ აღწერილი ალგორითმი 3, 4, 5 და 6 რგოლიან პირამიდას? ზოგადად, რომელ ძელზე გადაიტანს ლუწი რაოდენობის რგოლიან პირამიდას? და კენტი რაოდენობის რგოლებიან პირამიდას?

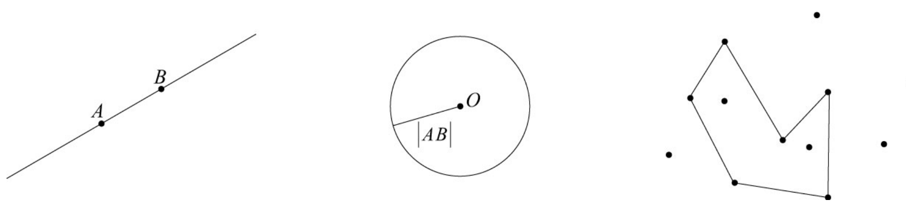
2.3 ძველი ბერძნული ამოცანები

ანტიკურ საბერძნეთში დასვეს ე.წ. „ფარგლითა და სახაზავით აგების" გეომეტრიული ამოცანები. აღსანიშნავია, რომ რამოდენიმე ამოცანა 2000 წელზე მეტ ხანს ამოუხსნელი რჩებოდა, სანამ XIX საუკუნეში მათემატიკურად არ დამტკიცდა, რომ მათი ალგორითმული გადაჭრა შეუძლებელია. ეს, ალბათ, ყველაზე ძველი ამოცანებია, რომელთაც ალგორითმული ამოხსნა არ აქვთ (თუმცა უნდა აღინიშნოს, რომ ლაპარაკია აგებაზე ***მხოლოდ ფარგლისა და სახაზავის გამოყენებით***, რაც იგივე ამოცანათა სხვა მეთოდებით გადაჭრას არ გამორიცხავს. სხვა სიტყვებით რომ ვთქვათ, ეს ამოცანები რესურსების შეზღუდვისას ვერ გადაიჭრება, მაგრამ რესურსების შეზღუდვის გარეშე მათი ამოხსნა არაა გამორიცხული).

მოცემულია: ფარგალი, სახაზავი და ორი წერტილი სიბრტყეზე; რაიმე გეომეტრიული ფიგურა; რაიმე ნამდვილი რიცხვი ξ.

შედეგი: მოცემული გეომეტრიული ფიგურისთვისა ან რიცხვისთვის დაადგინეთ, შეიძლება თუ არა მათი ფარგლითა და სახაზავით აგება.

შეზღუდვა: სახაზავით შეიძლება მოცემულ ორ A და B წერტილზე წრფის გავლება. თუ მოცემულია ნებისმიერი ორი წერტილი A, B და ნებისმიერი მესამე წერტილი O, ფარგლით შეიძლება O წერტილიდან $|A,B|$ სიგრძის რადიუსის მქონე წრეწირის შემოვლება (ნახ. 2.13).



ნახ. 2.13: სახაზავით (მარცხნივ), ფარგლით (შუაში) და წერტილებზე აგებული ფიგურები

თუ მოცემულია უკვე აგებულ წერტილთა რაიმე სიმრავლე $\mathcal{S} = \{A_1, A_2, ..., A_n\}$, ამ სიმრავლის რამოდენიმე წერტილზე გავლებული შეკრული ტეხილის მიერ შემოფარგლული სიბრტყის ნაწილს (თავისი შიგთავსით) ფარგლითა და სახაზავით აგებული ფიგურა ეწოდება.

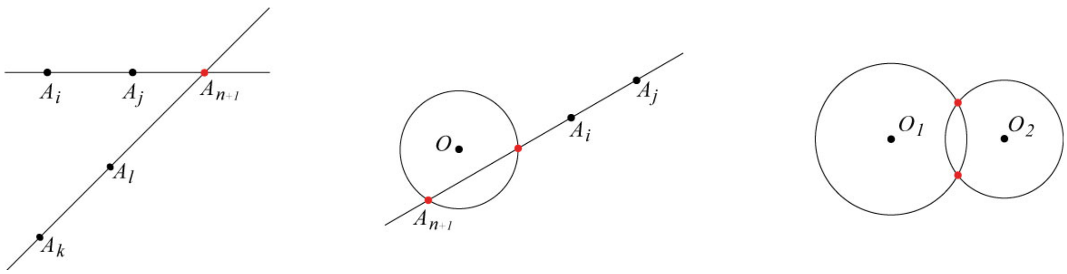
განმარტება 2.1: მათემატიკურ აღრიცხვაში შემოღებულია შემდეგი აღნიშვნები:

- $\forall$ ყოველი, ნებისმიერი;
- $\exists$ არსებობს;
- $\in$ ეკუთვნის;
- $\Rightarrow$ გამომდინარეობს

ახალი A_{n+1} წერტილი ითვლება ფარგლითა და სახაზავით აგებულად, თუ:

- $\exists\, A_i, A_j, A_k, A_l \in \mathcal{S}$ (იკითხება: ***არსებობს*** A_i, A_j, A_k, A_l, რომლებიც ***ეკუთვნის*** $\mathcal{S}$ სიმრავლეს) და A_{n+1} არის A_i, A_j წერტილებზე გავლებული წრფისა და A_k, A_l წერტილებზე გავლებული წრფის გადაკვეთის წერტილი (ნახ. 2.14 მარცხნივ);
- $\exists\, A_i, A_j, A_k, A_l, O \in \mathcal{S}$ და A_{n+1} არის A_i, A_j წერტილებზე გავლებული წრფისა და O წერტილზე $|A_k, A_l|$ სიგრძის რადიუსის მქონე წრეწირის გადაკვეთის წერტილი (ნახ. 2.14 შუაში);
- $\exists\, A_i, A_j, A_k, A_l, O_1, O_2 \in \mathcal{S}$ და A_{n+1} არის O_1 წერტილზე $|A_k, A_l|$ სიგრძის რადიუსის მქონე წრეწირისა და O_2 წერტილზე $|A_i, A_j|$ სიგრძის რადიუსის მქონე წრეწირის გადაკვეთის წერტილი (ნახ. 2.14 მარჯვნივ).

შენიშვნა: $A_i, A_j, A_k, A_l, O_1, O_2 \in \mathcal{S}$ წერტილთა შორის რამოდენიმე შეიძლება ერთმანეთს ემთხვეოდეს და $\mathcal{S}$ არის აქამდე აგებულ წერტილთა სიმრავლე.



ნახ. 2.14: ფარგლითა და სახაზავით ახალი წერტილების აგების შესაძლებლობები

რაიმე გეომეტრიული ფიგურა ითვლება აგებულად, თუ ფარგლითა და სახაზავით ზემოთ აღწერილი წესების დაცვით აიგება ისეთი სიმრავლე $\mathcal{S}$, რომ მასში მოიძებნოს ისეთი წერტილები, რომელთა ტეხილებით შეერთება ამ საძიებელ ფიგურას მოგვცემს.

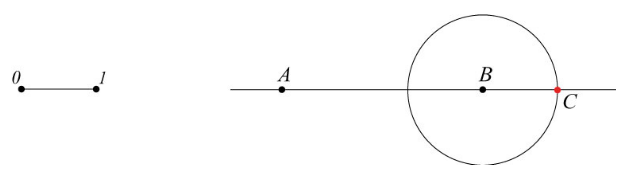
რაიმე რიცხვი ξ ითვლება აგებულად, თუ ფარგლითა და სახაზავით ზემოთ აღწერილი წესების დაცვით აიგება ისეთი სიმრავლე $\mathcal{S}$, რომ მასში მოიძებნოს ორი წერტილი, რომელთა შორის მანძილია ξ.

დასაწყისისათვის მოცემულია ორი წერტილი A' და B', რომელთა შორის მანძილი ერთის ტოლადაა მიჩნეული: $|A', B'| = 1$. სხვა სიტყვებით რომ ვთქვათ, აგებულია რიცხვი 1. იმისათვის, რომ ავაგოთ რიცხვი 2 (ანუ ფარგლისა და სახაზავის მეშვეობით ავაგოთ ისეთი წერტილები, რომელთა შორის მანძილი ორის ტოლია), შემდეგი ალგორითმი უნდა გამოვიყენოთ (ზოგადად, ეს ალგორითმი ააგებს ნებისმიერი მოცემული A და B წერტილისთვის რიცხვს $|A, B| + 1$):

მოცემულია: ორი წერტილი A და B.

1. A და B წერტილებზე გაავლე წრფე;
2. ფარგლით შემოხაზე წრეწირი ცენტრით B წერტილში და რადიუსით 1;

   ეს წრეწირი AB წრფეს გადაკვეთს ორ წერტილში: D (B წერტილიდან მარცხნივ) და ახალ C წერტილში, B წერტილიდან მარჯვნივ.
3. პასუხად გამოიტანე ორი წერტილი: A და C.



ნახ. 2.15: $|A, B| + 1$ სიგრძის მონაკვეთის აგება

ეს ალგორითმი აღვნიშნოთ როგორც N. თუ მისი მონაცემებია A და B წერტილები, $N(A, B) = (A, C)$. ადვილი საჩვენებელია, რომ $|A, C| = |A, B| + 1$.
ესე იგი, თუ მოცემულია ორი წერტილი A და B, რომელთა შორის მანძილია 1, შეიძლება $n \in \mathbb{N}$ რიცხვის აგება შემდეგი რეკურსიული ალგორითმით:

- $P_1 = (A, B)$;
- $P_n = N(P_{n-1})$.

სავარჯიშო 2.1: მოცემულია ოთხი წერტილი A, B, C, D. რა ალგორითმით შეიძლება $|A, B| + |C, D|$ სიგრძის მონაკვეთის აგება? გამოითვალეთ ამ ალგორითმის ბიჯების რაოდენობა და დაამტკიცეთ მისი სისწორე.

სავარჯიშო 2.2: მოცემულია ორი წერტილი A, B, სადაც $|A, B| > 1$. შეადგინეთ ალგორითმი, რომელიც $|A, B| - 1$ სიგრძის მონაკვეთს ააგებს. გამოითვალეთ ამ ალგორითმის ბიჯების რაოდენობა და დაამტკიცეთ მისი სისწორე.

თუ მოცემულია ორი წერტილი A და B, ადვილად შეიძლება $[A, B]$ მონაკვეთის შუა პერპენდიკულარული წრფის აგება, ანუ ისეთი ორი წერტილის აგება, რომლებზე გამავალი წრფეც ამ მონაკვეთის პერპენდიკულარულია და მის შუა წერტილზე გადის (ცხადია, რომ იგივე ალგორითმით შეიძლება ამავე მონაკვეთის შუა წერტილის დადგენა):

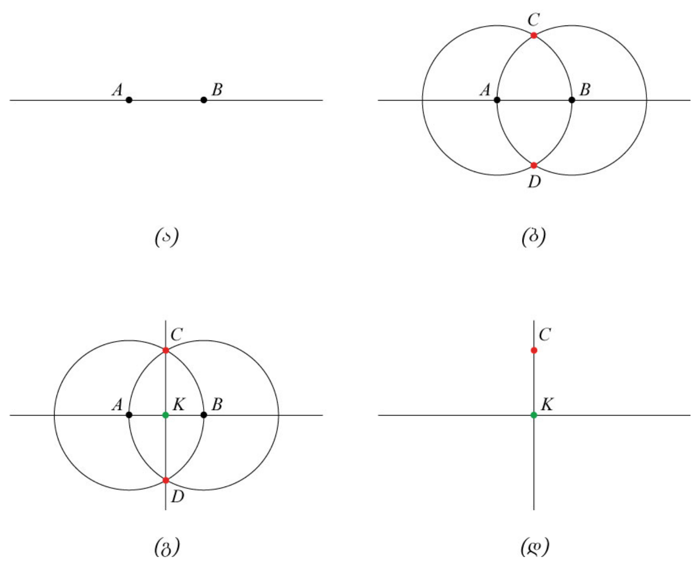
მოცემულია: ორი წერტილი A და B (ნახ. 2.16 *(ა)*).

- A წერტილზე შემოავლე $|A, B|$ რადიუსის წრეწირი;
- B წერტილზე შემოავლე $|A, B|$ რადიუსის წრეწირი (ნახ. 2.16 *(ბ)*)

  შედეგი: ამ ორი წრეწირის გადაკვეთის წერტილები C და D.
- შეაერთე C და D წერტილები წრფით (ნახ. 2.16 *(გ)*).

  შედეგი: ამ წრფისა და A, B მონაკვეთის გადაკვეთის წერტილი K.
- გამოიტანე პასუხი: ორი წერტილი C და K (ნახ. 2.16 *(დ)*).

სავარჯიშო 2.3: აჩვენეთ, რომ წინა ალგორითმით მიღებულ C და K წერტილებზე გავლებული წრფე $[A, B]$ მონაკვეთის შუა პერპენდიკულარულია.



ნახ. 2.16: $[A, B]$ მონაკვეთის შუა პერპენდიკულარულის აგება

ეს ალგორითმი აღვნიშნოთ როგორც $P(A, B)$. ამრიგად, $P(A, B) = (C, K)$, სადაც K $[A, B]$ მონაკვეთის შუა წერტილია.

თუ მოცემულია ორი წერტილი A და B და ერთი წერტილი C, რომელიც არ მდებარეობს (A, B) წრფეზე, მაშინ შეიძლება C წერტილიდან (A, B) წრფეზე პერპენდიკულარული წრფის დაშვება, ანუ ისეთი D წერტილის აგება (A, B) წრფეზე, რომ (C, D) წრფე (A, B) წრფის პერპენდიკულარული იყოს:

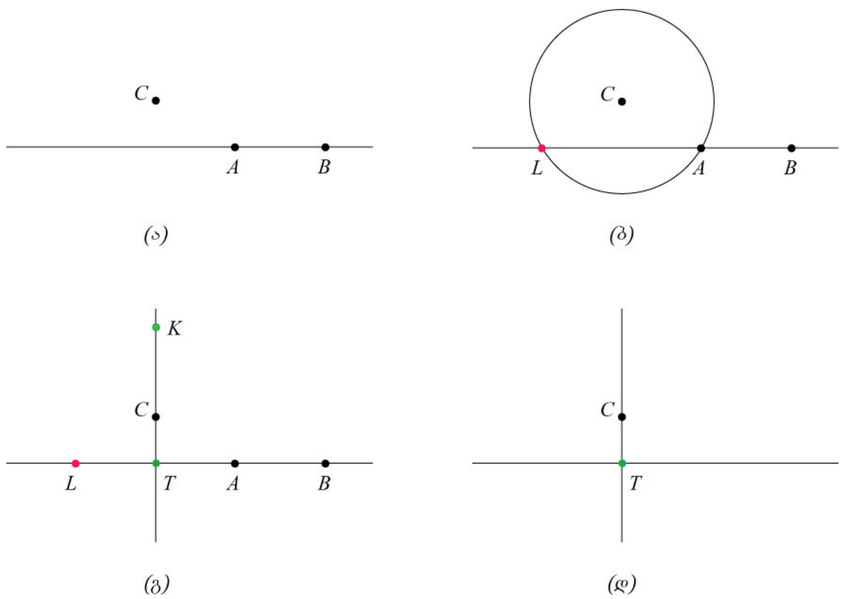
მოცემულია: ორი წერტილი A და B და ერთი წერტილი C, რომელიც არ დევს (A, B) წრფეზე (ნახ. 2.17 *(ა)*).

- C წერტილზე შემოავლე $|A, C|$ რადიუსის წრეწირი (ნახ. 2.17 *(ბ)*);

  შედეგი: ამ წრეწირისა და (A, B) წრფის გადაკვეთის მეორე წერტილი L.
- ჩაატარე ალგორითმი $P(A, L)$.

შედეგი: ორი წერტილი K და T, რომელთაგან T დევს (A, B) წრფეზე (ნახ. 2.17 *(გ)*).

- გამოიტანე პასუხი: ორი წერტილი C და T (ნახ. 2.17 *(დ)*).



ნახ. 2.17: წერტილიდან წრფეზე პერპენდიკულარულის დაშვების პროცესი

<u>სავარჯიშო 2.4:</u> ზემოთ მოყვანილ ალგორითმში A და L წერტილებზე უნდა ჩავატაროთ $P(A, L)$ ალგორითმი. დაწვრილებით აღწერეთ ნახაზებით ეს პროცესი, რომლის შედეგადაც მიიღება K და T წერტილები.

<u>სავარჯიშო 2.5:</u> რა მოხდება, თუ C წერტილში $|A, C|$ რადიუსით გავლებული წრეწირი (A, B) წრფეს მხოლოდ ერთ წერტილში გადაკვეთს და მეორე L წერტილი არ მიიღება?

<u>სავარჯიშო 2.6:</u> ზემოთ მოყვანილ ალგორითმში, $P(A, L)$ ალგორითმის შესრულების შემდეგ, რატომ მიიღება ორი დამატებითი წერტილი K და T?

<u>სავარჯიშო 2.7:</u> დაამტკიცეთ, რომ (C, T) წრფე (A, B) წრფის პერპენდიკულარულია.

<u>სავარჯიშო 2.8:</u> მოცემულია ერთ წრფეზე მყოფი სამი წერტილი A, B და მათ შორის მდებარე C. რა ალგორითმით შეიძლება C წერტილიდან (A, B) წრფის პერპენდიკულარული წრფის აგება?

<u>სავარჯიშო 2.9:</u> მოცემულია ორი წერტილი A და B და ერთი წერტილი C, რომელიც არ დევს (A, B) წრფეზე. რა ალგორითმით შეიძლება C წერტილიდან (A, B) წრფის პარალელური წრფის აგება (ანუ ისეთი D წერტილის აგება, რომ (C, D) წრფე (A, B) წრფის პარალელური იყოს)?

თუ აგებულია ორი რიცხვი $a_1, a_2 \in \mathbb{N}$, ანუ A_1, A_2, A_3, A_4 ისეთი, რომ $|A_1, A_2| = a_1$ და $|A_3, A_4| = a_2$, მაშინ შეიძლება ისეთი ორი B_1, B_2 წერტილის აგება ფარგლითა და სახაზავით, რომ $|B_1, B_2| = a_1 \cdot a_2$:

მოცემულია: ოთხი წერტილი A_1, A_2, A_3, A_4, სადაც $|A_1, A_2| = a_1$ და $|A_3, A_4| = a_2$.

- A_1 წერტილზე გაავლე (A_1, A_2) წრფის პერპენდიკულარული წრფე (ნახ. 2.18 *(ა)*) ;
- ამ წრფეზე A_1 წერტილიდან გადაზომე ერთის ტოლი მონაკვეთი;

  შედეგი: (A_1, A_2) წრფის პერპენდიკულარულ წრფეზე მდებარე წერტილი E, სადაც $|A_1, E| = 1$ (ნახ. 2.18 *(ბ)*).

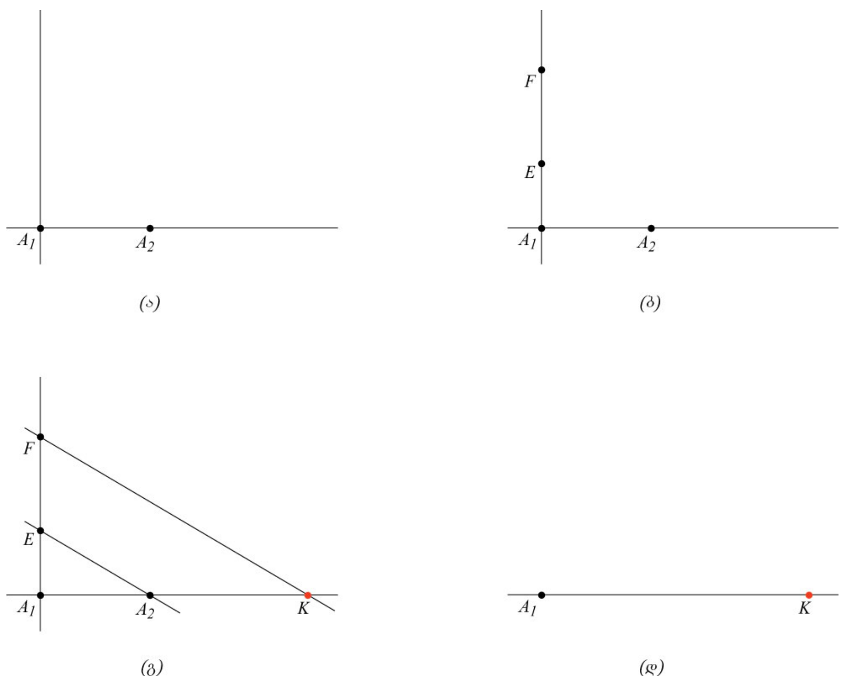
- იგივე წრფეზე A_1 წერტილიდან გადაზომე $|A_3, A_4|$ სიგრძის მონაკვეთი;

  შედეგი: (A_1, A_2) წრფის პერპენდიკულარულ წრფეზე მდებარე წერტილი F, სადაც $|A_1, F| = |A_3, A_4|$ (ნახ. 2.18 *(ბ)*).

- E და A_2 წერტილებზე გაავლე წრფე;
- F წერტილიდან გაავლე $|E, A_2|$ წრფის პარალელური წრფე;

  შედეგი: ამ წრფისა და (A_1, A_2) წრფის გადაკვეთის წერტილი K (ნახ. 2.18 *(გ)*).

- გამოიტანე პასუხი: ორი წერტილი A_1 და K (ნახ. 2.18 *(დ)*).



ნახ. 2.18: $|A_1, A_2| \cdot |A_1, F|$ სიგრძის მონაკვეთის აგების პროცესი

სავარჯიშო 2.10: სამკუთხედების მსგავსებით დაამტკიცეთ, რომ $|A_1, K| = a_1 \cdot a_2$.

სავარჯიშო 2.11: დაამტკიცეთ, რომ თუ $a_2 < 1$, ალგორითმი მაინც სწორად მუშაობს.

ანალოგიურად შეიძლება $\frac{a_1}{a_2}$ სიგრძის მონაკვეთის აგება, თუ მოცემულია ოთხი წერტილი A_1, A_2, A_3, A_4, სადაც $|A_1, A_2| = a_1$ და $|A_3, A_4| = a_2$.

მოცემულია: ოთხი წერტილი A_1, A_2, A_3, A_4, სადაც $|A_1, A_2| = a_1$ და $|A_3, A_4| = a_2$.

- A_1 წერტილზე გაავლე (A_1, A_2) წრფის პერპენდიკულარული წრფე (ნახ. 2.19 *(ა)*);
- ამ წრფეზე A_1 წერტილიდან გადაზომე ერთის ტოლი მონაკვეთი;

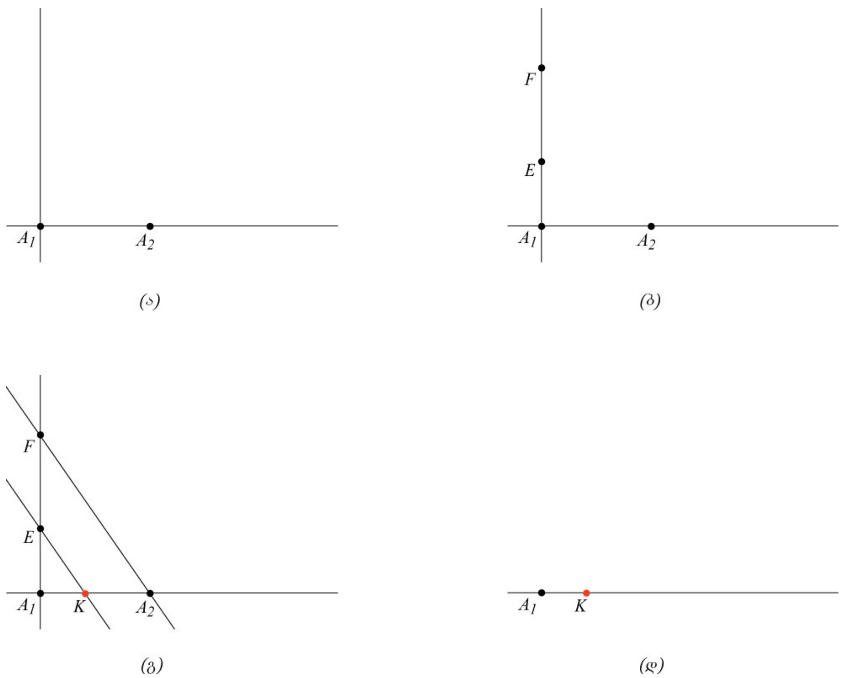
  შედეგი: (A_1, A_2) წრფის პერპენდიკულარულ წრფეზე მდებარე წერტილი E, სადაც $|A_1, E| = 1$ (ნახ. 2.19 *(ბ)*) .

- იგივე წრფეზე A_1 წერტილიდან გადაზომე $|A_3, A_4|$ სიგრძის მონაკვეთი;

  შედეგი: (A_1, A_2) წრფის პერპენდიკულარულ წრფეზე მდებარე წერტილი F, სადაც $|A_1, F| = |A_3, A_4|$ (ნახ. 2.19 *(ბ)*).

- F და A_2 წერტილებზე გაავლე წრფე;
- E წერტილიდან გაავლე $|F, A_2|$ წრფის პარალელური წრფე;

  შედეგი: ამ წრფისა და (A_1, A_2) წრფის გადაკვეთის წერტილი K (ნახ. 2.19 *(გ)*).
- გამოიტანე პასუხი: ორი წერტილი A_1 და K (ნახ. 2.19 *(დ)*).



ნახ. 2.19: $\frac{|A_1,A_2|}{|A_1,F|}$ სიგრძის მონაკვეთის აგების პროცესი

<u>სავარჯიშო 2.12:</u> სამკუთხედების მსგავსებით დაამტკიცეთ, რომ $|A_1, K| = \frac{a_1}{a_2}$.

ამრიგად ჩვენ გვაქვს ნებისმიერი რაციონალური რიცხვის აგების მეთოდი, ანუ ფარგლითა და სახაზავით მთლიანად შეიძლება აიგოს ნებისმიერი რაციონალურ რიცხვი $a \in \mathbb{Q}$.

ბუნებრივია შემდეგი შეკითხვა: შეიძლება თუ არა ირაციონალური რიცხვების აგება ფარგლითა და სახაზავით? პირველი ასეთი რიცხვი არის $\sqrt{2}$, რომელიც პითაგორას თეორემაზე დაყრდნობით აიგება:

მოცემულია: ორი წერტილი A_1, A_2.

- A_1 წერტილზე გაავლე (A_1, A_2) წრფის პერპენდიკულარული წრფე;
- ამ წრფეზე A_1 წერტილიდან გადაზომე ერთის ტოლი მონაკვეთი;

  შედეგი: (A_1, A_2) წრფის პერპენდიკულარულ წრფეზე მდებარე წერტილი E, სადაც $|A_1, E| = 1$.
- გამოიტანე პასუხი: ორი წერტილი A_2 და E.

<u>სავარჯიშო 2.13:</u> დახაზეთ ზემოთ მოყვანილი ალგორითმის დიაგრამები ისე, როგორც ეს წინა ალგორითმებისთვის იყო ნაჩვენები.

<u>სავარჯიშო 2.14:</u> დაამტკიცეთ, რომ $|A_2, E| = \sqrt{2}$, თუ $|A_1, A_2| = 1$.

ამ ალგორითმს ვუწოდოთ S. ესე იგი, თუ მოცემულია ორი წერტილი A, B ისე, რომ $|A, B| = \sqrt{a}$, მაშინ $S(A, B) = (B, C)$, სადაც $|B, C| = \sqrt{a+1}$.

აქედან გამომდინარეობს, რომ შემდეგი რეკურსიული ალგორითმი $H(n)$ ორ წერტილს გვაძლევს, რომელთა შორის მანძილია $\sqrt{n}$, $n \in \mathbb{N}$:

ალგორითმი $H(n)$:

- თუ $n = 1$, გამოიტანე ორი წერტილი A, B, სადაც $|A, B| = 1$ და ალგორითმი დაამთავრე;
- თუ $n > 1$:

  გაუშვი ალგორითმი $S(H(n-1))$.

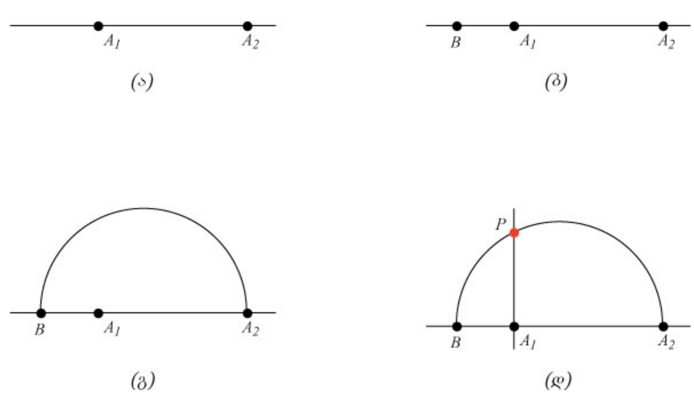
სავარჯიშო 2.15: ზემოთ მოყვანილი ალგორითმების საფუძველზე შეადგინეთ ალგორითმი, რომელიც ფესვს ნებისმიერი რაციონალური რიცხვიდან გამოიანგარიშებს.

ახლა კი განვიხილოთ შემდეგი ალგორითმი:

მოცემულია: ორი წერტილი A_1, A_2, სადაც $|A_1, A_2| = \xi$ (ნახ. 2.20 *(ა)*).

- A_1 წერტილის მარცხნივ (A_1, A_2) წრფეზე გადაზომე ერთის ტოლი მონაკვეთი და მიღებული წერტილი იყოს B, ანუ $|B, A_1| = 1$ (ნახ. 2.20 *(ბ)*);
- შემოავლე წრეწირი დიამეტრით $[B, A_2]$ (ნახ. 2.20 *(გ)*);
- A_1 წერტილიდან აღმართე (A_1, A_2) წრფის პერპენდიკულარული წრფე (ნახ. 2.20 *(დ)*);

  შედეგი: ამ წრფისა და წრეწირის გადაკვეთის წერტილი P (ნახ. 2.20 *(დ)*).
- გამოიტანე პასუხი: ორი წერტილი A_1 და P.



ნახ. 2.20: $\sqrt{|A_1, A_2|}$ სიგრძის მონაკვეთის აგების პროცესი

სავარჯიშო 2.16: სამკუთხედების მსგავსებით დაამტკიცეთ, რომ $|A_1, P| = \sqrt{|A_1, A_2|} = \sqrt{\xi}$.

სავარჯიშო 2.17: მოცემულია ორი წერტილი A და B. რა ალგორითმით შეიძლება წრეწირის შემოვლება, რომლის დიამეტრია $[A, B]$?

სავარჯიშო 2.18: მოცემულია სამი წერტილი O, A და B. O წერტილიდან გამოდის ორი სხივი $[O, A[$ და $[O, B[$, რომელიც O წერტილში ქმნის კუთხეს α. დაწერეთ ალგორითმი, რომელიც O, A და B მონაცემზე პასუხად მოგვცემს სამ წერტილს O, A და C ისე, რომ $\angle AOC = \frac{\alpha}{2}$.

ანტიკური ამოცანები:

- წრის კვადრატურა: მოცემულია O წერტილი და მის გარშემო შემოვლებული წრეწირი რადიუსით 1. ცხადია, რომ ამ წრის ფართობი იქნება π. შეიძლება თუ არა იგივე ფართობის კვადრატის აგება მხოლოდ ფარგლისა და სახაზავის გამოყენებით?

- მესამე ხარისხის ფესვი: მოცემულია ორი წერტილი, რომელთა შორის მანძილია a. შეიძლება თუ არა მხოლოდ ფარგლითა და სახაზავით ისეთი ორი წერტილის აგება, რომელთა შორის მანძილია $\sqrt[3]{a}$?

- სამმაგი ბისექტრისა: მოცემულია სამი წერტილი O, A და B. O წერტილიდან გამოდის ორი სხივი $[O, A[$ და $[O, B[$, რომელიც O წერტილში ქმნის კუთხეს α. შეიძლება თუ არა მხოლოდ ფარგლისა და სახაზავის გამოყენებით ავაგოთ ისეთი წერტილი C, რომ $\angle AOC = \frac{\alpha}{3}$?

- წესიერი მრავალკუთხედები: რამდენკუთხა წესიერი მრავალკუთხედის აგება შეიძლება მხოლოდ ფარგლისა და სახაზავის გამოყენებით? (ამოზნექილ მრავალკუთხედს ეწოდება წესიერი, თუ მისი ყველა გვერდის სიგრძე ერთმანეთის ტოლია.)

როგორც აღმოჩნდა, პირველი სამი ამოცანა ამოუხსნადია: არ არსებობს ისეთი ალგორითმი, რომელიც მხოლოდ ფარგლისა და სახაზავის მეშვეობით ააგებს ორ წერტილს, რომელთა შორის მანძილია π; ან ისეთი ალგორითმი, რომელიც ნებისმიერი a რიცხვიდან მესამე ხარისხის ფესვს ამოიღებს ან ისეთი ალგორითმი, რომელიც ნებისმიერ კუთხეს სამად გაყოფს (ისე, როგორც ის ალგორითმი, რომელიც ნებისმიერი რიცხვიდან კვადრატულ ფესვს ამოიღებს ან ნებისმიერ კუთხეს ორად გაყოფს).

ამის დამტკიცების იდეა შემდეგია:

ახალი წერტილის აგება შეიძლება მხოლოდ როგორც უკვე აგებულ წერტილებზე გავლებული ორი წრფის, ორი წრეწირისა ან ერთი წრეწირისა და ერთი წრფის გადაკვეთის წერტილისა. თუ ავაგებთ ორი გეომეტრიული ფიგურის გადაკვეთის წერტილს, მაშინ მისი დაშორება კოორდინატთა სათავიდან გამოითვლება შემდეგი პოლინომიური განტოლების ამონახსნით: $a_{2^n}x^{2^n} + a_{2^n-1}x^{2^n-1} + \cdots + a_1x + a_0 = 0$, სადაც n რაღაცა ნატურალური რიცხვია.
რადგან $\sqrt[3]{a}$ არ არის ასეთი სახის პოლინომის (ანუ ორის ხარისხის რიგის პოლინომის) ამონახსნი, ამიტომ ამ რიცხვის ფარგლითა და სახაზავით აგება შეუძლებელია.
როგორც XIX საუკუნეში გერმანელმა მათემატიკოსმა ლინდემანმა დაამტკიცა, π ტრანსცენდენტული რიცხვია, ანუ იგი არ არის არანაირი პოლინომიური განტოლების ამონახსნი და მით უმეტეს ვერ იქნება ორის ხარისხის რიგის განტოლების ამონახსნი, რითაც მტკიცდება, რომ ფარგლითა და სახაზავით π რიცხვის აგება შეუძლებელია.
მაგრამ არსებობს ფორმულა, რომელიც გვეუბნება, თუ რამდენ კუთხა წესიერი მრავალკუთხედის აგება შეიძლება მხოლოდ ფარგლისა და სახაზავის გამოყენებით: n კუთხა წესიერი მრავალკუთხედის აგება შეიძლება მაშინ და მხოლოდ მაშინ, თუ $\exists m, q_1, ..., q_l \in \mathbb{N}_0$ ისე, რომ $n = 2^m \cdot (2^{2^{q_1}} + 1) \cdot (2^{2^{q_2}} + 1) \cdots (2^{2^{q_l}} + 1)$.
ამ ფორმულიდან გამომდინარე შეიძლება წესიერი ხუთკუთხედის, ცხრამეტკუთხედისა და 65537 კუთხედის აგება, მაგრამ არ შეიძლება წესიერი 7-კუთხედის აგება.

სავარჯიშო 2.19: შეადგინეთ ალგორითმი, რომლის მეშვეობითაც შეიძლება წესიერი ექვსკუთხედის აგება.

სავარჯიშო 2.20: შეადგინეთ ალგორითმი, რომლის მეშვეობითაც შეიძლება წესიერი რვაკუთხედის აგება.

სავარჯიშო 2.21: შეადგინეთ ალგორითმი, რომლის მეშვეობითაც შეიძლება წესიერი ხუთკუთხედის აგება.

შენიშვნა: ზემოთ მოყვანილი ამოცანებისათვის არ არსებობს ალგორითმი, რომელიც მხოლოდ ფარგლითა და სახაზავით აგვაგებინებდა საჭირო წერტილებსა და ფიგურებს. ეს კი იმას არ ნიშნავს, რომ არ არსებობს სხვა რაიმე მეთოდი (თუ არ შევიზღუდებით მხოლოდ ფარგლითა და სახაზავით), რითაც ამ ამოცანებს გადავჭრით.

ღია ამოცანა: წესიერი მრავალკუთხედის ზემოთ მოყვანილ ფორმულაში $2^{2^q} + 1$ ე.წ. ფერმას მარტივი რიცხვია. დიდ ხანს ეგონათ, რომ ეს ფორმულა მხოლოდ მარტივ რიცხვებს იძლეოდა, მაგრამ აღმოჩნდა, რომ ეს ასე არაა. უფრო მეტიც: ეს ფორმულა ძირითადად შედგენილ რიცხვებს იძლევა. მაგრამ მნიშვნელოვანია შემდეგი საკითხი: სასრულია თუ არა ფერმას მარტივ რიცხვთა სიმრავლე? ან, სხვა სიტყვებით რომ ვთქვათ, შეგვხვდება თუ არა მიმდევრობაში $(2^{2^q}+1)_{q=0}^{\infty}$ უსასრულოდ ბევრი მარტივი რიცხვი? ამ შეკითხვაზე პასუხი ჯერ-ჯერობით უცნობია.

2.4 მოკლე დასკვნა

მეორე თავში განვიხილეთ ალგორითმების ე.წ. რეკურსიული და იტერაციული აღწერა. რეკურსიულ მეთოდში ალგორითმი თავის თავს იყენებს, მხოლოდ უფრო დაბალი პარამეტრებით, ხოლო იტერაციულში კი ბრძანებათა გარკვეულ მიმდევრობას რამოდენიმეჯერ იმეორებს, ანუ ე.წ. ციკლს ქმნის.
ორივე მეთოდს თავისი დადებითი და უარყოფითი მხარეები აქვს, რომელიც კონკრეტულ ამოცანასა და მის რეალიზაციაზეა დამოკიდებული. ამ საკითხებს ჩვენ შემდგომში დაწვრილებით განვიხილავთ.

ამოცანების რეკურსიული აღწერის კარგ მაგალითებს სიბრტყეზე ფარგლითა და სახაზავით წერტილებისა და ფიგურების აგების კლასიკური, ძველი ბერძნული ამოცანები წარმოადგენს, რომლებიც ათასწლეულის განმავლობაში ამოუხსნელი რჩებოდა და მხოლოდ მეცხრამეტე საუკუნეში, მათემატიკის საკმარისი განვითარების შედეგად გადაიჭრა.
წარმოდგენილი ამოცანები საინტერესოა იმ თვალსაზრისითაც, რომ აქ პირველად ჩანს რაღაც კონკრეტული პრობლემის ამოუხსნადობა: გარკვეულ ამოცანას ვერავინ ამოხსნის. ზოგადად, იმის დამტკიცება, რომ გარკვეულ არაამოხსნად ამოცანებზე ჩვენ შემდგომშიც ბევრი გვექნება სალაპარაკო, აქ კი შეიძლება იმის აღნიშვნა, რომ ზოგიერთი კლასიკური ამოცანის (მაგალითად, წრის კვადრატურის) ამოუხსნადობა რესურსების შეზღუდვასთანაა დაკავშირებული: შეუძლებელია ერთეულოვანი წრის ფართობის მქონე კვადრატის აგება ***მხოლოდ ფარგლისა და სახაზავის*** გამოყენებით. ზოგადად არაამოხსნადი ამოცანებისგან განსხვავებით, რომელთა ამოხსნა არანაირი რესურსით არ შეიძლება, ეს ამოცანა სხვა დამატებით რესურსების გამოყენებით ამოხსნადი ხდება.
მაგრამ ეს უკვე შემდგომი თემაა, რასაც ჩვენ კიდევ დაწვრილებით განვიხილავთ.

თავი 3

მათემატიკური ინდუქცია და მისი გამოყენება

3.1 მათემატიკური ინდუქცია

განვიხილოთ კენტ რიცხვთა მიმდევრობა:

$a_1 = 1, a_2 = 3, a_3 = 5, a_4 = 7, ...$

ადვილი შესამჩნევია რომ ამ მიმდევრობის ნებისმიერი წევრი შემდეგნაირად ჩაიწერება: $a_i = 2 \cdot i - 1$. ახლა კი გამოვიანგარიშოთ ამ მიმდევრობის პირველი n წევრის ჯამი:

$$S_n = a_1 + a_2 + a_3 + \cdots + a_n.$$

აღსანიშნავია, რომ პირველი n კენტი რიცხვის ჯამი რეკურსიულად შემდეგნაირად ჩაიწერება:

$$S_n = S_{n-1} + a_n$$

(პირველი $n-1$ კენტი რიცხვის ჯამს მიმატებული მე-n-ე კენტი რიცხვი).

<u>სავარჯიშო 3.1:</u> რეკურსიულად ჩაწერეთ $S_{n+1}, S_{n-1}, S_{n-2}$ და S_{n-3}.

პირველ რიგში განვიხილოთ რამოდენიმე კონკრეტული მაგალითი:

$$\begin{array}{lllll} S_1 = & a_1 & = & 1 \\ S_2 = & a_1 + a_2 & = & 4 \\ S_3 = & a_1 + a_2 + a_3 & = & 9 \\ S_4 = & a_1 + a_2 + a_3 + a_4 & = & 16 \\ S_5 = & a_1 + a_2 + a_3 + a_4 + a_5 & = & 25 \\ \cdots \end{array}$$

თუ ამ ცხრილის მარჯვენა მხარეს დავაკვირდებით, დავინახავთ, რომ იქ ნატურალური რიცხვების კვადრატები წერია: $S_1 = 1^2, S_2 = 2^2, S_3 = 3^2, S_4 = 4^2, S_5 = 5^2$.
ეს გვაწვდის პირველ მოსაზრებას იმის შესახებ, თუ რისი ტოლი შეიძლება იყოს ზოგადად პირველი n კენტი რიცხვის ჯამი: $S_i = i^2$.
მაგრამ ეს მხოლოდ მოსაზრებაა, რომელსაც დამტკიცება ჭირდება. ეს მოსაზრება თვით მილიონი მაგალითის გადამოწმებით ვერ დამტკიცდება: მილიონ მეერთე მაგალითი *შეიძლება* ამ მოსაზრებას არ აკმაყოფილებდეს. ასე რომ, საჭიროა რაღაცა ზოგადი მეთოდი, რომლითაც ასეთ რამეებს დავამტკიცებთ.
სწორედ ასეთია ე.წ. **მათემატიკური ინდუქციის** მეთოდი, რომელიც შემდეგი სამი ბიჯისაგან შედგება:

1. **ინდუქციის შემოწმება:** გადავამოწმოთ მოსაზრება $n = 1$ შემთხვევისათვის;
2. **ინდუქციის დაშვება:** დავუშვათ, რომ მოსაზრება ჭეშმარიტია $\forall k = 1, 2, ..., n$;

3. **ინდუქციის ბიჯი:** დავამტკიცოთ მოსაზრება $n+1$-თვის.

ამ სამი ბიჯის შესრულებისას შემდეგს მივაღწევთ: თუ მოსაზრება ჭეშმარიტია $n=1$-თვის, მე-2-ე ბიჯში ჩავსვავთ $n=1$. თუ მოსაზრება ჭეშმარიტია აგრეთვე $n+1$-თვის (მესამე პუნქტი), იგი ჭეშმარიტია 2-თვის. ესე იგი, შეიძლება მეორე პუნქტში ჩავსვათ $n=2$ და, მესამე პუნქტიდან გამომდინარე, მოსაზრება ჭეშმარიტია $n+1=2+1=3$ -თვის. ანალოგიური მსჯელობით დავამტკიცებთ ჭესმარიტებას $n=4$, $n=5$, $n=6$... შემთხვევებში, რითაც ეს მოსაზრება ნებისმიერი ნატურალური n -თვის ჭეშმარიტი იქნება.

ჩვენს ზემოთ მოყვანილ მაგალითში ეს ასე იქნება:

1. **ინდუქციის შემოწმება:** $n=1$: $S_1=1^2=1$;
2. **ინდუქციის დაშვება:** დავუშვათ, რომ $S_n=n^2$;
3. **ინდუქციის ბიჯი:** დავამტკიცოთ $S_{n+1}=(n+1)^2$.
   რეკურსიული ფორმულის თანახმად, $S_{n+1}=S_n+a_{n+1}=S_n+2\cdot(n+1)-1=S_n+2\cdot n+1$.
   ინდუქციის დაშვების თანახმად $S_n=n^2$ და ზედა ფორმულაში ჩასმით ვიღებთ: $S_{n+1}=n^2+2\cdot n+1=(n+1)^2$.
   მოსაზრება დამტკიცებულია.

როდესაც რეკურსიული ფორმულა ჩაიწერება არარეკურსიული სახით (ანუ ფორმულაში მხოლოდ ცვლადები და მუდმივები გვხვდება), ამბობენ, რომ რეკურსია გაიშალა და ფორმულა ჩაიწერა არარეკურსიული სახით.

ზემოთ აღწერილი პრინციპით დავამტკიცოთ კიდევ ერთი მათემატიკური მოსაზრება:

რეკურსიული ტოლობით მოცემულია მიმდევრობა $S_1=1, S_n=S_{n-1}+n$. დაამტკიცეთ, რომ $S_n=\frac{n\cdot(n+1)}{2}$.

1. **ინდუქციის შემოწმება:** $n=1$: $S_1=\frac{1\cdot(1+1)}{2}=1$;
2. **ინდუქციის დაშვება:** $S_n=\frac{n\cdot(n+1)}{2}$;
3. **ინდუქციის ბიჯი:** დავამტკიცოთ $S_{n+1}=\frac{(n+1)\cdot(n+2)}{2}$..
   რადგან $S_{n+1}=S_n+(n+1)$, ამიტომ, ინდუქციის დაშვების თანახმად, $S_{n+1}=\frac{n\cdot(n+1)}{2}+(n+1)=(n+1)(\frac{n}{2}+1)=\frac{(n+1)\cdot(n+2)}{2}$.

სავარჯიშო 3.2: მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ, რომ ნებისმიერი კენტი რიცხვი შემდეგი ფორმულით ჩაიწერება: $a_i=2\cdot i-1$.

სავარჯიშო 3.3: მოცემულია რეკურსიული ტოლობა $S_1=3$, $S_n=S_{n-1}+n$. გახსენით რეკურსია (ტოლობა ჩაწერეთ არარეკურსიული სახით).

სავარჯიშო 3.4: მოცემულია რეკურსიული ტოლობა $K_1=7$, $K_n=K_{n-1}+2n$. გახსენით რეკურსია (ტოლობა ჩაწერეთ არარეკურსიული სახით).

სავარჯიშო 3.5: მოცემულია რეკურსიული ტოლობა $P_1=1$, $P_n=P_{n-1}+2^n$. გახსენით რეკურსია (ტოლობა ჩაწერეთ არარეკურსიული სახით).

სავარჯიშო 3.6: მოცემულია რეკურსიული ტოლობა $L_1=7$, $L_n=2\cdot L_{n-1}$. გახსენით რეკურსია (ტოლობა ჩაწერეთ არარეკურსიული სახით).

3.2 მათემატიკური ინდუქციის გამოყენება

განვიხილოთ წინა თავში მოყვანილი ნავების ალგორითმის რეკურსიული ჩანაწერი $A_n=A_1,U,A_n$. მათემატიკური ინდუქციით შეიძლება მისი სისწორის მტკიცება:

- ინდუქციის დასაწყისი: A_1 ალგორითმი სწორია (ამის გადამოწმება ადვილია);
- ინდუქციის დაშვება: დავუშვათ, A_n ალგორითმი სწორია რაღაცა n ნატურალური რიცხვისათვის (და მასზე პატარა ყველა რიცხვისათვის);
- ინდუქციის ბიჯი: დავამტკიცოთ, რომ $A_{n+1} = A_1, U, A_n$ სწორია.

თუ დავამტკიცებთ, რომ A_{n+1} ალგორითმი სწორია და გვეცოდინება, რომ A_1 სწორია, მაშინ დავუშვებთ, რომ $n = 1$ და ამით დამტკიცდება, რომ $A_{n+1} = A_2$ სწორია. თუ A_2 სწორია და დამტკიცებული გვექნება, რომ A_{n+1} სწორია, დამტკიცდება, რომ A_3 სწორია და ა.შ. ნებისმიერი ნატურალური რიცხვისათვის.

ახლა კი დავამტკიცოთ $A_{n+1} = A_1, U, A_n$ ალგორითმის სისწორე: A_1, U ალგორითმების შესრულების შემდეგ წარმოიშვება ზუსტად ისეთივე სიტუაცია, როგორც n ნავის გაყვანის ამოცანაში. ხოლო ინდუქციის დაშვების თანახმად A_n ალგორითმი n ნავის გაყვანის ამოცანას სწორად ხსნის. ასე რომ, A_1, U, A_n $n+1$ ნავის გაყვანის ამოცანას სწორად ხსნის.

Q.E.D.

სავარჯიშო 3.7: სწორად ამოხსნის თუ არა შემდეგი ალგორითმი $A_n = A_{n-1}, U, A_1$ n ნავის გაყვანის ამოცანას?

ალგორითმის სისწორის მტკიცების შემდეგ საჭიროა მისი სისწრაფის, ანუ ბიჯების რაოდენობის დადგენა. A ალგორითმის ბიჯების რაოდენობას შემდეგნაირად აღნიშნავენ: $T(A)$. ჩვენს შემთხვევაში გვექნება $T(A_n)$.
რადგან ჯერ უნდა შევასრულოთ ალგორითმი A_1, ამის შემდეგ ალგორითმი U და ბოლოს ალგორითმი A_{n-1}, მაშინ A_n ალგორითმის ბიჯების რაოდენობა იქნება: $T(A_n) = T(A_1) + T(U) + T(A_{n-1})$ (ჯერ იმდენი, რამდენიც საჭიროა A_1 ალგორითმისათვის, შემდეგ იმდენი, რამდენიც საჭიროა U ალგორითმისათვის და ბოლოს იმდენი, რამდენიც საჭიროა A_{n-1} ალგორითმისათვის).
ეს ფორმულაც ჩაწერილია რეკურსიული სახით, რადგან იგი თავის თავს იყენებს, მხოლოდ უფრო დაბალი პარამეტრებით. მაგრამ მისი ჩაწერა არარეკურსიული სახითაც შეიძლება:

ჩვენ ვიცით, რომ $T(A_1) = 3$ და $T(U) = 1$ (შესაბამისი ალგორითმების გადამოწმებით ამაში ადვილად ვრწმუნდებით). აქედან გამომდინარე, ვიღებთ:

$$T(A_n) = T(A_{n-1}) + 4.$$

თავის მხრივ, $T(A_{n-1}) = T(A_{n-2}) + 4$, $T(A_{n-2}) = T(A_{n-3}) + 4$ ...

აქედან გამომდინარე,

$$T(A_n) = T(A_{n-1}) + 1 \cdot 4 = T(A_{n-2}) + 2 \cdot 4 = T(A_{n-3}) + 3 \cdot 4 = \cdots = T(A_1) + (n-1) \cdot 4 = 3 + (n-1) \cdot 4 = 4 \cdot n - 1.$$

სავარჯიშო 3.8: დაამტკიცეთ, რომ ამაზე უფრო სწრაფი ალგორითმი ვერ იარსებებს.

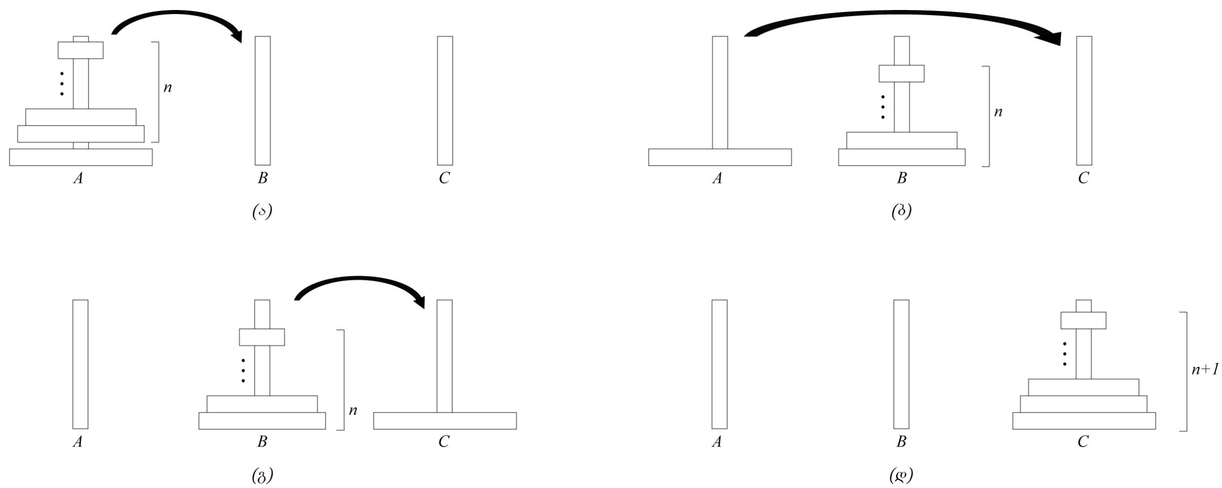
ანალოგიურად შეიძლება ჰანოის კოშკების $H_n^{X_1,X_2}$ ალგორითმის სისწორის მტკიცება და ბიჯების რაოდენობის გამოთვლა:

$H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}$.

- ინდუქციის დასაწყისი: $H_1^{X_1,X_2}$ ალგორითმი სწორია (ამის გადამოწმება ადვილია);
- ინდუქციის დაშვება: დავუშვათ, $H_n^{X_1,X_2}$ ალგორითმი სწორია რაღაცა n ნატურალური რიცხვისათვის (და მასზე პატარა ყველა რიცხვისათვის);
- ინდუქციის ბიჯი: დავამტკიცოთ, რომ $H_{n+1}^{X_1,X_2} = H_n^{X_1,X_3}, H_1^{X_1,X_2}, H_n^{X_3,X_2}$ სწორია.

