ფუნდამენტური ალგორითმები

ალექსანდრე გამყრელიძე

1 ქვეშ მიწერით მიმატების მეთოდი

მოცემულია ორი n ბიტიანი რიცხვი $x = (x_{n-1}, ..., x_0)$ და $y = (y_{n-1}, ..., y_0)$. იმისათვის, რომ გამოვიანგარიშოთ ამ რიცხვთა ჯამი $z = (z_n, ..., z_0) = x + y$, საჭიროა შემდეგი ოპერაციების ჩატარება:

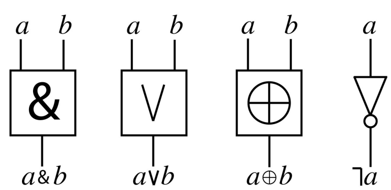
$$\begin{array}{cccc} x_{n-1} & ... & x_1 & x_0 \\ y_{n-1} & ... & y_1 & y_0 \\ \hline z_{n-1} & ... & z_1 & z_0 \end{array}$$

$z_i = x_i \oplus y_i \oplus c_i$,
$c_{i+1} = xy \vee (x_i \oplus y_i)c_i$.

მაგალითი:

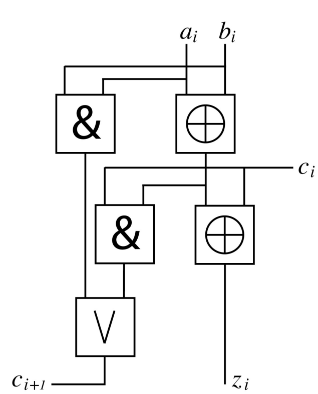
	7	6	5	4	3	2	1	0
x	1	1	0	0	1	0	0	1
y	1	1	1	1	0	0	1	0
z	1	0	1	1	1	0	1	1
c	1	1	0	0	0	0	0	0

ზემოთ მოყვანილი ბულის ალგებრის ფორმულები გრაფიკულადაც შეიძლება გამოვსახოთ:



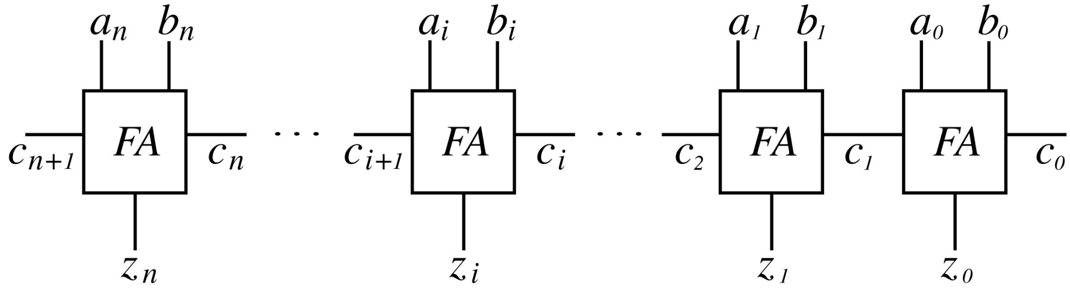
ნახ. 1: ლოგიკური ოპერაციების გრაფიკული გამოსახვა

აქედან გამომდინარე, შეგვიძლია შევადგინოთ z_i და c_i ცვლადების გამოსათვლელი სქემა:



ნახ. 2: z_i და c_{i+1} ცვლადების გამოსათვლელი სქემა

თუ ჩვენ ამ სქემას აღვნიშნავთ როგორც FA (ინგლისური Full Adder, ანუ სრული შემკრები), მაშინ ორი n ბიტიანი რიცხვის შეკრებისათვის საჭირო სქემა შემდეგნაირი იქნება:



ნახ. 3: ორი n ბიტიანი რიცხვის შეკრებისათვის საჭირო სქემა CRA_n

<u>სავარჯიშო 1.1:</u> გამოიანგარიშეთ, რისი ტოლია $T(FA)$ (ანუ იმ ბიჯების რაოდენობა, რაც საჭიროა FA სქემის ყველა შედეგის გამოსაანგარიშებლად) და $C(FA)$ (ანუ FA სქემაში არსებული ელემენტების რაოდენობა), თუ $T(\&) = T(\vee) = T(\neg) = 1$, $T(\oplus) = 3$ და $C(\&) = C(\vee) = C(\neg) = 1$, $C(\oplus) = 5$.

<u>შენიშვნა:</u> ხშირად იღებენ $T(\neg) = 0$ და $C(\neg) = 0$, ანუ სქემებში უარყოფის ელემენტებს უგულებელყოფენ იმის გამო, რომ მათი რეალიზაცია სხვა ელემენტების რეალიზაციასთან შედარებით საკმაოდ მცირეა და, ამავე დროს, უარყოფებს ხშირად იყენებენ დამხმარე ელემენტებად (სხვადასხვა ტექნიკური მიზეზებით ორ ერთმანეთზე მიყოლებულ უარყოფას სვამენ ხოლმე). ამას გარდა, $T(\neg(ab)) = T(ab)$, $T(\neg(a \vee b)) = T(a \vee b)$, $T(\neg ab) = T(ab)$, $T(\neg a \vee b) = T(\neg a \vee b)$.

<u>სავარჯიშო 1.2:</u> გამოიანგარიშეთ, რისი ტოლია $T(\oplus)$, $C(\oplus)$ და, აქედან გამომდინარე, $T(FA)$ იმის გათვალისწინებით, რომ $T(\neg) = C(\neg) = 0$.

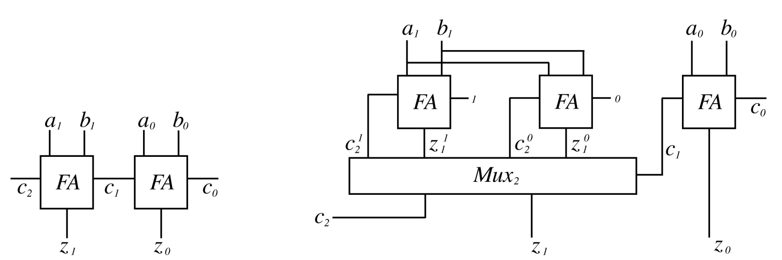
ზემოთ მოყვანილ სქემას ვუწოდებთ CRA_n.

ადვილი დასანახია, რომ z_1 ცვლადი არ გამოითვლება, სანამ არ იქნება გამოთვლილი c_1 და, ზოგადად, z_i ცვლადის გამოთვლა არ შეიძლება, სანამ არ იქნება გამოთვლილი c_{i-1}. აქედან გამომდინარე, $T(CRA_n) = 4n$, $C(CRA_n) = 9n$ (აქ და შემდგომში დავუშვებთ, რომ $T(\neg) = C(\neg) = 0$).

<u>სავარჯიშო 1.3:</u> დაამტკიცეთ $T(CRA_n) = 4n$ და $C(CRA_n) = 9n$ ტოლობები.

ბუნებრივია შემდეგი შეკითხვა: შესაძლებელია თუ არა ბიჯების რაოდენობისა და ელემენტების რიცხვის შემცირება?

თუ დავაკვირდებით ნახ. 4 *(ა)*-ში პირველ ორ FA ელემენტს, დავინახავთ შემდეგ მნიშვნელოვან ფაქტს: მარცხენა FA ელემენტი, რომელიც z_1 და c_1 ცვლადებს ითვლის, „ელოდება" c_0 ცვლადის გამოთვლას.



(ა) *(ბ)*

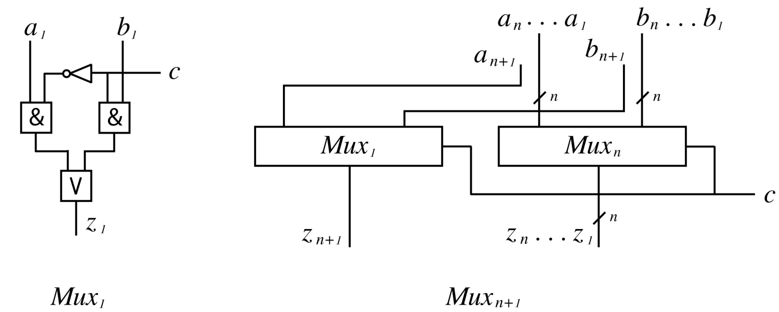
ნახ. 4:

იმის გამო, რომ მას შეიძლება მიეწოდოს მხოლოდ $c_1 = 0$ ან $c_1 = 1$, ჩვენ შეგვიძლია ***ერთდროულად*** გამოვითვალოთ z_1 და c_2 იმ შემთხვევისათვის, როდესაც $c_1 = 0$ (z_1^0, c_2^0) და იმ შემთხვევისათვის, როდესაც $c_1 = 1$ (z_1^1, c_2^1).შემდეგ, როდესაც c_1 გამოთვლილი იქნება, შეიძლება ამ ორი საშუალედო შედეგიდან ერთ-ერთის არჩევა (ნახ. 4 *(ბ)*).

ამ ნახაზში გვხვდება ახალი ელემენტი Mux_2, რომლის მუშაობის შედეგები შემდეგი ცხრილით შეიძლება გამოისახოს:

$$z_1 = \begin{cases} z_1^0, & \text{თუ } c_1 = 0, \\ z_1^1, & \text{თუ } c_1 = 1 \end{cases} \qquad c_2 = \begin{cases} c_2^0, & \text{თუ } c_1 = 0, \\ c_2^1, & \text{თუ } c_1 = 1. \end{cases}$$

ზოგადად, Mux_n შემდეგნაირად შეიძლება აღიწეროს (ნახ. 5) :

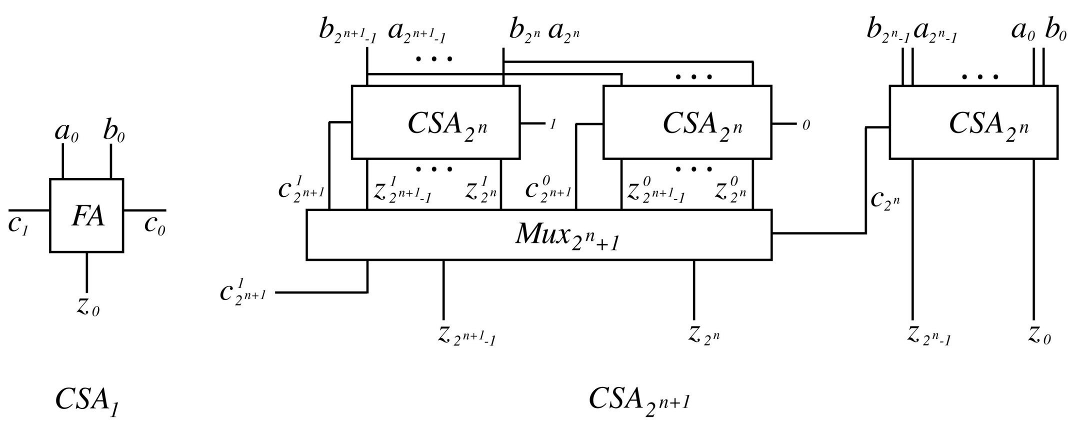


ნახ. 5:

$$z_i = \begin{cases} b_i, & \text{თუ } c = 0, \\ a_i, & \text{თუ } c = 1. \end{cases} \qquad i = \overline{1; n+1}.$$

ნახ. 4 -ში მოყვანილ სქემას უწოდებენ CSA_2. იგი ორ 2 ბიტიან რიცხვს $a = (a_1, a_0)$ და $b = (b_1, b_0)$ შეკრებს და 3 ბიტიან რიცხვს $z = (c_2, z_1, z_0)$ მოგვცემს პასუხად.

