ალგორითმები და მონაცემთა სტრუქტურები

წრფივი ძებნა. ანალიზი.

ზ. კუჭავა, LaTeX

ამოცანა : მასივში $A[n] = A[a_0, .., a_{n-1}]$ მოვძებნოთ მოცემული მნიშვნელობა ν.

თუ მნიშვნელობა ν მოიძებნა, გამოვიტანოთ მასივის პირველი (უმცირესი) ინდექსი i_0 რომლისთვისაც $\nu = a_{i_0}$. თუ ν არ მოიძებნა, ამოვბეჭდოთ შესაბამისი შეტყობინება. [2]34გვ:30ევ, [3]43გვ:61ევ, [1]396გვ, [4]273გვ, [6]25გვ.

1 მარტივი ძებნა

1.1 ალგორითმი

განვიხილოთ ინდექსი $i = 0$. ვამოწმებთ $\nu = a_i$. თუ ტოლობა სწორია საძიებელი ინდექსი ნაპოვნია. თუ არა ვამოწმებთ i ხომ არ არის ბოლო ინდექსი $n-1$. თუ $i = n-1$ სწორია ვამთავრებთ ამოცანას შესაბამისი შეტყობინებით, თუ არა ვზრდით ინდექს $i = i + 1$ და გადავდივართ პირველ შემოწმებაზე.

1.2 ფსევდოკოდი

ბლოკსქემის 1 შესაბამისი კოდი

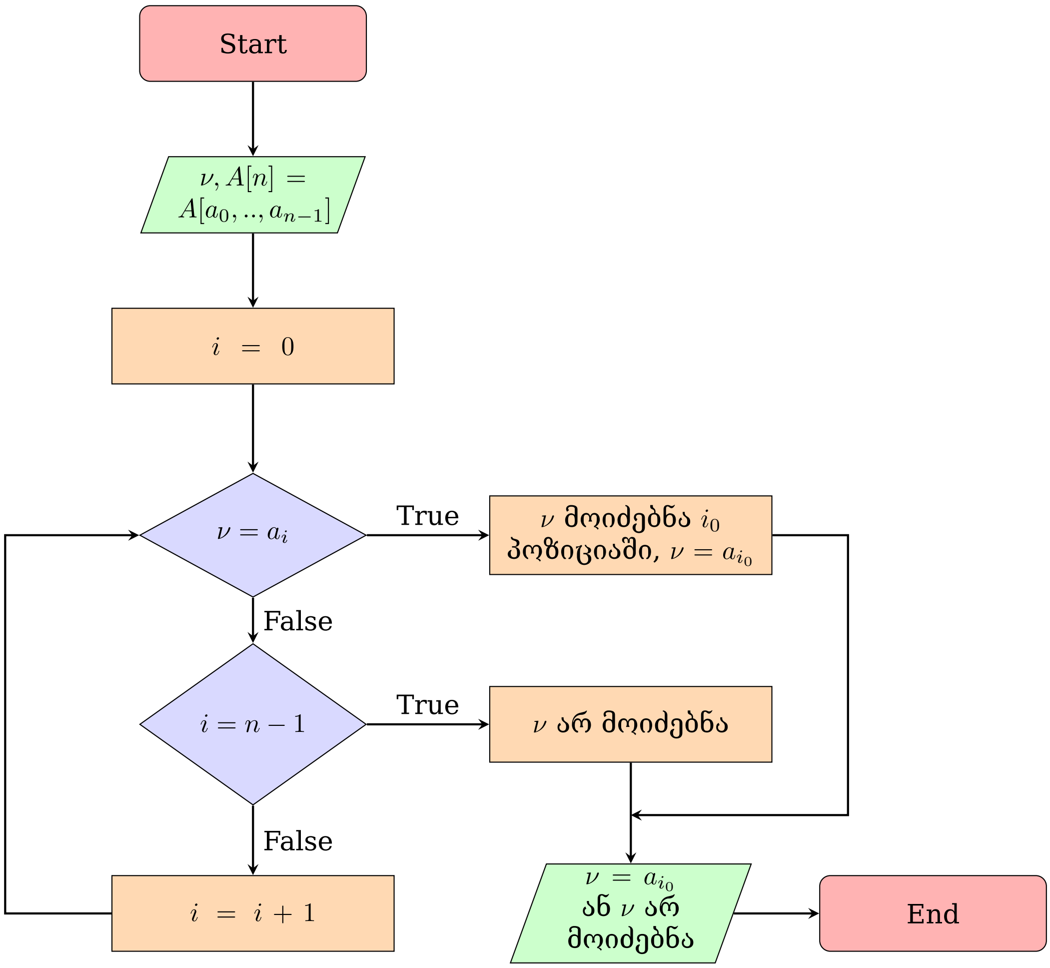
```
        i = 0;
LOOP:   if(v == a[i])
        {
            printf("found value in place %d\n", i);
            return 0;
        }
        else
        {
           if(i == n-1)
           {
            printf("value not found\n");
            return 1;
           }
           i = i + 1;
           goto LOOP;
        }
```

კოდი ციკლის ოპერატორის საშუალებით

```
for(i=0; i < n ;i++)
{
   if(v == a[i])
   {
       printf ( "Found value in place %d.\n" , i );
       return 0 ;
   }
}
printf ( "Value not found.\n" );
return 1 ;
```

1.3 ბლოკსქემა

სურათი 1



სურ. 1: მარტივი ძებნა

1.4 ანალიზი

1.4.1 საუკეთესო შემთვევა

როდესაც საძიებელი მნიშვნელობა $\nu = a_0$, მაშინ ხორციელდება მხოლოდ ერთი შემოწმება. ინდექსის ზრდათა რაოდენობა 0-ია.

1.4.2 უარესი შემთხვევა

როდესაც საძიებელი მნიშვნელობა ν არ არის მასივში $A[n]$, მაშინ ხორციელდება n-ჯერ პირველი შემოწმება და მეორე შემოწმებაც n-ჯერ - სულ შემოწმებათა რაოდენობაა $2 \cdot n$. ინდექსის ზრდათა რაოდენობა $n - 1$-ია.

1.4.3 საშუალო შემთხვევა

განვიხილით ჯერ შემთხვევა როდესაც საძიებელი ელემენტი ν არის მასივში. თუ ν არის i-ურ ადგილას ანუ $\nu = a_i$, მაშინ პირველი შემოწმება ხორციელდება $i+1$-ჯერ და მეორე შემოწმება i-ჯერ. ე.ი. სულ შემოწმებათა რაოდენობაა $2 \cdot i + 1$ და ინდექსის ზრდათა რაოდენობაა i.

