Kilka dokładnych rozwiązań zadania egzaminacyjnego

Antoni Kościelski

12 lutego 2018

1 Sformułowanie zadania

Rangą elementu $x[i]$ w ciągu $x[0], x[1], \ldots, x[n-1]$ (liczb całkowitych) nazywamy liczbę elementów tego ciągu mniejszych od $x[i]$.

Napisz funkcję lub algorytm realizujący następującą specyfikację:

Wejście:

parzysta liczba naturalna $n > 0$ i tablica liczb całkowitych x taka, że

$$x[0] \leqslant x[1] \leqslant \ldots \leqslant x[n/2-1] \;\text{ oraz }\; x[n/2] \leqslant x[n/2+1] \leqslant \ldots \leqslant x[n-1].$$

Wyjście:

ciąg liczb $r[0], r[1], \ldots, r[n-1]$, w którym $r[i]$ jest rangą $x[i]$ w ciągu $x[0], x[1], x[2], \ldots, x[n-1]$.

Opisz ideę swojego rozwiązania, uzasadnij poprawność i oszacuj złożoność obliczeniową.

2 Rozwiązania o złożoności $O(n^2)$

2.1 Algorytm naturalny

Zadania (lub problemy) można zwykle rozwiązać używając algorytmów, które określam jako naturalne, czyli wynikające bezpośrednio ze sformułowania lub definicji. W naszym przypadku mamy policzyć rangi dla wszystkich elementów danego ciągu, a więc robimy to, a więc realizujemy następujący algorytm

for (i=0, i<n, i++) {r[i] = ranga(i, x, n);}

gdzie **ranga** jest funkcją[1], która dla danego ciągu x długości n (tablicy indeksowanej liczbami od 0 do $n-1$) wylicza rangę liczby $x[i]$ w ciągu x.

W dalszym ciągu litera n będzie oznaczać liczbę lub zmienną w programie, o której jest mowa w specyfikacji zadania, podobnie x, które oznacza ciąg, ewentualnie tablicę o n elementach, a r to ciąg wyliczonych rang lub tablicę służąca do gromadzenia tych liczb.

Musimy jeszcze jakoś zdefiniować funkcję **ranga**. Zgodnie z naturalnym algorytmem funkcja ta powinna porównywać daną liczbę z każdym wyrazem danego ciągu i zliczać te wyrazy, które są mniejsze od danej liczby. Taki algorytm nie jest godny przedstawienia w tym tekście, ale podała go niemała liczba osób rozwiązujących zadanie. Jest poprawny dla wszystkich możliwych danych i nie wykorzystuje szczególnych własności danych, opisanych w specyfikacji.

[1] Będzie wiele funkcji o nazwie **ranga** i będą mogły się różnić postacią parametrów.

2.2 Ranga elementu ciągu niemalejącego w tym ciągu

Łatwo znajduje się rangę i-tego wyrazu niemalejącego ciągu x. Można to zrobić zgodnie z następującym algorytmem (ten algorytm może stać się treścią funkcji **ranga**):

if (x[i] $\leqslant$ x[0]) return 0;
while (x[i-1] = x[i]) i−−; return i;

Może warto zatrzymać się chwilę na poprawnością tego algorytmu. Algorytm ten może być wykonywany na dwa sposoby, zależnie od danych. Jeżeli $x[i] \leqslant x[0]$, to w danym ciągu nie ma elementów mniejszych od $x[i]$ i na mocy definicji ranga $x[i]$ jest równa 0. Stąd zwrócona przez algorytm wartość jest w tym przypadku rangą $x[i]$. W drugim przypadku poprawność algorytmu wynika z faktu, że każde dwa równe wyrazy danego ciągu mają w tym ciągu tę samą rangę (ranga zależy tylko od wartości wyrazu, a nie od jego położenia). Poprawność daje się dowieść formalnie metodą niezmienników sprawdzając, że stwierdzenie *ranga* $x[i]$ *jest równa randze* $x[i_0]$ jest niezmiennikiem pętli **while** w rozważanym algorytmie (i_0 to początkowa wartość zmiennej i).

Kontrowersje zwykle budzi pytanie, czy tak proste fakty trzeba dowodzić. Moim zdaniem jest to oczywiste. Każdy programista musi jakoś ustalać (choćby dla własnej potrzeby), czy pisze poprawne programy. Jedną metod badania poprawności jest jej uzasadnianie. Testowanie programów jest pomocne, ale niewystarczające. Dlatego student czasem musi przekonać osoby oceniające jego umiejętności, że potrafi badać (dowodzić) poprawność programów. Umiejętności studenta mogą być badane metodą testową („napisz program, który uważasz za poprawny, a my ocenimy jego poprawność"), ale nie jest to ani jedyna, ani najlepsza metoda. Dlatego również sprawdza się, czy i jak studenci uzasadniają poprawność programów, bo jest to bardzo ważna i niezbędna umiejętność.