ზოგადად, $CSA_{2^{n+1}}$, რომელიც ორ 2^{n+1} ბიტიან რიცხვს $A = (a_{2^n-1}, ..., a_0)$ და $B = (b_{2^n-1}, ..., b_0)$ შეკრებს და $2^{n+1}+1$ ბიტიან რიცხვს $z = (c_{2^n}, z_{2^n-1}, ..., z_0)$ მოგვცემს პასუხად, შემდეგნაირად აღიწერება (ნახ. 6):



ნახ. 6:

CSA_1, ანუ ორი ერთბიტიანი რიცხვის შემკრები არის ზემოთ განხილული სქემა FA.
ძირითადი იდეა „დაყავი და იბატონე" პარადიგმაზეა აგებული: მონაცემები ორ ნაწილად იყოფა —

$$A_1 = (a_{2^{n+1}-1}...a_{2^n}) \quad A_0 = (a_{2^n-1}...a_0)$$
$$B_1 = (b_{2^{n+1}-1}...b_{2^n}) \quad B_0 = (b_{2^n-1}...b_0)$$

შემდეგ გამოითვლება $Z_0 = A_0 + B_0$ და ***ამავდროულად*** $Z_1^0 = A_1 + B_1 + 0$ და $Z_1^1 = A_1 + B_1 + 1$. ამის შემდეგ, c_{2^n} სიგნალის მეშვეობით, ამოირჩევა

$$Z_1 = \begin{cases} Z_1^0, & \text{თუ } c_{2^n} = 0, \\ Z_1^1, & \text{თუ } c_{2^n} = 1 \end{cases} \qquad c_{2^{n+1}} = \begin{cases} c_{2^{n+1}}^0, & \text{თუ } c_{2^n} = 0, \\ c_{2^{n+1}}^1, & \text{თუ } c_{2^n} = 1 \end{cases}$$

აქ $Z_0 = (z_{2^n-1}, ..., z_0)$ და $Z_1 = (z_{2^{n+1}-1}, ..., z_{2^n})$.

ადვილი დასამტკიცებელია ამ სქემის სისწორე მათემატიკურ ინდუქციაზე დაყრდნობით:

- ინდუქციის შემოწმება: თუ $n = 0$, ცხადია, რომ $CSA_1 = FA$ და იგი ორ ერთ ბიტიან რიცხვს სწორად შეკრებს;
- ინდუქციის დაშვება: დავუშვათ, CSA_{2^n} სწორად შეკრებს ორ 2^n ბიტიან რიცხვს;
- ინდუქციის ბიჯი: დავამტკიცოთ, რომ $CSA_{2^{n+1}}$ სწორად შეკრებს ორ 2^{n+1} ბიტიან რიცხვს.

<u>სავარჯიშო 1.4:</u> დაამტკიცეთ, რომ თუ CSA_{2^n} სწორად შეკრებს ორ 2^n ბიტიან რიცხვს, მაშინ $CSA_{2^{n+1}}$ სწორად შეკრებს ორ 2^{n+1} ბიტიან რიცხვს.

რაც შეეხება ამ ალგორითმის ბიჯების რაოდენობას $T(CSA_{2^{n+1}})$, მისი გამოთვლა შემდეგნაირად შეიძლება:
უპირველესად ყოვლისა, უნდა გამოვითვალოთ ცვლადები Z_0, Z_1^0 და Z_1^1, რაც ***ერთდროულად*** შეიძლება მოხდეს $T(CSA_{2^n})$ ბიჯში. ამის შემდეგ უნდა ავირჩიოთ Z_1^0 და Z_1^1 ცვლადებიდან ერთ-ერთი Mux_{2^n+1} სქემის საშუალებით, რაც $T(Mux_{2^n+1}) = 2$ ბიჯშია შესაძლებელი.
აქედან გამომდინარე, $T(CSA_{2^{n+1}}) = T(CSA_{2^n}) + T(Mux_{2^n+1}) = T(CSA_{2^n}) + 2$. ამ რეკურსიული ფორმულის გახსნის შემდეგ მივიღებთ:

$$T(CSA_{2^{n+1}}) = O(\log n).$$

<u>სავარჯიშო 1.5:</u> დაამტკიცეთ ტოლობა $T(Mux_{2^n+1}) = 2$ (გამოიყენეთ ნახ. 5-ში მოყვანილი რეკურსიული სქემა).

$C(CSA_{2^{n+1}})$ ოპერაციათა რაოდენობის გამოსათვლელად გამოვიყენოთ ფორმულა:

$$C(CSA_{2^{n+1}}) = 3 \cdot C(CSA_{2^n}) + C(Mux_{2^n+1}).$$

<u>სავარჯიშო 1.6:</u> დაამტკიცეთ, რომ $C(CSA_{2^{n+1}})$ მართლაც ამ რეკურსიული ფორმულით გამოითვლება და გამოითვალეთ მისი მნიშვნელობა.

როგორც ვხედავთ, პარალელური ალგორითმებით შეიძლება შეკრების ამოცანის სწრაფად გადაჭრა: ორი n ბიტიანი რიცხვისათვის არა $O(n)$, არამედ $O(\log n)$ ბიჯია საჭირო, სამაგიეროდ იზრდება ელემენტების რაოდენობა. ეს გასაკვირი არ არის: მეტ ოპერაციას ვატარებთ, ოღონდ ერთდროულად და ამის ხარჯზე ვიგებთ დროს.

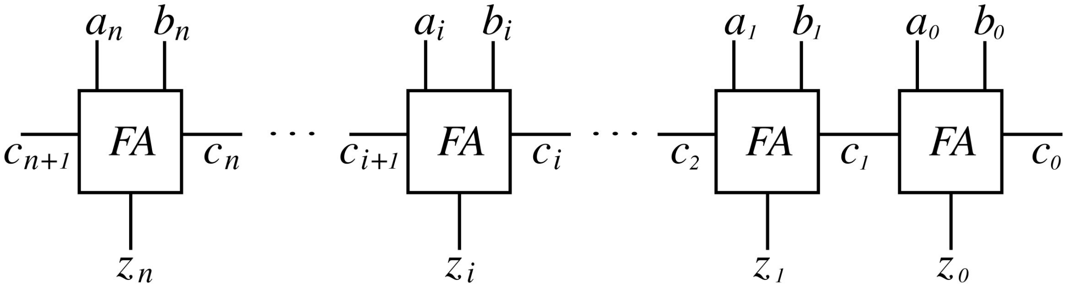
2 პარალელური პრეფიქსის გამოთვლის მეთოდი

დავუშვათ, მოცემულია რაიმე ასოციაციური ოპერატორი $\circ$, ანუ განსაზღვრულია $x \circ y$ და $(x \circ y) \circ z = x \circ (y \circ z)$ (კონკრეტულად $\circ$ შეგვიძლია ავიღოთ, როგორც ორის მოდულით მიმატება: $x \oplus y$).

პრეფიქსის ამოცანა შემდეგნაირად განისაზღვრება:

<u>განმარტება 2.1:</u> მოცემულია პოლინომი $P(x_{n-1}, ..., x_0) = x_n \circ x_{n-1} \circ \cdots \circ x_2 \circ x_1$. ამ პოლინომისათვის ***პრეფიქსის გამოთვლა*** ეწოდება შემდეგი ვექტორის გამოთვლას: $(P(x_n, ..., x_1), P(x_{n-1}, ..., x_1), P(x_{n-2}, ..., x_1), P(x_2, x_1), P(x_1))$.

იმისათვის, რომ გავიგოთ, თუ რა კავშირი აქვს პრეფიქსის გამოთვლის ამოცანას მიმატების ამოცანასთან, კიდევ ერთხელ განვიხილოთ CRA_n:



ნახ. 7: ორი n ბიტიანი რიცხვის შეკრებისათვის საჭირო სქემა CRA_n

რადგან $(z_{n-1}, ..., z_0)$ შედეგის ვექტორის გამოსათვლელად საჭიროა c_i მნიშვნელობის ცოდნა, ეს ამოცანა $(c_n, ..., c_1)$ ვექტორის სწრაფ გამოთვლაზე შეიძლება დავიყვანოთ.
აღსანიშნავია, რომ $c_i = 1$, თუ: $a_i \cdot b_i = 1$, ან $(a_i \oplus b_i) \cdot c_{i-1} = 1$. ეს ორი ფაქტი სიტყვიერად ასე შეიძლება ჩაიწეროს:
$c_i = 1$, თუ შესაბამის FA სქემაში $a_i \cdot b_i = 1$ (იმის და მიუხედავად, თუ რისი ტოლია c_{i-1}), ან $a_i \oplus b_i = 1$ და $c_{i-1} = 1$.
გარდა ამისა, თუ $a_i = b_i = 0$, მაშინ $c_i = 0$ იმის და მიუხედავად, თუ რისი ტოლია c_{i-1}.

აქედან გამომდინარე, შეიძლება ვთქვათ, რომ:

CRA_n ჯაჭვის ყოველი FA განსაღვრავს ფუნქციას P_i იმის და მიხედვით, თუ რისი ტოლია (a_i, b_i).

- თუ $a_i \vee b_i = 0$, მაშინ $c_i = P_i(c_{i-1}) = 0$;
- თუ $a_i \cdot b_i = 1$, მაშინ $c_i = P_i(c_{i-1}) = 1$;
- თუ $a_i \oplus b_i = 1$, მაშინ $c_i = P_i(c_{i-1}) = c_{i-1}$.