  პირველ რიგში $H_n^{X_1,X_3}$ ალგორითმით X_1 ძელიდან ზედა n რგოლი X_3 ძელზე უნდა გადავიტანოთ (ნახ. 3.1(ა)). ინდუქციის დაშვების თანახმად ეს პროცედურა სწორად შესრულდება (აქ გასათვალისწინებელია, რომ X_1 ძელზე ყველაზე დიდი რგოლი რჩება, რომელზეც ყველა დანარჩენი რგოლის დადება შეიძლება, რაც ამოცანის შეძლუდვას არ არღვევს. შემდეგ X_1 ძელზე დარჩენილ დიდ რგოლს გადავიტანთ X_2 ძელზე (ნახ.

3.1(ბ)), რის შემდეგაც $H_n^{X_3,X_2}$ ალგორითმით n რგოლს გადავიტანთ X_3 ძელიდან X_2 ძელზე (ნახ. 3.1(გ)). აქაც უნდა გავითვალისცინოთ, რომ, ინდუქციის დაშვების თანახმად, $H_n^{X_3,X_2}$ ალგორითმი ყველა წესის დაცვით მოქმედებს და X_2 ძელზე ყველაზე დიდი რგოლი დევს, რომელზედაც ნებისმიერი სხვა რგოლის დადება შეიძლება. შედეგად მივიღებთ $n+1$ რგოლს მესამე ძელზე (ნახ. 3.1(დ)).
ქვემოთ მოყვანილ ნახატში $X_1 = A, X_2 = C, X_3 = B$.



ნახ. 3.1: $n+1$ რგოლიანი პირამიდის გადატანისათვის საჭირო ოპერაციები

იმისათვის, რომ დავადგინოთ, თუ რამდენ ბიჯს ანდომებს ეს ალგორითმი, განვიხილოთ მისი რეკურსიული ჩანაწერი:

$H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}$.

ადვილი საჩვენებელია, რომ

$T(H_n^{X_1,X_2}) = T(H_{n-1}^{X_1,X_3}) + T(H_1^{X_1,X_2}) + T(H_{n-1}^{X_3,X_2})$.

სავარჯიშო 3.9: რას აღნიშნავს $T(H_{n+3}^{A,C}), T(H_3^{C,B}), T(H_7^{A,C})$?
სავარჯიშო 3.10: რისი ტოლია $T(H_1^{A,C})$ და $T(H_2^{A,C})$?
სავარჯიშო 3.11: დაამტკიცეთ, რომ $T(H_1^{A,C}) = T(H_1^{B,C})$ და ზოგადად: $T(H_n^{X_1,X_2}) = T(H_n^{Y_1,Y_2})\ \forall X_1, X_2, Y_1, Y_2 \in \{A, B, C\}$ (არ აქვს მნიშვნელობა, რომელი ძელიდან რომელზე გადავაწყობთ პირამიდას - ბიჯების რაოდენობა უცვლელია).

რადგან $H_n^{X_1,X_2} = H_{n-1}^{X_1,X_3}, H_1^{X_1,X_2}, H_{n-1}^{X_3,X_2}$, ჯერ უნდა შესრულდეს $H_{n-1}^{X_1,X_3}$, შემდეგ $H_1^{X_1,X_2}$ და ბოლოს $H_{n-1}^{X_3,X_2}$. აქედან გამომდინარე,

$$T(H_n^{X_1,X_2}) = T(H_{n-1}^{X_1,X_3}) + T(H_1^{X_1,X_2}) + T(H_{n-1}^{X_3,X_2}) = 2 \cdot T(H_{n-1}^{X_1,X_2}) + 1$$

(იხ. წინა სავარჯიშოები).

სავარჯიშო 3.12: მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ:

$$T(H_n^{X_1,X_2}) = 2^n - 1.$$

სავარჯიშო 3.13: განიხილეთ n ნავის გაყვანის იტერაციული ალგორითმი. მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ მისი სისწორე და გამოითვალეთ ბიჯების რაოდენობა.

სავარჯიშო 3.14: განიხილეთ n რგოლიანი ჰანოის კოშკის იტერაციული ალგორითმი და გამოითვალეთ მისი ბიჯების რაოდენობა.

შენიშვნა: ამ იტერაციული ალგორითმის ბიჯების დათვლა მარტივი არაა, რადგან არაა ცხადი, რამდენჯერ უნდა შესრულდეს ციკლი.

ჰანოის კოშკების ამოცანის მაგალითზე შეიძლება დავინახოთ ალგორითმების იტერაციული აღწერის უარყოფითი მხარე. თუ რეკურსიულ შემთხვევაში როგორც ბიჯების დათვლა, ასევე სისწორის მტკიცება მათემატიკური ინდუქციის გამოყენებით შედარებით ადვილია, იტერაციულ შემთხვევაში ეს საკმაოდ რთულდება.
მაგრამ არსებობს მაგალითები, რომლის განხილვისას აშკარაა იტერაციული ალგორითმის ეფექტურობა რეკურსიულთან შედარებით. ასეთი მაგალითია ფიბონაჩის მიმდევრობის გამოთვლის ამოცანა, რომელსაც ჩვენ ქვემოთ განვიხილავთ.

ზემოთ თქმულიდან გამომდინარეობს, რომ ყოველი ამოცანის იმპლემენტაციისას (ანუ რეალიზაციისას) დეტალურადაა გასაანალიზებელი მისი შინაარსი, მოთხოვნები, შეზღუდვები, მონაცემთა ტიპები, რომ მისთვის ეფექტური ალგორითმი შევქმნათ.

3.3 ფიბონაჩის მიმდევრობა

ცნობილმა იტალიელმა მეცნიერმა ლეონარდო და პიზამ (Leonardo da Pisa), რომელიც მეთორმეტე საუკუნის ბოლოსა და მეცამეტე საუკუნის დასაწყისში ცხოვრობდა და უფრო ***ფიბონაჩის*** სახელითაა ცნობილი (Fibonacci), შემდეგი ამოცანის გადაჭრა გადაწყვიტა:

გლეხი ზრდის კურდღლებს. ყოველი კურდღელი ბადებს ერთ კურდგელს, როდესაც ორი თვის გახდება და შემდეგ თითო კურდღელს ყოველთვიურად. რამდენი ***დედალი*** კურდღელი ეყოლება გლეხს n თვეში, თუ ჩავთვლით, რომ კურდღლები არ კვდებიან?

თუ n მცირეა, რაოდენობის გამოთვლა არაა რთული: პირველ და მეორე თვეში მას 1 კურდღელი ჰყავს, რადგან კურდღელი მხოლოდ ორი თვის შემდეგ იძლევა შთამომავლობას. მესამე თვეს მას 2 კურდღელი ეყოლება, ხოლო მეოთხეში კი 3, რადგან პირველმა კურდღელმა მისცა კიდევ 1 და მეორე ჯერ ორი თვის არაა. ამის შემდეგ მისი პირველი და მეორე კურდღელი ორივე შთამომავლობას იძლევა, ასე რომ, მეხუთე თვეში მას 5 კურდღელი ეყოლება. ზოგადად, მე-n-ე თვეში ახლად შემომატებულ კურდღელთა რიცხვი ტოლია იმ კურდღელთა რიცხვისა, რომლებიც სულ ცოტა 2 თვის არიან. აქედან გამომდინარე, თუ მე-n-ე თვეში კურდღელთა რაოდენობას აღვნიშნავთ როგორც F_n, მივიღებთ:

$$F_n = F_{n-1} + F_{n-2}$$

(ამ ტოლობას ფიბონაჩის პირობასაც უწოდებენ).

ჩვენ ვიცით აგრეთვე, რომ $F_1 = 1, F_2 = 1, F_3 = 3, F_4 = 5$. ტექნიკური მიზეზებით განსაზღვრავენ აგრეთვე $F_0 = 0$, რაც შემდეგნაირად განსაზღვრავს ე.წ. ფიბონაჩის მიმდევრობას:

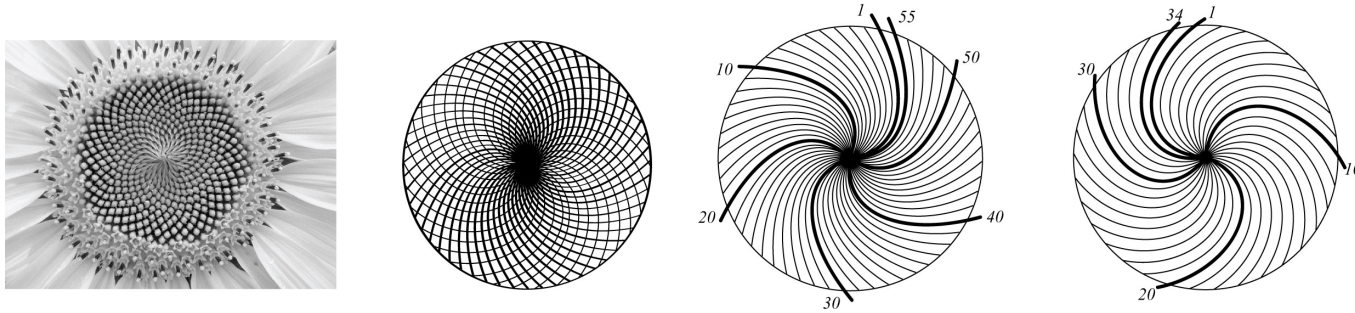
$$F_0 = 0;\ F_1 = 1;\ F_n = F_{n-1} + F_{n-2},$$

სადაც $n > 1$. ამ რეკურსიული ფორმულით გამოთვლილი რამოდენიმე რიცხვია:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, \ldots$

ასეთი სახით მიღებულ რიცხვებს ფიბონაჩის რიცხვებს უწოდებენ, ხოლო ამ მიმდევრობას -- ფიბონაჩის მიმდევრობას. ამას გარდა, F_n და F_{n+1} მეზობელი რიცხვებია.

აღსანიშნავია, რომ ეს მიმდევრობა ანტიკური ხანის საბერძნეთსა და შუა საუკუნეების ინდოეთშიც იყო ცნობილი. ზოგჯერ მის ნულოვან წევრს $F_0 = 0$ არ განიხილავენ ხოლმე და მის პირველ ორ წევრად F_1 და F_2 იღებენ.
როგორც აღმოჩნდა, ზემოთ მოყვანილი ფორმულა კურდღელთა რაოდენობას არასწორად ითვლის, მაგრამ სამაგიეროდ ფიბონაჩის რიცხვები ძალიან ხშირად გვხვდება ბუნებაში და მეცნიერებაშიც დიდ როლს თამაშობენ. თვით ეს მიმდევრობაც ბევრ საინტერესო თვისებას ავლენს.
ასე, მაგალითად, მზესუმზირას ნაყოფში თესლი განთავსებულია მომრგვალებულ წირებზე, რომელთა სქემაც ქვედა ნახატშია მოყვანილი (ნახ. 3.2).



ნახ. 3.2: მზესუმზირას ნაყოფში თესლის განთავსება

ამ ნახაზებიდან ჩანს, რომ თესლი ორი საპირისპიროდ მიმართული ფიგურის მსგავსადაა განლაგებული, სადაც წირების რაოდენობებია 55 და 34, რაც ორი მეზობელი ფიბონაჩის რიცხვია.
ამას გარდა, ხეებში ტოტების განშტოებისა ან ყვავილების ფურცლების რიცხვი ძირითადად ფიბონაჩის მიმდევრობის ერთ-ერთი წევრის ტოლია ხოლმე (მრავლად მოიძებნება ყვავილი ან მცენარე 3, 5, 8, 13 ფურცლით, მაგრამ გამონაკლისია 4, 6, 7 ან 9 ფურცლიანი მცენარე).

დამტკიცებულია, რომ ნებისმიერი ნატურალური რიცხვი ***ცალსახად*** ჩაიწერება ისეთი ფიბონაჩის რიცხვების ჯამის სახით, რომ ამ რიცხვებს შორის არ შეგვხვდება მეზობელი ფიბონაჩის რიცხვები.
მაგალითად, $n = 67$ შემდეგნაირად წარმოდგება: $67 = 1 + 3 + 8 + 55$ და ეს წარმოდგენა ერთად-ერთია (მართალია, $67 = 1 + 3 + 8 + 21 + 34$, მაგრამ აქ 21 და 34 მეზობელი რიცხვებია ფიბონაჩის მიმდევრობაში, რაც პირობას ეწინააღმდეგება).
ასეთი ცალსახა ჯამი წარმოშობს ფიბონაჩის რიცხვების მიმდევრობას, ანუ კოდს, რომელიც ცალსახად განსაზღვრავს ამ რიცხვს და კოდირების თეორიასა და პრაქტიკაში გამოიყენება.

ფიბონაჩის მიმდევრობის გამოყენებით გადაჭრილი იქნა მეოცე საუკუნის დასაწყისში უდიდესი გერმანელი მათემატიკოსის დავით ჰილბერტის მიერ დასმული ერთ-ერთი უმნიშვნელოვანესი ამოცანა (ჰილბერტის X პრობლემა).

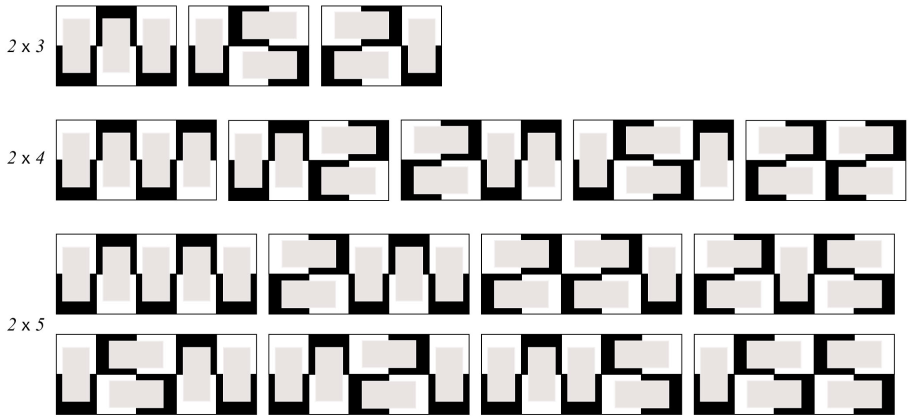
საინტერესოა ამ რიცხვების გამოყენება კომბინატორიკაში.
განვიხილოთ შემდეგი ამოცანა: მოცემულია n საფეხურიანი კიბე. თუ ჩვენ კიბის ავლა შეგვიძლია ისე, რომ თითო ნაბიჯში ერთ ან ორ საფეხურს ავდივართ, კიბის ავლის რამდენი სხვადასხვა ვარიანტი არსებობს ?
ცხადია, თუ $n = 1$, ჩვენ კიბის ავლის ერთად-ერთი საშუალება გვექნება. თუ $n = 2$, მაშინ გვექნება ორი შესაძლებლობა: ან თითო-თითო კიბის ავლის, ან ერთ ჯერზე ორის. $n = 3$ შემხვევაში გვექნება 3 შესაძლებლობა: 1+1+1 ან 1+2 ან 2+1. $n = 4$: 1+1+1+1 ან 1+1+2 ან 1+2+1 ან 2+1+1 ან 2+2, სულ 5 შესაძლებლობა.
თუ G_n აღნიშნავს n საფეხურიანი კიბის ზემოთ მოყვანილი პირობით ავლის ვარიანტების რაოდენობას, მაშინ შეიძლება G_{n+1} რიცხვის გამოთვლა შემდეგი ანალიზის საფუძველზე: თუ მოცემულია $n + 1$ საფეხურიანი კიბე, ჩვენ შეგვიძლია ჯერ ავიაროთ ერთი საფეხური და მერე n საფეხური G_n სხვადასხვა მეთოდით, ან ჯერ ავიაროთ 2 საფეხური და შემდეგ $n - 1$ საფეხური G_{n-1} სხვადასხვა მეთოდით. აქედან გამომდინარე, ვიღებთ ფორმულას

$$G_{n+1} = G_n + G_{n-1},$$

რაც ფიბონაჩის მიმდევრობის განმსაზღვრელი რეკურსიული ფორმულაა. განსხვავება მხოლოდ ისაა, რომ ამ მიმდევრობების პირველი და მეორე ელემენტი ფიბონაჩის მიმდევრობის მეორე და მესამე ელემენტების ტოლია. აქედან გამომდინარე ვიღებთ:

$$G_n = F_{n+1}.$$

მეორე ამოცანად შეიძლება ჭადრაკის დაფის დომინოს ქვებით გადაფარვის პრობლემა მოვიყვანოთ:
მოცემულია ჭადრაკის დაფის ფრაგმენტი ზომით $2 \times n$ და n ცალი დომინოს ქვა, რომელთა შორის თითო 2 კვადრატს ფარავს. რამდენი სხვადასხვა მეთოდით შეიძლება n ქვით $2 \times n$ ზომის ფრაგმენტის დაფარვა? ქვედა ნახატში მოყვანილია ამონახსნები 2×3, 2×4 და 2×5 ზომისათვის.



ნახ. 3.3: ჭადრაკის დაფის ფრაგმენტის გადაფარვა

სავარჯიშო 3.15: დავუშვათ, $2 \times n$ ფრაგმენტის გადაფარვა P_n ცალი სხვადასხვა მეთოდით შეიძლება (ზემოთ მოყვანილი ნახატიდან ჩანს, რომ $P_3 = 3$, $P_4 = 5$ და $P_5 = 8$). რა სახის რეკურსიული ფორმულით აღიწერება P_n? რა კავშირშია ეს მიმდევრობა ფიბონაჩის რიცხვებთან?

აქვე შეგვიძლია ჩამოვთვალოთ ფიბონაჩის მიმდევრობის რამოდენიმე თვისება:

- $F_1 + F_2 + \cdots + F_n = F_{n+2} - 1$;
- F_{3n} ლუწია;
- F_{5n} იყოფა 5-ზე;
- $F_1 + F_3 + F_5 + \cdots + F_{2n-1} = F_{2n}$;
- $F_0 - F_1 + F_2 - F_3 + \cdots - F_{2n-1} + F_{2n} = F_{2n-1} - 1$;
- $F_1^2 + F_2^2 + \cdots + F_n^2 = F_n \cdot F_{n+1}$;
- $F_{n-1} \cdot F_{n+1} - F_n^2 = (-1)^n$;

სავარჯიშო 3.16: მათემატიკურ ინდუქციაზე დაყრდნობით დაამტკიცეთ ზემოთ მოყვანილი ტოლობები.

უფრო რთულად დასამტკიცებელი ფაქტებია:

- თუ $n > 4$ და F_n მარტივია, მაშინ n მარტივია (შებრუნებული გამონათქვამი არ არის ჭეშმარიტი: $\exists p$ მარტივი რიცხვი ისეთი, რომ F_p არაა მარტივი);
- თუ $n, m \in \mathbb{N}$ და $gcd(m, n)$ ამ ორი რიცხვის უდიდეს საერთო გამყოფს აღნიშნავს, მაშინ $gcd(F_m, F_n) = F_{gcd(m,n)}$
- $F_{n+m} = F_{n-1}F_m + F_nF_{m+1}$;
- $F_{(k+1)n} = F_{n-1}F_{kn} + F_nF_{kn+1}$;
- $F_n = F_lF_{n-l+1} + F_{l-1}F_{n-1}$;
- $F_n = F_{(n+1)/2}^2 + F_{(n-1)/2}^2$, თუ n კენტია;
- $F_n = F_{n/2+1}^2 + F_{n/2-1}^2$, თუ n ლუწია;

ღია საკითხი: შეგვხვდება თუ არა ფიბონაჩის მიმდევრობაში უსასრულოდ ბევრი მარტივი რიცხვი?

სავარჯიშო 3.17: რისი ტოლია $gcd(46368, 21)$?

სავარჯიშო 3.18: დაწერეთ ალგორითმი, რომელიც მოცემული რიცხვისათვის გაარკვევს, არის თუ არა იგი ფიბონაჩის რიცხვი.

ბუნებრივად ისმის შემდეგი საკითხი: რამდენ ბიჯში შეიძლება გამოვითვალოთ F_n ზემოთ მოყვანილი რეკურსიული ფორმულის მეშვეობით?

სავარჯიშო 3.19: გამოითვალეთ $T(F_n)$ (ზემოთ მოყვანილი F_n რიცხვის გამოთვლისათვის საჭირო ბიჯების რაოდენობა).

ახლა კი განვიხილოთ ფიბონაჩის რიცხვების გამოთვლის ე.წ. იტერაციული ალგორითმი, რომელიც რამოდენიმეჯერ იმეორებს ერთსა და იმავე ოპერაციას:

ალგორითმი 3.1: ნავების გაყვანა (იტერაციული ვერსია)
მონაცემი: n (ნავების რაოდენობა)

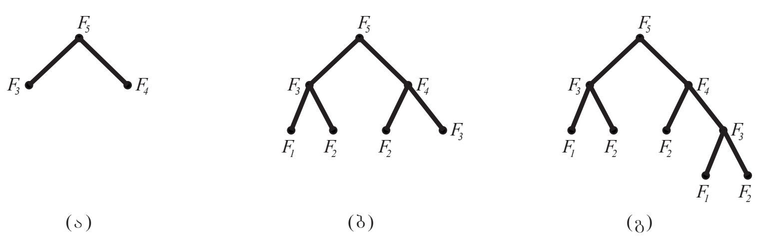
```
გაიმეორე n-ჯერ:
{
  c = a + b;
  b = a;
  a = c;
}
```
ალგორითმი დასრულებულია

სავარჯიშო 3.20: დაამტკიცეთ, რომ ამ ალგორითმის შესრულების შემდეგ a ცვლადში ფიბონაჩის მე-n-ე რიცხვი ეწერება. რა იქნება ჩაწერილი c და b ცვლადებში?

სავარჯიშო 3.21: დაამტკიცეთ, რომ ზემოთ აღწერილი იტერაციული ალგორითმით მე-n-ე ფიბონაჩის რიცხვის გამოთვლას $4n + 2$ ბიჯი დაჭირდება (მიმატებისა და მინიჭების ოპერაციები თითო ბიჯად ჩათვალეთ).

ამ ორი მაგალითიდან ნათლად ჩანს, რომ ფიბონაჩის რიცხვების გამოთვლა გაცილებით უფრო სწრაფია იტერაციული ალგორითმით, ვიდრე რეკურსიულით.

მაგრამ რამ გამოიწვია ასეთი განსხვავება გამოთვლის სისწრაფეში? ამ საკითხის გასაანალიზებლად განვიხილოთ რეკურსიული ალგორითმის გამოთვლის პროცესის გრაფიკული წარმოდგენა:



ნახ. 3.4: F_5 გამოთვლის პროცესი

თუ ყოველ წვეროს (წერტილს) წარმოვიდგენთ, როგორც ფიბონაჩის მიმდევრობის შესაბამის რიცხვს და ამ წვეროდან ხაზებით დაკავშირებულ ქვედა წვეროებს როგორც ამ რიცხვის გამოსაანგარიშებლად საჭირო სხვა რიცხვებს, ადვილად დავინახავთ, რომ F_5 რიცხვის გამოსაანგარიშებლად რეკურსიული ალგორითმი ორჯერ გამოიანგარიშებს F_3 რიცხვს, რაც თავის მხრივ კიდევ ორი რიცხვის ჯამისაგან შედგება. აქედან გამომდინარე, რამოდენიმე გამოთვლა „ზედმეტია": შესაძლებელი იქნებოდა, მაგალითად, F_3 რიცხვის ერთხელ გამოთვლა და შედეგის სადმე შენახვა მისი საჭიროების შემთხვევაში გამოყენების მიზნით, რაც ზედმეტ გამოთვლას თავიდან აგვაცილებდა. ამ მაგალითზე ნათლად ჩანს, თუ რა უარყოფითი მხარე აქვს რეკურსიის „ბრმად" გახსნას: ხშირად უკვე გამოთვლილი ნაწილები თავიდან გამოთვლება. სწორედ ეს იყო იმის მიზეზი, რომ 1950-1960ან წლებში რეკურსიული

ფუნქციები მთლიანად ამოიღეს (ან ფაქტიურად არ გამოიყენებოდა) დაპროგრამების ენებში. მაგრამ მოგვიანებით, კომპილატორების ოპტიმიზატორების განვითარების შედეგად, ეს პრობლემა დაძლეულ იქნა და ახლა უკვე რეკურსიული ფუნქციები ფართოდ გამოიყენება მათი ბევრი დადებითი მხარეების გამო.

რეკურსიულ სტრუქტურებში მონაცემთა ერთხელ გამოთვლისა და მათი შემდგომი გამოყენების მიზნით შენახვის იდეაზეა დაფუძნებული ე.წ. დინამიური დაპროგრამების პრინციპი, რომელსაც ჩვენ შემდგომ კურსებში დეტალურად განვიხილავთ.

<u>სავარჯიშო 3.22:</u> დახაზეთ F_6 და F_7 რიცხვების გამოთვლისთვის საჭირო დიაგრამები ისე, როგორც ეს ზემოთ მოყვანილ მაგალითში იყო. რამდენი „ზედმეტი" გამოთვლა ტარდება ამ შემთხვევებში?

ცხადია, რომ უფრო მოსახერხებელი იქნებოდა ფიბონაჩის რიცხვების არარეკურსიული ფორმულით გამოანგარიშება. მაშინ მის გამოთვლას არც თუ ისეთ დიდ დროს მოვანდომებდით. და მართლაც, ასეთი წარმოდგენა არსებობს:

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

<u>სავარჯიშო 3.23:</u> მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ ამ ფორმულის სისწორე.

სავსებით ლოგიკურია შემდეგი შეკითხვა: როგორ შეიძლება ამ ფორმულის გამოყვანა? რა გზით მიაგნო ვინმემ ასეთ რთულ ფორმულას?
პირველ რიგში საჭიროა მიმდევრობის რიცხვების დაკვირვება. ერთი შეხედვით, $F_n = F_{n-1} + F_{n-2}$ კანონზომიერების გარდა აქ არაფერი ჩანს:

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, \ldots$

მაგრამ თუ მეზობელ რიცხვებს ერთმანეთს შევუფარდებთ, შეიძლება დამატებითი კანონზომიერება დავინახოთ:

$T_n = \frac{F_n}{F_{n-1}} \quad (n > 1)$:

$T_2 = \frac{1}{1} = 1;$	$T_3 = \frac{2}{1} = 2;$	$T_4 = \frac{3}{2} = 1,5;$	$T_5 = \frac{5}{3} \approx 2,666667;$
$T_6 = \frac{8}{5} = 1,6;$	$T_7 = \frac{13}{8} = 1,625;$	$T_8 = \frac{21}{13} \approx 1,615;$	$T_9 = \frac{34}{21} \approx 1,619;$
$T_{10} = \frac{55}{34} \approx 1,6176;$	$T_{11} = \frac{89}{55} \approx 1,618;$	$T_{12} = \frac{144}{89} \approx 1,61798;$	$T_{13} = \frac{233}{144} \approx 1,61805;$
$T_{14} = \frac{377}{233} \approx 1,618026;$	$T_{15} = \frac{610}{377} \approx 1,618037;$	$T_{16} = \frac{987}{610} \approx 1,618033;$	$T_{17} = \frac{1597}{987} \approx 1,618034;$
$T_{18} = \frac{2584}{1597} \approx 1,618034;$	$T_{19} = \frac{4181}{2584} \approx 1,618034;$	$T_{20} = \frac{6765}{4181} \approx 1,6180339887;$	$T_{21} = \frac{10946}{6765} \approx 1,6180339887;$
$T_{22} = \frac{17711}{10946} \approx 1,6180339887;$	$T_{23} = \frac{28657}{17711} \approx 1,6180339887;$	$T_{24} = \frac{46368}{28657} \approx 1,6180339887...$	

ამ რიცხვებს რომ დავაკვირდეთ, შევამჩნევთ, რომ T_n, ანუ ფიბონაჩის მეზობელი რიცხვების ერთმანეთთან შეფარდება, ერთი რიცხვისკენ მიისწრაფვის, ანუ სულ უფრო და უფრო უახლოვდება ამ რიცხვს და მათ შორის განსხვავება დროთა განმავლობაში ნულს უახლოვდება). მართლაც, დამტკიცებულია, რომ

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \Phi \approx 1,6180339887.$$

აქ $\Phi = \frac{1+\sqrt{5}}{2}$ ე.წ. ოქროს კვეთაა და იტყვიან, რომ შეფარდების მიმდევრობის ზღვარი Φ ტოლია, თუ ინდექსი n მიისწრაფვის უსასრულობისკენ (ეს მათემატიკური ანალიზის საფუძვლების საკითხია და ჩვენ ამას დეტალებში არ შევეხებით, თუმცა ჩავთვლით, რომ მკითხველისთვის ცნობილია).

ჩვენს შემთხვევაში ეს იმას ნიშნავს, რომ დაწყებული რაღაცა ადგილიდან, ფიბონაჩის მიმდევრობა ***გეომეტრიული პროგრესიის*** თვისებებს ავლენს. აქედან გამომდინარე, შეგვიძლია ვივარაუდოთ, რომ არსებობს ისეთი მიმდევრობა

$$G_n = c \cdot q^n,$$

რომელიც ემთხვევა ფიბონაჩის მიმდევრობას (დაწყებული რაიმე ადგილიდან მაინც) რაღაცა $c, q \in \mathbb{N}$ რიცხვებისათვის. მაგრამ როგორ უნდა შევარჩიოთ c და q?

რა თქმა უნდა, თვით $(G_n)_{n=1}^{\infty}$ მიმდევრობაც უნდა აკმაყოფილებდეს ფიბონაჩის მიმდევრობის თვისებას:

$$G_n = G_{n-1} + G_{n-2}.$$

აქედან გამომდინარე,

$$c \cdot q^n = c \cdot q^{n-1} + c \cdot q^{n-2}$$

და, შესაბამისად, თუ ტოლობის ორივე მხარეს გავყოფთ $c \cdot q^{n-2}$ სიდიდეზე, მივიღებთ შემდეგ განტოლებას, რომლითაც q პარამეტრის დადგენას შევძლებთ:

$$q^2 = q + 1.$$

ამ კვადრატული განტოლების ამონახსნებია

$$q_1 = \frac{1+\sqrt{5}}{2} \quad \text{და} \quad q_2 = \frac{1-\sqrt{5}}{2}.$$

აქედან გამომდინარე, ვიღებთ ორ მიმდევრობას

$$G'_n = c \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n \quad \text{და} \quad G''_n = c \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n,$$

სადაც ორივე ფიბონაჩის პირობას აკმაყოფილებს.
დარჩა მხოლოდ ამ ორი მიმდევრობიდან ერთ-ერთისა ან მათი კომბინაციის ამორჩევა და c პარამეტრის დადგენა ისე, რომ მიღებული მიმდევრობა ფიბონაჩის $(F_n)_{n=1}^{\infty}$ მიმდევრობას დაემთხვას.

განვიხილოთ მიმდევრობა G'_n. თუ $n = 1$, ვიღებთ: $G'_1 = c \cdot \frac{1+\sqrt{5}}{2}$ და, აქედან გამომდინარე, რადგან ჩვენ გვინდა, რომ $G'_1 = F_1 = 1$, ვიღებთ:

$$c \cdot \frac{1+\sqrt{5}}{2} = 1.$$

ესე იგი, $c = \frac{2}{1+\sqrt{5}}$. მაგრამ ამ შემთხვევაში $G'_2 = \frac{2}{1+\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^2 \neq F_2 = 1$. ასე რომ, $(G'_n)_{n=1}^{\infty}$ მიმდევრობა ცალკე აღებული ფიბონაჩის მიმდევრობას ვერ დაემთხვევა.

<u>სავარჯიშო 3.24:</u> ანალოგიური მსჯელობით აჩვენეთ, რომ არც მიმდევრობა $(G''_n)_{n=1}^{\infty}$ დაემთხვევა ფიბონაჩის მიმდევრობას.

ახლა განვიხილოთ მიმდევრობა

$$H_n = G'_n - G''_n = c \cdot \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

<u>სავარჯიშო 3.25:</u> დაამტკიცეთ, რომ ზემოთ მოყვანილი მიმდევრობა $(H_n)_{n=1}^{\infty}$ ფიბონაჩის პირობას აკმაყოფილებს.

<u>სავარჯიშო 3.26:</u> განიხილეთ მიმდევრობა $H'_n = G'_n + G''_n$. აკმაყოფილებს თუ არა იგი ფიბონაჩის პირობას?

ცხადია, $H_0 = 0$ და ამით ამ მიმდევრობის ნულოვანი წევრი ფიბონაჩის მიმდევრობის ნულოვან წევრს ემთხვევა. ახლა კი განვიხილოთ H_1:

$$H_1 = c \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right) - \left(\frac{1-\sqrt{5}}{2} \right) \right).$$

რადგან ჩვენ გვინდა, რომ ეს წევრი ფიბონაჩის მიმდევრობის პირველ წევრს დაემთხვას, ვიღებთ განტოლებას:

$$H_1 = c \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right) - \left(\frac{1-\sqrt{5}}{2} \right) \right) = 1.$$

c ცვლადის მიმართ ამ განტოლების ამოხსნის შემდეგ ვიღებთ: $q = \frac{1}{\sqrt{5}}$. აქედან გამომდინარე,

$$H_1 = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right) - \left(\frac{1-\sqrt{5}}{2} \right) \right).$$

რადგან მიმდევრობა $(H_n)_{n=1}^{\infty}$ აკმაყოფილებს ფიბონაჩის პირობას და მისი პირველი ორი წევრი ფიბონაჩის მიმდევრობის პირველი ორი წევრის ტოლია, ეს ორი მიმდევრობა მთლიანად დაემთხვევა ერთმანეთს:

$$H_n = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

რ. დ. გ.

<u>სავარჯიშო 3.27:</u> განიხილეთ მიმდევრობა $H'_n = G'_n + G''_n = c' \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n + \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$.
შეიძლება თუ არა აქ c' ცვლადის ისე შერჩევა, რომ H' მიმდევრობა ფიბონაჩის მიმდევრობას დაემთხვას?

თუ ყურადღებით გავაანალიზებთ ფიბონაჩის რიცხვების ფორმულის გამოყვანის პროცესს, აღმოვაჩენთ რამოდენიმე ფუნდამენტურ მომენტს:

- თავდაპირველად ვატარებთ „ექსპერიმენტებს”: როგორც ფიზიკოსები აკვირდებიან ბუნებრივ მოვლენებს, ბიოლოგები - ცოცხალ ორგანიზმებს, ქიმიკოსები - ნივთიერებებს, ჩვენც ვაკვირდებით რიცხვთა მიმდევრობებს და მათ ურთიერთდამოკიდებულებას. ამ დაკვირვების პროცესს ***ინდუქცია*** ეწოდება (არ აგერიოთ ზემოთ მოყვანილ ***მათემატიკურ ინდუქციაში***). რიცხვებზე „ექსპერიმენტებში” სწორედ სხვადასხვა დამოკიდებულების - მეზობელთა სხვაობების, ჯამების, შეფარდებების, ნამრავლებისა და სხვა კომბინაციების დაკვირვება იგულისხმება;

- ერთ-ერთი ასეთი ექსპერიმენტის დაკვირვებისას აღმოვაჩინეთ გარკვეული კანონზომიერება: მეზობელ რიცხვთა შეფარდება რაღაც ერთი რიცხვისკენ იკრიბება;

- ამ კანონზომიერებამ გარკვეული ***ანალოგიის*** ^გატარების შესაძლებლობა მოგვცა: რადგან შეფარდება სულ უფრო და უფრო უახლოვდება რაღაც რიცხვს - ისე, როგორც გეომეტრიული პროგრესია - შესაძლებელია, რომ ამ იმდევრობას გეომეტრიული პროგრესიის მსგავსი სხვა თვისებებიც ქონდეს;

- აქედან გამომდინარე შეიქმნა კვლევის ახალი ობიექტი: ფიბონაჩის თვისების მქონე გეომეტრიული პროგრესია.

სწორედ ამ ობიექტის კვლევის შედეგად იქნა გამოყვანილი ეს ფორმულა.

ზოგადად, მეცნიერების დანიშნულებაც ასეთია: კვლევის ობიექტებზე დაკვირვება, სხვადასხვა ექსპერიმენტის ჩატარება, კანონზომიერებების აღმოჩენა და ამ კანონზომიერებების მიზეზების ძიება.
ჩვენი კვლევის ობიექტი ამ შემთხვევაში რიცხვთა მიმდევრობა იყო, კანონზომიერება კი - გეომეტრიულ პროგრესიასთან მსგავსება. ამის მიზეზი შეიძლება იყოს ის, რომ ჩვენი საწყისი მიმდევრობა (ამ შემთხვევაში ფიბონაჩის მიმდევრობა) გეომეტრიული პროგრესიის ***ანალოგია***.

3.4 პასკალის სამკუთხედი

ე.წ. პასკალის სამკუთხედში იგულისხმება სამკუთხა ცხრილი, რომლის ყოველი ელემენტი მის ზემოთ მდგარი ორი ელემენტის ჯამია:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

ასე, მაგალითად, მეოთხე სტრიქონის მეორე ელემენტი არის 3, რადგან მის ზემოთ მდგარი ორი ელემენტი (ანუ მესამე სტრიქონის პირველი და მეორე ელემენტი) არის 1 და 2.
ასეთი მეთოდით დალაგებულ რიცხვებს საკმაოდ ფართოდ გამოიყენებენ პრაქტიკაში. მართალია, იგი უკვე ძველ ინდოეთში, ძველ საბერძნეთსა და შუასაუკუნეების ირანშიც იყო ცნობილი, მისი თანამედროვე მეცნიერებაში გამოყენება ფრანგ მეცნიერს ბლეზ პასკალს ეკუთვნის და ჩვენში მის სახელს ატარებს (თუმცა აღმოსავლეთში ასევე ცნობილი ირანელი ფილოსოფოსისა და პოეტის ომარ ხაიამის სამკუთხედს უწოდებენ).

რაში შეიძლება გამოგვადგეს პასკალის სამკუთხედის რიცხვები? მათი ყველაზე ფართოდ გამოყენებად შეიძლება ორი რიცხვის ჯამის n ხარისხში აყვანა განვიხილოთ:

$$(x+y)^n = a_{n,0}x^n \cdot y^0 + a_{n,1}x^{n-1} \cdot y^1 + a_{n,2}x^{n-2} \cdot y^2 + \cdots + a_{n,n-1}x^1 \cdot y^{n-1} + a_{n,n}x^0 \cdot y^n$$

ყოველი ფიქსირებული n რიცხვისთვის ვიღებთ ე.წ. n რიგის ორ ცვლადიან მრავალწევრს ($n+1$ ელემენტიან ჯამს, სადაც ცვლადების ხარისხების ჯამი მუდმივია და უდრის n).
ასე, მაგალითად,

$(a+b)^0 = 1,$
$(a+b)^1 = a+b = 1a^1b^0 + 1a^0b^1,$
$(a+b)^2 = a^2+2ab+b^2 = 1a^2b^0 + 2a^1b^1 + 1a^0b^2,$
$(a+b)^3 = a^3+3a^2b+3ab^2+b3 = 1a^3b^0 + 3a^2b + 3ab^2 + 1a^0b^3$

თუ ზედა მაგალითებში დავაკვირდებით თითოეულ მრავალწევრს (ანუ პოლინომს), ადვილად დავინახავთ, რომ მათი კოეფიციენტები ზუსტად პასკალის სამკუთხედის სტრიქონებია. აღმოჩნდა, რომ ეს ყოველთვის ასეა: ორი რიცხვის ჯამის n ხარისხში აყვანის პოლინომი პასკალის სამკუთხედის $n+1$ სტრიქონის ელემენტებს ემთხვევა:

$$a_{i,j} = P_{i+1,j},$$

სადაც $P_{k,l}$ პასკალის სამკუთხედის k სტრიქონში l პოზიციაზე მდგარ ელემენტს აღნიშნავს.

უფრო მეტიც: პასკალის სამკუთხედის ყოველი $P_{i,j}$ ელემენტი ე.წ. ბინომიალურ კოეფიციენტს, ჯუფდებას აღნიშნავს:

$$P_{i,j} = \left(\begin{array}{c} i-1 \\ j-1 \end{array} \right)$$

თავის მხრივ ჯუფდება $C_p^k \equiv \left(\begin{array}{c} k \\ p \end{array} \right)$ k ელემენტიანი სიმრავლიდან p სხვადასხვა ელემენტის ამორჩევის შესაძლებლობათა რაოდენობას აღნიშნავს. მაგალითად, თუ $k=4$ და $p=2$, უნდა განვიხილოთ 4 ელემენტიანი სიმრავლე $\{a_1, a_2, a_3, a_4\}$ (ელემენტების სახელებს მნიშვნელობა არ აქვს) და ვნახოთ, მისი რამდენი სხვადასხვა 2 ელემენტიანი ქვესიმრავლე არსებობს (რამდენი სხვადასხვა ორი ელემენტის ამორჩევაა შესაძლებელი). ჩვენს შემთხვევაში გვექნება:

$\{a_1, a_2\}, \{a_1, a_3\}, \{a_1, a_4\}, \{a_2, a_3\}, \{a_2, a_4\}, \{a_3, a_4\}$, ანუ სულ 6 ცალი: $C_p^k \equiv \begin{pmatrix} k \\ p \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} = 6.$

ეს ყველაფერი ე.წ. კომბინატორიკის საკითხებია, რომლებსაც ჩვენ მოგვიანებით განვიხილავთ.

სავარჯიშო 3.28: დაწერეთ რეკურსიული ფორმულა, რომლითაც პასკალის სამკუთხედის $P_{n,m}$ ელემენტს გამოვითვლით.

შენიშვნა: არსებობს ჯუფდების გამოთვლის ფორმულა $C_p^k = \frac{k!}{p!(k-p)!}$, რომლის შინაარსსაც და გამოყვანის პროცესს ჩვენ შემდგომ თავში განვიხილავთ.

სავარჯიშო 3.29: წინა სავარჯიშოში გამოთვლილი რეკურსიული ფორმულის საფუძველზე დაწერეთ რეკურსიული ალგორითმი, რომლითაც პასკალის სამკუთხედის $P_{n,m}$ ელემენტს გამოვითვლით.

3.5 მოკლე დასკვნა

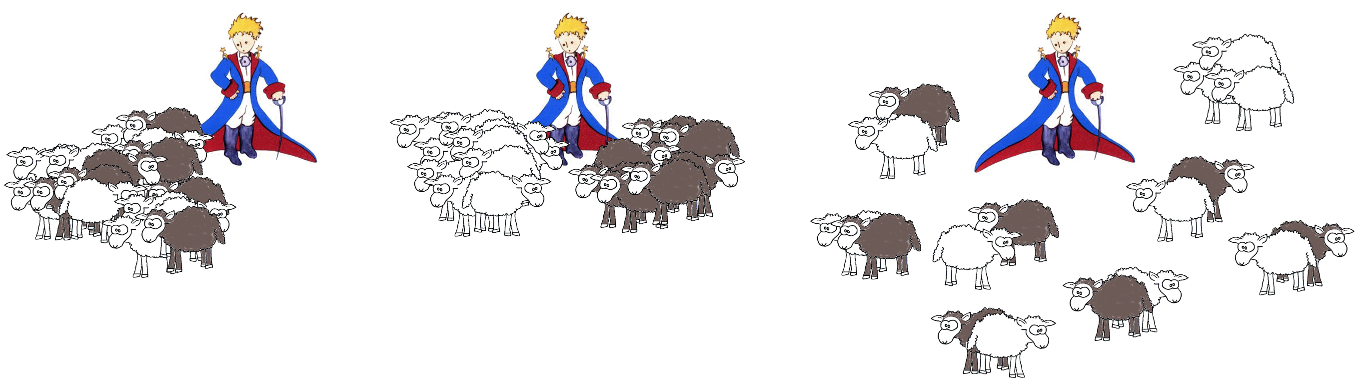
მესამე თავში ჩვენ განვიხილეთ ე.წ. მათემატიკური ინდუქციის პრინციპი, რომელიც ფართოდ გამოიყენება რეკურსიულად აგებული (ერთმანეთზე დამოკიდებული დალაგებული) სტრუქტურების კანონზომიერების მტკიცებაში და ეს პრინციპი გადავიტანეთ ალგორითმების სისწორის მტკიცებისა და ბიჯების დათვლის ტექნიკაზე.
აგრეთვე გავიცანით ორი უმნიშვნელოვანესი მიმდევრობა: ფიბონაჩის რიცხვებისა და პასკალის სამკუთხედის, რომელთა გარკვეული კანონზომიერებების მტკიცებაშიც ვაჩვენეთ მათემატიკური ინდუქციის გამოყენების მნიშვნელოვანი მაგალითები.
ფიბონაჩის მიმდევრობის გამოთვლის რეკურსიული და იტერაციული ალგორითმის ანალიზის საფუძველზე გამოჩნდა ამ ორი მეთოდის პრინციპული განსხვავება და რეკურსიული ალგორითმის სუსტი მხარეები და ამ სისუსტის ძირითადი მიზეზი.

თავი 4

სიმრავლეები და მათი სიმძლავრე

4.1 ბიექციური ასახვა და თვლადი სიმრავლლები

განვიხილოთ შემდეგი მაგალითი: პატარა პრინცს აქვს თეთრ და შავ ცხვრიანი ფარა. ცხვრის ეს ფარა შეიძლება *სასრულ ელემენტიან სიმრავლედ* განვიხილოთ (ნახ. 4.1 მარცხნივ). მას პირველ რიგში იმის გარკვევა უნდა, შავი ცხვარი მეტია ფარაში, თუ თეთრი. ამისათვის იგი ცალკე აყენებს თეთრ და შავ ცხვრებს, რითაც ფარას, ანუ *სიმრავლეს* ორ ნაწილად, ანუ *ორ ქვესიმრავლედ* ყოფს (ნახ. 4.1 შუაში).



ნახ. 4.1: პატარა პრინცი ცხვრებით

თეთრი და შავი ცხვრის ფარა ცალკეულ სიმრავლეებადაც შეიძლება განვიხილოთ. იმის დასადგენად, თუ რომელ სიმრავლეშია მეტი ელემენტი (ანუ თეთრ თუ შავ ფარაშია მეტი ცხვარი), პატარა პრინცი ამ სიმრავლეების თითოეულ ელემენტს ერთმანეთთან აწყვილებს (ნახ. 4.1 მარჯვნივ).
სხვა სიტყვებით რომ ვთქვათ, ერთი სიმრავლის თითოეულ ელემენტს (შავ ცხვარს) ერთ ელემენტს შეუსაბამებს მეორე სიმრავლიდან (თეთრ ცხვარს) ისე, რომ ორ სხვადასხვა ელემენტს ორი სხვადასხვა ელემენტი შეესაბამებოდეს (ორ სხვადასხვა თეთრ ცხვარს ორი სხვადასხვა შავი ცხვარი შეესაბამება).
რადგან ასეთი დაწყვილების შემდეგ დარჩა ზედმეტი თეთრი ცხვარი, ვასკვნით, რომ თეთრი ცხვარი მეტია.

ცხადია, რომ ასეთი სახის დაწყვილება ნებისმიერ ორ სასრულ სიმრავლეს შორის შეიძლება. მთავარია, რომ პირველი სიმრავლის ორი *სხვადასხვა* ელემენტი მეორე სიმრავლის ორ *სხვადასხვა* ელემენტთან დავაწყვილოთ. თუ ამდაგვარი დაწყვილების შემდეგ არც ერთ სიმრავლეში ზედმეტი (დაუწყვილებელი) ელემენტი არ დაგვრჩება, შეიძლება დავასკვნათ, რომ ამ სიმრავლეებში ელემენტების რაოდენობა ტოლია.

ასეთ დაწყვილებას ***ბიექცია*** ეწოდება.

ბუნებრივია შემდეგი შეკითხვა: შეიძლება თუ არა ამდაგვარი დაწყვილება უსასრულო სიმრავლეებში? თუ განვიხილავთ ნატურალურ რიცხვთა სიმრავლეს $\mathbb{N} = \{1, 2, 3, 4, ...\}$ და ლუწ რიცხვთა სიმრავლეს $2\mathbb{N} = \{2, 4, 6, 8, ...\}$, ასეთი დაწყვილება შეიძლება იყოს $(1 \leftrightarrow 2)$, $(2 \leftrightarrow 4)$, $(3 \leftrightarrow 6)$, $(4 \leftrightarrow 8)$, ... , $(n \leftrightarrow 2n)$, ...

იმის და მიუხედავად, რომ ეს პროცესი უსასრულოდ გაგრძელდება, დაბეჯითებით შეგვიძლია იმის თქმა, რომ $\mathbb{N}$ სიმრავლის ყველა ელემენტს $2\mathbb{N}$ სიმრავლის ზუსტად ერთი ელემენტი შეესაბამება და „ზედმეტი" ელემენტი არც ერთ სიმრავლეში არ დარჩება (ზუსტად იგივე მსჯელობით ვასკვნით, რომ $2\mathbb{N}$ სიმრავლის ყველა ელემენტს *ცალსახად* შეესაბამება $\mathbb{N}$ სიმრავლის ერთი ელემენტი).
როგორც ვთქვით, ასეთ დაწყვილებებს *ბიექცია* ანუ *ურთიერთცალსახა ასახვა* ეწოდება.

მათემატიკის ენაზე ეს შემდეგნაირად შეიძლება გამოისახოს:
განმარტება 4.1: განვიხილოთ ასახვა $f : A \to B$, სადაც A და B რაიმე სიმრავლეებია, ანუ f ასახვით A სიმრავლის ***ყოველ*** ელემენტს B სიმრავლის რაიმე ელემენტი შეესაბამება. თუ განვიხილავთ A სიმრავლის რაიმე ელემენტს $a \in A$, რომელიც f ასახვით აისახება B სიმრავლის ელემენტში $b \in B$, ანუ ფორმალური ჩანაწერით $f : a \mapsto b$, მაშინ b ელემენტს a ელემენტის *ანასახი*, ხოლო a ელემენტს კი b ელემენტის *წინარე სახე* ეწოდება.

- $f : A \to B$ ასახვას ეწოდება *სურექციული (ანუ სურექცია)*, თუ B სიმრავლის ყოველი ელემენტისთვის მოიძებნება A სიმრავლის რაღაც ელემენტი a ისეთი, რომ f ასახვით a ელემენტი b ელემენტში აისახება. ამ წინადადების ფორმალური ჩანაწერია $\forall b \in B, \exists a \in A, f(a) = b$
(სხვა სიტყვებით რომ ვთქვათ, არ არსებობს b სიმრავლეში ისეთი ელემენტი, რომელსაც a სიმრავლეში წინარე სახე არ აქვს - ანუ არც ერთი ელემენტი არ „გამოგვრჩენია");

- $f : A \to B$ ასახვას ეწოდება *ინიექციური (ანუ ინიექცია)*, თუ ყოველი ორი სხვადასხვა ელემენტი ორ სხვადასხვაში აისახება. ფორმალურად: $\forall a \neq b, f(a) \neq f(b)$;

- $f : A \to B$ ასახვას ეწოდება *ბიექციური (ანუ ბიექცია)*, თუ იგი ერთროულად ინექციურიც და სურექციულიცაა.

ყოველივე ზემოთ თქმულიდან გამომდინარეობს, რომ ორ (სასრულ ან უსასრულო) სიმრავლეში ერთი და იგივე რაოდენობის ელემენტებია, თუ მათ შორის არსებობს ბიექცია (ურთიერთცალსახა შესაბამისობის დადგენაა შესაძლებელი).
აქვე უნდა აღინიშნოს, რომ უსასრულო სიმრავლეებში ელემენტების ტოლ ან მეტ რაოდენობაზე ლაპარაკი არაკორექტულია. ამიტომაც ამბობენ, რომ ორი სიმრავლის ***სიმძლავრე*** ტოლია, თუ მათ შორის არსებობს ბიექცია. თუ ორ A და B სიმრავლეს შორის ბიექცია არ არსებობს, მაგრამ არსებობს ბიექცია მთელს A სიმრავლესა და B სიმრავლის რაიმე ქვესიმრავლეს (ანუ ნაწილს) შორის, მაშინ ამბობენ, რომ A სიმრავლის სიმძლავრე B სიმრავლის სიმძლავრეზე ნაკლებია, ან (რაც იგივეა) B სიმრავლის სიმძლავრე A სიმრავლის სიმძლავრეზე მეტია.

საინტერესოა ის ფაქტი, რომ ზემოთ ჩვენ რაღაცა სიმრავლისა და მის ქვესიმრავლეს (ნაწილს) შორის შევძელით ბიექციის დამყარება (ელემენტების სრული დაწყვილება), ანუ დავადგინეთ, რომ რაღაც სიმრავლეებში ზუსტად იმდენი ელემენტია, როგორც მის რაღაცა ნაწილში. ცხადია, რომ ასეთი რამ სასრულ სიმრავლეებში შეუძლებელია.

სწორედ ესაა *უსასრულო სიმრავლის* განმარტება:

განმარტება 4.2: უსასრულო ეწოდება ისეთ სიმრავლეს, რომელსაც თავისი ბიექტიური ქვესიმრავლე მოეძებნება. აქედან გამომდინარე, შეგვიძლია აგრეთვე *უსასრულობის* განსაზღვრაც: უსასრულობა უსასრულო სიმრავლეში შემავალ ელემენტთა რაოდენობაა.

სიმრავლის ელემენტთა რაოდენობას მის ***კარდინალურ რიცხვსაც*** უწოდებენ.

თუ A სიმრავლე სასრულია, იტყვიან, რომ იგი სასრული ***სიმძლავრისაა*** და მისი ელემენტების რაოდენობა აღინიშნება როგორც $|A|$. თუ სიმრავლე უსასრულოა, შეიძლება დაიწეროს: $|A| = \infty$.

თუ შეიძლება ნატურალურ რიცხვთა სიმრავლესა და რაიმე A სიმრავლეს შორის ბიექციის დამყარება, მაშინ იტყვიან, რომ A სიმრავლე ***თვლადია*** (ან თვლადი სიმძლავრისაა, მისი ელემენტების გადათვლაა შესაძლებელი).
ცხადია, რომ თუ ორ სიმრავლეს შორის ბიექციის დამყარება შეიძლება, ასეთი ბიექცია (დაწყვილება) მრავალნაირად უნდა იყოს შესაძლებელი.
მაგალითად, ნატურალურ და ლუწ რიცხვთა შორის შესაძლებელია შემდეგი დაწყვილებაც: $(1 \leftrightarrow 4)$, $(2 \leftrightarrow 6)$, $(3 \leftrightarrow 2)$, $(4 \leftrightarrow 8)$, $(5 \leftrightarrow 10)$, $(6 \leftrightarrow 12)$, ... , $(n \leftrightarrow 2n)$, ...

ზოგადად, თუ მოცემულია სასრული სიმრავლეები A და B ისეთი, რომ $|A| = |B| = n$, მაშინ მათ შორის არსებობს $n! = 1 \cdot 2 \cdot 3 \cdots n$ სხვადასხვა ბიექცია (დაწყვილება). აქედან გამომდინარე, უსასრულო სიმრავლეებს შორის ან ნული, ან უსასრულოდ ბევრი ბიექცია უნდა არსებობდეს.