მასივში არის ν-ს პოვნის n შესაძლებელი ადგილი, 0-დან $n-1$-მდე და საშუალოს გამოთვლისთვის ვაკეთებთ უმნიშვნელოვანეს დაშვებას, რომ ყველა ადგილი თანაბრად მოსალოდნელია ანუ მოსალოდნელობის ზომა თითოეული ადგილისთვის არის $\frac{1}{n}$. ამიტომ შემოწმებათა რაოდენობისთვის საშუალო გამოითვლება ფორმულით:

$$\sum_{i=0}^{n-1} \frac{2 \cdot i + 1}{n} = \frac{2}{n} \cdot \sum_{i=0}^{n-1} i + 1 = \frac{2}{n} \cdot \frac{n \cdot (n-1)}{2} + 1 = n \tag{1}$$

ახლა განვიხილით შემთხვევა, როდესაც საძიებელი ელემენტი ν შეიძლება იყოს და შეიძლება არც იყოს მასივში. ე.ი. ამ შემთხვევაში გვაქვს სულ $n+1$ შესაძლებელი შემთხვევა. ვიცით, რომ ν ელემენტის არ მოძებნის შემთხვევაში ხორციელდება $2 \cdot n$ შემოწმება.

თუ დავუშვებთ, რომ ν-ს არ ყოფნა მასივში ისევე მოსალოდნელია, როგორც მისი პოვნა ნებისმიერ ადგილას, მაშინ მოსალოდნელობის ზომა თითოეული შემთხვევისთვის იქნება $\frac{1}{n+1}$ და შემოწმებათა რაოდენობისთვის საშუალო გამოითვლება შემდეგნაირად:

$$\sum_{i=0}^{n-1} \frac{2 \cdot i + 1}{n+1} + \frac{2 \cdot n}{n+1} = n + 1 - \frac{1}{n+1} = n + \frac{n}{n+1} \tag{2}$$

ორივე შემთხვევის გაერთიანება შეგვიძლია ფორმულით

$$n + O(1), n \to \infty \tag{3}$$

თუ გვინდა დავთვალოთ საშუალო როგორც შემოწმებათა რაოდენობისთვის, ასევე ინდექსების ზრდათა მიმართ, მაშინ შემოვიღოთ შემოწმების ოპერაციისთვის აღნიშვნა C და ინდექსის ზრდის ოპერაციისთვის აღნიშვნა I. საშუალოსთვის მიიღება ფორმულა

$$\begin{aligned} C \cdot \left(\sum_{i=0}^{n-1} \frac{2 \cdot i + 1}{n+1} + \frac{2 \cdot n}{n+1} \right) + I \cdot \left(\sum_{i=1}^{n-1} \frac{i}{n+1} + \frac{n-1}{n+1} \right) = \\ = C \cdot \left(n + 1 - \frac{1}{n+1} \right) + I \cdot \left(\frac{n}{2} - \frac{1}{n+1} \right) \end{aligned} \tag{4}$$

შენიშვნა 1. კიდევ ერთხელ აღვნიშნოთ, რომ საშუალოს გამოთვლის დროს გაკეთებული იყო დაშვებები საძიებელი ელემენტის სხვადასხვა ვარიანტების "მოსალოდნელობასთან"დაკავშირებით. კერძოდ, ჩვენ დავუშვით, რომ ყველა ვარიანტი თანაბრად მოსალოდნელია.

შენიშვნა 2. ხშირად წრფივი ძებნის ალგორითმის ანალიზის დროს არ ითვლიან ციკლიდან გასვლისთვის საჭირო შემოწმებას (ჩვენ შემთხვევაში მეორე შემოწმება). თუ ჩავთვლით, რომ ყოველ საფეხურზე გვჭირდება მხოლოდ ერთი შემოწმება (ჩვენ შემთხვევაში პირველი), მაშინ საშუალო დაითვლება კიდევ უფრო მარტივად:

თუ ν ელემენტი ნამდვილად არის მასივში, მაშინ საშუალოა

$$\sum_{i=0}^{n-1} \frac{i+1}{n} = \frac{n+1}{2} = \frac{n}{2} + \frac{1}{2} \tag{5}$$

და თუ ν ელემენტი ან არის ან არ არის მასივში მაშინ საშუალოა

$$\sum_{i=0}^{n-1} \frac{i+1}{n+1} + \frac{n}{n+1} = \frac{n}{2} + \frac{n}{n+1} \tag{6}$$

ხშირად ასეთი განხილვა ხდება როდესაც ალგორითმი ჩაწერილია while, for, loop, do და სხვა მსგავსი ციკლის შექმნის რთული კონსტრუქციებით(1.2) და ერთერთი შედარება (ჩვენ შემთხვევაში მეორე შემოწმება) ჩამალულია ასეთ კონსტრუქციაში [3]44-45გვ:62-63ეგვ, [4]273-274გვ.

2 სწრაფი ძებნა

2.1 ალგორითმი

ამოცანა-ში შემოვიტანოთ ახალი, $n+1$ განზომილებიანი, მასივი $B[n+1]$. A მასივის ყველა ელემენტი გადავწეროთ B მასივის პირველ n პოზიციაში, ანუ მივანიჭოთ $\forall i$, $i = \overline{1,n}$ $b_i = a_i$. B მასივის $n+1$-ე პოზიციაში ჩავწეროთ საძიებელი მნიშვნელობა ν, ანუ მივანიჭოთ $b_n = \nu$.

მნიშვნელობა ν ვეძებოთ ახლა მასივში B. არსებითი განსხვავება წინა შემთხვევისგან არის ის, რომ ახლა ν ნამდვილად არის საძიებელ მასივში და აქედან გამომდინარე არ გვჭირდება მასივის საზღვრებს გარეთ გასვლის შემოწმება (წინა ალგორითმის მეორე შემოწმება).

ისევ განვიხილოთ ინდექსი $i = 0$. ვამოწმებთ $\nu = a_i$. თუ ტოლობა სწორია საძიებელი ინდექსი ნაპოვნია. თუ არა, ვზრდით ინდექს $i = i+1$ და გადავდივართ ისევ შემოწმებაზე. ის, რომ შექმნილი ციკლიდან აუცილებლად გამოვალთ და მნიშვნელობა ν მოძებნილი იქნება გარანტირებულია იმით, რომ საძიებელი მნიშვნელობა ν ნამდვილად არის B მასივის $n+1$-ე პოზიციაში.