როგორც ზემოთ აღვნიშნეთ, თუ $a_i = b_i = 0$, მაშინ $P_i = 0$ იმის და მიუხედავად, თუ რა არის c_{i-1}; თუ $a_i = b_i = 1$, მაშინ $P_i = 1$ იმის და მიუხედავად, თუ რა არის c_{i-1}; თუ $a_i = 1, b_i = 0$ ან $a_i = 0, b_i = 1$, მაშინ $P_i = c_{i-1}$.

აქედან გამომდინარე, შეგვიძლია დავწეროთ:

$$c_i = P_i(c_{i-1}) \circ P_{i-1}(c_{i-2}) \circ \cdots \circ P_2(c_1) \circ P_1(0).$$

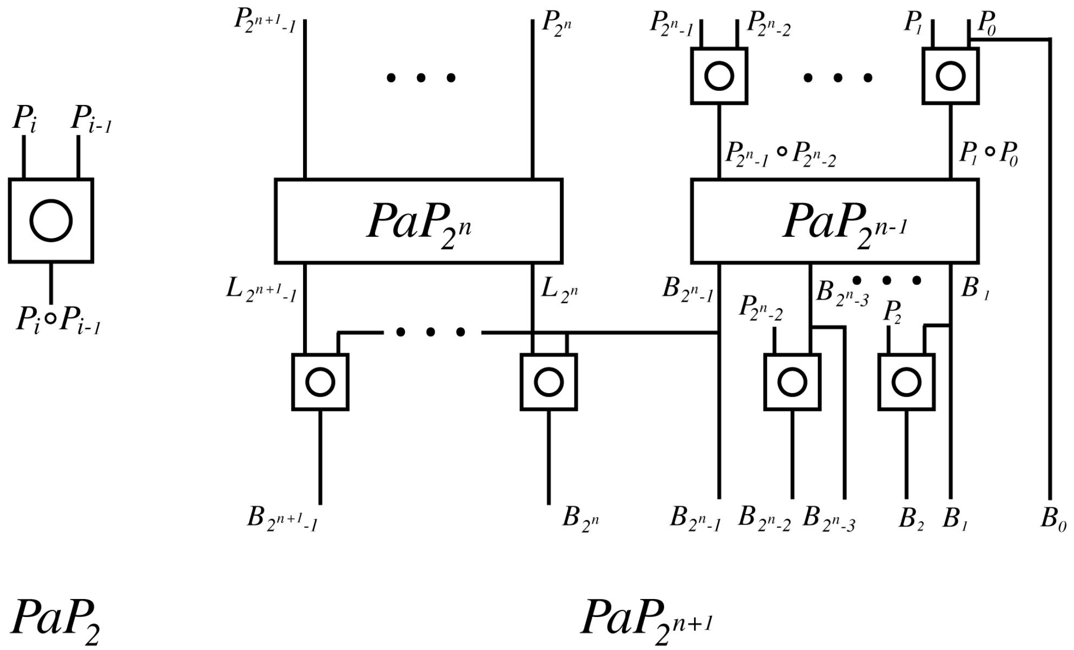
აქ ოპერაცია $\circ$ ფუნქციათა კომპოზიციაა, რაც თავის თავად ასოციაციურია. ამასთან ერთად უნდა აღინიშნოს, რომ:

- თუ $P_i(c_{i-1}) = 0$, მაშინ $P_i \circ P_{i-1} = P_i$ (პასუხად ისევ 0 მივიღებთ);
- თუ $P_i(c_{i-1}) = 1$, მაშინ $P_i \circ P_{i-1} = P_i$ (პასუხად ისევ 1 მივიღებთ);
- თუ $P_i(c_{i-1}) = c_{i-1}$, მაშინ $P_i \circ P_{i-1} = P_{i-1}$ (პასუხად ისევ $P_{i-1}(c_{i-2})$ მივიღებთ).

ესე იგი, c_i ცვლადის გამოსაანგარიშებლად საკმარისია $A_i = P_i \circ \cdots \circ P_1$ ფუნქციის გამოთვლა და მისთვის $A_i(c_0)$ მნიშვნელობის გამოანგარიშება.

აქედან გამომდინარე, c_i ცვლადების გამოთვლა ($i \in \{0, ..., n-1\}$) დაიყვანება პრეფიქსის ამოცანაზე გარკვეული ფუნქციებისათვის.

პრეფიქსის სწრაფი გამოთვლისათვის გამოიყენება ე.წ. პარალელური პრეფიქსის ალგორითმი, რომელიც შემდეგში მდგომარეობს:



ნახ. 8: პრეფიქსების გამოანგარიშებისათვის საჭირო პარალელური სქემა $PaP_{2^{n+1}}$

ორი ფუნქციის კომპოზიცია გამოითვლება სქემით, რომელსაც ჩვენ ვუწოდებთ PaP_2 (კონკრეტულად როგორი იქნება ეს სქემა, დამოკიდებულია მოცემულ ამოცანაზე - სხვადასხვა ამოცანისათვის სხვადასხვა სქემა იქნება საჭირო). დავუშვათ, რომ გვაქვს ორი სქემა PaP_{2^n} და $PaP_{2^{n-1}}$.

$PaP_{2^{n+1}}$ ფუნქციის გამოსაანგარიშებლად პირველ რიგში გამოვიანგარიშებთ წყვილებს $P_{2^n-1} \circ P_{2^n-2}$, $P_{2^n-3} \circ P_{2^n-4}$, ..., $P_1 \circ P_0$ და შემდგომ ამ მონაცემებით, $PaP_{2^{n-1}}$ სქემის მეშვეობით, $B_{2^n-1} = P_{2^n-1} \circ P_{2^n-2} \circ \cdots \circ P_0$, $B_{2^n-3} = P_{2^n-3} \circ P_{2^n-4} \circ \cdots \circ P_0$, ..., $B_1 = P_1 \circ P_0$, ე. ი., პრეფიქსებს კენტი ინდექსის მქონე ფუნქციებამდე, განსაკუთრებით კი $B_{2^n-1} = P_{2^n-1} \circ P_{2^n-2} \circ \cdots \circ P_0$, რაც მაღალი ინდექსის მქონე პრეფიქსების გამოთვლისთვისაა საჭირო.
ამავდროულად, PaP_{2^n} სქემის მეშვეობით, ვანგარიშობთ პრეფიქსებს $L_{2^{n+1}-1} = P_{2^{n+1}-1} \circ \cdots \circ P_{2^n}$, $L_{2^{n+1}-2} = P_{2^{n+1}-2} \circ \cdots \circ P_{2^n}$, ..., $L_{2^n+1} = P_{2^n+1} \circ P_{2^n}$, $L_{2^n} = P_{2^n}$.

ბოლოს, ***ერთდროულად*** ვანგარიშობთ პრეფიქსის იმ ელემენტებს, რომლებიც გვაკლია:

$$\begin{array}{lcl}
B_{2^{n+1}-1} & = & L_{2^{n+1}-1} \circ B_{2^n-1} = P_{2^{n+1}-1} \circ \cdots \circ P_0, \\
B_{2^{n+1}-2} & = & L_{2^{n+1}-2} \circ B_{2^n-1} = P_{2^{n+1}-2} \circ \cdots \circ P_0, \\
\ldots & & \\
B_{2^n} & = & L_{2^n} \circ B_{2^n-1} = P_{2^n} \circ \cdots \circ P_0, \\
B_{2^n-2} & = & P_{2^n-2} \circ B_{2^n-3} = P_{2^n-2} \circ \cdots \circ P_0, \\
B_{2^n-4} & = & P_{2^n-4} \circ B_{2^n-5} = P_{2^n-4} \circ \cdots \circ P_0, \\
\ldots & & \\
B_2 & = & P_2 \cdot B_1 = P_2 \circ P_1 \circ P_0.
\end{array}$$

ამით გამოთვლილი გვექნება ყველა $B_i = P_i \circ \cdots \circ P_0, \forall i \in \{0, ..., 2^{n+1} - 1\}$.

სავარჯიშო 2.7: დაამტკიცეთ, რომ $T(PaP_k) = O(\log k)$ და $C(PaP_k) = O(k)$.

იმისათვის, რომ კონკრეტულად განვსაზღვროთ, თუ როგორი უნდა იყოს $\circ$ ელემენტი სწრაფი მიმატების სქემისათვის, ჯერ უნდა განვსაზღვროთ P_i ფუნქციათა კოდირება:

თუ $P_i \equiv 0$, იგი ჩავწეროთ როგორც $(0, 0)$;
თუ $P_i \equiv 1$, იგი ჩავწეროთ როგორც $(0, 1)$;

თუ $P_i = c_i$, იგი ჩავწეროთ როგორც $(1,0)$.

აქედან გამომდინარე, ვიღებთ:

$(0,0)\circ(0,0)=(0,0)\circ(0,1)=(0,0)\circ(1,0)=(0,0);$
$(0,1)\circ(0,0)=(0,1)\circ(0,1)=(0,1)\circ(1,0)=(0,1);$
$(1,0)\circ(0,0)=(0,0);$
$(1,0)\circ(0,1)=(0,1);$
$(1,0)\circ(1,0)=(1,0).$

და მისი ზოგადი ფორმულა იქნება:

$(x_1,y_1)\circ(x_2,y_2)=(z_1,z_2).$

ზემოთ მოყვანილი ტოლობების თანახმად ვიღებთ:

$z_1 = x_1 \cdot x_2,$
$z_2 = x_1 \cdot y_2 \vee y_1.$

ამ ფორმულების რეალიზაციას ვუწოდოთ $\bigcirc$.

სავარჯიშო 2.8: გამოითვალეთ, რისი ტოლია $T(\bigcirc)$ და $C(\bigcirc)$.
მინიშნება: გამოყავით ის ნაწილები, რომლებიც ერთმანეთისგან დამოუკიდებლად და, აქედან გამომდინარე, ერთდროულად შეიძლება გამოვთვალოთ.

სავარჯიშო 2.9: რა სქემით შეიძლება P_i ფუნქციის მაკოდირებელი (x_i, y_i) ცვლადების გამოთვლა?
მინიშნება: როგორც ზემოთ იყო აღნიშნული, P_i ფუნქცია a_i და b_i ცვლადებით განისაზღვრება.

აქედან გამომდინარე, $c_i = P_i\circ\cdots\circ P_0(0)$ $(i=\overline{1;n})$ შეიძლება გამოვითვალოთ $O(\log n)$ ბიჯსა და $O(n)$ ოპერაციაში.