სავარჯიშო 4.1: აჩვენეთ, რომ არსებობს ბიექცია ნატურალურ რიცხვთა და $\mathbb{Z} = \{..., -n, ..., -3, -2, -1, 0, 1, 2, 3, ..., n, ...\}$ მთელ რიცხვთა სიმრავლეებს შორის.

სავარჯიშო 4.2: აჩვენეთ, რომ კენტ რიცხვთა სიმრავლე თვლადია.

ახლა კი განვიხილოთ რაციონალურ რიცხვთა სიმრავლე $\mathbb{Q} = \{\frac{a}{b}|\frac{a}{b}$ წილადის შეკვეცა შეუძლებელია$\}$. ეს სიმრავლე *ყველგან მკვრივია*, ანუ ნებისმიერ ორ რიცხვს შორის უსასრულოდ ბევრი რაციონალური რიცხვია: $]0;1[$ მონაკვეთში გვხვდება $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, ..., \frac{1}{n}, ...$ მიმდევრობა და, ზოგადად, ნებისმიერ $]q_1;q_2[$ მონაკვეთში გვხვდება უსასრულო მიმდევრობა $q_1 + \frac{q_2-q_1}{2}, q_1 + \frac{q_2-q_1}{3}, q_1 + \frac{q_2-q_1}{4}, ..., q_1 + \frac{q_2-q_1}{n}, ...$ (აქ $q_1, q_2 \in \mathbb{Q}$ რაციონალური რიცხვებია). აქედან გამომდინარე შეიძლება ვიფიქროთ, რომ $\mathbb{Q}$ სიმრავლეში „მეტი" ელემენტია, ვიდრე $\mathbb{N}$-ში, რადგან უკვე ნებისმიერ ორ ნატურალურ რიცხვს შორის უსასრულოდ ბევრი რაციონალური რიცხვია, თუმცა ჭეშმარიტია შემდეგი თეორემა:

თეორემა 4.1: დადებით რაციონალურ რიცხვთა სიმრავლე $\mathbb{Q}^+$ თვლადია.

დამტკიცება: ამ ფაქტის ჩვენება რაციონალურ რიცხვთა გადათვლის მეთოდის მოყვანით შეიძლება (მტკიცების ასეთ მეთოდს ***კონსტრუქციული*** ეწოდება: დასამტკიცებლად ჩვენ თვითონ მეთოდს ავაგებთ და ვნახავთ, რომ ამითი სასურველი შედეგის მიღებაა შესაძლებელი).

თავდაპირველად ჩამოვწეროთ *ყველა* რაციონალური რიცხვი:

$$
\begin{array}{|cccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} & \cdots \\
\frac{2}{1} & \frac{2}{3} & \frac{2}{4} & \cdots & \frac{2}{m} & \cdots \\
\frac{3}{1} & \frac{3}{2} & \frac{3}{4} & \cdots & \frac{3}{k} & \cdots \\
\vdots & & & & & \\
\frac{n}{1} & \frac{n}{2} & \frac{n}{4} & \cdots & \frac{n}{p} & \cdots \\
\vdots & & & & &
\end{array}
\qquad
\begin{array}{|cccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} & \cdots \\
\frac{2}{1} & \frac{2}{3} & \frac{2}{4} & \cdots & \frac{2}{m} & \cdots \\
\frac{3}{1} & \frac{3}{2} & \frac{3}{4} & \cdots & \frac{3}{k} & \cdots \\
\vdots & & & & & \\
\frac{n}{1} & \frac{n}{2} & \frac{n}{4} & \cdots & \frac{n}{p} & \cdots \\
\vdots & & & & &
\end{array}
$$

პირველ სტრიქონში ჩამოვწეროთ *ყველა* რაციონალური რიცხვი, რომლის მრიცხველია 1. მეორე სტრიქონში ისეთი, რომლის მრიცხველია 2 (მნიშვნელში ლუწ რიცხვებიანს არ განვიხილავთ, რადგან წილადი უკვეცი უნდა იყოს), მესამე სტრიქონში მნიშვნელით 3 და ა. შ..

ცხადია, რომ ამ *უსასრულო* ცხრილში *ყველა* რაციონალური რიცხვი შეგვხვდება (სხვა საკითხია, რომ ასეთი ცხრილი პრაქტიკაში შეუძლებელია - ეს *მხოლოდ* თეორიული სტრუქტურაა). ამის შემდეგ რაციონალურ რიცხვთა გადათვლა უკვე ადვილია: პირველ ნომრად ავიღებთ მარცხენა ზედა ელემენტს, შემდეგ - მეორე ნომრად - მეორე სტრიქონის პირველ ელემენტს და ავალთ დიაგონალზე ზემოთ (ესე იგი, მესამე ნომრად გვექნება პირველი სტრიქონის მეორე ელემენტი). რადგან ამის შემდეგ დიაგონალზე ზემოთ ასვლა აღარ შეიძლება, ჩამოვდივართ ქვედა (მესამე) სტრიქონში და იგივე პროცედურას ვაგრძელებთ: ავდივართ დიაგინალზე ზემოთ მანამ, სანამ საზღვარი არ შეგვხვდება (რის შედეგადაც გადავდივართ ახალ სტრიქონში), გზაში კი შემხვედრ რიცხვს გადავნომრავთ. ამ პროცესს უსასრულოდ ვაგრძელებთ.
ცხადია, რომ ამ ალგორითმით ყველა ნატურალურ რიცხვს ერთ რაციონალურ რიცხვს შევუსაბამებთ და ყველა რაციონალურ რიცხვს - ერთ ნატურალურს (რადგან ამ მეთოდით ცხრილში არსებულ ყველა რიცხვს ადრე თუ გვიან მივადგებით).

აქედან გამომდინარე, ნატურალურ და დადებით რაციონალურ რიცხვებს ცალსახად დავაწყვილებთ.

რ.დ.გ.

სავარჯიშო 4.3: ზემოთ დამტკიცებული თეორემის საფუძველზე აჩვენეთ, რომ რაციონალურ რიცხვთა სიმრავლე თვლადია.

აქამდე განხილული ყველა უსასრულო სიმრავლის სიმძლავრე თვლადი აღმოჩნდა. ბუნებრივია შემდეგი შეკითხვა: არის თუ არა ნებისმიერი უსასრულო სიმრავლე თვლადი?

პასუხი ერთობ მოულოდნელია: არსებობს არათვლადი სიმრავლე, ანუ ორი უსასრულო სიმრავლე შეიძლება სხვადასხვა სიმძლავრის იყოს (ანუ, უხეშად რომ ვთქვათ, სხვადასხვარაოდენობის ელემენტს შეიცავდეს)!

4.2 თეორემათა მტკიცების დიაგონალიზაციის მეთოდი: ყველა უსასრულო სიმრავლე ტოლი არ არის!

თეორემა 4.2: ნამდვილ რიცხვთა სიმრავლე $\mathbb{R}$ არ არის თვლადი.

ამ თეორემის დამტკიცება ადვილად გამომდინარეობს შემდეგი ლემიდან:

ლემა 4.1: $]0;1[$ სიმრავლე არ არის თვლადი.

დამტკიცება: აქ ჩვენ ორ უმნიშვნელოვანეს მეთოდს გამოვიყენებთ: წინააღმდეგობის დაშვებასა და დიაგონალიზაციას.
სანამ თვით დამტკიცებაზე გადავიდოდით, უნდა გავიხსენოთ ის ფაქტი, რომ ნებისმიერი ირაციონალური რიცხვი, რომელიც *არაა რაციონალური*, უსასრულო ათწილადის სახით წარმოდგება, მაგ. 327,123456798756453456..., ან 0,121284945767... და ა.შ.

განვიხილოთ $]0;1[$ მონაკვეთზე არსებული ირაციონალური რიცხვები და *დავუშვათ*, რომ ამ რიცხვთა სიმრავლე თვლადია, ანუ შეიძლება მათი ჩამოწერა ისე, რომ უსასრულო ცხრილში არც ერთი რიცხვი არ გამოგვრჩეს (ესაა სწორედ საწინააღმდეგოს დაშვებით მტკიცების პირველი ნაბიჯი: რადგან ვამტკიცებთ, რომ $]0;1[$ სიმრავლე არ არის თვლადი, ჯერ დავუშვათ მისი საწინააღმდეგო: რომ $]0;1[$ სიმრავლე არის თვლადი და შემდეგ - ლოგიკური მსჯელობის ჯაჭვის შედეგად - მივიღებთ წინააღმდეგობას, ანუ ისეთ დასკვნას, რომელიც აქამდე ცნობილ ჭეშმარიტებას ეწინააღმდეგება).
ესე იგი, ნებისმიერი რიცხვი ამ სიაში შეიძლება ჩაიწეროს, როგორც $0, d_{i,1}d_{i,2}d_{i,3}...d_{i,n}...$:

ზოგადი წარმოდგენა	მაგალითი
$D_1 = 0, d_{1,1}d_{1,2}d_{1,3}...d_{1,n}...$	$D_1 = 0,1298736178...$
$D_2 = 0, d_{2,1}d_{2,2}d_{2,3}...d_{2,n}...$	$D_2 = 0,8913467255...$
$D_3 = 0, d_{3,1}d_{3,2}d_{3,3}...d_{3,n}...$	$D_3 = 0,9871367513...$
$\vdots$	
$D_n = 0, d_{n,1}d_{n,2}d_{n,3}...d_{n,n}...$	$D_n = 0,8734368646...$
$\vdots$	

თუ შევძლებთ რაღაც C რიცხვის შედგენას, რომელიც ამ სიის არც ერთ ელემენტს არ დაემთხვევა, მივიღებთ წინააღმდეგობას, რადგან ეს სია უნდა შეიცავდეს $]0;1[$ მონაკვეთის ***ყველა*** რიცხვს (არც ერთი არ უნდა იყოს გამორჩენილი). მართლაც, თუ ავიღებთ $C = 0,[d_{1,1}+1][d_{2,2}+1][d_{3,3}+1]...[d_{n,n}+1]...$, ეს რიცხვი შედგება ისეთი ციფრებისგან, რომლებიც მივიღეთ ზედა ცხრილში ჩაწერილი რიცხვების მიერ შედგენილი ცხრილის დიაგონალზე მდგარ ციფრებს დამატებული 1 (თან $9+1 \equiv 0$). სხვა სიტყვებით რომ ვთქვათ, პირველი რიცხვის მძიმის შემდეგ პირველ ციფრს მიმატებული 1, მეორე რიცხვის მძიმის შემდეგ მეორე ციფრს მიმატებული 1, მესამე რიცხვის მძიმის შემდეგ მესამე ციფრს მიმატებული 1 და ა. შ.. ზედა მაგალითისათვის გვექნება $C = 0,208...7...$

რა თქმა უნდა, $C \neq D_1$, რადგან ეს რიცხვები მძიმის შემდეგ პირველ ციფრში განსხვავდებიან ერთმანეთისგან. ასევე $C \neq D_2$, რადგან ეს რიცხვები მძიმის შემდეგ მეორე ციფრში განხვავდებიან და ა.შ.: ანალოგიური მსჯელობით $C \neq D_i$ ნებისმიერი რიცხვისათვის, რომელიც ზედა ცხრილში მოვიყვანეთ.

ასე რომ, ჩვენს მიერ შედგენილი უსასრულო ათწილადი C არ ემთხვევა არც ერთ რიცხვს ზედა ცხრილიდან. სხვა სიტყვებით რომ ვთქვათ, ეს რიცხვი ცხრილში არაა, რაც პირველად დაშვებას ეწინააღმდეგება, რომ ჩვენ ყველა ირაციონალური რიცხვი ჩამოვწერეთ (გადავთვალეთ). რადგან ჩვენს მსჯელობაში არსად შეცდომა არ ყოფილა, წინააღმდეგობა გამოიწვია დაშვებამ, რომ ყველა ირაციონალური რიცხვის გადათვლა შეიძლება. ასე რომ, ნამდვილ რიცხვთა სიმრავლე არაა თვლადი: ***არ არსებობს*** ბიექცია ნამდვილ რიცხვებსა და ნატურალურ რიცხვებს შორის.

 რ.დ.გ.

ჩვენი მტკიცება დავიწყეთ წინააღმდეგობის დაშვებით: დავუშვით ის ფაქტი, რაც დასამტკიცებელი გამონათქვამის უარყოფაა - ამ შემთხვევაში ის, რომ $]0;1[$ მონაკვეთში ირაციონალურ რიცხვთა სიმრავლე თვლადია და, აქედან გამომდინარე, მათი ჩამოწერა შეიძლება.
ამ დაშვებიდან ლოგიკური მსჯელობით მივიღეთ წინააღმდეგობა: ყველა რიცხვი არ ყოფილა ჩამოწერილი, რაც იმას უნდა ნიშნავდეს, რომ დაშვება იყო არასწორი (დანარჩენ მსჯელობაში შეცდომა არ უნდა იყოს გაპარული, რაც ადვილი გადასამოწმებელია).

ასეთ მტკიცებას ***საწინააღმდეგოს დაშვების მეთოდი*** ეწოდება.

ამას გარდა, მსჯელობაში გამოვიყენეთ ე.წ. ***დიაგონალიზაციის მეთოდი:*** ჩამოვწერეთ შესაძლო ამონახსნის *ყველა* ვარიანტი და შექმნილი ცხრილის დიაგონალური ელემენტების ამორჩევითა და დამუშავებით მივიღეთ წინააღმდეგობა.

ზედა თეორემა ამტკიცებს უმნიშვნელოვანეს ფაქტს: ყველა უსასრულო სიმრავლე გადათვლადი არაა, ერთ უსასრულო სიმრავლეში შეიძლება „მეტი" ელემენტი იყოს, ვიდრე მეორე, ასევე უსასრულო სიმრავლეში.

ამაზე დაყრდნობით ძალიან მნიშვნელოვანი დასკვნის გამოტანაა შესაძლებელი, რაც შემდგომში მოხდება.

სავარჯიშო 4.4: დაამტკიცეთ, რომ დიაგონალიზაციის მეთოდზე დაყრდნობით (წინა თეორემის დამტკიცების ანალოგიურად) რაციონალურ რიცხვთა არათვლადობის დამტკიცება არ შეიძლება.

სავარჯიშო 4.5: ზემოთ მოყვანილ მტკიცებაში სად გამოვიყენეთ, რომ ირაციონალურ რიცხვთა წარმოდგენა უსასრულოა?

ბუნებრივად ჩნდება შემდეგი შეკითხვა: თუ ირაციონალურ რიცხვთა წარმოდგენა უსასრულოა, როგორ შეიძლება მათი გამოთვლა ალგორითმულად? პასუხი ისაა, რომ ირაციონალურ რიცხვთა ალგორითმულად გამოთვლა *შეუძლებელია*. სამაგიეროდ შეიძლება მათი ნებისმიერი სიზუსტით გამოთვლა: შეიძლება დაიწეროს ალგორითმი, რომელიც მძიმის შემდეგ ნებისმიერად ბევრ რიცხვს სწორად გამოითვლის.

მაგალითისათვის განვიხილოთ $\sqrt{2} = 1,41421356237310...$ ირაციონალური რიცხვის გამოთვლის ალგორითმი. მისი გამოთვლა შეიძლება შემდეგი რეკურსიული ფორმულით:

1. $a_0 = n > 0$;
2. $a_{n+1} = \frac{a_n + \frac{2}{a_n}}{2} = \frac{a_n}{2} + \frac{1}{a_n}$.

პირველ რიგში უნდა ითქვას, რომ რეკურსიის საწყის პარამეტრად ნებისმიერი დადებითი რიცხვი შეგვიძლია ავიღოთ. ეს შედეგზე გავლენას არ ახდენს, თუმცა მეტი ცდა დაგვჭირდება სასურველი სიზუსტის გამოსათვლელად.

თუ ავიღებთ $a_0 = 1$, მივიღებთ:

a_1=0,5+1=1,5;

a_2≈**1,41**667..;

a_3≈**1,4142**15...;

a_4≈**1,41421356237**46...

სავარჯიშო 4.6: გამოიანგარიშეთ $a_0 = 3$ საწყისი მნიშვნელობით მიღებული რიცხვები. რამდენ ბიჯში მივიღებთ მძიმის შემდეგ მერვე პოზიციამდე ყველგან სწორ ციფრს?

სავარჯიშო 4.7: დაამტკიცეთ, რომ $\sqrt{2}$ არაა რაციონალური. **მინიშნება:** დაუშვით საწინააღმდეგო. ვთქვათ, არსებობს ორი ურთიერთმარტივი რიცხვი a, b (ისეთი, რომ წილადი $\frac{a}{b}$ არ შეიკვეცება), $\frac{a}{b} = \sqrt{2}$ და მიიღეთ წინააღმდეგობა.

სავარჯიშო 4.8: რამდენი ბიჯი დაჭირდება ზემოთ მოყვანილ ალგორითმს $a_0 = 1$ საწყისი მონაცემით იმისათვის, რომ მძიმის შემდეგ 15, 17, 20, 23, 25, 27, 30 ციფრი გამოიანგარიშოს სწორად? ზოგადად, დაახლოებით რამდენი ბიჯი დაჭირდება მას იგივე საწყისი მონაცემით მძიმის შემდეგ $k \in \mathbb{N}$ ციფრის გამოსათვლელად?

თუ განვიხილავთ რაიმე სასრულ სიმრავლეს A, შესაძლებელია მისი ყველა ქვესიმრავლისაგან შემდგარი სიმრავლის აგება. ასე, მაგალითად, თუ $A = \{a_1, a_2, a_3\}$, მისი ყველა ქვესიმრავლისაგან შემდგარი სიმრავლე იქნება $\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$. ზოგადად, n ელემენტიანი სიმრავლის ქვესიმრავლეთა რაოდენობაა 2^n. ესეც კომბინატორიკის საკითხებია, რომლებსაც ჩვენ შემდგომში განვიხილავთ.
ცხადია, რომ სასრულ სიმრავლესა და მისი ყველა ქვესიმრავლისაგან შემდგარ სიმრავლეს შორის ბიექცია ვერ იარსებებს. როგორც დიაგონალიზაციის მეთოდით მტკიცდება, არც უსასრულო სიმრავლესა და მისი ქვესიმრავლეების სიმრავლეს შორის არსებობს ბიექცია, რაც იმას ნიშნავს, რომ ნებისმიერ A სიმრავლეზე უფრო მძლავრი სიმრავლეა მისი ყველა ქვესიმრავლის სიმრავლე, რომელიც აღინიშნება როგორც 2^A: $|A| < |2^A|$.

სავარჯიშო 4.9: დიაგონალიზაციის გამოყენებით დაამტკიცეთ, რომ ნებისმიერი უსასრულო A სიმრავლისთვის ჭეშმარიტია უტოლობა $|A| < |2^A|$.

4.3 მოკლე დასკვნა

მეოთხე თავში ჩვენ გავეცანით ე.წ. ბიექციურ ასახვებს და მათზე დაყრდნობით სიმრავლეთა სიმძლავრეები განვმარტეთ, რითაც გამოჩნდა, რომ ყველა უსასრულო სიმრავლე არაა ერთი სიმძლავრის, ანუ, უხეშად რომ ვთქვათ, ორი სიმრავლის ელემენტების დაწყვილების შემდეგ ერთ-ერთ სიმრავლეში შეიძლება დაგვრჩეს „ზედმეტი" ელემენტები - ზოგი უსასრულო სიმრავლე სხვა უსასრულო სიმრავლეზე „მეტ" ელემენტს შეიცავს: არსებობს სასრული, თვლადი და არათვლადი სიმძლავრის სიმრავლეები. უფრო მეტიც: ყოველ მოცემულ სიმრავლეზე მძლავრი სიმრავლის აგებაა შესაძლებელი.
ეს ერთობ უცნაური ფაქტი შემდგომში იმის დასამტკიცებლად გამოგვადგება, რომ ყველა ამოცანა ამოხსნადი არაა. უფრო მეტიც: გაცილებით „მეტი" არაამოხსნადი ამოცანა არსებობს, ვიდრე ამოხსნადი.

თავი 5

მონაცემთა კოდირება, ანბანი, ენა და გრამატიკა

5.1 მონაცემთა კოდირება

განვიხილოთ ქართული სიტყვები „ანბანი” და „ენა”. ეს ქართული ენის სიტყვებია, რომელთაც ენაში რაღაცა მნიშვნელობა (სემანტიკა) აქვს. სხვა საქმეა „გჰჭპყ” - ეს ქართული ენის სიტყვა არაა, თუმცა ქართული ანბანით კი არის ჩაწერილი. ამითი განსხვავდება ერთმანეთისაგან „ენის სიტყვა” და „ენის ანბანით ჩაწერილი სიტყვა”.
„ენის ანბანით ჩაწერილი სიტყვა” ამ ენის ანბანის ასოების მიმდევრობაა, რომელსაც რაღაცა სემანტიკური დატვირთვა (ანუ აზრი) შეიძლება ქონდეს, ან არ ქონდეს. რაიმე ანბანით ჩაწერილი სიტყვა შეიძლება იყოს სასრული, ან უსასრულო. როგორც წესი, ჩვენს ყოველდღიურობაში მხოლოდ სასრული სიტყვები გვხვდება. სასრული სიტყვა სასრული ზომისაა, რაც მასში შემავალი ასოების რაოდენობით განისაზღვრება.
მაგალითად, | ანბანი | = 6 და | ენა | = 3. თუ მოცემულია რაღაცა სიტყვა $w = w_1 w_2 ... w_n$, მისი სიგრძე (ანუ ასოების რაოდენობა) შემდეგნაირად აღინიშნება: $|w| = n$. $w(i)$ ამ სიტყვის i-ური ასოა. ასე, მაგალითად, „ანბანი”(4) = „ა” და „ელექტროფიკაცია”(7) = „ო”.
თუ მოცემულია ორი სიტყვა w_1 და w_2, მაშინ $w_1 \circ w_2 = w_1 w_2$ (ეს ორი სიტყვა ერთი მეორეს მიყოლებით). მაგალითად, „ფეხ”∘„ბურთი”=„ფეხბურთი”. თუ რაღაცა სიტყვა $w = u \circ v$, მაშინ ამბობენ, რომ u სიტყვა w სიტყვის პრეფიქსია, ხოლო v სიტყვა w სიტყვის სუფიქსია: $u \prec w$ და $v \succ w$. w სიტყვის n ასოიანი პრეფიქსი აღინიშნება როგორც $w[n]$, ხოლო მისი m ასოიანი სუფიქსი კი აღინიშნება როგორც $w\{m\}$ (არ აგერიოთ $w(n)$ -ში!!!).

სავარჯიშო 5.1: რას ნიშნავს შემდეგი ჩანაწერები: $|w|$, $w[|w|]$, $w(|w|)$, $w[|w|-1]$, $w[0]$, $w\{|w|\}$, $w\{|w|\}$, $w\{|w|-1\}$, $w\{0\}$?

სავარჯიშო 5.2: რას ნიშნავს შემდეგი ჩანაწერები: $|w|$, $w[|w|]$, $w(|w|)$, $w[|w|-2]$, $w[0]$, $w\{|w|\}$, $w\{|w|\}$, $w\{|w|-3\}$, $w\{0\}$, თუ $w =$„ელექტროფიკაცია” ?

სავარჯიშო 5.3: მოცემულია რაღაცა ანბანი A და ორი სიტყვა $w_1 \in A^m$ და $w_2 \in A^n$. რისი ტოლია $|w_1 \circ w_2|$?

თუ $Q = \{$ა, ბ, გ, დ, ... , ჯ, ჰ$\}$ ქართული ანბანია, მაშინ Q^n ყველა იმ სიტყვის სიმრავლეა, რომელიც ქართულ ანბანზეა შედგენილი და რომელთა ასოების რაოდენობაა (ანუ სიგრძეა) n: $Q^n = \{w \mid w(i) \in Q, (1 \leq i \leq n), |w| = n\}$. Q^* ყველა იმ სასრული სიტყვის სიმრავლეა, რომელიც Q ანბანის ასოებითაა შედგენილი:

$$Q^* = \bigcup_{i=1}^{\infty} Q^i = Q^1 \cup Q^2 \cup \cdots \cup Q^n \cup \cdots .$$

სასრული და უსასრულო სიგრძის სიტყვების გარდა არსებობს კიდევ ე.წ. „ცარიელი სიტყვა” ϵ, ანუ ისეთი სიტყვა, რომელიც არც ერთი ასოსაგან არ შედგება (ცარიელია). ცხადია, რომ $|\epsilon| = 0$, $\epsilon \prec w$ და $w \circ \epsilon = \epsilon \circ w = w$ ნებისმიერი w სიტყვისათვის.

ყველაფერი ზემოთ თქმული შეგვიძლია ჩამოვაყალიბოთ ერთ განმარტებაში:

განმარტება 5.1: ნებისმიერი სასრული სიმრავლე A შეგვიძლია განვიხილოთ, როგორც ანბანი. ამ ანბანზე შექმნილი სიტყვაა ამ ანბანის ელემენტების (ანუ ასოების) მიმდევრობაა. თუ w რაიმე A ანბანზე შექმნილი სიტყვაა, $|w|$ ამ სიტყვაში შემავალი ასოების რაოდენობაა. თუ $|w| = 0$, ასეთ სიტყვას ცარიელი ეწოდება და მას აღნიშნავენ სიმბოლოთი ϵ. თუ $|w| = \infty$, ასეთ სიტყვას ეწოდება უსასრულო. თუ მოცემულია ორი სიტყვა w და v (სადაც w სასრულია), მაშინ $w \circ v = wv$ ამ ორი სიტყვის ***კონკატენაცია***, ანუ შერწყმაა. ამბობენ, რომ u სიტყვა w სიტყვის პრეფიქსია ($u \prec w$), თუ $\exists v$ სიტყვა ისეთი, რომ $w = u \circ v$. ანალოგიურად, u სიტყვა w სიტყვის სუფიქსია, ($u \succ w$), თუ $\exists v$ სიტყვა ისეთი, რომ $w = v \circ u$. თუ w რაიმე სიტყვაა, მაშინ $w(n)$ მისი მე-n-ე ასოა, $w[n]$ მისი n ასოსაგან შემდგარი პრეფიქსი, ხოლო wn კი - მისი n ასოსაგან შემდგარი სუფიქსი.
თუ მოცემულია A ანბანი, მაშინ $A^n = \{w \,|\, |w| = n\}$ და $A^* = \{w \,|\, |w| < \infty\}$

სავარჯიშო 5.4: მოცემულია ორი სიტყვა $w_1 \in A^*$ და $w_2 \in B^*$, სადაც A და B რაღაცა ანბანებია. რა ანბანის სიტყვაა $w_1 \circ w_2$?

სავარჯიშო 5.5: მოცემულია სიტყვები $w_1 = 00134$, $w_2 = 65430$, $w_3 = 001$, $w_4 = 346$. ჭეშმარიტია თუ არა შემდეგი გამონათქვამი: $w_3 \circ w_4 = w_1 \circ w_4[6]$? პასუხი დაამტკიცეთ.

ამბობენ, რომ $w \in A^*$ სიტყვა $v \in A^*$ სიტყვას შეიცავს, თუ $\exists w_1, w_2 \in A^*$ და $w = w_1 \circ v \circ w_2$ (w_1 ან w_2 ცარიელიც შეიძლება იყოს). ამ შემთხვევაში იტყვიან, რომ v სიტყვა w სიტყვის ქვესიტყვაა. მაგალითად, თუ გვაქვს სიტყვა „მოდიფიკაცია", მაშინ მისი ქვესიტყვებია „დიფიკა", „კაცი", „მოდი", „კაცია". ამას გარდა, „მოდი" მისი პრეფიქსია, ხოლო „კაცია" კი - სუფიქსი. მაგრამ „მოდიკაცია" მისი ქვესიტყვა არაა, თუმცა შედგება ორი ქვესიტყვისაგან.

სავარჯიშო 5.6: ჭეშმარიტია თუ არა შემდეგი გამონათქვამები: $w \in A^{|w|}$, $w \in A^{|w|-1}$, $w[k] \in A^k$ თუ w სიტყვა A ანბანზეა შედგენილი და $k \in \mathbb{N}$? პასუხები დაამტკიცეთ.

ანალოგიურად სიტყვები შეიძლება შევადგინოთ ნებისმიერ სხვა ანბანზე, ანუ სასრულ სიმრავლეზე. მაგალითად, თუ მოცემულია ათობითი ანბანი $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, მასში შეიძლება ყველა ნატურალური რიცხვი ჩაიწეროს. ასეთ ჩანაწერს „რიცხვის ათობითი ჩანაწერი" ეწოდება, რადგან მის გამოსახატავად (ჩასაწერად) მხოლოდ ეს 10 ასო, ანუ ციფრი გამოიყენება.
როგორც აღმოჩნდა, შეიძლება უსასრულოდ ბევრი ანბანის შექმნა. თუ M_1 და M_2 სხვადასხვა ანბანებია, არსებობს ბიექტიური ასახვა $f : M_1^* \to M_2^*$, რაც იმას ნიშნავს, რომ ანბანის შერჩევას მნიშვნელობა არ აქვს: რაც ერთი ანბანით ჩაიწერება, იგივე სხვა ნებისმიერი ანბანითაც შეიძლება ჩაიწეროს.
მაგალითად, ქართული ანბანის სიტყვები ათობითი ანბანით შემდეგნაირად შეიძლება ჩაიწეროს:
პირველ რიგში ქართული ანბანის თითო ასო ათობითი ანბანის სიტყვებად უნდა ჩავწეროთ:

ა $\to$ 00	ბ $\to$ 01	გ $\to$ 02	დ $\to$ 03	ე $\to$ 04	ვ $\to$ 05	ზ $\to$ 06	თ $\to$ 07	ი $\to$ 08	კ $\to$ 09
ლ $\to$ 10	მ $\to$ 11	ნ $\to$ 12	ო $\to$ 13	პ $\to$ 14	ჟ $\to$ 15	რ $\to$ 16	ს $\to$ 17	ტ $\to$ 18	უ $\to$ 19
ფ $\to$ 20	ქ $\to$ 21	ღ $\to$ 22	ყ $\to$ 23	შ $\to$ 24	ც $\to$ 25	ჩ $\to$ 26	ძ $\to$ 27	წ $\to$ 28	ჭ $\to$ 29
ხ $\to$ 30	ჯ $\to$ 31	ჰ $\to$ 32							

შემდეგ ქართული ანბანით ჩაწერილი ყოველი სიტყვის ასო შესაბამისი ორეულით უნდა შევცვალოთ. მაგალითად, „კონსპექტი" შემდეგნაირად ჩაიწერება: „091312171404211808".

სავარჯიშო 5.7: როგორ ჩაიწერება ამ მეთოდებით სიტყვა „ელექტროფიკაცია" ? რომელი ქართული სიტყვაა ჩაწერილი სიტყვით „02001113250300" ?

თუ ცნობილია, რომელ ასოს რომელი ციფრების წყვილი (ორეული) შეესაბამება, ადვილი გამოსაანგარიშებელია ქართული სიტყვა. მაგრამ თუ ათობითში ჩაწერილი ეს სიტყვა ვინმეს ჩაუვარდა ხელში, ვინც არ იცის, თუ რომელ რიცხვს რომელი ასო შეესაბამება, ქართული სიტყვის აღდგენა გაძნელდება. ძალიან ძნელი იქნება საწყისი სიტყვის აღდგენა, თუ ჩვენ ასოებს ორ ნიშნა რიცხვებს შევუსაბამებთ ისე, როგორც ამას ჩვენ მოვინდომებთ, მაგალითად:

ა $\to$ 39	ბ $\to$ 27	გ $\to$ 99	დ $\to$ 03	ე $\to$ 38	ვ $\to$ 21	ზ $\to$ 76	თ $\to$ 78	ი $\to$ 87	კ $\to$ 90
ლ $\to$ 10	მ $\to$ 11	ნ $\to$ 13	ო $\to$ 31	პ $\to$ 37	ჟ $\to$ 65	რ $\to$ 16	ს $\to$ 17	ტ $\to$ 18	უ $\to$ 19
ფ $\to$ 47	ქ $\to$ 51	ღ $\to$ 66	ყ $\to$ 08	შ $\to$ 24	ც $\to$ 25	ჩ $\to$ 26	ძ $\to$ 00	წ $\to$ 01	ჭ $\to$ 09
ხ $\to$ 81	ჯ $\to$ 06	ჰ $\to$ 32							

თუ ცნობილია, რომელ ასოს რა ორეული შეესაბამება (მაგალითად ისე, როგორც ზედა ცხრილშია მოყვანილი), შეგვიძლია რაღაცა $f : Q \to A \times A$ ფუნქციის შედგენა (აქ Q ქართული ანბანია და $A = \{0, 1, 2, ..., 9\}$). თუ ეს ფუნქცია შედგენილია ზედა ცხრილის საშუალებით, მაშინ $f(\text{ა}) = 39$, $f(\text{ბ}) = 27$, $f(\text{ლ}) = 10$, $f(\text{შ}) = 24$ და ა.შ.

რაღაცა სიტყვა $w \in Q^n$ კი შემდეგი რეკურსიული ალგორითმით შეიძლება ჩავწეროთ ათობითი ანბანის გამოყენებით:

ალგორითმი 5.1: $P(w)$
მონაცემი: $w \in Q^{|w|}$

```
if(w = ϵ)
{
  ალგორითმი დაასრულე
}
else
{
  პასუხად გამოიტანე სიტყვა „P(w[|w| − 1]) ∘ f(w(|w|))"
}
```

(აქ f ფუნქცია ზემოთ მოყვანილი ცხრილითაა განსაზღვრული).

ამ ალგორითმში სამი სიახლეა შემოტანილი:

1. ეს ალგორითმი უფრო ფორმალურადაა ჩაწერილი, ვიდრე აქამდე მოყვანილ ყველა მაგალითში: ჩანაწერი „ალგორითმი $P(w)$" ნიშნავს, რომ ამ ალგორითმს სახელად ეწოდება P, ხოლო მონაცემად (ან, სამეცნიერო ტერმინოლოგია რომ ვიხმაროთ, არგუმენტად) მოცემული აქვს სიტყვა w;

2. იმის მაგივრად, რომ სიტყვიერად ვწეროთ: „თუ $w = \epsilon$, მაშინ ალგორითმი დაასრულე", ჩვენ ვწერთ $if(w = \epsilon)$ ალგორითმი **დაასრულე**. თუ if ოპერატორის ფრჩხილებში ჩასმული პირობა ჭეშმარიტია, მაშინ სრულდება მის შემდეგ ფიგურულ ფრჩხილებში ჩასმული ბრძანებები. თუ ეს პირობა მცდარია, მაშინ შესრულდება $else$ შემდეგ ფიგურულ ფრჩხილებში ჩასმული ბრძანებები;

3. იმის მაგივრად, რომ სიტყვიერად ვწეროთ: „ჩაატარე იგივე ოპერაციები $w[|w| - 1]$ მონაცემისათვის", ჩვენ ვწერთ $P(w[|w| - 1])$ (რადგან ამ ალგორითმს ეწოდება P, ამიტომ $P(w[|w| - 1])$ ნიშნავს: ჩაატარე ალგორითმი P მონაცემით $w[|w| - 1]$.

<u>სავარჯიშო 5.8:</u> დაწვრილებით აღწერეთ P(„ხელი") ალგორითმის მსვლელობა (რას აკეთებს ყოველ ბიჯში).

თუ ქართულ სიტყვებს ბოლოს მოყვანილი ცხრილის მიხედვით ჩავწერთ ათობით ანბანში, მაშინ საწყისი ტექსტის აღდგენა საკმაოდ გაძნელდება, თუ ასოებსა და ციფრთა ორეულებს შორის შესაბამისობები ცნობილი არ არის.

<u>სავარჯიშო 5.9:</u> რომელი ქართული სიტყვაა კოდირებული ზემოთ მოყვანილი ცხრილის მიხედვით ათობით ანბანზე შედგენილ სიტყვაში „99001131250300" ? როგორ შეიძლება ჩავწეროთ სიტყვა „წყალი" ?

აღსანიშნავია, რომ არსებობს მეთოდები, რომელთა საშუალებითაც შეიძლება ზედა ცხრილის მიხედვით კოდირებული ტექსტის გახსნა იმის და მიუხედავად, თუ ცხრილი ცნობილი არ არის: თუ ვიცით, რომ კოდირებულია ქართული ტექსტი, მოვძებნოთ იმ ორეულს, რომელიც ყველაზე ხშირად გვხვდება. რადგან ქართულ ენაში ყველაზე ხშირია ასო „ე", ამიტომ სავარაუდოა, რომ ის ორეულიც „ე" ასოს შესაბამისი იქნება. შემდეგ დავითვლით იმ ოთხეულების რაოდენობას, რომელიც „ე" ასოს შესაბამისი ორეულით იწყება. ქართულ ენაში გამოკვლეულია, თუ რომელი ასო გვხვდება ყველაზე ხშირად „ე" ასოს შემდეგ. ანალოგიურად და რამოდენიმე ექსპერიმენტის ჩატარების შედეგად ტექსტის გაშიფვრა შესაძლებელია.
მონაცემთა ამდაგვარი კოდირებითა და გახსნით დაკავებულია ინფორმატიკისა და მათემატიკის ერთ-ერთი განხრა - კრიპტოგრაფია.
თუ მოცემულია რაიმე ანბანი A, მაშინ $L \subset A^*$ ამ ანბანზე შედგენილი ენა ეწოდება.
ანბანსა და ენას ცენტრალური როლი ენიჭება ინფორმატიკაში, რადგან დამტკიცდა, რომ ნებისმიერი ამოცანა შეიძლება რაღაცა ენაში ჩაიწეროს და მისი ამოხსნის ძიება ამ ენაში გარკვეული სიტყვების ძიების ტოლფასია.
ინფორმატიკაში ძალიან მნიშვნელოვანია ე.წ. „ორობითი ანბანი" $\mathbb{B} = \{0, 1\}$. ნებისმიერი ინფორმაცია შეიძლება ჩაიწეროს ამ ანბანის სიტყვებით, ანუ ორობით კოდში.
მაგალითად, თუ მოცემულია რაიმე ნატურალური რიცხვი $n \in \mathbb{N}$, მისი ჩაწერა ორობით კოდში შემდეგი ალგორითმით შეიძლება:

ალგორითმი 5.2: Binary
მონაცემი: $n \in \mathbb{N}$

$if(n = 0)$
{
ამობეჭდე 0 და ალგორითმი **დაასრულე**
}
$else\ if(n = 1)$
{
ამობეჭდე 1 და ალგორითმი **დაასრულე**
}
$else$
{
Binary($\lfloor \frac{n}{2} \rfloor$) (ეს პროცედურა გაიმეორე $\lfloor \frac{n}{2} \rfloor$ მონაცემისათვის)
ამობეჭდე $\frac{n}{2}$ გაყოფისას მიღებული ნაშთი
(თუ n კენტია, ამობეჭდე „1") ;
(თუ n ლუწია, ამობეჭდე „0") ;
}

აღსანიშნავია, რომ ეს ალგორითმი რიგ-რიგობით ამობეჭდავს შესაბამისი ნატურალური რიცხვის ორობით ციფრებს (ანუ ბიტებს).

სავარჯიშო 5.10: განიხილეთ შემდეგი ალგორითმი:

ალგორითმი 5.3: Binary
მონაცემი: $n \in \mathbb{N}$

$if(n = 0)$
{
ამობეჭდე 0 და ალგორითმი **დაასრულე**
}
$else\ if(n = 0)$
{
ამობეჭდე 0 და ალგორითმი **დაასრულე**
}
$else$
{
ამობეჭდე $\frac{n}{2}$ გაყოფისას მიღებული ნაშთი
(თუ n კენტია, ამობეჭდე „1") ;
(თუ n ლუწია, ამობეჭდე „0") ;
Binary($\lfloor \frac{n}{2} \rfloor$) (ეს პროცედურა გაიმეორე $\lfloor \frac{n}{2} \rfloor$ მონაცემისათვის)
}

რა იქნება მისი პასუხი?

სავარჯიშო 5.11: გადაიყვანეთ ორობით კოდში შემდეგი რიცხვები: 13, 127, 17, 8, 16, 0.

თუ გვაქვს მოცემული ორობით კოდში ჩაწერილი რაღაც რიცხვი, მაგალითად $a = 110110_2$, მისი ცალკეული ციფრები (ანუ ბიტები) შეიძლება გადაინომროს ***მარჯვნიდან მარცხნივ***, დაწყებული ნულიდან: $a = a_5a_4a_3a_2a_1a_0$. აქ $a_5 = 1, a_4 = 1, a_3 = 0, a_2 = 1, a_1 = 1, a_0 = 0$. რა თქმა უნდა, ბიტები შეიძლება გადაგვენომრა მარცხნიდან მარჯვნივაც, ან ინდექსები დაწყებული ერთიდან, მაგრამ სტანდარტულად მიღებულია ზემოთ ნაჩვენები ნუმერაცია.

ანალოგიურად შეიძლება ნებისმიერი რიცხვის ნებისმიერი ანბანით ჩაწერა. თუ მოცემულია k ასოიანი ანბანი, მაშინ იტყვიან, რომ მისი სიტყვები ჩაწერილია k ბაზისით:

ალგორითმი 5.4: k-ary
მონაცემი: $n \in \mathbb{N}$

$if(n < k)$
{
ამობეჭდე k და ალგორითმი **დაასრულე**
}
$else$
{
Binary($\lfloor \frac{n}{k} \rfloor$) (ეს პროცედურა გაიმეორე $\lfloor \frac{n}{k} \rfloor$ მონაცემისათვის)
ამობეჭდე $\frac{n}{k}$ გაყოფისას მიღებული ნაშთი
}

სავარჯიშო 5.12: წინა სავარჯიშოში მოყვანილი რიცხვები ჩაწერეთ რვაობით, თექვსმეტობით და ორობით კოდებში.

სავარჯიშო 5.13: დაწერეთ ალგორითმი, რომელიც ორობით კოდში ჩაწერილ რიცხვს ათობით კოდში გადაიყვანს.

5.2 მოდულარული არითმეტიკა: უსასრულო სისტემის სიმულაცია სასრულით

ყველასათვის კარგადაა ცნობილი, თუ როგორ შეიძლება საათის ცნობა. თუ დღე-ღამეში 24 საათს ვივარაუდებთ (რაც საყოველთაოდაა მიღებული), მაშინ პირველი საათის შემდეგ იქნება 2, მას შემდეგ 3, 4, 5, ..., 23 და 23 საათის შემდეგ დგება 24, **ანუ 0** საათი. აქედან გამომდინარე, გვაქვს შემდეგი წესი: $0 + 1 = 1$, $1 + 2 = 3$, ..., $22 + 1 = 23$, $23 + 1 = 0$ და მთელი ციკლი თავიდან იწყება. რადგან სულ გამოყენებულია 24 რიცხვი 0, 1, ..., 23, ამბობენ, რომ გვაქვს განსაზღვრული მიმატება 24-ის **მოდულით**. ასე რომ, $12+7\, mod\, 24 = 19$, $23+1\, mod\, 24 = 0$, $12+15\, mod\, 24 = 3$, $503 + 20167\, mod\, 24 = 6$. ზოგადი პრინციპი ასეთია: ჩვეულებრივად ვკრებთ ორ რიცხვს, შედეგს ვყოფთ 24-ზე და ვიღებთ *ნაშთს*.
ანალოგიურად შეიძლება განისაზღვროს აგრეთვე გამრავლებაც: ორ რიცხვს ვამრავლებთ ერთმანეთზე, შედეგს ვყოფთ 24-ზე და ვიღებთ ნაშთს. მაგალითად, $3 \cdot 7\, mod\, 24 = 21$, $103 \cdot 17\, mod\, 24 = 23$.

სავარჯიშო 5.14: გამოითვალეთ $13 + 17\, mod\, 24$, $9 + 23\, mod\, 24$, $23 \cdot 5\, mod\, 24$, $5 \cdot 17\, mod\, 24$.

ჩვენს სიმრავლეში $\mathbb{Z}_{24} = \{0, 1, 2, 3, , 4..., 23\}$ გვხვდება აგრეთვე ე.წ. ***შეკრების ოპერაციის მიმართ ნეიტრალური ელემენტი*** 0, რომელიც მიმატებისას რიცხვს არ ცვლის: $a+0\, mod\, 24 = a$. აქედან გამომდინარე შეიძლება დავსვათ შემდეგი შეკითხვა: არსებობს თუ არა $\mathbb{Z}_{24}$ სიმრავლეში ყოველი $a \in \mathbb{Z}_{24}$ ელემენტის ***შებრუნებული*** ელემენტი $-a \in \mathbb{Z}_{24}$? რაიმე ელემენტს მისი შებრუნებულის მიმატებით უნდა ვიღებდეთ ნეიტრალურ ელემენტს: $a+(-a)\, mod\, 24 = 0$. მაგალითად, $11+13\, mod\, 24 = 0$, $23+1\, mod\, 24 = 0$, $7+17\, mod\, 24 = 0$ და ა.შ. აქედან ვასკვნით, რომ $a \in \mathbb{Z}_{24}$ რიცხვის შებრუნებულის საპოვნელად საკმარისია გამოვიანგარიშოთ $24 - a$.

უფრო რთულადაა საქმე, როდესაც რაიმე $a \in \mathbb{Z}_{24}$ რიცხვის შებრუნებულს ვეძებთ *გამრავლების* მიმართ: ვიპოვნოთ ისეთი $a^{-1} \in \mathbb{Z}_{24}$, რომ $a \cdot a^{-1}\, mod\, 24 = 1$.
აღსანიშნავია, რომ ***გამრავლების მიმართ ნეიტრალური ელემენიტია*** 1: $a \cdot 1\, mod\, 24 = a$ (რიცხვი გამრავლების შემდეგ უცვლელი რჩება).
თუ $13 \cdot 3\, mod\, 24 = 1$ და აქედან ვასკვნით, რომ $13 \in \mathbb{Z}_{24}$ რიცხვის შებრუნებული უნდა იყოს ისევ $3 \in \mathbb{Z}_{24}$ (და პირიქით), მაგრამ რიცხვის $6 \in \mathbb{Z}_{24}$ შებრუნებულს ვერაფრით ვერ ვიპოვნით. აქედან ვასკვნით, რომ გამრავლების მიმართ შებრუნებული ყველა რიცხვს შეიძლება არც ქონდეს.

საინტერესოა ის ფაქტიც, რომ 24 მოდულით არითმეტიკაში რამოდენიმე რიცხვი შეიძლება იყოს თავისი თავის შებრუნებული: $1 \cdot 1\, mod\, 24 = 1$ და $7 \cdot 7\, mod\, 24 = 1$. აქედან ვასკვნით, რომ $7 \in \mathbb{Z}_{24}$ რიცხვის შებრუნებული უნდა იყოს ისევ $7 \in \mathbb{Z}_{24}$, ანუ ეს რიცხვი თავისი თავის შებრუნებულია.

სავარჯიშო 5.15: გამოიანგარიშეთ $(-13)\, mod\, 24$, $(-1)\, mod\, 24$, $13^{-1}\, mod\, 24$, $-13\, mod\, 24$, $17^{-1} mod\, 24$.

სავარჯიშო 5.16: დაადგინეთ, $\mathbb{Z}_{24}$ სიმრავლეში რომელ რიცხვებს მოეძებნებათ გამრავლების მიმართ შებრუნებული და რომელს - არა. რა კანონზომიერებაა ამ რიცხვებსა და მოდულს (24) შორის?

ანალოგიურად შეიძლება განვსაზღვროთ არითმეტიკული ოპერაციები ნებისმიერი $m \in \mathbb{N}$ მოდულით: თუ $a, b \in \mathbb{Z}_m$, გამოვიანგარიშოთ $c = a + b$ ან $d = a \cdot b$, შედეგები გავყოთ m რიცხვზე და გამოვითვალოთ ნაშთი.

სავარჯიშო 5.17: გამოიანგარიშეთ $(-13)\, mod\, 27$, $(-1)\, mod\, 9$, $13^{-1}\, mod\, 27$, $-13\, mod\, 31$, $17^{-1} mod\, 41$.
სავარჯიშო 5.18: გამოითვალეთ $13 + 17\, mod\, 34$, $9 + 23\, mod\, 4$, $23 \cdot 5\, mod\, 32$, $5 \cdot 17\, mod\, 47$.

მათემატიკიდან ცნობილია შემდეგი მნიშვნელოვანი თეორემა:

თეორემა 5.1: $a \in \mathbb{Z}_k$ რიცხვს მოეძებნება შებრუნებული გამრავლების მიმართ მაშინ და მხოლოდ მაშინ, თუ a და k რიცხვები ურთიერთმარტივია.

ბუნებრივია შემდეგი შეკითხვა: რაში შეიძლება გამოვიყენოთ მოდულარული არითმეტიკა? ეს საკითხი დიდი ხნის წინ დაისვა და გარკვეული პერიოდის განმავლობაში ითვლებოდა კიდეც, რომ მოდულარული არითმეტიკა და, უფრო ზოგადად, სასრული სტრუქტურების შესწავლა (მაგალითად სასრული რგოლები ან ველები, სასრული ჯგუფები და სხვა) მხოლოდ თეორიული ხასიათის იყო და XIX საუკუნეში მიღებული დიდი შედეგების შემდეგ (როგორიცაა, მაგალითად, გალუას თეორიაზე დაყრდნობით მეხუთე ან მეტი რიგის განტოლებებისათვის ამონახსნის ფორმულების არარსებობის მტკიცება ან ზემოთ ნახსენები ფარგლითა და სახაზავით აგების კლასიკური ამოცანების გადაჭრა) პრაქტიკაში გამოყენების თვალსაზრისით აღარ გამოდგებოდა, მაგრამ XX საუკუნეში გამომთვლელი მანქანების განვითარებამ, მოდულარული არითმეტიკა და, ზოგადად, სასრული სტრუქტურების აქტუალობა ისევ წინა პლანზე წამოწია.
რადგან გამომთვლელ მანქანებში მეხსიერება შეზღუდულია, შეუძლებელია ისეთი უსასრულო სტრუქტურების წარმოდგენა, როგორიცაა ნატურალური, რაციონალური და, მით უმეტეს, ირაციონალური რიცხვები. აქედან გამომდინარე, შეუძლებელია არითმეტიკის სრულყოფილად წარმოდგენაც, რაც საკმაოდ დიდი პრობლემების შემქმნელი შეიძლება აღმოჩნდეს.
ერთ-ერთი გამოსავალია სასრული სტრუქტურების არითმეტიკის გამოყენება: ჩვენ ვიღებთ რაიმე რგოლს ერთეულოვანი ელემენტებით, მაგალითად $\mathbb{Z}_k$ (აქ k მარტივია) და გამოთვლასაც k რიცხვის მოდულით ვაწარმოებთ. პრაქტიკაში გამოყენებული ამოცანებისთვის დიდი k შერჩევით ყველა ოპერაციაც შეიძლება დამაკმაყოფილებელი იყოს, რითაც უსასრულო სტრუქტურის - ჩვეულებრივი არითმეტიკის - სიმულაცია გახდება შესაძლებელი სასრული სტრუქტურებით. რა თქმა უნდა, მოდულარული არითმეტიკა ყოველთვის არ მოგვცემს „სწორ” პასუხს - გარკვეულ შემთხვევებში შესაძლებელია ისეთი დიდი რიცხვები დაგვჭირდეს, რომლებიც $\mathbb{Z}_k$ სასრულ სიმრავლეში აღარ მოთავსდება, მაგრამ ეს შეცდომები პრაქტიკაში შესაძლებელია დიდ როლს არ თამაშობდეს.
ამას გარდა, არსებობს მეთოდები (მაგალითად, ჩინური თეორემა ნაშთების შესახებ), რითაც ორ ან მეტ პატარა რგოლში ჩატარებული გამოთვლიდან უფრო დიდ რგოლში ჩატარებული გამოთვლის გამოანგარიშება შეიძლება. მაგრამ ეს ყველაფერი კომპიუტერული ალგებრის საკითხებია, რასაც ჩვენ დაწვრილებით სხვა კურსებში განვიხილავთ.

5.3 ორობითი არითმეტიკის ელემენტები

განვიხილოთ ორობით კოდში ჩაწერილი რიცხვები $A = (a_3a_2a_1a_0) = (1101)$ და $B = (b_3b_2b_1b_0) = (1110)$. ათობით კოდში ჩაწერილი რიცხვების ანალოგიურად, აქაც „ქვეშ მიწერით” მიმატებაა შესაძლებელი:

$$
\begin{array}{ccccc} \color{red}{0} & 1 & 1 & 0 & 1 \\ \color{red}{0} & 1 & 1 & 1 & 0 \\ \hline x_4 & x_3 & x_2 & x_1 & x_0 \end{array} \qquad \begin{array}{c} 13 \\ 14 \\ \end{array}
$$

პირველ რიგში ვკრებთ „დაბალ” (მარჯვენა) ბიტებს და ვიღებთ: $x_0 = 1 \oplus 0 = 1$ (აქ $\oplus$ ორობით, ანუ ორის მოდულით მიმატებას ნიშნავს). როგორც ათობითში, ორობით მიმატებაშიც უნდა დავიხსომოთ 1 ან 0. ამ შემთხვევაში ვიხსომებთ $c_1 = 0$, რადგან პირველი ბიტების შესაკრებებში ორი ერთიანი არ გვხვდება.
ჯამის შემდგომი ბიტის გამოსაანგარიშებლად გვექნება: $x_1 = 0 \oplus 1 \oplus c_1 = 0 \oplus 1 \oplus 0 = 1$ (უნდა შევკრიბოთ შესაბამისი ბიტები და წინა ბიჯში „დახსომებული” ციფრი). ამ ბიჯში ვიხსომებთ $c_2 = 0$, რადგან x_1 ჯამში განხილულ სამ შესაკრებში ორზე ნაკლები 1 შეგვხვდა. შემდეგ ვითვლით $x_3 = 1 \oplus 1 \oplus d_2 = 1 \oplus 1 \oplus 0 = 0$, ხოლო დახსომებული იქნება $c_3 = 1$, რადგან აქ უკვე ორი ერთიანი შეგვხვდა x_2 ჯამის გამოთვლისას.
ვითვლით $x_3 = 1 \oplus 1 \oplus c_3 = 1 \oplus 1 \oplus 1 = 1$ და ვიხსომებთ $c_4 = 1$ (იგივე მოსაზრებით).
ბოლოს უნდა გამოვითვალოთ $x_4 = 0 \oplus 0 \oplus c_4 = 0 \oplus 0 \oplus 1 = 1$ და ვიღებთ შედეგს:

$$\begin{array}{ccccc c} 0 & 1 & 1 & 0 & 1 & \quad 13 \\ 0 & 1 & 1 & 1 & 0 & \quad 14 \\ \hline 1 & 1 & 0 & 1 & 1 & \quad 27 \end{array}$$

სავარჯიშო 5.19: დეტალურად ამოწერეთ $(10010101)_2 + (10010101)_2$, $(11110101)_2 + (00010101)_2$ და $(10010001) +_2 (10011101)_2$ რიცხვების შეკრების ბიჯები (ყოველ ბიჯში გამოანგარიშებული შედეგი და დახსომებული რიცხვი).