როდესაც მნიშვნელობა ν ნაპოვნია საკმარისია შევამოწმოთ B მასივის რომელი ინდექსისთვის შესრულდა ტოლობა. თუ ეს ინდექსი n-ია, მაშინ ν არ არის საწყის A მასივში. თუ ინდექსი მკაცრად ნაკლებია n-ზე მაშინ n არის საწყის A მასივში.

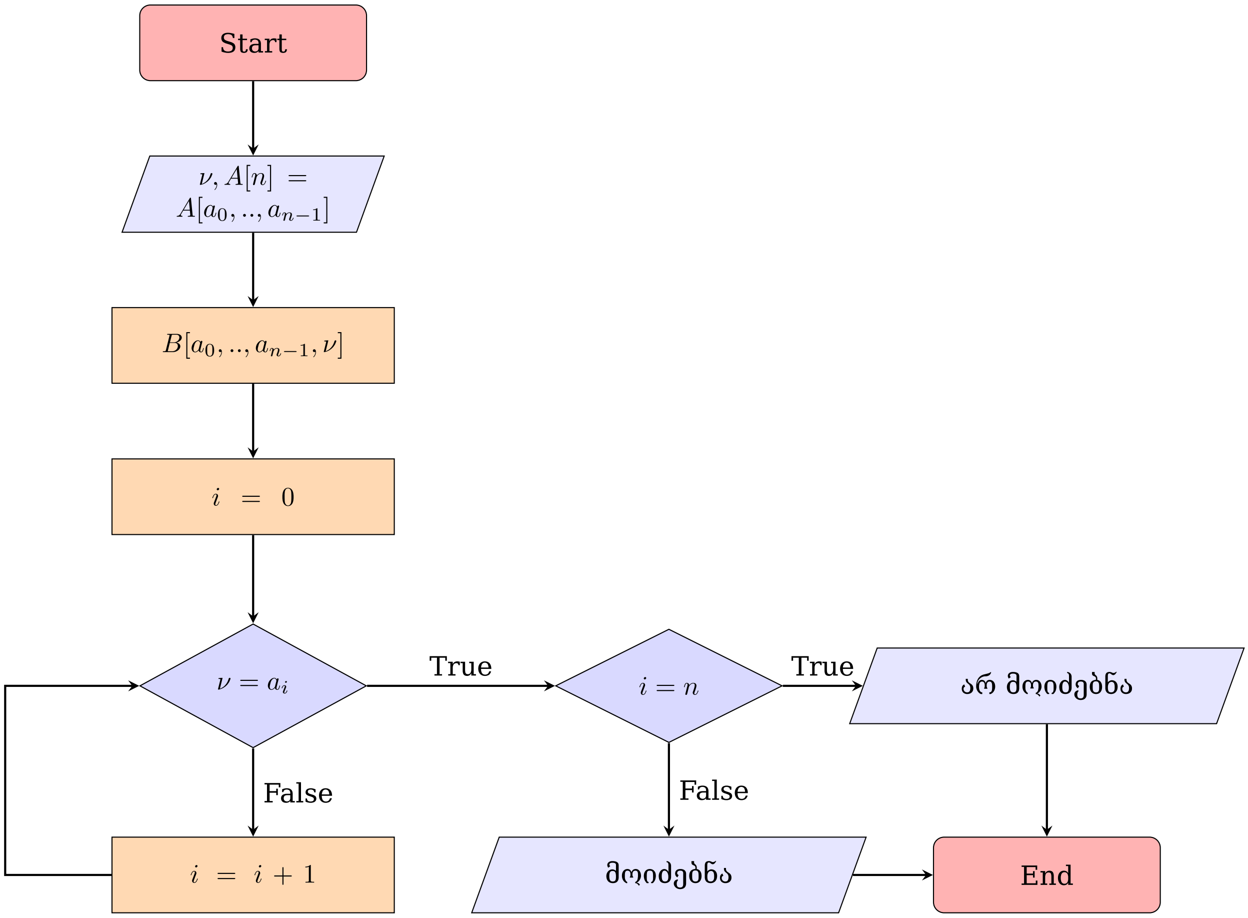
უპირატესობა იმაშია, რომ წინა შემთხვევისგან განსხვავებით მეორე შემოწმება გავიდა ციკლს გარეთ და ამდენად განხორციელდება მხოლოდ ერთხელ. ალგორითმის მორალია - გავიტანოთ ციკლიდან ყველა ოპერაცია, რომელთა გატანაც შესაძლებელია.

სწრაფი ძებნის ალგორითმისთვის ლიტერატურაში გამოიყენება ტერმინები: ბარიერის მეთოდი, ფიქტიური ელემენტის მეთოდი, sentinel მეთოდი, სტოპერის მეთოდი [2]34გვ:30ეგვ, [1]397-398გვ, [7] 103-104გვ

2.2 ფსევდოკოდი

```
        i = 0;
LOOP:   if(v == b[i])
          goto END;
        else
        {
          i = i + 1;
          goto LOOP;
        }
END:    if(i == n )
        {
          printf("value not found\n");
          return 1;
        }
        printf("found value in place %d\n", i);
        return 0;
```

2.3 ბლოკსქემა



სურ. 2: სწრაფი ძებნა

2.4 ანალიზი

2.4.1 საუკეთესო შემთვევა

საუკეთესო არის შემთხვევა როდესაც საძიებელი მნიშვნელობა ν არის B მასივის, ანუ რაც იგივეა A მასივის, პირველ პოზიციაში. ასეთ შემთხვევაში გვექნება ორი შედარების და 0 ინდექსის ზრდის ოპერაციები.

2.4.2 უარესი შემთხვევა

როდესაც საძიებელი მნიშვნელობა არ არის A მასივში, ანუ ν არის B მასივის მხოლოდ ბოლო, $n+1$-ე პოზიციაში ($b_n = \nu$), მაშინ განხორციელდება $n+2$ შედარების და n ინდექსის ზრდის ოპერაციები.

2.4.3 საშუალო შემთხვევა

დავუშვათ, რომ ν არის B მასივის i პოზიციაში, სადაც $0 \leqslant i \leqslant n$. ამ შემთხვევაში განხორციელდება $i+1+1 = i+2$ შედარება. ამგვარად, საძიებელი ელემენტის $n-1$ პოზიციაში ყოფნისას გვექნება $n+1$ შედარება. ამდენივე შედარება იქნება, როდესაც პირველ $n-1$ პოზიციაში საძიებელი ელემენტი არა გვაქვს. i პოზიციაში ν-ს ყოფნის შემთხვევაში გვექნება $i-1$ ინდექსის ზრდა როდესაც $1 \leqslant i \leqslant n$.