ზემოთ აღწერილი მეთოდი პირველად გერმანელმა მათემატიკოსებმა ლადნერმა და ფიშერმა (Ladner, Fischer) გამოაქვეყნეს [3] და მის საფუძველზე სწრაფი და მცირე ზომის ალგორითმი LF_n ააგეს, რომელიც ორ n ბიტიან რიცხვს კრებს.

მისი ძირითადი ნაწილი ამავე მათემატიკოსების მიერ შემუშავებულ ე.წ. პარალელ პრეფიქს ალგორითმზეა დაყრდნობილი Parallel Prefix), რომელიც ჩვენ ზემოთ აღვწერეთ.

3 სწრაფი და მცირე ზომის შემკრები ალგორითმი

ლადნერისა და ფიშერის მეთოდზე აგებული შემკრები ალგორითმი თეორიულ ქვედა ზღვარს აღწევს, რაც იმას ნიშნავს, რომ ვერ იარსებებს სხვა ალგორითმი, რომლის ბიჯების რაოდენობისა და ოპერაციათა რიცხვის ასიმპტოტური ზრდის რიგი უკეთესი იქნებოდა.

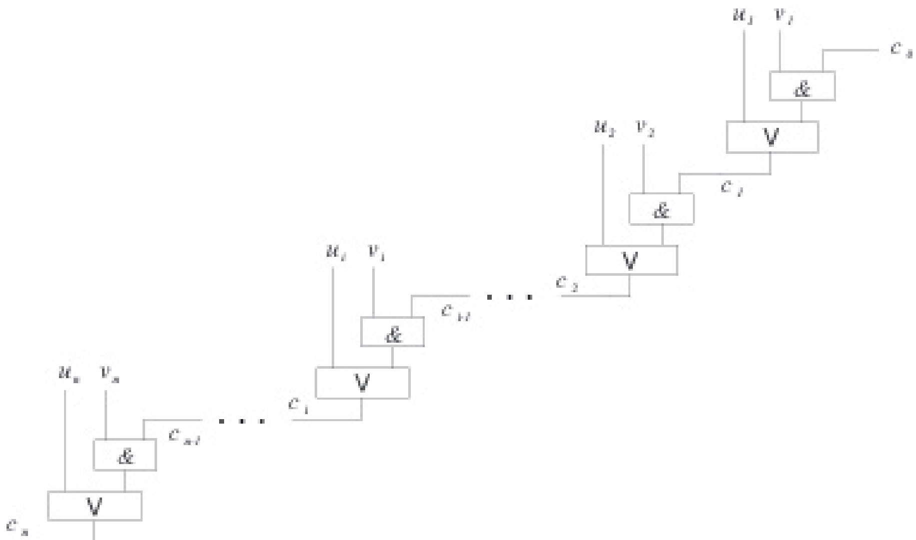
მაგრამ თუ გამოვიანგარიშებთ ბიჯებისა და ოპერაციათა რიცხვის ზუსტ რაოდენობას, დავინახავთ, რომ მათი გაუმჯობესება თეორიულად შესაძლებელია.

$$T(LF_n) = 2\log n + 2, \quad C(LF_n) \le 15n - 18\sqrt{n} - 1.$$

ამ თავში ჩვენ ერთ მეთოდს წარმოვადგენთ, რომლის საფუძველზედაც დღეისათვის ცნობილი ყველაზე სწრაფი შეკრების ალგორითმის შექმნაა შესაძლებელი. ამ მეთოდის მონაცემთა დაყოფის იდეა პირველად აღწერილი იყო ნაშრომში [1] და შემდეგ გავრცობილი და გამოყენებული სიმეტრიული და რეკურსიული სქემის შესაქმნელად ნაშრომში [8].

აქ ჩვენ ამ ალგორითმის ძირითად იდეას წარმოვადგენთ.

როგორც წინა თავში აღინიშნა, ორი რიცხვის ჯამის სწრაფად გამოთვლისათვის საკმარისია c_i სიგნალების სწრაფი გამოთვლა ($i=\overline{1;n}$). თუ დავუშვებთ, რომ $u_i = a_i \cdot b_i$ და $v_i = a_i \oplus b_i$, ადვილი დასანახია, რომ c_i სიგნალები შემდეგი სქემით გამოითვლება:



ნახ. 9: c_i სიგნალების გამოსათვლელი მიმდევრობითი სქემა

როგორც წინა თავში აღინიშნა, აქ მოყვანილი სქემის თითო ნაწილი გარკვეულ ფუნქციას განსაზღვრავს:
თუ დავაკვირდებით ამ სქემაში მოცემული ჯაჭვის ერთ ნაწილს, დავინახავთ, რომ იგი $\varphi_{u_i,v_i}(c_{i-1}) \mapsto c_i$ ფუნქციებს განსაზღვრავს, რომლებიც u_i, v_i ცვლადებით განისაზღვრება.

$$\begin{array}{lll}
\text{Generate}: & c_i = 1 & (u_i = 1, v_i = 0); \\
\text{Propagate}: & c_i = c_{i-1} & (u_i = 0, v_i = 1); \\
\text{Eliminate}: & c_i = 0 & (u_i = 0, v_i = 0).
\end{array}$$

(ანალოგიურად წინა თავში განხილული ფუნქციებისა).

c_{2^n} ცვლადისათვის ვიღებთ: $c_{2^n} = u_{2^n} \vee v_{2^n} \cdot c_{2^n-1}$. ამ ფორმულის იტერაციის შედეგად ვიღებთ:

$$c_{2^n} = u_{2^n} \vee v_{2^n} \cdot u_{2^n-1} \vee \cdots \vee (v_{2^n} \cdots v_{2^n-i}) \cdot u_{2^n-i-1} \vee \cdots \vee (v_{2^n} \cdots v_2) \cdot u_1$$

(აქ $c_0 = 0$).

თუ ამ ტოლობის მარჯვენა მხარეს $f_{2^n}(u_{2^n}, v_{2^n}, ..., u_2, v_2, u_1)$ ფუნქციად განვიხილავთ, მივიღებთ:

$$f_{2^n}(u_{2^n}, ..., u_1) = f_{2^{n-1}}(u_{2^n}, ..., u_{2^{n-1}+1}) \vee (v_{2^n} \cdots v_{2^{n-1}+1}) f_{2^{n-1}}(u_{2^{n-1}}, ..., u_1).$$

ამ ფორმულის r სიღრმემდე იტერაციის შედეგად ვიღებთ:

$$\begin{aligned}
f_{2^n}(u_{2^n}, ..., u_1) = \; & f_{2^r}(u_{2^n}, ..., u_{2^n-2^r+1}) \vee (v_{2^n} \cdots v_{2^n-2^r+1}) \cdot f_{2^r}(u_{2^n-2^r}, ..., u_{2^n-2\cdot 2^r+1}) \vee \cdots \vee \\
& (v_{2^n} \cdots v_{2^n-k\cdot 2^r+1}) \cdot f_{2^r}(u_{2^n-k\cdot 2^r}, ..., u_{2^n-(k+1)\cdot 2^r+1}) \vee \cdots \vee \\
& (v_{2^n} \cdots v_{2^r+1}) \cdot f_{2^r}(u_{2^r}, ..., u_1).
\end{aligned}$$

ცხადია, რომ ამ ფორმულის გამოსათვლელად საჭირო ბიჯების რაოდენობაა

$$T(f_{2^n}) = \max\{T(f_{2^r}), T(v_{2^n} \cdots v_{2^r+1})\} + n - r + 1.$$

თუ განვსაზღვრავთ მიმდევრობას $(m_i)_{i=1}^{\infty}$:

$$m_1 = 0, \quad m_i = m_{i-1} + (i-1) = \frac{i \cdot (i-1)}{2}$$

და ზედა ფორმულაში ჩავსვამთ $n = m_i$, $r = m_{i-1}$, მივიღებთ:

$$T(f_{2^{m_i}}) = \max\{T(f_{2^{m_{i-1}}}), T(v_{2^{m_i}} \cdots v_{2^{m_{i-1}}+1})\} + (m_i - m_{i-1}) + 1.$$

შენიშვნა: ინდუქციის დახმარებით ადვილად მტკიცდება შემდეგი უტოლობა:

$$T(f_{2^{m_i}}) \le m_{i+1} \qquad \forall i \in \mathbb{N}.$$

თუ $m \in N$ და $r = m_{t-1} < m \le m_t$, მივიღებთ:

$T(f_{2^m}) = T(f_{2^{m_{t-1}}}) + (m - m_{t-1}) + 1 \le m_t + m - m_{t-1} + 1 = (t-1) + m + 1 = t + m.$

$m_{t-1} < m$ უტოლობიდან გამომდინარეობს:

$$\frac{(t-1)\cdot(t-2)}{2} < m.$$

ამ უტოლობის ამოხსნის შედეგად ვიღებთ:

$$t < \frac{3 + \sqrt{8m+1}}{2} = 1.5 + \sqrt{2m + 0.25}.$$

აქედან გამომდინარეობს

$$T(f_{2^m}) < 1.5 + \sqrt{2m + 0.25} + m.$$

იგივე ფორმულა შეგვიძლია შემდეგნაირადაც ჩავწეროთ:

$$T(f_n) < \log n + \sqrt{2 \cdot \log n + 0.25} + 1.5 \ \ (\text{აქ } n = 2^m).$$

ანალოგიურად შეიძლება გამოვითვალოთ დანარჩენი c_i სიგნალებიც, რის შედეგადაც ორი n ბიტიანი რიცხვის შეკრების სწრაფი ალგორითმის შედგენა მოხერხდება.

ამ მეთოდით შედგენილი სქემა ლადნერისა და ფიშერის სქემაზე უფრო სწრაფია, რადგან იგი $2\log n$ ბიჯის ნაცვლად $\log n + O(\sqrt{\log n})$ ბიჯს იყენებს.

მოკლე დასკვნა

არითმეტიკის (და არა მარტო არითმეტიკის) ალგორითმების დაჩქარება მათი გაპარალელების შედეგადაა შესაძლებელი. გაპარალელების დროს გამოიყენება ე.წ. „გაყავი და იბატონე" პარადიგმა: შემომავალი მონაცემები ორ ან რამოდენიმე ნაწილად იყოფა, თითოეული ნაწილი სხვისგან დამოუკიდებლად და მასთან ერთად მუშავდება, რის შემდეგადაც საშუალედო მონაცემები ისე მუშავდება, რომ საბოლოოდ საჭირო შედეგი მივიღოთ.