საბოლოოდ მივიღებთ შემდეგ ალგორითმს:

Algorithm 5.5: ორობითი რიცხვების შეკრება

procedure $SumBinary((a_n, ..., a_0), (b_n, ..., b_0))$
  $c_0 = 0;$
  for ($i = 0, i \leq n, i++$)
   {
   $x_i = a_i \oplus b_i \oplus c_i;$
   if (a_i, b_i და c_i ცვლადებში ორი ან სამი 1 გვხვდება)
    then $c_{i+1} = 1;$
    else $c_{i+1} = 0;$
   }
  $x_{n+1} = c_{n+1};$

სავარჯიშო 5.20: ახსენით, რატომ არის საჭირო მხოლოდ x_{n+1} ცვლადის გამოთვლა და არა, მაგალითად, x_{n+2} ან x_{n+3}?

სავარჯიშო 5.21: დაამტკიცეთ ზემოთ მოყვანილი ალგორითმის სისწორე და გამოითვალეთ მისი ბიჯების რაოდენობა.

სავარჯიშო 5.22: შეკრების ანალოგიურად დაწერეთ ქვეშ მიწერით გამრავლების ალგორითმი, დაამტკიცეთ მისი სისწორე და გამოითვალეთ მისი ბიჯების რაოდენობა.

ადვილი დასანახია, რომ თუ მოცემული გვაქვს n ბიტიანი რიცხვები და შეგვიძლია მხოლოდ ამ სიგრძის რიცხვების წარმოდგენა, მივიღებთ 2^n მოდულით არითმეტიკას, ანუ გამოთვლას ჩავატარებთ $\mathbb{Z}_{2^n}$ სიმრავლეში, რადგან n ბიტით 2^n სხვადასხვა რიცხვის წარმოდგენა შეიძლება.

თუ განვიხილავთ 8 ბიტიან რიცხვს $11111111_2 = 255_{10}$, ვნახავთ, რომ $11111111_2 + 1 \, mod \, 2^8 = 0$, ანუ $-255 \, mod \, 256 = 1$.

ზოგადად, თუ მოცემულია n ბიტიანი რიცხვი 111...1, რომლის ბიტები ყველა ტოლია 1, მიმატების მიმართ მისი შებრუნებული რიცხვი იქნება 1.

სავარჯიშო 5.23: დაამტკიცეთ ზემოთ მოყვანილი გამონათქვამი: თუ მოცემულია n ბიტიანი რიცხვი 111...1, რომლის ბიტები ყველა ტოლია 1, მიმატების მიმართ მისი შებრუნებული რიცხვი იქნება 1.

ახლა კი განვიხილოთ რაიმე სხვა 8 ბიტიანი რიცხვი, მაგალითად $x = 11010010$. როგორ გამოვიანგარიშოთ მისი შებრუნებული მიმატების მიმართ?

ყოველივე ზემოთ თქმულიდან გამომდინარე, ლოგიკური იქნებოდა ჯერ გამოგვეანგარიშებინა რაღაც რიცხვი y, რომელიც შემდეგი თვისების მატარებელია: $x + y = 11111111$, რის შედეგადაც მივიღებთ: $-x \, mod \, 2^n = y + 1$.

როგორ შეიძლება შესაბამისი y რიცხვის გამოანგარიშება? არაა რთული საჩვენებელი, რომ $y = 00101101$, ანუ ისეთი რიცხვი, რომლის ბიტებში x რიცხვის ბიტის საწინააღმდეგო (ანუ უარყოფა) წერია: თუ წერია 0, ვიღებთ 1 და თუ წერია 1, ვიღებთ 0. ზოგადად, თუ მოცემულია რაიმე რიცხვი $x_{n-1}...x_1x_0$, მისი „შებრუნებული რიცხვი" იქნება $y = \overline{x_{n-1}}...\overline{x_1}, \overline{x_0}$, სადაც $\overline{x_i}$ აღნიშნავს x_i ბიტის უარყოფას: $\overline{1} = 0$, $\overline{0} = 1$.

აქედან გამომდინარე, ნებისმიერი n ბიტიანი რიცხვის მიმატების მიმართ შებრუნებული რიცხვი 2^n მოდულით იქნება ამ რიცხვის უარყოფას (ბიტების შებრუნებას) მიმატებული 1:

$$-(x_{n-1}...x_1x_0) \, mod \, 2^n = (\overline{x_{n-1}}...\overline{x_1}, \overline{x_0}) + 1.$$

5.4 ორობითი კოდირების გამოყენების მაგალითები და მომგებიანი სტრატეგია თამაშებში

როგორც ვნახეთ, ნებისმიერი ინფორმაციის (ნებისმიერ ანბანზე აგებული სიტყვისა თუ ენის) კოდირება შეიძლება რიცხვებით: ანბანის თითოეულ ასოს შევუსაბამებთ რაიმე ნატურალურ რიცხვს და, აქედან გამომდინარე, ნებისმიერ სიტყვას შესაბამისი რიცხვების კოდების მიმდევრობით შედგენილ ნატურალურ რიცხვს შევუსაბამებთ.
რადგან ნებისმიერი ნატურალური რიცხვი შეგვიძლია გადავიყვანოთ ორობით სისტემაში, ზემოთ თქმულიდან გამომდინარე, ნებისმიერი სიტყვაც შეგვიძლია წარმოვადგინოთ ორობითი ანბანის მეშვეობით.
ინფორმაციის კოდირება ორობით კოდში ფართოდ გამოიყენება პრაქტიკაში. თანამედროვე გამოთვლითი სისტემები მთლიანად ამ პრინციპს ეფუძნება და, სხვა ოპერაციებთან ერთად, მოდულარულ არითმეტიკაზე დაყრდნობით ახდენს უსასრულო სისტემების სასრულით სიმულაციასა და არითმეტიკულ ოპერაციებს.
ეს, რა თქმა უნდა, ორობითი სისტემის ყველაზე დიდი და მნიშვნელოვანი გამოყენების სფეროა. ქვემოთ ჩვენ უფრო ნაკლებად ცნობილ, მაგრამ საინტერესო გამოყენებას განვიხილავთ, რაც თამაშებში მომგებიანი სტრატეგიის შემუშავებას ეხება.

თამაში ასანთებით

მოცემულია ასანთების სამი გროვა. პირველ გროვაშია x_1 ასანთი, მეორეში x_2 და მესამეში x_3.
ორი მოთამაშე რიგ-რიგობით იღებს რამოდენიმე ასანთს ერთი და მხოლოდ ერთი გროვიდან. მოგებულია ის მოთამაშე, რომელიც ბოლოს აიღებს ასანთს და მოწინააღმდეგეს არაფერი აღარ დარჩება.

მაგალითად, პირველ გროვაშია 3 ასანთი, მეორეში 9 და მესამეში კი 6.

მოცემულია: $x_1 = 3$, $x_2 = 9$, $x_3 = 6$.

- პირველი მოთამაშე იღებს 3 ასანთს მეორე კონიდან: $x_2 = x_2 - 3$. დარჩება: $x_1 = 3$, $x_2 = 9 - 3 = 6$, $x_3 = 6$.
- მეორე მოთამშე ისევე მეორე კონიდან იღებს 2 ასანთს: $x_2 = x_2 - 2$. დარჩება: $x_1 = 3$, $x_2 = 6 - 2 = 4$, $x_3 = 6$.
- პირველი მოთამაშე იღებს 1 ასანთს მესამე კონიდან: $x_3 = x_3 - 1$. დარჩება: $x_1 = 3$, $x_2 = 4$, $x_3 = 6 - 1 = 5$.
- მეორე მოთამშე პირველი კონიდან იღებს 3 ასანთს: $x_1 = x_1 - 3$. დარჩება: $x_1 = 3 - 3 = 0$, $x_2 = 4$, $x_3 = 5$.
- პირველი მოთამაშე იღებს 1 ასანთს მესამე კონიდან: $x_3 = x_3 - 1$. დარჩება: $x_1 = 0$, $x_2 = 4$, $x_3 = 5 - 1 = 4$.
- მეორე მოთამშე მეორე კონიდან იღებს 3 ასანთს: $x_2 = x_2 - 3$. დარჩება: $x_1 = 0$, $x_2 = 4 - 3 = 1$, $x_3 = 4$.
- პირველი მოთამაშე იღებს 3 ასანთს მესამე კონიდან: $x_3 = x_3 - 3$. დარჩება: $x_1 = 0$, $x_2 = 1$, $x_3 = 4 - 3 = 1$.
- მეორე მოთამშე მეორე კონიდან იღებს 1 ასანთს: $x_2 = x_2 - 1$. დარჩება: $x_1 = 0$, $x_2 = 1 - 1 = 0$, $x_3 = 1$.
- პირველი მოთამაშე იღებს 1 ასანთს მესამე კონიდან: $x_3 = x_3 - 1$. დარჩება: $x_1 = 0$, $x_2 = 0$, $x_3 = 1 - 1 = 0$.

პირველმა მოთამაშემ მოიგო, რადგან მოწინააღმდეგეს სვლა აღარ დარჩა.

ამ თამაშში მომგებიანი სტრატეგია არსებობს, ანუ ისეთი ალგორითმი, რომლითაც ერთ-ერთი მოთამაშე ყოველთვის მოიგებს.
თითო კონაში ასანთების რაოდენობა x_1, x_2 და x_3 ორობით კოდში ჩავწეროთ: $x_1 = a_1a_2...a_n$, $x_2 = b_1b_2...b_n$, $x_3 = c_1c_2...c_n$. ჩვენს მაგალითში მივიღებთ:
$x_1 = 0011$, $x_2 = 1001$, $x_3 = 0101$.
შენიშვნა: x_2 ოთხი ასოსგან (ბიტისგან) შედგება, x_1 და x_3 რიცხვების ჩასაწერად კი საკმარისია 2 და შესაბამისად 3 ბიტი, მაგრამ ჩვენ სამივე რიცხვს ერთსა და იმავე სიგრძის სიტყვებად ვწერთ: თუ რაიმე ორობითი რიცხვი მოკლეა, მარცხენა მხარეს ნულების დამატებით მათი მნიშვნელობა არ იცვლება.

სამივე რიცხვს ვწერთ ერთმანეთის ქვემოთ და თითოეულ სვეტში ერთიანების რაოდენობას ვითვლით:

$$\begin{array}{cccc} a_1 & a_2 & ... & a_n \\ b_1 & b_2 & ... & b_n \\ c_1 & c_2 & ... & c_n \end{array}$$

თუ ყველა სვეტში ერთიანების რაოდენობა ლუწია, მაშინ პირველი სვლა მოწინააღმდეგეს უნდა დავუთმოთ. ამ შემთხვევაში, თუ მოწინააღმდეგე ერთი კონიდან რამოდენიმე ასანთს აიღებს, ერთიანების რაოდენობა ერთ სვეტში მაინც კენტი იქნება.
თუ ერთ-ერთ სვეტში მაინც ერთიანების რაოდენობა კენტია, ჩვენ ერთ-ერთი კონიდან იმდენი ასანთი უნდა ავიღოთ, რომ ერთიანების რაოდენობა ყველა სვეტში ლუწი გახდეს.
ზემოთ მოყვანილ მაგალითში:

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{array}$$

რადგან ერთი სვეტი მაინც არსებობს ისეთი, სადაც ერთიანების რაოდენობა კენტია, პირველი სვლა ჩვენი უნდა იყოს.
თუ მეორე კონაში (სტრიქონში) დავტოვებთ რიცხვს 0110, მაშინ ყველა სვეტში ერთიანების რაოდენობა ლუწი გახდება. ამიტომ მეორე კონაში უნდა დავტოვოთ 6 ასანთი (ანუ უნდა ავიღოთ 3).
დაგვრჩება:

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$

რამდენი ასანთიც არ უნდა აიღოს მოწინააღმდეგემ, აუცილებლად აღმოჩნდება ისეთი სვეტი, სადაც ერთიანების რაოდენობა კენტია. ვთქვათ, პირველი კონიდან მოაკლეს 2 და დაგვრჩა:

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$

ერთიანების რაოდენობა მარჯვენიდან მეორე სვეტშია კენტი. ამრიგად, თუ მეორე კონაში დავტოვებთ ოთხ ასანთს, დაგვრჩება:

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$

მოწინააღმდეგის მიერ რამოდენიმე ასანთის აღება ისევ იგივე ეფექტს გამოიწვევს: ერთ-ერთ სვეტში მაინც გაჩნდება კენტი რაოდენობის ერთიანი. ვთქვათ, მან აიღო მეორე კონიდან ყველა ასანთი:

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$

თუ ჩვენ მესამე კონაში დავტოვებთ ერთ ასანთს, ერთიანების რაოდენობა კვლავ ყველგან გალუწდება:

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}$$

მოწინააღმდეგე იძულებულია, ერთ-ერთი კონიდან დარჩენილი ერთი ასანთი აიღოს:

$$\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}$$

ბოლო სვლით ჩვენ ვიგებთ.

ამრიგად, გარკვეულ ვითარებებში სასურველია მონაცემთა ორობით კოდში ჩაწერა და შემდეგ ორობით ანბანზე შედგენილი სიტყვებით სტრატეგიის შემუშავება.

სავარჯიშო 5.24: დაწერეთ ალგორითმი, რომელიც ამ თამაშში მომგებიანი სტრატეგიით იმოქმედებს, ანუ მოცემული სამი რიცხვისათვის განსაზღვრავს, თვითონ დაიწყოს თუ არა და შემდეგ ყოველთვის მოიგებს.
თქვენი აზრით, რატომ იძლევა ეს ალგორითმი მოგების გარანტიას?

სავარჯიშო 5.25: დაამტკიცეთ, რომ თუ თამაში დამთავრებული არაა და ყველა სვეტში ერთიანების რაოდენობა ლუწია, ერთი სვლის შემდეგ აუცილებლად გაჩნდება ერთი სვეტი მაინც კენტი რაოდენობის ერთიანით.

კაზინოში მომგებიანი სტრატეგიის არარსებობის შესახებ

ახლა კი განვიხილოთ კაზინოში რულეტის მომგებიანი სტრატეგია. პირველ რიგში ვირჩევთ რომელიმე ფერს (მაგალითად, შავს) და ყოველ ჯერზე ვდებთ რაღაცა თანხას. თუ ეს ფერი მოვიდა, ვიგებთ დადებული თანხის ორმაგ რაოდენობას. თუ ჩვენი ფერი არ მოვიდა, დადებული თანხა იკარგება. იმისათვის, რომ ამ თამაშისათვის შევიმუშავოთ მომგებიანი სტრატეგია, უნდა გავითვალისწინოთ რამოდენიმე ზოგადი წესი:

1. პირველ ჯერზე ჩვენს ფერზე ვდებთ a_1 ოდენობის თანხას. ჯამში დახარჯული თანხაა a_1.
2. თუ ჩვენი ფერი მოვიდა, ვიღებთ მოგებულ თანხას $2a_1$ და ყველაფერს ვიწყებთ თავიდან.
3. თუ ჩვენი ფერი არ მოვიდა, მეორე ჯერზე ვდებთ a_2 ოდენობის თანხას. ჯამში ჩადებული თანხა იქნება a_1+a_2.
4. თუ ჩვენი ფერი მოვიდა, ვიღებთ მოგებულ თანხას $2a_2$ და ყველაფერს ვიწყებთ თავიდან.
5. თუ ჩვენი ფერი არ მოვიდა, მესამე ჯერზე ვდებთ a_3 ოდენობის თანხას. ჯამში ჩადებული თანხა იქნება $a_1 + a_2 + a_3$.

   და ასე ვაგრძელებთ მანამ, სანამ არ მოვა ჩვენი ფერი:
6. მე-n-ე ჯერზე ვდებთ a_n ოდენობის თანხას. ჯამში ჩადებული თანხა იქნება $a_1+a_2+\cdots+a_{n-1}+a_n$. თუ ჩვენი ფერი მოვიდა, მოგებული თანხა იქნება $2a_n$.
7. რადგან აქამდე ჩადებული თანხა იყო $a_1 + a_2 + \cdots + a_{n-1} + a_n$, სულ მოგებული გვექნება $2a_n - (a_1 + a_2 + \cdots + a_{n-1} + a_n) = a_n - (a_1 + a_2 + \cdots + a_{n-1})$ ოდენობის თანხა.

თუ $2a_n-(a_1+a_2+\cdots+a_{n-1}+a_n) = a_n-(a_1+a_2+\cdots+a_{n-1}) < 0$, მაშინ დახარჯული თანხა მოგებულზე მეტი იქნება, ანუ თამაშს წავაგებთ. ჩვენი ამოცანაა $a_1, a_2, ..., a_n, ...$ მიმდევრობა ისე შევარჩიოთ, რომ $a_n-(a_1+a_2+\cdots+a_{n-1}) > 0$. ერთი შესაძლებლობაა $a_i = 2^i$. ამ შემთხვევაში $a_n = 2^n$ და $(a_1 + a_2 + \cdots + a_{n-1}) = 2^n - 1$. აქედან გამომდინარე, $a_n - (a_1 + a_2 + \cdots + a_{n-1}) = 2^n - (2^n - 1) = 1$. ესე იგი, ამ სტრატეგიით (ყოველ ჯერზე დადებული თანხის გაორმაგებით) 1 ერთეულს მოვიგებთ.
თუ $(a_i)_{i=1}^{\infty}$ მიმდევრობას ისე შევარჩევთ, რომ $a_n-(a_1+a_2+\cdots+a_{n-1}) = n$, მაშინ ჩვენი ფერის მოსვლაზე ვიგებთ იმდენ თანხას, რამდენჯერაც მოგვიწია თანხის დადება.
ახლა გამოვიანგარიშოთ, თუ რა უნდა იყოს $(a_i)_{i=1}^{\infty}$ მიმდევრობა. $a_1 = 1$. a_n მოცემულია რეკურსიული ფორმულით: $a_n - (a_1 + a_2 + \cdots + a_{n-1}) = n$.

ამრიგად, ამ თამაშის მომგებიანი სტრატეგია შემდეგია:

Algorithm 5.6: მომგებიანი სტრატეგია კაზინოში

```
c_0 = 0;
i = 0;
While( არჩეული ფერი არ მოვიდა )
 {
  c_{i+1} = 2c_i + 1;
  დადე c_{i+1};
 }
```

სავარჯიშო 5.26: მათემატიკური ინდუქციით დაამტკიცეთ, რომ $a_n = 2^n - 1$ და $a_n = 2 \cdot a_{n-1} + 1$.

სავარჯიშო 5.27: დაამტკიცეთ ამ სტრატეგიის სისწორე.

სავარჯიშო 5.28: როგორ გგონიათ, რა ხერხით ახერხებს კაზინო ამ სტრატეგიისაგან თავის დაცვას?

თამაშთა თეორიაში არსებობს აქსიომა, რომლის მიხედვითაც შემთხვევით თამაშში (ანუ ისეთში, სადაც აქტუალური სვლა არაა დაოკიდებული არცერთ წინაზე - ასეთია, მაგალითად, კაზინოში თითქმის ყველა თამაში რულეტის ჩათვლით) მომგებიანი სტრატეგია ვერ იარსებებს შეზღუდული ბიუჯეტის პირობებში. ეს კი იმას ნიშნავს, რომ მომგებიანი სტრატეგია მხოლოდ შეუზღუდავი თანხების პირობებშია შესაძლებელი. რადგან ბუნებაში ყველა რესურსი (და მათ შორის ფულიც) შეზღუდულია, ასეთი სტრატეგია პრაქტიკულად არ არსებობს.

დასკვნა: კაზინოში შესული მოთამაშე ყოველთვის **წაგებულია**. მოგებულია მხოლოდ კაზინო.

5.5 ფორმალური ენა და გრამატიკა

განმარტება 5.2: თუ მოცემულია რაიმე ანბანი A, მაშინ ნებისმიერ სიმრავლეს $L \subset A^*$ ამ ანბანზე აგებული ენა ეწოდება.

ერთ-ერთ მაგალითად შეგვიძლია განვიხილოთ ქართული ენა, რომელიც ქართული ანბანის იმ სიტყვებისგან შედგება, რომლებსაც ჩვენს ენაში აზრი აქვს. ეს სასრული სიმრავლე იქნება, რომელიც ასობით ათასი სიტყვა-ფორმისაგან შედგება.
ერთ-ერთი აქტუალური ამოცანაა, მაგალითად, ბუნებრივი ენისათვის მართლწერის შემოწმება (spell checking). თუ მოცემული სიტყვისთვის იმის დადგენა გვინდა, სწორადაა იგი დაწერილი თუ არა, ერთ-ერთი საშუალება იქნება ქართული ენის სრულ ბაზაში გადამოწმება, გვხვდება თუ არა ეს სიტყვა. რა თქმა უნდა, ამის ტექნიკური რეალიზაცია შედარებით მარტივია, თუ ქართული ენის ყველა სიტყვა-ფორმისაგან შემდგარი დალაგებული სიმრავლე გვექნება. მაგრამ სიძნელე სწორედ ასეთი სიმრავლის შექმნაა, სადაც არც ერთი სიტყვა-ფორმა არ იქნება გამორჩენილი.
მეორე მეთოდი იქნებოდა ძირითადი ფორმების (ქართული სიტყვების ფუძეების) სიმრავლის შექმნა და მერე გრამატიკული წესებით სიტყვა-ფორმების აგება. მაგალითად, თუ გვაქვს მოცემული სიტყვა „ადამიანისებური”, არ იქნებოდა აუცილებელი ამ სიტყვა-ფორმის ბაზაში ჩაწერა. საკმარისია ბაზაში გვქონდეს ფუძე „ადამიან” და წესი, როგორ დაგენერირდება სიტყვა, კერძოდ [ფუძე] + „ისებური”. მაშინ სისტემა მიხვდებოდა, რომ ჩაწერილი სიტყვა შეიძლება დაგენერირდეს ზემოთ მოყვანილი წესით და მიხვდებოდა, რომ იგი უშეცდომოდაა ჩაწერილი.

სიტყვათა გენერაციის ასეთ წესს ენის (ფორმალური) გრამატიკა ეწოდება. დღეისათვის ქართული ენის ფორმალური აღწერის პრობლემა ბოლომდე გადაჭრილი არაა, რითაც ჩვენი ენის ავტომატური დამუშავების სისტემების ნაკლებობა აიხსნება.

ბუნებრივი ენების დამუშავების გარდა ფართოდ გამოიყენება ე.წ. ფორმალური ენები და მათი გრამატიკა. ფორმალური ენის მაგალითია ნებისმიერი დაპროგრამების ენა. ამას გარდა, ნებისმიერი მათემატიკური ამოცანის ამონახსნათა სიმრავლე შეგვიძლია განვიხილოთ როგორც ფორმალური ენა.

მაგალითისათვის განვიხილოთ შემდეგი ამოცანა: მოცემული $(a, b) \in \mathbb{N}^2$ რიცხვთა წყვილისთვის განსაზღვრეთ, აქვს თუ არა $a \cdot x + b = 0$ განტოლებას მთელი ამონახსნი.
ამ ამოცანის ამოხსნა სხვადასხვა გზით შეიძლება, მათ შორის სუფთა ალგებრულით. ერთ-ერთი სხვა გზა იქნებოდა ისეთი წყვილების სიმრავლის შედგენა, რომლებიც ამ ამოცანის პასუხებს შეადგენს:

$$P = \{(1,1), (1,2), (1,3), (1,4), ..., (7,7), (7,14), ..., (7,-14), ...\}$$

რადგან ეს სიმრავლე უსასრულოა, მისი ჩამოწერა შეუძლებელი იქნება. მაგრამ შესაძლებელია მისი ფორმალურად აღწერა:

$$P = \{(a,b) \mid b = k \cdot a; k \in \mathbb{Z}, a, b \in \mathbb{N}\}.$$

ზედა მაგალითში მოყვანილი ამოცანის ამოხსნა შემდეგნაირადაც შეიძლება ჩამოვაყალიბოთ: მოცემული (a, b) წყვილისათვის გაარკვიეთ, ჭეშმარიტია თუ არა გამონათქვამი $(a, b) \in P$.
როგორც ვხედავთ, ერთი და იგივე ამოცანის ჩამოყალიბება სხვადასხვანაირად შეიძლება. ზემოთ მოყვანილ მაგალითში ჩვენ ამოცანის ამონახსნათა მთელი სიმრავლე P აღვწერეთ და მერე დავსვით შეკითხვა, შედის თუ არა რიცხვთა გარკვეული წყვილი ამ სიმრავლეში. სხვა სიტყვებით რომ ვთქვათ, ამ ამოცანის მონაცემები და ამონახსნები რაღაც ენაზე აღვწერეთ და მერე ვცდილობთ იმის დადგენას, ეკუთვნის თუ არა მოცემული მონაცემი ამონახსნთა სიმრავლეს.

ისეთ ამოცანებს, რომელთა პასუხია მხოლოდ *„კი"* ან *„არა"*, ***„გადაწყვეტილების"*** ამოცანა ეწოდება. ასეთი ტიპის ამოცანებისთვის შეგვიძლია ამონახსნთა სიმრავლის შექმნა (ან აღწერა) და მერე მონაცემიდან გამომდინარე რაღაც ელემენტის ამ სიმრავლეში არსებობის გადამოწმება.

მათემატიკაში კარგადაა ცნობილი უსასრულო სიმრავლეების აღწერის მეთოდები. მაგალითად, $\{a|a = 2k, k \in \mathbb{N}\}$ (ლუწ რიცხვთა სიმრავლე), ან $\{b|b = p + 3, p$ მარტივია$\}$.
ანალოგიურად შეგვიძლია უსასრულო ენების აღწერაც. მაგალითად, ე.წ. პალინდრომების ენა შემდეგნაირად შეიძლება აღიწეროს (პალინდრომი ისეთ სიტყვას ეწოდება, რომელიც წინიდან და უკნიდან ერთნაირად იკითხება):

$$Pal_\Sigma = \{w|w = w^R, w \in \Sigma^*\}$$

შენიშვნა: ხშირად პალინდრომებს არა მარტო თითოეულ სიტყვას, არამედ ისეთ წინადადებსაც უწოდებენ, რომელთა წაკითხვა ორივე მხრიდან ერთნაირად შეიძლება (მაგალითად „აი ია"), მაგრამ ჩვენ მხოლოდ ცალკეული სიტყვებით შემოვიფარგლებით - ნებისმიერი წინადადება ხომ ისე შეგვიძლია განვიხილოთ, როგორც ცალკეული სიტყვა.

სავარჯიშო 5.29: მოიყვანეთ ქართულ, ინგლისურ და ორობით ანბანზე აგებული პალინდრომების მაგალითები (მათ შორის ისეთებიც, რომლებსაც ქართულ და ინგლისურ ენებზე აზრი აქვთ).

ერთი საკითხია რაღაც უსასრულო ენის გარკვეული სასრული მეთოდებით აღწერა და მეორე - ისეთი მექანიზმის (ალგორითმის) შექმნა, რომელიც მხოლოდ ამ ენის ელემენტებს დააგენერირებს.
ასეთ გენერატორს ამ ენის *გრამატიკას* უწოდებენ. $Pal_\mathbb{B}$ ენის გრამატიკა ორობით ანბანზე შედგენილ ენაზე შემდეგნაირად შეიძლება ჩამოვაყალიბოთ:

- ϵ, 0 და 1 პალინდრომია;
- თუ w პალინდრომია, ასევე პალინდრომია $0w0$ ან $1w1$;
- მხოლოდ ამ წესწბით შედგენილი სიტყვები (და არც ერთი სხვა) ქმნიან პალინდრომებს.

ფორმალური გრამატიკის სახით ეს შემდეგნაირად ჩაიწერება:

1. $S \to \epsilon$
2. $S \to 0$
3. $S \to 1$
4. $S \to 0S0$
5. $S \to 1S1$

ქართულად ეს წესები შემდეგნაირად შეიძლება ჩამოვაყალიბოთ:

1. ენის ახალი ელემენტი შეიძლება იყოს ცარიელი სიტყვა
2. ენის ახალი ელემენტი შეიძლება იყოს 0
3. ენის ახალი ელემენტი შეიძლება იყოს 1
4. თუ S ენის ელემენტია, ამავე ენის ელემენტი იქნება მას თუ წინიდან და უკნიდან მივუწერთ 0
5. თუ S ენის ელემენტია, ამავე ენის ელემენტი იქნება მას თუ წინიდან და უკნიდან მივუწერთ 1

ცხადია, რომ ამ წესებით შექმნილი ელემენტი პალინდრომი იქნება: პირველი სამი წესით პირდაპირ შეიქმნება სიტყვა, რომელიც აშკარად პალინდრომია (ცარიელი სიტყვა პალინდრომია: მასში არ მოიძებნება ისეთი პოზიცია i, რომ $\epsilon[i] \neq \epsilon\{i\}$). ამას გარდა, თუ P პალინდრომია, ასევე პალინდრომი იქნება $0P0$ და $1P1$ (წესები 4 და 5).
მაგრამ საიდან ვიცით, რომ ეს გრამატიკა ყველა პალინდრომს წარმოშობს, ანუ არ არსებობს ისეთი პალინდრომი, რომელიც ამ წესებით არ წარმოიშვება? ამის ჩვენება მათემატიკური ინდუქციით შეიძლება: თუ w სიტყვა პალინდრომია და $|w| = 0$, იგი ცარიელია და ზედა გრამატიკის პირველი წესით გენერირდება; თუ $|w| = 1$, მაშინ ან $w = 0$ ან $w = 1$ და იგი გენერირდება ან მეორე, ან მესამე წესით.
ახლა დავუშვათ, რომ ყველა პალინდრომი, რომლის სიგრძე $\leq n$ ამ გრამატიკით წარმოიშვება და განვიხილოთ რაღაც პალინდრომი w, სადაც $|w| = n + 1$. ცხადია, რომ ან $w = 1u1$ ან $0u0$, სადაც u პალინდრომია და $|u| \leq n$.

ინდუქციის დაშვების თანახმად, u სიტყვა ზედა გრამატიკით გენერირდება და, აქედან გამომდინარე, w სიტყვაც იგივე გრამატიკის მეოთხე ან მეხუთე წესით წარმოიშვება.

განვიხილოთ კიდევ ერთი მაგალითი: $L = \{0^n1^n|n \in \mathbb{N}_0\}$ ენის აღმწერი გრამატიკა შემდეგნაირად შეიძლება ჩავწეროთ:

1. $S \rightarrow \epsilon$;

2. $S \rightarrow 0S1$

როგორც მოყვანილი მაგალითებიდან შეიძლება დავასკვნათ, გრამატიკის წესები გარკვეული ფორმით შეიძლება ჩაიწეროს. S ე.წ. „საწყისი სიმბოლოა”: პირველი წესი გვეუბნება, თუ როგორ იწყება სიტყვის წარმოება. აქ 0,1 და ϵ ე.წ. ტერმინალებია: ცარიელი სიტყვა და ანბანის ელემენტები. ტერმინალები იმიტომ ეწოდებათ, რომ მათი შემდგომი „გაშლის” წესები აღარ არსებობს: სიმბოლო დაფიქსირდება და ყოველი შემდგომი გამოთვლისას უცვლელი რჩება, განსხვავებით ე.წ. „არატერმინალისგან”, ჩვენს შემთხვევაში სიმბოლო S, რომელიც ყოველი გამოთვლისას „გაიშლება”. ასე, მაგალითად, სიტყვა 00001111 შემდეგნაირად გაიშლება: საწყისი არატერმინალი S მეორე წესის თანახმად გარდაიქმნება სიტყვად $0S1$, მერე ისევ მეორე წესის თანახმად იგივე გარდაქმნით $00S11$, შემდეგ $000S111$, შემდეგ $0000S1111$ და ბოლოს - პირველი წესის თანახმად - $0000\,\epsilon\,1111 = 00001111$.

ზოგადად, გრამატიკა შედგება Σ ანბანისგან (რომელსაც გრამატიკაში ტერმინალების სიმრავლეს უწოდებენ), N არატერმინალური სიმბოლოებისგან , S საწყისი არატერმინალისგან და P წესების სიმრავლისაგან. ჩვენს მაგალითში $\Sigma = \mathbb{B}$, $N = \{S\}$, S ჩვენი საწყისი არატერმინალის სიმბოლოა და P წესები ზემოთ იყო ჩამოთვლილი.

<u>სავარჯიშო 5.30:</u> აღწერეთ გრამატიკა, რომელიც „სწორად” დასმულ ფრჩხილების ენას აგენერირებს (მაგალითად, სწორადაა დასმული ფრჩხილები „(()(()))” სიტყვაში, მაგრამ არასწორად სიტყვაში „(()))((())”.

აღსანიშნავია, რომ ზოგადად გრამატიკაში რამოდენიმე არატერმინალური სიმბოლო შეიძლება არსებობდეს. ჩვენს მიერ განხილული გრამატიკები ე.წ. „უკონტექსტო” ენებს აღწერს და ამიტომ მათაც უკონტექსტო გრამატიკა ეწოდებათ. ასეთი ტიპის გრამატიკაში წესების მარცხენა მხარეს მხოლოდ ერთი არატერმინალური სიმბოლო დგას (აქედან გამომდინარეობს სახელი „უკონტექსტო” - რადგან მხოლოდ ერთი არატერმინალური სიმბოლო „იშლება”, შინაარსს ანუ კონტექსტს მნიშვნელობა არ აქვს).
მნიშვნელოვანი ფაქტია ის, რომ უკონტექსტო გრამატიკას ყველა ენის აღწერა არ შეუძლია. მაგალითად, ენა $\{a^nb^nc^n|n \in \mathbb{N}_0\}$ ამ ტიპის გრამატიკით ვერ აღიწერება. და ზოგადად: არ არსებობს ისეთი გრამატიკა, რომელიც ყველა ენას აღწერს. უფრო მეტიც: დამტკიცებულია, რომ არსებობს ისეთი ენა, რომელიც ვერც ერთი გრამატიკით ვერ აღიწერება.

ასეთ საკითხებს თეორიული ინფორმატიკა შეისწავლის, რასაც ჩვენ მომდევნო კურსების ფარგლებში დაწვრილებით განვიხილავთ.

5.6 ფორმალური ენებისა და გრამატიკის გამოყენების საშუალებები

როგორც აქამდე აღინიშნა, ყოველი გადაწყვეტილების ამოცანა შეიძლება განვიხილოთ, როგორც რაიმე ანბანზე აგებული ენა, რომელიც ამ ამოცანის ამონახსნების სიმრავლეს ემთხვევა.

მაგალითისთვის განვიხილოთ შემდეგი ამოცანა:

მოცემულია მთელ რიცხვთა წყვილი (x, y). არის თუ არა იგი $28x-12y = 0$ ორცვლადიანი განტოლების ამონახსნი?

ცხადია, ეს გადაწყვეტილების ამოცანაა: მისი ყოველი პასუხი იქნება ან „კი”, ან „არა”. მისი ენა შედგება ყველა ამონახსნისგან:

$$L = \left\{(a, b)|a, b \in \mathbb{Z}, \frac{a}{b} = \frac{3}{7}\right\}$$

აქედან გამომდინარე, გადაწყვეტილების ამონახსნის ამოხსნა შეიძლება განვიხილოთ როგორც მისი შესაბამისი ენაში (ამონახსნთა სიმრავლეში) მონაცემის ძებნად. თუ ეს ენა აღიწერება რაიმე გრამატიკით, მისი შესაბამისი ალგორითმიც ამ გრამატიკის შესაბამისი ავტომატია: დამტკიცებულია, რომ ყველა გრამატიკას შეესაბამება

გარკვეული ავტომატი (ალგორითმი), რომელიც ამ გრამატიკის სიმულაციას ახდენს და გაარკვევს, შედის თუ არა მონაცემი შესაბამის ენაში.

ენასა და გრამატიკას კიდევ ერთი დიდი გამოყენება აქვს კომპილატორებში, მათ აგებასა და, ზოგადად, თეორიაში. ყოველი პროგრამა, რომელიც იწერება რაიმე დაპროგრამების ენაზე (ადამიანისთვის გასაგებ, ე.წ. მაღალი დონის ენაზე), უნდა იქნას გადათარგმნილი მანქანურ კოდებში (ე.წ. დაბალი დონის ენაზე), რაც ადამიანისთვის საკმაოდ მონოტონური, მოსაწყენი და შეცდომებით აღსავსე პროცესი იქნებოდა. ამიტომ ხდება პროგრამების ავტომატური „კომპილაცია", ანუ გადათარგმნა. აქ ფართოდ იყენებენ ენათა ფორმალურ აღწერასა და გრამატიკის საფუძველზე შესაბამის ალგორითმებს, მაგალითად, სინტაქსური (მართლწერის) ან სემანტიკური (შინაარსობრივი) შეცდომების აღმოსაჩენად, ან ერთი ენის სტრუქტურის მეორე ენის სტრუქტურაზე გადასატანად, მხოლოდ აქ არ შემოიფარგლებიან მხოლოდ უკონტექსტო გრამატიკებით, რომელთა მაგალითებიც ჩვენ განვიხილეთ, არამედ იყენებენ უფრო რთულ სტრუქტურებს, როგორიცაა კონტექსტური გრამატიკა, ატრიბუტული გრამატიკა და სხვა.

5.7 მოკლე დასკვნა

მეხუთე თავში ჩვენ განვიხილეთ მონაცემთა კოდირების მეთოდები და ვნახეთ, თუ როგორ შეიძლება მათი გადატანა სხვადასხვა ფუძის ათვლის სისტემებში, განსაკუთრებით კი ორობით სისტემაში.
შემდეგ ვნახეთ, თუ როგორ შეიძლება მოდულარული არითმეტიკის გამოყენებით უსასრულო სისტემების სიმულაცია სასრულით მინიმალური დანაკარგების გათვალისწინებით და ვაჩვენეთ, როგორ იგება ორობითი რიცხვების არითმეტიკა, რის შემდეგაც ორობითი კოდების გამოყენების მაგალითები განვიხილეთ.
ბოლოს განვმარტეთ ფორმალური ენა, მისი შესაბამისი გრამატიკა და მნიშვნელოვანი გამოყენების მაგალითები.

თავი 6

მიმართებები და დალაგება

6.1 მიმართებები

განვიხილოთ საქართველოს მოქალაქეთა სიმრავლე $A = \{w \,|\, w$ საქართველოს მოქალაქეა$\}$. რა თქმა უნდა, ამ სიმრავლეში ისეთი ადამიანების ქვესიმრავლეები იქნება, რომლებიც ერთმანეთთან მეგობრობენ. თუ $a, b \in A$ და a მეგობრობს b-თან, მაშინ b მეგობრობს a-თან. იმის აღსანიშნავად, რომ a და b მეგობრობენ, შეგვიძლია დავწეროთ: (a, b). თუ ჩამოვწერთ ყველა ასეთი მეგობრების წყვილებს, მივიღებთ რაღაცა სიმრავლეს $R = \{(a, b) \,|\, a, b \in A$, a და b ერთმანეთთან მეგობრობენ$\}$.
როგორც ვხედავთ, R სიმრავლე A სიმრავლეში რაღაცა კავშირის, ანუ მიმართების აღმნიშვნელია და მის ელემენტებზე გარკვეულ ინფორმაციას გვაწვდის. ადვილი შესამჩნევია, რომ $(a, b) \in R \Leftrightarrow (b, a) \in R$.
ახლა კი განვიხილოთ იგივე სიმრავლე A და მასზე განსაზღვრული მიმართება $R_1 = \{(a, b) \,|\, a, b \in A$, b არის a-ს წინაპარი$\}$. ცხადია, რომ თუ $(a, b) \in R_1 \Rightarrow (b, a) \notin R_1$. ეს R_1 სიმრავლე სხვა ტიპის მიმართების ინფორმაციას გვაწვდის A სიმრავლის ელემენტების შესახებ და R სიმრავლისგან რადიკალურად განსხვავდება, თუ მათ ზემოთ ნახსენებ თვისებებს გავითვალისწინებთ.

ახლა კი გადავიდეთ მიმართების ფორმალურ განსაზღვრებაზე, როსთვისაც პირველ რიგში საჭიროა

განმარტება 6.1: თუ მოცემულია ნებისმიერი ორი სიმრავლე A და B, მაშინ $A \times B = \{(a, b) \,|\, a \in A, b \in B\}$ A და B სიმრავლეების **„დეკარტული ნამრავლი"** ეწოდება.

მაგალითად, თუ $A = \{1, 2, 3, 4\}$ და $B = \{a, b, c\}$, მაშინ

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c), (4, a), (4, b), (4, c)\}.$$

აღსანიშნავია, რომ აქ მნიშვნელობა აქვს ელემენტების თანმიმდევრობას: პირველ ადგილზეა A სიმრავლის ელემენტი, ხოლო მეორეზე კი - B სიმრავლისა.

ცხადია, რომ $A \times A$ სიმრავლეც A სიმრავლის თავის თავთან დეკარტული ნამრავლია.

მაგალითად, თუ $A = \{a_1, a_2, a_3\}$, $A \times A = \{(a_1, a_1), (a_1, a_2), (a_1, a_3), (a_2, a_1), (a_2, a_2), (a_2, a_3), (a_3, a_1), (a_3, a_2), (a_3, a_3)\}$.

სავარჯიშო 6.1: განვიხილოთ შემდეგი ამოცანა: მოცემულია $n \in \mathbb{N}$. შეადგინეთ $A \times A$, სადაც $A = \{a_1, a_2, ..., a_n\}$. რა არის ამ ამოცანის მონაცემი? რა უნდა იყოს მისი შედეგი? დაწერეთ ალგორითმი, რომელიც ამ ამოცანას გადაჭრის.

განმარტება 6.2: თუ მოცემულია რაიმე ორი სიმრავლე A და B (ხშირად $A = B$), მაშინ $R \subset A \times B$ სიმრავლეს A და B სიმრავლეებზე განსაზღვრული მიმართება ეწოდება.

მაგალითად, ზემოთ განსაზღვრული R (მეგობრობის აღმნიშვნელი) და R_1 (წინაპრების აღმნიშვნელი) სიმრავლეები შესაბამისი A სიმრავლის სხვადასხვა მიმართების (დამოკიდებულებების) განმსაზღვრელია.

ორ სხვადასხვა სიმრავლეზე განსაზღვრული მიმართების მაგალითიად შეგვიძლია მოვიყვანოთ საქართველოს რეგიონებისა და ქალაქების სიმრავლეები:

$A = \{$ქართლი, კახეთი, რაჭა, იმერეთი, სამეგრელო$\}$ და $B = \{$ოზურგეთი, ონი, ფოთი, აგარა, ზუგდიდი, ვანი, თელავი, გურჯაანი, ქუთაისი$\}$.
მიმართება, რომელიც თითოეულ რეგიონს მასში არსებულ ქალაქს დაუკავშირებს, იქნება:

$$R_2 = \{ \text{(ქართლი, აგარა), (კახეთი, თელავი), (კახეთი, გურჯაანი), (რაჭა, ონი), (იმერეთი, ვანი),} \\ \text{(იმერეთი, ქუთაისი), (სამეგრელო, ფოთი), (სამეგრელო, ზუგდიდი)} \}.$$

განმარტება 6.3: ნებისმიერ $R \subset A \times B$ (ხშირ შემთხვევაში $A = B$) მიმართებას შეიძლება ქონდეს შემდეგი თვისებები::

- თუ $\forall a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R$ ან $(a_2, a_1) \in R$, მაშინ R მიმართებას სრული ეწოდება;
- თუ $\forall a \in A$, $(a, a) \in R \subset A \times B$, მაშინ R მიმართებას რეფლექსური ეწოდება;
- თუ $\forall a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R \Leftrightarrow (a_2, a_1) \in R$, მაშინ R მიმართებას სიმეტრიული ეწოდება;
- თუ $\forall a_1 \in A, a_2 \in B$, $a_1 \neq a_2$, $(a_1, a_2) \in R \Rightarrow (a_2, a_1) \notin R$, მაშინ R მიმართებას ანტისიმეტრიული ეწოდება;
- თუ $\forall a_1, a_2, a_3 \in A$, $a_1 \neq a_2$, $a_3 \neq a_2$, $\big((a_1, a_2) \in R, (a_2, a_3) \in R\big) \Rightarrow (a_1, a_3) \in R$, მაშინ R მიმართებას ტრანზიტული ეწოდება.

მაგალითად, ზემოთ განსაზღვრული R (მეგობრობის აღმნიშვნელი) მიმართება სიმეტრიულია, მაგრამ არ არის სრული, რადგან შეიძლება მოიძებნოს ორი ისეთი ადამიანი $a, b \in A$, რომელიც ერთმანეთთან არ მეგობრობს და ამიტომ $(a, b) \notin R$.
მეორე მიმართება R_1 (წინაპრების განმსაზღვრელი) ტრანზიტულია: თუ a-ს წინაპარია b ($(a, b) \in R_1$) და b-ს წინაპარია c ($(b, c) \in R_1$), a-ს წინაპარია c ანუ $(a, c) \in R_1$.
ეს მიმართება ასევე ანტისიმეტრიულია: თუ $(a, b) \in R_1 \Leftrightarrow (b, a) \notin R_1$.
მესამე მიმართება R_2 ანტისიმეტრიული და არასრულია: R_2 არ შეიცავს არც ერთ წყვილს, რომელშიც შედის ოზურგეთი.

სავარჯიშო 6.2: დაამტკიცეთ, რომ მიმართება $R_3 \subset \mathbb{N} \times \mathbb{N}$, $R_3 = \{(a, b) | a \leq b\}$ რეფლექსური და სრულია.

სავარჯიშო 6.3: დაწერეთ, რისი ტოლია შემდეგი სიმრავლეები:

(ა) $\{1\} \times \{1, 2\} \times \{1, 2, 3\}$;

(ბ) $\emptyset \times \{1, 2, 3\}$;

(გ) $2^{\{1,2\}}$, რაც არის $\{1, 2\}$ სიმრავლის ყველა შესაძლო ქვესიმრავლის სიმრავლე;

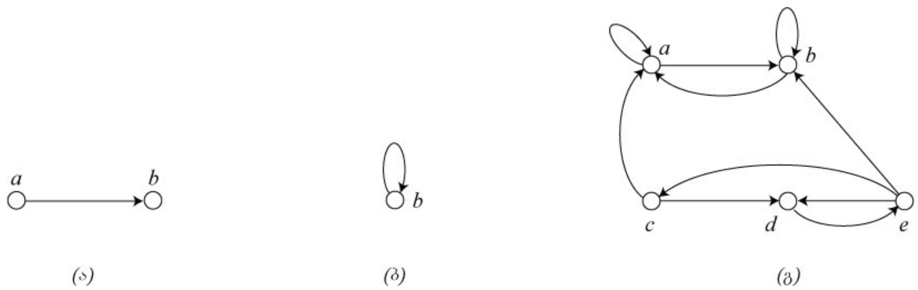
(დ) $2^{\{1,2\}} \times \{1, 2\}$.

თვალსაჩინოებისათვის პატარა სიმრავლეებზე მიმართებები გრაფიკულად შეიძლება გამოვსახოთ: თუ A სიმრავლეზე განსაზღვრულია რაიმე მიმართება R და $(a, b) \in R$, მაშინ a და b ელემენტები გამოისახება რგოლებად, ხოლო $(a, b) \in R$ კი a ელემენტიდან b ელემენტში მიმართული ისრით (ნახ. 6.1 (ა)). თუ $(b, b) \in R$, ეს გრაფიკულად b ელემენტის შესაბამისი რგოლიდან გამომავალი და იგივე რგოლში შემავალი ისრით გამოიხატება (ნახ. 6.1 (ბ)). თუ $A = \{a, b, c, d, e\}$, მაშინ $R = \{(a, a), (a, b), (b, b), (b, a)(c, a), (c, d), (d, e), (e, b), (e, c), (e, d)\}$ ისე შეიძლება წარმოვადგინოთ, როგორც ნახ. 6.1 (გ)) -ში.

განმარტება 6.4: რეფლექსურ, სიმეტრიულ და ტრანზიტულ მიმართებას „ტოლობის მიმართება” ან „ეკვივალენტურობის მიმართება” ეწოდება.

მაგალითად, თუ მოცემულია ცოცხალ ორგანიზმთა სიმრავლე A, მაშინ $R' = \{(a, b) | a$ და b ორივე ხერხემლიანია$\}$ ეკვივალენტურობის მიმართებაა, რადგან იგი რეფლექსური, სიმეტრიული და ტრანზიტულია.

სავარჯიშო 6.4: დაამტკიცეთ, რომ ზემოთ მოყვანილი მიმართება R' მართლაც რეფლექსური, სიმეტრიული და ტრანზიტულია.



ნახ. 6.1: მიმართებათა გრაფიკული წარმოდგენა

ეკვივალენტურობის მიმართება სიმრავლეს ე.წ. „ეკვივალენტურობის კლასებად" ყოფს, ანუ ისეთ ქვესიმრავლეებად, სადაც ერთმანეთის ეკვივალენტური (ანუ გარკვეული თვალსაზრისით მსგავსი) ელემენტები შედის. სხვა სიტყვებით რომ ვთქვათ, თუ რაიმე $A \neq \emptyset$ სიმრავლეზე განსაზღვრულია ეკვივალენტურობის მიმართება R, იგი განსაზღვრავს A სიმრავლის ისეთ ქვესიმრავლეებს $B_\alpha \subset A$, რომ $B_\alpha = \{a, b \in A \,|\, (a, b) \in R\}$ (ამ ქვესიმრავლეებში მხოლოდ ისეთი ელემენტები შედის, რომლებიც R მიმართების განსაზღვრებით ერთმანეთის „ეკვივალენტურია").

მაგალითად, თუ მოცემულია მიმართება $R = \{(a, b)|\, a$ და b ორივე ლუწია ან a და b ორივე კენტია$\}$, იგი ნატურალურ რიცხვთა $\mathbb{N}$ სიმრავლეში ორ ქვესიმრავლეს გამოყოფს - ლუწ და კენტ რიცხვთა ქვესიმრავლეებს (კლასებს): $N_1 = \{a_i|(a_k, a_l) \in R$ და ორივე ლუწია$\}$, $N_2 = \{a_i|(a_k, a_l) \in R$ და ორივე კენტია$\}$.

თუ მოცემულია $A = \{0, 1, 2, 3, 4, 5, 6, 7\}$, მაშინ მიმართება $R = \{(a, b) \in A \times A \,|\, a$ და b ორივე ლუწია ან a და b ორივე კენტია$\}$ გრაფიკულად შემდეგნაირად შეიძლება გამოისახოს:

ნახ. 6.2: სიმრავლის ორ დამოუკიდებელ კლასად დაყოფის მაგალითი

ადვილი დასანახია, რომ R მიმართება A სიმრავლეში ორ დამოუკიდებელ კლასს (ქვესიმრავლეს) გამოჰყოფს.

აღსანიშნავია, რომ ეს ერთმანეთის ეკვივალენტური ანუ ტოლი ელემენტები მოცემული მიმართებითაა განსაზღვრული. სხვა მიმართებას შეიძლება სხვა ეკვივალენტური ელემენტები გამოეყო. ამის მაგალითია იგივე A სიმრავლეზე განსაზღვრული $R' = \{(a, b)\,|\, a$ და b ორივე იყოფა 3-ზე ან a და b რიცხვიდან ერთი მაინც არ იყოფა 3-ზე$\}$.

სავარჯიშო 6.5: გრაფიკულად გამოხატეთ ბოლოს მოცემული მიმართება R' ისე, როგორც ეს წინა მაგალითში მოხდა.

რაიმე A სიმრავლის **ეკვივალენტურობის კლასები** შემდეგნაირად აღინიშნება: $[a] = \{b \,|\, (a, b) \in R\}$, სადაც R არის A სიმრავლის ეკვივალენტურობის მიმართება.
მაგალითად, თუ $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ და $R' = \{(a, b)\,|\, a$ და b ორივე იყოფა 3-ზე ან a და b ორივე არ იყოფა 3-ზე$\}$, $[6] = \{0, 3, 6, 9\}$ და $[2] = \{1, 2, 4, 5, 7, 8\}$.

სავარჯიშო 6.6: მოიყვანეთ ნატურალურ რიცხვთა სიმრავლეზე განსაზღვრული ეკვივალენტურობის მიმართების მაგალითი, რომელიც სამ ქვესიმრავლეს გამოჰყოფს. თითოეულ ასეთ კლასში ერთმანეთის ეკვივალენტური ელემენტებია.

ახლა კი განვიხილოთ ორი ნატურალური რიცხვი, რომელიც ათობით ანბანშია ჩაწერილი: 307 და 509. ჩვენ ვიცით, რომ $307 < 509$. ამ ორი რიცხვის ასეთი მიმართება სადღაც უნდა იყოს განსაზღვრული (ანალოგიურად

ჩვენ შეგვეძლო განგვესაზღვრა $509 < 307$. ჩვენ ვიცით, რომ $0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$. მაგრამ ამ ციფრების ასეთი მიმართება ცხადი არაა, ესეც ვიღაცის მიერაა დადგენილი და შემდეგ საყოველთაოდ მიღებული. ამრიგად, გვაქვს შემდეგი მიმართება $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ სიმრავლეზე:

$$
\begin{aligned}
R \ = \{ & \\
& (0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(0,9)\\
& (1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9)\\
& (2,3),(2,4),(2,5),(2,6),(2,7),(2,8),(2,9)\\
& (3,4),(3,5),(3,6),(3,7),(3,8),(3,9)\\
& (4,5),(4,6),(4,7),(4,8),(4,9)\\
& (5,6),(5,7),(5,8),(5,9)\\
& (6,7),(6,8),(6,9)\\
& (7,8),(7,9)\\
& (8,9)\\
\} &
\end{aligned}
$$

ეს მიმართება ე.წ. „დალაგებას" განსაზღვრავს, ანუ გვაძლევს იმის წესს, თუ როგორ შეიძლება დავალაგოთ სიმრავლის ელემენტები ზრდადობის მიხედვით.

განმარტება 6.5: სრულ, ანტისიმეტრიულ და ტრანზიტულ მიმართებას დალაგება ეწოდება. არასრულ, ანტისიმეტრიულ და ტრანზიტულ მიმართებას ნაწილობრივი დალაგება ეწოდება.

