ალგორითმები და სირთულე

ალექსანდრე გამყრელიძე

სარჩევი

თავი 1

ამოხსნადი და არაამოხსნადი ამოცანები, მათი კლასიფიკაცია და პრაქტიკული მნიშვნელობა

1.1 პოლინომურ დროში ამოხსნადი და მათი „მსგავსი” რთული ამოცანები

- **ეილერის ციკლი გრაფში** (EC)
  მოცემულია: გრაფი $G = (V, E)$.
  პასუხი: „კი” ან „არა”
  განსაზღვრეთ, არსებობს თუ არა მოცემულ გრაფში ისეთი გზა, რომელიც ყველა წიბოზე გაივლის ერთხელ და მხოლოდ ერთხელ?
  ეილერის ციკლის ამოცანა ფართოდ გამოიყენება პრაქტიკაში. მაგალითად, მოცემულია ქალაქის რუკა, რომლის მიხედვითაც დასაგეგმია ნაგვის ან საფოსტო მანქანების მარშრუტი. ცხადია, რომ მანქანამ ქალაქის ყველა ქუჩაზე ერთხელ მაინც უნდა გაიაროს. დროის დასაზოგად უნდა ვიპოვნოთ ისეთი მარშრუტი, რომლითაც ყველა ქუჩაზე ზუსტად ერთხელ გავივლით (თუ ეს შესაძლებელია). შენიშვნა: აქ უნდა ჩავთვალოთ, რომ ყველა ქუჩაზე გავლა მოძრაობის ინტენსიურობაზე დამოკიდებული არაა.
  იგივე ამოცანა შემდეგნაირადაც შეიძლება დავსვათ: შესაძლებელია თუ არა მოცემული გეომეტრიული ფიგურის ხელის აუღებლად ქაღალდზე დახაზვა ისე, რომ უკვე დახაზული აღარ გადავხაზოთ?
  როგორც პირველ სემესტრში ვნახეთ, ამ ამოცანის ამოხსნა რთული არ არის:
  არამიმართულ ბმულ გრაფში არსებობს ეილერის გზა, თუ კენტი ვალენტობის წვეროების რაოდენობა ორზე მეტი არ არის. აღსანიშნავია, რომ თუ გრაფში კენტი ვალენტობის წვერო არ არსებობს, მაშინ იგი ეილერის ციკლს შეიცავს.
  სავარჯიშო 1.1: დაამტკიცეთ, რომ კენტი ვალენტობის მქონე წვეროების რაოდენობა ლუწია.
  თუ გრაფი მიმართულია (და, რა თქმა უნდა, ძლიერად ბმული), მაშინ
  იგი შეიცავს ეილერის ციკლს, თუ მის ყველა წვეროში შემავალი და გამავალი წიბოების რაოდენობა ტოლია;
  იგი შეიცავს ეილერის გზას u წვეროდან v წვეროში, თუ u წვეროს შემავალ წიბოთა რაოდენობა ერთით ნაკლებია გამავალზე, ხოლო v წვეროსთვის კი პირიქით: შემავალ წიბოთა რაოდენობა ერთით მეტია გამავალზე, ხოლო ყველა დანარჩენ წვეროში შემავალ და გამომავალ წიბოთა რაოდენობა ტოლია.

  სავარჯიშო 1.2: დაამტკიცეთ ზემოთ მოყვანილი გამონათქვამები (იხ. პირველი სემესტრის მასალა).

  სავარჯიშო 1.3: დაწერეთ ალგორითმი, რომელიც მოცემული გრაფისთვის ეილერის ციკლის არსებობას დაადგენს.

  თუ გრაფში ეილერის ციკლი (ან გზა) არსებობს, მისი დადგენაც სწრაფად შეიძლება: პირველ რიგში უნდა ვიპოვნოთ *ნებისმიერი* ციკლი, დავიხსომოთ და საწყისი გრაფიდან მისი წიბოები ამოვშალოთ. ცხადია,

რომ ამოშლის შემდეგ დარჩენილ გრაფში ეილერის ციკლის არსებობის პირობა შენარჩუნებული იქნება (რადგან ციკლში შემავალი წვეროების ვალენტობა ორით შემცირდება). იგივე პროცედურა რეკურსიულად გავიმეოროთ მანამ, სანამ გრაფში წიბოები არ გამოიღევა. დაგვრჩება საწყის გრაფში არსებული ციკლების სიმრავლე, რომელთა გაერთიანებაც მთელს გრაფს მოგვცემს.

სავარჯიშო 1.4: დაამტკიცეთ, რომ მიღებული სიმრავლის ციკლების გაერთიანება მართლაც მთელ საწყის გრაფს მოგვცემს და ნებისმიერი ორი ციკლი საერთო წიბოს არ შეიცავს.

სავარჯიშო 1.5: დაწერეთ ალგორითმი, რომელიც ზემოთ აგებული ციკლების სიმრავლისგან ეილერის გზას გამოითვლის და შეაფასეთ მისი ბიჯების რაოდენობის ზედა ზღვარი.

ეილერის ციკლის ალგორითმი ბევრ პრაქტიკულ (მათ შორის ზემოთ ხსენებულ ნაგვის შეგროვების) ამოცანას გადაგვიჭრიდა, მაგრამ, სამწუხაროდ, პრაქტიკაში არსებული ამოცანების შესაბამის გრაფებში უმეტეს შემთხვევაში ეილერის ციკლის პირობა არ კმაყოფილდება. ასეთ შემთხვევაში დასმულია უფრო ზოგადი

ჩინელი ფოსტალიონის ამოცანა მოცემულ G გრაფში იპოვნეთ ისეთი უმოკლესი მარშრუტი, რომელიც ყველა წიბოზე ერთხელ მაინც გადაივლის.

ამ ამოცანის გადასაჭრელად პირველ რიგში უნდა შევქმნათ სრული შეწონილი $G' = (V', E')$ გრაფი, რომელიც G გრაფის კენტიანი წვეროებისგან შედგება და $(u, v) \in E'$ წიბოს წონა G გრაფის შესაბამის წვეროებს შორის მინიმალური გზის სიგრძის ტოლია. G' გრაფზე ოპტიმალური დაწყვილების ამოცანის (perfect matching) ამოხსნა (ანუ ისეთი წიბოების სიმრავლის გამოყოფა, რომლებითაც მთელი გრაფი გადაიფარება და არც ერთ წყვილს საერთო წვერო არ აქვს, თან ასეთ სიმრავლეებს შორის ვირჩევთ ისეთს, რომელთა წიბოების წონების ჯამი ინიმალურია) გვაძლევს იმ წიბოთა სიმრავლეს, რომელიც უნდა დაემატოს G გრაფს. ამ სახით შექმნილ G_1 გრაფზე ეილერის ციკლი უნდა არსებობდეს, რადგან ყოველ ორ კენტიან წვეროს ვაერთებთ წიბოთი და ყველა წვერო ლუწიანი გამოვა.

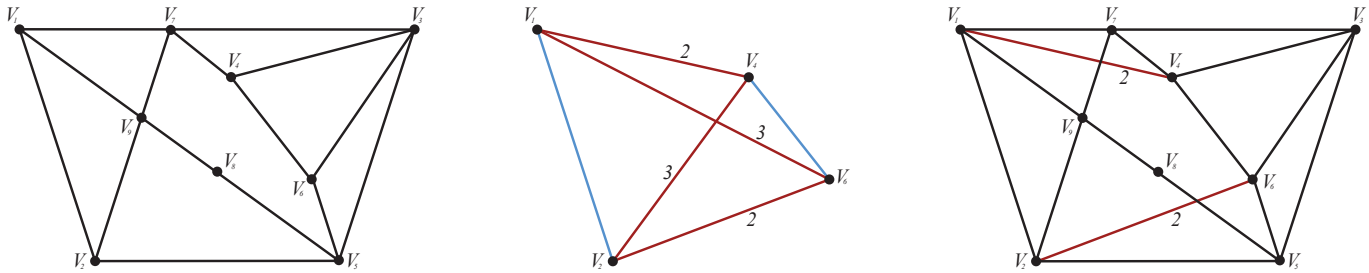
სავარჯიშო 1.6: დაწერეთ მოცემული გრაფის ოპტიმალური დაწყვილების გამოთვლის ალგორითმი და შეაფასეთ მისი ბიჯების რაოდენობის ზედა ზღვარი.

შენიშვნა: აქ გასათვალისწინებელია ის ფაქტი, რომ G' გრაფის ოპტიმალურ დაწყვილებაში G გრაფში არსებული წიბოები არ უნდა შედიოდეს.

თუ G_1 გრაფზე ეილერის ციკლში „ახალ” წიბოებს შესაბამისი უმოკლესი გზით შევცვლით, მივიღეთ უმოკლეს გზას, რომელიც ყველა წიბოზე ერთხელ მაინც გადაივლის.

რაც შეეხება მიმართულ გრაფს, G' გრაფის წიბოები უნდა იყოს მიმართული ნაკლებ გამომავალი წიბოების მქონე წვეროდან ნაკლებ შემავალი წიბოების მქონე წვეროსკენ, რითაც შეიქმნება ორად დაყოფილი მიმართული გრაფი და ოპტიმალური დაწყვილების ამოცანაც მასზე უნდა გადაიჭრას.

მაგალითისთვის განვიხილოთ 1.1 ნახაზში მოყვანილი G (ა), G' (ბ) და G_1 (გ) გრაფები.



(ა) (ბ) (გ)

ჩინელი ფოსტალიონის ამოცანის ამოსახსნელი გრაფები

სავარჯიშო 1.7: 1.1 ნახაზში მოყვანილი G_1 გრაფის მეშვეობით გამოიანგარიშეთ G გრაფის ოპტიმალური მარშრუტი.

სავარჯიშო 1.8: დაამტკიცეთ, რომ ზემოთ მოყვანილი მეთოდით გამოთვლილი გზა მინიმალური იქნება.

სავარჯიშო 1.9: დაწერეთ ალგორითმი, რომელიც ჩინელი ფოსტალიონოს ამოცანას გადაჭრის.

სავარჯიშო 1.10: რა სიგრძის იქნება ჩინელი ფოსტალიონის ამოცანის ამოხსნა ხეებისთვის?

ეილერის ციკლის პოვნის პარადიგმა ძალიან მნიშვნელოვანია გრაფებზე პარალელური ალგორითმების აგებაში. ხშირად გრაფზე აგებენ დამფარავ ხეებს და მერე იყენებენ ეილერის ციკლების ტექნიკას მათი დამუშავებისთვის.

ამას გარდა, მნიშვნელოვანია გრაფში ეილერის ციკლების რაოდენობის გამოთვლაც.

- **ჰამილტონის ციკლი გრაფში** (HC)

  მოცემულია: გრაფი $G = (V, E)$.

  პასუხი: „კი” ან „არა”

  განსაზღვრეთ, არსებობს თუ არა მოცემულ გრაფში ისეთი გზა, რომელიც ყველა წვეროზე გაივლის ერთხელ და მხოლოდ ერთხელ?

  ერთი შეხედვით ეს ამოცანა ეილერის ციკლის ამოცანის მსგავსია და შეიძლება ვიფიქროთ კიდეც, რომ უფრო მარტივად ამოიხსნაა შესაძლებელი. მაგრამ მისთვის პოლინომურ დროში მომუშავე ალგორითმი ჯერ-ჯერობით არაა ცნობილი. უფრო მეტიც: როგორც შემდგომში ვნახავთ, არც ისაა ცნობილი, შესაძლებელია თუ არა ამ ამოცანისთვის ამდაგვარი ალგორითმის დაწერა.

  ანალოგიურად შეიძლება ჩამოვაყალიბოთ ჰამილტონის გზის ამოცანაც (იგივე ჰამილტონის ციკლი, მხოლოდ საწყის წვეროში დაბრუნება აღარაა საჭირო).

  ჰამილტონის გზის ამოცანა უაღრესად მნიშვნელოვანია პრაქტიკაშიც. ასე, მაგალითად, გენურ ინჟინერიაში აქვთ გენეტური კოდის ნაწილები $a_1, ..., a_n$ და წესები, რომელი ნაწილი რომელს შეიძლება გადაებას. თუ ამ კოდებს განვიხილავთ, როგორც გრაფის წვეროებს და ერთმანეთზე გადაბმად ნაწილებს შორის გავავლებთ (მიმართულ) წიბოს, ჰამილტონის ციკლი გვიჩვენებს იმ გენეტურ კოდს, რომელიც შეიძლება შეიქმნას მოცემული ნაწილებიდან.

  მეორე გამოყენება შეიძლება იყოს სკოლის ავტობუსის მარშრუტის დაგეგმვა: თუ მოცემული გვაქვს მოსწავლეების საცხოვრებელი მისამართები (გრაფის წვეროები) და შემაერთებელი გზები (გრაფის წიბოები), ჰამილტონის (მინიმალური) ციკლი ოპტიმალურ მარშრუტს მოგვცემს.

  გარდა ამისა, როგორც ვნახანთ, ჰამილტონის ციკლის ამოცანა ე.წ. NP-სრულ ამოცანათა კლასს განეკუთვნება, რაც იმას ნიშნავს, რომ მისი სწრაფად ამოხსნა ჩვენს გარშემო არსებული ამოცანების უმეტესობის სწრაფად ამოხსნის გზას გვაჩვენებდა.

- **უმოკლესი გზა გრაფის წვეროებს შორის**

  მოცემულია: შეწონილი გრაფი $G = (V, E)$ და მისი ორი წვერო $u, v \in V$.

  პასუხი: მიმდევრობა $(x_1, ..., x_k)$, სადაც $x_i \in V$, $x_i \neq x_j$, თუ $i \neq j$.

  იპოვნეთ უმოკლესი გზა გრაფში მოცემულ ორ წვეროს შორის.

  უმოკლესი გზის ამოცანა უაღრესად მნიშვნელოვანია პრაქტიკაში, მაგ. მარშრუტიზაციის ამოცანებში.

  ამ ამოცანისთვის სწრაფი ალგორითმი იხ. მეორე სემესტრის კონსპექტში.

- **მაქსიმალური მარტივი გზა გრაფის წვეროებს შორის**

  მოცემულია: შეწონილი გრაფი $G = (V, E)$ და მისი ორი წვერო $u, v \in V$.

  პასუხი: მიმდევრობა $(x_1, ..., x_k)$, სადაც $x_i \in V$, $x_i \neq x_j$, თუ $i \neq j$.

  იპოვნეთ მაქსიმალური მარტივი გზა ამ ორ წვეროს შორის (მარტივი ეწოდება ისეთ გზას, რომელიც ციკლებს არ შეიცავს).

  არც ამ ამოცანისთვისაა პოლინომურ დროში მომუშავე ალგორითმი ცნობილი და არც არავინ იცის, შესაძლებელია თუ არა ამ ამოცანისთვის ამდაგვარი ალგორითმის დაწერა.

  ერთი შეხედვით ეს ამოცანა პრაქტიკულ გამოყენებას მოკლებულია, მაგრამ თუ დავუშვებთ, რომ ერთი წვეროდან მეორეზე გადასვლა რაიმე ეკონომიკურ ტრანსაქციასთანაა დაკავშირებული და გადასული წიბოს წონა ფინანსურ მოგებას (ან უარყოფითი წონის შემთხვევაში წაგებას) აღნიშნავს, მაშინ მაქსიმალური გზა მაქსიმალური მოგების ტოლფასია.

  აქვე უნდა აღინიშნოს ამ ამოცანის ჰამილტონის ციკლთან კავშირი: თუ არაშეწონილ გრაფში ჰამილტონის ციკლი არსებობს, მაშინ სწორედ ეს იქნება მაქსიმალური გზა.

  სავარჯიშო 1.11: როგორ შეიძლება მაქსიმალური გზის ამოცანის ამოხსნით ჰამილტონის ციკლის (და, შესაბამისად, გზის) ამოცანის გადაჭრა?

- **უდიდესი სრული ქვეგრაფი** *Clique*

  მოცემულია: გრაფი $G = (V, E)$.

  პასუხი: $k \in \mathbb{N}$.

  რა არის ისეთ წვეროთა მაქსიმალური ქვესიმრავლის ზომა, რომელშიც ნებისმიერი ორი წვერო ერთმანეთთან იქნება დაკავშირებული (ანუ G გრაფის ყველაზე დიდი სრული ქვეგრაფის წვეროების რაოდენობა)

  ეს ამოცანაც რთულია: მისთვის სწრაფი ალგორითმი ცნობილი არაა (და არც ისაა ცნობილი, ამოხსნადია თუ არა სწრაფად)

  უდიდესი საერთო ქვეგრაფის ამოცანა (შემოკლებით კლიკი, Clique) მსგავსი ობიექტების (კლასტერების) აღმოჩენაში უმთავრეს როლს თამაშობს. მაგალითად, თუ საჭიროა გაყალბების აღმოჩენა დოკუმენტების (ფულის, ფასიანო ქაღალდების, სადაზვეო მაქინაციების და სხვა) დიდ სიმრავლეში, პირველ რიგში უნდა განისაზღვროს „მსგავსება” (როგორც წესი, ეს ე.წ. „ჰემინგის მანძილის” ან შესაბამის ვექტორებს შორის კუთხის კოსინუსით ხდება). არსებულ დოკუმენტებს თუ განვიხილავთ, როგორც გრაფის წვეროებს, ხოლო მსგავს დოკუმენტებს წიბოებით შევაერთებთ (რაც უფრო დიდია მსგავსების კოეფიციენტი, მით უფრო მოკლეა მათ შორის გზა). როგორც წესი, არსებულ დოკუმენტებს (მაგ. ფულის ნომრებს) შორის დიდი მსგავსება არ უნდა იყოს, მაგრამ დიდი რაოდენობის დოკუმენტების გაყალბების შემთხვევაში მსგავსება უფრო დიდია ხოლმე. აქედან გამომდინარე, უდიდესი სრული ქვეგრაფი „მსგავს” ობიექტების დიდ რაოდენობას იპოვნის, რისი უფრო დაწვრილებითი ანალიზის შემდეგ გაყალბების პოვნა უფრო მარტივი იქნება.

  სწორედ ამ იდეებზე დაყრდნობით განავითარა ამერიკის გადასახადების სამსახურმა ყალბი დოკუმენტაციის აღმოჩენის სისტემა.

  როგორც ვთქვით, ამ ამოცანისთვის პოლინომური ალგორითმი არ არის ცნობილი (და არც ის ვიცით, შესაძლებელია თუ არა ასეთის არსებობა). როგორც შემდგომში ვნახავთ, ეს ამოცანაც ე.წ. NP-სრულ ამოცანათა კლასს განეკუთვნება და, როგორც გამოცდილებამ აჩვენა, მისი ზუსტად ამოხსნის ნაცვლად სხვა გზებია მოსაძებნი.

  ერთ-ერთი პირველი გამოსავალია არა *უდიდესი*, არამედ (ლოკალურად) მაქსიმალური სრული ქვეგრაფის მოძებნა, რაც ხშირად პრაქტიკულ ამოცანებში საკმარისია ხოლმე. ლოკალურად მაქსიმალური ეწოდება ისეთ სრულ ქვეგრაფს, რომლისთვისაც საწყისი გრაფის ნებისმიერი სხვა წვეროს მიერთება სრულობას დაუკარგავს.

  ლოკალურად მაქსიმალური სრული ქვეგრაფის საპოვნელად შეიძლება წვეროების დალაგება მათი ვალენტობის კლებადობის მიხედვით, პირველ (მაქსიმალური ვალენტობის მქონე) წვეროს კლიკის სიმრავლეში ჩავრთავთ, შემდეგ რიგ-რიგობით ჩავყვებით სიას და ჩავრთავთ იმ წვეროებს, რომლებიც სრულ გრაფს შეგვიქმნის. თუ შესაბამისი წვეროს კლიკთან მიკუთვნებას ერთი ბიტით აღვნიშნავთ და მათ ყველას ე.წ. ბიტ-ვექტორში გავაერთიანებთ, ამ ალგორითმის რეალიზაცია $O(|V| + |E|)$ დროში იქნება შესაძლებელი.

  სავარჯიშო 1.12: დაწერეთ პროგრამა, რომელიც ლოკალურად მაქსიმალურ სრულ ქვეგრაფს წრფივ დროში გამოითვლის.

  მეორე ხერხი, რაც შეიძლება პრაქტიკაში გამოგვადგეს, არა *სრული*, არამედ *მჭიდრო* ქვეგრაფების ძებნაა. ხშირად პრაქტიკულ ამოცანებში სრული ქვეგრაფების ნაცვლად მჭიდრო ქვეგრაფის მოძებნაც უკვე საკმარისია. აღსანიშნავია, რომ სრული ქვეგრაფი ამავდროულად მაქსიმალურად მჭიდრო ქვეგრაფია.

  მინიმუმ k რიგის (ვალენტობის) ქვეგრაფის პოვნა (ანუ ისეთის, რომლის წვეროების რიგიც არის მინიმუმ k) ადვილად შეიძლება: გრავში ამოვაგდებთ იმ წვეროებს (და შესაბამის წიბოებს), რომელთა რიგიც უფრო დაბალია. ამ პროცედურამ შეიძლება გამოიწვიოს დარჩენილი წვეროების რიგის შემცირება (თუ ისინი საკმარისად ბევრ ამოგდებულ წვეროსთან იყვნენ მიერთებულნი). ამ პროცედურას ვიმეორებთ მანამ, სანამ არ დაგვრჩება მხოლოდ შესაბამისად მაღალი რიგის წვეროები.

  სავარჯიშო 1.13: დაწერეთ ალგორითმი $O(|V| + |E|)$ რიგის ალგორითმი, რომელიც k-მჭიდრო ქვეგრაფს გამოითვლის.

  შენიშვნა: გამოიყენეთ გრაფის მეზობლობის სიით აღწერა და შეზღუდული ზომის პრიორიტეტული რიგი.

  ბოლოს უნდა აღინიშნოს, რომ თუ გრაფი პლანარულია, მასში მაქსიმალური სრული ქვეგრაფი 2, 3 ან 4 წვეროსგან უნდა შედგებოდეს (ტოპოლოგიის ერთ-ერთი თეორემა გვეუბნება, რომ K_5, ანუ 5 წვეროიანი სრული გრაფი პლანარული ვერ იქნება).

  სავარჯიშო 1.14: დაწერეთ ალგორითმი, რომელიც $O(|V|)$ დროში გამოითვლის პლანარული გრაფის მაქსიმალურ ქვეგრაფს.

  რაც შეეხება *Clique* ამოცანის ზოგად ამოხსნას, მისთვის მრავალი სხვადასხვა მიდგომაა განვითარებული, რაც ძირითადად ალბათური ევრისტიკის იდეებს ეყრდნობა, მაგრამ ეს თემა ჩვენი ამ სემესტრის კურსის ფარგლებს ცდება და შემდეგში უნდა იქნას განხილული.

- **იზოლირებული წვეროების სიმრავლე** IS

  მოცემულია: გრაფი $G = (V, E)$.

  პასუხი: $k \in \mathbb{N}$.

  რა არის ისეთ წვეროთა მაქსიმალური ქვესიმრავლის ზომა, რომელშიც ნებისმიერი ორი წვერო ერთმანეთთან არ იქნება დაკავშირებული (ანუ G გრაფის ყველაზე დიდი იზოლირებული წვეროების სიმრავლის ელემენტების რაოდენობა)

- **გრაფის გადაფარვა** VC

  მოცემულია: გრაფი $G = (V, E)$.

  პასუხი: $k \in \mathbb{N}$.

  რა არის ისეთ წვეროთა მინიმალური ქვესიმრავლის ზომა, რომლითაც გრაფის ყველა წიბო გადაიფარება? (თუ e წიბო v წვეროს შეიცავს, მაშინ ამბობენ, რომ v გადაფარავს e წიბოს)

- **გრაფის ქრომატული რიცხვი** CN

  მოცემულია: გრაფი $G = (V, E)$.

  პასუხი: $k \in \mathbb{N}$.

  მოცემული გრაფისთვის განსაზღვრეთ, მინიმუმ რამდენ ფრად შეიძლება მისი შეღებვა.

  განმარტება 1.15: მოცემულია გრაფი $G = (V, E)$. მისი **შეღებვა** ეწოდება ისეთ $\phi : V \to \mathbb{N}$ ასახვას, რომელიც ყოველ წვეროს რაიმე ნატურალურ რიცხვს (ფერს) შეუსაბამებს ისე, რომ მეზობლებს (წიბოთი შეერთებულ წვეროებს) ერთი და იგივე ფერი (ერთი და იგივე რიცხვი) არ შეესაბამებოდეს: $(u, v) \in E \Rightarrow \phi(u) \neq \phi(v)$.

ძალიან მნიშვნელოვანია შემდეგი ორი ამოცანა:

განმარტება 1.16: $G_1 = (V_1, E_1)$ და $G_2 = (V_2, E_2)$ გრაფს ეწოდება ერთმანეთის იზომორფული (აღინიშნება $G_1 \cong G_2$), თუ მოიძებნება ისეთი ბიექცია $f : V_1 \to V_2$, რომ $(v, u) \in E_1 \Leftrightarrow (f(v), f(u)) \in E_2$ (ანუ G_1 გრაფში ორი წვერო შეერთებულია წიბოთი მაშინ და მხოლოდ მაშინ, თუ მეორე გრაფში მათი სახეებია შეერთებული წიბოთი).

განმარტება 1.17: $G' = (V', E')$ გრაფს ეწოდება $G = (V, E)$ გრაფის ქვეგრაფი, თუ $V' \subset V$ და $(u, v) \in E' \Rightarrow (u, v) \in E$.

- **გრაფთა იზომორფიზმი** GI

  მოცემულია: ორი გრაფი G_1, G_2

  პასუხი: „კი" ან „არა"

  განსაზღვრეთ, არსებობს თუ არა მოცემულ გრაფებს შორის ერთმანეთის იზომორფიზმი, ანუ $G_1 \cong G_2$

- **ქვეგრაფის იზომორფიზმი** SGI

  მოცემულია: ორი გრაფი G_1, G_2

  პასუხი: „კი" ან „არა"

  განსაზღვრეთ, მოიძებნება თუ არა G_1 გრაფის ისეთი G' ქვეგრაფი, რომ $G' \cong G_2$?