ზოგჯერ კარგია ყველა შესაძლო ვარიანტის გამოთვლა, როგორც ეს CSA_n სქემაში გავაკეთეთ, რომ შემდეგ ერთ-ერთი საჭირო შედეგი ამოვირჩიოთ.
ხშირად მონაცემები ორ ტოლ ნაწილად იყოფა (მაგ. CSA_n, ან LF_n), ზოგში კი ორზე მეტ ნაწილად (როგორც ბოლო მაგალითში), თანაც ამ ნაწილების ზომები იტერაციის ბიჯზეა დამოკიდებული.

მაგრამ ყველა მეთოდში მოქმედებს ერთი პრინციპი:

მონაცემები დაანაწევრე, ერთმანეთისაგან დამოუკიდებლად და ერთდროულად დაამუშავე, შემდეგ მიღებული საშუალედო შედეგები შეაჯამე ისე, რომ საბოლოო სასურველი შედეგი მიიღო.

4 ქვეშ მიწერით გამრავლების მეთოდი

მოცემულია ორი n ბიტიანი რიცხვი $x = (x_{n-1}, ..., x_0)$ და $y = (y_{n-1}, ..., y_0)$. იმისათვის, რომ გამოვიანგარიშოთ ამ რიცხვთა ნამრავლი $z = (z_{2n-1}, ..., z_0) = x \cdot y$, საჭიროა შემდეგი M_n ალგორითმის ჩატარება:

1. გამოიანგარიშე $c_i = (x_{n-1} \cdot y_i, x_{n-2} \cdot y_i, ..., x_0 \cdot y_i), \forall i \in \{0; n-1\}$;
2. გამოიანგარიშე $x \cdot y = z = \sum_{i=0}^{n-1} c_i \cdot 2^i$.

<u>სავარჯიშო 4.1:</u> დაამტკიცეთ, რომ ორი n ბიტიანი რიცხვის გამრავლების შედეგად ***შესაძლებელია*** $2n$ ბიტიანი რიცხვის მიღება.

<u>სავარჯიშო 4.2:</u> დაამტკიცეთ, რომ ზემოთ მოყვანილი ალგორითმის ჩატარების შემდეგ მართლაც x და y რიცხვების ნამრავლს მივიღებთ.

მაგალითი:

									7	6	5	4	3	2	1	0
x									1	1	0	0	1	0	0	1
y									1	1	1	1	0	0	1	0
									1	1	1	1	0	0	1	0
								0	0	0	0	0	0	0	0	
							0	0	0	0	0	0	0	0		
						1	1	1	1	0	0	1	0			
					0	0	0	0	0	0	0	0				
				0	0	0	0	0	0	0	0					
			1	1	1	1	0	0	1	0						
		1	1	1	1	0	0	1	0							
z	1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0

<u>სავარჯიშო 4.3:</u> დაამტკიცეთ, რომ $C(M_n) = O(n^2)$. როგორ შეიძლება $T(M_n)$ სიდიდის ასიმპტოტური შეფასება ?

ბუნებრივია შემდეგი შეკითხვა: შეიძლება თუ არა $T(M_n) = n^2$ და $C(M_n) = n^2$ სიდიდეების შემცირება?

სიმარტივისათვის დავუშვათ, რომ $n = 2^k$ (თუ გვაქვს $2^{k-1} < n < 2^k$ შემთხვევა, შეგვიძლია 2^k ბიტიანი რიცხვის განხილვა, რომელშიც $2^k - n$ უფროსი ბიტი იქნება 0).

თუ მოცემულია ორი n ბიტიანი რიცხვი x და y, პირველ რიგში გავყოთ მათი ორობითი წარმოდგენა ორ ტოლ ნაწილად:

$$x = (x', x'') \text{ და } y = (y', y'').$$

აღსანიშნავია, რომ x', x'', y', y'' თავის მხრივ $2^{\frac{n}{2}}$ ბიტიანი რიცხვებია.

რადგან $x = x' \cdot 2^{\frac{n}{2}} + x''$ და $y = y' \cdot 2^{n-1} + y''$,

$$x \cdot y = (x' \cdot 2^{\frac{n}{2}} + x'') \cdot (y' \cdot 2^{\frac{n}{2}} + y'') = x' \cdot y' \cdot 2^n + (x' \cdot y'' + x'' \cdot y') \cdot 2^{\frac{n}{2}} + x'' \cdot y''. \tag{1}$$

თავის მხრივ,

$$x' \cdot y'' + x'' \cdot y' = (x' + x'')(y' + y'') - (x' \cdot y' + x'' \cdot y'')$$

და, აქედან გამომდინარე,

$$x \cdot y = x' \cdot y' \cdot 2^n + [(x' + x'')(y' + y'') - (x' \cdot y' + x'' \cdot y'')] \cdot 2^{\frac{n}{2}} + x'' \cdot y''. \tag{2}$$

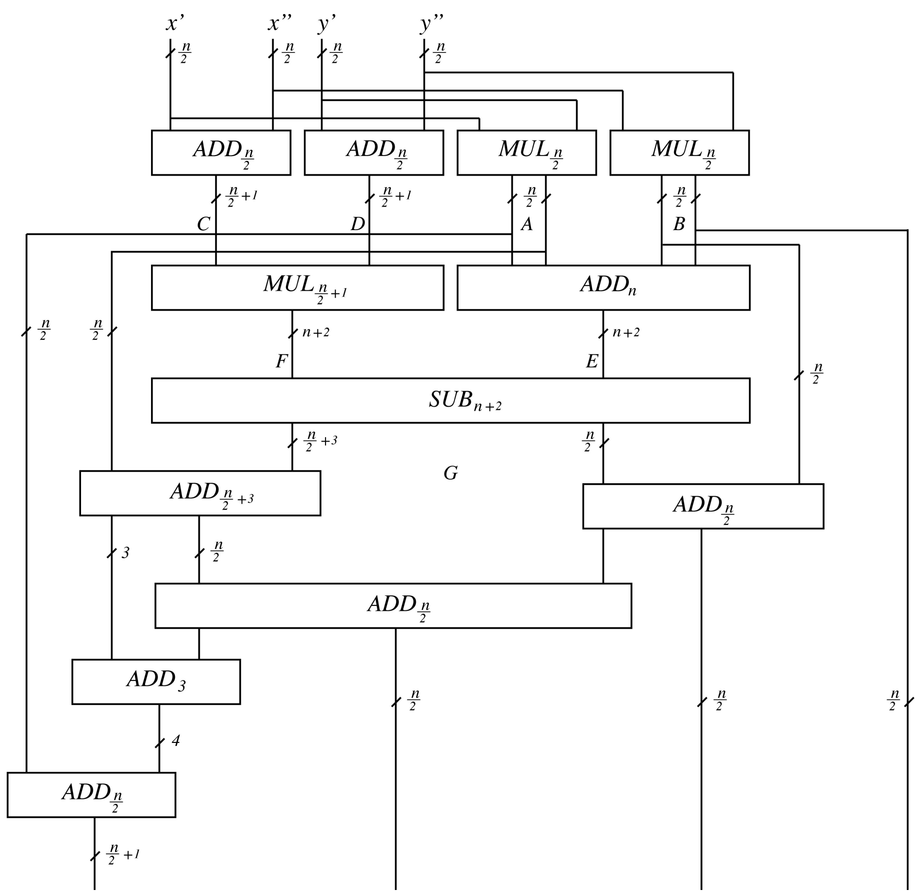
სანამ კონკრეტული ალგორითმის ჩამოყალიბებაზე გადავალთ, შემოვიტანოთ რამოდენიმე აღნიშნვა:

- ორი n ბიტიანი x და y რიცხვის შეკრების ალგორითმი აღვნიშნოთ როგორც $ADD_n(x, y)$;
- ორი n ბიტიანი x და y რიცხვის გამრავლების ალგორითმი აღვნიშნოთ როგორც $MUL_n(x, y)$;
- ორი n ბიტიანი x და y რიცხვის გამოკლების ალგორითმი აღვნიშნოთ როგორც $SUB_n(x, y)$.

მათი გამოყენებით ორი n ბიტიანი x და y რიცხვის გამრავლების ალგორითმი შემდეგნაირად შეიძლება ჩაიწეროს:

ალგორითმი $MUL_n(x, y)$

1. *პარალელურად* გამოიანგარიშე $A = x' \cdot y'$, $B = x'' \cdot y''$, $C = x' + x''$ და $D = y' + y''$;
   აქ ვიყენებთ ალგორითმებს $MUL_{\frac{n}{2}}(x', y')$, $MUL_{\frac{n}{2}}(x'', y'')$, $ADD_{\frac{n}{2}}(x', x'')$, $ADD_{\frac{n}{2}}(y', y'')$
2. *პარალელურად* გამოიანგარიშე $E = A + B$ და $F = C \cdot D$;
   აქ ვიყენებთ ალგორითმებს $ADD_n(A, B)$ და $MUL_{\frac{n}{2}+1}(C, D)$
3. გამოიანგარიშე $G = F - E$;
   აქ ვიყენებთ ალგორითმს $SUB_{n+2}(F, E)$
4. გამოიანგარიშე $A \cdot 2^n + G \cdot 2^{\frac{n}{2}} + B$;
   აქ უნდა აღინიშნოს, რომ $X \cdot 2^i$ არანაირი ოპერაციის ჩატარებას არ მოითხოვს – ეს მხოლოდ X რიცხვის i ბიტით მარცხნივ „ჩაჩოჩებაა".



ნახ. 10:

ეს ალგორითმი პირველად 1963 წელს კარაცუბამ და ოფმანმა წარმოადგინეს.

<u>სავარჯიშო 4.4:</u> დაამტკიცეთ, რომ $T(MUL_n) = O(\log^2 n)$ და $C(MUL_n) = O(n^{\log 3})$.

<u>სავარჯიშო 4.5:</u> გააანალიზეთ, თუ რატომ გამოიყენება კარაცუბასა და ოფმანის ალგორითმში ზემოთ მოყვანილი ტოლობა (2) და არა უფრო მარტივი ტოლობა (1).

კარაცუბასა და ოფმანის ალგორითმიც „გაჰყავი და იბატონე" პარადიგმაზეა აგებული. როგორც ზემოთ განხილული სხვა ალგორითმები, ისიც მონაცემებს ჰყოფს ორ ნაწილად, მათ ცალ-ცალკე ამუშავებს და შემდეგ მიღებულ შედეგს აჯამებს.

5 შონჰაგესა და შტრასენის გამრავლების მეთოდი

გამრავლების ამოცანის სირთულე დღეისათვის ბოლომდე შესწავლილი არ არის. ორი n ბიტიანი რიცხვის გამრავლებისათვის საჭირო ოპერაციების რაოდენობის თეორიული ქვედა ზღვარია $\Omega(n)$, რაც იმას ნიშნავს, რომ არაა გამორიცხული ისეთი ალგორითმის შექმნა, რომლის ოპერაციათა რაოდენობა $O(n)$ იქნება.