სავარჯიშო 6.7: დაამტკიცეთ, რომ ბოლოს მოყვანილი მიმართება R დალაგებაა.

სავარჯიშო 6.8: მოიყვანეთ ზემოთ განსაზღვრულ A სიმრავლეზე ნაწილობრივი დალაგების მაგალითი.

თუ $(a, b) \in R$ და R დალაგებაა, მაშინ ვწერთ: $a < b$.

თუ გვაქვს მოცემული დალაგება ზემოთ მოყვანილ ანბანზე A, ადვილად შეიძლება A^* სიმრავლის სიტყვების დალაგებაც შემდეგი ალგორითმით:

ალგორითმი 6.1: $C(w, v)$
მოცემულია: $w = (w_1, w_2, ..., w_n), v = (v_1, v_2, ..., v_m) \in A^*$

თუ $|w| = |v| = 0$, მაშინ $w = v$ და ალგორითმი დაასრულე.
თუ $w(1) < v(1)$, მაშინ $(w, v) \in R$ (ან, რაც იგივეა, $w < v$) და ალგორითმი დაასრულე.
თუ $v(1) < w(1)$, მაშინ $(v, w) \in R$ (ან, რაც იგივეა, $v < w$) და ალგორითმი დაასრულე.
ჩაატარე $C(w\{|w| - 1\}, v\{|v| - 1\})$ (იგივე ალგორითმი w და v სიტყვების სუფიქსებისათვის).

აუცილებლად გასათვალისცინებელია, რომ $\epsilon < a, \forall a \neq \epsilon \in A$.

სავარჯიშო 6.9: სიტყვიერად ახსენით, თუ რას ნიშნავს ზედა ალგორითმში მოყვანილი მათემატიკური ჩანაწერები „თუ $w(|w|) < v(|v|)$, მაშინ..." და „$C(w\{|w| - 1\}, v\{|v| - 1\})$".

სავარჯიშო 6.10: დაამტკიცეთ ამ ალგორითმის სისწორე. გამოითვალეთ მისი ბიჯების რაოდენობა, თუ $|w| = |v|$ და შემდეგ თუ $|w| \neq |v|$.

სავარჯიშო 6.11: დაწერეთ, თუ რისი ტოლია ზემოთ მოყვანილ A სიმრავლეზე განსაზღვრული დალაგების მიმართება, რომლის მიხედვითაც $1 \leq 3 \leq 2 \leq 5 \leq 8 \leq 4 \leq 0 \leq 9 \leq 7 \leq 6$.

სავარჯიშო 6.12: მოიყვანეთ A სიმრავლეზე განსაზღვრული ორი სხვადასხვა ნაწილობრივი დალაგების მაგალითი. არის თუ არა $R = \emptyset$ ამ სიმრავლის ნაწილობრივი დალაგება?

სავარჯიშო 6.13: არის თუ არარაიმე A სიმრავლეზე განსაზღვრული $R = \emptyset$ მიმართება ტრანზიტული?

სავარჯიშო 6.14: არსებობს თუ არარაიმე A სიმრავლეზე განსაზღვრული $R \neq \emptyset$ მიმართება, რომელიც ერთდროულად სიმეტრიულიცაა და ანტისიმეტრიულიც?

სავარჯიშო 6.15: დაამტკიცეთ, რომ თუ R_1 და R_2 რაღაცა სიმრავლეზე განსაზღვრული ნაწილობრივი დალაგებებია, მაშინ $R_1 \cap R_2$ იგივე სიმრავლეზე განსაზღვრული ნაწილობრივი დალაგებაა.

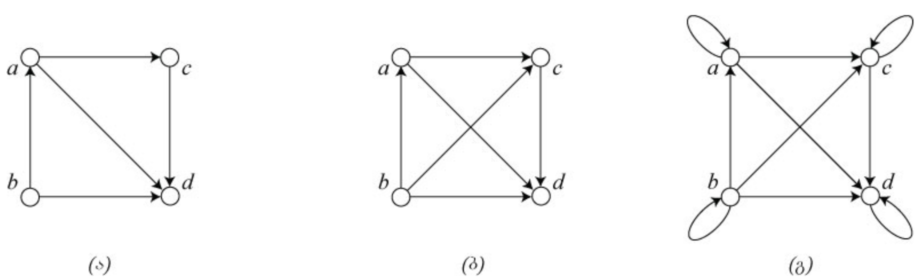
სავარჯიშო 6.16: მოცემულია ნებისმიერი სიმრავლე S, რომელიც თავის მხრივ რაღაცა სიმრავლეებისაგან შედგება. დაამტკიცეთ, რომ $R_S = \{(A,B) \,|\, A, B \in S, A \subseteq B\}$ ნაწილობრივი დალაგებაა.

სავარჯიშო 6.17: დავუშვათ, $S = 2^{\{1,2,3\}}$, რაც არის $\{1,2,3\}$ სიმრავლის ყველა შესაძლო ქვესიმრავლის სიმრავლე. ამოწერეთ ამ სიმრავლის ყველა ელემენტი და დიაგრამის სახით გამოსახეთ წინა სავარჯიშოში განსაზღვრული მიმართება R_S, რომელიც ამ სიმრავლეზეა განსაზღვრული. ცალკე ამოწერეთ S სიმრავლის მინიმალური ელემენტები, ანუ ისეთი ელემენტები a_i, რომელთათვისაც $(a_i, b) \in R_S, \forall b \in S$.

სავარჯიშო 6.18: როგორ განისაზღვრება ნებისმიერი A სიმრავლის რაღაცა R დალაგების შედეგად მიღებული მაქსიმალური ელემენტები?

ახლა კი განვიხილოთ ნახ. 6.3 (ა) -ში მოყვანილი მიმართება. ადვილი საჩვენებელია, რომ იგი არც რეფლექსური და არც ტრანზიტულია.

სავარჯიშო 6.19: აჩვენეთ, რომ ნახ. 6.3 (ა) -ში მოყვანილი მიმართება არც რეფლექსური და არც ტრანზიტულია.



ნახ. 6.3: მიმართების რეფლექსური და ტრანზიტული ჩაკეტვა

ამ მიმართების სიმრავლისათვის რამოდენიმე ახალი წყვილის (ან გრაფიკულად ისრის) ჩამატებით შეიძლება მივიღოთ ტრანზიტული მიმართება (ნახ. 6.3 (ბ). დამატებით ყველა a ელემენტისათვის (a,a) წყვილის დამატებით კი ეს მიმართება რეფლექსურიც ხდება (ნახ. 6.3 (გ)).

ანალოგიური პროცედურა - დამატებითი წყვილებით გაფართოვება ისე, რომ ნებისმიერი მიმართება ტრანზიტული და რეფლექსური გახდეს, შეიძლება ნებისმიერ მიმართებაზე ჩავატაროთ. მიღებულ მიმართებას საწყისი მიმართების რეფლექსური და ტრანზიტული ჩაკეტვა ეწოდება.

განმარტება 6.6: ნებისმიერი R მიმართების ტრანზიტული და რეფლექსური ჩაკეტვა R^* ეწოდება ისეთ რეფლექსურ და ტრანზიტულ მიმართებას, რომლისთვისაც $R \subset R^*$ და R^* სიმრავლის ელემენტების რაოდენობა მინიმალურია იმ სიმრავლეების ელემენტების რაოდენობათა შორის, რომლებიც R მიმართებას ქვესიმრავლედ შეიცავენ, ანუ R^* სიმრავლე R სიმრავლიდან რაც შეიძლება ცოტა წყვილის დამატებით უნდა მიიღებოდეს.

სავარჯიშო 6.20: დაწერეთ ალგორითმი, რომელიც ნებისმიერი A სასრული სიმრავლის რაიმე R მიმართებისათვის მის რეფლექსურ და ტრანზიტულ ჩაკეტვას გამოიანგარიშებს (ანუ შეადგენს შესაბამის სიმრავლეს). დაამტკიცეთ მისი სისწორე და გამოიანგარიშეთ ბიჯების რაოდენობა, თუ $|A| = n$.

6.2 დალაგებისა და ეკვივალენტურობის გამოყენების მაგალითები: ძებნა, ოპერაციები სიმრავლეებზე და ნაშთთა კლასები

დალაგება და ნაწილობრივი დალაგება ცენტრალურ როლს თამაშობს ინფორმატიკაში, რადგან ამოცანათა უდიდესი ნაწილი მონაცემთა რაღაცა წესის მიხედვით დალაგების შედეგად საკმაოდ მარტივდება.

ამის მაგალითია ქართულ ანბანზე Q შემოტანილი დალაგება ა < ბ < გ < დ < $\cdots$ < ჯ < ჰ. თუ ჩვენ ამის საფუძველზე ქართულ სიტყვებსაც დავალაგებთ (ანუ შემოვიტანთ დალაგების წესს Q^* სიმრავლეზე), ქართულ ლექსიკონში რაიმე მოცემული w სიტყვის მოძებნა გაადვილდება: ლექსიკონს გადავშლით შუაში და ამოვიკითხავთ პირველივე სიტყვას v. თუ $w = v$, სიტყვა მოძებნილია. თუ ჩვენი საძებნი სიტყვა ამ სიტყვის წინაა (ანუ $w < v$), მაშინ იგივე ოპერაციას გავიმეორებთ ლექსიკონის პირველ ნახევარში (თუ $v < w$, ვიღებთ მეორე ნაწილს): გადავშლით ამ ნაწილის შუაში და ანალოგიურ პროცედურას გავიმეორებთ.

სავარჯიშო 6.21: დაწერეთ ალგორითმი, რომელიც ქართულ ანბანზე განსაზღვრული ორი სიტყვისათვის w და v განსაზღვრავს, $w = v$ თუ $w < v$ თუ $v < w$.
შენიშვნა: ეს ალგორითმი ათობითში ჩაწერილი რიცხვების შედარების ალგორითმის მსგავსია.

სავარჯიშო 6.22: დაამტკიცეთ წინა სავარჯიშოში მოყვანილი ალგორითმის სისწორე და გამოითვალეთ მისი ბიჯების რაოდენობა, თუ $|w| = n$ და $|v| = m$.

ზოგადად, თუ მოცემულია რაიმე A ანბანი და $S = \{u_1, u_2, ..., u_n \in A^*\}$ სიტყვათა დალაგებული სიმრავლე, მოცემული w სიტყვის მოძებნა ამ სიმრავლეში შეიძლება შემდეგი ალგორითმით:

ალგორითმი $L(S, w)$

მოცემულია: $S = \{u_1, u_2, ..., u_n \in A^*\}$ სიტყვათა სიმრავლე და რაღაცა სიტყვა w.

შედეგი: ვიპოვნოთ ისეთი $u_i \in S$, რომ $u_i = w$.

- თუ $S = \emptyset$, მაშინ დაბეჭდე: „სიტყვა სიმრავლეში არ მოიძებნა" და ალგორითმი დაასრულე.
- თუ $u_{\lfloor \frac{|S|}{2} \rfloor} = w$, მაშინ დაბეჭდე: „$\lfloor \frac{|S|}{2} \rfloor$-ური ელემენტია w" და ალგორითმი დაასრულე.
- თუ $u_{\lfloor \frac{|S|}{2} \rfloor} < w$, მაშინ ჩაატარე $L(\{u_{\lfloor \frac{|S|}{2} \rfloor + 1}, ..., u_n\}, w)$.
- თუ $u_{\lfloor \frac{|S|}{2} \rfloor} > w$, მაშინ ჩაატარე $L(\{u_1, ..., u_{\lfloor \frac{|S|}{2} \rfloor}\}, w)$.

სავარჯიშო 6.23: ინდუქციის გამოყენებით დაამტკიცეთ ამ ალგორითმის სისწორე. გამოითვალეთ მისი ბიჯების რაოდენობა, თუ $|S| = n$.

დალაგების გამოყენება შეიძლება ასევე მათემატიკური საკითხების გადაწყვეტისას. თუ მოცემული გვაქვს რაიმე ფუნქციის მნიშვნელობები გარკვეულ წერტილებში, მისი მინიმუმის ან მაქსიმუმის პოვნა, ცხადია, დალაგებულ სიმრავლეში ელემენტარულია (განსხვავებით დაულაგებელი სიმრავლისაგან).
ამას გარდა, სიმრავლეთა თანაკვეთა, გაერთიანება, დამატება და მრავალი სხვა ოპერაცია დალაგებულ სიმრავლეებზე უფრო ადვილი იქნება.

სავარჯიშო 6.24: მოცემულია დალაგებული სიმრავლეები A და B (სიმარტივისთვის დაიშვით, რომ ორივე სიმრავლე რიცხვებს შეიცავს). დაწერეთ ალგორითმი, რომლითაც გამოვითვლით $A \cap B$. დაამტკიცეთ მისი სისწორე და გამოითვალეთ ბიჯების რაოდენობა.

სავარჯიშო 6.25: მოცემულია დალაგებული სიმრავლეები A და B (სიმარტივისთვის დაიშვით, რომ ორივე სიმრავლე რიცხვებს შეიცავს). დაწერეთ ალგორითმი, რომლითაც გამოვითვლით $A \cup B$. დაამტკიცეთ მისი სისწორე და გამოითვალეთ ბიჯების რაოდენობა.

სავარჯიშო 6.26: მოცემულია დალაგებული სიმრავლეები A და B (სიმარტივისთვის დაიშვით, რომ ორივე სიმრავლე რიცხვებს შეიცავს). დაწერეთ ალგორითმი, რომლითაც გამოვითვლით $A \backslash B$. დაამტკიცეთ მისი სისწორე და გამოითვალეთ ბიჯების რაოდენობა.

რაც შეეხება ეკვივალენტურობის მიმართების გამოყენებას, წინა თავში განხილული მოდულარული არითმეტიკა სწორედ ამ სტრუქტურებს ეფუძვნება:
თუ განვიხილავთ $\mathbb{Z}_k = \{0, 1, ..., k-2, k-1\}$ რიცხვთა სიმრავლეს, ეს წარმოიშვება $\mathbb{Z}$ მთელ რიცხვთა სიმრავლეში არსებული რიცხვების k რიცხვზე გაყოფითა და ნაშთის აღებით, ანუ ყველა ის რიცხვი, რომელიც k რიცხვზე გაყოფისას ერთსა და იმავე ნაშთს იძლევა, ეკვივალენტურადაა გამოცხადებული და გვრჩება შემდეგი ელემენტები:

$$
\begin{aligned}
0 \equiv [0] &= \{0, k, -k, 2k, -2k, 3k, -3k, 4k, -4k, ...\}, \\
1 \equiv [1] &= \{1, -1, k+1, -(k+1), 2k+1, -(2k+1), ...\}, \\
& \quad ..., \\
k-2 \equiv [k-2] &= \{k-2, -(k-2), 2k-2, -(2k-2), ...\}, \\
k-1 \equiv [k-1] &= \{k-1, -(k-1), 2k-1, -(2k-1), ...\}
\end{aligned}
$$

საერთოდ, ალგებრაში ხშირად იყენებენ ე-წ- ფაქტორრგოლებს, რომელსაც რაიმე რგოლის ერთ ელემენტზე გაყოფითა და ნაშთების აღებით წარმოქმნიან ხოლმე, რაც ეკვივალენტურობის კლასების წარმოქმნით ხდება.

6.3 მოკლე დასკვნა

მეექვსე თავში განვიხილეთ მიმართებების განმარტება, აქედან გამომდინარე ეკვივალენტურობისა და დალაგების მიმართებები და მათი გამოყენების საშუალებები. სიმრავლეთა ეკვივალენტურობის კლასებად დაყოფა გამოიყენება არა მხოლოდ საბუნებისმეტყველო დარგებში (მაგალითად ბიოლოგიაში ცოცხალი ორგანიზმების კლასიფიკაციისთვის), არამედ მათემატიკაში ნაშთთა კლასებში, რაც წინა თავში იყო განხილული მოდულარული არითმეტიკის აღწერის დროს, ხოლო დალაგება კი ცენტრალურ როლს თამაშობს თითქმის ყველა ამოცანაში: საკმარისია განვიხილოთ სიტყვის დალაგებულ ლექსიკონში ძებნის ამოცანა და შევადაროთ იგი დაულაგებელ სიმრავლეში ძებნას.

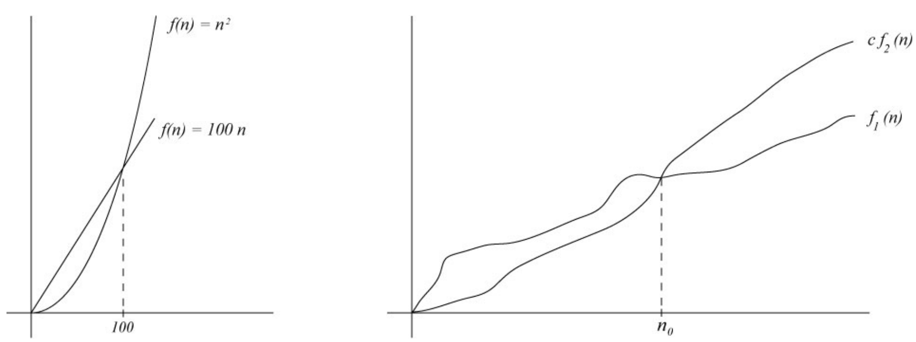
თავი 7

ალგორითმების სისწრაფის შეფასება

7.1 ფუნქციათა ზრდის რიგი

განვიხილოთ ორი ფუნქცია: $f_1(n) = n^2$ და $f_2(n) = 100 \cdot n, n > 0$. ცხადია, რომ $f_2(n) > f_1(n)$, თუ $0 < n < 100$. მაგრამ თუ $n > 100$, მაშინ $f_1(n) > f_2(n)$. ესე იგი, დაწყებული რაღაცა ადგილიდან, $f_1(n) > f_2(n)$ (ნახ. 7.1 მარცხნივ). ასეთ შემთხვევებში - როდესაც დაწყებული რაღაცა ადგილიდან ერთი ფუნქციის მნიშვნელობა ყოველთვის აჭარბებს მეორე ფუნქციის შესაბამის მნიშვნელობას - ამბობენ, რომ f_1 ფუნქცია უფრო სწრაფად იზრდება, ვიდრე f_2. მაგალითად, $f_1(n) = n$ უფრო სწრაფად იზრდება, ვიდრე $f_2(n) = \log n$ (აქ და შემდგომში $\log n = \log_2 n, \ln n = \log_e n$ და $\lg n = \log_{10} n$).

შენიშვნა: აქ და შემდგომში განხილული ფუნქციები დადებითია.



ნახ. 7.1: ორი ფუნქციის გრაფიკი

სავარჯიშო 7.1: $f_1(n)$ და $f_2(n)$ ფუნქციებს შორის რომელი იზრდება უფრო სწრაფად? (პასუხი დაამტკიცეთ)

1. $f_1(n) = 10 \cdot n^2$, თუ $f_2(n) = 15 \cdot n^2$; 2. $f_1(n) = 0.1 \cdot n^2$, თუ $f_2(n) = n$; 3. $f_1(n) = 10^6 \cdot \log n$, თუ $f_2(n) = n$; 4. $f_1(n) = 10 \cdot \log n^2$, თუ $f_2(n) = 20 \cdot \log n$; 5. $f_1(n) = 2^n$, თუ $f_2(n) = 15^{10} \cdot n^7$.

სავარჯიშო 7.2: დაამტკიცეთ, რომ $f_1(n)$ ფუნქცია უფრო სწრაფად იზრდება, ვიდრე $f_2(n)$, თუ:

1. $f_1(n) = n^2$, $f_2(n) = 15 \cdot n \cdot \log n$; 2. $f_1(n) = n^3$, $f_2(n) = 1983 \cdot n$; 3. $f_1(n) = \log n$, $f_2(n) = 10 \log\log n$; 4. $f_1(n) = \log n^2$, $f_2(n) = 100\sqrt{\log n}$; 5. $f_1(n) = n$, $f_2(n) = \log^7 n$.

გამონათქვამი „დაწყებული რაღაცა ადგილიდან f_1 ფუნქციის მნიშვნელობა ყოველთვის აჭარბებს f_2 ფუნქციის შესაბამის მნიშვნელობას" მათემატიკურად შემდეგნაირად ჩაიწერება: $\exists n_0 \in \mathbb{N}, \ \forall n > n_0, \ f_1(n) > f_2(n)$.

თუ მოცემულია ორი ფუნქცია $f_1(n), f_2(n)$ და $\exists c \in \mathbb{N}$ ისეთი, რომ დაწყებული რაღაცა ადგილიდან $f_1(n) < c \cdot f_2(n)$, მაშინ ამბობენ, რომ $f_1(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი არ აღემატება $f_2(n)$ ფუნქციის ასიმპტოტური ზრდის რიგს.

ამ შემთხვევაში აგრეთვე ამბობენ, რომ f_1 ფუნქციის ზრდის რიგი ზემოდანაა შემოსაზღვრული f_2 ფუნქციის ზრდის რიგით, ანუ f_2 ფუნქციის ზრდის რიგი f_1 ფუნქციის ზრდის რიგის ზედა ზღვარია.

მაგალითად, თუ $f_1(n) = 10 \cdot n$ და $f_2(n) = n$, $f_1(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი არ აღემატება $f_2(n)$ ფუნქციის ასიმპტოტური ზრდის რიგს, რადგან $\exists c = 11$ და $f_1(n) = 10 \cdot n < c \cdot f_2(n) = 11 \cdot n$.
ასიმპტოტური ზრდის რიგი გვიჩვენებს, „დაახლოებით რა სისწრაფით" იზრდება მოცემული ფუნქცია. ზედა მაგალითში შეგვეძლო აგრეთვე დაგვეწერა: $\exists c = 1$ და $c \cdot f_1(n) = 10 \cdot n > f_2(n) = n$. ასე რომ, ერთ შემთხვევაში $f_1(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი არ აღემატება $f_2(n)$ ფუნქციის ასიმპტოტური ზრდის რიგს, მეორე შემთხვევაში კი პირიქით. ასეთ დროს იტყვიან, რომ ამ ორი ფუნქციის ასიმპტოტური ზრდის რიგი ტოლია, ანუ ორივე „დაახლოებით ერთი სისწრაფით იზრდება". თუ მოცემულია ორი ფუნქცია $f_1(n), f_2(n)$ და $\exists c \in \mathbb{N}$ ისეთი, რომ დაწყებული რაღაცა ადგილიდან $f_1(n) < c \cdot f_2(n)$, მაგრამ $\nexists d \in \mathbb{N}$ ისეთი, რომ დაწყებული რაღაცა ადგილიდან $f_2(n) < d \cdot f_1(n)$, მაშინ ამბობენ, რომ $f_2(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი უფრო მაღალია, ვიდრე $f_1(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი. ცხადია, რომ თუ $f_1(n)$ და $f_2(n)$ ფუნქციების ასიმპტოტური ზრდის რიგი ტოლია, შეიძლება ასევე ითქვას, რომ $f_1(n)$ ფუნქციის ასიმპტოტური ზრდის რიგი არ აღემატება $f_2(n)$ ფუნქციის ასიმპტოტური ზრდის რიგს (და პირიქით).

ქვემოთ მოყვანილია ცხრილი, რომელიც რამოდენიმე ფუნქციის ზრდის რიგს გვიჩვენებს.

n	$\log n$	n	$n \cdot \log n$	n^2	2^n	$n!$
10	3	10	30	100	1.024	3.628.800
20	4	20	80	400	1.048.576	$\gg 10^{15}$
30	5	30	150	900	1.073.741.824	
40	5	40	200	1600	1.099.511.627.776	
50	6	50	300	2500	$>10^{15}$	
100	7	100	700	10^4	$>10^{30}$	
1.000	10	1.000	10.000	10^6		
10.000	13	10.000	130.000	10^8		
100.000	17	100.000	1.700.000	10^{10}		
1.000.000	20	1.000.000	20.000.000	10^{12}		
10.000.000	23	10.000.000	230.000.000	10^{14}		
100.000.000	27	100.000.000	2.700.000.000	10^{16}		
1.000.000.000	30	1.000.000.000	30.000.000.000	10^{18}		

როგორც ვხედავთ, ამ ფუნქციათა შორის ყველაზე ნელა $f(n) = \log n$ ფუნქცია იზრდება, ყველაზე სწრაფად კი $f(n) = n!$. ამ ბოლო ფუნქციის მნიშვნელობა $n = 20$ -თვის უკვე ძალიან დიდია - როგორც ვარაუდობენ, $2^{100} = 10^{30}$ ჩვენს სამყაროში არსებული ატომების რაოდენობას აღემატება და, აქედან გამომდინარე, 10^{15} ძალიან დიდი რიცხვია.

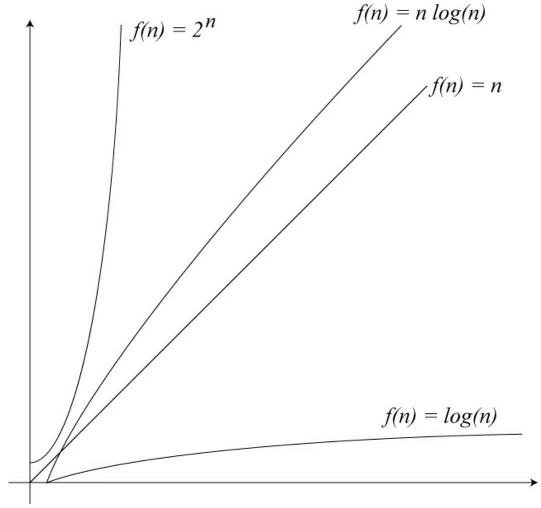
სავარჯიშო 7.3: დაამტკიცეთ, რომ $f_1(n) = 10n^2$ და $f_2(n) = 10^{-6} \cdot n^2$ ფუნქციათა ასიმპტოტური ზრდის რიგი ტოლია.

სავარჯიშო 7.4: ტოლია თუ არა შემდეგი ორი ფუნქციის ასიმპტოტური ზრდის რიგი (პასუხები დაამტკიცეთ):

1. $f_1(n) = n^2,\ f_2(n) = 15 \cdot n^2 \cdot \log\log n$; 2. $f_1(n) = \log n^3,\ f_2(n) = 1983 \cdot n$; 3. $f_1(n) = \log^2 n,\ f_2(n) = 10 \log n$; 4. $f_1(n) = \log n^2,\ f_2(n) = 100\sqrt{\log n}$; 5. $f_1(n) = n,\ f_2(n) = \log\log^7 n$.

ნახ. 7.2 გვიჩვენებს რამოდენიმე ფუნქციის ზრდის სისწრაფეს, საიდანაც შეიძლება მათი ასიმპტოტური ზრდის რიგის დანახვა. ყველაზე ნელა იზრდება ლოგარითმული ფუნქცია $f(n) = \log n$; შემდეგია წრფივი ფუნქცია $f(n) = n$. მასზე სწრაფად იზრდება ფუნქცია $n \cdot \log n$ და ყველაზე დიდი ზრდის რიგი აქვს $f(n) = 2^n$ ფუნქციას.

სავარჯიშო 7.5: $f_1(n)$ და $f_2(n)$ ფუნქციებს შორის რომლის ასიმპტოტური ზრდის რიგია უფრო მაღალი?



ნახ. 7.2: რამოდენიმე ფუნქციის გრაფიკი

1. $f_1(n) = \log^2 n,\ f_2(n) = \sqrt{n}$; 2. $f_1(n) = n^3,\ f_2(n) = 1983 \cdot n^2$; 3. $f_1(n) = n \cdot \log n,\ f_2(n) = 2^{\log n}$; 4. $f_1(n) = n^2 \cdot \log n$, $f_2(n) = n^2$; 5. $f_1(n) = \sqrt[3]{n},\ f_2(n) = (\log \log n)^7$.

თუ მოცემულია რაიმე ფუნქცია $f(n)$, შეგვიძლია გამოვყოთ ყველა იმ ფუნქციათა სიმრავლე $O(f(n))$ (იკითხება: ო-დიდი $f(n)$), რომელთა ასიმპტოტური ზრდის რიგი ამ $f(n)$ ფუნქციის ზრდის რიგს არ აღემატება (ანუ ამ სიმრავლეში შემავალი ყველა ფუნქცია ამ ფუნქციის „ქვედა ზღვარია" - დაწყებული რაღაცა ადგილიდან ყოველთვის უფრო ნაკლები იქნება):

$$O(f(n)) = \{g(n) | \exists n_0, c \in \mathbb{N}, \forall n > n_0,\ c \cdot f(n) > g(n)\}.$$

ცხადია, რომ $O(f(n))$ სიმრავლე უსასრულოა, ამიტომ მასში შემავალი ყველა ფუნქციის ამოწერა შეუძლებელია. მაგრამ შესაძლებელია ამ სიმრავლეში შემავალი რამოდენიმე ფუნქციის მაგალითის მოყვანა:

თუ $f(n) = n$, მაშინ $g(n) = 100 \cdot n \in O(f(n))$, რადგან $\exists c = 101$ და $c \cdot f(n) = 101 \cdot n > 100 \cdot n = g(n)$.
ანალოგიურად შეგვიძლია დავამტკიცოთ: $100n \in O(n \cdot \log n)$, $700n \in O(n^2)$, $200n^2 \in O(2^n)$.

<u>სავარჯიშო 7.6:</u> დაამტკიცეთ, რომ $100n \in O(n \cdot \log n)$, $700n \in O(n^2)$, $200n^2 \in O(2^n)$.

<u>სავარჯიშო 7.7:</u> მოიყვანეთ $O(n \log n)$ სიმრავლის 5 ელემენტი.

<u>ლემა 7.1:</u> თუ $f_1(n)$ ფუნქცია არ იზრდება უფრო სწრაფად, ვიდრე $f_2(n)$ ფუნქცია, მაშინ $O(f_1(n)) \subset O(f_2(n))$.

<u>დამტკიცება:</u> რადგან $f_1(n)$ ფუნქცია არ იზრდება უფრო სწრაფად, ვიდრე $f_2(n)$ ფუნქცია, ამიტომ $\exists d \in \mathbb{N}, f_1(n) < d \cdot f_2(n)$. ახლა განვიხილოთ ნებისმიერი $g(n) \in O(f_1(n))$. განმარტების თანახმად $\exists c \in \mathbb{N}, g(n) < c \cdot f_1(n)$. ზემოთ მოყვანილი უტოლობის თანახმად, $g(n) < c \cdot d \cdot f_2(n)$. ესე იგი, $\exists d \cdot c \in \mathbb{N}$ ისეთი, რომ $g(n) < c \cdot d \cdot f_2(n)$, რაც განმარტების თანახმად ნიშნავს, რომ $g(n) \in O(f_2(n))$.

Q.E.D.

აქედან გამომდინარე, გამონათქვამი „f_1 ფუნქცია არ იზრდება უფრო სწრაფად, ვიდრე f_2 ფუნქცია" შემდეგი მათემატიკური ჩანაწერის ტოლფასია: $O(f_1(n)) \subset O(f_2(n))$.

<u>სავარჯიშო 7.8:</u> მოიყვანეთ $f_1(n)$ და $f_2(n)$ ფუნქციების მაგალითები, რომელთათვისაც $O(f_1(n)) \subset O(f_2(n))$ და, ამავდროულად, $O(f_1(n)) \neq O(f_2(n))$.

<u>ლემა 7.2:</u> $O(f(n))$ სიმრავლეებისათვის ჭეშმარიტია:

1. $O(k \cdot f(n)) = O(f(n))$ $(k \in \mathbb{N})$;

2. $O(f(n)+k) = O(f(n))$ $(k \in \mathbb{N})$;

3. თუ $O(f_1(n)) \subset O(f_2(n))$, მაშინ $O(f_1(n)+f_2(n)) = O(f_2(n))$.

დამტკიცება:

1. თუ ვაჩვენებთ, რომ $O(k \cdot f(n)) \subset O(f(n))$ და $O(k \cdot f(n)) \supset O(f(n))$, ტოლობა დამტკიცდება.
განვიხილოთ ნებისმიერი $g(n) \in O(k \cdot f(n))$. განმარტების თანახმად $\exists c \in \mathbb{N}$ ისეთი, რომ $g(n) < c \cdot k \cdot f(n)$ (რადგან k ნატურალურია). ეს კი განმარტების თანახმად იმას ნიშნავს, რომ $g(n) \in O(f(n))$: $\exists c \cdot k \in \mathbb{N}$ ისეთი, რომ $g(n) < c \cdot k \cdot f(n)$.
ახლა კი განვიხილოთ ნებისმიერი $g(n) \in O(f(n))$. განმარტების თანახმად $\exists d \in \mathbb{N}$ ისეთი, რომ $d \cdot f(n) > g(n)$. თუ უტოლობის ორივე მხარეს გავამრავლებთ k რიცხვზე, მივიებთ: $k \cdot d \cdot f(n) > k \cdot g(n) > g(n)$ (რადგან $k \in \mathbb{N}$). აქედან გამომდინარე, $\exists d \in \mathbb{N}$ ისეთი, რომ $d \cdot (k \cdot f(n)) > g(n)$. ესე იგი, $g(n) \in O(k \cdot f(n))$ ($O(k \cdot f(n))$ სიმრავლის განმარტების თანახმად).

Q.E.D.

სავარჯიშო 7.9: დაამტკიცეთ ზემოთ მოყვანილი ლემას მე-2-ე და მე-3-ე პუნქტები.

ანალოგიურად შეიძლება ნებისმიერი $f(n)$ ფუნქციის ზრდის რიგის ქვედა ზღვარი (ომეგა-დიდი $f(n)$) განვმარტოთ:

$$\Omega(f(n)) = \{g(n) | f(n) \in O(g(n))\}.$$

ეს ყველა იმ ფუნქციათა სიმრავლეა, რომელთა ასიმპტოტური ზრდის რიგი $f(n)$ ფუნქციის ასიმპტოტური ზრდის რიგზე ნაკლები არაა.

სავარჯიშო 7.10: დაამტკიცეთ, რომ $f_1(n) = 10n^2$ და $f_2(n) = 10^{-6}n^2$ ფუნქციათა ზრდის რიგის ქვედა ზღვარი ტოლია.

სავარჯიშო 7.11: ტოლია თუ არა შემდეგი ორი ფუნქციის ასიმპტოტური ზრდის რიგის ქვედა ზღვარი? (პასუხები დაამტკიცეთ):

1. $f_1(n) = n^2,\ f_2(n) = 15 \cdot n^2 \cdot \log\log n$; 2. $f_1(n) = \log n^3,\ f_2(n) = 1983 \cdot n$; 3. $f_1(n) = \log^2 n,\ f_2(n) = 10 \log n$; 4. $f_1(n) = \log n^2,\ f_2(n) = 100\sqrt{\log n}$; 5. $f_1(n) = n,\ f_2(n) = \log\log^7 n$.

სავარჯიშო 7.12: დაამტკიცეთ, რომ $n \cdot \log n \in \Omega(100n)$, $n^2 \in \Omega(100n)$, $2^n \in \Omega(100n)$.

სავარჯიშო 7.13: მოიყვანეთ $\Omega(\log n)$ სიმრავლის 5 ელემენტი.

სავარჯიშო 7.14: დაამტკიცეთ, რომ თუ $f_1(n)$ ფუნქცია არ იზრდება უფრო სწრაფად, ვიდრე $f_2(n)$ ფუნქცია, მაშინ $\Omega(f_2(n)) \subset \Omega(f_1(n))$.

სავარჯიშო 7.15: მოიყვანეთ $f_1(n)$ და $f_2(n)$ ფუნქციების მაგალითები, რომელთათვისაც $\Omega(f_1(n)) \subset \Omega(f_2(n))$ და, ამავდროულად, $\Omega(f_1(n)) \neq \Omega(f_2(n))$.

7.2 ალგორითმების ბიჯების რაოდენობის შეფასება: ზედა, ქვედა და ზუსტი ზღვარი

განვიხილოთ შემდეგი ამოცანა:

მოცემულია: რიცხვების მიმდევრობა $a_1, a_2, ..., a_n \in \mathbb{N}$ და დამატებით ერთი რიცხვი $b \in \mathbb{N}$.

შედეგი: „კი" ან „არა"

შეზღუდვა: „კი" მაშინ და მხოლოდ მაშინ, თუ $\exists i \in \mathbb{N},\ 1 \leq i \leq n,\ a_i = b$.

სხვა სიტყვებით რომ ვთქვათ, ალგორითმი ადგენს, მოიძებნება თუ არა $a_1, ..., a_n$ მიმდევრობაში ერთი მაინც რიცხვი $a_i = b$.

ქვემოთ მოყვანილია რეკურსიული ალგორითმი, რომელიც ამ ამოცანას ხსნის:

ალგორითმი $K(a_1, a_2, ..., a_n, b)$

1. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე;
2. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე;
3. ჩაატარე ალგორითმი $K(a_2, ..., a_n, b)$.

განვიხილოთ ამ ალგორითმის ბიჯები საწყის მონაცემებზე $a_1 = 3, a_2 = 7, a_3 = 0, a_4 = 8, b = 2$.

ალგორითმი $K(3, 7, 0, 8, 2)$ (აქ $a_1 = 3, a_2 = 7, a_3 = 0, a_4 = 8, b = 2$).

1. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა (ცარიელია), დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს არ სრულდება)
2. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე; (ეს არ სრულდება)

   $K(7, 0, 8, 2)$ (აქ $a_1 = 7, a_2 = 0, a_3 = 8, b = 2$).
3. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს არ სრულდება)
4. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე; (ეს არ სრულდება)

   $K(0, 8, 2)$ (აქ $a_1 = 0, a_2 = 8, b = 2$).
5. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს არ სრულდება)
6. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე; (ეს არ სრულდება)

   $K(8, 2)$ (აქ $a_1 = 8, b = 2$).
7. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს არ სრულდება)
8. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე; (ეს არ სრულდება)

   $K(2)$ (აქ $a_1, a_2, ..., a_n$ მიმდევრობა ცარიელია).
9. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს სრულდება)

მაგრამ თუ ამოცანის საწყისი მონაცემებია $a_1 = 3, a_2 = 7, a_3 = 0, a_4 = 8, b = 3$, მაშინ ალგორითმის მსვლელობა შემდეგნაირი იქნებოდა:

ალგორითმი $K(3, 7, 0, 8, 3)$ (აქ $a_1 = 3, a_2 = 7, a_3 = 0, a_4 = 8, b = 3$).

1. თუ მიმდევრობა $a_1, a_2, ..., a_n$ შემოსული არაა, დაბეჭდე „არა" და ალგორითმი დაასრულე; (ეს არ სრულდება)
2. თუ $a_1 = b$ დაბეჭდე „კი" და ალგორითმი დაასრულე; (ეს სრულდება)

ამ მაგალითიდან ჩანს, რომ ალგორითმების ბიჯების რაოდენობა დამოკიდებულია მის მონაცემთა რაოდენობასა და თვითონ მონაცემთა მნიშვნელობებზე.

განასხვავებენ ბიჯების შეფასების სამ შემთხვევას:

- უარესი შემთხვევის ანალიზს (worst-case), ანუ მაქსიმუმ რამდენი ბიჯი დაგვჭირდება ამ ამოცანის გადასაჭრელად, მაშინაც კი, როდესაც ყველაზე „ცუდი" მონაცემები შემოგვივა?
- საუკეთესო შემთხვევის ანალიზს (best-case), ანუ მინიმუმ რამდენი ბიჯი დაგვჭირდება ამ ამოცანის გადასაჭრელად, როდესაც ყველაზე „კარგი" მონაცემები შემოგვივა?

- საშუალო შემთხვევის ანალიზს (average-case), ანუ საშუალოდ რამდენი ბიჯი დაგვჭირდება ამ ამოცანის გადასაჭრელად?

ადვილი დასანახია, რომ ჩვენს ზედა ამოცანაში ალგორითმი $n+1$ მონაცემის დამუშავებას (n ელემენტიან მასივში რაღაცა b რიცხვის პოვნას) მაქსიმუმ $2n+1$ ბიჯსა და მინიმუმ 2 ბიჯს მოანდომებს.

სავარჯიშო 7.16: დაამტკიცეთ ზემოთ მოყვანილი გამონათქვამი.

სხვა სიტყვებით რომ ვთქვათ, უარესი შემთხვევის ანალიზის შედეგად მიღებული ფუნქცია $f(n)$ გვეუბნება, რომ „n ცალი მონაცემისათვის მოცემული ალგორითმის ბიჯების რაოდენობა არასოდეს არ გადააჭარბებს $f(n)$ ფუნქციას", ხოლო საუკეთესო შემთხვევის ანალიზის შედეგად მიღებული ფუნქცია კი გვეუბნება, რომ „მოცემული ალგორითმის ბიჯების რაოდენობა ვერასოდეს ვერ იქნება ამ ფუნქციაზე ნაკლები".

რაც შეეხება გამოთვლის საშუალო დროის დადგენას, ეს პროცესი მათემატიკურ სტატისტიკას ემყარება და ამ კურსში მას არ განვიხილავთ.

ცხადია, რომ n მონაცემის დამუშავების მაქსიმალური და მინიმალური დრო მონაცემთა რაოდენობის ცვლილებასთან ერთად იცვლება, ანუ ეს არის ფუნქცია, რომელიც დამოკიდებულია $n \in \mathbb{N}$ ცვლადზე. ჩვენს ზედა მაგალითში ბიჯების მაქსიმალური რაოდენობაა $f_1(n) = 2n+1$, ხოლო მინიმალური კი $f_2(n) = 2$.

განვიხილოთ ალგორითმი, რომელიც n ცალი მონაცემის დამუშავებას მაქსიმუმ $f(n)$ ბიჯს ანდომებს, ხოლო 1 ბიჯის დამუშავებას კი 10^{-9} წამს ანდომებს.

ქვემოთ მოყვანილი ცხრილი, სადაც წარმოდგენილია $f(n)$ ფუნქციის რამოდეიმე მაგალითი. მასში ნაჩვენებია, მაქსიმუმ რამდენ ხანს მოანდომებს ეს ალგორითმი n მონაცემის დამუშავებას. ამ ცხრილში $1\mu s = 10^{-6}$ წმ, $1ms = 10^{-3}$ წმ ($1\mu s$ იკითხება: 1 მიკრო წამი, $1ms$ იკითხება: 1 მილი წამი).

n	$f(n) = \log n$	$f(n) = n$	$f(n) = n \cdot \log n$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = n!$
10	0,003 μs	0,01 μs	0,033 μs	0,1 μs	1 μs	3,63 ms
20	0,004 μs	0,02 μs	0,086 μs	0,4 μs	1 ms	77,1 წელი
30	0,005 μs	0,03 μs	0,147 μs	0,9 μs	1 წმ	8,4×10^{15} წელი
40	0,005 μs	0,04 μs	0,213 μs	1,6 μs	18,3 წთ	
50	0,006 μs	0,05 μs	0,282 μs	2,5 μs	13 დღე	
100	0,007 μs	0,1 μs	0,644 μs	10 μs	4×10^{13} წელი	
1.000	0,010 μs	1 μs	9,966 μs	1 ms		
10.000	0,013 μs	10 μs	130 μs	100 ms		
100.000	0,017 μs	9,10 ms	1,67 ms	10 წმ		
1.000.000	0,020 μs	1 ms	19,93 ms	16.7 წთ		
10.000.000	0,023 μs	0,01 წმ	0,23 წმ	1,16 დღე		
100.000.000	0,027 μs	0,1 წმ	2,66 წმ	115,7 დღე		
1.000.000.000	0,03 μs	1 წმ	29,9 წმ	31,7 წელი		

ამ ცხრილიდან ჩანს, რომ თუ ალგორითმის ბიჯების რაოდენობაა $f(n) = n \cdot \log n$ ან უფრო ნელა ზრდადი ფუნქცია, მაშინ მისი გამოთვლები საკმაოდ სწრაფი იქნება. თუ $f(n) = n^2$, გამოთვლები სწრაფი იქნება დაახლოებით 50.000.000 ელემენტამდე. მაგრამ თუ $f(n) = 2^n$, ასეთი ალგორითმი პრაქტიკაში ვერ გამოიყენება 53-ზე მეტი მონაცემისათვის. თუ ალგორითმის ზედა ზღვარია $f(n) = n!$, მაშინ იგი პრაქტიკულად საერთოდ ვერ გამოიყენება.

თავი 8

დალაგების ალგორითმები და მათი დროითი სირთულის ანალიზი

დალაგებულ სიმრავლეებზე ბევრი ამოცანის სწრაფად გადაჭრა შეიძლება, მათ შორის ძებნისაც: მილიონიანი ქალაქის ტელეფონის წიგნში მოცემული სახელისა და გვარის პიროვნების ნომრის პოვნა, როგორც წესი, სულ რამოდენიმე წამს გრძელდება, დაულაგებელ სიმრავლეში ძებნას კი ყველა თავს აარიდებდა. დალაგების სხვა გამოყენება სტატისტიკური მონაცემების გადამუშავებაშია: დავუშვათ, გვაინტერესებს, საქართველოში 25 - 30 წლის ასაკის რამდენ მოქალაქეს აქვს მიღებული განათლება თბილისის სახელმწიფო უნივერსიტეტში? ამისათვის საქართველოს მოქალაქეთა ბაზა უნდა დავახარისხოთ ჯერ ასაკის მიხედვით, შემდეგ კი - განათლების მიღების ინსტიტუტის მიხედვით. შედეგად მიღებული ორი ცხრილიდან ადვილად შეიძლება სტატისტიკის დადგენა.
თუ საჭიროა დიდ მონაცემთა ბაზაში გამეორებების აღმოფხვრა, აქაც შედეგს ბაზის დალაგებით ეფექტურად მივიღებთ.
დიდი ბაზის გადამუშავების ერთ-ერთი პირველი მაგალითია ამერიკის შეერთებული შტატების მოსახლეობის 1880 წლის აღწერა. 1500 ადამიანი შვიდი წლის განმავლობაში ალაგებდა ბაზას საჭირო მონაცემების მიხედვით. იმ დროისათვის უცნობმა ინჟინერმა ჰერმან ჰოლერითმა (Herman Hollerith) დაახლოებით ათი წელი მოანდომა დამახარისხებელი მანქანების შექმნას, რომლის საშუალებითაც 1890 წლის აღწერა (მეტი მოსახლეობითა და დასამუშავებელი მონაცემით) ხუთასმა თანამშრომელმა ორ წელიწადზე ნაკლებ დროში დაასრულა. ჰოლერითის ფირმა, რომელიც 1924 წლიდან International Business Machines (IBM) Corporation სახელითაა ცნობილი, შემდგომშიც დიდ როლს თამაშობდა მეცნიერებასა და ტექნიკაში. ყველაფერი კი დალაგების ალგორითმებითა და მათი იმპლემენტაციით დაიწყო.

როგორც ვიცით, ერთსა და იმავე სიმრავლეზე შესაძლებელია სხვადასხვა დალაგების განსაზღვრა (აქ და შემდგომში - თუ სხვაგვარად არ იქნა აღნიშნული - განვიხილავთ სრულ და რეფლექსურ დალაგებას $\leq$).

<u>სავარჯიშო 8.17:</u> $\mathbb{N}$ ნატურალურ რიცხვთა სიმრავლეზე მოიყვანეთ ხუთი სხვადასხვა დალაგების მაგალითი.

<u>სავარჯიშო 8.18:</u> მოცემულია ორი სიმრავლე A დალაგებით $\leq_A$ და B დალაგებით $\leq_B$. როგორ შეიძლება განვსაზღვროთ დალაგება $A \times B$ სიმრავლეზე?

<u>სავარჯიშო 8.19:</u> განსაზღვრეთ ისეთი სრული დალაგება $\leq$ კომპლექსურ რიცხვთა $\mathbb{C} = \mathbb{R} \times \mathbb{R}$ სიმრავლეზე, რომ $a \leq b \Rightarrow |a| \leq |b|$ (თუ $c = (c_1, c_2) \in \mathbb{C}, |c| = \sqrt{c_1^2 + c_2^2}$).

8.1 ძებნა და ჩასმა დალაგებულ მიმდევრობებში

განვიხილოთ შემდეგი ***ძებნის ამოცანა:*** მოცემულია დალაგებუი მიმდევრობა $A = (a_1, a_2, ..., a_n), a_1 \leq a_2 \leq ... \leq a_n$ და ელემენტი c. დაადგინეთ, ჭეშმარიტია თუ არა $c \in A$. სხვა სიტყვებით რომ ვთქვათ, უნდა დავადგინოთ, არის თუ არა c ელემენტი A სიაში.
ცხადია, რომ ჩვენ შეგვიძლია A მიმდევრობის ყველა ელემენტის c რიცხვთან შედარება და თუ $\exists i, a_i = c$, პასუხი იქნება „კი", წინააღმდეგ შემთხვევაში - „არა".

<u>სავარჯიშო 8.20:</u> დაწერეთ რეკურსიული ალგორითმი, რომელიც ზემოთ აღწერილი მეთოდით დაადგენს, შედის თუ არა c ელემენტი A მიმდევრობაში. დაამტკიცეთ, რომ თუ $|A| = n$, ყველაზე ცუდ შემთხვევაში $O(n)$ ბიჯი იქნება საჭირო. რამდენი ბიჯი დაჭირდება ალგორითმს ყველაზე კარგ შემთხვევაში?

ამ მეთოდში ჩვენ არ ვითვალისწინებთ იმ ფაქტს, რომ A სიმრავლე დალაგებულია. არა და, დალაგებულ მიმდევრობაში, თუ ავიღებთ A სიმრავლის ნებისმიერ ელემენტს a_i და $c > a_i$, მაშინ ცხადია, რომ ძებნა მიმდევრობის a_i ელემენტიდან მარცხენა ნაწილში საჭირო არაა. ამ იდეაზეა აგებული შემდეგი ალგორითმი: A მიმდევრობის შუა ელემენტს ვუწოდოთ a (თუ შუა ელემენტი არ არსებობს, ვიღებთ სიის მარჯვენა ნახევრის მინიმალურ ელემენტს). თუ $a = c$, მაშინ ვადგენთ, რომ $c \in A$ და ალგორითმს ვასრულებთ. თუ $c < a$, მაშინ ძებნა A სიის მარცხენა ნახევარში უნდა გავაგრძელოთ, წინააღმდეგ შემთხვევაში - მარჯვენაში. ამ პროცედურას ვიმეორებთ მანამ, სანამ საძიებელი სიმრავლე ცარიელი არ იქნება (ანუ საძიებელი ელემენტი არ მოიძებნება).

<u>მაგალითი:</u>

საწყისი მონაცემი: $A = (-12, -8, 1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 17)$, $c = 1$.

$$
\begin{array}{llllll}
c < a? & 1 = 5?\ \text{არა};\ 1 < 5?\ \text{კი} & & 1 = 2?\ \text{არა};\ 1 < 2?\ \text{კი} & & 1 = -8?\ \text{არა};\ 1 < -8?\ \text{არა} \\
A & (\underbrace{-12, -8, 1, 2, 3, 4}, [5], 8, 9, 11, 12, 13, 17) & \rightarrow & (\underbrace{-12, -8, 1}, [2], 3, 4) & \rightarrow & (-12, [-8], \underbrace{\ 1\ }) \\
 & & & & & \\
 & 1 = 1?\ \ \text{კი} & & & & \\
\rightarrow & ([1]) \rightarrow\ \text{პასუხი: არსებობს} & & & &
\end{array}
$$

<u>მაგალითი:</u>

საწყისი მონაცემი: $A = (-12, -8, 1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 17)$, $c = 15$.

$$
\begin{array}{llllll}
c < a? & 15 = 5?\ \text{არა};\ 15 < 5?\ \text{არა} & & 15 = 12?\ \text{არა};\ 15 < 12?\ \text{არა} & & 15 = 17?\ \text{არა};\ 15 < 17?\ \text{კი} \\
A & (-12, -8, 1, 2, 3, 4, [5], \underbrace{8, 9, 11, 12, 13, 17}) & \rightarrow & (8, 9, 11, [12], \underbrace{13, 17}) & \rightarrow & (\underbrace{\ 13\ }, [17]) \\
 & & & & & \\
 & 15 = 13?\ \ \text{არა} & & & & \\
\rightarrow & ([13]) & \rightarrow & () & \rightarrow & \text{პასუხი: არ არსებობს}
\end{array}
$$

იტერაციულად ეს ალგორითმი შემდეგნაირად შეიძლება ჩაიწეროს:

მოცემულობა: რაციონალური რიცხვებისგან შემდგარი, დალაგებული სასრული მიმდევრობა A და რაციონალური რიცხვი c;
შედეგი: პასუხი შეკითხვაზე $c \in A$?

Find(A, c)
სანამ A მიმდევრობა ცარიელი არაა, გაიმეორე შემდეგი ოპერაციები:
{
  $a = A$ მიმდევრობის შუა ელემენტი;
  if(a = c) {*return*(„კი”)}
    else if(a < c) $A = A$ სიმრავლის მარჯვენა ნახევარი;
      else $A = A$ სიმრავლის მარცხენა ნახევარი
}
{*return*(„არა”)}

<u>სავარჯიშო 8.21:</u> იგივე ამოცანა ჩაწერეთ რეკურსიულად და დაამტკიცეთ მისი სისწორე.

<u>სავარჯიშო 8.22:</u> დაამტკიცეთ, რომ ზემოთ მოყვანილი ალგორითმის დროის ზედა ზღვარია $O(\log n)$, სადაც $n = |A|$ (შედარებისა და სიმრავლის მარცხენა ან მარჯვენა ნაწილის აღების ოპერაციები თითო ბიჯად ჩათვალეთ). რა არის ძებნის ამოცანის დროის ქვედა ზღვარი?

იგივე პრინციპზეა აგებული დალაგებულ სიმრავლეში ***ელემენტის ჩამატების ამოცანა:*** მოცემულ დალაგებულ მიმდევრობაში A უნდა ჩავამატოთ ახალი ელემენტი c ისე, რომ მიღებული მიმდევრობაც დალაგებული იყოს.

განსხვავება ისაა, რომ თუ პოვნის ამოცანაში ვეძებდით ელემენტს $a_i = c$, აქ ვეძებთ ორ ერთმანეთის მიყოლებით მყოფ ელემენტს a_i, a_{i+1} ისეთს, რომ $a_i \leq c \leq a_{i+1}$ და c ამ ორ ელემენტს შორის უნდა ჩავსვათ (ცხადია, თუ $c < a_1$ან $c > a_n$, ახალი ელემენტი სიის თავში ან შესაბამისად ბოლოში უნდა ჩაისვას). თვალსაჩინოებისათვის მოვიყვანოთ შემდეგი მაგალითი:

<u>მაგალითი:</u>

საწყისი მონაცემი: $A = (6, 8, 9, 11, 12, 13, 17)$, $c = 7$.