თუ დავუშვებთ, რომ ν-ს პოვნა თანაბრად მოსალოდნელია B მასივის ყველა $n+1$ პოზიციაში, მაშინ საშუალო შედარებათა რაოდენობა დაითვლება ფორმულით:

$$\sum_{i=0}^{n-1} \frac{i+2}{n+1} + \frac{n+2}{n+1} = \frac{1}{n+1} \cdot \sum_{i=0}^{n-1} (i+2) + O(1) \tag{7}$$

შეგვიძლია საშუალო ჩავწეროთ ფორმულით

$$\frac{n}{2} + O(1), n \to \infty \tag{8}$$

როგორც მოსალოდნელი იყო საშუალოდ სწრაფი ძებნის ალგორითმისთვის ოპერაციათა რაოდენობა დაახლოებით 2-ჯერ ნაკლებია ვიდრე წრფივი ძებნისთვის ფორმულა (2)-ში .

შენიშვნა 1. კიდევ ერთხელ იყო გაკეთებული დაშვება, რომ საძიებელი ელემენტის პოვნის სხვადასხვა ვარიანტები თანაბრად მოსალოდნელია.

შენიშვნა 2. წრფივი და სწრაფი ძებნის ალგორითმების უფრო ზუსტი შედარებისთვის საჭიროა გავითვალისწინოთ A მასივიდან B მასივის შექმნისთვის საჭირო ოპერაციების რაოდენობაც. აღსანიშნავია, რომ B ტიპის მასივის შექმნა შესაძლებელია ახალი მასივის შექმნის გარეშეც, ანუ A მასივის გამოყენებით.

შენიშვნა 3. ცალკე დავითვალოთ ინდექსის ზრდათა რაოდენობა საშუალო შემთხვევისთვის. როგორც ზემოთ უკვე ნახსენები იყო თუ ν არის B მასივის i პოზიციაში, მაშინ განხორციელდება $i-1$ ინდექსის ზრდა. საშუალო დაითვლება ფორმულით:

$$\sum_{i=1}^{n} \frac{i-1}{n+1} + \frac{n}{n+1} = \frac{1}{n+1} \cdot \sum_{i=1}^{n} i = \frac{n}{2} \tag{9}$$

3 ზესწრაფი ძებნა

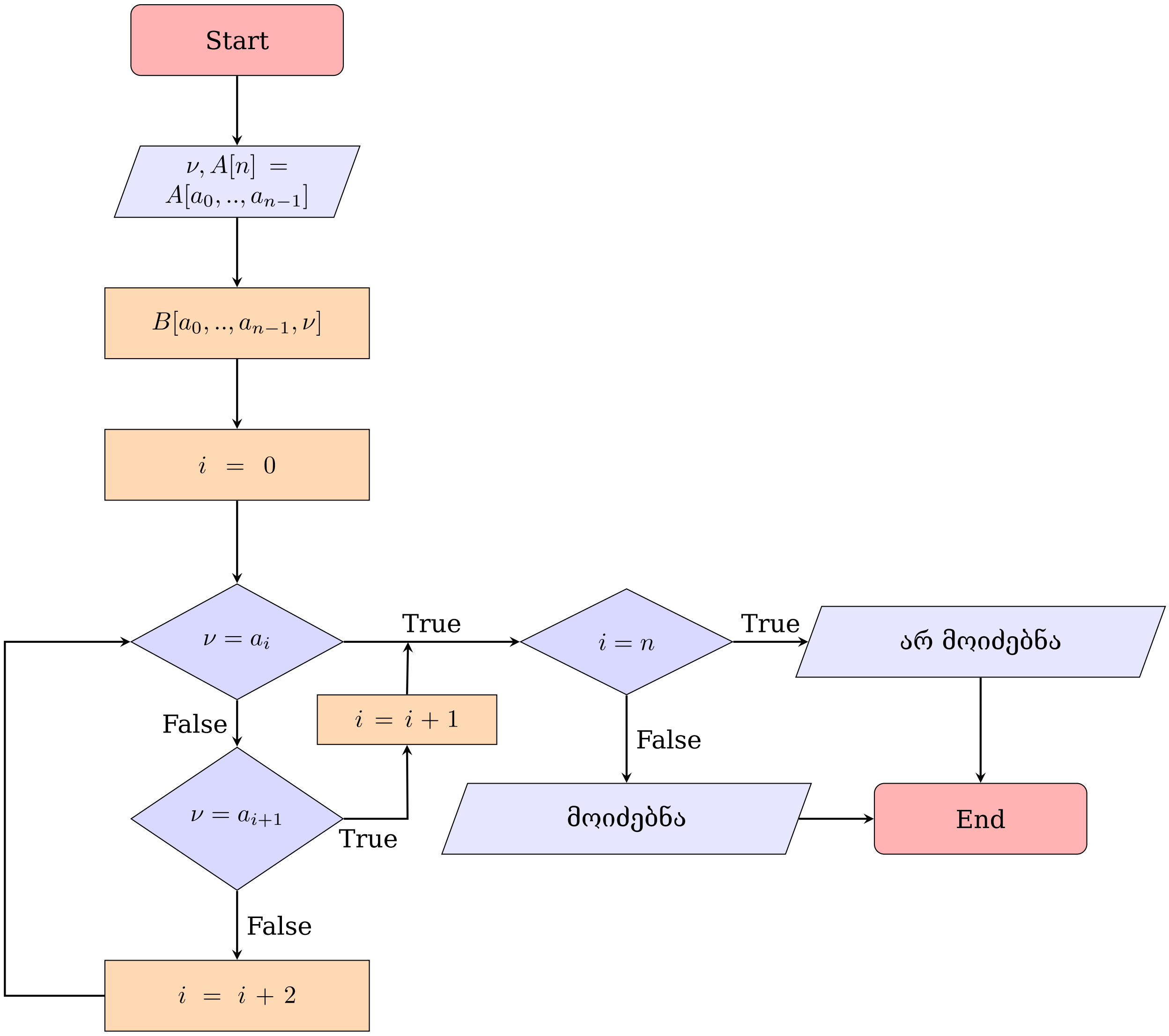
3.1 ალგორითმი

სწრაფი ალგორითმის (2.1) კიდევ უფრო აჩქარება შეიძლება შემდეგნაირად: დავუმატოთ ციკლს შიგნით, ინდექსის გაზრდამდე, კიდევ ერთი შემოწმება ინდექსით ერთით მეტ ელემენტთან რაც, ცხადია, არ გამოიწვევს B მასივის გარეთ გასვლას. ციკლში ინდექსის ზრდა უკვე 2-ით იქნება.