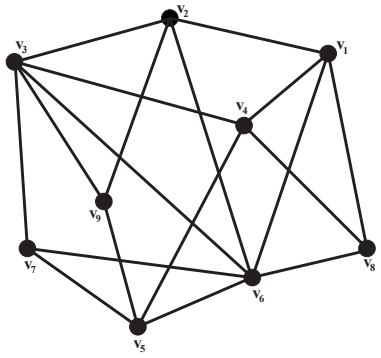
- **კომი ვოიაჟერის (მოგზაური ვაჭრის) ამოცანა** TSP

  მოცემულია: შეწონილი გრაფი G

  პასუხი: $k \in \mathbb{R}$

  განსაზღვრეთ ისეთი მინიმალური მარშრუტი, რომელიც ყველა წვეროზე ერთხელ მაინც გაივლის

სავარჯიშო 1.18: განვიხილოთ გრაფი G, რომელიც მოცემულია შემდეგ ნახაზში.
სავარჯიშო გრაფი
მასში შეიძლება გამოიყოს სამ ელემენტიანი იზოლირებული სიმრავლე $\{v_7, v_9, v_4\}$. შესაძლებელია თუ არა მასში უფრო დიდი იზოლირებული სიმრავლის პოვნა? რას მოგვცემს პასუხად $IS(G)$?



სავარჯიშო 1.19: ზემოთ მოყვანილი G გრაფისთვის რა იქნება $Clique(G)$? $CN(G)$? $HC(G)$? $VC(G)$? $TSP(G)$? არსებობს თუ არა მასში ეილერის ციკლი? (პასუხი დაამტკიცეთ)

სავარჯიშო 1.20: იგივე G გრაფში იპოვნეთ მაქსიმალური მარტივი გზა v_1 და v_2 წვეროებს შორის.

სავარჯიშო 1.21: C++ ენაზე დაწერეთ პროგრამა, რომელიც მატრიცის სახით მოცემული გრაფისთვის გადაჭრის ეილერის ციკლის ამოცანას.

პარაგრაფის ბოლოს კვლავ მოვიყვანოთ რამოდენიმე მნიშვნელოვანი ამოცანა.

- **ნატურალური რიცხვის სიმარტივის ტესტი**

  მოცემულია: ნატურალური რიცხვი $n \in \mathbb{N}$.

  პასუხი: „კი” ან „არა”

  დაადგინეთ, მარტივია თუ არა n.

  შენიშვნა: ეს ამოცანა ანტიკური დროიდან მოყოლებული რთულად ითვლებოდა. მისი უდიდესი პრაქტიკული გამოყენების (მაგალითად, კრიპტოგრაფიაში) მიუხედავად, სწრაფი ალგორითმი ცნობილი არ იყო, სანამ 2002 წელს ინდოელმა ინფორმატიკოსებმა $O(k^{12})$ ზედა ზღვრის მქონე ალგორითმი შეიმუშავეს, რომელიც შემდგომში კიდევ უფრო გაუმჯობესდა. დაწვრილებითი ინფორმაციისთვის იხილეთ

  Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, PRIMES is in P

  Annals of Mathematics, vol. 160, pp. 781 - 793, Springer Verlag, 2004

  აღსანიშნავია, რომ ალგორითმის დროის ზედა ზღვარი ყოველთვის მონაცემის სიგრძეზეა დამოკიდებული, ამიტომაც ზემოთ მოყვანილ ფორმულაში ვიხმარეთ პარამეტრი k, რომელიც მოცემული n რიცხვის ბიტების რაოდენობას აღნიშნავს: $k = \log n$.

- **გრაფის პლანარულობა**

  მოცემულია: გრაფი G

  პასუხი: „კი” ან „არა”

  დაადგინეთ, არის თუ არა იგი პლანარული?

ისეთი ამოცანების მაგალითებად, რომელთათვისაც რეალურ დროში ამოხსნადი ალგორითმი ცნობილი არაა (მაგრამ არც ისაა დამტკიცებული, რომ მათთვის ასეთი ალგორითმი არ არსებობს), შეიძლება მოვიყვანოთ:

- **რიცხვის ფაქტორიზაცია**

  მოცემულია: ნატურალური რიცხვი $n \in \mathbb{N}$

  პასუხი: ნატურალურ რიცხვთა მიმდევრობა $p_1, ..., p_h$

  დაშალეთ n მარტივ მამრავლებად: $n = p_1, ..., p_h$

- **გრაფის ოპტიმალურად დახატვა**

  მოცემულია: გრაფი G

  პასუხი: გრაფის ნახაზი სიბრტყეზე

  დახაზეთ გრაფი სიბრტყეზე ისე, რომ წიბოების გადაკვეთის რაოდენობა მინიმალური იყოს.

- **ბულის ფუნქციათა ჭეშმარიტება (SAT)**

  მოცემულია: ბულის ალგებრის n ცვლადზე დამოკიდებული ფუნქცია $f : \mathbb{B}^n \to \mathbb{B}$, რომელიც ჩაწერილია კონიუნქციური ნორმალური ფორმით

  პასუხი: „კი" ან „არა"

  არსებობს თუ არა მისი ცვლადების ისეთი მნიშვნელობები, რომ ეს ფუნქცია იყოს ჭეშმარიტი (ანუ იღებდეს მნიშვნელობას 1)?

1.2 ოპტიმიზაციისა და გადაწყვეტილების ამოცანები

ადვილი შესამჩნევია, რომ აქ მოყვანილ ამოცანებში ზოგიერთის პასუხია „კი„ ან „არა" და ზოგიერთის – რაღაც კონკრეტული რიცხვი ან სხვა მათემატიკური სტრუქტურა, რომელიც გარკვეულ წინაპირობას აკმაყოფილებს.
ისეთ ამოცანებს, რომლთა პასუხი შეიძლება იყოს „კი„ ან „არა", ***გადაწყვეტილების ამოცანა*** ეწოდება, ხოლო მეორე ტიპისას კი - ***ოპტიმიზაციის***.
გადაწყვეტილების ამოცანებია, მაგალითად, ჰამილტონის ციკლის, ეილერის ციკლის, გრაფის პლანარულობის ტესტის, სიმარტივის ტესტის, გრაფის იზომორფიზმის, ქვეგრაფის იზომორფიზმის და სხვა ამოცანები. მათში იმის დადგენაა საჭირო, ეკუთვნის თუ არა მონაცემი ამა თუ იმ კლასს: თუ მოცემული გვაქვს ყველა შესაძლო პლანარული გრაფის სიმრავლე, რომელსაც ჩვენ პირობითად ვუწოდებთ G_P, მოცემული G გრაფისთვის მხოლოდ შემდეგი საკითხის გარკვევაა საჭირო: $G \in G_P$?
ასევე, ჰამილტონის ციკლის მქონე გრაფთა სიმრავლე თუ აღინიშნება როგორც G_H, ჰამილტონის ციკლის ამოცანა იგივეა, რაც შეკითხვა $G \in G_H$? ამიტომ ამდაგვარ ამოცანებს უწოდებენ „გადაწყვეტილების ამოცანას": გადაწყვიტეთ, ეკუთვნის თუ არა მონაცემი რაღაც სიმრავლეს (ატარებს თუ არა გარკვეულ თვისებას).

ამისგან განსხვავებულია, მაგალითად, გადაფარვის ამოცანა VC. აქ ყველა შესაძლო გადაფარვიდან უნდა ავირჩიოთ მინიმალური. ასევე CN შეღებვის ამოცანაც: ყველა შესაძლო შეღებვიდან მინიმალური; TSP: ყველა შესაძლო მარშრუტიდან, რომელიც ყველა ქალაქზე გადის, მინიმალური და ა.შ. სხვა სიტყვებით რომ ვთქვათ, აქ შესაძლო ამონახსნებიდან ვირჩევთ რაღაც თვალსაზრისით ოპტიმალურს. ამიტომაც ასეთი ტიპის ამოცანებს უწოდებენ „ოპტიმიზაციისას".

განმარტება 1.22: თუ მოცემულია რაიმე გადაწყვეტილების ამოცანა A მონაცემთა სიმრავლით X, მაშინ $L_A = \{x \mid x \in X,\ A(x) =$კი$\}$ მონაცემთა ქვესიმრავლეს (ისეთ მონაცემთა სიმრავლეს, რომლებზეც ამ ამოცანის პასუხი იქნებოდა „კი") ამ A ამოცანის ***ენა*** ეწოდება.

სავარჯიშო 1.23: სიტყვიერად ახსენით, რა არის $L_{EC}, L_{TSP_k}, L_{VC_k}$ და L_{CN_k}.

საინტერესოა კავშირი ოპტიმიზაციისა და გადაწყვეტილების ამოცანებს შორის.
წარმოვიდგინოთ, რომ გვაქვს შემდეგი გადაწყვეტილების ამოცანა:

გრაფის k ფრად შეღებვა (CN_k)
მოცემულია: გრაფი G და რიცხვი $k \in \mathbb{N}$
პასუხი: „კი" ან „არა"
გაეცით პასუხი შეკითხვას: შეიძლება თუ არა G გრაფის k ფრად შეღებვა?

თუ გვაქვს ამ ამოცანის ალგორითმი, ადვილად შევძლებთ CN ოპტიმიზაციის ამოცანის გადაჭრასაც:

ალგორითმი 1.1: გრაფის შეღებვის ალგორითმი
მოცემულია: გრაფი G

```
1: k = 1;
2: while(CN_j(G) = „არა");
3: k++
4: return(k)
```

ანალოგიურად, თუ გვაქვს VC_k გადაწყვეტილების ამოცანის ალგორითმი, რომელიც მოგვცემს პასუხს „კი" მაშინ და მხოლოდ მაშინ, თუ მოცემული გრაფი გადაიფარება k წვეროთი, მისი მეშვეობით ადვილად შეიძლება გადაიჭრას შესაბამისი ოპტიმიზაციის ამოცანაც.

სავარჯიშო 1.24: ჩამოაყალიბეთ IS, $Clique$, TSP და VC შესაბამისი გადაწყვეტილების ამოცანები.

სავარჯიშო 1.25: მოცემულია IS_k, $Clique_k$, TSP_k და VC_k გადაწყვეტილების ამოცანების ალგორითმები. მათი გამოყენებით დაწერეთ შესაბამისი ოპტიმიზაციის ამოცანების ალგორითმები.

1.3 არაამოხსნადი ამოცანები

ერთ-ერთი ყველაზე ძველი ამოცანა, რომელსაც ალგორითმული ამოხსნა არ აქვს (გარკვეული რესურსებით შეზღუდვის გათვალისწინებით), ანტიკური საბერძნეთიდან მოდის და რამოდენიმე ათასი წელი გადაუჭრელი რჩებოდა - ფიგურების ფარგლითა და სახაზავით აგება (იხ. პირველი სემესტრის კურსი „ალგორითმების შესავალი"). როგორც XIX საუკუნეში მათემატიკოსებმა დაამტკიცეს, შეუძლებელია ისეთი ალგორითმების პოვნა, რომლითაც ერთეულოვანი წრის ფართობის მქონე კვადრატის აგებას შევძლებდით ფარგლისა და სახაზავის გამოყენებით. სამაგიეროდ შეიძლება ასეთი ფართობის მქონე კვადრატს ნებისმიერი სიზუსტით მივუახლოვდეთ - ამოცანის ოდნავი შეცვლისას იგი ამოხსნადი ხდება: მოცემულია წრე ფართობით S_1 და რაიმე დადებითი რიცხვი $\epsilon > 0$ (სიზუსტე). ფარგლისა და სახაზავის გამოყენებით ააგეთ ისეთი კვადრატი, რომლის ფართობი S_2 მოცემული სიზუსტით იქნება მოცემული წრის ფართობის სიახლოვეში, ანუ $|S_1 - S_2| \leq \epsilon$.
მეორე მაგალითად შეიძლება ჰილბერტის X პრობლემის მოყვანა: მოცემულია ნებისმიერი დიოფანტეს პოლინომიური განტოლება მთელი კოეფიციენტებით: $a_n \cdot x_n^n + \cdots + a_1 \cdot x_1^1 + a_0 = 0$, სადაც $a_i \in \mathbb{Q}, 0 \leq i \leq n, n \in \mathbb{N}$. შეიმუშავეთ მეთოდი (ანუ ალგორითმი), რომელიც ნებისმიერი ასეთი პოლინომისათვის გაგვცემდა პასუხს (კი ან არა) შეკითხვაზე, აქვს თუ არა ამ განტოლებას მხოლოდ რაციონალური ფესვები? როგორც აღმოჩნდა, არც ამ ამოცანას აქვს ალგორითმული ამოხსნა (აქვე შევნიშნოთ, რომ გარკვეულ კონკრეტულ შემთხვევებში ეს საკითხი გადაიჭრება. აქ ლაპარაკია იმაზე, რომ არ არსებობს ისეთი ალგორითმი, რომელიც ნებისმიერი მონაცემისთვის გაგვცემდა პასუხს).
აღსანიშნავია, რომ ეს ამოცანები ალგორითმის განსაზღვრებაზე ადრე ჩამოყალიბდა და წმინდა მათემატიკური ხერხებით იქნა გადაჭრილი, თუმცა მათ ინფორმატიკაში ძალიან მნიშვნელოვანი როლი ითამაშეს.
პირველი ამოცანა (ანუ იგივე პრობლემა), რომლის ამოუხსნადობაც ალგორითმულ მეთოდებზე დაყრდნობით დამტკიცდა, ე.წ. შეჩერების ამოცანაა: ნებისმიერი ალგორითმისათვის (პროგრამისთვის) და მისი მონაცემისათვის დაადგინეთ, შეჩერდება თუ ჩაიციკლება ეს ალგორითმი მოცემულ მონაცემზე. ამის კონკრეტული მაგალითია სამი პროგრამა, რომელთაგან პირველი რიგ-რიგობით გადაამოწმებს ნატურალურ რიცხვებს და შეჩერდება მაშინ და მხოლოდ მაშინ, თუკი ისეთ ლუწ რიცხვს აღმოაჩენს, რომელიც ორი სხვა ლუწი რიცხვის ჯამს წარმოადგენს.

ალგორითმი 1.2: ლუწ რიცხვთა შემოწმება (პირველი ვერსია)

```
1: n = 0;
2: do
3: n++;
4: while(2n ≠ 2k + 2p)
```

მეორე ალგორითმი რიგ-რიგობით გადაამოწმებს ნატურალურ რიცხვებს და შეჩერდება მაშინ და მხოლოდ მაშინ, თუ ისეთ კენტ რიცხვს აღმოაჩენს, რომელიც ერთი ლუწი და ერთი კენტი რიცხვის ჯამს წარმოადგენს.

ალგორითმი 1.3: ლუწ რიცხვთა შემოწმება (მეორე ვერსია)

```
1: n = 0;
2: do
3: n++;
4: while(2n ≠ 2k + (2p − 1))
```

ცხადია, რომ პირველ ალგორითმზეც და მეორეზეც წინასწარ შეიძლება იმის თქმა, შეჩერდება თუ არა იგი.
მაგრამ განვიხილოთ ალგორითმი, რომელიც რიგ-რიგობით გადაამოწმებს ლუწ რიცხვებს და შეჩერდება მაშინ და მხოლოდ მაშინ, თუ განხილული რიცხვი მაქსიმუმ ორი მარტივი რიცხვის ჯამისაგან არ შედგება.

ალგორითმი 1.4: ლუწ რიცხვთა შემოწმება (მესამე ვერსია)

```
1: n = 1;
2: do
3: n++;
4: while(2n ≠ p1 + p2)        /* ეს ლუწი რიცხვი არ წარმოდგება ორი მარტივი რიცხვის ჯამის სახით */
```

ალგორითმი 1.5: შეჩერების დიაგონალური ალგორითმი D
მონაცემი: რაიმე ალგორითმი A

```
1: while(H(A(A))== „კი”)        /* შენიშვნა: აქ „A” ალგორითმის აღმწერი სიტყვაა (მაგ. მისი კოდი) */
```

ეს უკანასკნელი ამოცანა თანამედროვე მათემატიკის ერთ-ერთი უდიდესი ღია პრობლემაა (გოლდბახის ჰიპოთეზა) და, ცხადია, ჯერ-ჯერობით წინასწარ ვერ ვიტყვით, შეჩერდება თუ არა ზემოთ მოყვანილი ალგორითმი. ამრიგად, მოცემული ალგორითმის ანალიზით ზოგჯერ დაბეჯითებით შეიძლება იმის თქმა, შეჩერდება თუ არა იგი, ზოგჯერ კი -- არა.

განმარტება 1.26: მოცემულია რაიმე A ალგორითმი და მისი რაიმე X მონაცემი. საკითხს, შეჩერდება თუ არა A ალგორითმი X მონაცემზე ***შეჩერების ამოცანა*** ეწოდება.

კარგი იქნებოდა ისეთი ალგორითმის აღმოჩენა, რომელიც *ნებისმიერი* A ალგორითმისა და X მონაცემისთვის გვეტყოდა, ჩაიციკლება თუ არა $A(X)$. ამით მრავალი ღია პრობლემა გადაიჭრებოდა (მაგ. გოლდბახის ჰიპოთეზა). მაგრამ, სამწუხაროდ, ასეთი რამ შეუძლებელია, რაც შემდეგ თეორემაში მტკიცდება:

თეორემა 1.27 შეჩერების ამოცანა არაამოხსნადია
დამტკიცება: დავუშვათ საწინააღმდეგო: ვთქვათ, არსებობს ისეთი H ალგორითმი, რომელიც მონაცემად იღებს ნებისმიერი A ალგორითმისა და X მონაცემის აღმწერ სიტყვებს და გვეტყვის, შეჩერდება თუ არა $A(X)$. სხვა სიტყვებით რომ ვთქვათ,

$$H(A,X) = \begin{cases} \text{„კი”}, & A \text{ ალგორითმი შეჩერდება } X \text{ მონაცემზე;} \\ \text{„არა”}, & A \text{ ალგორითმი არ შეჩერდება } X \text{ მონაცემზე.} \end{cases}$$

ახლა კი შევქმნათ ისეთი ალგორითმი $D(X)$, რომელიც მონაცემად იღებს რაღაც X სიტყვას, რომელიც თავის თავად რაიმე ალგორითმის აღწერაა:

ალგორითმი 1.6: შეჩერების დიაგონალური ალგორითმი D
მონაცემი: რაიმე ალგორითმი A

```
1: while(H(A(A))== „კი”)        /* შენიშვნა: აქ „A” ალგორითმის აღმწერი სიტყვაა (მაგ. მისი კოდი) */
```

ესე იგი, D ალგორითმი მონაცემად მიიღებს რაიმე A ალგორითმის აღწერას და შეჩერდება მაშინ და მხოლოდ მაშინ, თუ A ალგორითმი მისსავე აღმწერი სიტყვის მონაცემზე არ შეჩერდება (აქ გასათვალისწინებელია ის ფაქტი, რომ თვით ალგორითმის აღწერაც რაღაც სიტყვაა, რომელიც მონაცემად შეიძლება ავიღოთ).
ახლა განვიხილოთ, თუ რა მოხდება, როდესაც D ალგორითმი თავის აღწერას მიიღებს მონაცემად. $D(D)$ ნიშნავს, რომ D ალგორითმი მისსავე აღმწერი სიტყვის მონაცემზე შეჩერდება მაშინ და მხოლოდ მაშინ, თუ D ალგორითმი მისსავე აღმწერი სიტყვის მონაცემზე არ შეჩერდება, ეს კი წინააღმდეგობაა!

რ.დ.გ.

ეს ფაქტი არ ნიშნავს აუცილებლად იმას, რომ რაღაცა ამოცანებისთვის ვერ ვიტყვით, ჩაიციკლება თუ არა მათი ალგორითმები. უსასრულოდ ბევრი ამოცანისათვის დაგვჭირდება ახალი მეთოდების ძიება, რომ ამ შეკითხვაზე პასუხი გავცეთ. შეჩერების ამოცანა ალგორითმულად ამოხსნადი რომ ყოფილიყო, მათემატიკური ლოგიკისა და ინფორმატიკის ერთ-ერთი უმნიშვნელოვანესი განხრა -- თეორემათა ავტომატური მტკიცება -- ადვილად გადაიჭრებოდა, ახლა კი საჭიროა თეორემათა კლასიფიკაცია და ყოველი ცალკეული კლასისათვის ავტომატური მტკიცებების მეთოდების ძიება. მსგავსი პრობლემები წამოიჭრება ასევე პროგრამული კოდის სისწორის ავტომატურ მტკიცებაშიც.

ძალიან მნიშვნელოვანია ასევე მეორე არაამოხსნადი ამოცანა, რომელიც „პოსტის შესაბამისობის პრობლემის” Post Correspondence Problem, მოკლედ *PCP* სახელითაა ცნობილი:

პოსტის შესაბამისობის პრობლემა (Post Correspondence Problem, PCP)
მოცემულია: $P = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n)), x_i, y_i \in \Sigma^+$ არაცარიელი სიტყვების წყვილთა მიმდევრობა, რომელსაც ჩვენ „ამოცანის პირობას", ან მოკლედ „პირობას" ვუწოდებთ.
$I = (i_1, ..., i_k), i_j \in \{1, 2, ..., n\}$ მიმდევრობას ეწოდება მოცემული პირობის ამონახსნი, თუ $x_{i_1} \circ x_{i_2} \circ \cdots \circ x_{i_n} = y_{i_1} \circ y_{i_2} \circ \cdots \circ y_{i_n}$.
თვალსაჩინოებისთვის P პრობლემა შეგვიძლია წარმოვიდგინოთ, როგორც დომინოს ქვები, რომელზედაც ერთ მხარეს x_i და მეორეზე კი y_i სიტყვა წერია, თან თითოეული ტიპის ქვა უსასრულოდ ბევრი შეიძლება იყოს (შესაძლებელია, რომ I ამოხსნაში $i_k = i_l$, თუ $k \neq l$.
მაშინ ამოცანა შემდეგში მდგომარეობს: შეიძლება თუ არა დომინოს ქვები ისე დავალაგოთ ერთმანეთის გვერდით, რომ ზედა ნაწილში წარმოქმნილი სიტყვა (სადაც მხოლოდ x_i ქვესიტყვები გვექნება) დაემთხვას ქვედა ნაწილში წარმოქმნილ სიტყვას?

მაგალითი 1.28: მოცემულია პრობლემა $P = ((1, 101), (10, 00), (011, 11))$, რაც გვაძლევს: $x_1 = 1, x_2 = 10, x_3 = 011$ და $y_1 = 101, y_2 = 00, y_3 = 11$. ამის ამონახსნი იქნება $I = (1, 3, 2, 3)$:
$x_1 \circ x_3 \circ x_2 \circ x_3 = 1 \circ 011 \circ 10 \circ 011 = 101 \circ 11 \circ 00 \circ 11 = 101110011 = 101 \circ 11 \circ 00 \circ 11 = y_1 \circ y_3 \circ y_2 \circ y_3$
თუ ამას წარმოვიდგენთ, როგორც დომინოს ქვების მიმდევრობას, მივიღებთ:

$$\boxed{\frac{1}{101}}\boxed{\frac{011}{11}}\boxed{\frac{10}{00}}\boxed{\frac{011}{11}}$$

ცხადია, რომ თუ ერთ ამოცანას რამოდენიმე ამოხსნა აქვს, მათი კომბინაციაც ამოხსნას გვაძლევს. ამიტომ შეიძლება წამოიჭრას შეკითხვა: თუ მოცემულია რაიმე ამოხსნა, შეიძლება თუ არა მისი სხვა ამოხსნების კომბინაციებად წარმოდგენა? წინა მაგალითში მოყვანილი ამოხსნა არ შეიძლება ასე დაიყოს. ანუ, როგორც ამბობენ, იგი **პრიმიტიულია**.
ზედა მაგალითის განხილვისას შეიძლება წარმოიშვას შთაბეჭდილება, რომ პოსტის პრობლემა სულაც არ არის რთული. ამიტომ განვიხილოთ შემდეგი
მაგალითი 1.29:
$P = ((001, 0), (01, 011), (01, 011), (10, 001))$; $x_1 = 001, x_2 = 01, x_3 = 01, x_4 = 10, y_1 = 0, y_2 = 011, y_3 = 101, y_4 = 001$.
მისი უმოკლესი ამოხსნაც კი 66 წყვილისაგან შედგება, რითაც ამ პრობლემის სირთულის პირველი შეფასება შეიძლება:

$$\begin{aligned} I = (&2, 4, 3, 4, 4, 2, 1, 2, 4, 3, 4, 3, 4, 4, 3, 4, 4, 2, 1, 4, 4, 2, 1, 3, 4, 1, 1, 3, 4, 4, 4, 2, 1, 2, 1, 1, 1, 3, 4, 3, 4, 1, 2, 1, 4, 4, 2, 1, 4, 1, \\ &1, 3, 4, 1, 1, 3, 1, 1, 3, 1, 2, 1, 4, 1, 1, 3) \end{aligned}$$

სრული გადარჩევის მეთოდით შეიძლება შეიქმნას ალგორითმი, რომელიც პოსტის ამოცანის I ამოხსნას მოგვცემდა, თუ ასეთი არსებობს. ამოხახსნის არარსებობის შემთხვევაში ასეთი ალგორითმი ჩაიციკლება.

სავარჯიშო 1.30: დაწერეთ პროგრამა, რომელიც სრული გადარჩევით მოგვცემს PCP ამოცანის პასუხს ასეთის არსებობის შემთხვევაში. დაამტკიცეთ მისი სისწორე და დაითვალეთ ბიჯების რაოდენობა იმ დაშვებით, რომ მოცემული პირობისთვის ამონახსნი არსებობს.

უხეშად, ამოცანები ორ კლასად შეიძლება დაიყოს: ისეთები, რომლისთვისაც ალგორითმული ამოხსნა არ არსებობს (ანუ ამ ამოცანების გადაჭრა სასრულ დროში არ შეიძლება - მათი ყველა ალგორითმი მინიმუმ ერთ მონაცემზე მაინც ჩაიციკლება) და ისეთებად, რომლებისთვისაც ალგორითმული ამოხსნა არსებობს. ისეთ ამოცანებში, რომლებისთვისაც ალგორითმები არსებობს, გამოყოფენ ამოცანათა სირთულის რამოდენიმე კლასს. აღმოჩნდა, რომ ერთ-ერთ კლასში ისეთი ამოცანებია, რომელთათვისაც რეალურ (ანუ პოლინომიურ) დროში ამოხსნა არსებობს, ხოლო მეორეში - ისეთები, რომელთათვისაც რეალურ დროში ამოხსნა დღეისათვის ცნობილი არ არის. აქვე უნდა აღინიშნოს, რომ „ეფექტური", „სწრაფი", „რეალურ დროში" ამოხსნადი ალგორითმები იგივეა, რაც „პოლინომურ დროში ამოხსნადი" (იხ. პირველი სემესტრის ლექციათა კურსი).