$a_{i-1} \leq c \leq a_i$?	$7 \leq 6$? არა;		$7 \leq 8$? კი;		
A	$([6], 8, 9, 11, 12, 13, 17)$	$\rightarrow$	$(6, [8], 9, 11, 12, 13, 17)$	$\rightarrow$	პასუხი: $A = (6, [7], 8, 9, 11, 12, 13, 17)$.

<u>მაგალითი:</u>

საწყისი მონაცემი: $A = (6, 8, 9, 11, 12, 13, 17)$, $c = 20$.

$a_{i-1} \leq c \leq a_i$?	$20 \leq 6$? არა;		$20 \leq 8$? არა;		$20 \leq 9$? არა;	
	$([6], 8, 9, 11, 12, 13, 17)$	$\rightarrow$	$(6, [8], 9, 11, 12, 13, 17)$	$\rightarrow$	$(6, 8, [9], 11, 12, 13, 17)$	$\rightarrow$
$a_{i-1} \leq c \leq a_i$?	$20 \leq 11$? არა;		$20 \leq 12$? არა;		$20 \leq 13$? არა;	
	$(6, 8, 9, [11], 12, 13, 17)$	$\rightarrow$	$(6, 8, 9, 11, [12], 13, 17)$	$\rightarrow$	$(6, 8, 9, 11, 12, [13], 17)$	$\rightarrow$
$a_{i-1} \leq c \leq a_i$?	$20 \leq 17$? არა;					
	$(6, 8, 9, 11, 12, 13, [17])$	$\rightarrow$	პასუხი: $A = (6, 8, 9, 11, 12, 13, 17, [20])$			

როგორც ვხედავთ, იმის მიხედვით, თუ როგორი მონაცემი გვექნება, ბიჯების რაოდენობა შეიძლება არ იყოს დამოკიდებული მონაცემთა სიგრძეზე (როგორც პირველ მაგალითში), ან იყოს დაახლოებით იმდენივე, რამდენიც მონაცემთა სიგრძეა (მეორე მაგალითი). აქედან გამომდინარე, ამ მეთოდით ჩასმის ბიჯების რაოდენობის ქვედა და ზედა ზღვარი იქნება $\Omega(1)$ და $O(n)$ (სხვა სიტყვებით რომ ვთქვათ, მონაცემებისგან დამოუკიდებლად, დაგვჭირდება მინიმუმ ორი ბიჯი - შედარება და ჩასმა - ან მაქსიმუმ $n+1$ ბიჯი - ყოველ ელემნტთან შედარება და ბოლოს ჩასმა).

თუ გავითვალისწინებთ იმ ფაქტს, რომ A მიმდევრობა დალაგებულია, მაშინ შეგვიძლია იგივე მეთოდი გამოვიყენოთ, როგორც ელემენტის ძებნის ამოცანაში: ვეძებთ ისეთ ელემენტს a_i, რომლისთვისაც სრულდება პირობა $a_{i-1} \leq c \leq a_i$ ან $a_i \leq c \leq a_{i+1}$. თუ $c \leq a_1$ ან $c \geq a_n$, ახალი ელემენტი ემატება შესაბამისად თავში ან ბოლოში. თვალსაჩინოებისათვის განვიხილოთ შემდეგი მაგალითი:

საწყისი მონაცემი: $A = (6, 8, 9, 11, 12, 13, 17)$, $c = 20$.

$$
\begin{array}{llllll}
a_{i-1} \leq c \leq a_i? & 20 \leq 11?\ \text{არა;} & & 20 \leq 13?\ \text{არა;} & & \\
 & (\underbrace{6, 8, 9, [11], 12, 13, 17}) & \rightarrow & (6, 8, 9, 11, \underbrace{12, [13], 17}) & \rightarrow & \\
 & 20 \leq 17?\ \text{არა;} & & & & \\
 & (6, 8, 9, 11, 12, 13, \underbrace{[17]}) & \rightarrow & \text{პასუხი: } A = (6, 8, 9, 11, 12, 13, 17, [20]) & &
\end{array}
$$

ზოგადად ეს მეთოდი შემდეგი ალგორითმით შეიძლება ჩაიწეროს:

მოცემულობა: რაციონალური რიცხვებისგან შემდგარი, დალაგებული სასრული არაცარიელი მიმდევრობა $A = (a_1, a_2, ..., a_n)$ და რაციონალური რიცხვი c;

შედეგი: $A \cup \{c\}$ სიმრავლის ელემენტებისგან შემდგარი ***დალაგებული*** მიმდევრობა.

InsertFast($a_1, a_2, ..., a_n, c$ *)*
if($c \leq a_1$**)**
  $\{A = (c, a_1, a_2, ..., a_n)$; *return(A)* $\}$ /* თუ შესაძლებელია, პირველ ელემენტად ჩავსვათ */

if($c \geq a_n$**)**
  $\{A = (a_1, a_2, ..., a_n, c)$; *return(A)* $\}$ /* თუ შესაძლებელია, ბოლო ელემენტად ჩავსვათ */

$min = 1$;
$max = n$; /* დავადგინოთ საძიებელი ნაწილის საზღვრები */
do
$\{$

$i = \lceil \frac{min+max}{2} \rceil$; /* საძიებელი ნაწილის შუა ელემენტის ინდექსი */

if($a_{i-1} \le c \le a_i$)
 $\{A = (a_1, ..., a_{i-1}, c, a_i, ..., a_n)$; *return(A)* }

if($a_i \le c \le a_{i+1}$) /* თუ შესაძლებელია, ახალი ელემენტი საჭირო ადგილზე ჩავსვათ */
 $\{A = (a_1, ..., a_i, c, a_{i+1}, ..., a_n)$; *return(A)* }

if($c < a_i$)
 $max = i$; /* საძიებელი ნაწილის ახალი საზღვრები: მარცხენა ნახევარი */
 else $min = i$; /* საძიებელი ნაწილის ახალი საზღვრები: მარჯვენა ნახევარი */
}

სავარჯიშო 8.23: დაამტკიცეთ, რომ ზემოთ მოყვანილი ალგორითმი ყოველთვის შეჩერდება.

სავარჯიშო 8.24: დაამტკიცეთ ზემოთ მოყვანილი ალგორითმის სისწორე.

სავარჯიშო 8.25: დაამტკიცეთ, რომ ზემოთ მოყვანილი ალგორითმის დროის ქვედა ზღვარია $\Omega(1)$. რაზეა დამოკიდებული მისი ზედა ზღვარი?

აღსანიშნავია, რომ ეს ალგორითმი პირველზე (მიყოლებით ძებნა და მერე ჩასმა) გაცილებით უფრო სწრაფი შეიძლება იყოს (მაგალითად, $2^{10} = 1024$ ელემენტიან სიაში ჩამატებას მიყოლებითი ალგორითმი დაახლოებით 2000 ბიჯს ***შეიძლება*** ანდომებდეს, ხოლო *InsertFast* ალგორითმი (მონაცემთა სათანადო სტრუქტურის შერჩევისას) დაახლოებით 70 ბიჯში ამთავრებდეს მუშაობას.
ეს იმას არ უნდა ნიშნავდეს, რომ მიმდევრობითი ალგორითმი ***ყოველთვის*** უფრო ნელი იქნება - გარკვეული მონაცემებისთვის იგი შეიძლება უფრო სწრაფიც იყოს, მაგრამ ყველაზე ცუდ შემთხვევაში *InsertFast* ალგორითმი გაცილებით უფრო სწრაფია.

სავარჯიშო 8.26: დაამტკიცეთ, რომ $A = (1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$ და $c = 3$ მონაცემისათვის მიმდევრობით ჩასმის ალგორითმი უფრო სწრაფია, ვიდრე *InsertFast*.

იმისათვის, რომ მიმდევრობაში ელემენტი ჩავამატოთ, საჭიროა ალგორითმი, რომელიც მონაცემთა სტრუქტურაზეა დამოკიდებული. თუ გვაქვს ელემენტების ვექტორი, საჭირო ადგილზე ჩასამატებლად მის მარჯვნივ მყოფი ელემენტები უნდა გადავანაცვლოთ თითო პოზიციით მარჯვნივ, რომ ადგილი „გათავისუფლდეს":

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.	1	2	4	5	6	

$\longrightarrow$

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.	1	2	4	5		6

$\longrightarrow$

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.	1	2	4		5	6

$\longrightarrow$

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.	1	2		4	5	6

$\longrightarrow$

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.	1		2	4	5	6

$\longrightarrow$

ცვლადი	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
მნიშვნ.		1	2	4	5	6

ყველაზე ცუდ შემთხვევაში, თუ მოცემული გვაქვს n ელემენტიანი ვექტორი და გვჭირდება პირველ პოზიციაზე ჩამატება, მოგვიწევს n გადანაცვლების ოპერაციის ჩატარება. აქედან გამომდინარე, ვექტორში ჩამატების ოპერაცია ყველაზე კარგ შემთხვევაში $\Omega(1)$, ყველაზე ცუდ შემთხვევაში კი $O(n)$ ოპერაციის ჩატარებას მოითხოვს (მისი მუშაობის დრო წრფივია).

სავარჯიშო 8.27: ზემოთ აღწერილი მეთოდის დახმარებით დაწერეთ ალგორითმი *Insert(* A, k, c *)*, რომელიც $A = (a_1, a_2, ..., a_n)$ მიმდევრობის k პოზიციაზე c ელემენტს ჩაამატებს და პასუხად $(a_1, ..., a_{k-1}, c, a_k, ..., a_n)$ მიმდევრობას მოგვცემს.

სავარჯიშო 8.28: დაწერეთ ალგორითმი *InsertList(* A, k, c *)*, რომელიც A ბმული სიის k პოზიციაზე c ელემენტს ჩაამატებს.

თავი 9

დალაგების მარტივი ალგორითმები

9.1 დალაგება ამორჩევით

თუ მოცემულია დასალაგებელი მიმდევრობა A, შეგვიძლია შემდეგნაირად მოვიქცეთ:

SelectSort(მონაცემი: რაციონალური რიცხვებისგან შემდგარი სასრული მიმდევრობა A *)*

- სანამ A მიმდევრობა ცარიელი არაა, გაიმეორე შემდეგი ოპერაციები:

  A მიმდევრობაში იპოვნე მინიმალური ელემენტი, ამოშალე იქიდან და ჩართე B მიმდევრობაში

$$\begin{array}{lllllllllll} A & (5,1,13,-8,17) & \rightarrow & (5,1,13,17) & \rightarrow & (5,13,17) & \rightarrow & (13,17) & \rightarrow & (17) & \rightarrow & () \\ B & () & & (-8) & & (-8,1) & & (-8,1,5) & & (-8,1,5,13) & & (-8,1,5,13,17) \end{array}$$

სავარჯიშო 9.29: იგივე ალგორითმი ჩაწერეთ რეკურსიულად.

სავარჯიშო 9.30: დაამტკიცეთ, რომ ამ ალგორითმის დასრულების შემდეგ B მიმდევრობაში დალაგებული A სიმრავლე ჩაიწერება.

ზოგადად, ალგორითმის ბიჯების რაოდენობა დამოკიდებულია მისი ***მონაცემების სიგრძეზე***. ჩვენს მაგალითში კი მონაცემის სიგრძე დამოკიდებულია A მიმდევრობაში შემავალ ელემენტთა რაოდენობაზე და თითოეული რიცხვის სიგრძეზე. თუ $|A| = n$ და A მიმდევრობის თითოეული ელემენტი k ბიტისაგან შედგება, მონაცემთა სიგრძე იქნება $n \cdot k$.
ახლა კი გამოვითვალოთ, თუ რამდენ ბიჯს მოანდომებს *SelectSort* ალგორითმი მონაცემად მიღებული მიმდევრობის დალაგებას. ამისათვის უნდა გვქონდეს სიმრავლეში მინიმალური ელემენტის პოვნის ალგორითმი, რომელიც შემდეგნაირად შეიძლება ჩაიწეროს:

Min(მონაცემი: რაციონალური რიცხვებისგან შემდგარი სასრული მიმდევრობა $A = (a_1, a_2, ..., a_n)$ *)*
$min = a_1;$
$for(\ i = 2; i \leq n; i{+}{+}\ )$
{
$if(\ a_i < min)$
$min = a_i;$
}

სავარჯიშო 9.31: დაამტკიცეთ ზემოთ მოყვანილი ალგორითმის სისწორე.

Min ალგორითმის ბიჯების რაოდენობა შემდეგნაირად გამოითვლება:

min ცვლადისათვის A მიმდევრობის პირველი ელემენტის მინიჭება: 1 ბიჯი;
for ციკლი n-ჯერ მეორდება; მასში ხდება ერთი შედარება $a_i < min$ და თუ ეს ჭეშმარიტია, ერთი მინიჭება $min = a_i$.
აქედან გამომდინარე, ყველაზე ცუდ შემთხვევაში *for* ციკლში გვექნება თითო შედარება და თითო მინიჭება, სულ 2 ოპერაცია. ყველაზე კარგ შემთხვევაში კი მხოლოდ ერთი შემოწმება და არც ერთი მინიჭება.

ესე იგი, სულ ბიჯების რაოდენობა იქნება:

საუკეთესო შემთხვევაში: $n+1$;
უარეს შემთხვევაში: $2n+1$.

აღსანიშნავია, რომ ბიჯების გამოთვლის დროს ჩვენ არ გავითვალისწინეთ ციკლის მთვლელის შედარებისა და გაზრდის ოპერაციები, ციკლიდან გამოსვლის ოპერაცია და სხვა ტექნიკური დეტალები, რომლებიც ჩვეულებრივ ბიჯების გამოთვლისას არ ითვლება ხოლმე. სწორეთ ასეთი დეტალების უგულებელყოფის მიზნითაა შემოღებული ზედა და ქვედა ზღვრის შეფასება O და Ω აღნიშნვით.

ამ ალგორითმის ქვედა და ზედა ზღვარი შეიძლება გამოისახოს როგორც $\Omega(n+1) = \Omega(n)$ და $O(2n+1) = O(n)$. აქედან გამომდინარე, შეიძლება შეფასდეს ზუსტი ზღვარი $\Theta(n)$.
ყოველივე თქმულიდან ვიღებთ: $T(Min) \in \Theta(n)$.
თუ ალგორითმის გამოთვლის დროის ზედა ზღვარია $O(n)$, მაშინ იტყვიან, რომ მისი დროის ზრდის რიგი არის წრფივი.

SelectSort ალგორითმის მსვლელობაში საჭიროა აგრეთვე ელემენტის ამოშლა. თუ A მიმდევრობა წარმოდგენილი იქნება როგორც ბმული სია, ამის მოხერხება 5 ბიჯში შეიძლება (იხ. წინა კურსის მასალაში ბმული სიების ოპერაციები).
აქედან გამომდინარე, A სიიდან x პოზიციაზე მდგომი ელემენტის ამოშლის $Erase(A,x)$ ალგორითმის მოქმედების დრო იქნება $T(Erase(A,x)) \in \Theta(5) = \Theta(1)$.

ადვილი დასანახია, რომ *SelectSort* ალგორითმის მუშაობის დროს ჯერ უდა შემოწმდეს, ცარიელია თუ არა A სიმრავლე (ერთი ბიჯი), შემდეგ (თუ A ცარიელი არაა) მასში მოიძებნოს მინიმალური ელემენტი ($c_1 \cdot n$ ბიჯი), ბოლოს ეს ელემენტი ამოიშალოს და ჩაიწეროს B მიმდევრობაში (ორივე ოპერაცია ჯამში c_2 ბიჯი):

$$T(SelectSort(A)) = T(SelectSort(A - \{A \text{ სიმრავლის მინიმალური ელემენტი}\})) + 1 + c_1 \cdot n + c_2,$$

თუ ჩავთვლით, რომ $|A| = n$. რეკურსიის გახნის შედეგად (იმის გათვალისწინებით, რომ
$|A - \{A \text{ სიმრავლის მინიმალური ელემენტი}\}| = n-1$, მივიღებთ:

$$T(SelectSort(A)) = T(SelectSort(\emptyset)) + c_1(n+(n-1)+(n-2)+\cdots+1) + (c_2+1)n = 1 + c_1\frac{n(n+1)}{2} + (c_2+1)n \in O(n^2).$$

ამ შემთხვევაში იტყვიან, რომ *SelectSort* ალგორითმის დროის ზრდის ზედა ზღვარია (მუშაობის დროა) $O(n^2)$, ან მისი დროის ზრდის ზედა ზღვარი (მუშაობის დრო) კვადრატულია.

სავარჯიშო 9.32: გამოითვალეთ, რა იქნება პროგრამის დროის ზედა ზღვარი, თუ ბმული სიის ნაცვლად ავიღებთ ვექტორს და ელემენტის ამოშლის შემდეგ ცარიელი ადგილის „ამოვსება" (მარცხენა ან მარჯვენა ელემენტების თითო პოზიციით გადაწევა) დაგვჭირდება.

სავარჯიშო 9.33: დაწერეთ პროგრამა, რომელიც ამორჩევით დალაგების ალგორითმის მიხედვით A მიმდევრობას დაალაგებს ისე, რომ *არ* გამოიყენებს მეორე მიმდევრობას: ყოველ ჯერზე არჩეული მინიმალური ელემენტი ისევე A მიმდევრობაში ჩაწეროს. ამ პროგრამის დროის ზედა ზღვარიც კვადრატული უნდა იყოს.

სავარჯიშო 9.34: რა განსხვავება იქნება გამოთვლის დროში, თუ წინა სავარჯიშოში მოყვანილი ამოცანისათვის ალგორითმს ჯერ ბმული სიის, შემდეგ კი ვექტორის გამოყენებით დავწერთ? შეიცვლება თუ არა დროის ზრდის რიგი? შეიცვლება თუ არა რეალური გამოთვლის დრო?

9.2 დალაგება ჩადგმით:

მოცემული A მიმდევრობის დალაგება შემდეგი მეთოდითაც შეიძლება:

- სანამ A მიმდევრობა ცარიელი არაა, გაიმეორე შემდეგი ოპერაციები:

აირჩიე A მიმდევრობის პირველი ელემენტი, ამოშალე იქიდან და ჩართე B მიმდევრობაში (რომელიც დალაგებულია) საჭირო ადგილზე ისე, რომ მიღებული მიმდევრობა დალაგებული იყოს.

ეს მეთოდი ფართოდ გამოიყენება პრაქტიკაში: მაგალითად, კარტის თამაშის დროს რიგ-რიგობით აღებულ კარტს „ვახარისხებთ" - ახალს უკვე დალაგებულ მიმდევრობაში საჭირო ადგილზე ვსვამთ.

$$\begin{array}{lllllllllll} A & (5,13,8,1,7) & \rightarrow & (13,8,1,7) & \rightarrow & (8,1,7) & \rightarrow & (1,7) & \rightarrow & (7) & \rightarrow & () \\ B & () & & ([5]) & & (5,[13]) & & (5,[8],13) & & ([1],5,8,13) & & (1,5,[7],8,13) \end{array}$$

აღსანიშნავია, რომ დალაგების ზემოთ აღნიშნული მეთოდები რაღაცა თვალსაზრისით ერთმანეთის „შებრუნებულია": პირველ მეთოდში ჯერ A მიმდევრობის საჭირო ადგილიდან ელემენტს ვარჩევთ და მერე მას B მიმდევრობის თავში ვსვამთ. მეორე მეთოდში კი ჯერ A მიმდევრობის თავიდან პირველ ელემენტს ვიღებთ და მას ვსვამთ B მიმდევრობაში საჭირო ადგილზე.

რადგან ჩასმით დალაგების ალგორითმში A სიმრავლიდან ვიღებთ ერთ ელემენტს და მას B სიმრავლეში ვსვამთ ისე, რომ ეს უკანასკნელი დალაგებული იყოს, შეგვიძლია გამოვიყენოთ *InsertFast* ალგორითმიც:

მოცემულობა: რაციონალური რიცხვებისგან შემდგარი სასრული მიმდევრობა $A = (a_1, a_2, ..., a_n)$;

შედეგი: A მიმდევრობის ელემენტებისგან შემდგარი დალაგებული მიმდევრობა B.

InsertionSort(A)
$B = \emptyset$;
while(A სიმრავლე ცარიელი არაა)
{
$a = A$ მიმდევრობის პირველი ელემენტი;
InsertFast(B, a) /* გამოვიყენოთ *InsertFast* ალგორითმი B სიმრავლეში ახალი ელემენტის ჩასამატებლად */
$A = A \backslash \{a\}$ /* A მიმდევრობიდან ამოვაგდეთ პირველი ელემენტი */
}

InsertionSort ალგორითმის ბიჯების რაოდენობა შემდეგნაირად შეიძლება დავითვალოთ:
while ციკლი n-ჯერ მოერდება; მასში კი შემდეგი ოპერაციები სრულდება:

- A მიმდევრობის პირველი ელემენტის გამოყოფა (1 ბიჯი);
- B მიმდევრობაში a ელემენტის საჭირო ადგილზე ჩამატება *InsertionSort* ალგორითმის გამოყენებით . $T(\textit{InsertFast}(B, a)) = O(\log |B|) = O(\log n)$ ბიჯი;
- A მიმდევრობიდან პირველი ელემენტის ამოგდება (1 ბიჯი).

აქედან გამომდინარე, სულ გვექნება

$$T(\textit{InsertionSort}(A)) \leq (c \log n + 2) + (c \log(n-1) + 2) + \cdots + (c \log 1 + 2) = 2n + c(\log n + \log(n-1) + \cdots + \log 2 + \log 1)$$

და ვიღებთ:

$$T(\textit{InsertionSort}(A)) \in O(\log n + \log(n-1) + \cdots + \log 2 + \log 1) = O(n \log n).$$

სავარჯიშო 9.35: დაამტკიცეთ ტოლობა $O(\log n + \log(n-1) + \cdots + \log 2 + \log 1) = O(n \log n)$.

სავარჯიშო 9.36: ზემოთ მოყვანილი ალგორითმის რომელი ნაწილები განსაზღვრავენ მუშაობის დროის ფუნქციის ზედა ზღვარს (სხვა სიტყვებით რომ ვთქვათ, რომელი ბიჯების უგულებელყოფა შეიძლება O აღნიშნვაში)?

სავარჯიშო 9.37: მონაცემთა რა სტრუქტურა უნდა ავირჩიოთ, რომ ალგორითმის ბიჯების ზედა ზღვარი იყოს $O(n \log n)$? რა შეიძლება მოხდეს სხვა სტრუქტურის არჩევის შემდეგ?

9.3 სწრაფი დალაგება

თუ მოცემულია დასალაგებელი მიმდევრობა $A=(a_n,...,a_{n/2+1}a_{n/2},,...,a_1)$, დალაგების პროცედურის დაჩქარება შეიძლება მონაცემთა ორ ტოლ ნაწილად დაყოფით, მათი ცალ-ცალკე დალაგებით და დალაგებული ქვემიმდევრობების ერთმანეთში ისე შერწყმით, რომ მიღებული მიმდევრობა დალაგებული იყოს. ყოველივე ეს ერთ მაგალითზე განვიხილოთ:

მოცემულია დასალაგებელი მიმდევრობა $A=(3,7,1,15,12,2,13,6)$. მისი მონაცემები დავყოთ ორ ტოლ ნაწილად: $A=(A_2,A_1)$, სადაც $A_2=(3,7,1,15)$ და $A_1=(12,2,13,6)$. თითოეული ქვემიმდევრობის დალაგების შედეგად ვიღებთ: $Sort(A_2)=(1,3,7,15)$ და $Sort(A_1)=(2,6,12,13)$. ცხადია, რომ A მიმდევრობის მინიმალური ელემენტი ან $Sort(A_1)$, ან $Sort(A_2)$ მიმდევრობის მინიმალური (მარცხენა) ელემენტი იქნება. თუ ამ ელემენტს შესაბამისი მიმდევრობიდან ამოვშლით, დარჩენილი მიმდევრობებიდან მინიმალური ელემენტი A მიმდევრობის მეორე ელემენტი იქნება. ამ პროცესს ვაგრძელებთ მანამ, სანამ ერთ-ერთი მიმდევრობა არ დაცარიელდება, რის შემდეგაც არაცარიელ მიმდევრობას პასუხს მივაწერთ:

A_1	$([1],3,14,15)$	$(3,14,15)$	$([3],14,15)$	$(14,15)$	$(14,15)$	$(14,15)$	$(14,15)$
A_2	$(2,6,12,13)$	$([2],6,12,13)$	$(6,12,13)$	$([6],12,13)$	$([12],13)$	$([13])$	$(\ )$
B	$(\ )$	(1)	$(1,2)$	$(1,2,3)$	$(1,2,3,6)$	$(1,2,3,6,12)$	$(1,2,3,6,12,13)$

საბოლოო პასუხი: $(B,A_1)=(1,2,3,6,12,13,14,15)$

ყოველივე ეს შემდეგი ალგორითმით შეიძლება ჩაიწეროს:

საწყისი მონაცემი: ორ (თითქმის) ტოლ ნაწილად დაყოფილი მიმდევრობა $A=(A_2,A_1)=(a_n,...,a_{\frac{n}{2}+1},a_{\frac{n}{2}},...,a_1)$, $a_i\le a_j$, $a_{i+\frac{n}{2}}\le a_{j+\frac{n}{2}}$, $1\le i<j\le\frac{n}{2}$.

```
MergeSort(A)
if( A მიმდევრობა ერთ ელემენტიანია ) return(A)    /* პირდაპირ ერთი ელემენტი დააბრუნე */
B1 = MergeSort(A1);                               /* დაალაგე მიმდევრობის ორივე ნაწილი */
B2 = MergeSort(A2);
Merge(B1, B2);                                    /* შეურიე დალაგებული ნახევრები ისე, რომ მიღებული მიმდევრობა
                                                     დალაგებული იყოს */
```

ზემოთ მოყვანილ ფსევდო კოდში გამოყენებულია ქვეპროგრამა $Merge$, რომელიც შემდეგნაირად შეიძლება ჩაიწეროს:

საწყისი მონაცემი: ორი დალაგებული მიმდევრობა $A=(a_n,...,a_1)$ და $B=(b_m,...,b_1)$.

```
Merge(A, B)
C = ( );
do
{
  if( A მიმდევრობა ცარიელია ) return( (C, B) );
  if( B მიმდევრობა ცარიელია ) return( (C, A) );
  if( A მიმდევრობის მინიმალური ელემენტი < B მიმდევრობის მინიმალური ელემენტი )
    {
     C = ( C, A მიმდევრობის მინიმალური ელემენტი);
     A მიმდევრობიდან ამოაგდე მინიმალური ელემენტი;
    }
  else
    {
     C = ( C, B მიმდევრობის მინიმალური ელემენტი);
     B მიმდევრობიდან ამოაგდე მინიმალური ელემენტი;
    }
}
```

<u>სავარჯიშო 9.38:</u> ინდუქციის გამოყენებით დაამტკიცეთ მოყვანილი ალგორითმების სისწორე.

სავარჯიშო 9.39: დაამტკიცეთ, რომ თუ $|A| + |B| = n$, მაშინ $T(Merge(A, B)) \in O(n)$ და იყენებს არა უმეტეს $n - 1$ ელემენტების შედარებას.

$MergeSort$ ალგორითმის დროის ზრდის რიგის შეფასება შემდეგნაირად შეიძლება:

თეორემა 9.1: $MergeSort$ ალგორითმი იყენებს მაქსიმუმ $\lceil n \log n \rceil$ შედარების ოპერაციას და მისი ბიჯების მაქსიმალური რაოდენობა ეკუთვნის $O(n \log n)$ სიმრავლეს.

დამტკიცება: თუ $MergeSort$ ალგორითმი n სიგრძის მიმდევრობას ალაგებს, მის მიერ გამოყენებულ შედარების ოპერაციათა მაქსიმალური რაოდენობა აღვნიშნოთ როგორც $C(n)$. რა თქმა უნდა, $C(1) = 0$ და ალგორითმის ანალიზითა და $Merge$ ფუნქციის შედარების ოპერაციათა რაოდენობის გამოთვლით ვიღებთ:

$$C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + (n - 1) = 2C(\lceil n/2 \rceil) + (n - 1).$$

რეკურსიის გახსნის შედეგად ვიღებთ:

$$C(n) = 2C(\lceil n/2 \rceil) + (n-1) = 2^{\lceil \log n \rceil} + (n + n/2 + n/4 + \cdots 1) - \log n = n + n(1/2 + 1/4 + \cdots + 1/2^{\log n}) - \log n \leq \lceil n \log n \rceil.$$

სავარჯიშო 9.40: მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ, რომ თუ $n > 1$,
$n + n(1/2 + 1/4 + \cdots + 1/2^{\log n}) - \log n \leq \lceil n \log n \rceil$.

სავარჯიშო 9.41: დაამტკიცეთ, რომ $MergeSort$ ალგორითმის დროის ზრდის რიგი იქნება $O(n \log n)$ (ჯერ დაამტკიცეთ, რომ ამ ალგორითმის მუშაობის დრო დიდად არ აღემატება შედარების ოპერაციათა რიცხვს და აქედან გამოიტანეთ დასკვნა).

სავარჯიშო 9.42: დაწერეთ პროგრამა, რომელიც $MergeSort$ ალგორითმის ბმულ სიებზე რეალიზაცია იქნება.

სავარჯიშო 9.43: დაწერეთ ალგორითმი, რომლის საშუალებითაც n ელემენტიან დალაგებულ მიმდევრობაში k ელემენტს $O(k \log k + n)$ დროში ჩავსვამთ.

ახლა კი განვიხილოთ მეთოდი, რომელშიც დალაგების „რთული ნაწილი" რეკურსიულ გამოძახებამდე ხდება:

მოცემულია დასალაგებელი მიმდევრობა $A = (a_n, ..., a_1)$. თავიდან ვირჩევთ მიმდევრობის ერთ-ერთ (მაგალითად, პირველ) ელემენტს $a = a_1$ და გამოვყოფთ სამ ნაწილს: $C_1 = \{a_i | a_i < a\}, C_2 = \{a_i | a_i = a\}, C_3 = \{a_i | a_i > a\}$. სიტყვებით რომ ვთქვათ, პირველი სიმრავლე შედგება A მიმდევრობის ყველა იმ ელემენტისაგან, რომელიც არჩეულ ელემენტზე ნაკლებია, მეორე - ისეთებისგან, რომელიც არჩეული ელემენტის ტოლია და მესამე - ისეთებისაგან, რომლებიც არჩეულ ელემენტზე მეტია. ცხადია, რომ თუ შემდგომ ეტაპზე პირველ და მესამე სიმრავლეს ცალ-ცალკე დავალაგებთ, მაშინ დალაგებული A სიმრავლე იქნება:

$$Sort(A) = (Sort(C_1), C_2, Sort(C_3)).$$

დალაგების ამ მეთოდს $QuickSort$ ეწოდება, რომელიც ფართოდ გამოიყენება პრაქტიკაში, იმის და მიუხედავად, რომ მისი მაქსიმალური ბიჯების ზრდის რიგი კვადრატული შეიძლება იყოს გარკვეული მონაცემებისათვის: $T(QuickSort(A)) \in O(n^2)$, სადაც $|A| = n$.

საწყისი მონაცემი: $A = (a_n, ..., a_1)$ რიცხვთა მიმდევრობა.

$Quicksort(A)$
if($|A| = 1$*)* $return(A)$;
ნებისმიერი მეთოდით აირჩიე ერთი ელემენტი $a \in A$;
ააგე სიმრავლეები $C_1 = \{a_i | a_i < a\}, C_2 = \{a_i | a_i = a\}, C_3 = \{a_i | a_i > a\}$;
$return((QuickSort(C_1), C_2, QuickSort(C_3))$;

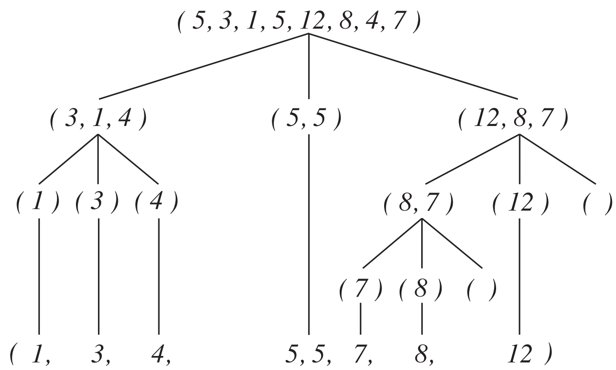
აღსანიშნავია, რომ საწყისი a ელემენტის არჩევა შეიძლება ნებისმიერი მეთოდით: ან შემთხვევით, ან ფიქსირებული (მაგალითად, პირველი, ან ბოლო, ან სხვა) ელემენტის, რაც მუდმივ დროში შეიძლება. ამას გარდა, C_1, C_2

და C_3 სიმრავლეების აგება წრფივ დროში შეიძლება, ისევე, როგორც საბოლოო პასუხის გამოტანა იმ პირობით, თუ ეს ქვესიმრავლეები დალაგებულია.
აქედან გამომდინარე, ვიღებთ ბიჯების რაოდენობის შეფასების შემდეგ რეკურსიულ ფორმულას:

$$T(QuickSort(A)) = O(1) + O(n) + T(QuickSort(C_1)) + T(QuickSort(C_3)).$$

სამაგიეროდ უმეტეს შემთხვევაში ეს ალგორითმი $O(n \log n)$ დროში ალაგებს მონაცემებს, ზუსტად კი მისი ბიჯების რაოდენობა უმეტეს შემთხვევაში გვექნება $T(Quicksort(A)) < 2n \lg n$, სადაც $|A| = n$.

მაგალითისათვის განვიხილოთ $A = (5, 3, 1, 5, 12, 8, 4, 7)$ (ნახ. 9.1). პირველ რიგში ვიღებთ პირველ ელემენტს $a = 5$ და ვაგებთ $C_1 = \{3, 1, 4\}$; $C_2 = \{5, 5\}$, $C_3 = \{12, 8, 7\}$ სიმრავლეებს. C_2 სიმრავლის ელემენტები პირდაპირ უნდა გავიდეს პასუხში, ხოლო C_1 და C_3 იგივე პრინციპით უნდა დაიყოს (ნახაზის მეორე სტრიქონი).
ერთ ელემენტიანი ქვესიმრავლეები პირდაპირ უნდა გამოვიტანოთ პასუხში, ხოლო სულ ცოტა ორ ელემენტიანი (როგორიცაა $\{8, 7\}$) იგივე პრინციპით უნდა დაიშალოს. აღსანიშნავია ის ფაქტი, რომ ცარიელი სიმრავლეების პასუხში არ უნდა გამოვიტანოთ.



ნახ. 9.1: $QuickSort$ ალგორითმის გამოთვლის სქემა

სავარჯიშო 9.44: დაამტკიცეთ, რომ არსებობს ისეთი საწყისი მიმდევრობა A, რომლის დალაგებასაც $QuickSort$ ალგორითმი კვადრატულ დროს მოანდომებს.

სავარჯიშო 9.45: დაწერეთ ალგორითმი, რომელიც ***მიმდევრობით*** მოსული $(a_1, a_2, ..., a_n)$ ელემენტების სიიდან k-ურ ელემენტს ამოარჩევს (მინიმალური ელემენტი პირველია, მეორე იქნება ის ელემენტი, რომელიც მინიმალურზე ნაკლები ან ტოლია და ა.შ.). ამ ალგორითმის მეხსიერების ხარჯვის ზედა ზღვარი უნდა იყოს $O(k)$ (მეხსიერების ხარჯვის ფუნქციის ზედა ზღვარი ბიჯების რაოდენობის ზედა ზღვრის ანალოგიურად გამოითვლება).

9.4 დალაგების ამოცანის ქვედა ზღვარი

აქამდე ალგორითმების ანალიზის დროს ჩვენ მათ ზედა და ქვედა ზღვარს ვითვლიდით. თუ რაიმე ალგორითმი გარკვეულ ამოცანას ჭრის (მაგ. მონაცემთა მიმდევრობის დალაგებას) მისი ზედა ზღვარი გვეუბნება იმას, თუ რამდენად სწრაფად შეიძლება ამ ამოცანის გადაჭრა. იმის დადგენა, თუ რამდენ დროს მოანდომებს ***ყველაზე სწრაფი ალგორითმი*** მოცემული ამოცანის გადაჭრას, საკმაოდ ძნელია და მას ამოცანის ქვედა ზღვრის დადგენა ეწოდება (არ აგერიოთ ***ალგორითმის*** ქვედა ზღვარში, რომელიც გვიჩვენებს, სულ ცოტა რამდენი ბიჯი ჭირდება ამ კონკრეტულ ალგორითმს ყველაზე კარგ შემთხვევაში). მოცემული ამოცანის ქვედა ზღვარი გვეუბნება, რომ ვერავინ დაწერს ისეთ ალგორითმს, რომლის მუშაობის მაქსიმალური დრო ამ ქვედა ზღვარზე სწრაფი იქნება. არაა გასაკვირი, რომ იმის დადგენა, რომ რაღაცის გაკეთებას ვერავინ შეძლებს, საკმაოდ რთული პროცესია და არ არსებობს ერთი მეთოდი, რომლითაც ამას ყველა ამოცანისათვის გავაკეთებდით: ამოცანათა სხვადასხვა ჯგუფს სხვადასხვანაირი მიდგომა ჭირდება.

აქ ჩვენ დავამტკიცებთ, რომ შედარების ოპერაციებზე დაფუძნებული ძებნის ამოცანის ქვედა ზღვარია $\Omega(n \log n)$. სხვა სიტყვებით რომ ვთქვათ, ვერავინ დაწერს ისეთ ალგორითმს, რომელიც შედარების ოპერაციებზე იქნება დაფუძნებული (როგორებსაც ჩვენ აქამდე განვიხილავდით) და რომლის ბიჯების ზედა ზღვრის (მაქსიმალური რაოდენობის) ზრდის რიგი იქნება უკეთესი, ვიდრე $O(n \log n)$.

ამისათვის შემოვიღოთ შემდეგი

განმარტება 9.1: მოცემული $A = (a_1, a_2, ..., a_n)$ მიმდევრობის ***პერმუტაცია*** მისი ელემენტების გადანაცვლებას ეწოდება (ლათინური სიტყვიდან "permutare" - გაცვლა).

მაგალითად, $A = (a_1, a_2, a_3, a_4, a_5)$ მიმდევრობის ***ერთ-ერთი*** პერმუტაციის შედეგია $(a_5, a_1, a_2, a_4, a_3)$, ან $(a_1, a_2, a_3, a_5, a_4)$, ან $(a_3, a_2, a_5, a_1, a_4)$. თვით ეს მიმდევრობაც $(a_1, a_2, a_3, a_4, a_5)$ A მიმდევრობის პერმუტაციის შედეგია, რომელიც ყველა ელემენტს თავის ადგილზე ტოვებს.

პერმუტაციათა აღწერის სხვადასხვა მეთოდი არსებობს, მაგრამ ხშირად მათ რიცხვთა მიმდევრობით გამოსახავენ ხოლმე, რომელიც გვიჩვენებს, თუ რომელ პოზიციაზე უნდა გადავიდეს საწყისი მიმდევრობის ესა თუ ის ელემენტი. მაგალითად, $\sigma = (2, 3, 5, 4, 1)$ პერმუტაციით A მიმდევრობა გადავა $(a_5, a_1, a_2, a_4, a_3)$ მიმდევრობაში: მისი პირველი ელემენტი გადავა მეორე ადგილზე, მეორე - მესამეზე, მესამე - მეხუთეზე, მეოთხე ისევე მეოთხეზე დარჩება და მეხუთე გადავა პირველ ადგილზე. ანალოგიურად, $\rho = (4, 2, 1, 5, 3)$ პერმუტაციით A მიმდევრობის ელემენტები გადავა $(a_3, a_2, a_5, a_1, a_4)$ მიმდევრობაში, ხოლო $(1, 2, 3, 4, 5)$ კი საწყის მიმდევრობას უცვლელს დატოვებს.

სავარჯიშო 9.46: მათემატიკური ინდუქციის გამოყენებით დაამტკიცეთ, რომ n ელემენტიანი მიმდევრობის $n!$ სხვადასხვა პერმუტაცია არსებობს.

სავარჯიშო 9.47: რომელი პერმუტაციებით მიიღება საწყისი (a, b, c, d, e, f, g, h) მიმდევრობიდან *(ა)* (a, d, h, b, c, f, e, g), *(ბ)* (d, a, h, b, g, f, e, e) მიმდევრობები?

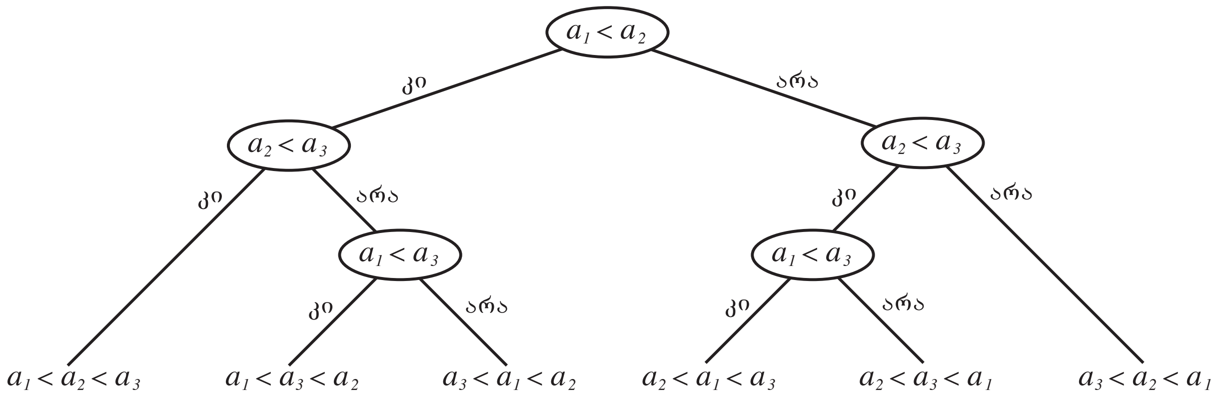
სავარჯიშო 9.48: რა მიმდევრობებში გადაიყვანს საწყისი (a, b, c, d, e, f, g, h) მიმდევრობიდან *(ა)* $(1, 2, 4, 3, 6, 5, 7, 8)$, *(ბ)* $(8, 7, 6, 5, 4, 3, 2, 1)$ პერმუტაცია?

აღსანიშნავია, რომ დალაგებაც საწყისი მიმდევრობის პერმუტაციაა. ფაქტიურად, დალაგების ალგორითმის ამოცანაა, რაც შეიძლება სწრაფად გააანალიზოს შემოსული მონაცემი და შესაბამისი პერმუტაციით დალაგებულ მიმდევრობაში გადაიყვანოს.
ხშირად დალაგების ალგორითმისათვის მონაცემთა თანმიმდევრობის შესწავლის ერთად-ერთი საშუალება მხოლოდ მისი ელემენტების შედარებაა. მაგალითად, თუ სამ ელემენტიან მიმდევრობაში დავასკვენით, რომ $a_2 < a_1$ და $a_1 < a_3$, მაშინ დალაგების პერმუტაცია იქნება $(2, 1, 3)$ და დალაგებული სიმრავლე გამოვა (a_2, a_1, a_3). ამ შემთხვევაში იტყვიან, რომ დალაგების ეს ალგორითმი შედარების ოპერაციებზეა აგებული. აქამდე განხილული ყველა ალგორითმი ასეთი იყო.
ზემოთ თქმულიდან გამომდინარეობს, რომ ჩვენ შეგვიძლია პერმუტაციების გამოთვლის ხის აგება. იმავე სამ ელემენტიანი მიმდევრობის მაგალითზე შეგვიძლია ავაგოთ გამოთვლის ხე, რომელიც მოყვანილია ნახაზში 9.2.

ასეთ სტრუქტურას „გადაწყვეტილების ორობითი ხე" ეწოდება. „ორობითი" იმიტომ, რომ ყოველ კვანძს (ფოთლების გარდა) ზუსტად ორი შვილი ყავს, ხოლო „გადაწყვეტილების" იმიტომ, რომ გამოთვლის დროს რაღაცა შეკითხვას პასუხი უნდა გავცეთ (გადაწყვეტილება მივიღოთ) და შემდეგ შესაბამის გზას გავყვეთ. რადგან ამ კონკრეტულ მაგალითში დასმულ შეკითხვაზე ($a_i < a_j$?) ორი სხვადასხვა პასუხია შესაძლებელი, ეს სქემა ორობით ხეში კარგად ჯდება.

ამ პრინციპით ნებისმიერი $A = (a_1, ..., a_n)$ მიმდევრობის ორობითი გადაწყვეტილების ხის აგება შეიძლება, რომელსაც ფოთლებში A მიმდევრობის ყველა პერმუტაცია ექნება. ცხადია, რომ ნებისმიერ ალგორითმს, რომელიც შედარებებზეა აგებული, დალაგების დროს ასეთი ხის ზემოდან ქვემოთ გასვლა მოუწევს და პასუხი (დალაგება) ის შესაბამისი პერმუტაცია იქნება, რომელსაც ხის ფოთოლში მივაღწევთ.

მტკიცებათა სიმარტივისათვის დავუშვათ, რომ დასალაგებელი მიმდევრობის ყველა ელემენტი ერთმანეთისაგან განსხვავებულია (თუ ორი ან რამოდენიმე ელემენტი ერთმანეთის ტოლია, ამ შემთხვევისათვისაც შეიძლება ანალოგიური თეორემების დამტკიცება).



ნახ. 9.2: სამ ელემენტიანი პერმუტაციის ხე

მნიშვნელოვანია შემდეგი

ლემა 9.3: თუ μ და σ ერთი და იგივე მიმდევრობის სხვადასხვა პერმუტაციაა, მაშინ შესაბამის ორობით გადაწყვეტილების ხეს ორი სხვადასხვა ფოთოლი $\ell_\mu \neq \ell\sigma$ ექნება, რომელიც ამ პერმუტაციებს შეესაბამება.

სავარჯიშო 9.49: საწინააღმდეგოს დაშვებით დაამტკიცეთ ზემოთ მოყვანილი ლემა.

ეს თითქოს და ელემენტარული ლება გადამწყვეტია დალაგების ალგორითმის ქვედა ზღვრის გამოთვლაში, რადგან აქედან გამომდინარეობს, რომ ყოველი გადაწყვეტილების ორობითი ხე, რომელიც n მონაცემს ზრდადობის მიხედვით ალაგებს, აუცილებლად უნდა შეიცავდეს $n!$ ფოთოლს.

რადგან T სიღრმის ორობით ხეს მაქსიმუმ 2^T ფოთოლი შეიძლება ქონდეს, ვიღებთ:

$$2^T \geq n! \quad \text{და, აქედან გამომდინარე, } T \geq \log n!.$$

სავარჯიშო 9.50: მათემატიკურ ინდუქციაზე დაყრდნობით დაამტკიცეთ, რომ T სიღრმის ორობით ხეს მაქსიმუმ 2^T ფოთოლი შეიძლება ქონდეს.

ე.წ. სტირლინგის ფორმულის თანახმად, რომელიც ფართოდ გამოიყენება კომბინატორიკაში და ჩვენ აქ დაუმტკიცებლად მივიღებთ, გვაქვს:

$$T \geq \underbrace{\log n! \geq \log\left(\frac{n}{e}\right)^n}_{\text{სტირლინგის ფორმულა}} = n\log n - n\log e,$$

სადაც e ე.წ. ნატურალური ლოგარითმის ფუძე (ან, როგორც მას სხვანაირადაც უწოდებენ ეილერის რიცხვია) - მუდმივა $e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = \sum_{n=0}^{\infty}\frac{1}{n!} = 2,718281828459045235...$

შენიშვნა: თავისი სრული სახით სტირლინგის ფორმულა შემდეგნაირად გამოისახება:

$$\log n! \sim \log\left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi n},$$

რაც მარცხენა და მარჯვენა ნაწილში მოცემული ფუნქციების „მსგავსებას" ნიშნავს:

$$\lim_{n\to\infty}\frac{\log n!}{\log\left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi n}} = 1.$$

ეს ფორმულა იმითიცაა საინტერესო, რომ მასში მეცნიერების ორი უმნიშვნელოვანესი მუდმივა - π და e ერთად ფიგურირებს.

აქედან გამომდინარე, ზემოთ მოყვანილი გადაწყვეტილების ორობითი ხე, რომელიც საჭირო პერმუტაციამდე მიგვიყვანს, ***დაახლოებით*** $n \log n$ სიღრმისაა და ჩვენ დავამტკიცეთ

თეორემა 9.2: შედარების ოპერაციებზე აგებული დახარისხების ალგორითმის ქვედა ზღვარია $\Omega(n \log n)$. უფრო ზუსტად მისი გამოთვლა შეიძლება ფორმულით $n \log n - O(n)$.

სავარჯიშო 9.51: აჩვენეთ, რომ შედარების ოპერაციებზე აგებული ნებისმიერი ალგორითმი, რომელიც დაულაგებელი n ელემენტიანი სიის მინიმალურ ელემენტს იპოვნის, სულ ცოტა $n-1$ შედარებას მოითხოვს.

სავარჯიშო 9.52: აჩვენეთ, რომ შედარების ოპერაციებზე აგებული ნებისმიერი ალგორითმი, რომელიც დაულაგებელი n ელემენტიანი სიიდან მინიმალურ და მის შემდგომ (ანუ ორ უმცირეს) ელემენტს იპოვნის, სულ ცოტა $n-1+\log n$ შედარებას მოითხოვს.
მოიყვანეთ ასეთი (ოპტიმალური) ალგორითმის მაგალითი.

თავი 10

ბულის ალგებრა და მისი გამოყენება

რადგან ჩვენ ორობით სისტემაში მოქმედებას ვაპირებთ, აუცილებელია შესაბამისი ოპერაციების განსაზღვრაც. თუ ჩვენ ათობით არითმეტიკაში (შესაბამისად ალგებრაში) მიმატების, გამრავლების, გაყოფის ოპერაციები გვაქვს შემოღებული, ანალოგიური ოპერაციები უნდა შემოვიღოთ ორობით ალგებრაშიც, ანუ **ბულის ალგებრაში**, როგორც ამას მისი ფუძემდებლის, ინგლისელი მათემატიკოსის ჯორჯ ბულის (George Boole) პატივსაცემად უწოდებენ.

10.1 ბულის ალგებრის ელემენტები

ბულის ლოგიკა და, აქედან გამომდინარე, ბულის ალგებრა $\mathbb{B} = \{0, 1\}$ ორობით ანბანზეა განსაზღვრული. ზოგადად რომ ვთქვათ, ეს კლასიკური ლოგიკის მათემატიკურ ენაზე გადატანის ერთ-ერთი (ყველაზე გავრცელებული) მაგალითია. ლოგიკაში გვაქვს ჭეშმარიტი და მცდარი გამონათქვამები: ყოველი გამონათქვამი (მაგალითად, „$3+4=7$", „$12-3=1$", „ხვალ მზე ამოვა", „გუშინ წვიმდა" და ა.შ.) ***ან*** ჭეშმარიტია, ***ან*** მცდარი - სხვა რამ შეუძლებელია.

ბულის ძირითადი იდეა გამონათქვამების ***მათემატიკურ ცვლადებზე გადატანა*** იყო: ყოველი გამონათქვამი X ან ჭეშმარიტია (მაშინ $X = 1$), ან მცდარი ($X = 0$). ასევე შესაძლებელია გამონათქვამების კომბინირებაც, მაგალითად: „გუშინ მზე ამოვიდა ***და ამავდროულად*** წვიმდა", ან „$2+3=5$ და ამავდროულად $2-7=1$".

ამ მაგალითებში, თუ X =„გუშინ მზე ამოვიდა", Y =„წვიმდა", მაშინ სრული გამონათქვამი Z =„გუშინ მზე ამოვიდა ***და ამავდროულად*** წვიმდა" მათემატიკურად შემდეგნაირად ჩაიწერება: $Z = X\&Y$. საბოლოო ჯამში, თუ გუშინ მართლა ამოვიდა მზე ($X = 1$) და ამ დროს მართლაც წვიმდა ($Y = 1$), მაშინ $Z = X\&Y = 1$ ჭეშმარიტი იქნება.

მეორეს მხრივ, თუ X' =„$2+3=5$" (ჭეშმარიტია) და Y' =„$2-7=1$" (მცდარია), ცხადია, რომ $X' = 1$ და $Y' = 0$. აქედან გამომდინარე, $X'\&Y' = 0$ და გამონათქვამი Z =„$2+3=5$ და ამავდროულად $2-7=1$" მცდარია.

ანალოგიურად შეიძლება შემდეგი გამონათქვამების შედგენა: „ხვალ იწვიმებს ***ან*** ხვალ ქარი იქნება"; „$3+7=11$ ***ან*** $2-5=-3$". ცხადია, რომ ასეთი გამონათქვამები ჭეშმარიტია, თუ ერთი მაინც ჭეშმარიტია. მათემატიკურად ეს შემდეგნაირად შეიძლება ჩამოყალიბდეს: X =„ხვალ იწვიმებს", Y =„ხვალ ქარი იქნება", $Z = X \vee Y$=„ხვალ იწვიმებს ***ან*** ხვალ ქარი იქნება"; X' =„$3+7=11$", Y' =„$2-5=-3$", $Z' = X' \vee Y' = 1$: აქ ***ან*** ერთი უნდა შესრულებულიყო, ***ან*** მეორე.

მესამე მნიშვნელოვანი ოპერაცია ***უარყოფაა***: გამონათქვამის შებრუნებულის აღება.

მაგალითად, „ხვალ იწვიმებს"$\longrightarrow$„ხვალ *არ* იწვიმებს; „$3+7=13 \longrightarrow 3+7 \neq 13$" და ა.შ. X გამონათქვამის უარყოფა მათემატიკურად შემდეგნაირად ჩაიწერება: $\neg X$.

ბულის ალგებრის ეს სამი ოპერაცია ქართულ ენაზე შემდეგნაირად შეიძლება ჩამოვაყალიბოთ: გამონათქვამი $Z = X\&Y$ ჭეშმარიტია, თუ X ***და*** Y გამონათქვამი ორივე ჭეშმარიტია; $Z = X \vee Y$ ჭეშმარიტია, თუ X ***ან*** Y ჭეშმარიტია; $Z = \neg X$ ჭეშმარიტია, თუ X მცდარია.