3.2 ფსევდოკოდი

```
        i = 0;
LOOP:   if(v == b[i])
            goto END;
        else
        {
            if(v == b[i + 1])
                goto END;
            else
            {
                i = i + 2;
                goto LOOP;
            }
        }
END:   if(i == n )
        {
            printf("value not found\n");
            return 1;
        }
        printf("found value in place %d\n", i);
        return 0;
```

3.3 ბლოკსქემა

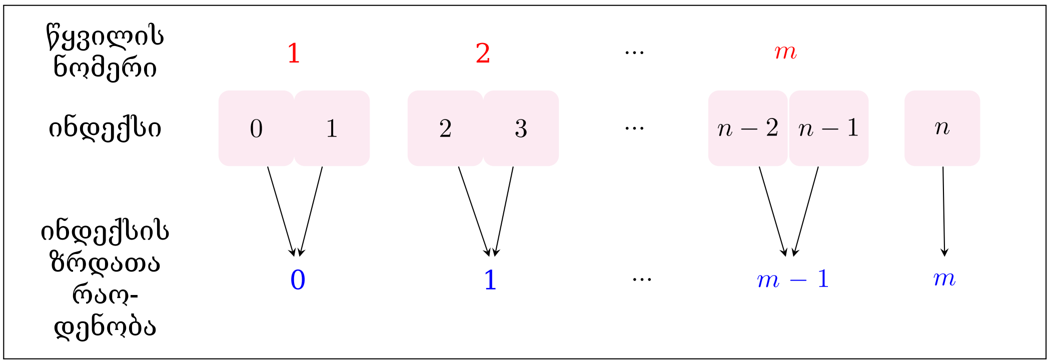


სურ. 3: ზესწრაფი ძებნა

3.4 სირთულის ანალიზი

გასაგებია, რომ ამ ალგორითმისთვის შედარებათა რაოდენობა იქნება იგივე რაც სწრაფი ძებნის (2.1) შემთხვევისთვის. ამდენად გაუმჯობესება შეეხება მხოლოდ ინდექსის ზრდათა რაოდენობას.

განვიხილოთ ორი შემთხვევა:



სურ. 4: $|A|$ ლუწი, $|B|$ კენტი

შემთხვევა 1: დავუშვათ საწყის A მასივში არის ლუწი წევრთა რაოდენობა, სურათი 4, ანუ $n = 2 \cdot m$, სადაც $n = \overline{2, \infty}$ და $m = \overline{1, \frac{n}{2}}$. B მასივში, შესაბამისად, იქნება ელემენტთა კენტი რაოდენობა. B მასივი წარმოადგენს m რაოდენობის წყვილს და ერთ, საძიებელ მნიშვნელობა ν-ს.

პირველი წყვილი წარმოიდგინება ინდექსებით $(0, 1)$ და თუ საძიებელი ელემენტი რომელიმე ამ პოზიციაშია, მაშინ მას შეესაბამება ინდექსის ზრდათა 0 რაოდენობა. მეორე წყვილი წარმოიდგინება ინდექსებით $(2, 3)$ და თუ საძიებელი ელემენტი რომელიმეშია ამ პოზიციათაგან, მაშინ მას შეესაბამება 1 ინდექსის ზრდათა რაოდენობა.

ზოგადად განვიხილოთ $(i, i+1)$-ე წყვილი, $i = \overline{0, n-2}$. თუ საძიებელი ელემენტი ერთერთ ამ პოზიციაშია, მაშინ ამ სიტუაციას შეესაბამება $\left\lfloor \frac{i}{2} \right\rfloor = \frac{i}{2}$ ინდექსის ზრდათა რაოდენობა და ამ წყვილის ნომერია $\frac{i}{2} + 1$.

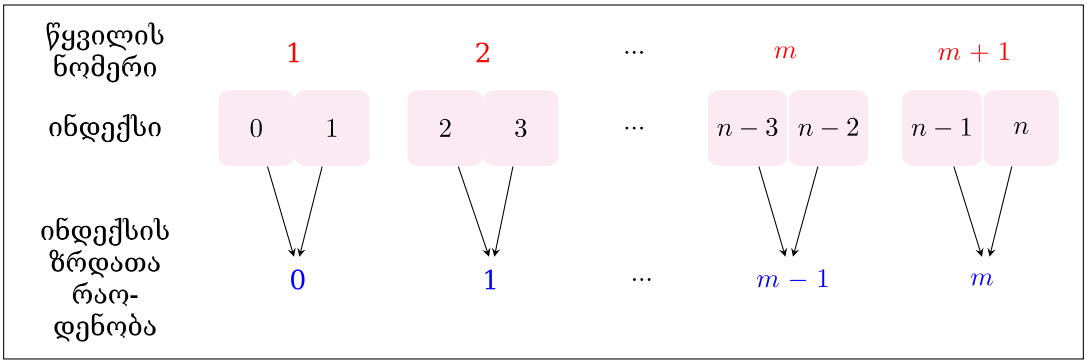
ბოლო წყვილია $(n-2, n-1)$. ანუ, ესაა m-ური წყვილი $m = \frac{n}{2}$-ის, და თუ საძიებელი ელემენტი რომელიმე ამ პოზიციაშია, მაშინ მას შეესაბამება ინდექსის ზრდათა $m - 1$ რაოდენობა.

B მასივის ბოლო, n-ური ელემენტისთვის ინდექსის ზრდათა რაოდენობაა m.

საშუალო დაითვლება ფორმულით:

$$\frac{1}{n+1} \cdot \left[2 \cdot 0 + 2 \cdot 1 + 2 \cdot 2 + \ldots + 2 \cdot (m-1) + m\right] =$$

$$= \frac{1}{n+1} \cdot \left[2 \cdot \sum_{i=1}^{m-1} i + m\right] = \frac{m^2}{n+1} = \frac{n^2}{4 \cdot (n+1)} = \frac{n}{4} - \frac{1}{4} + \frac{1}{4 \cdot (n+1)} \quad (10)$$



სურ. 5: $|A|$ კენტი, $|B|$ ლუწი

შემთხვევა 2: დავუშვათ საწყის A მასივში არის კენტი წევრთა რაოდენობა, სურათი 5, ანუ $n-1=2\cdot m$, სადაც $n=\overline{1,\infty}$ და $m=\overline{0,\frac{n-1}{2}}$. B მასივში, შესაბამისად, იქნება ელემენტთა ლუწი რაოდენობა. B მასივი, ამ შემთხვევაში, წარმოადგენს $m+1$ რაოდენობის წყვილს, სადაც ბოლო წყვილის მეორე ელემენტია საძიებელი მნიშვნელობა ν.