მნიშვნელოვანია შემდეგი გარემოება: ფარგლითა და სახაზავით კონკრეტული გეომეტრიული ფიგურების აგების ამოცანა გადაუჭრელია მხოლოდ გარკვეული მეთოდების გამოყენებით -- თუ ჩვენ საკითხს ისე დავსვამთ, შეიძლება თუ არა იგივე ფიგურების რაიმე მეთოდით აგება, მაშინ პასუხი დადებითი იქნება. დანარჩენი აქ განხილული არაამოხსნადი ამოცანები კი ზოგადად გადაუჭრელია -- არ არსებობს რაიმე მეთოდი, რომელიც ამ შეკითხვებზე პასუხს სასრულ დროში გაგვცემდა. ეს კი იმას ნიშნავს, რომ ამოცანის ჩამოყალიბებისას მნიშვნელოვან როლს თვით რესურსების შერჩევა თამაშობს.

მნიშვნელოვანი შენიშვნა: ჩვენ აქ შეჩერების ამოცანის არაამოხსნადობა გარკვეული აპარატის, კერძოდ ჩვენი ფსევდოკოდის მეშვეობით დავამტკიცეთ, რაც არაამოხსნადობის მტკიცების გარკვეულ ინტუიციას იძლევა. მაგრამ არავის უთქვამს, რომ არ არსებობს რაიმე სხვა ენა ან მეთოდი, რითაც ეს ამოცანა სასრულ დროში ამოიხსნებოდა. ლოგიკურია შეკითხვა, რა აპარატია ისეთი „უნივერსალური", რომლით დამტკიცებული თეორემაც ნებისმიერ სხვა გამოთვლის საშუალებასაც მოიცავს? ამაზე ცალსახა პასუხი არ არსებობს, თუმცა საყოველთაოდ მიღებულია ე.წ. ***ჩერჩის თეზისი,*** რომელიც ცნობილმა ამერიკელმა მეცნიერმა ალონსო ჩერჩმა ჩამოაყალიბა, რომლის მიხედვითაც თუ რაიმე ამოცანა არ ამოიხსნება ე.წ. ტიურინგის მანქანით (რომელსაც ხშირად ასევე ტიურინგ-პოსტის მანქანასაც უწოდებენ), მაშინ იგი ვერანაირი სხვა მეთოდით ვერ გადაიჭრება სასრულ დროში. აქვე უნდა აღინიშნოს, რომ ტიურინგის მანქანა მარტივი გამომთვლელის მათემატიკური მოდელია, რომელიც ხშირად გამოიყენება თეორიულ ინფორმატიკაში, როგორც ზოგადი გამომთვლელი, მაგრამ მისი ფორმალურად ჩამოყალიბება და მასზე დაყრდნობით თეორემათა მტკიცება ჩვენი კურსის ფარგლებს ცდება - ჩვენ ალგორითმების ზოგადი განმარტებითა და ფსევდოკოდით შემოვიფარგლებით.

1.4 რეკურსიული და რეკურსიულად გადათვლადი ამოცანები

როგორც უკვე აღვნიშნეთ, გადაწყვეტილების ამოცანების დახმარებით ოპტიმიზაციის ამოცანის გადაჭრა შეიძლება. მაგრამ როგორ შეგვიძლია თვით გადაწყვეტილების ამოცანის ჩამოყალიბება? ამ შეკითხვაზე პასუხის გასაცემად განვიხილოთ ზოგადი მიდგომა: გადაწყვეტილების ამოცანაში გვაინტერესებს, აკმაყოფილებს თუ არა რაიმე A მონაცემი B თვისებას. მაგალითად, A მონაცემად შეიძლება განვიხილოთ რაიმე გრაფი და B თვისებად – ჰამილტონის ციკლი (ამ შემთხვევაში მივიღეთ HC), ან კონიუნქციური ნორმალური ფორმით ჩაწერილი ფუნქცია და თვისება, რომ ეს ფუნქცია ცვლადების რაიმე მნიშვნელობაზე იღებს პასუხს 1 (ამოცანა SAT) და მრავალი სხვა.

სავარჯიშო 1.31: ჩამოაყალიბეთ IS_k, $Clique_k$, TSP_k და VC_k გადაწყვეტილების ამოცანების მონაცემები და შესაბამოწმებელი თვისებები.

სიმარტივისთვის ჩვენ აქ და შემდგომში ამოცანასა და მისი ამონახსნების სიმრავლეს (ამოცანის ენას) გავაიგივებთ და უბრალოდ „ამოცანას" ან „პრობლემას" ვუწოდებთ.

განმარტება 1.32: თუ მოცემული L_A ამოცანისთვის არსებობს რაიმე ალგორითმი, რომელიც მოცემული x სიტყვისთვის შეჩერდება იმ შემთხვევაში, თუ $x \in L_A$ (და „კი" პასუხსაც მოგვცემს), მაგრამ **შეიძლება** არ შეჩერდეს, თუ $x \notin L_A$, მაშინ ასეთ ენას (ამოცანას) ***რეკურსიულად გადათვლადი*** ეწოდება. თუ იგივე ამოცანისთვის არსებობს ისეთი ალგორითმი, რომელიც ნებისმიერი სიტყვისთვის შეჩერდება და $x \in L_A$ შეკითხვაზე სწორ პასუხსაც გაგვცემს, მაშინ ასეთ ენას (ამოცანას) ***რეკურსიული*** ეწოდება.

სავარჯიშო 1.33: რა დამოკიდებულებაა რეკურსიულ ამოცანათა და რეკურსიულად გადათვლადი ამოცანების სიმრავლეებს შორის? პასუხი დაასაბუთეთ.

აღსანიშნავია, რომ ტერმინი „რეკურსიული" და „რეკურსიულად გადათვლადი" 1930ან წლებში განვითარებული რეკურსიული ფუნქციების თეორიიდან მოდის: ნებისმიერი *გამოთვლადი* ფუნქცია შეიძლება აღიწეროს რამოდენიმე მარტივი ფუნქციის რეკურსიული ჯაჭვის აგებით. მაგრამ არსებობს ისეთი ფუნქციებიც, რომელთა გამოთვლაც ამ მეთოდით არ შეიძლება ან გარკვეულ შემთხვევაში არ შეიძლება (როგორც ეს წინა განმარტებაშია აღნიშნული).

რეკურსიულად გადათვლადი ამოცანის ერთ-ერთი ყველაზე ცნობილი მაგალითია ზემოთ მოყვანილი PCP. მისი არაამოხსნადობის დამტკიცება ცნობილი არაამოხსნადი ამოცანის, კერძოდ კი შეჩერების პრობლემის მასზე „დაყვანით" მტკიცდება (ანუ PCP ამოცანის მეშვეობით შეჩერების ამოცანის გადაჭრის შესაძლებლობით - დაახლოებით ისე, როგორც გადაწყვეტილების ამოცანით ოპტიმიზაციის პრობლემას ვხსნიდით). დაყვანის არსსა და ტექნიკას ჩვენ შემდგომში დაწვრილებით განვიხილავთ.

სავარჯიშო 1.34: დაამტკიცეთ, რომ შემდეგი ამოცანები რეკურსიულად გადათვლადია (ჩათვალეთ, რომ ალგორითმის მონაცემი ორობით ანბანზეა მოცემული):

- მოცემული A ალგორითმისთვის განსაზღვრეთ, შეჩერდება თუ არა იგი $\mathbb{B}^n, n \in \mathbb{N}$ ენის ნებისმიერ მონაცემზე;
- მოცემული A ალგორითმისთვის განსაზღვრეთ, შეჩერდება თუ არა იგი $\mathbb{B}^n, n \in \mathbb{N}$ ენის რაიმე მონაცემზე;

- ორი A და B ალგორითმისთვის განსაზღვრეთ, არსებობს თუ არა $w \in \mathbb{B}^*$ მონაცემი, რომლისთვისაც $A(w) \neq B(w)$.

სავარჯიშო 1.35: შემდეგი გამონათქვამებიდან რომელია ჭეშმარიტი და რომელი - მცდარი? პასუხი დაასაბუთეთ

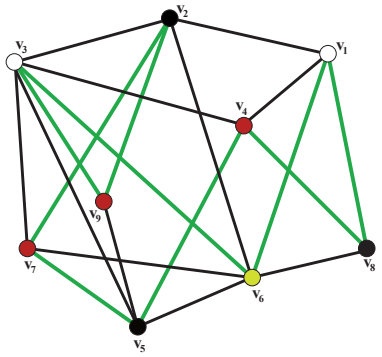
- თუ ამოცანა რეკურსიულად გადათვლადია, იგი ასევე რეკურსიულიცაა;
- თუ ამოცანა რეკურსიულია, იგი ასევე რეკურსიულად გადათვლადიცაა;
- K რეკურსიულად გადათვლადი ამოცანის ალგორითმი შეიძლება არ შეჩერდეს რაიმე $x \in L_K$ მონაცემზე;
- K რეკურსიულად გადათვლადი ამოცანის ალგორითმი შეიძლება არ შეჩერდეს რაიმე $x \notin L_K$ მონაცემზე;
- K რეკურსიულად გადათვლადი ამოცანის ალგორითმი შეიძლება შეჩერდეს რაიმე $x \notin L_K$ მონაცემზე;
- K რეკურსიული ამოცანის ალგორითმი შეიძლება შეჩერდეს რაიმე $x \notin L_K$ მონაცემზე;
- K რეკურსიული ამოცანის ალგორითმი შეიძლება არ შეჩერდეს რაიმე $x \notin L_K$ მონაცემზე.

თავი 2

ამოცანათა სერტიფიცირება და მასზე დაფუძნებული სირთულის NP და co-NP კლასი

2.1 სერტიფიცირება და სერტიფიკატი

განვიხილოთ შემდეგ ნახაზში მოცემული გრაფი. არსებობს თუ არა მასში ჰამილტონის ციკლი?



გრაფი ჰამილტონის ციკლისა და ოთხ ფრად შეღებვის სერტიფიკატით
ამ შეკითხვაზე პასუხის გაცემა ზოგადად რთულია. მაგრამ შესაძლებელია ვინმემ ჰამილტონის ციკლის არსებობა დაამტკიცოს შესაბამისი გზის წარმოდგენით: $(v_3, v_9, v_2, v_7, v_5, v_4, v_8, v_1, v_6, v_3)$.
ცხადია, რომ შესაძლებელია იმის გადამოწმება, ეს შემოთავაზებული გზა მართლაც ჰამილტონის ციკლია, თუ არა.
თუ იგივე გრაფზე დავსვამთ ოთხ ფრად შეღებვის ამოცანას, მაშინ შესაძლებელი იქნება ამ ამოცანის დადებითი პასუხის დამტკიცება კონკრეტული შეღებვის მოცემით (ნახ. 2.1).

განმარტება 2.1: თუ რაიმე ამოცანისთვის შეიძლება შესაძლო ამონახსნის შემოთავაზება და ამ შემოთავაზებული სავარაუდო ამონახსნის გადამოწმების ალგორითმიც არსებობს, ამ შემოთავაზებულ ამონახსნს ***„სერტიფიკატი"***, ხოლო გადამოწმების ალგორითმს კი სერტიფიცირების პროცესი (ან, მოკლედ, ***სერტიფიცირება***) ეწოდება.

A ამოცანის სერტიფიცირების ალგორითმს აღვნიშნავთ როგორც V_A. ცხადია, რომ მისი მონაცემები იქნება თვით ამ A ამოცანის რაღაც ობიექტი X და შემოთავაზებული პასუხი y. მაგალითად, $V_{HC}(X, y)$, სადაც X რაღაც კონკრეტული გრაფია და y ამ გრაფის გზის სიტყვა, რომელიც მის ჰამილტონის ციკლს უნდა აღწერდეს.

სავარჯიშო 2.2: დაწერეთ HC, CN_k, VC_k და IS_k ამოცანების სერტიფიცირების ალგორითმები. რა იქნება ამ ალგორითმების მონაცემი და რა – პასუხი?

სავარჯიშო 2.3: რა იქნება V_{GI} და V_{SGI} მონაცემები?

2.2 NP კლასი, როგორც სწრაფად სერტიფიცირებად ამოცანათა სიმრავლე

ყველა ამოხსნადი ამოცანა, რომელიც აქამდე გვქონდა განხილული, პოლინომურ დროში, ანუ „სწრაფად" სერტიფიცირებადი იყო. ასეთ პრობლემებს ცალკე კლასში (სიმრავლეში) აერთიანებენ, რომელსაც NP ეწოდება:

$$\text{NP} = \{A \mid A \text{ ამოცანა პოლინომურ დროში სერტიფიცირებადია}\}$$

ამოცანების (და, საერთოდ, გამოსაკვლევი ობიექტების) ერთ კლასში გაერთიანება იმითაა მოსახერხებელი, რომ მათი შემდგომი შესწავლა და აღწერა უფრო მოსახერხებელი იყოს. მეორე მნიშვნელოვანი კლასია P, რომელიც პოლინომურ დროში ამოხსნადი ამოცანებისგან შედგება:

$$\text{P} = \{A \mid A \text{ ამოცანა პოლინომურ დროში ამოხსნადია}\}$$

ცხადია, რომ შესაძლებელია, რაიმე ამოცანა რომელიმე კლასში შედიოდეს (ანუ იყოს ამ კლასისთვის დამახასიათებელი თვისების მატარებელი), მაგრამ ეს არ იყოს ცნობილი. ასე, მაგალითად, რიცხვის სიმარტივის ტესტის ამოცანისთვის (რომელსაც ასევე *Prime* ეწოდება: მოცემული რიცხვისთვის განსაზღვრეთ, მარტივია იგი თუ არა) არ იყო ცნობილი სწრაფი (პოლინომური) ალგორითმი და მას სხვადასხვა ხერხით (მაგალითად შემთხვევითი ალგორითმებით) ხსნიდნენ, მაგრამ XXI საუკუნის დასაწყისში ინდოელმა მეცნიერებმა პოლინომური ალგორითმი გამოიგონეს, რითიც $Prime \in \text{NP}$ დამტკიცდა.
ასევე არ არის ცნობილი, ამოიხსნება თუ არა VC_k ამოცანა პოლინომურ დროში, ამიტომაც მას ვერ მივაკუთვნებთ P კლასს, თუმცა არაა გამორიცხული, რომ ეს ასე იყოს.
ცხადია, რომ თუ ამოცანა ამოიხსნება პოლინომურ დროში, იგი სწრაფად სერტიფიცირებადიც იქნება, ანუ $\text{P} \subset \text{NP}$.

სავარჯიშო 2.4: დაამტკიცეთ, რომ $HC, CN_k, VC_k, IS_k, GI, SGI \in \text{NP}$

სავარჯიშო 2.5: P=NP დაშვებით, რა თვისება ექნება ნებისმიერ სწრაფად სერტიფიცირებად ამოცანას?

სავარჯიშო 2.6: დაამტკიცეთ $\text{P} \subset \text{NP}$.

ამ სავარჯიშოთი მივუახლოვდით თანამედროვე მეცნიერების ერთ-ერთ უდიდეს დია საკითხს: რადგან $\text{P} \subset \text{NP}$, არ არის გამორიცხული, რომ $\text{P} = \text{NP}$, მაგრამ ასევე შესაძლებელია, რომ $\text{P} \neq \text{NP}$ (თეორიულად ისიცაა შესაძლებელი, რომ ორივე შემთხვევა ჭეშმარიტი იყოს, ანუ ორი ერთმანეთისგან დამოუკიდებელი თეორია არსებობდეს, ერთში პირველი შემთხვევა იყოს ჭეშმარიტი, მეორეში კი მისი უარყოფა, მაგრამ ეს საკითხი ჩვენი კურსის ფარგლებს ცდება).
რომელია აქედან ჭეშმარიტი, ეს ჯერ-ჯერობით უცნობია, თუმცა ამ საკითხის გადაჭრაზე ძალიან ბევრი თეორიული და პრაქტიკული შედეგია დამოკიდებული.

2.3 NP და co-NP კლასები

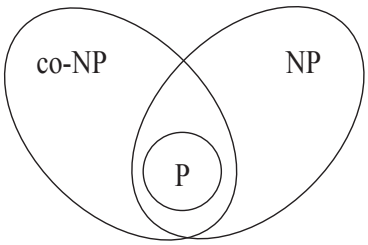
განვიხილოთ ნებისმიერი გადაწყვეტილების ამოცანა A და შესაბამისი ენა L_A. ამ ამოცანის პასუხი უნდა იყოს „კი", თუ $x \in L_A$ და „არა", თუ $x \notin L_A$.
ახლა „შევაბრუნოთ" A ამოცანა და შევქმნათ ისეთი $\bar{A}$ ამოცანა, რომლისთვისაც $L_{\bar{A}} = \bar{L}_A$. ანუ ამ ამოცანის პასუხი უნდა იყოს A ამოცანის პასუხის შებრუნებული: „კი", თუ მონაცემი $x \notin A$ და „არა", თუ $x \in A$.
თუ $A \in \text{NP}$, რა შეიძლება ითქვას $\bar{A}$ ამოცანაზე? პირველი არის ის, რომ სულაც არ არის აუცილებელი, რომ $x \notin A$ პოლინომურ დროში იყოს გადამოწმებადი. უფრო მეტიც: რადგან (როგორც წესი) $\{x \notin A\}$ სიმრავლე გაცილებით ფართოა, ვიდრე A, უნდა ვივარაუდოთ, რომ ნებისმიერი ასეთი მონაცემის გადამოწმება პოლინომურ დროში ვერ უნდა მოხერხდეს.
განმარტება 2.7:

$$\text{co-NP} = \{\bar{A} \mid A \in \text{NP}\}.$$

ცხადია, რომ უნდა არსებობდეს ისეთი ამოცანებიც, რომლებიც შედიან როგორც NP, ასევე co-NP კლასში. აქედან ვიღებთ შემდეგ სქემას, რომელიც სავარჯიშოს სახითაა მოყვანილი:
NP, co-NP და P სიმრავლეების ***შესაძლო*** დამოკიდებულება

სავარჯიშო 2.8: (ლემა 2.9:) დაამტკიცეთ, რომ

$$\text{P} \in \text{co-NP} \cap \text{NP}$$



სავარჯიშო 2.10: ანალოგიურად ჩამოაყალიბეთ, რა უნდა იყოს co-P.

ამ სავარჯიშოებიდან ცხადი ხდება, რომ ***იმ შემთხვევაში***, თუ P=NP, co-NP=NP.

შენიშვნა: ამ გამონათქვამის შებრუნებული (ანუ ის ფაქტი, რომ თუ P≠NP, მაშინ co-NP≠NP) დამტკიცებული არ არის.

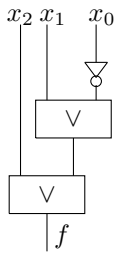
ანალოგიური საკითხები – რა დამოკიდებულებაა ამოცანათა სიმრავლეებს შორის – ძალიან მნიშვნელოვანია ამოცანათა შესწავლისა და მათთვის სწრაფი ალგორითმების შექმნისთვის და დაწვრილებით შემდგომში იქნება განხილული.

2.4 ლაკონური სქემები და ამოცანები NP კლასის მიღმა

ბუნებრივია შეკითხვა: რა ტიპის ამოცანა *არ* ეკუთვნის NP კლასს? ცხადია, რომ ყველა არაამოხსნადი (მათ შორის რეკურსიულად გადათვლადი) ამოცანა ასეთი უნდა იყოს. მაგრამ არსებობს თუ არა ისეთი *ამოხსნადი* ამოცანა, რომელიც არ იქნება სწრაფად სერტიფიცირებადი?
აღმოჩნდა, რომ ასეთი არაერთი ამოცანა არსებობს და მათი გაგებისათვის ჯერ ერთი მნიშვნელოვანი კონცეფციის გააზრებაა საჭირო.

განმარტება 2.11: ნებისმიერი n ბიტიანი $w \in \mathbb{B}^n$ სიტყვის ***ლაკონური სქემა*** (succinct circuit representation, SCR) ეწოდება ისეთ S სქემას, რომლის შესაბამისი f ფუნქციის ცხრილის შედეგი არის w, ანუ $f(k_2) = w(k)$, სადაც k_2 წარმოადგენს k რიცხვს ორობით კოდში.

მაგალითი 2.12: $w = 11111101$ სიტყვის ლაკონური სქემა სამი ელემენტისაგან შედგება, რადგან იგი $f(x_0, x_1, x_2) = \bar{x}_0 \vee x_1 \vee x_2$ ფუნქციის ცხრილის შედეგია (ნახ. 2.4).
შენიშვნა: უნდა აღინიშნოს, რომ, რადგან სიტყვებში ცვლადების გადანომვრა ხდება მარჯვნიდან მარცხნივ, ხოლო ცხრილში კი პირიქით, ცხრილში მოყვანილი ფუნქციის შედეგი „შებრუნებულია".



x_0	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1
f	1	0	1	1	1	1	1	1

$w = 11111101$ სიტყვის ლაკონური სქემა და შესაბამისი ფუნქციის ცხრილი
რადგან ნებისმიერი m ცვლადიანი ფუნქციის ცხრილის შედეგის სიგრძეა 2^m, n სიგრძის w სიტყვის წარმოდგენა უნდა შეიძლებოდეს $\log n$ ცვლადიანი ფუნქციით, რაც იმას ნიშნავს, რომ ***ზოგ შემთხვევაში*** თავდაპირველ მონაცემთან შედარებით ექსპონენციური მოგება შეიძლება გვქონდეს (აქვე უნდა აღინიშნოს, რომ m ცვლადიანი ფუნქციის ზომაც ექსპონენციური შეიძლება იყოს და არანაირი მოგება აღარ გვქონდეს).

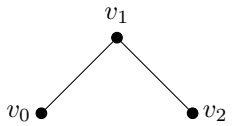
ერთის მხრივ, ასეთი *ლაკონური* წარმოდგენა ძალიან მომგებიანია, რადგან ობიექტები ექსპონენციურად ნაკლები რესურსებით შეგვიძლია აღვწეროთ, მაგრამ, მეორეს მხრივ, შედეგის გადამოწმებაც ექსპონენციურად დიდ დროს

შეიძლება მოითხოვდეს: თუ მონაცემის სიგრძე k ექსპონენციურად ნაკლებია შედეგების რაოდენობაზე, მაშინ ცხადია, რომ გადამოწმებაც შესაბამისად მეტ დროს შეიძლება მოითხოვდეს. აქ უკვე „გადამოწმების" სირთულის პირველი იდეა გამოიკვეთა.

ახლა ვნახოთ, თუ როგორ შეიძლება წარმოვადგინოთ გრაფები ლაკონური სქემების საშუალებით.

განმარტება 2.13: $G = (V, E)$ გრაფის ლაკონური სქემა ეწოდება $f : \mathbb{B}^{2n} \to \mathbb{B}$ ფუნქციის რეალიზაციას, სადაც $f(x_0, ..., x_n, y_0, ..., y_n) = 1 \Leftrightarrow (v_k, v_l) \in E$, თუ $(x_0, ..., x_n)$ და $(y_0, ..., y_n)$ შესაბამისად k და l რიცხვების ორობითი წარმოდგენაა.
სხვა სიტყვებით რომ ვთქვათ, თუ G გრაფის წვეროებს გადავნომრავთ და მათ ნომრებს ორობით კოდში წარმოვადგენთ, იმის შესამოწმებლად, არის თუ არა k და l წვეროს შორის წიბო, უნდა შევამოწმოთ, მათი ორობითი კოდების კონკატენაცია თუ გვაძლევს f ფუნქციაში შედეგს 1.

მაგალითი 2.14: განვიხილოთ 2.4 ნახაზში მოცემული გრაფი, რომლის შესაბამისი ფუნქციის ცხრილიც იქნება მოყვანილი. x ცვლადის ნაცვლად ნებისმიერი მნიშვნელობის ჩასმა შეიძლება (სიმარტივისთვის ავიღოთ 0).



x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
y_0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
y_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
f	0	1	0	x	1	0	1	x	0	1	0	x	x	x	x	x

გრაფი შესაბამისი ფუნქციის ცხრილით
სავარჯიშო 2.15: დახაზეთ ზემოთ მოყვანილი გრაფის ლაკონური სქემა.
თუ გრაფს შესაბამისი ლაკონური სქემით აღვწერთ, რამდენად ნაკლები სიგრძის მონაცემს მივიღებთ მატრიცული სახით აღწერასთან შედარებით?

სავარჯიშო 2.16: დაწერეთ პროგრამა, რომელიც G გრაფის ლაკონური სქემის ფუნქციას მიიღებს მონაცემად და წიბოების რაოდენობას დაითვლის. ჩათვალეთ, რომ ფუნქცია მოცემულია დნფ სახით. როგორ კოდირებას აირჩევდით, რომ მონაცემის აღწერილობა რაც შეიძლება პატარა იყოს?
დაითვალეთ ამ ალგორითმის ბიჯების ზედა ზღვარი.

როგორც ნაჩვენებია ნაშრომში H. Galperin, and A. Wigderson (1983), "Succinct representations of graphs", Information and Control, 56:3, pp. 183 - 198, ამ სახით ჩაწერილი გრაფისთვის ჰამილტონის ციკლის ამოცანა HC არ არის გადამოწმებადი პოლინომურ დროში. მას „ლაკონური ჰამილტონის ციკლის" ამოცანა Succinct Hamiltonian Cycle, მოკლედ SHC ეწოდება.
ასევე მტკიცდება, რომ ამდაგვარად დასმული HC, CN_k, VC_k, IS_k და მრავალი სხვა ამოცანაც არ ეკუთვნის NP კლასს.