მაგრამ დღეისათვის ცნობილი ყველაზე ეფექტური - შონჰაგესა და შტრასენის ალგორითმი - $O(\log n)$ ბიჯსა და $O(n \cdot \log n \cdot \log\log n)$ ოპერაციათა რაოდენობას მოითხოვს ([2]).

ამის გათვალისწინებით შეიძლება ითქვას, რომ თეორიული კვლევის შედეგად შესაძლებელია ან ქვედა ზღვრის გაუმჯობესება, ან ამაზე უფრო ეფექტური ალგორითმის შექმნა.

შონჰაგესა და შტრასენის ალგორითმი დაფუძნებულია ე.წ. ჩინურ თეორემაზე ნაშთების შესახებ, რომელიც საშუალებას გვაძლევს, გამოთვლები დიდ $\mathbb{Z}_m$ რგოლში დავიყვანოთ პარალელურ გამოთვლებზე პატარა $\mathbb{Z}_{m_i}$ რგოლებში, სადაც მიმდევრობა $(m_i)_{i=1}^k$ გარკვეულ მოთხოვნებს უნდა აკმაყოფილებდეს.

თეორემა 5.1: მოცემულია $m = m_1 \cdots m_k$, სადაც ნებისმიერი ორი ელემენტი m_i და m_j ურთიერთმარტივია. დავუშვათ, $r_j = \frac{m}{m_j}$ და $s_j = r_j^{-1} \; mod \, m_j$. მაშინ განტოლებათა სისტემას $a \equiv a_i \; mod \, m_i$ აქვს ცალსახა ამონახსნი $\mathbb{Z}_m$ რგოლში:

$$a \equiv \sum_{1 \le i \le k} a_i \cdot s_i \cdot r_i \; mod \, m.$$

დამტკიცება: პირობის თანახმად, m_j და $\frac{m}{m_j}$ ურთიერთმარტივია. აქედან გამომდინარე, $s_j = r_j^{-1} \; mod \, m_j$ ცალსახად უნდა არსებობდეს (ცნობილი ფაქტი ალგებრიდან).

პირველ რიგში უნდა ვაჩვენოთ, რომ a ზემოთ მოყვანილი განტოლებათა სისტემის ამონახსნია (და შემდეგ მისი ერთად-ერთობა დავამტკიცოთ). ნებისმიერი $i \neq j$ რიცხვებისათვის r_j არის m_i რიცხვის ჯერადი და, აქედან გამომდინარე, $r_j \equiv 0 \; mod \, m_i$. რადგან $r_i \cdot s_i \equiv 1 \; mod \, m_i$, ჭეშმარიტია შემდეგი ტოლობა: $a \equiv a_i \cdot r_i \cdot s_i \equiv a_i \; mod \, m_i$.

თუ რომელიმე რიცხვი $b \neq a$ ასევე ამ სისტემის ამონახსნია, მაშინ $a - b \equiv 0 \; mod \, m_i$, $1 \le i \le k$, რაც იმას ნიშნავს, რომ $a - b$ არის m_i რიცხვის ჯერადი. მაგრამ რადგან m_i და m_j ურთიერთმარტივებია $\forall i \neq j$, $a - b$ უნდა იყოს m რიცხვის ჯერადი და, აქედან გამომდინარე, ან $a \notin \{0, ..., m-1\}$, ან $b \notin \{0, ..., m-1\}$, რასაც წინააღმდეგობამდე მივყავართ.

Q.E.D.

აქედან გამომდინარე, თუ გვინდა გამოვითვალოთ $c = a \cdot b \; mod \, m$ რაღაცა დიდი m რიცხვისათვის, შესაძლებელია $c_i = a \cdot b \; mod \, m_i$, $1 \le i \le k$ გამოთვლა პატარა რგოლებში m_i (თუ ეს რიცხვები ჩინური თეორემის პირობებს აკმაყოფილებს) და მიღებული საშუალედო შედეგებიდან c_i საბოლოო შედეგის ადვილად მიღება შეიძლება.

აქ მთავარია, რომ მცირე ზომის რგოლებში არითმეტიკული ოპერაციების ჩატარება უფრო ადვილია, ვიდრე დიდი ზომის რგოლში.

სავარჯიშო 5.1: ფორმალურად აჩვენეთ, რომ თუ $r_j \equiv 0 \; mod \, m_i$ და $r_i \cdot s_i \equiv 1 \; mod \, m_i$, ჭეშმარიტია შემდეგი ტოლობა: $a \equiv a_i \cdot r_i \cdot s_i \equiv a_i \; mod \, m_i$.

ჩინურ თეორემასთან ერთად გამოიყენება ე.წ. ფურიეს დისკრეტული გარდაქმნა და მისი შებრუნებული:

განმარტება 5.1: მოცემულია კომუტაციური რგოლი $\mathcal{R}$ ერთეულოვანი ელემენტით e და მისი n რიგის პრიმიტიული ერთეულოვანი ფესვი w (ანუ $w^n = e$). ამას გარდა, მოცემულია $\mathcal{R}$ რგოლზე განსაზღვრული პოლინომი $k(x)$ კოეფიციენტებით $a = (a_0, ..., a_{n-1})$. ფურიეს დისკრეტული გარდაქმნა $DFT_n(a)$ შემდეგნაირი ვექტორით განისაზღვრება:

$$DFT_n(a) = (k(w^0), ..., k(w^{n-1})).$$

აღვნიშნოთ რამოდენიმე ფაქტი დამტკიცების გარეშე:

1. ფურიეს დისკრეტული გარდაქმნის გამოთვლა შესაძლებელია $O(\log n)$ ბიჯსა და $O(n \log n)$ ოპერაციაში;

2. ფურიეს დისკრეტული გარდაქმნის შებრუნებული DFT_n^{-1} არსებობს, თუ $\mathcal{R}$ რგოლში n და w ელემენტებს შებრუნებული ელემენტები მოეძებნებათ: $\exists n^{-1}, w^{-1}, n \cdot n^{-1} = e, w \cdot w^{-1} = e$;

3. ფურიეს დისკრეტული გარდაქმნის შებრუნებულის — DFT_n^{-1} — გამოთვლა შესაძლებელია $O(\log n)$ ბიჯსა და $O(n \log n)$ ოპერაციაში.

აღსანიშნავია, რომ შონჰაგესა და შტრასენის ალგორითმი თეორიული თვალსაზრისით ყველა დღეისათვის ცნობილ მეთოდს ჯობია, მაგრამ მისი პრაქტიკაში გამოყენება შეუძლებელია, რადგან $O(n \cdot \log n \cdot \log\log n)$ აღნიშნვაში მუდმივები ძალიან დიდია.

შემდგომში დავუშვათ, რომ $n = 2^k$, $k \in \mathbb{N}$ და გვინდა ორი ნატურალური რიცხვის $x, y \in \{1, ..., 2^n\}$ გამრავლება, ანუ $p \equiv x \cdot y \ mod \ (2^n + 1)$.

სავარჯიშო 5.2: აჩვენეთ, რომ თუ $x \in \{1, 2^n\}$ ან $y \in \{1, 2^n\}$, ნამრავლის გამოთვლა ადვილია.

ჩვენ განვიხილავთ იმ შემთხვევას, როდესაც $1 < x, y < 2^n$.
ამისათვის $n = 2^k$ ბიტიანი რიცხვები x და y დავყოთ $b = 2^{\lfloor k/2 \rfloor}$ ბლოკად, რომელთა სიგრძეც იქნება $l = n/b = 2^{\lceil k/2 \rceil}$. შედეგად მივიღებთ $x = (x_{b-1}, ..., x_0)$ და $y = (y_{b-1}, ..., b_0)$, სადაც $x_i, y_i \in \{0, 1\}^l$. თუ ჩავთვლით, რომ f და g $(b-1)$ რიგის პოლინომებია კოეფიციენტებით $x' = (x_0, ..., x_{b-1})$ და $y' = (y_0, ..., y_{b-1})$, მაშინ $x = f(2^l)$ და $y = g(2^l)$. აქედან გამომდინარე, $p \equiv x \cdot y = h(2^l)$, სადაც $h(x) = f(x) \cdot g(x)$. ამით ჩვენ ორი რიცხვის გამრავლების ამოცანა პოლინომების გამრავლებაზე დავიყვანეთ, რაც, თავის თავად, ბევრ ბიჯს მოითხოვს, მაგრამ ამ კონკრეტულ შემთხვევაში ჩვენ არა $h(x)$ პოლინომის ზოგადი სახე, არამედ მისი მნიშვნელობა 2^l წერტილში გვაინტერესებს.

ამრიგად, მიღებული ალგორითმი შეიძლება შემდეგნაირად ჩაიწეროს:

1. გამოიანგარიშე $DFT_{2n}^{-1}(DFT_{2n}(a) * DFT_{2n}(b)) \ \ mod \ \ (2^l + 1)$;
2. გამოიანგარიშე $DFT_{2n}^{-1}(DFT_{2n}(a) * DFT_{2n}(b)) \ \ mod \ \ b$;
3. გამოიანგარიშე $DFT_{2n}^{-1}(DFT_{2n}(a) * DFT_{2n}(b)) \ \ mod \ \ (2^l + 1)b$;
4. აქედან საბოლოო შედეგის გამოანგარიშება ადვილად შეიძლება.

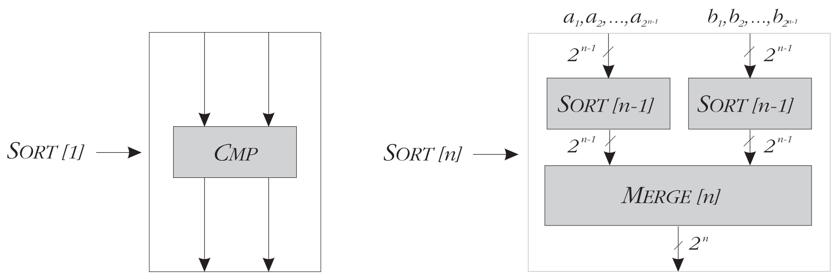
პირველ 2 ბიჯში გამოითვლება შედეგი შედარებით მცირე რგოლებში. შემდეგ კი ჩინურ თეორემაზე დაყრდნობით გამოითვლება საბოლოო შედეგი.