წყვილითვის $(i,i+1)$-ის ინდექსის ზრდათა რაოდენობაა $\frac{i}{2}$ და წყვილის ნომერია $\frac{i}{2}+1$. საშუალო დაითვლება ფორმულით:

$$\frac{1}{n+1}\cdot\Big[2\cdot 0+2\cdot 1+2\cdot 2+\ldots+2\cdot(m-1)+2\cdot m\Big]=$$
$$=\frac{1}{n+1}\cdot\Big[2\cdot\sum_{i=1}^{m}i\Big]=\frac{m\cdot(m+1)}{n+1}=\frac{(n+1)\cdot(n-1)}{4\cdot(n+1)}=\frac{n-1}{4}=\frac{n}{4}-\frac{1}{4}\quad(11)$$

ორივე შემთხვევის გაერთიანება შეგვიძლია ფორმულით

$$\frac{n}{4}+O(1),n\to\infty\quad(12)$$

როგორც მოსალოდნელი იყო სწრაფ ძებნასთან შედარებით, (9), ზესწრაფ ძებნაში ინდექსის ზრდის ოპერაციათა რაოდენობა საშუალოდ დაახლოებით ორჯერ ნაკლებია.

3.5 ჯამის ორი ოპერაციის შედარება.

ანალიზის დაწყებისთვის დარწმუნებული უნდა ვიყოთ, რომ მეორე ელემენტის შემოწმებაში, a_{i+1} ელემენტის შედარების დროს ν მნიშვნელობასთან, ინდექსის $(i+1)$-ს განხილვა არსებითად განსხვავდება ციკლის ინდექსის ზრდის ოპერაციისგან, ანუ ახალი ოპერაციის შემოტანა არ ხდება.

ამისათვის ჯერ გადავწეროთ ზესწრაფი ძებნის ალგორითმი C პროგრამირების ენაზე და მერე ვნახოთ შესაბამისი ასემბლერის კოდი.

```
1  #include <stdio.h>
2  int main()
3  {
4      int b[] = {1, 2, 3, 5, 5, 6, 7, 8, 9, 5};
5      int v = 5;
6      size_t i = 0;
7      while (v != b[i])
8      {
9          if(v == b[i+1])
10         {
11             i++;
12             break;
13         }
14         i = i + 2;
15     }
16
17     if(i == 9)
18     {
19         printf ( "Value not found.\n" );
20         return 1 ;
21     }
22     printf ( "Found value in place %lu.\n" , i );
23     return 0 ;
24 }
```

როგორც ვხედავთ ინდექსის ზრდის ოპერაციები, რომელთაგან ჩვენთვის საინტერესო მხოლოდ მეორეა, არის მე-11 და მე-14 სტრიქონებში და მასივში ინდექსის ზრდა მე-9 სტრიქონში.

შესაბამისი ასემბლერის კოდი

```
1  00000000000006f0 <main>:
2  6f0: push   %rbp            # rbp 0x7fffffffdfe0  Stack base pointer
3  6f1: mov    %rsp,%rbp
4  6f4: sub    $0x40,%rsp      # rsp 0x7fffffffdfa0  Stack pointer
5  6f8: movl   $0x1,-0x40(%rbp)  # 1 -> b[0] = %rbp-0x40
6  6ff: movl   $0x2,-0x3c(%rbp)
7  706: movl   $0x3,-0x38(%rbp)
8  70d: movl   $0x5,-0x34(%rbp)
9  714: movl   $0x5,-0x30(%rbp)
10 71b: movl   $0x6,-0x2c(%rbp)
11 722: movl   $0x7,-0x28(%rbp)
12 729: movl   $0x8,-0x24(%rbp)
13 730: movl   $0x9,-0x20(%rbp)
14 737: movl   $0x5,-0x1c(%rbp)  # 5 -> b[9] = %rbp-0x1c
15 73e: movl   $0x5,-0xc(%rbp)   # v = 5 -> %rbp-0xc
16 745: movq   $0x0,-0x8(%rbp)   # i = 0 -> %rbp-0x8
17 74c:
18 74d: jmp    76c <main+0x7c>
19 74f: mov    -0x8(%rbp),%rax             # i = %rbp-0x8 -> %rax
20 753: add    $0x1,%rax                   # i+1 = 0x1+%rax -> %rax
21 757: mov    -0x40(%rbp,%rax,4),%eax     # b[i+1] = %rbp-0x40+4*%rax -> %eax
22 75b: cmp    -0xc(%rbp),%eax         # v= %rbp-0xc დარდება b[i+1] = %eax
23 75e: jne    767 <main+0x77>
24 760: addq   $0x1,-0x8(%rbp)         # i + 1 = %rbp-0x8+$0x1 -> i = %rbp-0x8
25 765: jmp    779 <main+0x89>
26 767: addq   $0x2,-0x8(%rbp)         # i + 2 = %rbp-0x8+0x2 -> %rbp-0x8
27 76c: mov    -0x8(%rbp),%rax         # i = %rbp-0x8 -> %rax
28                                     #(%eax aris %rax-is მარცხენა 32biti)
29 770: mov    -0x40(%rbp,%rax,4),%eax     # b[i] = %rbp-0x40+4*%rax -> %eax
30 774: cmp    -0xc(%rbp),%eax             # v= %rbp-0xc დარდება b[i] = %eax
31 777: jne    74f <main+0x5f>
32 779: cmpq   $0x9,-0x8(%rbp)             # i = %rbp-0x8 დარდება $0x9
33 77e: jne    793 <main+0xa3>
34 780: lea    0xbd(%rip),%rdi
35 787: callq  590 <puts@plt>
36 78c: mov    $0x1,%eax
37 791: jmp    7b0 <main+0xc0>
38 793: mov    -0x8(%rbp),%rax
39 797: mov    %rax,%rsi
40 79a: lea    0xb4(%rip),%rdi
41 7a1: mov    $0x0,%eax
42 7a6: callq  5a0 <printf@plt>
43 7ab: mov    $0x0,%eax
44 7b0: leaveq
45 7b1: retq
46 7b2: nopw   %cs:0x0(%rax,%rax,1)
47 7b9:
48 7bc: nopl   0x0(%rax)
```

როგორც მე-16 სტრიქონიდან ვხედავთ ინდექსი ინახება მეხსიერებაში და მისი ზრდა ხდება 24(ეს ზრდა არაა ჩვენი ინტერესის საგანი) და 26 სტრიქონებში შესა-

ბამისი მუდმივების მეხსიერებისთვის მიმატებით. ხოლო მასივის ინდექსის ზრდა ხდება რეგისტრში 20 სტრიქონში. ამდენად, შეკრების ოპერაციები განსხვავდება.