ეს ყველაფერი მათემატიკურ ენაზე შემდეგნაირად ჩამოყალიბდება: ორი X, Y გამონათქვამის ***კონიუნქცია*** $X\&Y$ ორ ცვლადიან ფუნქციას $f : \mathbb{B}^2 \to \mathbb{B}$ ეწოდება, რომლის მნიშვნელობაა 1, თუ ორივე ცვლადის მნიშვნელობაა 1; ორი X', Y' გამონათქვამის ***დიზიუნქცია*** $X \vee Y$ ორ ცვლადიან ფუნქციას $g : \mathbb{B}^2 \to \mathbb{B}$ ეწოდება, რომლის მნიშვნელობაა 1, თუ ერთ-ერთი ცვლადის მნიშვნელობაა 1; Z გამონათქვამის ***უარყოფა*** $\neg Z$ ერთ ცვლადიან ფუნქციას $h : \mathbb{B} \to \mathbb{B}$ ეწოდება, რომლის მნიშვნელობაა 1, თუ Z ცვლადის მნიშვნელობაა 0.

ყოველივე ეს ცხრილის სახითაც შეიძლება გამოვსახოთ:

X	Y	$X\&Y$	$X \vee Y$	$\neg X$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

ამ ცხრილში მოცემულია აღსაწერი ფუნქციების მნიშვნელობები ცვლადების (ამ შემთხვევაში X და Y) ყველა შესაძლო კომბინაციისათვის.
შენიშვნა: კონიუნქცია და უარყოფა სხვადასხვანაირად აღიწერება ხოლმე. სიმარტივისთვის შეგვიძლია დავწეროთ: $X\&Y = X \cdot Y = XY,\ \neg X = \overline{X}$.

ბულის ალგებრაში უსასრულოდ ბევრი ფუნქციის მოყვანა შეიძლება, მაგრამ მთავარი ისაა, რომ ყველა ეს ფუნქცია ზემოთ მოყვანილი დიზიუნქციის, კონიუნქციისა და უარყოფის საშუალებით გამოისახება.
ფუნქციების მაგალითად შეგვიძლია მოვიყვანოთ:

$$\begin{aligned} f(x_1,x_2,x_3) &= \overline{x_1}x_2 \vee x_3,\\ g(x_1,x_2,x_3,x_4) &= (x_1 \vee x_2\overline{x_3} \vee \overline{x_4}, x_1x_2),\\ h(x_1,x_2) &= (x_1x_2, x_1 \vee x_2, x_1\overline{x_2} \vee \overline{x_1}x_2)\end{aligned}$$

ამ მაგალითში f ფუნქცია სამ ცვლადიანია (თითოეული ცვლადი $\mathbb{B}$ სიმრავლიდან) და ერთ ელემენტს გვაძლევს პასუხად (იგივე $\mathbb{B}$ სიმრავლიდან). ამ შემთხვევაში იტყვიან, რომ ეს ფუნქცია $\mathbb{B}^3$ სიმრავლეს ასახავს $\mathbb{B}$ სიმრავლეში: $f : \mathbb{B}^3 \to \mathbb{B}$.
g ფუნქცია 4 ცვლადს ასახავს ორ პარამეტრიან პასუხში, ესე იგი $g : \mathbb{B}^4 \to \mathbb{B}^2$, ხოლო $h\mathbb{B}^2 \to \mathbb{B}^3$.
ზოგადად, თუ რამიე ფუნქცია ϕ n ცვლადიანია, ხოლო ეს ცვლადები მნიშვნელობას რაიმე A სიმრავლიდან შეიძლება იღებდნენ და მისი პასიხი m პარამეტრიანია და ამ პასუხის ელემენტები C სიმრავლიდან შეიძლება იყოს, იტყვიან, რომ ეს ფუნქცია A^n სიმრავლეს (ანუ A სიმრავლის ელემენტებისაგან შემდგარ n სიგრძის სიტყვას - ვექტორს) ასახავს C^m სიმრავლეში (ანუ C სიმრავლის ელემენტებისაგან შემდგარ m სიგრძის სიტყვაში - ვექტორში).
მათემატიკურად ეს შემდეგნაირად ჩაიწერება: $\phi : A^n \to C^m$.
ამ თავში ჩვენ $f : \mathbb{B}^n \to \mathbb{B}^m$ ფუნქციებს განვიხილავთ. ასეთ ფუნქციებს დისკრეტულსაც, ანუ ყველგან წყვეტილს უწოდებენ. ანალოგიურად, ასეთ ფუნქციებზე შედგენილ მათემატიკას დისკრეტული მათემატიკა ეწოდება.

განვიხილოთ დისკრეტული ფუნქცია $f(x_1,x_2,x_3,x_4) = x_1\overline{x_2}x_3 \vee x_4 \vee \overline{x_2x_3} \vee x_1x_2x_3x_4$. ამ ფუნქციის გამოსათვლელად საჭიროა შემდეგი ბიჯები:

1. $z_1 = x_2x_3$;
2. $z_2 = x_1z_1 = x_1x_2x_3$;
3. $z_3 = z_2x_4 = x_1x_2x_3x_4$;
4. $z_4 = \overline{z_1} = \overline{x_2x_3}$;
5. $z_5 = \overline{x_2}$;
6. $z_6 = x_1z_5 = x_1\overline{x_2}$;
7. $z_7 = z_6x_3 = x_1\overline{x_2}x_3$;
8. $z_8 = z_7 \vee x_4 = x_1\overline{x_2}x_3 \vee x_4$;
9. $z_9 = z_8 \vee z_4 = x_1\overline{x_2}x_3 \vee x_4 \vee \overline{x_2x_3}$;
10. $z_{10} = z_9 \vee z_3 = x_1\overline{x_2}x_3 \vee x_4 \vee \overline{x_2x_3} \vee x_1x_2x_3x_4$.

ლოგიკურად ისმის ორი შეკითხვა: რამდენი ოპერაციის ჩატარება გვიხდება ამ გამოსახულების გამოსათვლელად? რამდენი ბიჯია (დროა) საჭირო ამ გამოსახულების გამოსათვლელად? ამ შეკითხვებზე პასუხის გაცემა შემდეგნაირად შეიძლება:
ოპერაციათა რაოდენობის დასათვლელად საკმარისია ლოგიკური ოპერაციების (კონიუნქცია, დიზიუნქცია, უარყოფა) დათვლა: $C(f(x_1,x_2,x_3,x_4)) = 10$.

რაც შეეხება დროს (ბიჯების რაოდენობას) $T(f(x_1,x_2,x_3,x_4))$, ზემოთ მოყვანილ გამოთვლის მეთოდში ეს ოპერაციათა რაოდენობას დაემთხვა, რადგან ჩვენ ყველა ოპერაციას რიგ-რიგობით ვატარებდით.
ამ მაგალითში გასათვალისწინებელია ის ფაქტი, რომ რამოდენიმე ოპერაცია ***ერთდროულად*** შეიძლება ჩატარდეს: მაგალითად, შესაძლებელია (x_1x_3), $\overline{x_2}$ და $(x_2x_3$ გამოსახულებების გამოთვლა, რადგან ისინი ერთმანეთზე დამოკიდებული არაა, განსხვავებით, მაგალითად, $y_1 = x_1x_2$ და $y_2 = x_1x_2x_3$ გამოსახულებელისაგან, რომელთა გამოთვლა ერთდროულად არ შეიძლება: ერთი მეორეზეა დამოკიდებული.

აქედან გამომდინარე, შეგვიძლია შემდეგი „პარალელური" ალგორითმის შემოთავაზება:

1. $z_1 = x_2x_3$ და ***ამავდროულად*** $z_2 = x_1x_4$ და ***ამავდროულად*** $z_5 = \overline{x_2}$ და ***ამავდროულად*** $z_6 = x_1x_3$;
2. $z_3 = z_1z_2 = x_1x_2x_3x_4$ და ***ამავდროულად*** $z_4 = \overline{z_1} = \overline{x_2x_3}$ და ***ამავდროულად*** $z_7 = z_5z_6 = x_1\overline{x_2}x_3$;
3. $z_8 = z_7 \vee x_4 = x_1\overline{x_2}x_3 \vee x_4$ და ***ამავდროულად*** $z_9 = z_4 \vee z_3 = \overline{x_2x_3} \vee x_1x_2x_3x_4$;
4. $z_{10} = z_8 \vee z_9 = x_1\overline{x_2}x_3 \vee x_4 \vee \overline{x_2x_3} \vee x_1x_2x_3x_4$.

აქ მნიშვნელოვანია ის ფაქტი, რომ გარკვეული ოპერაციები ***ერთდროულად*** სრულდება, რის ხარჯზეც ფუნქციის ***სიღრმე*** (ანუ გამოთვლის ბიჯების რაოდენობა) მცირდება.

აქედან გამომდინარე ვიღებთ შემდეგ განსაზღვრებას:

განმარტება 10.1: $f : \mathbb{B}^n \to \mathbb{B}^m$ ბულის ფუნქციის ოპერაციათა რაოდენობა $C(f(x_1,...,x_n)$ მასში შემავალი კონიუნქციის, დიზიუნქციისა და უარყოფების რაოდენობათა ჯამის ტოლია; იგივე ფუნქციის სიღრმე $C(f(x_1,...,x_n)$ მისი რეალიზაციისათვის პარალელურად ჩატარებულ ოპერაციათა ბიჯების რაოდენობის ტოლია.

აღსანიშნავია, რომ თუ ფუნქცია მრავალგანზომილებიანია, როგორც, მაგალითად, $f(x_1,x_2) = (\overline{x_1} \vee \overline{x_2}, x_2)$, მისი ელემენტების რაოდენობის გამოსათვლელად უნდა დავითვალოთ ყველა პასუხისთვის (ამ შემთხვევაში $C(\overline{x_1} \vee \overline{x_2}) = 3$, $C(x_2) = 0$ და დავითვალოთ მათი ჯამი: $C(f(x_1,x_2)) = 3+0 = 3$, ხოლო სიღრმის დასათვლელად უნდა გამოვიანგარიშოთ თითოეულის სიღრმე და ავიღოთ მათი მაქსიმუმი (ფუნქიის ყოველი მნიშვნელობის გამოთვლა შეიძლება ერთდროულად): $T(\overline{x_1} \vee \overline{x_2}) = 2$, $T(x_2) = 0$, $T(\overline{x_1} \vee \overline{x_2}, x_2) = \max\{T(\overline{x_1} \vee \overline{x_2}), T(x_2)\} = \max\{2,0\} = 2$.

სავარჯიშო 10.1: განიხილეთ ფუნქციები $f(x_1,x_2,x_3) = \overline{x_1}x_2 \vee x_3$, $g(x_1,x_2,x_3,x_4) = (x_1 \vee x_2\overline{x_3} \vee \overline{x_4}, x_1x_2)$ და $h(x_1,x_2) = (x_1x_2, x_1 \vee x_2, x_1\overline{x_2} \vee \overline{x_1}x_2)$. დაითვალეთ მათი ოპერაციათა რაოდენობა და სიღრმე.

იმ ამოცანებში, რომელთა განხილვას ჩვენ ვაპირებთ, მნიშვნელოვან როლს ორის მოდულით მიმატება ასრულებს. ეს განპირობებულია იმით, რომ კომპიუტერული სისტემები ორობით ანბანზეა აგებული.
ორობითი მიმატება (როგორც მას სხვანაირად უწოდებენ) განსაზღვრულია შემდეგნაირად:
ფუნქცია $f(x,y) = x \oplus y = 1$ მაშინ და მხოლოდ მაშინ, თუ მისი ცვლადებიდან ***ზუსტად ერთი*** ტოლია ერთის: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$.
როგორ შეიძლება ამ ფუნქციის კონიუნქიის, დიზიუნქციისა და უარყოფის მეშვეობით ჩაწერა? პირველ რიგში ქართულ ენაზე ჩამოვაყალიბოთ, თუ ცვლადების რა მნიშვნელობებისათვის ხდება ფუნქცია 1:
$f(x,y) = x \oplus y$ ფუნქცია ერთის ტოლი ხდება მაშინ და მხოლოდ მაშინ, თუ $x = 1$ და $y = 0$ ***ან*** $x = 0$ და $y = 1$. ლოგიკურად, თუ ცვლადი არის 1, მისი უარყოფა უნდა იყოს 0. ამიტომაც ვიღებთ შემდეგ გამონათქვამს:$f(x,y) = x \oplus y$ ფუნქცია ერთის ტოლი ხდება მაშინ და მხოლოდ მაშინ, თუ $x = 1$ და $\neg y = 1$ ***ან*** $\neg x = 1$ და $y = 1$. ეს შემდეგ ფუნქციას განაპირობებს: $f(x,y) = x \oplus y = \neg x \cdot y \vee \neg x \cdot y$. აქ „$x = 0$" (ანალოგიურად „$y = 0$") გამონათქვამების „$\neg x = 1$" („$\neg y = 1$") გამონათქვამებად შეცვლა იმიტომ დაგვჭირდა, რომ $x \cdot y$ გამოსახულება გამოსულიყო 1, თუ ***ზუსტად*** ერთი ცვლადია 1.

ზოგადად, თუ რაიმე უცნობი ფუნქცია მოცემულია ცხრილის სახით, მისი ფორმულებით ჩაწერა შემდეგნაირად შეიძლება:
მოცემულია ფუნქცია $f(x_1,x_2,...,x_n)$;

- გამოყავით ცვლადების ის კომბინაციები, რომლისთვისაც ფუნქცია ხდება 1 (ზედა შემთხვევაში ესაა $x = 0, y = 1$ და $x = 1, y = 0$);
- თითოეული ასეთი კომბინაციისათვის შეადგინეთ კონიუნქციებისაგან შემდგარი გამოსახულება. თუ $c_i = 0$, აიღეთ $\neg c_i$, წინააღმდეგ შემთხვევაში თვითონ c_i (ზედა შემთხვევაში ესაა $\neg x \cdot y$ და $x \cdot \neg y$);

- ეს გამოსახულებები შეაერთეთ დიზიუნქციებით (ზედა შემთხვევაში ვიღებთ $\neg x \cdot y \vee x \cdot \neg y$).

ასეთი სახით ჩაწერილ ფუნქციებს, სადაც კონიუნქციებით შეკრული ცვლადებით (ან მათი უარყოფებით) გამოსახულებები შეერთებულია დიზიუნქციებით, დიზიუნქციური ნორმალური ფორმა ეწოდება.

სავარჯიშო 10.2: ცრილით მოცემული ფუნქციები ჩაწერეთ დიზიუნქციური ნორმალური ფორმით:

x_0	x_1	x_2	f_1	f_2	f_3	f_4
0	0	0	1	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	1	1	0	1	1
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	1	0	0	0	0

აღსანიშნავია, რომ (ჩვეულებრივი ალგებრის მსგავსად) ჭეშმარიტია შემდეგი ტოლობები:

$$x(y \vee z) = x \cdot y \vee x \cdot z; \;\; x \vee 0 = x; \;\; x \vee 1 = 1; \;\; x \cdot 0 = 0; \;\; x \vee 0 = x.$$

სავარჯიშო 10.3: დაამტკიცეთ ზემოთ მოყვანილი ტოლობების ჭეშმარიტება (მოიყვანეთ თითოეული ფუნქციის ცხრილი და შეადარეთ მათი მნიშვნელობები).

სავარჯიშო 10.4: დაამტკიცეთ: $x \vee x \cdot y = x$; $\neg x \vee x \cdot y = \neg x \vee y$.

როგორც სიმრავლეთა თეორიაში, ასევე ბულის ლოგიკაშიც მნიშვნელოვანია ე.წ. დე მორგანის კანონები:

$$x \vee y = \overline{\overline{x} \cdot \overline{y}}; \;\; x \cdot y = \overline{\overline{x} \vee \overline{y}}.$$

სავარჯიშო 10.5: დაამტკიცეთ დე მორგანის კანონში მოყვანილი ფორმულები.

სავარჯიშო 10.6: რისი ტოლია $\neg(\neg x)$?

დე მორგანის კანონებზე დაყრდნობით დისიუნქციური ნორმალური ფორმის გადაყვანა შეიძლება ე.წ. კონიუნქციურ ნორმალურ ფორმაში - ისეთ გამოსახულებაში, რომელიც შედგება ცვლადების დიზიუნქციური გაერთიანებებით და ამ გამოსახულებათა კონიუნქციებით გაერთიანებებისაგან. კონიუნქციური ნორმალური ფორმით ჩაწერილი ფუნქციების მაგალითებია $(x_2 \vee \neg x_3)(x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \neg x_3)$ და $(x_1 \vee x_3)(x_1 \vee x_2)(x_1 \vee \neg x_2 \vee x_3)$, მაგრამ არა $(x_2 \vee \neg x_3)(x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \neg x_3) \vee x_1$.

თუ მოცემული გვაქვს რაიმე ფუნქცია, ზოგჯერ მისი ოპერაციებისა და ბიჯების რაოდენობის შემცირება შეიძლება ზემოთ მოყვანილი ტოლობების მეშვეობით: $(x_1 \vee x_2\overline{x_3} \vee \overline{x_4}x_1x_2) = x_1(1 \vee \overline{x_4}x_2) \vee x_2\overline{x_3} = x_1 \vee x_2\overline{x_3}$.
იგივე ფუნქციის კონიუნქციური ნორმალური ფორმით ჩაწერა შემდეგნაირად შეიძლება:
$x_1 \vee x_2\overline{x_3} = \neg(\overline{x_1} \cdot \overline{(x_2\overline{x_3})}) = \neg(\overline{x_1} \cdot (\overline{x_2} \vee x_3))$.

სავარჯიშო 10.7: შემდეგი ფუნქციები ჩაწერეთ დიზიუნქციური ნორმალური ფორმით:

$$\begin{aligned} f(x_1,x_2,x_3) &= (x_2 \vee \neg x_3)(x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \neg x_3); \\ g(x_1,x_2,x_3) &= (x_1 \vee x_3)(x_1 \vee x_2)(x_1 \vee \neg x_2 \vee x_3); \\ h(x_1,x_2,x_3) &= (x_1 \vee \neg x_2)(x_1 \vee x_2)(x_1 \vee x_2 \vee x_3). \end{aligned}$$

თავი 11

არითმეტიკული ოპერაციები n ბიტიან რიცხვებზე

11.1 n ბიტიანი რიცხვების მიმატება

მიმატების ოპერაცია იმდენად ხშირია ჩვენს ყოველდღიურ ცხოვრებაში, რომ ბევრ ადამიანს, ალბათ, არც კი მოსვლია თავში აზრად ის ფაქტი, რომ ეს პროცესი არც თუ ისე მარტივია. მაგალითად, ყველა სწრაფად გამოგვითვლის $5+3=8$, მაგრამ $3434164136861+3289747301047=6723911437908$ არც თუ ისე მცირე დროსა და ყურადღებას მოითხოვს. ზოგადად, რაც უფრო გრძელია შესაკრები რიცხვები, მით უფრო დიდ დროს ვანდომებთ გამოთვლას. მიუხედავად იმისა, რომ შეკრების ყველაზე მარტივი ალგორითმი - ქვეშ მიწერით მიმატება - საყოველთაოდ ცნობილია, ჩვენ მაინც შევეცდებით მის განხილვასა და გაანალიზებას და ამას როგორც ათობით, ასევე ორობითი რიცხვების მაგალითზე გავაკეთებთ.

ქვეშ მიწერით მიმატების მეთოდი

საყოველთაოდ ცნობილი მეთოდის გარჩევა მარტივი მაგალითით დავიწყოთ:

$$+\begin{array}{r}427\\613\\\hline 0\end{array}\quad 1 \qquad +\begin{array}{r}427\\613\\\hline 40\end{array}\quad 0 \qquad +\begin{array}{r}427\\613\\\hline 040\end{array}\quad 1 \qquad +\begin{array}{r}427\\613\\\hline 1040\end{array}$$

ზოგადად, თუ მოცემულია ორი n ციფრიანი რიცხვი $a_{n-1}...a_0$ და $b_{n-1}...b_0$, მისი ჯამი $d_n...d_0$ შემდეგი ალგორითმით შეიძლება გამოვიანგარიშოთ:

$c_0 = 0;$
for($i=0, i<n, i++$ *)*
{
$d_i = a_i + b_i + c_i$ *mod* 10;
if($a_i + b_i + c_i > 9$ *)*
$c_{i+1} = 1;$
else $c_{i+1} = 0;$
}
$d_n = c_n;$

იმის დასამტკიცებლად, რომ მოყვანილი ალგორითმი მართლაც სწორ შედეგს მოგვცემს, საჭიროა შემდეგი მათემატიკური ფორმულა: $(x_n x_{n-1}...x_0) = 10^n \cdot x_n + 10^{n-1} \cdot x_{n-1} + ... + 10^0 \cdot x_0$.

<u>სავარჯიშო 11.1:</u> დაამტკიცეთ ტოლობა $(x_n x_{n-1}...x_0) = 10^n \cdot x_n + 10^{n-1} \cdot x_{n-1} + ... + 10^0 \cdot x_0$.

<u>სავარჯიშო 11.2:</u> წინა სავარჯიშოს შედეგის გამოყენებით დაამტკიცეთ ზემოთ მოყვანილი ალგორითმის სისწორე.

იგივე მეთოდით ორობითი როცხვების შეკრებაც შეგვიძლია:

მოცემულია ორი n ბიტიანი ორობითი რიცხვი $a = (a_{n-1}...a_0)_2$ და $b = (b_{n-1}...b_0)_2$. გამოიანგარიშეთ მისი ჯამი $(d_n...d_0)_2$:

$$\begin{array}{ccccc} + & a_{n-1} & ... & a_1 & a_0 \\ & b_{n-1} & ... & b_1 & b_0 \\ \hline d_n & d_{n-1} & ... & d_1 & d_0 \end{array}$$

$c_0 = 0;$
for$(\ i = 0, i < n, i++)$
$\{$
$z_i = a_i + b_i + c_i \ mod\ 2;$
if$(\ a_i + b_i + c_i \geq 2\ )$
$c_{i+1} = 1;$
else $c_{i+1} = 0;$
$\}$
$d_n = c_n;$

სავარჯიშო 11.3: დაამტკიცეთ ტოლობა $(x_nx_{n-1}...x_0)_2 = 2^n \cdot x_n + 2^{n-1} \cdot x_{n-1} + ... + 2^0 \cdot x_0$.

სავარჯიშო 11.4: წინა სავარჯიშოს შედეგის გამოყენებით დაამტკიცეთ ზემოთ მოყვანილი ალგორითმის სისწორე.

აღსანიშნავია, რომ $c_{i+1} = 1$ მაშინ და მხოლოდ მაშინ, თუ a_i, b_i და c_i ცვლადებს შორის ორი ან სამი ერთის ტოლია. ამის განსაზღვრა შემდეგნაირად შეიძლება: თუ $a_i = b_i = 1$, მაშინ პირობა სრულდება. თუ ამ ორი ცვლადიდან ზუსტად ერთია ერთის ტოლი, მაშინ ამავდროულად მესამე ცვლადიც (ანუ c_i) უნდა იყოს 1. იმის დადგენა, არის თუ არა ორი ცვლადიდან ზუსტად ერთი ერთიანის ტოლი, შეიძლება ორის მოდულით მიმატებით: $a_i \oplus b_i$. აქედან გამომდინარე, იმის დადგენა, გვხვდება თუ არა ორი ან სამი ერთიანი სამ ცვლადში, შემდეგი ფორმულით შეიძლება: $c_{i+1} = a_ib_i \vee (a_i \oplus b_i)c_i$ (აღსანიშნავია, რომ $a_ib_i = 1$ მაშინ და მხოლოდ მაშინ, თუ ორივე ცვლადი არის 1).

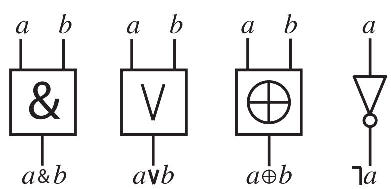
აქედან გამომდინარე z_i და c_i $(i = 1, ..., n-1)$ ცვლადების გამოსათვლელად გვაქვს შემდეგი ფორმულები:

$d_i = a_i \oplus b_i \oplus c_i,$
$c_{i+1} = a_ib_i \vee (x_i \oplus y_i)c_i,$
$d_n = c_n.$

მაგალითი:

	7	6	5	4	3	2	1	0
a	1	1	0	0	1	0	0	1
b	1	1	1	1	0	0	1	0
d	1	0	1	1	1	0	1	1
c	1	1	0	0	0	0	0	0

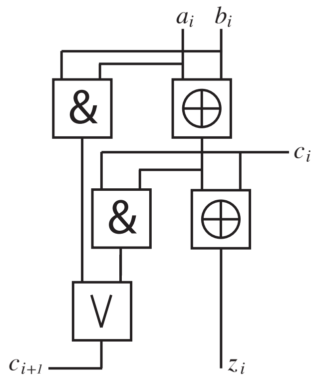
ზემოთ მოყვანილი ბულის ალგებრის ფორმულები გრაფიკულადაც შეიძლება გამოვსახოთ:



ნახ. 11.1: ლოგიკური ოპერაციების გრაფიკული გამოსახვა

კონიუნქიის, დიზიუნქციისა და უარყოფის ოპერაციებს ბულის ალგებრის ელემენტარულ ოპერაციებსაც უწოდებენ.

აქედან გამომდინარე, შეგვიძლია შევადგინოთ z_i და c_i ცვლადების გამოსათვლელი სქემა:

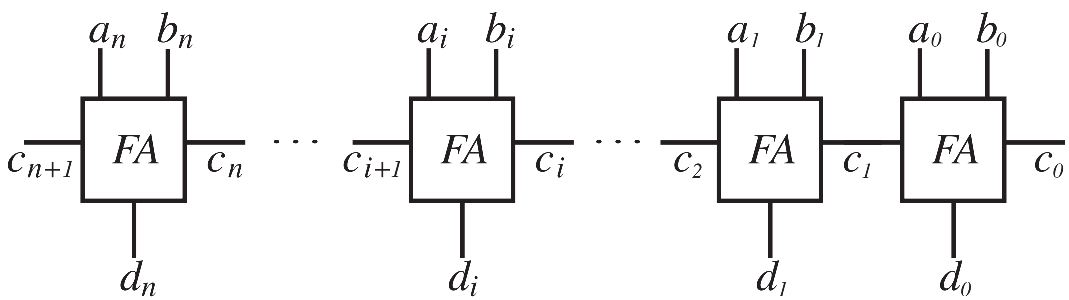


ნახ. 11.2: z_i და c_{i+1} ცვლადების გამოსათვლელი სქემა

აღსანიშნავია, რომ $\oplus$ ოპერაცია იმდენად ხშირად გამოიყენება, რომ მისი სქემა ერთი სიმბოლოთია აღნიშნული, თუმცა იგი რამოდენიმე ოპერაციის შესრულებას მოითხოვს.

სავარჯიშო 11.5: გამოიანგარიშეთ, რისი ფოლია $T(\oplus)$ და $C(\oplus)$.

თუ ჩვენ ამ სქემას ალვნიშნავთ როგორც FA (ინგლისური Full Adder, ანუ სრული შემკრები), მაშინ ორი n ბიტიანი რიცხვის შეკრებისათვის საჭირო სქემა (რომელსაც ვუწოდებთ CRA_n) შემდეგნაირი იქნება:



ნახ. 11.3: ორი n ბიტიანი რიცხვის შეკრებისათვის საჭირო სქემა CRA_n

სავარჯიშო 11.6: გამოიანგარიშეთ, რისი ტოლია $T(FA)$ (ანუ იმ ბიჯების რაოდენობა, რაც საჭიროა FA სქემის ყველა შედეგის გამოსაანგარიშებლად) და $C(FA)$ (ანუ FA სქემაში არსებული ელემენტების რაოდენობა), თუ $T(\&) = T(\vee) = T(\neg) = 1$, და $C(\&) = C(\vee) = C(\neg) = 1$. აქვე გამოიყენეთ წინა სავარჯიშოში გამოთვლილი $T(\oplus)$ და $C(\oplus)$.

შენიშვნა: ხშირად იღებენ $T(\neg) = 0$ და $C(\neg) = 0$, ანუ სქემებში უარყოფის ელემენტებს უგულებელყოფენ იმის გამო, რომ მათი რეალიზაცია სხვა ელემენტების რეალიზაციასთან შედარებით საკმაოდ მცირეა და, ამავე დროს, უარყოფებს ხშირად იყენებენ დამხმარე ელემენტებად (სხვადასხვა ტექნიკური მიზეზებით ორ ერთმანეთზე მიყოლებულ უარყოფას სვამენ ხოლმე). ამას გარდა, ტექნიკურად შესაძლებელია ელემენტების სქემის ისეთი რეალიზაცია, რომ $T(\neg(ab)) = T(ab)$, $T(\neg(a \vee b)) = T(a \vee b)$, $T(\neg ab) = T(ab)$, $T(\neg a \vee b) = T(\neg a \vee b)$.

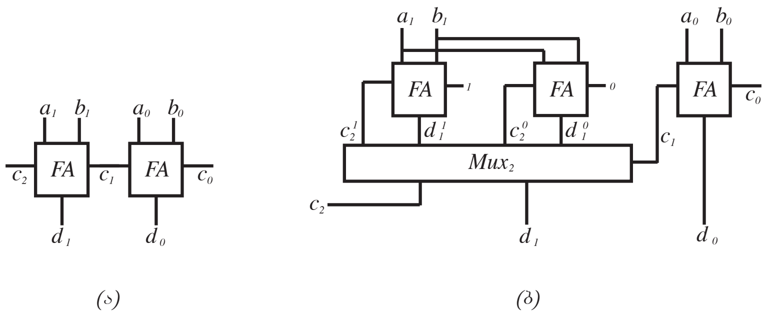
სავარჯიშო 11.7: გამოიანგარიშეთ, რისი ტოლია $T(\oplus)$, $C(\oplus)$ და, აქედან გამომდინარე, $T(FA)$ იმის გათვალისწინებით, რომ $T(\neg) = C(\neg) = 0$.

ადვილი დასანახია, რომ z_1 ცვლადი არ გამოითვლება, სანამ არ იქნება გამოთვლილი c_1 და, ზოგადად, z_i ცვლადის გამოთვლა არ შეიძლება, სანამ არ იქნება გამოთვლილი c_{i-1}. აქედან გამომდინარე, $T(CRA_n) = 4n$, $C(CRA_n) = 9n$ (აქ და შემდგომში დავუშვებთ, რომ $T(\neg) = C(\neg) = 0$).

სავარჯიშო 11.8: დაამტკიცეთ $T(CRA_n) = 4n$ და $C(CRA_n) = 9n$ ტოლობები.

სავარჯიშო 11.9: დახაზეთ CRA_1, CRA_2, CRA_3 და CRA_4 სქემები.

ბუნებრივია შემდეგი შეკითხვა: შესაძლებელია თუ არა ბიჯების რაოდენობისა და ელემენტების რიცხვის შემცირება?
თუ დავაკვირდებით ნახ. 11.4 *(ა)*-ში პირველ ორ FA ელემენტს, დავინახავთ შემდეგ მნიშვნელოვან ფაქტს: მარცხენა FA ელემენტი, რომელიც z_1 და c_1 ცვლადებს ითვლის, „ელოდება" c_0 ცვლადის გამოთვლას.

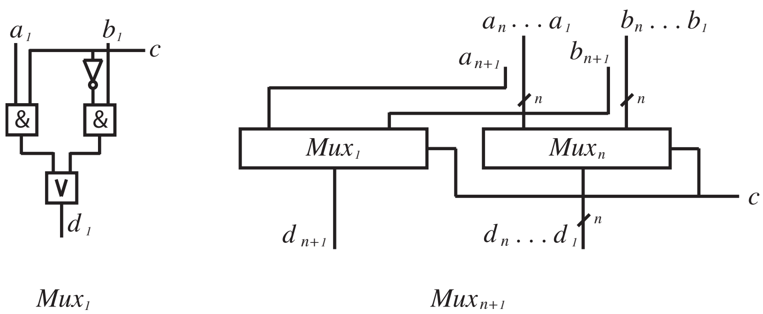


ნახ. 11.4: ორი n ბიტიანი რიცხვის მიმატების პარალელური სქემა

იმის გამო, რომ მას შეიძლება მიეწოდოს მხოლოდ $c_1 = 0$ ან $c_1 = 1$, ჩვენ შეგვიძლია ***ერთდროულად*** გამოვითვალოთ z_1 და c_2 იმ შემთხვევისათვის, როდესაც $c_1 = 0$ (z_1^0, c_2^0) და იმ შემთხვევისათვის, როდესაც $c_1 = 1$ (z_1^1, c_2^1).შემდეგ, როდესაც c_1 გამოთვლილი იქნება, შეიძლება ამ ორი საშუალედო შედეგიდან ერთ-ერთის არჩევა (ნახ. 11.4 *(ბ)*).
ამ ნახაზში გვხვდება ახალი ელემენტი Mux_2, რომლის მუშაობის შედეგები შემდეგი ცხრილით შეიძლება გამოისახოს:

$$z_1 = \begin{cases} z_1^0, & \text{თუ } c_1 = 0, \\ z_1^1, & \text{თუ } c_1 = 1 \end{cases} \qquad c_2 = \begin{cases} c_2^0, & \text{თუ } c_1 = 0, \\ c_2^1, & \text{თუ } c_1 = 1. \end{cases}$$

ზოგადად, Mux_n შემდეგნაირად შეიძლება აღიწეროს (ნახ. 11.5) :

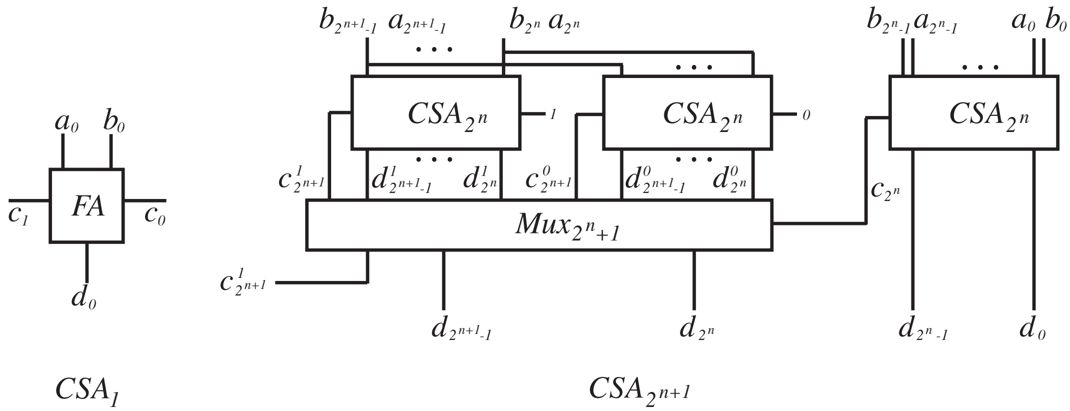


ნახ. 11.5: Mux_n – ორი n ბიტიანი რიცხვის ამორჩევის სქემა

$$z_i = \begin{cases} b_i, & \text{თუ } c = 0, \\ a_i, & \text{თუ } c = 1. \end{cases} \qquad i = \overline{1; n+1}.$$

ნახ. 11.4 -ში მოყვანილ სქემას უწოდებენ CSA_2 (Carry Select Adder – Carry bit დახსომებულ ბიტს ეწოდება, Select - შერჩევა, Adder -შემკრები). იგი ორ 2 ბიტიან რიცხვს $a = (a_1, a_0)$ და $b = (b_1, b_0)$ შეკრებს და 3 ბიტიან რიცხვს $z = (c_2, z_1, z_0)$ მოგვცემს პასუხად.

ზოგადად, $CSA_{2^{n+1}}$, რომელიც ორ 2^{n+1} ბიტიან რიცხვს $A = (a_{2^n-1}, ..., a_0)$ და $B = (b_{2^n-1}, ..., b_0)$ შეკრებს და $2^{n+1} + 1$ ბიტიან რიცხვს $z = (c_{2^n}, z_{2^n-1}, ..., z_0)$ მოგვცემს პასუხად, შემდეგნაირად აღიწერება (ნახ. 11.6):



ნახ. 11.6: $CSA_{2^{n+1}}$ – ორი 2^{n+1} ბიტიანი რიცხვის შეკრების პარალელური სქემა

CSA_1, ანუ ორი ერთბიტიანი რიცხვის შემკრები არის ზემოთ განხილული სქემა FA.
ძირითადი იდეა „დაყავი და იბატონე” პარადიგმაზეა აგებული: მონაცემები ორ ნაწილად იყოფა —

$$A_1 = (a_{2^{n+1}-1}...a_{2^n}) \quad A_0 = (a_{2^n-1}...a_0)$$
$$B_1 = (b_{2^{n+1}-1}...b_{2^n}) \quad B_0 = (b_{2^n-1}...b_0)$$

შემდეგ გამოითვლება $Z_0 = A_0 + B_0$ და ***ამავდროულად*** $Z_1^0 = A_1 + B_1 + 0$ და $Z_1^1 = A_1 + B_1 + 1$. ამის შემდეგ, c_{2^n} სიგნალის მეშვეობით, ამოირჩევა

$$Z_1 = \begin{cases} Z_1^0, & \text{თუ } c_{2^n} = 0, \\ Z_1^1, & \text{თუ } c_{2^n} = 1 \end{cases} \qquad c_{2^{n+1}} = \begin{cases} c^0_{2^{n+1}}, & \text{თუ } c_{2^n} = 0, \\ c^1_{2^{n+1}}, & \text{თუ } c_{2^n} = 1 \end{cases}$$

აქ $Z_0 = (z_{2^n-1}, ..., z_0)$ და $Z_1 = (z_{2^{n+1}-1}, ..., z_{2^n})$.

ადვილი დასამტკიცებელია ამ სქემის სისწორე მათემატიკურ ინდუქციაზე დაყრდნობით:

- ინდუქციის შემოწმება: თუ $n = 0$, ცხადია, რომ $CSA_1 = FA$ და იგი ორ ერთ ბიტიან რიცხვს სწორად შეკრებს;
- ინდუქციის დაშვება: დავუშვათ, CSA_{2^n} სწორად შეკრებს ორ 2^n ბიტიან რიცხვს;
- ინდუქციის ბიჯი: დავამტკიცოთ, რომ $CSA_{2^{n+1}}$ სწორად შეკრებს ორ 2^{n+1} ბიტიან რიცხვს.

<u>სავარჯიშო 11.10:</u> დაამტკიცეთ, რომ თუ CSA_{2^n} სწორად შეკრებს ორ 2^n ბიტიან რიცხვს, მაშინ $CSA_{2^{n+1}}$ სწორად შეკრებს ორ 2^{n+1} ბიტიან რიცხვს.

რაც შეეხება ამ ალგორითმის ბიჯების რაოდენობას $T(CSA_{2^{n+1}})$, მისი გამოთვლა შემდეგნაირად შეიძლება: უპირველესად ყოვლისა, უნდა გამოვითვალოთ ცვლადები Z_0, Z_1^0 და Z_1^1, რაც ***ერთდროულად*** შეიძლება მოხდეს $T(CSA_{2^n})$ ბიჯში. ამის შემდეგ უნდა ავირჩიოთ Z_1^0 და Z_1^1 ცვლადებიდან ერთ-ერთი Mux_{2^n+1} სქემის საშუალებით, რაც $T(Mux_{2^n+1}) = 2$ ბიჯშია შესაძლებელი.
აქედან გამომდინარე, $T(CSA_{2^{n+1}}) = T(CSA_{2^n}) + T(Mux_{2^n+1}) = T(CSA_{2^n}) + 2$. ამ რეკურსიული ფორმულის გახსნის შემდეგ მივიღებთ:

$$T(CSA_{2^{n+1}}) = O(\log n).$$

<u>სავარჯიშო 11.11:</u> დაამტკიცეთ ტოლობა $T(Mux_{2^n+1}) = 2$ (გამოიყენეთ ნახ. 11.5-ში მოყვანილი რეკურსიული სქემა).

$C(CSA_{2^{n+1}})$ ოპერაციათა რაოდენობის გამოსათვლელად გამოვიყენოთ ფორმულა:

$$C(CSA_{2^{n+1}}) = 3 \cdot C(CSA_{2^n}) + C(Mux_{2^n+1}).$$

სავარჯიშო 11.12: დაამტკიცეთ, რომ $C(CSA_{2^{n+1}})$ მართლაც ამ რეკურსიული ფორმულით გამოითვლება და გამოითვალეთ მისი მნიშვნელობა.

როგორც ვხედავთ, პარალელური ალგორითმებით შეიძლება შეკრების ამოცანის სწრაფად გადაჭრა: ორი n ბიტიანი რიცხვისათვის არა $O(n)$, არამედ $O(\log n)$ ბიჯია საჭირო, სამაგიეროდ იზრდება ელემენტების რაოდენობა. ეს გასაკვირი არ არის: მეტ ოპერაციას ვატარებთ, ოღონდ ერთდროულად და ამის ხარჯზე ვიგებთ დროს. აქვე უნდა აღინიშნოს, რომ არსებობს ორი n ბიტიანი რიცხვის მიმატების პარალელური ალგორითმი, რომლის დროის ზედა ზღვარია $O(\log n)$ და ელემენტების რაოდენობის ზედა ზღვარია $O(n)$ (სხვა სიტყვებით რომ ვთქვათ, შესაძლებელია ლოგარითმულ დროში გამოთვლა ისე, რომ ელემენტების რაოდენობა ძალიან არ გაიზარდოს), მაგრამ მათი განხილვა ჩვენი კურსის პროგრამას ცდება.

11.2 n ბიტიანი რიცხვების გამოკლება

წინა პარაგრაფში განხილული ალგორითმებით ფიქსირებული $n \in \mathbb{N}$ ბიტიანი სისტემების აგება შეიძლება. როდესაც აწყობილია სისტემა ფიქსირებული n ბიტიანი ორობითი რიცხვების დასამუშავებლად, მაქსიმალური რიცხვი, რაც შეიძლება წარმოვადგინოთ, იქნება $2^n - 1$: $(11...1)_2$. მასზე ერთით მეტი რიცხვი უკვე $n+1$ ბიტიანი იქნება: $(100...0)$, სადაც მარჯვენა n ბიტი ნულის ტოლია, ანუ თუ ჩვენ მხოლოდ n ბიტიან რიცხვებს განვიხილავთ და უფრო მაღალ ბიტებს უბრალოდ ვაგდებთ, ნებისმიერ არითმეტიკულ ოპერაციას 2^n მოდულით არითმეტიკაში ვატარებთ, რაც იმას ნიშნავს, რომ ნებისმიერი $0 \leq x < 2^n$ რიცხვისათვის შეგვიძლია გამოვიანგარიშოთ ისეთი შესაბამისი y, რომ $x + y = 0 \ mod \ 2^n$, ან, სხვა სიტყვებით რომ ვთქვათ, x რიცხვის შებრუნებული (უარყოფითი) მოდულით 2^n.
ბუნებრივია შეკითხვა: როგორ შეიძლება გამოვიანგარიშოთ მოცემული x რიცხვის შებრუნებული y? თუ განვიხილავთ რიცხვს $0 \ mod \ 2^n = 2^n = (11...1)_2 + 1_2 \ mod \ 2^n$, შეიძლება დავასკვნათ, რომ x რიცხვის შებრუნებულის გამოსათვლელად უნდა გამოვიანგარიშოთ ისეთი z რიცხვი, რომ $x + z = (11...1)_2$ და შემდეგ $y = z + 1$.

სავარჯიშო 11.13: დაამტკიცეთ, რომ თუ მოცემული n ბიტიანი x რიცხვისთვის მოვძებნით ისეთ z რიცხვს, რომ $x + z = (11...1)_2$, მაშინ x რიცხვის შებრუნებული (მოდულით 2^n) იქნება $y = z + 1$.

ცხადია, თუ ავიღებთ $x = (x_{n-1}x_{n-2}...x_0)_2$, მაშინ $z = \overline{x} = (\overline{x}_{n-1}\overline{x}_{n-2}...\overline{x}_0)_2$.

სავარჯიშო 11.14: დაამტკიცეთ, რომ მოცემული $x = (x_{n-1}x_{n-2}...x_0)_2$ და $z = (\overline{x}_{n-1}\overline{x}_{n-2}...\overline{x}_0)_2$ რიცხვებისათვის ჭეშმარიტია ტოლობა $x + z = 0 \ mod \ 2^n$.

აქედან გამომდინარე, $x = (x_{n-1}x_{n-2}...x_0)_2$ რიცხვის შებრუნებულია $y = \overline{x} + 1 \ mod \ 2^n = (\overline{x}_{n-1}\overline{x}_{n-2}...\overline{x}_0)_2 + 1 \ mod \ 2^n$.

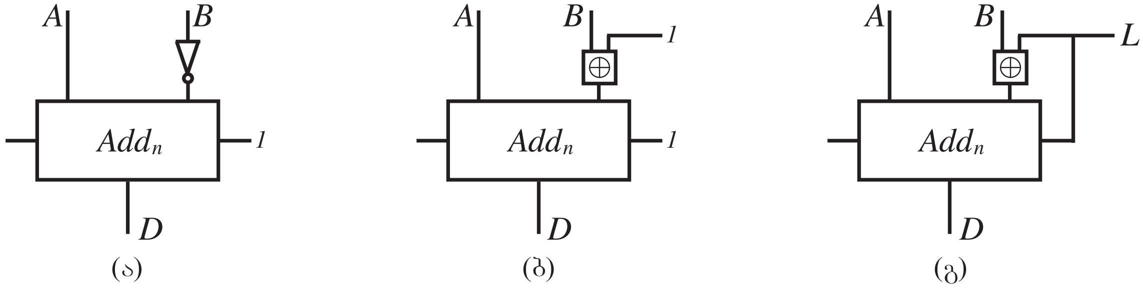
ყოველივე ზემოთ თქმულიდან შეიძლება დავასკვნათ, რომ მოცემული $a = (a_{n-1}a_{n-2}...a_0)_2$ და $b = (b_{n-1}b_{n-2}...b_0)_2$ რიცხვის სხვაობის გამოსათვლელად უნდა გამოვითვალოთ შემდეგი ჯამი: $d = a - b \ mod \ 2^n = a + \overline{b} + 1 \ mod \ 2^n$.
ეს კი ნახ. 11.7(ა)-ში ნაჩვენებ სქემას მოგვცემს.

აქ Add_n ორი n ბიტიანი რიცხვის მიმატების სქემაა, რომლის კონკრეტული რეალიზაცია ამ შემთხვევაში მნიშვნელოვანი არაა.

სავარჯიშო 11.15: დაამტკიცეთ, რომ ნახ. 11.7(ა)-ში ნაჩვენებ სქემა მართლაც A და B რიცხვების სხვაობას გამოითვლის.

რადგან $\neg x = x \oplus 1$, 11.7(ა) და 11.7(ბ) ნახაზებში ნაჩვენები სქემები ერთსა და იგივე შედეგს იძლევა. აქედან გამომდინარე, შესაძლებელია 11.7(გ) ნახაზში ნაჩვენები სქემით შეკრებისა და გამოკლების ერთიანი სქემის შექმნა: თუ მაკონტროლებელი სიგნალი $L = 1$, შესრულდება გამოკლება, ხოლო თუ $L = 0$ — მიმატება.

სავარჯიშო 11.16: დაამტკიცეთ, რომ $\neg x = x \oplus 1$ და $x = x \oplus 0$.



ნახ. 11.7: ორი n ბიტიანი რიცხვის გამოკლების სქემა (ა), (ბ) და მიმატება-გამოკლების კომბინირებული სქემა (გ)

<u>სავარჯიშო 11.17:</u> დაამტკიცეთ, რომ თუ 11.7(გ) ნახაზში ნაჩვენებ სქემაში მაკონტროლებელი სიგნალი $L = 1$, შესრულდება გამოკლება, ხოლო თუ $L = 0$ — მიმატება.

აღსანიშნავია, რომ რეალურ სისტემებში შედეგი $D = (d_n, d_{n-1}...d_0)_2$ (და საერთოდ რიცხვები) $n + 1$ ბიტიანი სიტყვებია, რომლებშიც უფროსი ბიტი d_n ნიშანს გვიჩვენებს: თუ $d_n = 1$, D რიცხვი უარყოფითია, წინააღმდეგ შემთხვევაში კი დადებითი. მაგრამ ჩვენ 2^n მოდულით არითმეტიკული ოპერაციებით შემოვიფარგლებით (ნიშნის გარეშე), რითიც უფრო ადვილია ძირითადი პრინციპების გაგება და შემდგომ სხვადასხვა პრაქტიკული რეალიზაციის ათვისება.

თავი 12

n ბიტიანი რიცხვების გამრავლება

12.1 ქვეშ მიწერით გამრავლება

„ქვეშ მიწერით გამრავლება" პირველი ალგორითმია, რომელსაც სკოლაში ვსწავლობთ:

$$
\begin{array}{rr} \times & 427 \\ & 613 \\ \hline & 1281 \end{array}
\qquad
\begin{array}{rr} \times & 427 \\ & 613 \\ \hline & 1281 \\ & 427 \end{array}
\qquad
\begin{array}{rr} \times & 427 \\ & 613 \\ \hline & 1281 \\ & 427 \\ & 2562 \end{array}
\qquad
\begin{array}{rr} \times & 427 \\ & 613 \\ \hline & 1281 \\ & 427 \\ & 2562 \\ \hline & 261751 \end{array}
$$

$A = (a_{n-1}...a_0)$ და $B = (b_{n-1}...b_0)$ რიცხვების გადასამრავლებლად გამოიანგარიშე $C_k = 10^k \cdot A \cdot b_k$ და მათი ჯამი $C = C_0 + C_1 + \cdots + C_{n-1}$. აღსანიშნავია, რომ $A \cdot b_k$ რიცხვის გამოსათვლელად ცალკე ალგორითმია საჭირო, ხოლო $10^k x$ მოცემული x რიცხვის k პოზიციით მარცხნივ „ჩაჩოჩებას" ნიშნავს (ან მარჯვნივ შესაბამისი რაოდენობის ნულების მიწერას).

<u>სავარჯიშო 12.1:</u> დაამტკიცეთ, რომ ზემოთ მოყვანილი ქვეშ მიწერით გამრავლების მეთოდი მართლაც სწორ პასუხს იძლევა.
<u>მინიშნება:</u> $B = (b_{n-1}...b_0)$ რიცხვი წარმოადგინეთ შემდეგი ჯამის სახით: $B = 10^{n-1} \cdot b_{n-1} + \cdots + 10^0 \cdot b_0$.

ანალოგიურად შეგვიძლია გამოვიანგარიშოთ ორობითში წარმოდგენილი რიცხვების ნამრავლიც:

$$
\begin{array}{rr} \times & 10110 \\ & 10011 \\ \hline & 10110 \\ & 10110 \\ & 00000 \\ & 00000 \\ & 10110 \\ \hline & 110100010 \end{array}
$$

ცხადია, რომ $A = (a_{n-1}...a_0)$ და $B = (b_{n-1}...b_0)$ ორობითი რიცხვის გამრავლების შემთხვევაში შემდეგნაირად უნდა მოვიქცეთ: გამოვიანგარიშოთ $C_k = 2^k \cdot A \cdot b_k = 2^k \cdot A \,\&\, b_k$ და მათი ჯამი $C = C_0 + C_1 + \cdots + C_{n-1}$.

<u>სავარჯიშო 12.2:</u> ათობითი რიცხვების ალგორითმის ანალოგიურად დაამტკიცეთ ამ მეთოდის სისწორე.

<u>სავარჯიშო 12.3:</u> დაამტკიცეთ, რომ ქვეშ მიწერით გამრავლების მეთოდის ოპერაციათა რაოდენობის ზედა ზღვარია $O(n^2)$. რა არის მისი ბიჯების რაოდენობის ზედა ზღვარი? შეიძლება თუ არა ამ მეთოდის პარალელიზაცია?

<u>სავარჯიშო 12.4:</u> განიხილეთ ათობითში ჩაწერილი n ბიტიანი რიცხვების ქვეშ მიწერით გამრავლების ალგორითმი $MultDec_n$ და ანალოგიური ალგორითმი $MultBin_n$, რომელიც ორობითში ჩაწერილ n ბიტიან რიცხვებს ამრავლებს. რა განსხვავებაა $C(MultDec_n)$ და $C(MultBin_n)$ ზედა ზღვრებს შორის? $T(MultDec_n)$ და $T(MultBin_n)$ ზედა ზღვრებს შორის? პასუხი დაამტკიცეთ.

12.2 გამრავლების პარალელური მეთოდი: ვოლესის ხე (Wallace Tree)

1964 წელს ავსტრალიელმა მეცნიერმა კრის ვოლესმა (Chris Wallace) გამრავლების პარალელიზაციის იდეა წამოაყენა, რომელსაც ჩვენს მაგალითზე განვიხილავთ.

C_i ცვლადები გამოვიანგარიშოთ როგორც ქვეშ მიწერით მეთოდში:

$$\begin{array}{rr} & 10110\\ \times & 10011\\ \hline C_0 & 10110\\ C_1 & 10110\\ C_2 & 00000\\ C_3 & 00000\\ C_4 & 10110\\ \hline C & 110100010\\ \text{დახსომებული} & \color{red}{001111100}\end{array}$$

$$\begin{array}{rr} & a_4\ a_3\ a_2\ a_1\ a_0\\ \times & b_4\ b_3\ b_2\ b_1\ b_0\\ \hline C_0 & c_{0,4}c_{0,3}c_{0,2}c_{0,1}c_{0,0}\\ C_1 & c_{1,4}c_{1,3}c_{1,2}c_{1,1}c_{1,0}\phantom{c_{0,0}}\\ C_2 & c_{2,4}c_{2,3}c_{2,2}c_{2,1}c_{2,0}\phantom{c_{0,0}c_{0,0}}\\ C_3 & c_{3,4}c_{3,3}c_{3,2}c_{3,1}c_{3,0}\phantom{c_{0,0}c_{0,0}c_{0,0}}\\ C_4 & c_{4,4}c_{4,3}c_{4,2}c_{4,1}c_{4,0}\phantom{c_{0,0}c_{0,0}c_{0,0}c_{0,0}}\\ \hline C & c_9\ c_8\ c_7\ c_6\ c_5\ c_4\ c_3\ c_2\ c_1\ c_0\end{array}$$

ცვლადებისათვის შემოვიტანოთ აღნიშვნა $C_i = (c_{i,4}c_{i,3}c_{i,2}c_{i,1}c_{i,0})_2$ და ყოველ ასეთ $c_{i,j}$ ცვლადს ვუწოდოთ 2^{i+j} რიგის. აქედან გამომდინარე, გვექნება 1 ცალი $2^0 = 1$ რიგის, 2 ცალი 2^1 რიგის, სამი 2^2 რიგის, ოთხი 2^3 რიგის, ხუთი 2^4 რიგის, ისევე ოთხი 2^5 რიგის, სამი 2^6 რიგის, ორი 2^7 რიგის და ერთი 2^8 რიგის ცვლადი.
$c_{0,0}$ ცვლადი პირდაპირ უნდა გადავიდეს, როგორც საბოლოო პასუხის ყველაზე დაბალი ბიტი: $c_0 = c_{0,0}$ (2^0 რიგის ცვლადი მხოლოდ ერთია, ასე რომ, მას არაფერი ემატება).
2^1 რიგის ცვლადები უნდა შევკრიბოთ, შედეგად ვიღებთ ერთ 2^1 რიგის პასუხს (მათ ორობით ჯამს) და ერთ 2^2 რიგის პასუხს (დახსომებულ ბიტს, რომელიც შემდეგში უფრო მაღალი ბიტების ჯამს დაემატება).
ანალოგიურად ვაჯამებთ 2^2 რიგის სამ ცვლადს, პასუხად ვიღებთ ერთ 2^2 რიგის და ერთ 2^3 რიგის ბიტს.
ამ წესით ვაჯამებთ 2^i რიგის ცვლადებს (ორს ან სამს ერთად) და შედეგად ვიღებთ 2^i და 2^{i+1} რიგის ბიტებს, რომლებიც შემდეგ ბიჯში უნდა დავაჯგუფოთ და იგივე წესით ავჯამოთ.
ამ პროცესს ვიმეორებთ მან ამ, სანამ არ მივიღებთ ყოველი რიგში ორ ან ერთ ბიტს. ბოლოს ჩვეულებრივი შემკრებით ვაჯამებთ იმ ნაწილს, რომელიც ყოველი რიგის ორ-ორი ბიტისაგან შედგება.
ყოველივე ეს სქემატურად ნაჩვენებია ნახაზში 12.1.
აღსანიშნავია, რომ HA ორი ბიტის შემკრები სქემაა, რომელიც a და b ერთ ბიტიანი რიცხვების ორობით ჯამს და დახსომებულ ბიტს გამოითვლის. ფაქტიურად ეს იგივე FA სქემაა, სადაც $C = 0$.