თავი 3

NP-სრული და NP-რთული ამოცანები

3.1 ამოცანების ერთმანეთზე დაყვანის პრინციპი

დავუშვათ, მოცემული გვაქვს რაიმე A ამოცანა, რომლის ალგორითმიც ჩვენთვის ცნობილი არ არის. თუ არსებობს ისეთი B ამოცანა ჩვენთვის ცნობილი ალგორითმით და ისეთი ალგორითმი R_{AB}, რომლითაც A ამოცანის ნებისმიერ X მონაცემს ისე გარდავქმნით B ამოცანის ისეთ X' მონაცემად, რომ ამ უკანასკნელმა პირველი ამოცანის ამოხსნა მოგვცეს (ანუ $A(X) = B(X')$), მაშინ ამბობენ, რომ A ამოცანა B ამოცანაზე დაიყვანება (იხ. შემდეგი დიაგრამა).

$$\begin{array}{ccc} A(X) & \stackrel{?}{=} & Y \\ {\scriptstyle R_{AB}}\big\downarrow & & \\ B(X') & = & Y \end{array}$$

ცხადია, რომ R_{AB} ალგორითმიც (რომელსაც ასევე A ამოცანის B ამოცანაზე ***დაყვანის*** პროცესსაც უწოდებენ) სწრაფი უნდა იყოს, რომ ასეთ ოპერაციას რაიმე აზრი ქონდეს.
ასეთ შემთხვევაში ამბობენ, რომ B ამოცანა „არანაკლები სირთულისაა", ვიდრე A და ეს ცხადიცაა: B ამოცანის ამოხსნა უფრო „მარტივი" რომ ყოფილიყო, მაშინ საწყის ამოცანას სწრაფად („მარტივად") დავიყვანდით მასზე, ამოვხსნიდით და საერთო ჯამში A ამოცანის ამოხსნაც მარტივი იქნებოდა.
ამ შემთხვევაში წერენ: $A \leq_P B$ და ამბობენ, რომ A ამოცანა B ამოცანაზე დაიყვანება (პოლინომურ დროში).

შენიშვნა: აქ ნახმარი ტერმინები „მარტივი" და „რთული" მკაცრად შემდგომში იქნება განმარტებული, ახლა კი ინტუიციის გასანვითარებლად უფრო ზოგადი აღწერით შემოვიფარგლებით.

ამ იდეების უკეთ გასაგებად განვიხილოთ რამოდენიმე მაგალითი.

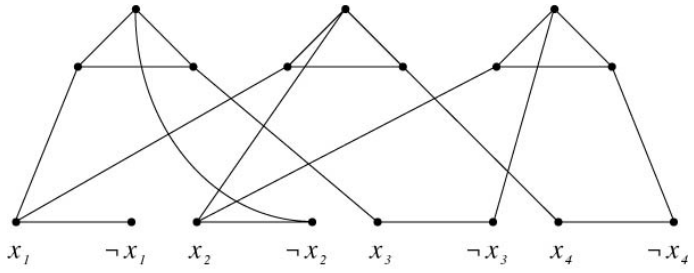
- IS_k და $Clique_k$: თუ მოცემულია იზოლირებული სიმრავლის ამოცანა, მისი გრაფის გარდაქმნა შეიძლება შემდეგნაირად: კვანძების სიმრავლე იგივე რჩება. თუ საწყის გრაფში ორი წვერო წიბოთი შეერთებული არაა, გარდაქმნილ გრაფში ეს ორი წვერო უნდა შეერთდეს და პირიქით: თუ საწყის გრაფში ორი წვერო წიბოთია შეერთებული, მაშინ გარდაქმნილ გრაფში მათ შორის წიბო არ იარსებებს. ადვილი საჩვენებელია, რომ ასეთ გარდაქმნილ გრაფში სრული სიმრავლის ამოცანა პასუხს გაგვცემს იზოლირებული სიმრავლის ამოცანაზე, ანუ პირველი ამოცანის მეორეზე დაყვანა შეიძლება.

- VC_k და $3SAT$: (ეს ბოლო ამოცანა იგივეა, რაც SAT, მხოლოდ ყოველ ფრჩხილში ზუსტად სამი ცვლადია წარმოდგენილი)

  დასაწყისისთვის განვიხილოთ კერძო მაგალითი: მოცემულია ფუნქცია, რომელიც ჩაწერილია კონიუნქციური ნორმალური ფორმით და შედგება 4 ცვლადისა და 3 დიზიუნქციური ჩანაწერისაგან (ანუ სამი ფრჩხილისაგან, რომელშიც ასევე სამი ცვლადი შედის):

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee \neg x_2 \vee x_3)(x_1 \vee x2 \vee x4)(x_2 \vee \neg x_3 \vee \neg x_4).$$

ყოველ ცვლადსა და მის უარყოფას შევუსაბამოთ ერთმანეთთან წიბოთი შეერთებული თითო კვანძი გრაფში, ხოლო ყოველ დიზიუნქციურ ჩანაწერს შევუსაბამოთ სამი ერთმანეთთან წიბოთი შეერთებული კვანძი (იხ. ნახ. 3.1). თუ დიზიუნქციურ ჩანაწერში (ფრჩხილში) შედის რომელიღაცა ცვლადი x_i ან $\neg x_i$, ამ ცვლადის (ან მისი უარყოფის) შესაბამისი კვანძი ჩანაწერის ერთ-ერთ კვანძს უერთდება წიბოთი.



რედუქციის სქემა

ადვილი საჩვენებელია, რომ k ცვლადიანი და c დიზიუნქციურ ჩანაწერიანი მოცემული ფუნქციის ასეთი სახით გარდაქმნის შედეგად მიღებული გრაფი გადაიფარება $k + 2c$ კვანძით მაშინ და მხოლოდ მაშინ, თუ ეს ფუნქცია შეიძლება ჭეშმარიტი გახდეს ცვლადების გარკვეული მნიშვნელობებისათვის (თუ არსებობს ცვლადების ასეთი მნიშვნელობები, მაშინ უნდა შეირჩეს მათი შესაბამისი კვანძები).

ზოგადად, ყოველი ცვლადისთვის შევქმნით ორ კვანძს, რომელიც წიბოთია ერთმანეთთან დაკავშირებული; ასევე ყოველი ფრჩხილისთვის შევქმნით სამ კვანძს და მათაც შევაერთებთ (მივიღებთ იმდენ სამკუთხედს, რამდენი ფრჩხილიც გვაქვს, თანაც თითო კვანძს ფრჩხილის თითო ცვლადი შეესაბამება). ბოლოს, თუ ფრჩხილში გვხვდება რაიმე ცვლადი ან მისი უარყოფა, შესაბამისი ცვლადის ან მისი უარყოფის შესაბამისი წვერო ამ ფრჩხილის შესაბამისი სამკუთხედის რომელიმე წვეროს უკავშირდება. ასე შექმნილი გრაფი უნდა გადაეცეს გადაფარვის VC_m ამოცანას. სადაც $m = k + 2c$. აქ k ცვლადების, c კი ფრჩხილების რაოდენობაა.

საგარჯიშო 3.1: აჩვენეთ, რომ ზემოთ მოყვანილი გრაფი გადაიფარება $k + 2c$ წვეროთი მაშინ და მხოლოდ მაშინ, თუ შესაბამისი ფუნქცია არაა ნულოვანი.

საგარჯიშო 3.2: დაწერეთ პროგრამა, რომელიც $3SAT$ შესაბამისად მოცემული დიზიუნქციური ფორმიდან ზემოთ აღნიშნულ გრაფს გამოითვლის (და შეძლებისამებრ დახაზავს).

- CN_k და IS_m:
  საგარჯიშო 3.3: აჩვენეთ, რომ $CN_k \leq_P IS_m$

თუ $A \leq_P B$ და $B \leq_P A$, მაშინ ამბობენ, რომ ორივე ამოცანა ერთი და იმავე სირთულისაა და წერენ $A =_P B$.

საგარჯიშო 3.4: აჩვენეთ, რომ $VC_k = CN_p$ და $VC_k = IS_p$ (გაითვალისწინეთ, რომ შესაძლებელია $k \neq p$).

3.2 NP რთული და NP სრული ამოცანები

განმარტება 3.5: თუ ამოცანათა რაიმე კლასიდან (სიმრავლიდან) *ყველა* ამოცანა რაიმე S ამოცანაზე დაიყვანება, მას ამ კლასის მიმართ ***რთული*** ეწოდება. ბუნებრივია, რადგან ჩვენი განხილვის მთავარი ობიექტი NP კლასია, ამიტომაც პირველ რიგში NP-რთულ ამოცანებს განვიხილავთ.
NPრთული ამოცანა ისეთიც შეიძლება იყოს, რომელიც თვითონ ეკუთვნის ამ კლასს. ამ შემთხვევაში მას NP-***სრული*** ეწოდება.

ბუნებრივია შეკითხვა: არსებობს კი ისეთი ამოცანა, რომელზედაც NPკლასის *ნებისმიერი* ამოცანა დაიყვანება? ასეთი ამოცანების არსებობა ძალიან საინტერესო შედეგების გამომწვევი იქნებოდა, რადგან მხოლოდ NP-სრული S ამოცანის ამოხსნით ნებისმიერი სხვა $A \in$ NP ამოცანის ამოხსნის გზას ვიპოვნიდით: A ამოცანას დავიყვანდით S ამოცანაზე და მისი ამოხსნით საწყისი ამოცანის პასუხსაც მივიღებდით (რა თქმა უნდა, თვითონ დაყვანის ალგორითმის პოვნაც საჭიროა, მაგრამ ეს უფრო ადვილია, მით უმეტეს, რომ ასეთის არსებობა გარანტირებულია).

აღმოჩნდა, რომ NP-სრული და, შესაბამისად, NP-რთული ამოცანაც ძალიან ბევრი არსებობს.

იმის საჩვენებლად, რომ რაიმე A ამოცანა NP-სრულია, საკმარისია იმის ჩვენება, რომ რაიმე უკვე ცნობილი NPსრული ამოცანა S მასზე დაიყვანება: თუ ვიცით, რომ ნებისმიერი $X \in$ NP ამოცანისთვის $X \leq_P S$ და $S \leq_P A$, მაშინ $X \leq_p A$.

სავარჯიშო 3.6: დაამტკიცეთ $A \leq_P B \leq_P C \Rightarrow A \leq_P C$ (მინიშნება: გაიაზრეთ, რას ნიშნავს $X \leq_P Y$ და შემდეგ ერთმანეთის მიყოლებით ანალოგიური ალგორითმების ჩატარება რამდენად სწრაფ ალგორითმს მოგვცემს).
შენიშვნა: ამ სავარჯიშოს ერთი საინტერესო „მახე" აქვს, რომელიც აუცილებლად უნდა გაითვალისწინოთ: თუ ვინმე ეცდება A ამოცანიდან B ამოცანაზე დაყვანის R_{AB} ალგორითმისა და, შესაბამისად, R_{BC} დაყვანის ალგორითმების ჩვეულებრივი „კომპოზიციით" A ამოცანიდან C ამოცანაზე დაყვანის R_{AC} ალგორითმის აგებას, შეიძლება შემდეგ პრობლემას წააწყდეს: $x \in A$ მონაცემის გარდაქმნისას მივიღოთ გაცილებით უფრო გრძელი მონაცემი, ვიდრე $|x|$ (ანუ A ამოცანიდან C ამოცანაზე დაყვანის ალგორითმის მონაცემის სიგრძე), რაც ცალკე R_{BC} ალგორითმის მუშაობის დროზე არ იმოქმედებდა (რადგან დროის ზედა ზღვარი ამ გრძელი მონაცემის სიგრძესთან შედარებით დიდი არ იქნებოდა), მაგრამ საწყისი x მონაცემის სიგრძესთან შედარებით კომპოზიცია ცუდ შედეგს მოიტანს.

როგორც ზემოთ ნათქვამიდან და სავარჯიშოებიდენ ჩანს, შედარებით ადვილი უნდა იყოს ერთი ამოცანის მეორეზე დაყვანის პროცესის პოვნა (თუმცა ჩვენ აქ შედარებით მარტივ მაგალითებს განვიხილავდით: ეს პროცესი უფრო რთულიც შეიძლება იყოს, მაგრამ ძალიან რთული არაა). გაცილებით უფრო რთულია *პირველი* NP-სრული ამოცანის პოვნა, როდესაც, არაფორმალურად რომ ვთქვათ, პირველი ხელმოსაჭიდი არ გვაქვს და უნდა დამტკიცდეს, რომ ყველა ამოცანა მასზე შეიძლება დავიყვანოთ.
NPრთული ამოცანების თეორია 1970ანი წლების დასაწყისში განვითარდა ამერიკელი მეცნიერის სტივენ კუკის (Stephen Cook) და, მისგან დამოუკიდებლად, მოსკოველი მეცნიერის ლეონიდ ლევინის შრომებში – ორივემ ასეთი სრული პრობლემის არსებობა დაამტკიცა, თუმცა სხვადასხვა გზით. სამეცნიერო ლიტერატურაში ეს შედეგი კუკ-ლევინის თეორემის სახელითაა ცნობილი.
კუკის თეორემა გვეუბნება, რომ SAT ამოცანა NP-სრულია. ამ შედეგის პრაქტიკული მნიშვნელობა გამოჩნდა ამერიკელი მეცნიერის რიჩარდ კარპის (Richard Karp) 1972 წელს გამოქვეყნებულ ნაშრომში, სადაც მან 21 ამოცანის NPსრულობა დაამტკიცა: იქიდან გამომდინარე, რომ SAT NPსრულია, მან ეს ამოცანა დაიყვანა $3SAT$ პრობლემაზე, შემდეგ ეს უკანასკნელი VC ამოცანაზე, შემდეგ ეს IS-ზე და ა.შ. ამდაგვარი ჯაჭვით ყველა ამ ამოცანის NPსრულობა დამტკიცდა. დღეისათვის ამ კლასის ათასობით ამოცანა არსებობს, რომელთა თეორიულ და პრაქტიკულ მნიშვნელობას ჩვენ შემდგომში განვიხილავთ.

მართალია, პირველი კონკრეტული NPსრული ამოცანა არის SAT, სიმარტივისთვის ჩვენ ე.წ. $CirquitSAT$ (ან, როგორც ზოგიერთი ავტორი მოკლედ აღნიშნავს, $CSAT$) ამოცანის NPსრულობას დავამტკიცებთ და აქედან გამომდინარე ავაგებთ ჯაჭვს $CSAT \leq_P SAT$ ჩვენებით.
თვით $CSAT$ არის იგივე SAT ამოცანის ანალოგი, რომელიც სქემებზეა გადატანილი: მოცემულია $f : \mathbb{B}^n \to \mathbb{B}$ ფუნქციის სქემა C. არსებობს თუ არა მისი ისეთი მონაცემი, რომ C იძლეოდეს პასუხს 1 ?

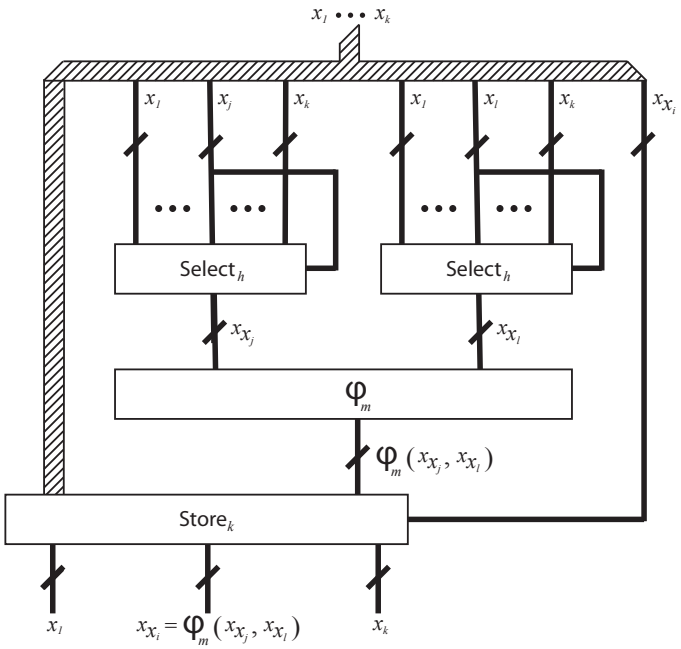
თეორემა 3.7 $CSAT$ ამოცანა NP-სრულია

დამტკიცება: დავუშვათ, მოცემულია რაიმე $A \in$ NP ამოცანა. მისთვის უნდა არსებობდეს სერტიფიცირების ალგორითმი $V_A(X, y)$, რომელიც A ამოცანის X ობიექტისთვის y შემოთავაზებულ პასუხს პოლინომურ დროში გადაამოწმებს (მაგალითად, A შეიძლება იყოს ჰამილტონის ციკლის ამოცანა, X რაიმე კონკრეტული გრაფი და y ამ გრაფის რაიმე გზა). ზოგადობის შეუზღუდავად დავუშვათ, რომ X და y მონაცემები ორობით ანბანშია ჩაწერილი.
თუ ვაჩვენებთ, რომ პოლინომურ დროში შესაძლებელია $V_A(X, y)$ ალგორითმის შესაბამისი სქემის აგება (ანუ ისეთი CV_A სქემის, რომელიც X და y მონაცემების შესაბამის შემავალ სიგნალებზე იგივე პასუხს მოგვცემს, როგორც $V_A(X, y)$, ყველაფერი დამტკიცებული იქნება: თუ მოცემული გვექნება A ამოცანის რაიმე X' ობიექტი და უნდა დავადგინოთ, გვაძლევს თუ არა $A(X')$ პასუხს „კი", შესაძლებელი იქნება CV_A სქემაში X' შესაბამისი სიგნალების დაფიქსირება, შემდეგ ამის გათვალისწინებით ამ სქემის გამარტივება (რაც წინა სემესტრში გვქონდა), რითაც მივიღებთ სქემას SCV_A, სადაც CV_A სქემისგან განსხვავებით გვექნება მხოლოდ y ელემენტების შესაბამისი მონაცემების შემომავალი სიგნალები. ასე რომ, თუ $CSAT(SCV_A)$ ამოცანა გვეტყვის პასუხს „კი", მაშინ უნდა არსებობდეს ისეთი y მონაცემი, რომლისთვისაც $V_A(X', y)$=„კი", რაც იმას ნიშნავს, რომ მოცემული $A(X')$=„კი". და თუ $CSAT(SCV_A)$ ამოცანა გვეტყვის პასუხს „არა", მაშინ ***არ*** უნდა არსებობდეს ისეთი y მონაცემი, რომლისთვისაც $V_A(X', y)$=„კი", ანუ $A(X')$=„არა".

ზემოთ ნათქვამიდან გამომდინარე, დასამტკიცებელია შემდეგი ლემა:

ლემა 3.8: ნებისმიერი R ალგორითმიდან, რომლის დროის ფუნქციაა $f_R(n)$, $O(f_R(n)^{const})$ დროში შეიძლება აიგოს შესაბამისი CR სქემა.

დამტკიცება: აქ ჩვენ არა მკაცრ დამტკიცებას, არამედ ძირითად იდეას მოვიყვანთ. ყოველი ალგორითმი შეგვიძლია განვიხილოთ, როგორც რაღაც $x_1, ..., x_k$ ცვლადების მიმდევრობა, რომელზედაც ყოველ ბიჯზე $x_i = \phi(y_j, y_l)$ ოპერაცია ტარდება, სადაც y_j, y_l ან ცვლადები, ან მუდმივებია (მაგ. $x_3 = x_{507} + x_3$, ან $x_9 = x_{71} \cdot 12$ და ა.შ.).
ყოველი x_i ცვლადისთვის, რომელიც შეგვხვდება ჩვენს ალგორითმში, განვათავსოთ (თითო ცვლადისთვის შესაბამისი ბიტების რაოდენობის) ვერტიკალური მავთულები.
ცხადია, რომ ყოველი ϕ ოპერაციისთვის შესაძლებელია ფიქსირებული სქემის შექმნა (იმ პირობით, თუ მონაცემების ბიტების რაოდენობა წინასწარაა დაფიქსირებული).
თუ გვაქვს გამოსახულება $x_{x_i} = \phi_m(x_{x_j}, x_{x_l})$, სადაც ϕ რაიმე კონკრეტული ოპერაციაა, მისი რეალიზაცია შეიძლება ქვემოთ მოყვანილი სქემით.

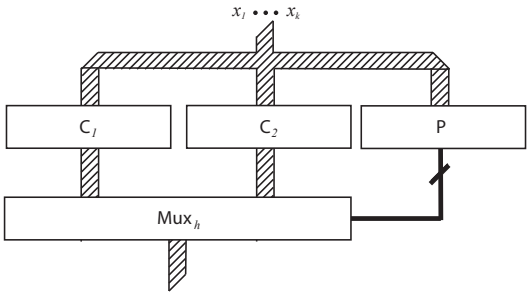


$x_{x_i} = \phi_m(x_{x_j}, x_{x_l})$ სქემა

აქ ცვლადების მიმდევრობა გამოსახულია, როგორც დაშტრიხული ზოლი, რომელსაც ჩვენ „მონაცემთა ზოლს" ვუოდებთ. ამ ზოლიდან შესაძლებელია ნებისმიერ ადგილზე საჭირო ინფორმაციის (ამ შემთხვევაში ცვლადების შესაბამისი მავთულების) გამოტანა. Select$_h$ ისეთი სქემაა, რომელიც ზემოდან მიღებული ცვლადებიდან (ამ შემთხვევაში ყველა ცვლადიდან) აირჩევს მეიმდენეს, რაც მარჯვენა სიგნალებშია კოდირებული, ხოლო Store$_n$ მისი შებრუნებულია: ზემოდან იღებს ყველა ცვლადს და ასევე გამოთვლის შედეგს, ქვემოთ კი გამოუშვებს უცვლელად ყველა ცვლადს ერთის გარდა: მარჯვნივ მიღებულ ინფორმაციაში კოდირებული რიცხვის პოზიციაზე მდგარ ცვლადს შეცვლის გამოთვლის შედეგით.
საბოლოო ჯამში მივიღებთ ცვლადების აქტუალურ მიმდევრობას.
თუ ალგორითმის ყოველ სტრიქონში ჩაწერილ ბრძანებას ამ მეთოდით გადავაკეთებთ სქემებად, მათი ერთი მეორეზე ვერტიკალურად გადაბმის შედეგად მივიღებთ ამ ალგორითმის შესაბამის სქემას, სადაც გამოსასვლელ სიგნალებში კოდირებული იქნება საწყისი ცვლადების საბოლოო მნიშვნელობები თავდაპირველად მოცემული მიმდევრობით.
თუ ალგორითმის ნაწილში ჩაწერილია

if(P)
then კოდი C_1
else კოდი C_2

მისი შესაბამისი სქემა იქნება



if(P) then C_1 else C_2 სქემა

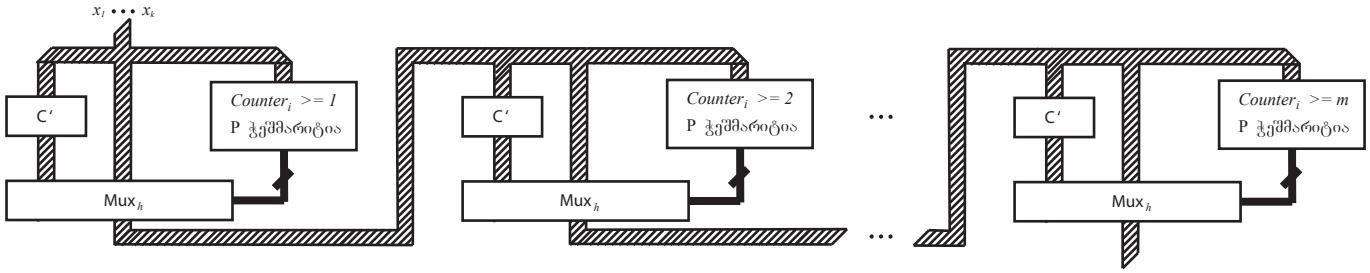
ბოლოს გასარკვევია, თუ როგორ შეიძლება ციკლის რეალიზაცია:

for(P)
{ კოდი C }

პირველ რიგში უნდა განვსაზღვროთ, თუ ***მაქსიმუმ*** რამდენჯერ შეიძლება დატრიალდეს ციკლი და C კოდის შესაბამისი სქემაც ამდენჯერ გადავაბათ ვერტიკალურად ერთმანეთს.
მეორე საკითხია, თუ ამ ჯაჭვის რომელი კომპონენტის გამოთვლა უნდა იყოს საბოლოო (ცხადია, რომ უმეტეს შემთხვევაში ციკლი უფრო ნაკლებჯერ დატრიალდება).
ამისათვის შემოვიღოთ ერთი დამატებითი ცვლადი $Counter_i$ (რომელიც ასევე საერთო ცვლადების მიმდევრობაში იქნება რომელიმე პოზიციაზე) და ციკლი შემდეგნაირად გადავწეროთ:

$Counter_i = 0$;
for(P)
{
$Counter_i = Counter_i + 1$;
კოდი C
}

ზემოთ ხსენებული ჯაჭვის m-ურ კომპონენტში მოწმდება პირობა $Counter_i \geq m$ და თუ ეს ასეა, პასუხად ამ კომპონენტის გამოთვლა იქნება გადაცემული, წინააღმდეგ შემთხვევაში კი – წინა კომპონენტის (ნახ. 3.2)



for ციკლის სქემა

შენიშვნა: აქ ჩავთვალოთ, რომ $Counter_i = 0$ ოპერაცია უკვე რეალიზებულია და C' სქემა C კოდის რეალიზებასაც და მთვლელის გაზრდასაც მოიცავს. ამას გარდა, P გამონათქვამი ყველა შრისთვის სხვადასხვა ცვლადს მოიცავს და, აქედან გამომდინარე, რაღაც ადგილიდან დაწყებული მცდარი იქნება.

ცხადია, რომ ამ მეთოდით ნებისმიერი ალგორითმის სქემის შექმნა შეიძლება და ამ სქემის „შრეების" რაოდენობა ალგორითმის მაქსიმალური ბიჯების რაოდენობის მუდმივ ხარისხს (უფრო ზუსტად, კუბს) არ გადააჭარბებს.

რ.დ.გ.

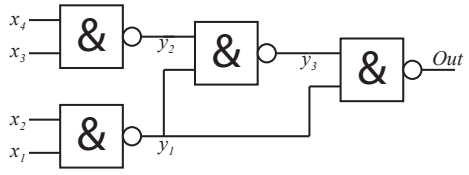
სავარჯიშო 3.9: ახსენით, თუ რის გამო იქნება ზემოთ მოყვანილი ჯაჭვის რაღაც ადგილამდე P გამონათქვამი ჭეშმარიტი და შემდეგ კი – ყველგან მცდარი. რა ტიპის პირობაა P?