C პროგრამირების ენაში ცნობილია Storage-class specifiers([8]109გვ) მეხსიერების სპეციფიკატორები, რომელთაგან სპეციფიკატორი register([8]110გვ) განკუთვნილია ობიექტზე ყველაზე სწრაფი მიმართვის არჩევისთვის. თუ ახლა C კოდის მე-6 სტრიქონს გადავწერთ როგორც

```
register size_t i = 0;
```

მაშინ ასემბლერის კოდი მიიღებს შემდეგ სახეს

```
1  6f5:  sub    $0x38,%rsp
2  6f9:  movl   $0x1,-0x40(%rbp)
3  700:  movl   $0x2,-0x3c(%rbp)
4  707:  movl   $0x3,-0x38(%rbp)
5  70e:  movl   $0x5,-0x34(%rbp)
6  715:  movl   $0x5,-0x30(%rbp)
7  71c:  movl   $0x6,-0x2c(%rbp)
8  723:  movl   $0x7,-0x28(%rbp)
9  72a:  movl   $0x8,-0x24(%rbp)
10 731:  movl   $0x9,-0x20(%rbp)
11 738:  movl   $0x5,-0x1c(%rbp)
12 73f:  movl   $0x5,-0x14(%rbp)
13 746:  mov    $0x0,%ebx
14 74b:  jmp    764 <main+0x74>
15 74d:  lea    0x1(%rbx),%rax
16 751:  mov    -0x40(%rbp,%rax,4),%eax
17 755:  cmp    -0x14(%rbp),%eax
18 758:  jne    760 <main+0x70>
19 75a:  add    $0x1,%rbx
20 75e:  jmp    76d <main+0x7d>
21 760:  add    $0x2,%rbx
22 764:  mov    -0x40(%rbp,%rbx,4),%eax
23 768:  cmp    -0x14(%rbp),%eax
24 76b:  jne    74d <main+0x5d>
25 76d:  cmp    $0x9,%rbx
26 771:  jne    786 <main+0x96>
27 773:  lea    0xba(%rip),%rdi
28 77a:  callq  590 <puts@plt>
29 77f:  mov    $0x1,%eax
30 784:  jmp    79f <main+0xaf>
31 786:  mov    %rbx,%rsi
32 789:  lea    0xb5(%rip),%rdi
33 790:  mov    $0x0,%eax
34 795:  callq  5a0 <printf@plt>
35 79a:  mov    $0x0,%eax
36 79f:  add    $0x38,%rsp
37 7a3:  pop    %rbx
38 7a4:  pop    %rbp
39 7a5:  retq
```

ამჯერად ინდექსი ინახება რეგისტრში მე-13სტრიქონში და მისი ზრდა ციკლში 19(რომელიც არაა ჩვენი ინტერესის საგანი) და 21 სტრიქონებში ხდება ისევ რეგისტრში შეკრებით ისევე, როგორც მასივში ინდექსის ზრდა მე-15 სტრიქონში. ამდენად, ამ შემთხვევაში შეკრების ოპერაციები არ განსხვავდება და ერთადერთი

მოგება იტერაციების რაოდენობის შემცირებაა.

საჭიროა აღინიშნოს, რომ მეხსიერების სპეციფიკატორისთვის register([8]110გვ) ციტირებული C პროგრამირების ენის სტანდარტი არ განსაზღვრავს ობიექტისთვის აუცილებლად რეგისტრის დანიშვნას, არამედ მოითხოვს, რომ მიმართვა ობიექტზე იყოს იმდენად სწრაფი რაც შესაძლებელია და ამდენად მისი ეფექტურობა დამოკიდებულია იმპლემენტაციაზე.

შენიშვნა. 2-ზე მეტი მასივის ელემენტის შემოწმების შემთხვევა არის [5]-ზე.

4 საშუალო არათანაბარი მოსალოდნელობის პირობებში

4.1 წრფივი ძებნა

წრფივი ძებნის შედარებათა რაოდენობის საშუალოს (2) გამოთვლის დროს არსებითად ვერყდნობოდით დაშვებას, რომ საძიებელი ელემენტის ძებნისთვის ყველა შემთხვევა თანაბრად მოსალოდნელია ანუ მასივის თითოეულ პოზიციაში საძიებელი ელემენტის მოძებნის მოსალოდნელობის ზომა არის იგივე რაც საძიებელი ელემენტის ვერ მოძებნის მოსალოდნელობის ზომა და არის $\frac{1}{n+1}$.

მაგრამ ყოველთვის ეს დაშვება არ არის სამართლიანი. განვიხილოთ შემთხვევა, მაგალითად, როდესაც ნამვილად ვიცით, რომ საძიებელი ელემენტი მასივის პირველივე, ე.ი. ნულოვან, ინდექსშია: $a[0] = \nu$. მაშინ წილების/ალბათობების შესაბამისი განაწილებაა $p_0 = 1, p_1 = 0, \cdots, p_n = 0, p_{n+1} = 0$ და თუ გავიხსენებთ, რომ შედარებების სირთულის ფუნქციის მნიშვნელობებია $x_0 = 1, x_1 = 3, \cdots, x_n = 2n-1, x_{n+1} = 2n$, მაშინ საშუალო ამ შემთხვევისთვის გამოდის

$$Ef_c = 1 \cdot 1 + \sum_{i=1}^{n-1}(2 \cdot i + 1) \cdot 0 + 2n \cdot 0 = 1 \tag{13}$$

ანუ მოსალოდნელი მნიშვნელობა დაემთხვა პირველ პოზიციაში საძიებელი ელემენტის პოვნისთვის საჭირო შედარებების რაოდენობას.

თუ ვიცით, რომ საძიებელი ელემენტი ნამდვილად მხოლოდ მასივის მეორე ელემენტშია, ანუ, ინდექსი 1-ის, $a[1] = \nu$, მაშინ განაწილება იქნება $p_0 = 0, p_1 = 1, p_3 = 0, \cdots, p_n = 0, p_{n+1} = 0$ და საშუალოსთვის მივიღებთ

$$Ef_c = 1 \cdot 0 + 3 \cdot 1 + \sum_{i=2}^{n-1}(2 \cdot i + 1) \cdot 0 + 2n \cdot 0 = 3 \tag{14}$$

ამგვარად, ამ შემთხვევაში, მოსალოდნელი მნიშვნელობა დაემთხვა მეორე პოზიციაში საძიებელი ელემენტის პოვნისთვის საჭირო შედარებების რაოდენობას.