Chwilę uwagi warto też poświęcić złożoności podanego algorytmu. Łatwo przekonać się, że wartości zmiennej i przyjmowane podczas działania algorytmu tworzą ciąg malejący o wartościach od 0 do n (dlaczego?), gdzie n to wielkość danej tablicy. Kolejne zmiany wartości tej zmiennej następują po wykonaniu określonej liczby czynności. Stąd złożoność czasowa tej procedury jest rzędu $O(n)$. Co więcej, tego oszacowania nie daje poprawić, o czym świadczy ciąg $0, 1, 1, 1, \ldots, 1$. Licząc rangę ostatniego elementu tego ciągu zgodnie z podanym przepisem trzeba wykonać przynajmniej $n-1$ czynności (na przykład $n-1$ razy nadajemy wartość zmiennej i). Jak widać można jakoś argumentować, że rozważana procedura ma złożoność rzędu $O(n)$. Czasem warto to zrobić z różnych względów, na przykład z wyżej podanych powodów lub dla sprawdzenia własnych umiejętności.

2.3 Ranga liczby w ciągu niemalejącym

Czasem może być potrzebne ogólniejsze pojęcie rangi i bardziej ogólna procedura, która dla danego ciągu podaje rangę dowolnej liczby, niekoniecznie występującej w danym ciągu. Taką procedura, działającą dla ciągów monotonicznych, może być wzorowana na poprzedniej i może mieć następującą postać:

if (x[n-1] < a) return n;
i = 0; while (x[i] < a) i++; return i;

Analizę poprawności i złożoności tej procedury pozostawiam Czytelnikowi.

2.4 Przykład rozwiązania o złożoności $O(n^2)$

Przypuśćmy, że mamy następującą procedurę **ranga**. Proste rozwiązanie naszego zadania możemy otrzymać dopisując do niej jedną instrukcję:

int ranga(int a, p, k)

/*Zakładamy, mamy daną tablicę x, ciąg $x[p], x[p+1], \ldots, x[k-1]$ jest dobrze określonym, niemalejącym podciągiem x i chcemy wyliczyć rangę liczby a we wskazanym podciągu x.*/

{ if (x[k-1] < a) return k - p;

i = p; while (x[i] < a) i++; return i; }

for (i=0, i<n, i++) r[i] = ranga(x[i],0,n/2) + ranga(x[i],n/2,n);

3 Rozwiązania o złożoności $O(n \log n)$

3.1 Ranga w ciągu niemalejącym rekurencyjnie

Rangę można też obliczać za pomocą następującej funkcji rekurencyjnej.

int ranga(int a, p, k)

/*Zakładamy, mamy daną tablicę x, ciąg $x[p], x[p+1], \ldots, x[k-1]$ jest dobrze określonym, niemalejącym podciągiem x i chcemy wyliczyć rangę liczby a we wskazanym podciągu x.*/

s = p + (k – p)/2;

if (x[s] < a)

{ if (p == s) return 1;

else return s – p + ranga(a, s, k); }

else { if (p == s) return 0;

else return ranga(a, p, s); }

Poprawność tej procedury dowodzimy indukcyjnie. Dokładniej, pokazujemy, że działa poprawnie dla wszystkich danych i robimy to przez indukcję ze względu na różnicę $k-p$ (powinna to być liczba naturalna).

Natomiast złożoność tej procedury potrafimy ocenić pisząc i rozwiązując rekurencyjną definicję funkcji, która danym liczbom p i k przyporządkowuje liczbę wywołań procedury **ranga** podczas realizacji obliczeń dla tych danych.

3.2 Ranga za pomocą wyszukiwania binarnego

Rangę możemy obliczać też za pomocą algorytmów wzorowanych na wyszukiwaniu binarnym. Poniżej znajdują się dwa takie algorytmy. Pierwszy znajduje rangę dowolnej liczby w ciągu niemalejącym, drugi wylicza rangę wyrazu takiego, danego ciągu.

int ranga(int a, k)

/*Zakładamy, mamy daną tablicę x, ciąg $x[0], x[1], \ldots, x[k-1]$ jest dobrze określonym, niemalejącym podciągiem x i chcemy wyliczyć rangę liczby a we wskazanym podciągu x.*/

p = 0; s = k;

```
while (p < s) {
    s = p + (k − p)/2;
    if ( x[s] < a) p = s;
    else k = s; };
return k;
```