სავარჯიშო 3.10: ახსენით, თუ რატომ შეიძლება გახდეს საჭირო ყოველ ციკლში თავისი მთვლელის შემოღება. რატომ არ შეიძლებოდა ერთი მთვლელით ოპერირება?

პირველი NP-სრული ამოცანის გამოყოფის შემდეგ უნდა აიგოს გარკვეული „ჯაჭვი" იმ ამოცანებთან, რომლებიც ჩვენ აქამდე განვიხილეთ და რომლებიც ერთი მეორეზე დავიყვანეთ. ამით დამტკიცდება ამ თითოეული ამოცანის NP-სრულობაც.
ლოგიკური იქნებოდა, თუ *CSAT* ამოცანას *SAT* ამოცანაზე დავიყვანდით: მოცემული სქემისთვის ჩავწერდით შესაბამის ფუნქციას კონიუნქციური ნორმალური ფორმით. ცხადია, თუ ეს ფუნქცია დაკმაყოფილდება, მაშინ ასევე დაკმაყოფილდება თავდაპირველი სქემაც.
მაგრამ ეს მიდგომა გარკვეულ ხიფათს შეიცავს: როგორც წინა სემესტრის მასალიდანაა ცნობილი, ფუნქციის დიზიუნქციური ნორმალური ფორმის პოვნა ექსპონენციურ დროს მოითხოვს (ცვლადების რაოდენობასთან შედარებით), ჩვენ კი პოლინომური დაყვანა გვაინტერესებს.

ამ საკითხის მოგვარებაში გვეხმარება ე.წ. „ცაიტინის გარდაქმნა" (Tseytin transformation).
თეორემა 3.11 მოცემულია სქემა, რომელიც იყენებს მხოლოდ *NAND* კომპონენტს (ანუ $\overline{x \cdot y}$). მისი შესაბამისი ფუნქცია პოლინომურ დროში ჩაიწერება კონიუნქციური ნორმალური ფორმით.

დამტკიცება: $z = \overline{x \cdot y}$ გამოსახულებაში შემავალი x, y, z სამეულის ურთიერთდამოკიდებულება (ანუ რა კომბინაციებია დასაშვები) გამოიხატება ფუნქციით $(x \vee z)(y \vee z)(\bar{x} \vee \bar{y} \vee \bar{z})$ (ეს გამოსახულება იღებს მნიშვნელობას 1 მაშინ და მხოლოდ მაშინ, თუ z ცვლადის მნიშვნელობა x და y ცვლადების მნიშვნელობათა კონიუნქციის უარყოფაა).
განვიხილოთ მოცემული C სქემა (მაგალითისთვის განვიხილოთ სქემა, რომელიც ნაჩვენებია ნახაზში 3.2).



NAND ელემენტებზე აგებული სქემა ცვლადებით

ყოველი *NAND* ელემენტის გამომავალი სიგნალიც ცალკე ცვლადით მოვნიშნოთ, ხოლო ამ სქემის გამომავალ სიგნალს ვუწოდოთ *Out*.
მაშინ ქვედა მარცხენა *NAND* ელემენტის აღწერა შეიძლება გამოსახულებით $(x_1 \vee y_1)(x_2 \vee y_1)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{y}_1)$, მარცხენა ზედასი – $(x_3 \vee y_2)(x_4 \vee y_2)(\bar{x}_3 \vee \bar{x}_4 \vee \bar{y}_2)$, შუა ელემენტის როგორც $(y_1 \vee y_3)(y_2 \vee y_3)(\bar{y}_1 \vee \bar{y}_2 \vee \bar{y}_3)$ და ბოლო ელემენტის – $(y_1 \vee Out)(y_3 \vee Out)(\bar{y}_1 \vee \bar{y}_3 \vee \overline{Out})$.
ქართული სიტყვებით რომ ვთქვათ, ყოველი გამოსახულება გვეუბნება: „y_1, y_2, y_3 და *Out* გამოთვლის კანონიერი შედეგია. იმისათვის, რომ გავარკვიოთ, იღებს თუ არა ეს სქემა ოდესმე მნიშვნელობას 1, საკმარისია ამ გამოსახულებების კონიუნქციებით შეკვრა და – რადგან გვაინტერესებს, არის თუ არა შედეგი 1 – თვით *Out* ცვლადიც კონიუნქციით უნდა მივუწეროთ. შედეგად ვიღებთ:

$$\begin{array}{l}(x_1 \vee y_1)(x_2 \vee y_1)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{y}_1) \wedge \\ (x_3 \vee y_2)(x_4 \vee y_2)(\bar{x}_3 \vee \bar{x}_4 \vee \bar{y}_2) \wedge \\ (y_1 \vee y_3)(y_2 \vee y_3)(\bar{y}_1 \vee \bar{y}_2 \vee \bar{y}_3) \wedge \\ (y_1 \vee Out)(y_3 \vee Out)(\bar{y}_1 \vee \bar{y}_3 \vee \overline{Out}) \wedge Out\end{array}$$

სავარჯიშო 3.12: დაამტკიცეთ, რომ ამ მეთოდით შედგენილი კონიუნქციური ნორმალური ფორმა მოგვცემს საწყისი სქემის ფუნქციას.

სავარჯიშო 3.13: დაამტკიცეთ, რომ ნებისმიერი სქემა შეგვიძლია გადავიყვანოთ მხოლოდ *NOR* ელემენტებით აგებულ სქემაში (კონიუნქცია, დიზიუნქცია და უარყოფა აღწერეთ *NOR* ელემენტების გამოყენებით).

ამ სავარჯიშოებით მთელი დამტკიცება დასრულებულია.

რ.დ.გ.

თავი 4

არაამოხსნადობისა და NP სრული ამოცანების პრაქტიკული მნიშვნელობა

NPკლასისა და მისთვის რთული ან სრული ამოცანების უკეთ შესწავლისთვის საჭიროა ასევე იმის გააზრება, თუ რა კავშირშია იგი მის გარეთ მდგომ (კერძოდ კი არაამოხსნად ამოცანათა) სიმრავლესთან. აქ არსებობს ერთი მნიშვნელოვანი ფაქტი, რომელსაც ჩვენ ახლა განვიხილავთ.

4.1 შეჩერების ამოცანის NP-რთულობა

საინტერესოა ის ფაქტი, რომ არსებობს ისეთი არაამოხსნადი ამოცანა (კერძოდ კი შეჩერების ამოცანა H), რომელზედაც ყველა რეკურსიული ამოცანა დაიყვანება. ეს კი იმას ნიშნავს, რომ შეჩერების ამოცანა ყველა ამოხსნად ამოცანაზე „რთული" უნდა იყოს. რა თქმა უნდა, ეს ფაქტი პირდაპირ პრაქტიკულ შედეგს ვერ მოგვცემს, რადგან არაამოხსნადი ამოცანის რეალიზაციაც არაფერს მოგვცემს, მაგრამ მისი გამოყენებით მნიშვნელოვანი თეორიული და, აქედან გამომდინარე, პრაქტიკული შედეგების მიღება შეიძლება.
სანამ კონკრეტული თეორემის დამტკიცებას განვიხილავთ, ჯერ უნდა გავიაზროთ, თუ რას ნიშნავს ერთი ამოცანის მეორეზე დაყვანა ამ ამოცანების შესაბამისი ენის ტერმინებში.
განვიხილოთ სიმრავლე L_H, რომელიც ისეთი K ალგორითმისა და X მონაცემის (K, X) წყვილებისგან შედგება, რომ $K(X)$ აუცილებლად შეჩერდება.
ახლა კი განვიხილოთ ნებისმიერი რეკურსიული A ამოცანის შესაბამისი ენა L_A, რომელიც იმ X მონაცემებისგან შედგება, რომელთათვისაც $A(X) =$„კი".
დაყვანის ალგორითმით A ამოცანის X მონაცემი უნდა გადავაკეთოთ ისეთ A' ალგორითმსა და X' მონაცემში, რომ $X \in L_A \Leftrightarrow (A', X') \in L_A$.

ამის გათვალისწინებით დავამტკიცოთ

თეორემა 4.1 შეჩერების ამოცანა NP-რთულია.

დამტკიცება: განვიხილოთ ნებისმიერი გადაწყვეტილების ამოცანა NPკლასიდან. რადგან იგი რეკურსიულია, უნდა არსებობდეს შესაბამისი ალგორითმი A, რომელიც მას ამოხსნის (იმ ფაქტს, რომ ეს ალგორითმი ძალიან ნელი - ექსპონენციურიც კი შეიძლება იყოს, აქ არანაირი მნიშვნელობა არ აქვს, რადგან ჩვენ უბრალოდ ამოხსნადობა და არა მისი სისწრაფე გვაინტერესებს).
ჩვენ გვაინტერესებს, თუ რა პასუხს მოგვცემს A ალგორითმი X მონაცემზე, ანუ $A(X)$.
შევქმნათ ალგორითმი H_A შემდეგნაირად:

ალგორითმი 4.1: H_A
მოცემულია: ალგორითმი A და მისი მონაცემი X

```
1: w = A(X);
2: while(¬w)
```

ცხადია, რომ ეს ალგორითმი შეჩერდება მაშინ და მხოლოდ მაშინ, თუ A ალგორითმი X მონაცემზე მოგვცემს პასუხს „კი". აქედან გამომდინარე, თავდაპირველი ამოცანა დავიყვანეთ შეჩერების ამოცანაზე: თუ H_A შეჩერდება, თავდაპირველი ამოცანის პასუხი ყოფილა „კი" და თუ არ შეჩერდება, თავდაპირველი ამოცანის პასუხი იქნებოდა „არა".

ბუნებრივია შეკითხვა: არსებობს თუ არა ისეთი ამოცანა NPკლასის მიღმა, რომელიც არ იქნება NP-რთული? თუ ამ კლასის გარეთ ყველა ამოცანა უფრო „რთულია", ვიდრე ამ კლასში შემავალი?
ცნობილი თეორემა, რომელსაც დაწვრილებით შემდგომში განვიხილავთ, გვეუბნება, რომ ნებისმიერი სირთულის ამოცანაზე უფრო რთული ამოცანის შედგენაა შესაძლებელი (ანუ „მჯობნის მჯობნი არ დაილევა"). მაგრამ აქ გვაინტერესებს, არსებობს თუ არა უფრო „სუსტი " ამოცანა NPკლასს გარეთ?
თუ განვიხილავთ ზემოთ ნახსენებ co-NP კლასის სრულ ამოცანებს, co-NP ≠NPდაშვებით, ეს ამოცანა ვერ იქნება NP-რთული. მაგრამ ეს მხოლოდ მნიშვნელოვანი დაშვებით, რაც ჯერ-ჯერობით უცნობია.

4.2 NP-სრული ამოცანების მნიშვნელობა

მას შემდეგ, რაც 1972 წელს რიჩარდ კარპმა თავის ცნობილ სტატიაში SAT ამოცანიდან გამომდინარე დაყვანის პრინციპით 21 ამოცანის NPსრულობა დაამტკიცა, ამ საკითხის მიმართ ინტერესი არ შენელებულა. ამის ძირითადი მიზეზი არის ის, რომ ნებისმიერი NPსრული ამოცანის სწრაფად ამოხსნა სხვა ნებისმიერი სერტიფიცირებადი ამოცანის სწრაფად ამოხსნას ნიშნავს და იმ დროს ბევრმა მკვლევარმა სწორედ რომელიმე NPსრული ამოცანის სწრაფად ამოხსნაში დაინახა მთავარი პრობლემა.
სამწუხაროდ, უამრავი მცდელობის მიუხედავად, ვერც ერთი ასეთი ამოცანა სწრაფად ვერ გადაჭრილა, თუმცა ამის იმედი ბევრჯერ გაჩენილა და დღესაც - თითქმის ნახევარი საუკუნის შემდეგ – კიდევ არ მილეულა, თუმცა საგრძნობლად კი შემცირდა.
მეორე მიზეზი, თუ რატომაა NP-სრულობა ასეთი მნიშვნელოვანი, მისი „უნივერსალურიბაა": რადგან ყველა ამოცანა ამ ტიპის პრობლემებზე დაიყვანება, ისინი ამა თუ იმ ფორმით იჩენენ თავს სხვადასხვა, ერთი შეხედვით შორს მდგომ ამოცანებშიც. აქედან გამომდინარე, თუ გვეცოდინება „გადაკეთებული" NPსრული ამოცანის რაიმე სახით გადაჭრის გზები, ეს შეიძლება ბევრი სხვა ამოცანის გადაჭრაში დაგვეხმაროს.
რადგან ერთი NPსრული ამოცანის სწრაფი ამოხსნა ყველა დანარჩენის სწრაფ ამოხსნასაც გამოიწვევს, ეს ფაქტი კი ძალიან გასაკვირი იქნებოდა NPსრული ამოცანების ერთმანეთისაგან საკმაოდ განსხვავებული სტრუქტურისა და ერთი შეხედვით „მარტივი" ამოცანებთან შედარებით აშკარა სირთულეების გამო, დღეისათვის მკვლევართა აბსოლუტური უმრავლესობა მიიჩნევს, რომ ამ კლასის ამოცანების ზუსტ ამოხსნაზე დროის კარგვა არაა მიზანშეწონილი და მათი გადაჭრის ალტერნატიულ გზებს ეძებს (მაგალითად მიახლოებითი ალგორითმები, სადაც ოპტიმიზაციის ამოცანას რაიმე პატარა მისაღები ცდომილებით ამოვხსნით, ან ალბათური ალგორითმებით, რომლითაც გამოთვლილი პასუხი დიდი ალბათობით სწორია, მაგრამ შესაძლებელია არასწორიც იყოს, მაგრამ ამ ალგორითმის ბევრჯერ გაშვებისას ყველაზე ხშირად მიღებული პასუხი პრაქტიკულად სწორი იქნება, თუმცა შეცდომის ალბათობა მაინც იარსებებს და ა.შ.).
ყოველივე ზემოთ თქმულიდან გამომდინარე, უცნობი ამოცანისთვის პირველ რიგში გასარკვევია, არის თუ არა იგი NPსრული და თუ არის, ამოხსნის ალტერნატიული გზებია საძებნი.

იმისათვის, რომ რაიმე ახალი ამოცანის NPსრულობა დამტკიცდეს, საჭიროა რაიმე ცნობილი NPსრული ამოცანის მასზე დაყვანა. თეორიულად ნებისმიერი ცნობილი სრული ამოცანის დაყვანა უნდა იყოს შესაძლებელი, მაგრამ, როგორც პრაქტიკამ გვაჩვენა, რაიმე კონკრეტულ ამოცანაზე ზოგი საწყისი NPსრული ამოცანიდან დაყვანა გაცილებით უფრო მარტივია, ვიდრე სხვა საწყისი ამოცანიდან.
გამოცდილებიდან გამომდინარე, ადვილი დაყვანის საპოვნელად ოთხ საწყის NPსრულ ამოცანას განიხილავენ:

- VC_k (წიბოების გადაფარვა): გამოიყენება ისეთ ამოცანებზე დასაყვანად, რომლებიც გრაფებში ამორჩევას ეხება;
- HP (ჰამილტონის გზა): ამოცანებისთვის გრაფებზე, რომლებიც დამოკიდებულია გადალაგებებსა და მარშრუტიზაციაზე;
- IP (რიცვთა დაყოფა: მოცემულია ნატურალურ რიცხვთა სიმრავლე; შეიძლება თუ არა მისი დაყოფა ორ ქვესიმრავლედ ისე, რომ თითოეულ ქვესიმრავლეში შესულ რიცხვთა ჯამი ტოლი იყოს?): ისეთი ამოცანებისთვის, რომელთა სირთულე დამოკიდებულია დიდ რიცხვებზე;

- 3-*SAT* (სამცვლადიანი დნფ ფუნქციების შესრულებადობა): ისეთი ამოცანებისთვის, რომლებსაც არც ერთი ზემოთ ჩამოთვლილი საწყისი ამოცანა არ შეესაბამება.

სავარჯიშო 4.2: განიხილეთ ზურგჩანთის დისკრეტული ამოცანა: მოცემულია ზურგჩანთა W მოცულობით და n ცალი ტვირთი, თითოეული მოცულობით $a_1, ..., a_n$. შეიძლება თუ არა $a_{i_1}, ..., a_{i_k}$ ტვირთის შერჩევა ისე, რომ ზურგჩანთა მთლიანად შეივსოს (ანუ $a_{i_1} + \cdots + a_{i_k} = W$)?
დაამტკიცეთ ამ ამოცანის NPსრულობა.

სავარჯიშო 4.3: განიხილეთ ზურგჩანთის ამოცანის დინამიურ დაპროგრამებაზე დაფუძნებული ალგორითმი (იხ. წინა სემესტრის მასალა).
რა არის ამ ალგორითმის დროის ზედა ზღვარი?

სავარჯიშო 4.4: განიხილეთ ქვეგრაფის იზომორფულობის ამოცანა *SGI*. რომელი საწყისი ამოცანიდანაა მისი NPსრულობის დამტკიცება ყველაზე მოსახერხებელი? დაიყვანეთ ეს ამოცანა ქვეგრაფის იზომორფიზმზე.

სავარჯიშო 4.5: დაამტკიცეთ შემდეგი ამოცანის NPსრულობა.
პროგრამათა ექვივალენტურობა: მოცემულია ცვლადების სასრული სიმრავლე X და მნიშვნელობათა სასრული სიმრავლე V, ასევე ორი პროგრამა P_1 და P_2, რომელიც შემდეგი ტიპის ბრძანებებისგან შედგება:

$x_0 \longleftarrow \text{if}(x_1 = x_2) \text{ then } x_3 \text{ else } x_4$

არსებობს თუ არა X სიმრავლის ცვლადების ისეთი მნიშვნელობები V სიმრავლიდან, რომ P_1 და P_2 პროგრამა სხვადასხვა პასუხს იძლეოდეს? ან, სხვა სიტყვებით რომ ვთქვათ, იძლევა თუ არა ეს ორი პროგრამა ერთსა და იმავე მონაცემზე ერთსა და იმავე პასუხს?

4.3 არაგამოთვლადი ფუნქციების გამოყენება

რაც არ უნდა გასაკვირი იყოს, არაგამოთვლადი ფუნქცია შეიძლება ძალიან მნიშვნელოვან როლს თამაშობდეს პრაქტიკაში. ერთ-ერთი ასეთი გამოყენების სადემონსტრაციოდ განვიხილოთ ე.წ. „საქმიანი თახვის" ამოცანა.
როგორც ცნობილია, თახვები უაღრესად ბეჯითი ცხოველები არიან: გადადიან ტყეში, იღებენ მორებს, მდინარეში მიცურავენ და წყალსაგუბებლებს აშენებენ. ასეთ იქით-აქეთ სირბილს თახვი თავს არ ანებებს, სანამ თავის სამუშაოს ბოლომდე არ დაასრულებს. თახვების მსგავსად მოქმედებს ალგორითმიც: გამოთვლის პროცესში ცვლადებს შორის იქით-აქეთ დარბის და მეხსიერებაში გარკვეულ სიმბოლოებს წერს.
1962 წელს უნგრელმა მათემატიკოსმა ტიბორ რადომ წამოაყენა „საქმიანი თახვის" ამოცანა: მოცემულია ორობით კოდში ჩაწერილი n სიგრძის ალგორითმი, რომელიც მუშაობას ცარიელი სიტყვით (როგორც მონაცემი) და ნულის ტოლი ცვლადებით იწყებს და როდესღაც ჩერდება. რამდენი 1 იქნება დაწერილი მეხსიერებაში მანქანის გაჩერების შემდეგ? და ზოგადად: მაქსიმუმ რამდენი 1 შეიძლება დაწეროს n სიგრძის ალგორითმმა გაჩერებამდე, თუ იგი მუშაობას ცარიელი მეხსიერებით (ნულის ტოლი ცვლადებით) დაიწყებს? ამ რიცხვს გამოსახავენ $\Sigma(n)$ ფუნქციით, რომელსაც ***რადოს ფუნქციას*** უწოდებენ. ანალოგიურად შეიძლება დავსვათ შეკითხვა: მაქსიმუმ რამდენი ბიჯის შემდეგ გაჩერდება n სიგრძის ალგორითმი? ეს რიცხვი აღინიშნება ფუნქციით $S(n)$ (თუ ალგორითმი არ შეჩერდება, მაშინ $S(n) = 0$). როგორც დამტკიცებულია, ეს ორივე ფუნქცია უფრო სწრაფად იზრდება, ვიდრე *ნებისმიერი* გამოთვლადი ფუნქცია, ანუ ორივე არაგამოთვლადია:

n	1	2	3	4	5	6
$\Sigma(n)$	1	4	6	13	≥ 4098	$\geq 10^{1439}$
$S(n)$	1	6	21	107	$\geq 47.176.870$	$\geq 10^{2879}$

შენიშვნა: რადოს ფუნქცია განისაზღვრება ე.წ. ტიურინგის მანქანებით - გამოთვლის მოდელით, რომელიც სტანდარტულადაა მიჩნეული. აქ ჩვენ მის ანალოგს განვიხილავთ, რომელიც ინტუიციურ დონეზე (სიმკაცრის ხარჯზე) შედეგების მნიშვნელობას წარმოაჩენს.

ამ ფუნქციების გამოუთვლადობის მიუხედავად, უაღრესად დიდი მნიშვნელობა აქვს მათ დათვლას კონკრეტული n-თვის. მაგალითად, თუ შევადგენთ ზემოთ ნახსენებ ალგორითმს, რომელიც რიგ-რიგობით გადაამოწმებს ნატურალურ რიცხვებს და გაჩერდება მაშინ და მხოლოდ მაშინ, თუ ისეთ ლუწ რიცხვს აღმოაჩენს, რომელიც არ წარმოდგება ორი მარტივი რიცხვის ჯამის სახით, მაშინ შეიძლება გოლდბახის ჰიპოთეზის შემოწმება: დავითვლით

ამ ალგორითმის სიგრძეს n, გამოვითვლით $S(n)$ ფუნქციას და ამ მანქანას ვამუშავებთ $S(n)$ ბიჯის განმავლობაში. თუ იგი ამ დროში არ შეჩერდა, ესე იგი აღარასდროს შეჩერდება და ჩვენ რიცხვთა თეორიის ერთ-ერთ უმნიშვნელოვანეს საკითხს გადავჭრით.
სამწუხაროდ, ჯერ-ჯერობით ასეთი პერსპექტივა საკმაოდ ბუნდოვანია.

სავარჯიშო 4.6: დაამტკიცეთ, რომ $S(n)$ ფუნქცია არაა გამოთვლადი.
მინიშნება: შეჩერების ამოცანა დაიყვანეთ $S(n)$ ფუნქციის გამოთვლის ამოცანაზე.

სავარჯიშო 4.7: დაამტკიცეთ შემდეგი გამონათქვამის ჭეშმარიტება:
(A არაგამოთვლადია, $A \leq_p B$) $\Leftrightarrow$ (B არაგამოთვლადია)

ამ სავარჯიშოდან გამომდინარეობს მნიშვნელოვანი თეორემა:

თეორემა 4.8 ნებისმიერი არაგამოთვლადი ამოცანა ნებისმიერ გამოთვლად ამოცანაზე რთულია.

თავი 5

P vs. NP

თანამედროვე მეცნიერების ერთ-ერთი უმნიშვნელოვანესი დია საკითხია P vs. NP ამოცანა, რომლის ჩამოყალიბებასაც, ფაქტიურად, წინა თავების მასალა მივუძღვენით. იმ ფაქტიდან გამომდინარე, რომ მკაცრად ვერც ტოლობისა და ვერც უტოლობის დამტკიცება ვერ მოხერხდა, მეცნიერებმა იმაზე დაიწყეს ფიქრი, თუ რა მოხდება ამა თუ იმ ვარიანტის ჭეშმარიტების შემთხვევაში. ამ საკითხების გამოკვლევას ეძღვნება ამ თავის თემა, სადაც ბოლოში ერთი ძალიან მნიშვნელოვანი თეორემა დამტკიცდება შუალედური ენის არსებობის შესახებ.

5.1 P vs. NP ამოცანა და მისი გადაჭრის ვარიანტები

P vs. NP ამოცანის პასუხის სხვადასხვა ვარიანტი არსებობს:

1. P=NP: ამ შემთხვევაში ყველა ამოცანა, რომელიც პოლინუმურ დროში სტრიფიცირებადია, ასევე პოლინომურ დროში ამოხსნადი იქნება. მაგრამ ეს სულაც არ ნიშნავს იმას, რომ ადვილი იქნება ასეთი ალგორითმების პოვნა. ეს პირველ რიგში ე.წ. „არსებობის თეორემა" იქნება, ანუ გვეცოდინება, რომ სწრაფი ამოხსნა შესაძლებელია, მაგრამ ასეთი ამოხსნის პოვნა ცალკე ამოცანაა. რა თქმა უნდა, შესაძლებელია ე.წ. „კონსტრუქციული დამტკიცება", ანუ ვინმე რომელიმე NPსრული ამოცანისთვის დაწერს სწრაფ ალგორითმს, რითიც ტოლობა დამტკიცდება და ასევე კონკრეტული პოლინომური ამოხსნაც იქნება მოცემული, თუმცა ასეთი პერსპექტივა დღევანდელი მიღწევებიდან გამომდინარე საკმაოდ შორია.

   ხუმრობით იმასაც ამბობენ, რომ P = NPშემთხვევაში ვინც გაუსის დამტკიცებას გაიგებს, ის გაუსივით ჭკვიანი იქნება.

2. P ≠ NPეს შედეგი იმის მაჩვენებელი იქნება, რომ არსებობს ისეთი (არაერთი) სწრაფად სერტიფიცირებადი ამოცანა, რომლის სწრაფად ამოხსნა შეუძლებელია. პრაქტიკაში ეს იმას უნდა ნიშნავდეს, რომ NP- სრული ამოცანებისთვის ამოხსნის თავისებური გზებია საძებნი: ან მიახლოებითი ამოხსნა (ოპტიმიზაციის ამოცანებში), ან მონაცემთა შეზღუდვა, ან ე.წ. ევრისტიკის ძიება და სხვა. ამ საკითხებს ცოტა მოგვიანებით შევეხებით.