აღსანიშნავია ის ფაქტი, რომ ამ ალგორითმში არა ***მონაცემები*** იყოფა ნაწილებად, არამედ ***გამოთვლის სივრცე***. ესეც „დაჰყავი და იბატონე" პარადიგმის ერთ-ერთი გამოყენებაა, ოღონდ სრულიად განსხვავებული კუთხით. ს

6 პარალელური დალაგება

6.1 Odd-Even-Merge-Sort

ამ ნაწილში განხილულ დალაგების სქემას *Sort[n]* ([7]) შემდეგი იერარქიული სტრუქტურა აქვს:



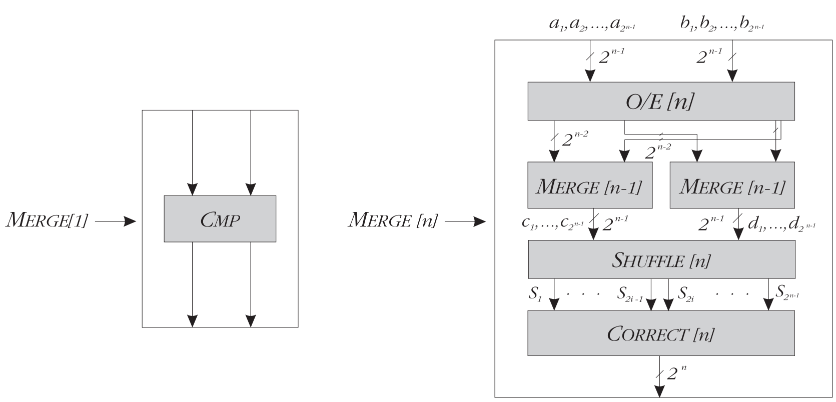
ნახ. 11: სქემა *Sort[n]*

მისი ძირითადი კომპონენტია *CMP* რომელიც ორ რიცხვს ალაგებს და ამ იერარქიულ დონეზე არ ჩანს. აშკარაა, რომ *CMP = Sort[1].* თუ *Sort[n]* სქემის იერარქიულ სტრუქტურას ბოლომდე ჩავყვებით, დავინახავთ, რომ იგი მხოლოდ გარკვეული წესით ერთმანეთთან მიერთებული *CMP* ელემენტებისაგან შედგება.

სავარჯიშო 6.1: დახაზეთ *CMP* სქემა, რომელიც ორ ერთბიტიან რიცხვს სწორად დაახარისხებს.

სავარჯიშო 6.2: დახაზეთ *Sort[4]*, *Sort[8]* და *Sort[16]* სქემები გაშლილი სახით (ისე, რომ მხოლოდ *CMP* ელემენტები გვხვდებოდეს).

Merge[n] ელემენტებს სწორად ალაგებს, თუ $(a_i)_1^{2^{n-1}}$ და $(b_i)_1^{2^{n-1}}$ მიმდევრობები თავიანთ მხრივ სწორად იყო დალაგებული (ნახ. 12).

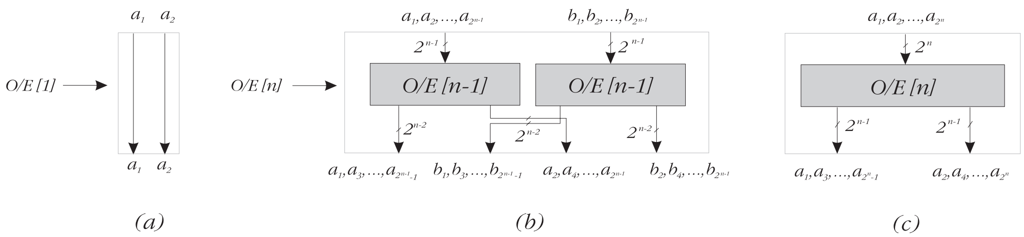


ნახ. 12: სქემა *Merge[n]*

Sort[1] სქემის სისწორე აშკარაა. დავუშვათ, რომ *Sort[i]* ჭეშმარიტია $\forall i \in \{1, ..., n-1\}$.
რა თქმა უნდა, *Sort[n]* სქემის სისწორის დასამტკიცებლად საკმარისი იქნება *Merge[n]* სწემის სისწორის დამტკიცება.

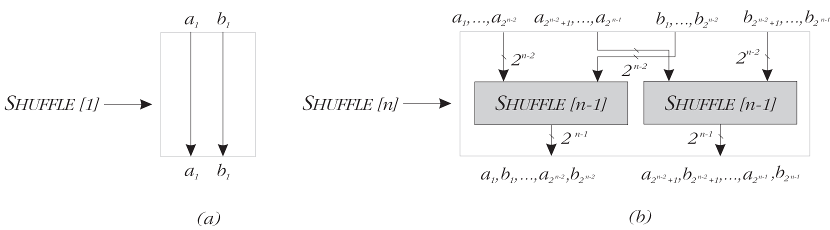
6.1.1 *Merge[n]* სქემის სისწორის მტკიცება

Merge[n] სქემის რეკურსიული სტრუქტურა ნაჩვენებია ნახაზში 12.
სქემა $O/E[n]$ განსაზღვრულია ნახაზში 13.
თუ ამ სქემას დასამუშავებლად მივცემთ ელემენტებს $\{a_i\}_{i=1}^{2^n}$, იგი პასუხად მარცხენა მხარეს კენტ, ხოლო მარჯვნივ კი ლუწ ინდექსიან ელემენტებს მოგვცემს.



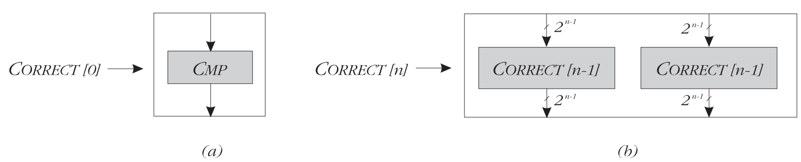
ნახ. 13: $O/E[n]$ სქემის პრინციპი

სქემა *Shuffle[n]* ზემოთ განხილული *O/E[n]* სქემის შებრუნებულია (ნახ. 14).



ნახ. 14: სქემა *Shuffle[n]*

ბოლოს, *Correct[n]* ისეთი სქემაა, რომელიც 2^n გვერდი-გვერდ დასმული *CMP* ელემენტებისაგან შედგება და ორ მეზობელ ელემენტს ახარისხებს (ნახ. 15).



ნახ. 15: სქემა *Correct[n]*

სავარჯიშო 6.3: ინდუქციის გამოყენებით დაამტკიცეთ *O/E[n]*, *Shuffle[n]* და *Correct[n]* სქემების სისწორე.

Merge[n] სქემის სისწორეც ინდუქციაზე დაყრდნობით შეიძლება დავამტკიცოთ.
ცხადია, რომ $n = 1$ შემთხვევაში იგი სწორად მუშაობს, რადგან ერთი *CMP* ელემენტისაგან შედგება (რომლის სისწორეშიც გადამოცმების შედეგად შეიძლება დავრწმუნდეთ).

ინდუქციის დაშვება: დავუშვათ, რომ *Merge[i]* სქემა სწორად მუშაობს $i \in \{1, 2, ..., n-1\}$ ინდექსებისათვის.

ინდუქციის ბიჯი: $n \longrightarrow n+1$.

შენიშვნა: ინდუქციის დაშვების თანახმად ვიღებთ: $c_j \geq c_{j+i}$, $d_j \geq d_{j+i}$, $i + j \in \{1, ..., 2^n\}$.

Merge[n] სქემას შემდეგი მნიშვნელოვანი თვისებები აქვს:

- თუ $S_{2j} > S_{2j-1}$, მაშინ ეს შეცდომა სქემის ბოლო ნაწილში - *Correct[n]* სქემაში გამოსწორდება (იხ. ზემოთ მოყვანილი კონსტრუქციის სქემა);
- დავუშვათ, $S_{2j+1} \leq S_{2j}$ და $S_{2j+1} = c_k$, $S_{2j} = d_l$ $(k, l \in N)$.
  ახლა კი განვიხილოთ შემთხვევა S_{2j-m}, $m \in N$. არსებობს ორი შესაძლებლობა:
  1. $s_{2j-m} = c_{k-p} \leq c_{k+1} = s_{2j+1}$;

2. $s_{2j-m} = d_{l-p} \le d_{l+1} = s_{2j+1}$.

ორივე შემთხვევაში ვიღებთ: $S_{2j+1} \le S_{2j}$.

ზემოთ თქმულიდან გამომდინარეობს, რომ *Merge[n]* სქემის სისწორის დამტკიცებისათვის უნდა ვაჩვენოთ უტოლობა $s_{2j+1} \le s_{2j}$.

ზოგადობის დარღვევის გარეშე შეგვიძლია დავუშვათ: $j = 1$, $s_2 = d_1$, $s_3 = c_2$ (მის შემდეგ დამტკიცებას ადვილად განვავრცობთ ყველა ნატურალური რიცხვისათვის $j \in N$).

რადგან ინდუქციის დაშვების თანახმად *Merge[i]* სქემის ყველა ნაწილი სწორად მუშაობს, ვიღებთ:

$$c_1 \in \{a_1, b_2\}; d_1 \in \{a_2, b_1\}; c_2 \in \{a_1, a_3, b_2, b_4\}.$$

აქედან გამომდინარე, მხოლოდ შემდეგი შემთხვევები შეიძლება მივიღოთ:

- $d_1 = b_1$

  1. $c_1 = a_1, c_2 = a_3, \Longrightarrow d_1 = b_1 \ge a_2 \ge a_3 = c_2$, ესე იგი, $\underline{d_1 \ge c_2}$
  2. $c_1 = a_1, c_2 = b_2 \Longrightarrow d_1 = b_1 \ge b_2 = c_2$, ესე იგი $\underline{d_1 \ge c_2}$
  3. $c_1 = b_2, c_2 = a_1 \Longrightarrow d_1 = b_1 \ge b_2 \ge a_1 = c_2$, ესე იგი, $\underline{d_1 \ge c_2}$
  4. $c_1 = b_2, c_2 = b_4 \Longrightarrow d_1 = b_1 \ge b_4 = c_2$, ესე იგი, $\underline{d_1 \ge c_2}$

  შენიშვნა: პირველ ორ შემთხვევაში გამორიცხულია $c_2 = b_4$, ხოლო ბოლო ორში კი $c_2 = a_3$.