Nierówności $p \leqslant s \leqslant k$ są niezmiennikami pętli
Jeżeli $x[k-1] < a$, to wartość zmiennej k nie ulega zmianie.
Jeżeli $a \leqslant x[0]$, to wartość p jest stale równa 0, a równość $s = k$ jest niezmiennikiem pętli.
Nierówność $x[p] < a \leqslant x[k-1]$ są niezmiennikami pętli W przeciwnym wypadku, nieróność $x[p] < a \leqslant x[k]$

```
int ranga(int x[ ], int a, p, k)
    /*Zakładamy, mamy daną tablicę x, ciąg x[0], x[1], ..., x[k] jest dobrze
    określonym, niemalejącym podciągiem x i chcemy wyliczyć rangę liczby a
    we wskazanym podciągu x.*/
while (p < k) {
    s = p + (k − p)/2;
    if ( x[s] < a) p = s;
    else k = s; };
return k;
```

3.3 Przykład rozwiązania o złożoności $O(n \log n)$

```
y = scal(x, 0, n/2, n);

for (i = 0, i < n, i++) r[i] = ranga (y, x[i], 0, n-1);
```

4 Najefektywniejsze rozwiązania

Główna idea prowadząca do bardziej efektywnych rozwiązań polega na tym, aby w obliczeniach rangi elementu $x[i+1]$ wykorzystać informacje zgromadzone podczas obliczania rangi $x[i]$. Możemy to zrobić przeglądając dane ciągi i zliczając te wyrazy, które są mniejsze od $x[i]$, a następnie uzupełniamy obliczenia zliczając wyrazy przynajmniej równe $x[i]$ i mniejsze od $x[i+1]$.

4.1 Pierwszy pomysł

Może najpierw spróbujmy rozwiązać nieco prostsze zadanie. Zakładamy więc, że mamy dane dwie tablice x i y, obie są indeksowane od 0 do $m-1$ i zawierają niemalejące ciągi liczb. Chcemy wyliczyć rangi elementów tablicy x w połączonych ciągach z tablic x i y.

W tym celu stworzymy odpowiednią strukturę danych złożoną z danych tablic oraz

1) zmiennej i, która będzie przechowywać indeks elementu tablicy x, tego elementu, którego ranga jest aktualnie obliczana,

2) zmiennej j, która będzie przechowywać indeks do tablicy y to miejsce w tablicy y, do którego jej zawartość została przeanalizowana,

3) zmiennej s, której wartość będzie określona przez przeliczone fragmenty obu danych tablicy i będzie zawierać zgromadzoną informację o rangach $x[i]$ oraz dalszych elementów, w szczególności będzie szacować od dołu rangi $x[k]$ dla $k > i$.

```
int i, j, s = 0;
while (i < m ) {
    if (i > 0 && x[i-1] = x[i]) { r[i] = r[i-1]; i++; s++; }
    else {
        while ( j < m && x[j] < x[i]) { j++; s++; }
        r[i] = s; i++; s++; }
    }
```

Głównym niezmiennikiem tego programu jest stwierdzenie, że

$$s = | \{k < i | x[k] \leqslant x[i]\} | + | \{k < j | y[k] < x[i]\} |.$$

Co więcej jest niezmiennik obu pętli występujących w programie i gwarantuje on, poprawność dokonywanych obliczeń rang. Formalny dowód poprawności wymaga wykazania, że niezmiennikiem głównej pętli **while** jest także stwierdzenie, że

$$\forall k < i \;\; r[k] \text{ jest rangą } x[k] \text{ w połączonych ciągach } x \text{ i } y.$$

Złożoność programu

Przedstawiony program można nieco uprościć i przyśpieszyć usuwając z niego zmienną s i zastępując podstawienie $r[i] = s$ przez $r[i] = i + j$.

4.2 Ostatni program

```
int i = 0, j = n/2; p; e = 0;
if (x[i] < x[j]) { p = i; r[i] = 0; i++};
else { p = j; r[j] = 0, j++; };
```

/*Zdefiniowaliśmy pewną strukturę danych i określiliśmy jej stan początkowy. Zmienne i i j to indeksy odpowiednio do pierwszej i drugiej części tablicy x, początkowo wskazują ich pierwsze elementy. Z tablicy x będziemy wybierać elementy w pewnym porządku tak, jak podczas scalania. Pierwszy element został już wybrany. Dodatkowo, zmienna p będzie przechowywać indeks ostatniego, wcześniej wybranego elementu tablicy x (lub innego o tej samej wartości), a zmienna e będzie przechowywać liczbę wystąpień elementu $x[p]$, poza samym $x[p]$, w przeanalizowanym fragmencie ciągu.*/

```
while (i < n/2 && j < n) {
    w = i + j - n/2;
    if (x[i] < x[j]) { k = i; i++; } else { k = j; j++; }
    if (x[k] = x[p]) e++; else { e = 0; p := k }
```

```
    r[k] = w - e; }
while (i < n/2) {
    if (x[i] = x[p]) e++ ; else { e = 0; p := i }
    r[i] = i + n/2 - e; i++; }
while (j < n) {
    if (x[j] = x[p]) e++ ; else { e = 0; p := j }
    r[k] = j - e; j++; }
```