3. ორივე დაშვება *თავის თავად და ერთმანეთისგან დამოუკიდებლად,* ჭეშმარიტია. ასეთი შემთხვევები მათემატიკაში ცნობილია: ნებისმიერი მათემატიკური თეორია დაფუძნებულია აქსიომებზე, რომელთა ლოგიკური კომბინაციებით მიიღება თეორემები, მათი ლოგიკური კომბინაციით კიდევ დამატებითი თეორემები და ამგვარად შეიძლება თეორიის გაფართოება. მთავარი ის არის, რომ ლოგიკური დასკვნის ჯაჭვით არ მივიღოთ ორი ურთიერთგამომრიცხავი თეორემა (ეს არის ლოგიკის ერთ-ერთი ძირითადი პრინციპი: ერთ თეორიაში არ უნდა მტკიცდებოდეს რაიმე გამონათქვამი და მისი უარყოფა. ამ შემთხვევაში იტყვიან, რომ ეს სისტემა არაწინააღმდეგობრივია). თუ ვინმე შეძლებს გამოთვლის თეორიის აქსიომების შექმნას, საიდანაც (ცნობილი) თეორემების გამოყვანა შეიძლება, ამ აქსიომებს P ≠ NP თეზისს მიუმატებს და დაამტკიცებს, რომ მიღებული სისტემა არაწინააღმდეგობრივია, მაშინ შეიძლება იმის დაჯერება, რომ P = NPჭეშმარიტია. მაგრამ, ამის და მიუხედავად, თუ იგივე აქსიომებს დავუმატებთ P ≠ NPაქსიომას და ასევე არაწინააღმდეგობრივ სისტემას მივიღებთ, ასევე იარსებებს თეორია, რომელშიც P ≠ NP. ეს ორი თეორია *ერთმანეთისგან დამოუკიდებლად* ჭეშმარიტი იქნება.

ეს საკმაოდ რთული საკითხებია, რომელთა დაწვრილებით შესწავლა ჩვენი კურსის ფარგლებს ცდება, თუმცა სპეციალური კურსების ფარგლებში განიხილება.

5.2 P ≠ NP დაშვებიდან გამომდინარე შედეგები

NP-სრული ამოცანებისთვის სწრაფი ამოხსნის თითქმის ორმოცდაათი წლის განმავლობაში წარუმატებელი ძიების შედეგად ალგორითმების თეორიითა და პრაქტიკით დაინტერესებული პირები იძულებილები გახდნენ, თავიანთი კვლევა P ≠ NP დაშვების შედეგად ეწარმოებინათ: დავუშვათ, რომ P ≠ NP. როგორ ამოვხსნათ რთული ამოცანები? რა სტრუქტურა ექნება NPდა co-NP სიმრავლეებს? რა დამოკიდებულებაა მათ შორის? ამ საკითხებზე შემდგომში გვექნება ლაპარაკი.

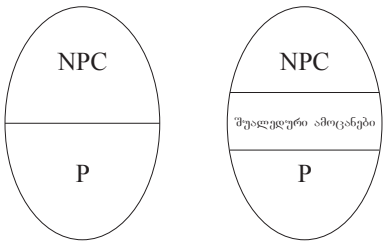
აღსანიშნავია ის ფაქტიც, რომ თუ წინა პარაგრაფში განხილული შესაძლებლობების მესამე პუნქტს განვიხილავთ, ანუ ორივე შესაძლებლობა იქნება დასაშვები და P = NP შემთხვევას მივიღებთ, NP-სრული ამოცანებისთვის მაინც ვერ შევძლებთ კონკრეტული სწრაფი ალგორითმის დაწერას.

<u>სავარჯიშო 5.1:</u> დაამტკიცეთ, რომ თუ არც P = NPდა არც P ≠ NPგამოიწვევს წინააღმდეგობას და ამ შემთხვევაში დავუშვებთ, რომ P = NP, NP-სრული ამოცანებისთვის სწრაფი ალგორითმის არსებობა ჭეშმარიტი იქნება, მაგრამ ასეთ ალგორითმს ვერავინ აღმოაჩენს.

ასევე საინტერესოა, რომ P ≠ NP დაშვების პირობებში *შესაძლებელია,* ამოცანა „დაამტკიცეთ P ≠ NP" თვითონ NP-სრული აღმოჩნდეს და, აქედან გამომდინარე, დამტკიცების პოვნაც რეალურ დროში შეუძლებელი იყოს.

5.2.1 შუალედური ენის არსებობა

P ≠ NP დაშვების პირობებში NPსიმრავლის ორი ვარიანტი არსებობს, რომელიც ნაჩვენებია სურათში 5.2.1.



(ა) (ბ)

NP სიმრავლის შესაძლო სტრუქტურა P ≠ NP დაშვების პირობებში
ან NP სიმრავლე ზუსტად ორ P და NPC კლასად იყოფა, ან უნდა არსებობდეს ისეთი ენა (ამოცანა), რომელიც არ ამოიხსნება პოლინომურ დროში და არც NP-სრულია.
შემდეგი თეორემა სწორედ ამ ფაქტს ამტკიცებს.

<u>თეორემა 5.2</u> თუ P ≠ NP, მაშინ არსებობს ისეთი ენა NP სიმრავლეში, რომელიც არ ეკუთვნის P სიმრავლეს და ***ამავდროულად*** არ არის NP-სრული.

<u>დამტკიცება:</u> რადგან ალგორითმების სიმრავლე თვლადია, შესაძლებელია ყველა ისეთი $M_1, M_2, \ldots$ ალგორითმის გადანომვრა, რომელთა მუშაობის დრო ზემოდან პოლინომური ფუნქციითაა შემოსაზღვრული. ასევე შეიძლება $R_1, R_2, \ldots$ პოლინომურ დროში დაყვანათა ალგორითმების გადანომვრაც. რადგან ეს შესაძლებელია, უნდა არსებობდეს ალგორითმი TM_1, რომელიც რიგ-რიგობით მოგვცემს ასეთ M_i მიმდევრობას; ანალოგიურად უნდა არსებობდეს TM_2 ტიურინგის მანქანაც, რომელიც რიგ-რიგობით მოგვცემს R_i მიმდევრობას (ანუ M_i და, შესაბამისად, R_i ალგორითმების აღწერას).

შენიშვნა: აღსანიშნავია, რომ ამდაგვარი გადათვლის პოვნა ენათა ყველა კლასისთვის შესაძლებელი არაა. არსებობს ისეთ ენათა კლასიც, რომელთათვისაც გადათვლის პოვნა შესაძლებელი, მაგრამ ძალიან ძნელია.

ამას გარდა, TS იყოს ისეთი ალგორითმი, რომელიც გადაჭრის SAT ამოცანას (რადგან დავუშვით, რომ P ≠ NP, ეს ალგორითმი პასუხს ექსპონენციურ დროში გამოიანგარიშებს).
ახლა კი აღვწეროთ ისეთი L ენა, რომელიც არ იქნება P სიმრავლეში და არც NP- სრულია. ასეთ ენას ჩვენ პირდაპირ ვერ აღვწერთ, ამიტომ მოვიყვანთ ისეთი TK ალგორითმის მაგალითს, რომელიც მას გადაწყვეტს (მხოლოდ მის სიტყვებს მიიღებს), რითაც დავამტკიცებთ, რომ ასეთი ენა უნდა არსებობდეს.

თუ მოცემულია სპეციალური ფუნქცია $f : \mathbb{N} \to \mathbb{N}$ (რომლის განსაზღვრაც თეორემის დამტკიცების მთავარი ნაწილია), ზემოთ აღნიშნული ალგორითმის ფუნქცია (რომელსაც ჩვენ ასევე აღვნიშნავთ, როგორც TK) შემდეგნაირად განისაზღვრება:

$$TK(x) = \begin{cases} \text{„კი”} \\ \text{„არა”} \end{cases}, \text{ თუ } TS(x) = \text{ „კი” და ამავდროულად } f(|x|) \text{ ლუწია}$$

სხვა სიტყვებით რომ ვთქვათ, TK მხოლოდ მაშინ იღებს x სიტყვას, თუ x მოცემულია კონიუნქციური ნორმალური ფორმით, ცვლადების რომელიმე კომბინაციაზე იგი 1 ხდება (სრულდება) და მის სიგრძეზე გამოთვლილი f ფუნქცია ლუწია.

<u>სავარჯიშო 5.3:</u> ფორმალურად განსაზღვრეთ TK ალგორითმის ენა L_{TK}.

რაც შეეხება f ფუნქციას, იგი მონოტონურად ზრდადია, ანუ $f(n+1) \geq f(n)$ და $f(0) = 0$. თანაც f ძალიან ნელა იზრდება.
განვსაზღვროთ ალგორითმი TF, რომელიც ამ ფუნქციას გამოითვლის. TF ორ ეტაპად მუშაობს, თანაც ყოველი ეტაპი n ბიჯს გრძელდება.
პირველ ფაზაში TF ცდილობს მაქსიმალურად ბევრი $f(0), f(1), f(2), \ldots$ გამოანგარიშებას (რამდენსაც მოასწრებს n ბიჯში). დავუშვათ, რომ ბოლოს გამოთვლილი სიდიდე იყო $f(i) = k$. $f(n)$ სიდიდე იქნება ან k, ან $k+1$ იმის მიხედვით, თუ როგორ განვითარდება გამოთვლა მეორე ფაზაში.
მეორე ფაზაში მუშაობა იმაზეა დამოკიდებული, პირველ ფაზაში გამოთვლილი k რიცხვი კენტია თუ ლუწი. დავუშვათ, რომ $k = 2i$ ლუწია. მაშინ TF ითვლის $M_i(z)$, $S(z)$ და $TF(|z|)$, სადაც z რიგ-რიგობით გაირბენს ლექსიკოგრაფიულად დალაგებული Σ^* სიმრავლის ყველა ელემენტს და შეჩერდება მაშინ და მხოლოდ მაშინ, თუ აღმოაჩენს ისეთ z სიტყვას, რომ

$$TK(z) \neq M_i(z).$$

TK მანქანის განსაზღვრების თანახმად, იძებნება ისეთი z სიტყვა, რომლისთვისაც

- $M_i(z) =$„კი” და $S(z) =$„არა” ან $f(|z|)$ კენტია;
- $M_i(z) =$„არა” და $S(z) =$„კი” და $f(|z|)$ ლუწია.

აქაც გამოთვლა n ბიჯის შემდეგ წყდება და თუ ასეთი z არ აღმოჩნდა, $f(n) = k$. თუ ამ n ბიჯში ასეთი z აღმოჩნდა, მაშინ $f(n) = k+1$.

თუ პირველ ფაზაში გამოთვლილი $k = 2i+1$ კენტია, მაშინ მეორე ფაზა შემდეგნაირად ვითარდება: TF ითვლის $R_i(z)$, $S(R_i(z))$ და $F(|T_i(z))|)$, სადაც z ასევე ლექსიკოგრაფიულად გარბის Σ^* სიმრავლის ყველა მნიშვნელობას. აქ უკვე TF ეძებს ისეთ z სიტყვას, რომლისთვისაც

$$TK(R_i(z)) \neq S(z).$$

უნდა აღინიშნოს, რომ R_i დაყვანაა და, აქედან გამომდინარე, სიტყვას იძლევა.

TK მანქანის განსაზღვრების თანახმად, იძებნება ისეთი z სიტყვა, რომლისთვისაც

- $S(z) =$„კი” და ან $S(R_i(z)) =$„არა” ან $f(|z|)$ კენტია;
- $S(z) =$„არა” და $S(R_i(z)) =$„კი” და $f(|z|)$ ლუწია.

თუ გამოთვლისთვის გამოყოფილ n ბიჯში ასეთი z მოიძებნა, მაშინ $f(n) = k+1$. წინააღმდეგ შემთხვევაში $f(n) = k$.

აღსანიშნავია, რომ TF მკაცრად განსაზღვრული მანქანაა და $O(n)$ ბიჯში გამოითვლის $f(n)$ ფუნქციას.
აქედან გამომდინარე, ასევე TK იქნება მკაცრად განსაზღვრული.
<u>სავარჯიშო 5.4:</u> აჩვენეთ, რომ $L_{TK} \in$ NP.

ახლა დავუშვათ, რომ $L_{TK} \in$ P. მაშინ იარსებებს ისეთი M_i დასაწყისში ნახსენები გადანომვრის მიხედვით, რომლისთვისაც $TK(z) = M_i(z), \forall z$ (სხვა სიტყვებით რომ ვთქვათ, TK მანქანა ამ სიაში შეგვხვდება). მაგრამ ასეთ შემთხვევაში TF მანქანის მეორე ფაზაში არ აღმოჩნდება ისეთი z, რომლისთვისაც $TK(z) \neq M_i(z)$ და $f(n)$ ფუნქცია აღარ გაიზრდება და $f(n) = 2i, \forall n > n_0$. ეს კი იმას ნიშნავს, რომ დაწყებული რაღაც ადგილიდან L ენა

„გადაიზრდება” SAT ამოცანაში (ანუ $TK(x) = SAT(x)$) და, აქედან გამომდინარე, იქნება NPსრული, რაც მას P ≠ NPდაშვებით P სიმრავლიდან გამორიცხავს.

ახლა დავუშვათ, რომ $L \in$ NPC. მაშინ უნდა არსებობდეს დაყვანა R_i SAT ამოცანიდან L_{TK} ენაზე, რაც იმას ნიშნავს, რომ $TK(R_i(z)) = S(z), \forall z$. აქედან გამომდინარე, TF მანქანის მეორე ფაზაში $k = 2i+1$ შემთხვევისათვის შესაბამისი z არ მოიძებნება, რის შედეგადაც დაწყებული რაღაც ადგილიდან $f(n)$ არ გაიზრდება და დარჩება კენტი. აქედან გამომდინარე, L_{TK} ენა სასრული გამოვა, ანუ $L_{TK} \notin$ NPC.

ორივე დაშვებას ($L_{TK} \in$ P და $L_{TK} \in$ NPC) წინააღმდეგობამდე მივყავართ. აქედან გამომდინარე, L_{TK} „საშუალედო” უნდა იყოს, ანუ $L_{TK} \notin$ P და $L \notin$ NPC.

რ.დ.გ.

<u>სავარჯიშო 5.5:</u> დაამტკიცეთ, რომ ნებისმიერი სასრული ენა P სიმრავლეს ეკუთვნის.

<u>სავარჯიშო 5.6:</u> ზედა თეორემის დამტკიცების რომელ ადგილებში გამოვიყენეთ $P \neq$ NP?

P ≠ NP შემთხვევაში შუალედური ენების არსებობა ძალიან მნიშვნელოვანია პრაქტიკაშიც. მაგალითად, ნატურალურ რიცხვთა ფაქტორიზაციის ან გრაფთა იზომორფიზმის ამოცნისთვის ჯერ-ჯერობით ვერავინ მოიფიქრა პოლინომური ალგორითმი და ვერც მათი NP სრულობის დამტკიცება შეძლო. ამან გააჩინა საფუძვლიანი ეჭვი, რომ ეს ორი ამოცანა სწორედ ასეთ საშუალედო კლასს უნდა ეკუთვნოდეს, მაგრამ ეს მხოლოდ იმ პირობით, რომ P ≠ NP.

თავი 6

რთულ ამოცანათა ამოხსნის გზები

ალგორითმების შედგენისას უნდა გავითვალისწინოთ სამი ასპექტი (რა თქმა უნდა, ყველა ეს საკითხი აქ ჩამოყალიბებული სახით აღარ იქნება აქტუალური, თუ P = NP):

1. სისწრაფე: შედგენილი ალგორითმი ჩვენთვის დამაკმაყოფილებელი სისწრაფით უნდა მუშაობდეს
2. მონაცემთა სისრულე: ალგორითმი უნდა ითვლიდეს შედეგს მთელს განსაზღვრის არეზე
3. სიზუსტე: ყოველ მონაცემზე ალგორითმი ზუსტად იმ შედეგს უნდა იძლეოდეს, როგორიც მას მოეთხოვება

დღეისათვის არაა ცნობილი, შეიძლება თუ არა NP სრული ამოცანებისათვის სამივე პუნქტის ერთდროულად შესრულება (P vs. NP პრობლემა). ამიტომ ასეთი ამოცანებისათვის ეძებენ ისეთ ალგორითმებს, რომლებიც ერთ-ერთ პუნქტს არ ასრულებს:

პირველ რიგში უნდა განვსაზღვროთ, საჭიროა თუ არა ამოცანის ამოხსნა ყველა იმ მონაცემზე, რაც თეორიულად მოეთხოვება – ხშირად თეორიაში მოთხოვნილი ბევრი მონაცემი პრაქტიკაში სრულებით არ გვხვდება. მაგალითად, გრაფის შეღებვის ამოცანა ადვილად გადაიჭრება, თუ არ განვიხილავთ ნებისმიერ გრაფს და მხოლოდ პლანარულებით შემოვიფარგლებით: ამ შემთხვევაში საკმარისია 4 ფერი. ამას გარდა, არაორიენტირებულ გრაფებზე დასმული თითქმის ყველა ამოცანა ეფექტურად გადაიჭრება, თუ ნებისმიერი გრაფის მაგივრად განვიხილავთ ხეებს (აციკლურ გრაფებს). ამითი აიხსნება მატროიდებზე აგებულ ალგორითმთა ეფექტურობაც.

მიახლოებითი ალგორითმები: ყოველ მონაცემზე მიღებული შედეგი ზუსტი შედეგის δ სიახლოვეშია. ამის მაგალითად TSP ამოცანა შეიძლება მოვიყვანოთ: თუ, მაგალითად, ოპტიმალური განვლილი მანძილია 1000 კმ, პრაქტიკაში არ იქნება დიდი განსხვავება, თუ ამ საუკეთესო შედეგის ნაცვლად გვექნება გავლილი 1003 კმ, ან რაიმე სხვა ოპტიმალურთან ახლოს მყოფი მანძილი.
მეორე მაგალითია გრაფის წვეროებით გადაფარვის ამოცანა, რომლის მიახლოებით ალგორითმსაც ქვემოთ მოვიყვანთ.

ალგორითმი 6.1: გრაფის გადაფარვის მიახლოებითი ალგორითმი
მოცემულია: გრაფი $G = (V, E)$

```
1: W = ∅;
2: do
3:  შემთხვევით აირჩიე აქტუალური G გრაფის წიბო (e1, e2);
4:  W = W ∪ {e1, e2};
5:  G გრაფიდან ამოშალე e1 და e2 წვერო ყველა მიერთებული წიბოთი;
6: while(G არაცარიელია)
7: return(W)
```

მოცემული გრაფისათვის შემთხვევითი პრინციპით აირჩიე ერთი წიბო, ჩართე მისი ორივე წვერო საძიებელ სიმრავლეში და გრაფში წაშალე ყველა ის წიბო, რომელიც დაკავშირებულია ამ წვეროებთან. თუ ამ ოპერაციების შემდეგ მიღებული გრაფი ცარიელია, ალგორითმი დაასრულე. თუ არა და იგივე გაიმეორე.
ადვილი საჩვენებელია, რომ ამ ალგორითმის გაშვების შემდეგ მიღებული შედეგი მაქსიმუმ ორჯერ თუ აჭარბებს ოპტიმალურ შედეგს (დღეისათვის ეს ყველაზე ეფექტური ალგორითმია ამ ამოცანისათვის).

სავარჯიშო 6.1: აჩვენეთ, რომ ზემოთ მოყვანილი ალგორითმის მიერ გამოანგარიშებული გადაფარვის სიმრავლის ზომა მაქსიმუმ ორჯერ აჭარბებს ოპტიმალურ შედეგს.

6.1 რთულ ამოცანათა მიახლოებითი ამოხსნა

როგორც ვნახეთ, ამოცანები რამოდენიმე ძირითად კლასად შეიძლება დაიყოს: გამოთვლადი და არაგამოთვლადი, P და NP. არსებობს კიდევ მრავალი ამოცანათა კლასი, მაგალითად ისეთის, რომლის ამოხსნაც (გაპარალელების შედეგად) ლოგარითმულ დროში $O(\log n)$ შეიძლება (დახარისხება, არითმეტიკული ოპერაციები და ა.შ.).
ასეთ ამოცანათა კლასს „ნიკის კლასს" უწოდებენ და აღნიშნავენ როგორც NC. დღეისათვის ერთ-ერთი ძალიან მნიშვნელოვანი საკითხია, შესაძლებელია თუ არა P კლასში მყოფი ყველა ამოცანის ისე გაპარალელება, რომ მისი გამოთვლის დრო პოლინომიურიდან პოლი-ლოგარითმულზე დავიდეს, ანუ ზედა ზღვარი გახდეს $O(\log^k n)$ გავრცელებულია ეჭვი, რომ ორი ნატურალური რიცხვის უდიდესი საერთო გამყოფის ამოცანა არ ეკუთვნის NC კლასს: ევკლიდეს ალგორითმის პრინციპული გაუმჯობესება უკვე რამოდენიმე ათასი წელია ვერ მოხერხდა. აქედან გამომდინარე, ევკლიდეს ალგორითმის შესწავლა სირთულის თეორიის თვალსაზრისით ძალიან საინტერესო უნდა იყოს.

ბუნებრივად ისმის შეკითხვა, რაში გვეხმარება სირთულის თეორია? რას გვმატებს არადეტერმინისტული ავტომატების შექმნა, რომლის არამც თუ პრაქტიკული რეალიზაცია, არამედ გამოთვლის პროცესის გაგებაც კი შეუძლებელია?
ამ შეკითხვაზე შესაძლო პასუხი მათემატიკური ანალიზის პარალელით შეიძლება: რაში გვჭირდება ნამდვილ რიცხვთა სიმრავლე, თუ მის უდიდეს ნაწილს საერთოდ ვერც კი წარმოვიდგენთ? მაგრამ ამ რიცხვთა შემოღებამ საგრძნობლად გააადვილა კალკულუსის და, საერთოდ, მათემატიკის განვითარება, თეორემათა მტკიცება და რეალურ ცხოვრებაში არსებული პრობლემების კლასიფიკაცია. როგორც ცნობილია, ანალიზის აგება ტრანსცენდენტულ რიცხვთა გარეშეც შეიძლება, მაგრამ ამ შემთხვევაში თეორემათა მტკიცება წარმოუდგენლად ძნელდება. ასეა ინფორმატიკაშიც: არადეტერმინისტულ მანქანათა ჰიპოთეტური მოდელის შემოტანა თეორემათა მტკიცებაში, ამოცანათა კლასიფიკაციაში და სათანადო დასკვნების გამოტანაში გვიწყობს ხელს.

თუ გვხვდება ახალი ამოცანა, პირველ რიგში უნდა დავადგინოთ, შეიძლება თუ არა მისი ალგორითმულად ამოხსნა. თუ ამოხსნადია, უნდა დავადგინოთ, არის თუ არა იგი NP სრული. რადგან NP სრული ამოცანებისათვის არაა ცნობილი ეფექტური ამონახსნი და თეორიული შედეგებიც იმაზე მიგვითითებს, რომ ასეთი ალგორითმის პოვნა პრაქტიკაში ძალიან ძნელი ან შეუძლებელი უნდა იყოს, ამიტომ არ ეძებენ ასეთი ამოცანების ზუსტ ალგორითმებს (უხეშად რომ ვთქვათ, ძებნაში დროს არ კარგავენ). NP სრული ამოცანებისათვის სწრაფი ალგორითმების პოვნის უიმედობას აძლიერებს შემდეგი შედეგი:

არ არსებობს ისეთი პოლინომიური ალგორითმი, რომელიც ნებისმიერი არადეტერმინისტული სასრული ავტომატისათვის მის ექვივალენტურ მინიმალური მდგომარეობების მქონე დეტერმინისტულ სასრულ ავტომატს შეადგენს.

ერთად-ერთი იმედი არის ის, რომ ეს თეორემა ერთი რაიმე უნივერსალური მეთოდის არსებობას გამორიცხავს. ცალკეულ შემთხვევებში არაა გამორიცხული ოპტიმალური ზომის დეტერმინისტული სასრული ავტომატის შედგენა და თუ ეს ზომა პოლინომიური აღმოჩნდა, $P = NP$ დამტკიცება.

ალგორითმების შედგენისას უნდა გავითვალისწინოთ სამი ასპექტი (რა თქმა უნდა, ყველა ეს საკითხი აღარ იქნება აქტუალური, თუ $P = NP$):

1. სისწრაფე: შედგენილი ალგორითმი ჩვენთვის დამაკმაყოფილებელი სისწრაფით უნდა მუშაობდეს
2. მონაცემთა სისრულე: ალგორითმი უნდა ითვლიდეს შედეგს მთელს განსაზღვრის არეზე
3. სიზუსტე: ყოველ მონაცემზე ალგორითმი ზუსტად იმ შედეგს უნდა იძლეოდეს, როგორიც მას მოეთხოვება

დღეისათვის არაა ცნობილი, შეიძლება თუ არა NP სრული ამოცანებისათვის სამივე პუნქტის ერთდროულად შესრულება ($P = NP$ პრობლემა). ამიტომ ასეთი ამოცანებისათვის ეძებენ ისეთ ალგორითმებს, რომლებიც ერთ-ერთ პუნქტს არ ასრულებს:

პირველ რიგში უნდა განვსაზღვროთ, საჭიროა თუ არა ამოცანის ამოხსნა ყველა იმ მონაცემზე, რაც თეორიულად მოეთხოვება -- ხშირად თეორიაში მოთხოვნილი ბევრი მონაცემი პრაქტიკაში სრულებით არ გვხვდება. მაგალითად, გრაფის შეღებვის ამოცანა ადვილად გადაიჭრება, თუ არ განვიხილავთ ნებისმიერ გრაფს და მხოლოდ პლანარულებით შემოვიფარგლებით: ამ შემთხვევაში საკმარისია 4 ფერი. ამას გარდა, არაორიენტირებულ გრაფებზე დასმული თითქმის ყველა ამოცანა ეფექტურად გადაიჭრება, თუ ნებისმიერი გრაფის მაგივრად განვიხილავთ ხეებს (აციკლურ გრაფებს). ამითი აიხსნება მატროიდებზე აგებულ ალგორითმთა ეფექტურობაც.