- $d_1 = a_2$

  1. $c_1 = a_1, c_2 = a_3 \Longrightarrow d_1 = a_2 \ge a_3 = c_2$, ესე იგი, $\underline{d_1 \ge c_2}$
  2. $c_1 = a_1, c_2 = b_2 \Longrightarrow d_1 = a_2 \ge b_1 \ge b_2 = c_2$, ესე იგი, $d_1 \ge c_2$
  3. $c_1 = b_2, c_2 = a_1 \Longrightarrow d_1 = a_2 \ge d_{i+1} = b_1 \ge b_2 \ge a_1$, ესე იგი $a_2 \ge a_1$ და, ინდუქციის დაშვების თანახმად, $a_2 = a_1 \quad (i > 0)$
  4. $c_1 = b_2, c_2 = b_4 \Longrightarrow d_1 = a_2 \ge b_1 \ge b_4 = c_2$, ესე იგი, $d_1 \ge c_2$

ამით *Merge[n]* სქემის სისწორე სრულად დამტკიცდება.

<u>სავარჯიშო 6.4:</u> გამოიანგარიშეთ *Merge[n]* სქემის ბიჯებისა და ელემენტების რაოდენობა.

<u>სავარჯიშო 6.5:</u> გამოიანგარიშეთ *Sort[n]* სქემის ბიჯებისა და ელემენტების რაოდენობა.

6.2 Column-Sort

ახლა კი განვიხილოთ დალაგების ერთი პარალელური ალგორითმი Columnsort ([4]), რომლის გადატანა ადვილად შეიძლება სხვადასხვა პარალელურ სტრუქტურაზე ([6]).

თვით ეს ალგორითმი წინა ნაწილში განხილული დალაგების განზოგადოებაა. მისი აღწერა შეიძლება მატრიცის ელემენტების დალაგების მაგალითზე.

განვიხილოთ რაღაცა მოცემული ტიპის (მაგ. k ბიტიანი რიცხვების) $r \times s$ მატრიცი Q. ესე იგი, უნდა დავახარისხოთ $N = r \cdot s$ ელემენტი, სადაც $s|r$ და $r \ge 2(s-1)^2$:

$$Q = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,s-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,s-1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{r-1,0} & a_{r-1,1} & \dots & a_{r-1,s-1} \end{pmatrix}$$

ალგორითმის შესრულების შემდეგ ელემენტები ზრდადი თანმიმდევრობით განლაგდება. მისი მუშაობა რვა ძირითადი ნაწილისაგან შედგება. 1,3,5 და 7 ნაწილში პარალელურად ლაგდება მატრიცის სვეტები ზრდადი თანმიმდევრობით. ხოლო 2,4,6 და 8 ნაწილში მატრიცის ელემენტების გარკვეული წესების მიხედვით პერმუტაცია ხდება:

ბიჯები 1, 3, 5, 7: სვეტების დახარისხება

ამ ბიჯებში მატრიცის ყოველი სვეტის ელემენტები პარალელურად დალაგდება;

ბიჯები 2, 4: მატრიცის ტრანსპონირება (შესაბამისად უკუტრანსპონირება):

მეორე ბიჯში სვეტების ელემენტები სტრიქონებში გადანაწილდება. მეოთხე ბიჯი მეორეს შებრუნებულია;

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,s-1} \\ \vdots & \vdots & & \vdots \\ a_{r-1,0} & a_{r-1,1} & \cdots & a_{r-1,s-1} \end{pmatrix} \underset{4}{\overset{2}{\rightleftarrows}} \begin{pmatrix} a_{0,0} & a_{1,0} & \cdots & a_{s-1,0} \\ a_{s,0} & a_{s+1,0} & \cdots & a_{2(s-1),0} \\ \vdots & \vdots & & \vdots \\ a_{r-s,s-1} & a_{r-s+1,s-1} & \cdots & a_{r-1,s-1} \end{pmatrix}$$

ეს ოპერაციები მხოლოდ კვადრატული მატრიცის შემთხვევაში შეესაბამება ტრანსპონირებას, მაგრამ მსგავსების გამო ჩვენ მათ მაინც ტრანსპონირებასა და უკუტრანსპონირებას ვუწოდებთ.

ბიჯები 6, 8: ელემენტთა $\lfloor \frac{r}{2} \rfloor$ პოზიციით ციკლური წაძვრა ქვემოთ (შესაბამისად ზემოთ), თანაც ბოლო სვეტის $\lfloor \frac{r}{2} \rfloor$ ელემენტი დამატებით სვეტში უნდა გადავიდეს. წაძვრის შედეგად გაჩენილი თავისუფალი ადგილები პირველ და ბოლო (დამატებით) სვეტში $-\infty$ (და შესაბამისად $+\infty$) ელემენტებით შეივსება:

$$\left(\begin{array}{|cccc|} \hline a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\ \vdots & \vdots & & \vdots \\ a_{\lfloor \frac{r}{2} \rfloor -1,0} & a_{\lfloor \frac{r}{2} \rfloor -1,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor -1,s-1} \\ \hline a_{\lfloor \frac{r}{2} \rfloor,0} & a_{\lfloor \frac{r}{2} \rfloor,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor,s-1} \\ \vdots & \vdots & & \vdots \\ a_{r-1,0} & a_{r-1,1} & \cdots & a_{r-1,s-1} \\ \hline \end{array}\right) \underset{8}{\overset{6}{\rightleftarrows}} \left(\begin{array}{|cccc||c|} \hline -\infty & a_{\lfloor \frac{r}{2} \rfloor,0} & \cdots & a_{\lfloor \frac{r}{2} \rfloor,s-2} & a_{\lfloor \frac{r}{2} \rfloor,s-1} \\ \vdots & \vdots & & \vdots & \vdots \\ -\infty & a_{r-1,0} & \cdots & a_{r-1,s-2} & a_{r-1,s-1} \\ \hline a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} & +\infty \\ \vdots & \vdots & & \vdots & \vdots \\ a_{\lfloor \frac{r}{2} \rfloor -1,0} & a_{\lfloor \frac{r}{2} \rfloor -1,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor -1,s-1} & +\infty \\ \hline \end{array}\right)$$

ზემოთ მოყვანილი ალგორითმი განვიხილოთ კონკრეტულ მაგალითზე:

$$
\begin{pmatrix} 8 & 22 & 6 \\ 27 & 18 & 21 \\ 5 & 11 & 24 \\ 17 & 23 & 2 \\ 9 & 26 & 19 \\ 25 & 10 & 3 \\ 12 & 13 & 7 \\ 14 & 15 & 4 \\ 1 & 16 & 20 \end{pmatrix}
\xrightarrow{\text{სორტირება}}
\begin{pmatrix} 1 & 10 & 2 \\ 5 & 11 & 3 \\ 8 & 13 & 4 \\ 9 & 15 & 6 \\ 12 & 16 & 7 \\ 14 & 18 & 19 \\ 17 & 22 & 20 \\ 25 & 23 & 21 \\ 27 & 26 & 24 \end{pmatrix}
\xrightarrow{\text{ტრანსპონირება}}
\begin{pmatrix} 1 & 5 & 8 \\ 9 & 12 & 14 \\ 17 & 25 & 27 \\ 10 & 11 & 13 \\ 15 & 16 & 18 \\ 22 & 23 & 26 \\ 2 & 3 & 4 \\ 6 & 7 & 19 \\ 20 & 21 & 24 \end{pmatrix}
\xrightarrow{\text{სორტირება}}
$$

$$
\begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 8 \\ 6 & 7 & 13 \\ 9 & 11 & 14 \\ 10 & 12 & 18 \\ 15 & 16 & 19 \\ 17 & 21 & 24 \\ 20 & 23 & 26 \\ 22 & 25 & 27 \end{pmatrix}
\xrightarrow[\text{ტრანსპონირება}]{\text{უკუ-}}
\begin{pmatrix} 1 & 9 & 17 \\ 3 & 11 & 21 \\ 4 & 14 & 24 \\ 2 & 10 & 20 \\ 5 & 12 & 23 \\ 8 & 18 & 26 \\ 6 & 15 & 22 \\ 7 & 16 & 25 \\ 13 & 19 & 27 \end{pmatrix}
\xrightarrow{\text{სორტირება}}
\begin{pmatrix} 1 & 9 & 17 \\ 2 & 10 & 20 \\ 3 & 11 & 21 \\ 4 & 12 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 18 & 26 \\ 13 & 19 & 27 \end{pmatrix}
\xrightarrow{\text{წაძვრა}}
$$

$$
\begin{pmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 18 & 26 \\ -\infty & 13 & 19 & 27 \\ 1 & 9 & 17 & +\infty \\ 2 & 10 & 20 & +\infty \\ 3 & 11 & 21 & +\infty \\ 4 & 12 & 22 & +\infty \\ 5 & 14 & 23 & +\infty \end{pmatrix}
\xrightarrow{\text{სორტირება}}
\begin{pmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 17 & 26 \\ -\infty & 9 & 18 & 27 \\ 1 & 10 & 19 & +\infty \\ 2 & 11 & 20 & +\infty \\ 3 & 12 & 21 & +\infty \\ 4 & 13 & 22 & +\infty \\ 5 & 14 & 23 & +\infty \end{pmatrix}
\xrightarrow[\text{წაძვრა}]{\text{უკუ-}}
\begin{pmatrix} 1 & 10 & 19 \\ 2 & 11 & 20 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 17 & 26 \\ 9 & 18 & 27 \end{pmatrix}
$$

<u>სავარჯიშო 6.6:</u> გამოიანგარიშეთ ზემოთ მოყვანილი ალგორითმით n ელემენტის დახარისხებისათვის საჭირო ბიჯების რაოდენობა.

ლიტერატურა

[1] V. M. Khrapchenko. *O vremeni slozjenija parallelnogo summatora (On the Time Delay of a Parallel Adder)* Dokladi Akademii Nauk, 19, 1967

[2] A. Schnhage und V. Strassen, *Schnelle Multiplikation groer Zahlen* Computing 7 (1971), pp. 281292

[3] R.E. Ladner and M.J. Fischer. *Parallel Prefix Computation.* Journal of the ACM 27, 831-838, 1980

[4] T. Leighton. *Tight Bounds on the Complexity of Parallel Sorting IEEE* Transactions on Computers, Vol. C34(4):344-354, April 1985

[5] I. Wegwner *Effiziente Algorithmen für grundlegende Funktionen* B.G.Teubner, Stuttgart, 1989

[6] A. Gamkrelidze, Th. Burch. *A Parametrized Sorting System for a Large Set of k-bit Elements* Technical Report, A 04/1998, University of Saarland

[7] D. E. Knuth. *Sorting and Searching, The Art of Computer Programming.* Addison - Wesley, 1998.

[8] A. Gamkrelidze. *Einige Optimierungsmethoden Hierarchischer Schaltkreise* PhD thesis, Universität des Saarlandes, 2001