ახლა დავუშვათ ვიცით, რომ საძიებელი ელემენტი ნამდვილად არ არის მოცემულ მასივში. მაშინ შესაბამისი განაწილებაა $p_0 = 0, p_1 = 0, \cdots, p_n = 0, p_{n+1} = 1$ და, ამგვარად, მოსალოდნელი მნიშვნელობაა

$$Ef_c = \sum_{i=0}^{n-1} (2 \cdot i + 1) \cdot 0 + 2n \cdot 1 = 2n \tag{15}$$

განვიხილოთ ცოტა უფრო რთული შემთხვევაც როდესაც ν-ს მასივში მოძებნის შემთხვევის მოსალოდნელობა ისეთივეა, როგორც მისი ვერ მოძებნის შემთხვევის მოსალოდნელობა, ანუ ორივე არის $\frac{1}{2}$. გამოთვლების ჩატარებისთვის, აგრეთვე, დავუშვათ, რომ მასივის ყოველ პოზიციაში მოძებნა არის თანაბრად მოსალოდნელი ანუ მასივის ყოველ პოზიციაში საძიებელი ელემენტის პოვნის მოსალოდნელობა არის $\frac{1}{2n}$ ანუ განაწილებაა $p_0 = \frac{1}{2n}, p_1 = \frac{1}{2n}, \cdots, p_n = \frac{1}{2n}, p_{n+1} = \frac{1}{2}$. მაშინ 2 ფორმულა ღებულობს სახეს

$$Ef_c = \sum_{i=0}^{n-1} (2 \cdot i + 1) \cdot \frac{1}{2n} + 2n \cdot \frac{1}{2} = \frac{1}{2n}\left(2\sum_{i=0}^{n-1} i + n\right) + \frac{2 \cdot n}{2} = \frac{n}{2} + n = \frac{3}{2} \cdot n \tag{16}$$

4.2 სწრაფი წრფივი ძებნა

დავუბრუნდეთ სწრაფი წრფივი ძებნის ალგორითმის საშუალოს განხილვას არათანაბარი მოსალოდნელობის პირობებში.

საშუალო შედარებათა რაოდენობა, (7), დათვლილი გვქონდა შემთხვევისთვის, როდესაც ν-ს პოვნა თანაბრად მოსალოდნელი იყო B მასივის ყველა $n+1$ პოზიციაში, ანუ ν-ს პოვნა A მასივის ყოველ n პოზიციაში ისევე იყო მოსალოდნელი, როგორც მისი საერთოდ ვერ პოვნა A-ში და მოსალოდნელობის ზომა ყველა შესაძლო შემთხვევისთვის იყო $\frac{1}{n+1}$.

ახლა განვიხილოთ შემთხვევა, როდესაც ν-ს პოვნა ისევეა მოსალოდნელი, როგორც მისი ვერ პოვნა - ანუ მოსალოდნელობის ზომა პოვნისთვის და ვერ პოვნისთვის არის ერთი და იგივე და უდრის $\frac{1}{2}$-ს. ანუ, მოსალოდნელობის ზომა B მასივის ერთი, უკანასკნელი, ინდექსისთვის n არის $\frac{1}{2}$ და იგივეა რაც B მასივის ინდექსებისთვის 0-დან $n-1$-ის ჩათვლით ყველა ინდექსისათვის ერთად.

საშუალო შედარებათა რაოდენობისთვის გვექნება ფორმულა:

$$\frac{1}{2 \cdot n} \cdot \sum_{i=0}^{n-1} (i+2) + \frac{n+2}{2} = \frac{3n}{4} + O(1), n \to \infty \tag{17}$$

განვიხილოთ კიდევ ერთი ბუნებრივი მაგალითი როდესაც საძიებელი ელემენტის პოვნა პირველ, 0-ან, პოზიციაში გაცილებით უფრო მოსალოდნელია ვიდრე ყველა სხვა პოზიციაში მისი პოვნა ან საერთოდ ვერ პოვნა. სიზუსტისთვის დავუშვათ, რომ პირველ პოზიციაში ν-ს პოვნა მოსალოდნელია, მაგალითად, $\dfrac{9}{10}$ ზომით და ამდენად ყველა სხვა ვარიანტისთვის დარჩა მოსალოდნელობის ზომა $\dfrac{1}{10}$.

გვექნება:

$$\frac{9}{10} \cdot 2 + \frac{1}{10 \cdot n} \cdot \sum_{i=1}^{n-1} (i+2) + \frac{n+2}{10 \cdot n} = \frac{1}{20} \cdot n + O(1), n \to \infty \tag{18}$$

მივიღეთ თანაბარი მოსალოდნელობის შემთხვევასთან შედარებით, (7), საშუალოდ დაახლოებით 10-ჯერ ნაკლები ოპერაციათა რაოდენობა.

ზოგად შემთხვევაში უნდა განვიხილოთ ალგორითმის სირთულის ფუნქციის მუდმივობის სიმრავლეები და მათი შესაბამისი მოსალოდნელობები.

თუ, მაგალითად, $T(n)$ სირთულის ფუნქცია ღებულობს მნიშვნელობებს $x_1, x_2, ..., x_m$ და x_i-ის, $i = \overline{1,m}$, წინასახის მოსალოდნელობის ზომა არის p_i, სადაც $0 \leqslant p_i \leqslant 1, \sum_{i=1}^{m} p_i = 1$, მაშინ T-ს საშუალო დაითვლება ფორმულით

$$ET = \sum_{i=1}^{m} x_i \cdot p_i \tag{19}$$

ბევრი საინტერესო და პრაქტიკული არათანაბარი მოსალოდნელობის მაგალითი განხილული არის მონოგრაფიაში [1]399-403გვ.

ლიტერატურა

[1] Donald Ervin Knuth, *The Art of Computer Programming*, Volume 3, Second Edition

[2] N. Wirth, *Algorithms and Data Structures*

[3] Jeffrey J. McConnell, *Analysis of Algorithms: an Active Learning Approach* , 2001

[4] Paul Deitel, Harvey Deitel, *C How to Program with an introduction to C++*, 8-th Edition

[5] evrika

[6] G.H. Gonnet, R. Baeza-Yates *Handbook of Algorithms and Data Structures In Pascal and C* , Second Edition, 1991

[7] Ю.Ю.Громов, С.И.Татаренко *ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ*, Тамбов, 1994

[8] n1570-C11.pdf (ISO/IEC 9899:2011)