მიახლოებითი ალგორითმები: ყოველ მონაცემზე მიღებული შედეგი ზუსტი შედეგის δ სიახლოვეშია. ამის მაგალითად გრაფის წიბოებით გადაფარვის ზემოთ ნახსენები ამოცანის ამოხსნა შეიძლება მოვიყვანოთ. მოცემული გრაფისათვის შემთხვევითი პრინციპით აირჩიე ერთი წიბო, ჩართე მისი ორივე წვერო საძიებელ სიმრავლეში და გრაფში წაშალე ყველა ის წიბო, რომელიც დაკავშირებულია ამ წვეროებთან. თუ ამ ოპერაციების შემდეგ მიღებული გრაფი ცარიელია, ალგორითმი დაასრულე. თუ არა და იგივე გაიმეორე.
ადვილი საჩვენებელია, რომ ამ ალგორითმის გაშვების შემდეგ მიღებული შედეგი მაქსიმუმ ორჯერ თუ აჭარბებს ოპტიმალურ შედეგს (დღეისათვის ეს ყველაზე ეფექტური ალგორითმია ამ ამოცანისათვის).

განმარტება 6.2: მოცემულია რაღაც ამოცანა, რომლის x მონაცემზე ოპტიმალური ამოხსნაა $Opt(x) \in \mathbb{R}$ და ალგორითმი A, რომლის შედეგი x მონაცემზე არის $A(x) \in \mathbb{R}$. იტყვიან, რომ ამ ალგორითმის შედეგი ამოცანის ოპტიმალური შედეგის $0 < \delta \leq 1$ სიახლოვეშია, თუ სრულდება შემდეგი უტოლობა:

$$\frac{|A(x) - Opt(x)|}{max\{A(x), Opt(x)\}} \leq \delta.$$

თუ რაიმე ამოცანას მოეძებნება ისეთი ალგორითმი, რომელიც ოპტიმალური შედეგის δ სიახლოვეშია, მას δ-მიახლოებადი ამოცანა ეწოდება.

ასეთი განსაზღვრებით გრაფების გადაფარვის ამოცანის ზემოთ მოყვანილი ალგორითმი ოპტიმალური შედეგის $\delta = \frac{1}{2}$ სიახლოვეშია.

გადაფარვის ამოცანა ე.წ. „ნაწილობრივ მიახლოვებად" ამოცანათა კლასში შედის: ისეთი ამოცანებისა, რომლებსაც ნებისმიერი სიზუსტით ვერ მივუახლოვდებით, მაგრამ გარკვეული სიზუსტით -- კი.

განმარტება 6.3: ამოცანა ნაწილობრივ მიახლოებადია, თუ წინასწარ ფიქსირებული $0 < \delta \leq 1$ რიცხვისთვის არსებობს δ-მიახლოებადი ალგორითმი.

არსებობს აგრეთვე ამოცანები, რომლისთვისაც შეიძლება ერთი რაიმე მიახლოებითი ალგორითმის დაწერა, რომელიც ნებისმიერი წინასწარ ფიქსირებული δ სიზუსტით მიუახლოვდება ოპტიმალურ შედეგს. ასეთ ამოცანებს სრულად მიახლოვებადი ეწოდება. სხვა სიტყვებით რომ ვთქვათ, დაიწერება ალგორითმი, რომელიც მონაცემად როგორც საწყისი ამოცანის მონაცემს, ასევე δ სიზუსტეს მიიღებს და შესაბამისი სიზუსტის პასუხს გამოითვლის.

განმარტება 6.4: ამოცანა სრულად მიახლოებადია, თუ არსებობს ალგორითმი $B(x, \delta)$, რომელიც ყოველ მონაცემზე ამ ამოცანის ოპტიმალური ამოხსნის δ სიახლოვეში მყოფ პასუხს გამოითვლის.

ასეთი სრულად მიახლოებადი ამოცანის მაგალითია ზურგჩანთის ამოცანა.

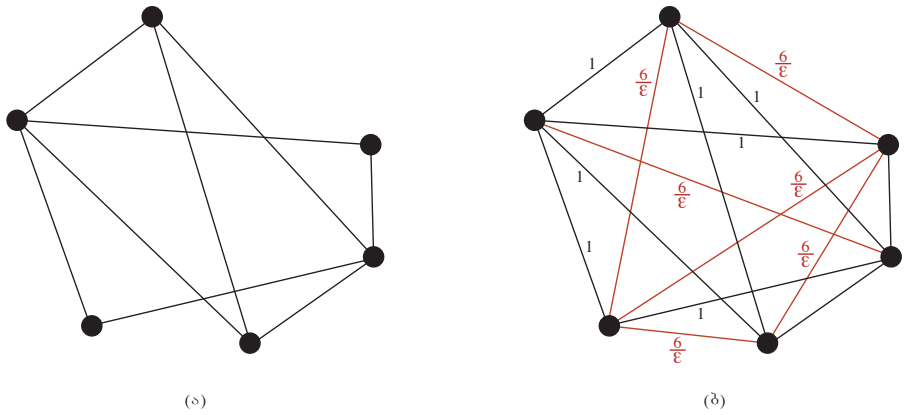
6.1.1 თეორემა TSP ამოცანის არამიახლოებადობის შესახებ და მისგან გამომდინარე შედეგები

დამტკიცებულია, რომ TSP ამოცანაში ასეთი მიახლოებები შეუძლებელია: ნებისმიერი $\delta > 0$ რიცხვისათვის მოიძებნება ისეთი კონკრეტული მონაცემი, რომლისთვისაც ნებისმიერი მიახლოებითი ალგორითმის შედეგი ზუსტი შედეგის δ მიდამოში არ იქნება, მაგრამ ამ ამოცანის კერძო შემთხვევების განხილვა ვითარებას ცვლის: თუ დამატებით მოვითხოვთ, რომ მოცემულ გრაფში სამკუთხედის უტოლობა სრულდებოდეს, ასეთი ამოცანა ნაწილობრივ მიახლოვებადი ხდება. თუ კიდევ დავადებთ შეზღუდვას, რომ გრაფი იყოს მოცემული ევკლიდურ სიბრტყეზე, მაშინ იგი სრულად მიახლოებადი ხდება.

თეორემა 6.5 თუ $P \neq NP$, მაშინ TSP არ არის მიახლოებადი პოლინომურ დროში.

დამტკიცება: დავამტკიცოთ, რომ NP სრულია ამოცანა „ϵ მიახლოებით გამოიანგარიშეთ TSP", რაც $P \neq NP$ პირობით თეორემას დაამტკიცებს.

ამ ამოცანაზე დავიყვანოთ ჰამილტონის ციკლის ამოცანა. თუ მოცემულია ნებისმიერი გრაფი $G = \{V, E\}$, შევადგინოთ სრული შეწონილი გრაფი $G' = \{V', E', w\}$, რომელშიც $V' = V$, $u, v \in V' \Rightarrow (u, v) \in E'$ და $w(u, v) = 1$, თუ $(u, v) \in E$, $w(u, v) = \frac{|V|}{\epsilon}$, თუ $(u, v) \notin E$ (ანუ ახლად შედგენილი სრული გრაფის ახალი წიბოების წონა მაღალია, ხოლო ძველ გრაფში არსებული წიბოების კი დაბალი, ნახ. 6.1).



ნახ. 6.1: *HC* და *TSP* ამოცანების გრაფები

ცხადია, თუ G გრაფში არსებობს ჰამილტონის ციკლი, G' გრაფზე ϵ სიზუსტით ამოხსნილი *TSP* მოგვცემს პასუხს A, რომელიც $|V|$ რიცხვის ϵ მიდამოში იქნება: $\frac{A-|V|}{A} \leq \epsilon$.
თუ G გრაფში ჰამილტონის ციკლი არ არსებობს, მაშინ ოპტიმალური მარშრუტი $Opt \geq |V| + \frac{|V|}{\epsilon}$. აქედან გამომდინარე, მისი ϵ სიზუსტით გამოთვლილი პასუხი $A \geq |V| + \frac{|V|}{\epsilon}$, რაც $|V|$ ამონახსნის ϵ მიდამოში ვეღარ იქნება.

ესე იგი, თუ G' გრაფზე ამოვხსნით *TSP* ამოცანას ϵ სიზუსტით, ცალსახად შევძლებთ იმის დადგენას, არსებობს თუ არა ჰამილტონის ციკლი G გრაფში და *TSP* ამოცანის ϵ სიზუსტით ამოხსნა *NP* სრული გამოდის.

რ.დ.გ.

ამრიგად, *NP* სრული ამოცანები სამ კლასად შეიძლება დაიყოს:

1. სრულად მიახლოვებად ამოცანათა კლასი — სამწუხაროდ ძალიან ვიწრო. მასში შედის, მაგალითად, დავალებათა შესრულების ამოცანა ორ პროცესორზე

2. ნაწილობრივ მიახლოვებად ამოცანათა კლასი — აგრეთვე მცირე, მაგრამ პირველზე უფრო ფართო, რომელშიც, მაგალითად, გრაფთა გადაფარვის ამოცანა შედის

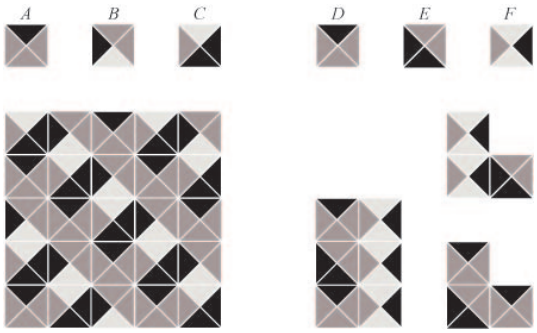
3. არამიახლოვებად ამოცანათა კლასი — ყველაზე ფართო კლასი, რომელშიც შედის TSP, გრაფებში დამოუკიდებელ წვეროთა სიმრავლის ამოცანა, სრული ქვეგრაფის გამოყოფის ამოცანა, გრაფის შეღებვის ამოცანა და სხვა.

აღსანიშნავია ის გარემოება, რომ არაამოხსნად ამოცანებშიც შეიძლება მონაცემთა სიმრავლის ან პირობის ისე შეზღუდვა, რომ იგი ამოხსნადად გადაიქცეს (როგორც, მაგალითად, TSP ამოცანაში მონაცემთა შეზღუდვისას იგი ნაწილობრივ მიახლოვებადი ან სრულად მიახლოვებადიც კი ხდებოდა).
საინტერესოა, რომ ასეთ შემთხვევებში შეცვლილი ამოცანები ხშირად *NP* სრული ხდება. ამის საილუსტრაციოდ განვიხილოთ „ფართობის დაფარვის" ამოცანა:

ფართობის გადაფარვის ამოცანა (Tiling Problem, TP)

მოცემულია n სხვადასხვა ტიპის ერთნაირი ზომის კვადრტული ფიგურა $a_1, a_2, ..., a_n$ (თითო ტიპის ფიგურის რაოდენობა უსასრულოა). მოცემულია აგრეთვე წესები, თუ რა ტიპის კვადრატების დადება შეიძლება ერთმანეთის გვერდით და რა ტიპისა -- ერთმანეთის ზემოთ და ქვემოთ. ამას გარდა, ფიგურათა დატრიალება არ შეიძლება. დაადგინეთ, შეიძლება თუ არა მოცემული კვადრატებითა და წესებით ნებისმიერად დიდი ფართობის მქონე კვადრატის აგება, თუ ამ კვადრატის მარცხენა ქვედა კუთხეში განთავსებული კვადრატული ფიგურის ტიპი წინასწარაა მოცემული. ნახ. 6.2-ში მარცხნივ ნაჩვენებია ამ ამოცანის ერთი მაგალითი: მოყვანილია სამი ტიპის კვადრატული ფიგურა A, B, C, რომელთა გვერდები რაღაცა ფერადაა შეღებილი. ორი ფიგურის ერთმანეთზე მიდება შეიძლება მაშინ და მხოლოდ მაშინ, თუ ამ ფიგურების შესაბამის გვერდებზე ფერები ერთნაირია.

ადვილი დასამტკიცებელია, რომ A, B, C ფიგურებითა და A ფიგურის მარცხენა ქვედა კუთხეში დადებით ნებისმიერი ფართობის კვადრატის აგება შეიძლება. მაგრამ თუ ავიღებთ D, E, F ფიგურებს, მაშინ შეიძლება მხოლოდ 2×2 კვადრატის აგება, ისიც იმ პირობით, თუ მარცხენა ქვედა კუთხეში D ფიგურას დავდებთ.



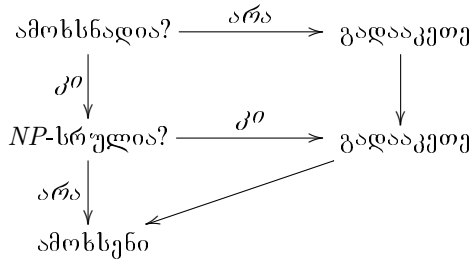
ნახ. 6.2: ფართობის დაფარვის ამოცანა

დამტკიცებულია, რომ ამ ამოცანისათვის ალგორითმული ამოხსნა არ არსებობს. მაგრამ თუ პირობას შევცვლით და ვიკითხავთ, შეიძლება თუ არა რაღაცა $k \times k$ ($k \in \mathbb{N}$) კვადრატის დაფარვა (უსასრულოდ დიდის მაგივრად), მაშინ ეს ამოცანა *NP* სრული ხდება.

საინტერესოა, რომ ამ ამოცანაზე ნებისმიერი სხვა ამოცანა დაიყვანება: ეს არის ალგორითმების ერთგვარი სიმულაცია: ნებისმიერი A ამოცანის X მონაცემის კოდირება შეიძლება ისეთ ფერად კვადრატებად, რომ მათი მეშვეობით მეოთხედი სიბრტყე დაიფარება მაშინ და მხოლოდ მაშინ, თუ $A(X) =$„კი”.

6.1.2 მოკლე დასკვნა ალგორითმების თეორიის თვალსაზრისით

ყოველივე ზემოთ თქმულიდან გამომდინარე, ამოცანათა ამოხსნის შემდეგი სტრატეგია არსებობს: ახალი ამოცანის დასმისას უნდა გავარკვიოთ, შეიძლება თუ არა მისი ალგორითმულად გადაჭრა. თუ ასეთი რამ შეუძლებელია, უნდა განვსაზღვროთ, რა დამატებითი შეზღუდვაა საჭირო მონაცემებსა და პირობაზე, რომ იგი ამოხსნადი გახდეს.



აღსანიშნავია, რომ ხშირ შემთხვევაში არაამოხსნადი ამოცანის გადაკეთების შემდეგ იგი *NP* სრული ხდება. დღევანდელი მიდგომით, თუ რაიმე ამოცანაზე დამტკიცდა, რომ იგი *NP* სრულია, მის ზუსტ ამოხსნაზე დროს აღარ კარგავენ და გადაჭრის ალტერნატიულ გზებს ეძებენ: უნდა განისაზღვროს, რა დამატებითი შეზღუდვაა საჭირო მონაცემებსა და პირობაზე, რომ იგი პოლინომურ დროში ამოხსნადი გახდეს, ან როგორ შეიძლება მისი ამონახსნის რაღაცა δ სიზუსტით გამოთვლა – არის თუ არა იგი სრულად მიახლოებადი, ნაწილობრივ მიახლოებადი ან არამიახლოებადი. ამას გარდა, შესაძლებელია ამოცანათა ალბათური მეთოდებით გადაჭრა: ისეთი ალგორითმის დაწერა, რომ პასუხი ***ხშირ შემთხვევაში*** სწორი იყოს. რა თქმა უნდა, ასე მიღებულ პასუხზე დარწმუნებით ვერ ვიტყვით, რომ დასმული ამოცანის ამოხსნას გვაძლევს, მაგრამ თუ მას ბევრჯერ გავუშვებთ და მიღებული პასუხებიდან ისეთს ავარჩევთ, რომელიც ყველაზე ხშირად გვხვდება, დიდი ალბათობით შეგვიძლია ვივარაუდოთ, რომ ეს ზუსტ ამოხსნას დაემთხვევა.

ამდაგვარი „ევრისტიკების” შედგენასა და ამოცანათა კლასიფიკაციაში გვეხმარება სირთულის თეორია, რომლის გარეშეც ასეთი რამ შეუძლებელი იქნებოდა.

თავი 7

რაისის თეორემა: სად გადის გამოთვლადობის ზღვარი?

რაისის თეორემა (ზოგიერთ წყაროში რაის-მიჰილ-შაპიროს თეორემა - Rice-Myhill-Shapiro theorem) გვეუბნება, რომ არ არსებობს ზოგადი მეთოდი, რომლითაც *ნებისმიერი* ალგორითმისთვის დავადგენთ, ითვლის თუ არა იგი რაიმე კონკრეტულ, წინასწარ განსაზღვრულ შედეგს (ანუ გამოითვლის თუ არა მოცემული პროგრამა მოცემულ ფუნქციას) გარდა ორი ტრივიალური მაგალითისა. ეს კი იმას ნიშნავს, რომ ყოველ ალგორითმს (ან ალგორითმთა კლასს) ცალკე მეთოდი უნდა მოვუძებნოთ, თუ როგორ გავაანალიზოთ მისი ფუნქცია.
გარდა იმისა, რომ ამ თეორემის შედეგად ძალიან ბევრი არაამოხსნადი ამოცანა აღმოაჩინეს, მისი შედეგები უფრო ღრამა და ერთობ ფილოსოფიურია: ადამიანის აზროვნება მნიშვნელოვან (გლობალურ) საკითხებს ვერ ჩაწვდება.

7.1 განსაზღვრება და ძირითადი შედეგები

განმარტება 7.1: ნებისმიერი A ალგორითმის მიერ განსაზღვრული ენა (მოკლედ ალგორითმის ენა) $L(A)$ ეწოდება სიმრავლეს $L(A) = \{w \mid A(w) =$„კი”$\}$.
განმარტება 7.2: მოცემული A ალგორითმის აღწერილობა ორობით ანბანზე აღვნიშნოთ როგორც $\langle A \rangle$.
$L_{Alg} = \{\langle A \rangle \mid A$ რეკურსიული ენის განმსაზღვრელი ალგორითმია$\}$ ყველა არსებული რეკურსიული ენის განმსაზღვრელი ალგორითმის აღწერილობათა სიმრავლეა.
თუ $\mathcal{C}$ *რეკურსიულ* ენათა რაღაც სიმრავლეა, $L_{\mathcal{C}} = \{\langle M \rangle \mid L(M) \in \mathcal{C}\}$ ყველა ისეთი ალგორითმის აღწერილობების სიმრავლეა, რომელიც $\mathcal{C}$ სიმრავლის რომელიმე ენას განსაზღვრავს.
ასეთ $L_{\mathcal{C}}$ სიმრავლეს (ენას) ეწოდება ***ტრივიალური ენა ალგორითმების შესახებ***, თუ $L_{\mathcal{C}} = \emptyset$ ან $L_{\mathcal{C}} = L_{Alg}$. წინააღმდეგ შემთხვევაში მას ალგორითმების შესახებ არატრივიალური ენა ეწოდება.
ანალოგიურად, $\mathcal{C}$ სიმრავლეს ***ენათა თვისების სიმრავლეს*** უწოდებენ. ენათა თვისების სიმრავლე ტრივიალურია, თუ იგი ყველა რეკურსიულ ენას შეიცავს, ან ცარიელია.

სხვა სიტყვებით რომ ვთქვათ, ენათა თვისების სიმრავლე $\mathcal{C}$ არატრივიალურია, თუ იგი არაცარიელია და არსებობს ისეთი რეკურსიული ენა, რომელიც მასში არ შედის (ანუ მოიძებნება ორი ენა L_1, L_2, რომელთაგან ერთი მასში შედის, მეორე კი – არა: $L_1 \in \mathcal{C}, L_2 \notin \mathcal{C}$).
ანალოგიურად, ალგორითმების შესახებ ენა $L_{\mathcal{C}}$ არატრივიალურია, თუ არსებობს ორი ისეთი ალგორითმი A, B, რომ $A \in L_{\mathcal{C}}$ და $B \notin L_{\mathcal{C}}$ და $\mathcal{C} \neq \emptyset$ (იგი ენათა თვისების არატრივიალურ სიმრავლეზეა განსაზღვრული).

სავარჯიშო 7.3: განიხილეთ ზემოთ მოყვანილი განსაზღვრება $L_{\mathcal{C}} = \{\langle M \rangle \mid L(M) \in \mathcal{C}\}$. რისი ტოლი უნდა იყოს $\mathcal{C}$, რომ $L_{\mathcal{C}} = L_{Alg}$?

მაგალითი 7.4: არატრივიალური ენებია
$L_\emptyset = \{\langle M \rangle \mid L(M) = \{\emptyset\}$ (ცარიელი ენა)$\}$
$L_\Sigma = \{\langle M \rangle \mid L(M) = \Sigma^*$ (მოცემულ Σ ანბანზე შედგენილ ყველა სიტყვათა სიმრავლე)$\}$
$L_\epsilon = \{\langle M \rangle \mid \epsilon \in L(M)\}$

სავარჯიშო 7.5: დაამტკიცეთ ზემოთ მოყვანილ ენათა არატრივიალურობა.

თეორემა 7.6 (რაისის თეორემა) ნებისმიერი $L_{\mathcal{S}}$ არატრივიალური ენა ალგორითმების შესახებ არ არის გამოთვლადი (არ არსებობს ისეთი ალგორითმი A_S, რომლისთვისაც $A_S(x) =$„კი"$\Leftrightarrow x \in L_{\mathcal{S}}$, ანუ მოცემული მონაცემისთვის ვერ განვსაზღვრავთ, არის თუ არა იგი ისეთი ალგორითმის აღწერილობა, რომელიც $\mathcal{S}$ სიმრავლეში შემავალ რომელიმე ენას განსაზღვრავს).

რაისის თეორემით მტკიცდება, რომ თუ მოცემულია რაიმე სიტყვა, მასზე მხოლოდ იმის დადგენა შეგვიძლია, არის თუ არა იგი რაიმე ალგორითმის აღწერა. იმის დადგენა, თუ რას გამოითვლის მოცემული ალგორითმი ან გამოითვლის თუ არა იგი რაიმე კონკრეტულ ფუნქციას, შეუძლებელია.

დამტკიცება: რაისის თეორემის დამტკიცების ძირითადი იდეა შეჩერების ამოცანის დაყვანაა ალგორითმების შესახებ არატრივიალური ენის განსაზღვრის ამოცანაზე: თუ მოცემულია $\langle M, w\rangle$ რაიმე ალგორითმის და მისი მონაცემის აღწერა, ავაგებთ ისეთ ალგორითმს M', რომ ($M(w)$ შეჩერდება) $\Leftrightarrow$ ($L(M') \in \mathcal{S}$). ეს კი იმას ნიშნავს, რომ თუ არსებობს რაიმე მეთოდი, რომლითაც დავადგენთ $L(M') \in \mathcal{S}$ ჭეშმარიტებას ან მცდარობას, შეჩერების ამოცანაც გადაჭრილი გვექნება.
რადგან $\mathcal{S}$ არატრივიალურია, უნდა არსებობდეს ორი ენა $L_1 \in \mathcal{S}$ და $L_2 \notin \mathcal{S}$. ზოგადობის შეუზღუდავად შეგვიძლია ჩავთვალოთ, რომ $L_2 = \emptyset$, რადგან თუ $\emptyset \in \mathcal{S}$, არატრივიალურ კლასად შეგვიძლია გამოვაცხადოთ $\overline{\mathcal{S}}$.

სავარჯიშო 7.7: დაამტკიცეთ, რომ თუ S ენა არატრივიალურია, ასევე არატრივიალური იქნება $\overline{S}$.

განვიხილოთ ისეთი ალგორითმი M_L, რომ $L(M_L) = L_1$. მისი და $\langle M, w\rangle$ აღწერილობის გამოყენებით შევქმნათ ახალი ალგორითმი M':

ალგორითმი 7.1: $\mathcal{S}$ არატრივიალური ენის განსაზღვრის ალგორითმი M' ($\langle M, w\rangle$ წყვილის გათვალისწინებით)
მოცემულია: სიტყვა $x \in \mathbb{B}^*$

1: მოახდინე M მანქანის სიმულაცია w მონაცემზე;
2: თუ M შეჩერდება w სიტყვაზე, მოახდინე M_L მანქანის სიმულაცია x მონაცემზე;
3: თუ M_L მიიღებს x სიტყვას, გამოიტანე პასუხი „კი" (მიიღე x სიტყვა)

შენიშვნა: მოცემულ აღწერილობაში მნიშვნელოვანია, რომ თუ M ალგორითმი შეჩერდება w სიტყვაზე და ამავდროულად M_L ალგორითმიც სასრულ დროში მიიღებს x სიტყვას, მაშინ M' ალგორითმი მიიღებს სიტყვას x. წინააღმდეგ შემთხვევაში მისი გამოთვლის პროცესი უსასრულოდ გაგრძელდება და x სიტყვა არ იქნება მიღებული.¥

განვიხილოთ შემთხვევა, როდესაც M ალგორითმი w სიტყვაზე შეჩერდება. მაშინ M' მიიღებს სიტყვას x მაშინ და მხოლოდ მაშინ, თუ M_L მიიღებს სიტყვას x. აქედან გამომდინარე, $L(M') = L_1 \in \mathcal{S}$.
მაგრამ თუ M ალგორითმი w სიტყვაზე არ შეჩერდება, მაშინ M' ვერც ერთ სიტყვას ვერ მიიღებს, რადგან M_L მანქანის სიმულაცია არ მოხდება და, აქედან გამომდინარე, $L(M') = \emptyset \notin \mathcal{S}$ (ანუ იმ სიტყვათა სიმრავლე, რომელზედაც პასუხია „კი", ცარიელია).

ზემოთ თქმულიდან ადვილად შეიძლება დავინახოთ, რომ თუ M' მანქანა $\mathcal{S}$ სიმრავლის ენას განსაზღვრავს, მაშინ M ალგორითმი შეჩერებულა w მონაცემზე და თუ M' $\overline{\mathcal{S}}$ სიმრავლის ენას (ამ შემთხვევაში ცარიელ ენას) განსაზღვრავს, M მანქანა w მონაცემზე არ შეჩერებულა. აქედან გამომდინარე, ალგორითმებზე არატრივიალური ენის განსაზღვრა რომ ყოფილიყო შესაძლებელი, ზემოთ აღწერილი მეთოდებით M მანქანიდან M' მანქანას შევქმნიდით და $L(M') \in S$ საკითხის ამოხსნით M მანქანის შეჩერების საკითხიც ამოხსნადი იქნებოდა, რითაც წინააღმდეგობას ვიღებთ.