სავარჯიშო 12.5: FA სქემის გამოყენებით დახაზეთ HA სქემა.

ზემოთ მოყვანილ მეთოდს ვოლესის ხე ეწოდება, რადგან მის სქემას ხის სტრუქტურა აქვს: ყოველ შრეში შესაკრებთა რაოდენობა იკლებს. უხეშად რომ დავითვალოთ, სამი შესაკრები ორზე დადის.
მისი ძირითადი იდეაც ესაა: HA სქემის გამოყენებით სამი შესაკრები a, b, c ორ ისეთ შესაკრებზე x, y დავიყვანოთ, რომ $a + b + c = x + 2y$.

სავარჯიშო 12.6: მაქსიმუმ რამდენ ბიტიანი რიცხვი შეიძლება მივიღოთ ორი n ბიტიანი რიცხვის გამრავლების შედეგად?

12.3 კარაცუბა-ოფმანის გამრავლების მეთოდი

1960ან წლებში მოსკოვში მომუშავე მათემატიკოსებმა ანატოლი კარაცუბამ და იური ოფმანმა გამრავლების მეთოდი შეიმუშავეს, რომლის ოპერაციათა რაოდენობის ზედა ზღვარი $O(n^2)$-ზე უკეთესია, რითაც მნიშვნელოვანი ნაბიჯი გადადგეს ეფექტური ალგორითმების შემუშავების თვალსაზრისით.

ძირითადი იდეა მარტივია: თუ მოცემულია ორი n ბიტიანი რიცხვი $A = (a_{n-1}...a_0)_2$, $B = (b_{n-1}...b_0)_2$ და ვეძებთ $C = (c_{2n-1}...c_0)_2 = A \cdot B$, ჯერ მონაცემები ორ ტოლ ნაწილად დავყოთ: $A = (A_1A_0)_2$, $B = (B_1B_0)_2$, სადაც $A_1 = (a_{n-1}...a_{\frac{n}{2}})_2$, $A_0 = (a_{\frac{n}{2}-1}...a_0)_2$, $B_1 = (b_{n-1}...b_{\frac{n}{2}})_2$, $B_0 = (b_{\frac{n}{2}-1}...b_0)_2$.
მივიღებთ $A = 2^{\frac{n}{2}}A_1 + A_0$, $B = 2^{\frac{n}{2}}B_1 + B_0$ და

$$\begin{aligned} A \cdot B &= (2^{\frac{n}{2}} A_1 + A_0)(2^{\frac{n}{2}} B_1 + B_0) = \\ &= 2^n \cdot A_1 B_1 + 2^{\frac{n}{2}}(A_0 B_1 + A_1 B_0) + A_0 B_0. \end{aligned}$$

როგორც ვხედავთ, ჩასატარებელია 4 გამრავლება, ოღონდ უკვე $\frac{n}{2}$ ბიტიანი რიცხვების. თუ გამრავლების ალგორითმს აღვნიშნავთ როგორც $Mult_n$ და მას რეკურსიულად გამოვიყენებთ, მივიღებთ ელემენტების რაოდენობის შემდეგ შეფასებას:

$$C(Mult_n) = 4 \cdot C(Mult_{\frac{n}{2}}) + 3 \cdot C(Add_{\frac{n}{2}}) \in O(4^{\log n}) = O(n^{\log 4}) = O(n^2)$$

და, როგორც ვხედავთ, ელემენტების რაოდენობას *ვერ* ვამცირებთ. ეს კი იმითაა გამოწვეული, რომ ზემოთ მოყვანილ გამოსახულებაში 4 გამრავლება გვხვდება. თუ რამენაირად მათ შემცირებას მოვახერხებთ, ელემენტების რაოდენობის ზედა ზღვარს კვადრატულზე უკეთესს გავხდით.

სავარჯიშო 12.7: დაამტკიცეთ, რომ $C(Mult_n) = 4 \cdot C(Mult_{\frac{n}{2}}) + 3 \cdot C(Add_{\frac{n}{2}}) \in O(4^{\log n})$.
კარაცუბამ და ოფმანმა შემდეგი ძირითადი იდეა წამოაყენეს: ზედა გამრავლების ფორმულაში $A_1 \cdot B_1$ და $A_0 \cdot B_0$ ნამრავლს გვერდს ვერ ავუვლით. ამიტომ ფრჩხილებში მოცემულ გამოსახულება უნდა შევცვალოთ ისეთით, რომელიც ერთ ახალ გამრავლებასა და $A_1 \cdot B_1$ და $A_0 \cdot B_0$ ელემენტებს შეიცავს.
მისი გამოთვლა შემდეგნაირად შეიძლება:
$A_0 B_1 + A_1 B_0 = X - A_1 \cdot B_1 - A_0 \cdot B_0$. აქედან გამომდინარე,

$$X = A_0 \cdot B_1 + A_1 \cdot B_0 + A_1 \cdot B_1 + A_0 \cdot B_0 = A_0(B_0 + B_1) + A_1(B_0 + B_1) = (A_1 + A_0) \cdot (B_1 + B_0).$$

საბოლოოდ ვიღებთ ნამრავლის ფორმულას:

$$A \cdot B = 2^n \cdot A_1 \cdot B_1 + 2^{\frac{n}{2}}((A_1 + A_0) \cdot (B_1 + B_0) - A_1 \cdot B_1 - A_0 \cdot B_0) + A_0 \cdot B_0.$$

ერთი შეხედვით გამოსახულება უფრო გართულდა, მაგრამ იმის გამო, რომ $A_1 \cdot B_1$ და $A_0 \cdot B_0$ ერთხელ უკვე გამოვითვალეთ, ფრჩხილებში მყოფ გამოსახულებაში მისი ახლად გამოთვლა საჭირო აღარაა, რადგან აქ მისი ადრე გამოთვლილი მნიშვნელობის გამოყენებაა შესაძლებელი.

საბოლოდ ვიღებთ ელემენტთა რაოდენობის შემდეგ შეფასებას:

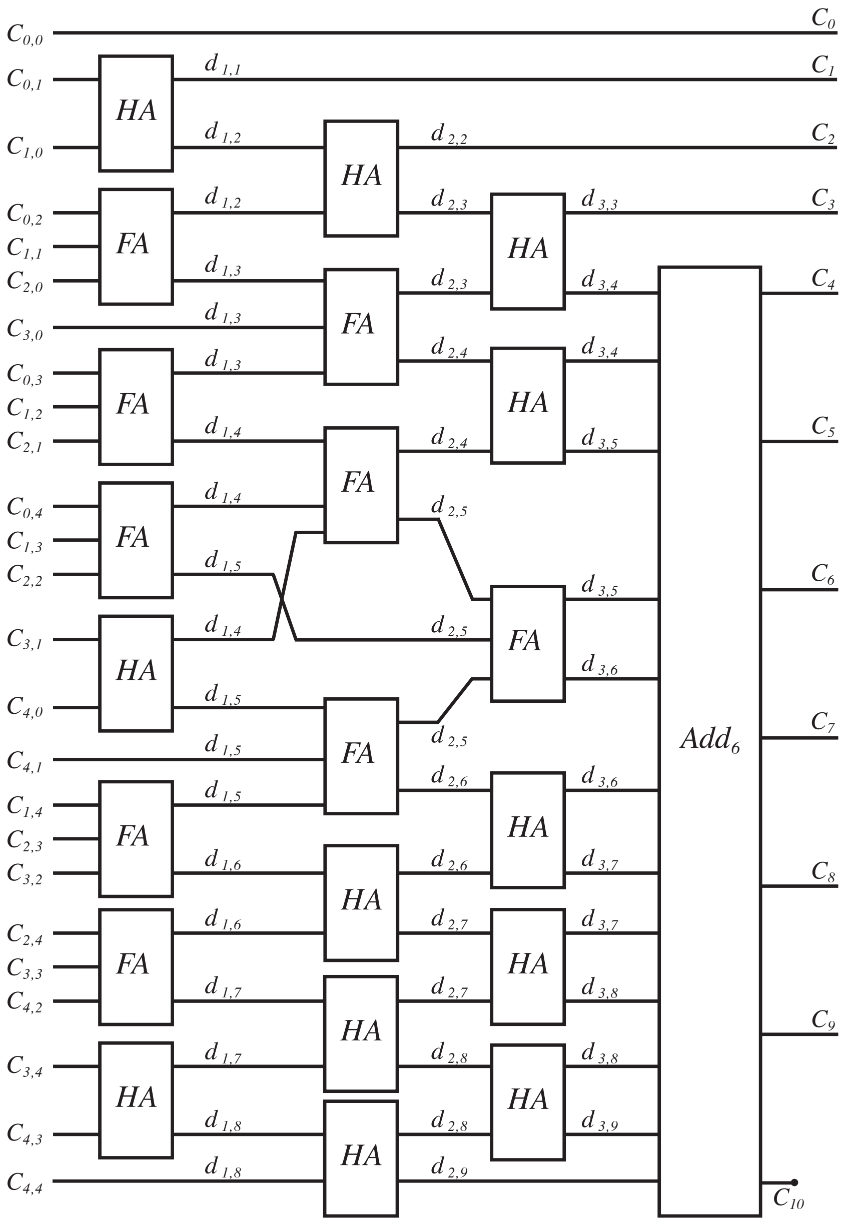
$$C(Mult_n) = 3 \cdot C(Mult_{\frac{n}{2}}) + 4 \cdot C(Add_{\frac{n}{2}}) + 2 \cdot C(Sub_{\frac{n}{2}})$$

რადგან $C(Add_k) = C(Sub_k) = const \cdot k$ (როგორც ვნახეთ, ეს ორი ოპერაცია ერთი და იგივე სქემით შეგვიძლია ჩავატაროთ), ვიღებთ:

$$C(Mult_n) = 3 \cdot C(Mult_{\frac{n}{2}}) + 6 \cdot C(Add_{\frac{n}{2}}) = 3 \cdot C(Mult_{\frac{n}{2}}) + const \cdot n \in O(3^{\log n}) = O(n^{\log 3}).$$

ამით დამტკიცდა, რომ შესაძლებელია გამრავლების ოპერაციის კვადრატულზე უკეთეს ელემენტების რაოდენობით ჩატარება, რაც თავის დროზე ძალიან მნიშვნელოვანი შედეგი იყო.

სავარჯიშო 12.8: დაამტკიცეთ, რომ $3 \cdot C(Mult_{\frac{n}{2}}) + const \cdot n \in O(n^{\log 3})$.



ნახ. 12.1: ორი 5 ბიტიანი რიცხვის გამრავლების ვოლესის ხის შემკრები ნაწილი

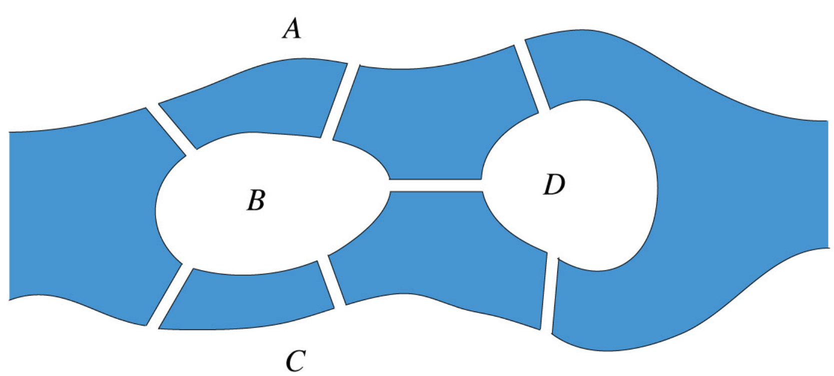
თავი 13

გრაფთა თეორიის ელემენტები

13.1 გრაფების განსაზღვრება და ძირითადი თვისებები

ყველა დროის ერთ-ერთმა უდიდესმა მათემატიკოსმა ლეონარდ ეილერმა, რომელიც ერთი ხანი ქალაქ კიონიგსბერგში (ახლანდელ კალინინგრადში) ცხოვრობდა, შემდეგი ამოცანა დასვა:

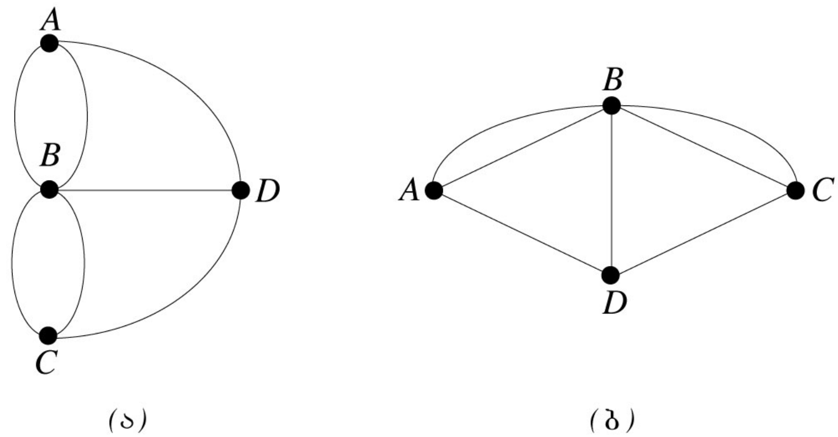
ეილერის ამოცანა ხიდების შესახებ: ქალაქში მოედინება მდინარე, რომელიც მას ორ ნაწილად ჰყოფს. ამას გარდა, თვითონ მდინარეში ორი კუნძულია (ნახ. 13.1). ხმელეთის ნაწილები, რომლებიც ნახაზზე ლათინური ასოებითაა აღნიშნული, ერთმანეთთან შეერთებულია ხიდებით. შეიძლება თუ არა ქალაქს შემოვუაროთ ისე, რომ ყველა ხიდზე გადავიდეთ ***ერთხელ და მხოლოდ ერთხელ***?



ნახ. 13.1: კიონიგსბერგის მდინარე და ხიდები

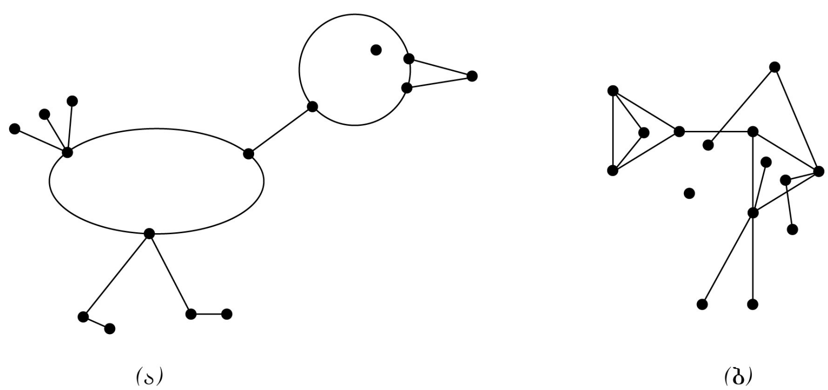
აღსანიშნავია, რომ მოცემული სურათის საჩვენებლად ხატვა სულაც არაა საჭირო: საკმარისია ხმელეთის ნაწილები აღვნიშნოთ წერტილებით, ხოლო მათი შემაერთებელი ხიდები კი ხაზებით (ნახ. 13.2).

ასეთ სტრუქტურას - წერტილებს და მათ შემაერთებელ ხაზებს - *გრაფი* ეწოდება. ზემოთ მოყვანილი ორივე ნახაზი (ა) და (ბ) ერთსა და იმავე გრაფს აღწერს, იმის და მიუხედავად, რომ ერთი შეხედვით ნახაზები სხვადასხვაა: გრაფში მთავარია იმის შესახებ ინფორმაციის მიღება, თუ რომელი წერტილი რომელ სხვა წერტილებთანაა შეერთებული.
გრაფის წერტილებს მისი *კვანძები* ეწოდება, ხოლო ხაზებს კი - წიბოები.
ფორმალურად გრაფი შემდეგნაირადაც შეგვიძლია აღვწეროთ:
მოცემულია წვეროთა სიმრავლე $V = \{A, B, C, D\}$ და წიბოები $E = \{(A, B), (B, A), (A, D), (B, C), (C, B), (B, D), (C, D)\}$.
აქედან გამომდინარე, გრაფის ნახაზის დახაზვა სულაც არაა აუცილებელი: იგი წვეროთა და წიბოთა სიმრავლეებითაც შეიძლება სრულფასოვნად აღიწეროს. მთავარია იმის ცოდნა, თუ რა წვეროებია მოცემული და რომელი წვეროებია წიბოებით შეერთებული.
ასე რომ, გრაფი G შეიძლება განვსაზღვროთ, როგორც ორი სიმრავლისაგან შემდგარი სტრუქტურა: $G = (V, E)$.



ნახ. 13.2: კიონიგსბერგის ხიდების ამოცანის გრაფები

მაგალითისათვის მოვიყვანოთ გრაფი, რომელიც ნაჩვენებია ქვედა ნახაზში.



ნახ. 13.3: ერთი და იგივე გრაფის სხვადასხვა გრაფიკული წარმოდგენა

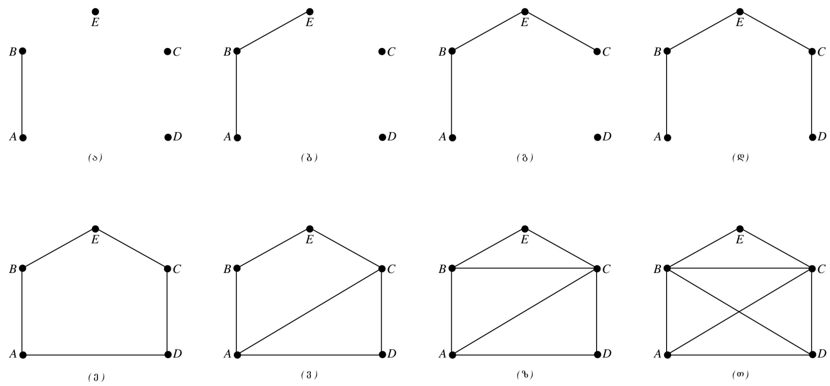
ადვილი დასანახი არაა, რომ (ა) და (ბ) ნახაზებში ერთი და იგივე გრაფია ნაჩვენები. მაგრამ თუ წვეროებს სახელებს დავარქმევთ და გადავამოწმებთ, თუ რომელი წვეროა რომელთან შეერთებული, მათ ექვივალენტურობას (ტოლობას) წვეროებისა და წიბოების სიმრავლეების ტოლობით დავამტკიცებთ.

სავარჯიშო 13.1: აღწერეთ 13.3 (ა) და 13.3 (ბ) ნახაზებში მოყვანილი გრაფების წვეროებისა და წიბოების სიმრავლეები და დაამტკიცეთ მათი ტოლობა.

ზემოთ მოყვანილ გრაფებში არსებობს ე.წ. *იზოლირებული* წვერო - ანუ ისეთი, რომელიც სხვა წვეროებთან არაა დაკავშირებული. შეიძლება არსებობდეს ისეთი გრაფებიც, რომლებიც ორი ნაწილისგან (გრაფისგან) შედგება. ასეთ გრაფებს *არა* ბმული ეწოდებათ. ჩვენ ძირითადად ბმულ გრაფებს განვიხილავთ, ანუ ისეთებს, სადაც *ნებისმიერ* ორ წვეროს შორის შემაერთებელი გზა არსებობს.

როგორც ვნახეთ, ერთი და იგივე გრაფი შეიძლება სხვადასხვანაირად დავხაზოთ. ზოგადად, გრაფის დახაზვას მნიშვნელობა არ აქვს. მთავარია გვქონდეს ინფორმაცია იმის შესახებ, თუ რომელი წვერო რომელთანაა შეერთებული.

ნახ. 13.4-ში ნაჩვენებია, თუ როგორ შეიძლება დაიხაზოს გრაფი ისე, რომ ერთ წიბოზე ორჯერ არ გადავიაროთ.



ნახ. 13.4: გრაფის დახატვის ეტაპები

სიტყვიერად ეს შემდეგნაირად შეიძლება გამოვთქვათ: „A წვეროდან ხაზი გაავლე B წვეროში, იქიდან E-ში, შემდეგ C-ში, D-ში, ისევ A-ში, შემდეგ C-ში, B-ში და ბოლოს ისევ D-ში".
უფრო ფორმალურად ეს შემდეგი სიტყვით შეიძლება გამოვსახოთ:

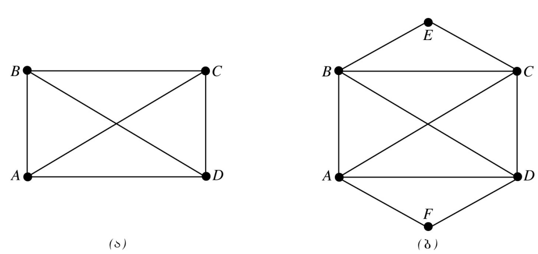
$$ABECDACBD.$$

გრაფის დახაზვის პროცესი შეიძლება შევადაროთ გრაფზე „სიარულს": ერთი წვეროდან მეორეში წიბოს გავლება ამ წვეროდან მეორეში „გადასვლის" ტოლფასია.

თუ ასეთი წესით ერთი წვეროდან მეორეში გადავალთ (ისე, რომ ერთსა და იმავე წიბოზე ორჯერ არ გადავივლით), განვლილი წიბოების ერთობლიობას „გზას" ვუწოდებთ. ასე, მაგალითად, 13.4 (დ) -ში განვლილი გზა იქნება $ABECD$. იმ წიბოებს, რომლებზედაც გზის გავლისას გადავივლით, ამ გზით *დაფარული* წიბოები ეწოდება.

ამრიგად, ეილერის ამოცანაც შეიძლება ასე დავსვათ: მოცემულ გრაფში არსებობს თუ არა ისეთი გზა, რომელიც ყველა წიბოს (მხოლოდ ერთხელ) დაფარავს? თუ ასეთი გზა არსებობს, მას ეილერის ციკლს უწოდებენ.
ზემოთ მოყვანილი ამოცანა შეიძლება ჩამოვაყალიბოთ ასეთნაირადაც: არსებობს თუ არა მოცემულ გრაფში ეილერის ციკლი?

თუ განვიხილავთ ორ გრაფს, რომელიც ნაჩვენებია 13.5 ნახაზში, ადვილად დავრწმუნდებით, რომ $ABECDACBDFA$ გზა სწორედ ნახ. 13.5(ბ) გრაფს ზემოთ აღწერილი პირობებით გადაფარავს, მაგრამ ნახ. 13.5(ა) გრაფისათვის ასეთი გზის პოვნა ძნელია.
ბევრი ცდის შემდეგ გაჩნდება ეჭვი, რომ ასეთი გზა საერთოდ არ არსებობს (ანუ ამ გრაფის ერთი ხელის მოსმით დახაზვა არ შეიძლება, თუ ერთ წიბოზე მაინც ორჯერ არ გადავივლით.



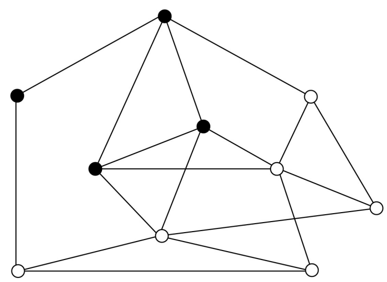
ნახ. 13.5: ორი გრაფის მაგალითი

იმის დასადგენად, თუ რომელი გრაფის შემოვლა შეიძლება ისე, რომ ყველა წიბო დაიფაროს *მხოლოდ* ერთხელ, შემოვიტანოთ შემდეგი განმარტება:

თუ მოცემულია გრაფი $G = \{V, E\}$, მისი ნებისმიერი $v \in V$ წვეროს *რიგი* $deg(v)$ ეწოდება მასთან მიერთებულ წიბოთა რაოდენობას. ცხადია, რომ ნებისმიერი გრაფის წვეროს რიგი შეიძლება იყოს ან კენტი, ან ლუწი. თუ წვეროს რიგია ლუწი, მას „ლუწიან”, წინააღმდეგ შემთხვევაში კი „კენტიან” წვეროს ვუწოდებთ.

13.5 (ბ) ნახაზში მოყვანილი გრაფისათვის $deg(A) = 4$, ხოლო $deg(F) = 2$.

ქვემოთ მოყვანილია გრაფი, რომლის კენტიანი წვეროები თეთრადაა ნაჩვენები, ხოლო ლუწიანი კი - შავად.

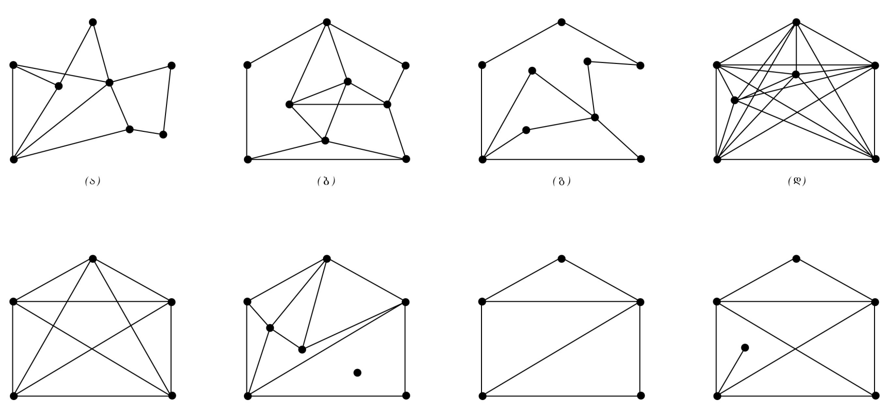


ნახ. 13.6: კენტიანი და ლუწიანი წვეროები გრაფში

ამ განმარტებაზე დაყრდნობით შეიძლება შემოვიტანოთ ეილერის ციკლის არსებობის კრიტერიუმი, რომელიც ეილერის თეორემის სახელითაა ცნობილი:

თეორემა 13.1: (ეილერის თეორემა გრაფში ციკლების არსებობის შესახებ)
ნებისმიერ გრაფში ეილერის ციკლი იარსებებს მაშინ და მხოლოდ მაშინ, თუ მასში ***კენტიანი წვეროების*** რაოდენობა *არ* აჭარბებს ორს.
თუ გრაფში კენტიანი წვერო ***არ არსებობს***, მაშინ მასში მოიძებნება ეილერის ***ჩაკეტილი*** ციკლი: რომელი წვეროდანაც დავიწყებთ შემოვლას, იმაშივე დავამთავრებთ.
შენიშვნა: თუ გრაფში კენტიანი წვეროების რაოდენობაა ***ორი***, მაშინ მასში იარსებებს ეილერის ***ღია*** ციკლი: ერთი კენტიანი წვეროდან შემოვლას ვიწყებთ და მეორეში ვამთავრებთ.

სავარჯიშო 13.2: ნახ. 13.7-ში ნაჩვენები გრაფებიდან რომელს შეიძლება ქონდეს ეილერის ციკლი?
მითითება: გამოიყენეთ ეილერის თეორემა.



ნახ. 13.7: გრაფების მაგალითები

სავარჯიშო 13.3: ნახ. 13.7-ში ნაჩვენებ გრაფებში დაითვალეთ ყოველი წვეროს რიგის ჯამი (მათემატიკურ ენაზე ეს შემდეგნაირად ჩაიწერება: მოცემული $G = (V, E)$ გრაფისათვის

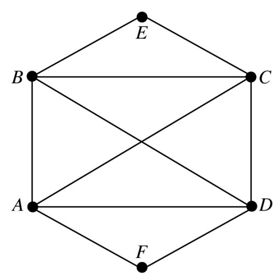
$$\sigma(G) = \sum_{v \in V} deg(v).$$

ეს რიცხვი შეადარეთ იმავე გრაფის წიბოების რაოდენობას. რა კანონზომიერება შეიძლება დავადგინოთ წვეროთა რიგების ჯამსა და წიბოების რაოდენობას შორის? მათემატიკურ ენაზე რომ ვთქვათ, რა დამოკიდებულებაა $\sigma(G)$ და $|E|$ რიცხვებს შორის? (აქ $|E|$ არის E სიმრავლის ელემენტების რაოდენობა, რაც ჩვენს შემთხვევაში გრაფის წიბოთა რიცხვის ტოლია.)

სავარჯიშო 13.4: დაამტკიცეთ, რომ ნებისმიერი $G = (V, E)$ გრაფისათვის $\sigma(G) = 2|E|$.

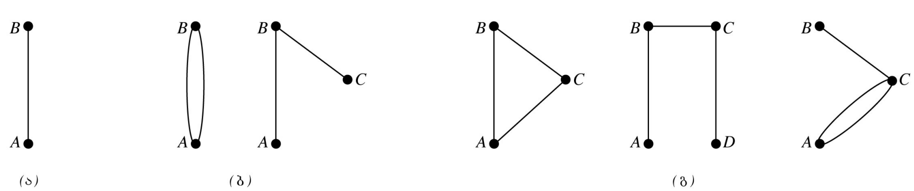
სავარჯიშო 13.5: დაამტკიცეთ, რომ არ იარსებებს ისეთი გრაფი, რომელშიც კენტიანი წვეროების რაოდენობა იქნება კენტი რიცხვი (ანუ ყველა გრაფში კენტიანი წვეროების რაოდენობა ლუწია: $\forall G = (V, E), \exists i \in \mathbb{N}, \sigma(G) = 2 \cdot i$).

სავარჯიშო 13.6: დახაზეთ გრაფი, რომელიც წარმოიშვება ნახ. 13.8 ნაჩვენები გრაფიდან *(ა)* $ABCD$, *(ბ)* $ABECA$, *(გ)* BEC და *(დ)* $FDBCE$ გზების ამოგდების შედეგად.



ნახ. 13.8: გრაფის მაგალითი

ახლა კი შეგვიძლია ***ეილერის თეორემის დამტკიცება:***
ჯერ დავამტკიცოთ, რომ თუ კენტიანი წვეროების რაოდენობა აჭარბებს ორს, ასეთი ციკლი ვერ იარსებებს. როდესაც ვიწყებთ გრაფზე შემოვლას, ცხადია, რომ ერთი წვეროდან უნდა დავიწყოთ და მეორეში უნდა დავამთავროთ. ყველა დანარჩენ წვეროში რამდენჯერაც *შევალთ*, ზუსტად იმდენჯერ უნდა *გამოვიდეთ*. აქედან გამომდინარე, მასთან მიერთებულ წიბოთა რაოდენობა უნდა იყოს *ლუწი*. ესე იგი, თუ კენტიან წვეროთა რაოდენობა ორზე მეტია, ერთიდან დავიწყებთ, მეორეში დავამთავრებთ, მაგრამ დაგვრჩება ისეთი წვეროც, რომელშიც გრაფის შემოვლისას შევალთ და ვეღარ გამოვალთ ისე, რომ უკვე გავლილ წვეროს მეორედ არ გადავუაროთ.
ახლა კი ინდუქციაზე დაყრდნობით დავამტკიცოთ, რომ ისეთ გრაფებში, სადაც კენტიან წვეროთა რაოდენობა ზუსტად ორია, იარსებებს ეილერის გახსნილი ციკლი, ხოლო ისეთში, სადაც კენტიანი წვერო არ არსებობს, ეილერის შეკრული ციკლი იარსებებს.
დასაწყისისათვის განვიხილოთ ერთ, ორ და სამ წიბოიანი გრაფები (ნახ. 13.9).

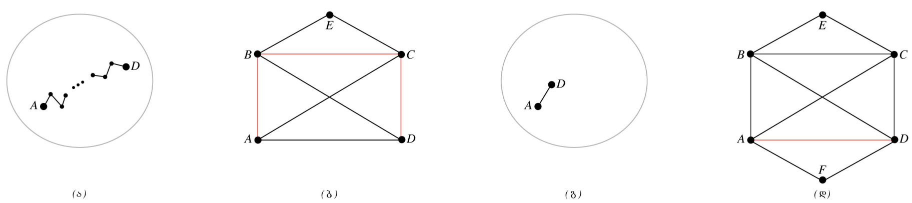


ნახ. 13.9: ერთ, ორ და სამ წიბოიანი გრაფები

ინდუქციის შემოწმება: ადვილი შესამოწმებელია, რომ ამ გრაფებში ეილერის თეორემა ჭეშმარიტია.

ინდუქციის დაშვება: დავუშვათ, რომ ეილერის თეორემა ჭეშმარიტია ყველა ბმული გრაფისათვის, რომლის წიბოთა რაოდენობა ნაკლებია რაღაცა n რიცხვზე.

ინდუქციის ბიჯი: დავამტკიცოთ თეორემა ნებისმიერი n წიბოიანი გრაფისათვის.
დავუშვათ, რომ ასეთ გრაფს ორი კენტიანი წვერო A და D აქვს. რადგან ჩვენ ბმულ გრაფებს განვიხილავთ, უნდა არსებობდეს გზა A წვეროდან D წვეროში, რომელიც კიდევ რაღაცა წვეროებზე გაივლის (ნახ. 13.10 (ა)). ასეთი გრაფის კონკრეტული მაგალითი მოყვანილია ნახაზში 13.10 (ბ) (იქ გამოყოფილი გზა A წვეროდან D წვეროში სხვა ფერითაა შეღებილი).



ნახ. 13.10: ზოგადი გრაფები

თუ ერთ-ერთ ასეთ გზას ამოვირჩევთ და მის წიბოებს გრაფიდან ამოვშლით (უნდა მივაქციოთ ყურადღება იმას, რომ ამოშლის შედეგად გრაფი არ დაიშალოს ცალკეულ ნაწილებად), მივიღებთ სხვა გრაფს, რომელსაც *მხოლოდ* ლუწიანი წიბოები აქვს.

სავარჯიშო 13.7: დაამტკიცეთ, რომ ზემოთ მოყვანილი გრაფიდან შერჩეული გზის ამოგდების შედეგად მიღებულ გრაფში *მხოლოდ* ლუწიანი წვეროები დაგვრჩება.

რადგან დარჩენილ გრაფში წიბოების რაოდენობა n რიცხვზე ნაკლები იქნება, მისთვის ჭეშმარიტი იქნება ეილერის თეორემა და ***იარსებებს*** ისეთი ჩაკეტილი ციკლი, რომელიც A წვეროში დაიწყება და მასშივე დასრულდება. შემდეგ კი *ამოგდებული გზის* წიბოებით გადავალთ A წვეროდან D წვეროში, რითაც საწყის n წიბოიან გრაფში შევადგენთ ეილერის ღია გზას.

ანალოგიური მსჯელობით შეგვიძლია დავამტკიცოთ, რომ ნებისმიერ n წიბოიან გრაფში, რომელიც არ შეიცავს კენტიან წვეროებს, არსებობს ეილერის ჩაკეტილი ციკლი, მხოლოდ აქ უკვე ორ მეზობელ წვეროს ვიღებთ და მათ შემაერთებელ წიბოს ამოვაგდებთ (იმ პირობით, რომ ამ წიბოს ამოგდების შემდეგ გრაფი ორ ნაწილად არ დაიშლება), რის შედეგადაც ორ კენტიანი წვეროს მქონე გრაფს ვიღებთ.

სავარჯიშო 13.8: დაამტკიცეთ, რომ თუ მოცემულია n წიბოიანი გრაფი, რომელიც არ შეიცავს კენტიან წვეროებს, მასში ეილერის ჩაკეტილი ციკლი იარსებებს.

სავარჯიშო 13.9: დაწერეთ ალგორითმი, რომელიც ნებისმიერი მოცემული გრაფისათვის განსაზღვრავს, არსებობს თუ არა მასში ეილერის ციკლი.

ალგორითმების თეორიასა და პრაქტიკაში ძალიან მნიშვნელოვანია ე.წ. ***ჰამილტონის ციკლის*** ამოცანა: მოცემული გრაფისთვის განსაზღვრეთ, შეიძლება თუ არა მასში მოვძებნოთ ისეთი გზა, რომელიც ***ყველა*** წვეროზე გაივლის ზუსტად ერთხელ და დაბრუნდება იმავე წვეროში, საიდანაც დაიწყო შემოვლა? აქ გასათვალისწინებელია ის ფაქტი, რომ გარკვეული წიბოები შემოვლისას შეიძლება გამოვტოვოთ.

სავარჯიშო 13.10: ნახ. 13.7-ში მოყვანილი გრაფებიდან რომლებს აქვთ ჰამილტონის ციკლი?

თუ ნებისმიერ გრაფში ეილერის ციკლის არსებობის დადგენა საკმაოდ ადვილად შეიძლება, ჰამილტონის ციკლის არსებობის დასადგენად დღეისათვის ცნობილი ყველა ალგორითმი ძალიან ნელა მუშაობს და დიდი გრაფებისათვის გამომთვლელ მანქანებზე ათასობით წელსაც კი მოანდომებს.
ეს შეიძლება იმით იყოს გამოწვეული, რომ ამ ამოცანისათვის ჯერ-ჯერობით ვერავინ მოიფიქრა სწრაფი ალგორითმი, ან იმით, რომ ასეთი ალგორითმი ***არ არსებობს***. მაგრამ რადგან ჰამილტონის ციკლის ამოცანა *უაღრესად მნიშვნელოვანია*, ეს ცენტრალური ღია საკითხია კომპიუტერულ მეცნიერებაში.

როგორც აქამდე ვნახეთ, გრაფების წარმოდგენა შეიძლება სიმრავლეების სახით. მეორენაირად გრაფის წარმოდგენა ე.წ. *ბმულობის მატრიცით* შეიძლება. მაგალითისათვის განვიხილოთ ნახ. 13.8-ში მოყვანილი გრაფი.

	A	B	C	D	E	F
A	0	1	1	1	0	1
B	1	0	1	1	1	0
C	1	1	0	1	1	0
D	1	1	1	0	0	1
E	0	1	1	0	0	0
F	1	0	0	1	0	0

ზოგადად, $A = (a_{i,j})_{i=1}^{n}$ რაიმე $G = (V, E)$ გრაფის ბმულობის მატრიცია, თუ $V = \{v_1, ..., v_n\}$ და

$$(v_i, v_j) \in E \Leftrightarrow a_{i,j} = 1.$$

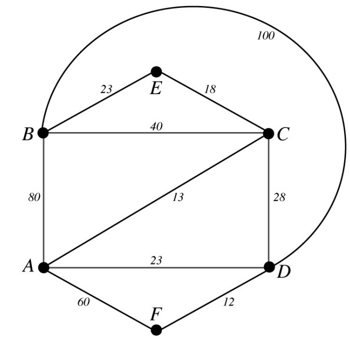
ზემოთ მოყვანილ მაგალითში გრაფის ბმულობის მატრიცი იქნება

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

სავარჯიშო 13.11: შეადგინეთ ნახ. 13.7-ში ნაჩვენები გრაფების ბმულობის მატრიცები.

სავარჯიშო 13.12: არსებობს თუ არა ისეთი გრაფი, რომლის ბმულობის მატრიცი არაა სიმეტრიული (ანუ $\exists i, j$ ისეთი, რომ $a_{i,j} \neq a_{j,i}$)?

განვიხილოთ შემდეგი ამოცანა: მოცემულია რუკა, რომელზეც ნაჩვენებია ქალაქები და მათი შემაერთებელი გზები (ანუ მოცემულია გრაფი, სადაც წვერო რომელიღაცა ქალაქს აღნიშნავს, ხოლო წიბო - ორი ქალაქის შემაერთებელ გზას).



ნახ. 13.11: რუკის გრაფი

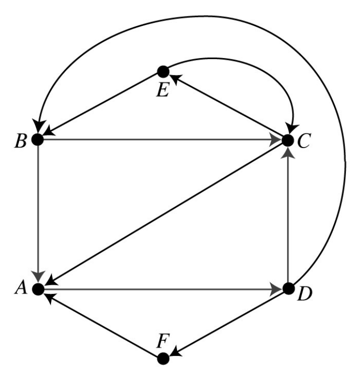
ამას გარდა, ყოველ წიბოს გვერდით აწერია რაღაც რიცხვი, რომელიც მოცემულ ორ ქალაქს შორის არსებული გზის სიგრძეს აღნიშნავს.

აღსანიშნავია, რომ ეს გრაფი გარკვეულ ინფორმაციას იძლევა და მისი წვეროების განლაგება სიბრტყეზე, ფაქტიურად, ნებისმიერი შეიძლება იყოს. ასე, მაგალითად, ADF სამკუთხედში შესაბამისი წიბოების წონები არ აკმაყოფილებენ სამკუთხედის უტოლობას, მაგრამ მოცემულ ამოცანაში შესაძლებელია, რომ, მაგალითად, გზა იყოს არაპირდაპირი.

მოცემული გრაფის ბმულობის მატრიცი იქნება

$$M = \begin{pmatrix} 0 & 80 & 13 & 23 & 0 & 60 \\ 80 & 0 & 40 & 100 & 23 & 0 \\ 13 & 40 & 0 & 28 & 18 & 0 \\ 23 & 100 & 28 & 0 & 0 & 12 \\ 0 & 23 & 18 & 0 & 0 & 0 \\ 60 & 0 & 0 & 12 & 0 & 0 \end{pmatrix}$$

თუ გრაფში წიბოები მიმართულია (ესე იგი, ისრითაა ნაჩვენები, თუ რომელი წვეროდან რომლისაკენაა მიმართული წიბო), მაშინ მისი შეერთების მატრიცი იქნება არასიმეტრიული (ნახ. 13.12).

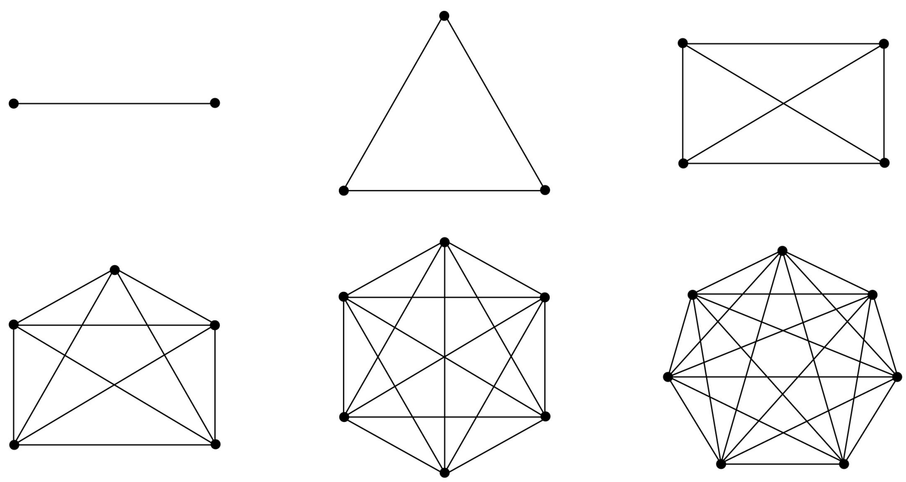


ნახ. 13.12: მიმართული გრაფი

ამ მაგალითში გვაქვს წიბო A წვეროდან D წვეროში, მაგრამ არა პირიქით. ზემოთ მოყვანილი გრაფის მატრიცი იქნება

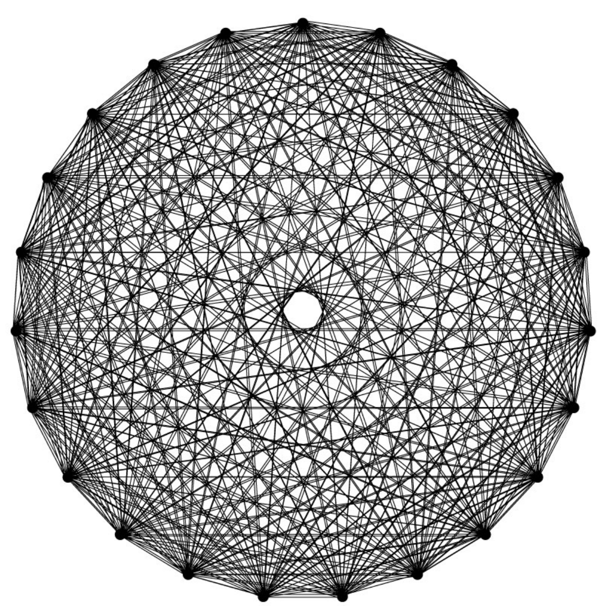
$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

გრაფს ეწოდება ***სრული***, თუ მისი ყველა წვერო ერთმანეთთანაა დაკავშირებული. n წვეროიანი სრული გრაფი აღინიშნება როგორც K_n. მახაზში 13.13 ნაჩვენებია ორ, სამ, ოთხ, ხუთ, ექვს და შვიდ წვეროიანი სრული გრაფები.



ნახ. 13.13: $K_i, 2 \leq i \leq 7$

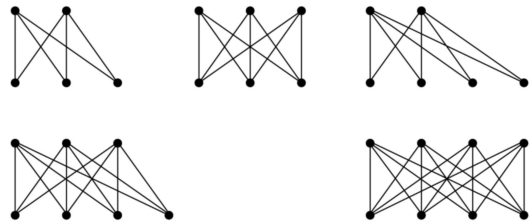
ცხადია, რომ დიდი სრული გრაფი საკმაოდ ძნელი დასახატია. მაგალითისათვის მოგვყავს K_{23} (ნახ. 13.14).



ნახ. 13.14: K_{23}

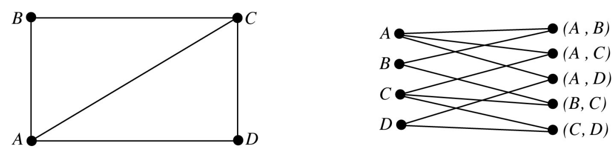
თეორიასა და პრაქტიკაში ძალიან მნიშვნელოვანია ე.წ. ***ორად გაყოფილი სრული*** გრაფი, რომლის წვეროები ორ სიმრავლედ შეგვიძლია გავყოთ ისე, რომ თითოეულ სიმრავლეში შესული წვეროები ერთმანეთთან შეერთებული არ იყოს, სამაგიეროდ ერთი სიმრავლის წვერო მეორე სიმრავლის *ყველა* წვეროსთან იყოს მიერთებული. ორად დაყოფილ სრულ გრაფს, რომლის ერთ სიმრავლეში n, ხოლო მეორეში კი m წვეროა, აღინიშნავენ როგორც $K_{n,m}$.

ნახაზში 13.15 მოყვანილია მცირე ზომის ორად დაყოფილი სრული გრაფები.



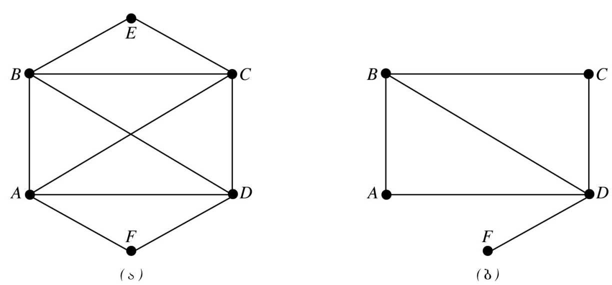
ნახ. 13.15: $K_{2,3}$, $K_{3,3}$, $K_{2,4}$, $K_{3,4}$, $K_{4,4}$

აღსანიშნავია, რომ ორად დაყოფილი გრაფი შეიძლება სრული არ იყოს, ანუ ერთი სიმრავლის წვერო მეორე სიმრავლის რომელიმე წვეროსთან მიერთებული არ იყოს.
ორად დაყოფილი გრაფების შესწავლა ძალიან მნიშვნელოვანი საკითხია, რადგან ***ნებისმიერ*** გრაფს შეესაბამება ერთი და მხოლოდ ერთი ორად დაყოფილი გრაფი, რომელიც შემდეგნაირად იგება:
ერთ სიმრავლეში გავაერთიანებთ მოცემული გრაფის წვეროებს, ხოლო მეორეში კი დავუმატებთ იმდენ ახალ წვეროს, რამდენი წიბოცაა მოცემულ გრაფში (ყოველ წიბოს ერთი ახალი წვერო შეესაბამება) და შემდეგ პირველი სიმრავლიდან ზუსტად ორ წვეროს მეორე სიმრავლის ერთ წვეროსთან შევაერთებთ, თუ ეს ორი წვერო საწყის გრაფში წიბოთი იყო შეერთებული (მაგალითისათვის იხ. ნახ. 13.16).



ნახ. 13.16: გრაფი და მისი შესაბამისი ორად დაყოფილი გრაფი

ასევე ძალიან მნიშვნელოვანია *ქვეგრაფის*, ანუ გრაფის „ნაწილის" ცნება. თუ ერთი გრაფი მეორედან წვეროებისა და მათი შემაერთებელი წიბოებისაგან ან წიბოების ნაწილისაგან შედგება, მაშინ ამ გრაფს მეორე გრაფის *ქვეგრაფს* უწოდებენ (ნახ. 13.17).



ნახ. 13.17: გრაფი (ა) და მისი ერთ-ერთი ქვეგრაფი

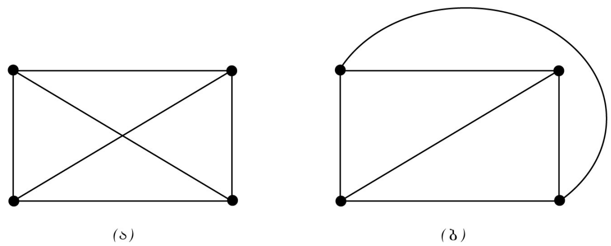
აღსანიშნავია, რომ ერთ გრაფს შეიძლება მრავალი ქვეგრაფი ქონდეს.

სავარჯიშო 13.13: მოიყვანეთ ნახ. 13.17 (ა)-ში მოყვანილი გრაფის ხუთი ქვეგრაფის მაგალითი.

ზოგადად, იმის გარკვევა, არის თუ არა ერთი გრაფი მეორეს ქვეგრაფი, ძალიან რთულია: დღეისათვის არ არის ცნობილი ისეთი ალგორითმი, რომელიც ნებისმიერი ორი G და G' გრაფისათვის *სწრაფად*, ანუ პოლინომიურ დროში გაარკვევს, არის თუ არა G გრაფი G' გრაფის ქვეგრაფი.

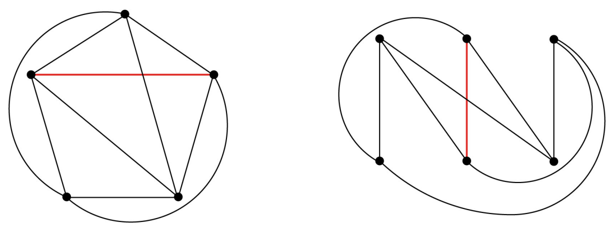
ახლა კი განვიხილოთ ნახ. 13.18-ში მოყვანილი გრაფის ორნაირი დიაგრამა.

როგორც ვხედავთ, ერთში ორი წიბო იკვეთება, მეორეში კი - არა. ისეთ გრაფს, რომლის დახატვა ისეთნაირად შეიძლება, რომ წიბოებმა ერთმანეთი არ გადაკვეთონ, ***პლანარული***, ანუ ***ბრტყელი*** ეწოდება.



ნახ. 13.18: ერთი გრაფის ორნაირი დახატვა

ძალიან მნიშვნელოვანია შემდეგი ამოცანა: მოცემული გრაფისათვის გაარკვიეთ, შეიძლება თუ არა მისი ბრტყლად დახატვა (ანუ არის თუ არა ეს გრაფი პლანარული).
როგორც აღმოჩნდა, უმცირესი *არაბრტყელი* გრაფების მაგალითებია K_5 და $K_{3,3}$.



ნახ. 13.19: მინიმალური არაბრტყელი გრაფები

აქედან გამომდინარე, ვერც ერთი გრაფი, რომელიც ამ ორიდან ერთ-ერთს მაინც შეიცავს, ბრტყლად *ვერ* დაიხატება.

ბუნებრივია შემდეგი შეკითხვა: რა საჭიროა გრაფის პლანარულობის დადგენა? რაში გვეხმარება ის ფაქტი, რომ გრაფი პლანარულია?
როგორც აღმოჩნდა, მრავალი ამოცანა, რომელიც ნებისმიერ (არაპლანარულ) გრაფზე რთული ამოსახსნელია, პლანარულისთვის ადვილად გადაიჭრება: პლანარული გრაფებისთვის შემუშავებულია ბევრი ისეთი ალგორითმი, რომელიც სწრაფად (ანუ პოლინომიურ დროში) ხსნის ამოცანას, მაგრამ არაპლანარულ გრაფზე არასწორად მუშაობს. ასე რომ, თუ დავადგენთ, რომ გრაფი პლანარულია, მასზე ასეთი სწრაფი ალგორითმის გამოყენება შესაძლებელი იქნება.

ამას გარდა, ძლაინ მნიშვნელოვანი ამოცანაა გრაფის მინიმალური გადაკვეთის წერტილებით სიბრტყეზე დახატვა მაგრამ, სამწუხაროდ, არც ამ ამოცანისთვისაა პოლინომიური (ანუ სწრაფი) ალგორითმი ცნობილი.