რ.დ.გ.

რაისის თეორემის შედეგები: შემდეგი ამოცანები არაამოხსნადია:

- მოცემულია ორი ალგორითმი M_1 და M_2. $L(M_1) = L(M_2)$? შენიშვნა: ეს ე.წ. რაისის გაფართოებული თეორემის შედეგია, რომელიც შემდგომში იქნება ჩამოყალიბებული

- მოცემულია ორი ფორმალური გრამატიკა G_1 და G_2. $L(G_1) = L(G_2)$? შენიშვნა: იგივე, რაც ზემოთ
- მოცემული ალგორითმისთვის გაარკვიეთ, თუ რომელ ენას გამოითვლის იგი
- ექნება მოცემული ალგორითმის მიერ გამოთვლილ ფუნქციას რაიმე მონაცემზე მნიშვნელობა 1?
- გამოითვლის თუ არა მოცემული ალგორითმი ერთსა და იმავე შედეგს ყოველ მონაცემზე?
- მიიღებს თუ არა მოცემული ალგორითმი ყველა მონაცემს?
- მოცემული L ენისთვის დაადგინეთ, განსაზღვრავს თუ არა მას რომელიმე ალგორითმი (ანუ გამოთვლადია თუ არა ეს ენა)

სავარჯიშო 7.8: დაამტკიცეთ ზემოთ მოყვანილი ამოცანების არაამოხსნადობა.

მაგალითი 7.9: განვიხილოთ შემდეგი ამოცანა: მოცემულია ალგორითმი M. არის თუ არა $L(M)$ პალინდრომების ენა ($w \in L(M) \Leftrightarrow w = w^R$)?
რაისის თეორემის გამოსაყენებლად ეს საკითხი შესაბამის ტერმინებში უნდა ჩამოვაყალიბოთ. პირველ რიგში ჩამოვაყალიბოთ პალინდრომული ენა

$$Pal = \{\langle M\rangle \,|L(M) \text{ შეიცავს ყველა პალინდრომს}\}$$

ცხადია, რომ ეს ენა არაა ტრივიალური და რაისის თეორემის თანახმად არაამოხსნადი უნდა იყოს.

რ.დ.გ.

სავარჯიშო 7.10: დაამტკიცეთ, რომ ზემოთ მოყვანილი ენა არაა ტრივიალური (მინიშნება: მოიყვანეთ ორი ალგორითმის მაგალითი. ერთი იღებს მხოლოდ პალინდრომებს და მეორე ისეთ სიტყვასაც, რომელიც პალინდრომი არ არის).

მაგალითი 7.11: რაისის თეორემის მეშვეობით იმის ჩვენებაც შეიძლება, რომ ამოცანათა სირთულის ავტომატური მტკიცება შეუძლებელია. განვიხილოთ ენათა შემდეგი კლასი:

$$P = \{L|L \text{ ენა ამოიხსნება რაიმე (დეტერმინისტული) ალგორითმის მიერ პოლინომურ დროში}\}$$

შენიშვნა: რას ნიშნავს „დეტერმინისტული" და „არადეტერმინისტული" ალგორითმი, ამას შემდგომში განვსაზღვრავთ. ამ ეტაპზე ჩავთვალოთ, რომ იგულისხმება იმ ტიპის ალგორითმი, რომლებსაც ჩვენ განვიხილავთ).

სავარჯიშო 7.12: ჩაწერეთ ამ ამოცანის პირობა რაისის თეორემის ტერმინებში და დაამტკიცეთ P კლასის არაამოხსნადობა.

აქედან გამომდინარე, არაამოხსნადია შემდეგი ამოცანა:

მოცემულია ალგორითმი M. შეიძლება თუ არა $L(M)$ ენის პოლინომურ დროში განსაზღვრა?

7.2 გასათვალისწინებელი პრობლემები

მნიშვნელოვანი შენიშვნა: რაისის თეორემის შედეგად, არ არსებობს ისეთი *ზოგადი* ალგორითმი, რომელიც *ნებისმიერი* (M, L) ალგორითმისა და ენის წყვილისთვის განგვისაზღვრავდა, განსაზღვრავს თუ არა M ალგორითმი L ენას. ეს არ ნიშნავს იმას, რომ კონკრეტული M', L' წყვილისთვის ასეთი რამის დადგენა შეუძლებელია. ზოგ შემთხვევაში ეს შესაძლებელია, თუმცა (ფაქტიურად) ყველა წყვილისთვის ცალკე ალგორითმის (მეთოდის) შექმნა შეიძლება გახდეს საჭირო.

ამას გარდა, შესაძლებელია ისეთი ამოცანების წამოჭრა, რომლებსაც რაისის თეორემა არ ესადაგება, თუმცა ამის დანახვა მარტივი არაა. ასეთ შემთხვევებში მიღებული მტკიცება ან მთლიანად არასწორია, ან გარკვეულ უზუსტობებს შეიცავს. ამდაგვარი პრობლემების თავიდან ასაცილებლად განვიხილოთ რამოდენიმე კონკრეტული მაგალითი.

განმარტება 7.13: ალგორითმის ბრძანებას, რომლის შედეგადაც მისი გამოთვლა დასრულდება, ვუწოდოთ „საბოლოო" (ან „სამიზნე"), ხოლო ყველა დანარჩენ ბრძანებას – შუალედური.

მაგალითი 7.14: განვიხილოთ შემდეგი გადაწყვეტილების ამოცანა:
მოცემული M ალგორითმისთვის დაადგინეთ, არსებობს თუ არა ისეთი მონაცემი w, რომლითაც M გამოთვლის პროცესში ყველა შუალედურ ბრძანებას ერთხელ მაინც შეასრულებს.

შეიძლება ვიფიქროთ, რომ თვისება „რომელიმე w მონაცემისთვის M ალგორითმი გამოთვლის პროცესში ყველა შუალედურ ბრძანებას ერთხელ მაინც შეასრულებს" არაა ტრივიალური და რაისის თეორემის თანახმად ეს ამოცანა არ უნდა იყოს გამოთვლადი.
იმის მიუხედავად, რომ ეს ამოცანა მართლაც არაა ამოხსნადი, ეს ფაქტი რაისის თეორემის პირობებში ვერ ჯდება: თვით თვისების კონცეფცია აქ არასწორადაა გამოყენებული.
ეს გადაწყვეტილების ამოცანა შეიძლება შემდეგი ენის სახით ჩამოვაყალიბოთ:

$$T_{TM} = \left\{ \langle M \rangle \;\middle|\; \begin{array}{l} \text{რომელიმე } w \text{ მონაცემისთვის } M \text{ ალგორითმი} \\ \text{გამოთვლის პროცესში ყველა შუალედურ} \\ \text{ბრძანებას ერთხელ მაინც შეასრულებს} \end{array} \right\}$$

ზემოთ ნახსენები გამონათქვამი არ აღწერს ენათა კლასს ტიურინგის მანქანათა შესახებ, რაც რაისის თეორემის პირობის აუცილებელი ნაწილია; T_{TM} უფრო ტიურინგის მანქანის კონკრეტული თვისების გამომხატველია. სხვა სიტყვებით რომ ვთქვათ, ლაპარაკია არა ენათა კლასზე, არამედ ალგორითმების კონკრეტულ კლასზე: ამ სიმრავლეში ალგორითმები არაა რაღაცა ენის თვისებების კრიტერიუმებით შერჩეული.
ჩვენ იმის დამტკიცებაც შეგვიძლია, რომ ენათა კლასი, რომელიც ამ თვისებას გამოხატავს, არ არსებობს. ამისათვის საკმარისია ორი M_1 და M_2 ალგორითმის პოვნა, რომლისთვისაც $L(M_1) = L(M_2)$, მაგრამ $\langle M_1 \rangle \in T_{TM}$ და $\langle M_2 \rangle \notin T_{TM}$.

	ალგ. M_1		ალგ. M_2
1:	$n = 5$;		$n = 5$;
2:	$w =$ read();	/* წაიკითხე სიმბოლო */	$w = read()$;
3:	if($w == '1'$)		if($w == '1'$)
4:	then goto [2];		then goto [2];
5:	else if($w == '\#'$)		else if($w == '\#'$)
6:	then $k = n + 1$;		then return(„კი")
7:	return(„კი")		else return(„არა")
8:	else $k = n * 2$;		
9:	return(„არა")		

$L(M_1) = L(M_2)$, მაგრამ $\langle M_1 \rangle \notin T_{TM}$ და $\langle M_2 \rangle \in T_{TM}$
სავარჯიშო 7.15: დაამტკიცეთ, რომ ნახ. 7.2-ში მოყვანილი ალგორითმებისთვის $L(M_1) = L(M_2)$, მაგრამ $\langle M_1 \rangle \notin T_{TM}$ და $\langle M_2 \rangle \in T_{TM}$.

სავარჯიშო 7.16: დაამტკიცეთ T_{TM} ენის არაამოხსნადობა მასზე შეჩერების ამოცანის დაყვანით.

მაგალითი 7.17: მოცემულია ორი ალგორითმი M_1 და M_2. $L(M_1) = L(M_2)$?
არც აქ შეიძლება რაისის თეორემის გამოყენება იმ სახით, რომლითაც ის ზემოთ იქნა ჩამოყალიბებული, რადგან აქ ორ ენაზეა ლაპარაკი. ამ ამოცანის გადასაჭრელად რაისის თეორემის განვრცობაა საჭირო.

განმარტება 7.18: ალგორითმბის მიერ ამოცნობად (ანუ რეკურსიულ) ენათა წყვილების სიმრავლეს

$$S_2 = \{(L_1, L_2) | L_1, L_2 \text{ რეკურსიული ენებია}\}$$

ენათა წყვილების თვისება ეწოდება. S_2 თვისება არატრივიალურია, თუ არსებობს ამოცნობად ენათა ისეთი ორი წყვილი (L_1, L_2) და (L_3, L_4), რომ $(L_1, L_2) \in S_2$ და $(L_3, L_4) \notin S_2$.
თეორემა 7.19 (რაისის გაფართოებული თეორემა ენათა წყვილებისთვის) თუ ენათა წყვილების თვისება S_2 არატრივიალურია, მაშინ

$$L_S = \{\langle M_1, M_2 \rangle | (L(M_1), L(M_2)) \in S_2\}$$

ენა არაამოხსნადია.

სავარჯიშო 7.20: დაამტკიცეთ ზემოთ მოყვანილი თეორემა.

სავარჯიშო 7.21: გამოიყენეთ ზემოთ ჩამოყალიბებული თეორემა იმის დასამტკიცებლად, რომ M_1 და M_2 ალგორითმებისთვის $L(M_1) = L(M_2)$ დადგენა შეუძლებელია.

თავი 8

არადეტერმინისტული ალგორითმები

აქამდე ჩვენ მხოლოდ ისეთ ალგორითმებს განვიხილავდით, რომელშიც შემდეგი ნაბიჯი ცალსახადაა განსაზღვრული აქამდე ჩატარებული ნაბიჯებისა და ცვლადების მნიშვნელობებით. მაგალითად, თუ ალგორითმში არსებობს ფრაგმანტი

$a = b + c;$
$d = a * 2;$

მაშინ წინასწარ შეგვიძლია ვთქვათ, რომ $a = b + c$ ბრძანების შემდეგ შესრულდება ბრძანება $d = a * 2$. ასევე, თუ ალგორითმში არსებობს ფრაგმენტი

$if (a > b + c)$
$then \;\; d = a * 2;$
$else \;\; d = a/2;$

ცვლადების (ამ შემთხვევაში a, b, c) მნიშვნელობებით ცალსახადაა განსაზღვრული, თუ რომელი ბრძანება ($d = a * 2$ თუ $d = a/2$) შესრულდება.

ასეთ ალგორითმებს ***დეტერმინისტული*** ეწოდება („დეტერმინიზებული" = „განსაზღვრული").

თეორიულ ინფორმატიკაში შემუშავებულია ეგრეთ წოდებული „არადეტერმინისტული" ალგორითმების იდეა, რომელსაც შემდგომში განვიხილავთ.

8.1 არადეტერმინისტული ალგორითმის იდეა

განვიხილოთ ისეთი ალგორითმი, რომელიც შემდეგ ფრაგმენტს შეიცავს:

$a = b + c;$
$d = a * 2$ ან $d = a/2$

რა თქმა უნდა, ანალოგიური განშტოებები მრავლად შეიძლება გვხვდებოდეს ალგორითმში, რაც განშტოებების განშტოებებს, მათ განშტოებებს და ა.შ. შეიძლება იწვევდეს.

ლოგიკურია შემდეგი შეკითხვა: როგორ უნდა ირჩევდეს ალგორითმი, რომელი განშტოებით გააგრძელოს გამოთვლა? საქმე ისაა, რომ ეს მხოლოდ თეორიული მოდელია: პრაქტიკაში მისი რეალიზაცია არ არსებობს. ამიტომ უნდა ჩავთვალოთ, რომ თუ განშტოებათა ერთი გზით წასვლა სწორ პასუხამდე მიგვიყვანს, ეს ალგორითმიც სწორედ ამ გზით წავიდოდა და სწორ პასუხს მოგვცემდა.

ყოველივე ზემოთ თქმულის საილუსტრაციოდ განვიხილოთ რამოდენიმე მაგალითი.

მაგალითი 8.1: განვიხილოთ დაულაგებელ სიმრავლეში ძებნის ამოცანა: მოცემულია სასრული სიმრავლე $A = \{a_1, a_2, ..., a_k\}$ და რაიმე რიცხვი b. განსაზღვრეთ, ჭეშმარიტია თუ არა $b \in A$.
ამის არადეტერმინისტული ალგორითმი იქნება:

$x = a_1$ ან $x = a_2$ ან ... ან $x = a_k$
return$(x == b)$

ცხადია, რომ თუ $b \in A$, არადეტერმინისტული ალგორითმი აირჩევს ზუსტად ამ რიცხვს და ბიჯების რაოდენობა არ იქნება დამოკიდებული საწყისი A სიმრავლის ზომაზე, ანუ არადეტერმინისტული ალგორითმით დაულაგებელ სიმრავლეში ელემენტის ძებნა შესაძლებელია დროში $O(\log b)$.
იმის მაგივრად, რომ ზედა ალგორითმის პირველ სტრიქონში დაგვეწერა $x = a_1$ ან $x = a_2$ ან ... ან $x = a_k$, სიმარტივისთვის შეგვიძლია დავწეროთ:

არადეტერმინისტულად აირჩიე $x \in A$;
return$(x == b)$

და, რადგან არადეტერმინისტულად ამორჩევას შემდგომშიც ხშირად გამოვიყენებთ, მისთვის შემოვიღოთ აღნიშვნა:

არადეტერმინისტულად აირჩიე ერთ-ერთი ლელემენტი A სიმრავლიდან $\Leftrightarrow$ NDPick(A).

ცხადია, რომ ალგორითმი

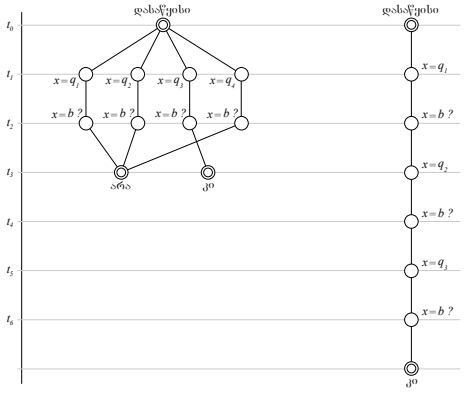
$x =$ NDPick(A)
return$(x == b)$

მოგვცემს პასუხს „კი" მაშინ და მხოლოდ მაშინ, თუ $b \in A$.
ამ მსჯელობიდან გამომდინარე ადვილია ისეთი არადეტერმინისტული ალგორითმის შექმნა, რომელიც პასუხად პოლინომურ დროში ჰამილტონის ციკლის (ან გზის) ამოცანის სერტიფიკატს მოგვცემს მისი არსებობის შემთხვევაში.

სავარჯიშო 8.2: დაწერეთ არადეტერმინისტული ალგორითმი, რომელიც მოცემული გრაფისთვის ჰამილტონის გზის სერტიფიკატს შექმნის მისი არსებობის შემთხვევაში.

არადეტერმინისტული და დეტერმინისტული ალგორითმების გამოთვლის სქემა გრაფიკულად გამოვხატოთ შემდეგ მაგალითზე.

მაგალითი 8.3: მოცემულია ოთხ რიცხვიანი სიმრავლე $\{q_1, q_2, q_3, q_4\}$ და რიცხვი $b = q_3$. დაადგინეთ, არის თუ არა b მოცემულ სიმრავლეში.
ამ ამოცანის დეტერმინისტული ალგორიძმი რიგ-რიგობით ამოწმებს ელემენტებს და როდესაც მიაგნებს $q_3 = b$ ელემენტს, შეჩერდება და გვეტყვის პასუხს „კი" (ნახ. 8.1 მარჯვნივ).



არადეტერმინისტული და დეტერმინისტული ალგორითმების გამოთვლის დიაგრამები

დეტერმინისტულისაგან განსხვავებით, არადეტერმინისტული ალგორითმის მუშაობის პროცესი შეიძლება გამოვსახოთ იგივე ნახაზში მარცხნივ მოყვანილი დიაგრამით.

ეს დიაგრამა შეგვიძლია განვიხილოთ, როგორც მიმართული გრაფი, რომლის წიბოები მიმართულია ზემოდან ქვემოთ. აღსანიშნავია, რომ არადეტერმინისტული ალგორითმის გამოთვლის დიაგრამაში შეიძლება არსებობდეს (და უმეტეს შემთხვევაში არსებობს კიდეც) განშტოებები, რომლებიც „ერთ დონეზე" გამოთვლის მაჩვენებელია (ანუ საწყისი წვეროდან ერთი და იმავე მანძილითაა დაშორებული წვეროები). ამ მიმართული გრაფის (ხის) რომელიმე (შესაძლოა არაერთ) გზას შეიძლება მივყავდეთ პასუხამდე „კი". ამ შემთხვევაში იტყვიან, რომ მოცემული არადეტერმინისტული ალგორითმი მონაცემს მიიღებს.
თუ ასეთი გზა არაერთია, მაშინ გამოთვლის ბიჯების რაოდენობა იქნება იმ მინიმალური გზს სიგრძე, რომელსაც დასაწყისიდან „კი" პასუხამდე მივყავართ.

ზოგადად გვაქვს შემდეგი განმარტება:

განმარტება 8.4: გამოთვლის დიაგრამის მიმართულ გრაფში დასაწყისიდან „კი" ან „არა" პასუხამდე მიმავალ გზას ამ დიაგრამის ***გამოთვლის გზა*** ეწოდება.

ახლა კი ვნახოთ, თუ როგორ შეიძლება რაიმე NPსრული ამოცანის ამოხსნა არადეტერმინისტული ალგორითმებით.
თუ დავუშვებთ, რომ მოცემულ $G = (V, E)$ გრაფის v წვეროსთვის $\mathrm{Nbr}(v, G)$ ფუნქციით ამ წვეროს მეზობლების სიმრავლის გამოთვლა შეიძლება, მაშინ ჰამილტონის გზის ამოცანის არადეტერმინისტული ალგორითმი შემდეგნაირად შეიძლება აღიწეროს:

NDHP(G)
w_1 =NDPick(V);
for($i = 2, i \leq |V|, i++$)
w_i =NDPick(Nbr(w_{i-1}));
$V_{HP}(G, (w_1, ..., w_{|V|}))$

სავარჯიშო 8.5: დაამტკიცეთ, რომ ზემოთ მოყვანილი ალგორითმი სწორად ამოხსნის ჰამილტონის ციკლის ამოცანას.

სავარჯიშო 8.6: გრაფის როგორი აღწერილობა იქნება ყველაზე ხელსაყრელი ზემოთ მოყვანილ ალგორითმში?

სავარჯიშო 8.7: ზემოთ მოყვანილი ალგორითმისთვის დახაზეთ შემდეგი ორი გრაფის გამოთვლის დიაგრამები:
$G_1 = (V_1, E_1)$, $V_1 = \{v_1, v_2, v_3, v_4\}$, $E_1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_2)\}$,
$G_2 = (V_2, E_2)$, $V_2 = \{v_1, v_2, v_3, v_4, v_5\}$, $E_2 = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_2), (v_2, v_5)\}$.

სავარჯიშო 8.8: რამდენი გამოთვლის გზა ექნება წინა სავარჯიშოში მოყვანილ დიაგრამებს? მათგან რამდენი მიდის „კი" პასუხამდე?

სავარჯიშო 8.9: შეაფასეთ ზემოთ განხილული ალგორითმის n წვეროიან გრაფზე გამოთვლისას წარმოქმნილი გზების რაოდენობის ზედა ზღვარი.

სავარჯიშო 8.10: გამოითვალეთ ზემოთ მოყვანილი NDHP ალგორითმის დროის ზედა ზღვარი.

8.2 NP კლასი არადეტერმინისტული ალგორითმების თვალსაზრისით

როგორც ვნახეთ, არადეტერმინისტული ალგორითმებით NPსრული ამოცანების ამოხსნა შეიძლება პოლინომურ დროში.

სავარჯიშო 8.11: დაამტკიცეთ, რომ არადეტერმინისტული ალგორითმებით NPკლასის ნებისმიერი ამოცანის ამოხსნა შეიძლება პოლინომურ დროში.
სავარჯიშო 8.12: დაამტკიცეთ, რომ თუ რაიმე ამოცანა ამოიხსნება არადეტერმინისტული ალგორითმებით პოლინომურ დროში, მისთვის იარსებებს პოლინომური სერტიფიკატი.

ამ სავარჯიშოთი დავამტკიცეთ მნიშვნელოვანი თეორემა:

თეორემა 8.13 NP=$\{L|L$ ამოცანა ამოიხსნება არადეტერმინისტული ალგორითმით პოლინომურ დროში$\}$

აქედან გამომდინარეობს ამ კლასის სახელწოდებაც: **N**ondeterministic **P**olynomial.

სავარჯიშო 8.14: დაწერეთ VC_k ამოცანის არადეტერმინისტული ალგორითმი.
დაამტკიცეთ მისი სისწორე და შეაფასეთ მისი ბიჯების რაოდენობის ზედა ზღვარი.
სავარჯიშო 8.15: დაწერეთ $3SAT$ ამოცანის არადეტერმინისტული ალგორითმი.
დაამტკიცეთ მისი სისწორე და შეაფასეთ მისი ბიჯების რაოდენობის ზედა ზღვარი.
სავარჯიშო 8.16: დაწერეთ SAT ამოცანის არადეტერმინისტული ალგორითმი.
დაამტკიცეთ მისი სისწორე და შეაფასეთ მისი ბიჯების რაოდენობის ზედა ზღვარი.
რითი განსხვავდება ეს ალგორითმი წინა სავარჯიშოში მოყვანილი $3SAT$ ალგორითმისგან?

ყოველივე ზემოთ თქმულიდან გამომდინარე ადვილია შემდეგი თეორემის დამტკიცება:

თეორემა 8.17 ამოცანა სერტიფიცირებადია პოლინომურ დროში მაშინ და მხოლოდ მაშინ, თუ არსებობს არადეტერმინისტული ალგორითმი, რომელიც მას პოლინომურ დროში ამოხსნის.

სავარჯიშო 8.18: დაამტკიცეთ ზემოთ მოყვანილი თეორემა.

8.2.1 არადეტერმინისტული ალგორითმების დეტერმინიზაცია

როგორც აღვნიშნეთ, არადეტერმინისტულ და დეტერმინისტულ ალგორითმებს შორის ძირითადი განსხვავება „განშტოებებია": X ან Y, სადაც X და Y თავის თავად გადაწყვეტილების ამოცანების ქვეალგორითმებად შეიძლება განვიხილოთ (თანაც ამდაგვარი განშტოებები თვით X და Y ნაწილებშიც რეკურსიულად შეიძლება გვხვდებოდეს).
ცხადია, თუ რიგ-რიგობით ჩავატარებთ ჯერ X და შემდეგ Y ალგორითმს, საბოლოო პასუხი შეგვიძლია შემდეგნაირად მივიღოთ: $X \vee Y$.
ამაში მდგომარეობს დეტერმინიზაციის ძირითადი იდეა: ყოველი განშტოება გამოითვალოს ცალ-ცალკე და შემდეგ პასუხების დიზიუნქცია ავიღოთ.
ცხადია, რომ ყველა განშტოების ამ მეთოდით გამოთვლა უარეს შემთხვევაში რესურსების ექსპონენციურ ზრდას გამოიწვევს.

სავარჯიშო 8.19: დაამტკიცეთ, რომ თუ არადეტერმინისტული ალგორითმი $O(f(n))$ დროში მუშაობს, მისი დეტერმინიზაციის შედეგად დროის ზედა ზღვარი მიმდევრობით ალგორითმში იქნება $O(2^{f(n)})$.

სავარჯიშო 8.20: რამდენით გაიზრდება გამოყენებული მეხსიერების რაოდენობა? პასუხი დაამტკიცეთ.

სავარჯიშო 8.21: დაამტკიცეთ, რომ დეტერმინიზაციის ზემოთ მოყვანილი მეთოდით პროცესორების ფიქსირებული რაოდენობის მანქანაზე პარალელიზაციის შედეგად მუშაობის დროის ზედა ზღვარი ასევე ექსპონენციური იქნება.

აღსანიშნავია, რომ ეს არ ნიშნავს იმას, რომ დეტერმინიზაციის შედეგად ამდაგვარი ზრდა აუცილებელია: არ არის გამორიცხული, რომ არსებობდეს რაიმე უფრო ეფექტური მეთოდი, რომლითაც $o(2^{f(n)})$ ზედა ზღვარს მივიღებდით, მაგრამ ეს ჯერ ცნობილი არ არის და თანამედროვეობის ერთ-ერთ უმნიშვნელოვანეს საკითხს, ჩვენს მიერ აქამდე არაერთხელ განხილულ P vs. NP საკითხს უკავშირდება.

სავარჯიშო 8.22: თქვენი სიტყვებით ახსენით, თუ როგორ უკავშირდება დეტერმინიზაციის ამოცანა P vs. NP საკითხს.