Marian M. Kędzierski
Treść zadania

Marian M. Kędzierski
Piotr Niedźwiedź
Opracowanie

Piotr Niedźwiedź
Program

Dostępna pamięć: 256 MB. *OI, Etap III, dzień próbny, 13.04.2010*

Monotoniczność

Schematem monotoniczności *ciągu liczb całkowitych $a_1, a_2, \ldots, a_n$ nazwiemy ciąg $s_1, s_2, \ldots, s_{n-1}$ złożony ze znaków $<$, $>$ lub $=$. Znak s_i reprezentuje relację pomiędzy liczbami a_i i a_{i+1}. Na przykład, schematem monotoniczności ciągu $2, 4, 3, 3, 5, 3$ jest $<, >, =, <, >$.*

Powiemy, że ciąg liczb $b_1, b_2, \ldots, b_{n+1}$, o schemacie monotoniczności $s_1, s_2, \ldots, s_n$, **realizuje** *pewien schemat monotoniczności $s'_1, s'_2, \ldots, s'_k$, jeżeli dla każdego całkowitego $i = 1, 2, \ldots, n$ zachodzi $s_i = s'_{((i-1) \bmod k)+1}$. Innymi słowy, ciąg $s_1, s_2, \ldots, s_n$ uzyskujemy, powtarzając odpowiednio długo ciąg $s'_1, s'_2, \ldots, s'_k$ i ewentualnie odrzucając kilka ostatnich wyrazów tego powtórzenia. Na przykład, ciąg $2, 4, 3, 3, 5, 3$ realizuje następujące schematy monotoniczności:*

- $<, >, =$
- $<, >, =, <, >$
- $<, >, =, <, >, <, <, =$
- $<, >, =, <, >, =, >, >$

i wiele innych.

Dany jest ciąg liczb całkowitych $a_1, a_2, \ldots, a_n$. Twoim zadaniem jest znalezienie najdłuższego jego podciągu $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$ ($1 \leqslant i_1 < i_2 < \ldots < i_m \leqslant n$) realizującego pewien zadany schemat monotoniczności $s_1, s_2, \ldots, s_k$.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n oraz k ($1 \leqslant n \leqslant 20\,000$, $1 \leqslant k \leqslant 100$), oddzielone pojedynczym odstępem i oznaczające odpowiednio długość ciągu (a_i) oraz długość schematu monotoniczności (s_j).

W drugim wierszu znajduje się n liczb całkowitych a_i pooddzielanych pojedynczymi odstępami, oznaczających wyrazy badanego ciągu ($1 \leqslant a_i \leqslant 1\,000\,000$).

W trzecim wierszu znajduje się k znaków s_j postaci `<`, `>` lub `=`, pooddzielanych pojedynczymi odstępami, oznaczających kolejne wyrazy schematu monotoniczności.

Wyjście

W pierwszym wierszu standardowego wyjścia Twój program powinien wypisać jedną liczbę całkowitą m oznaczającą maksymalną długość podciągu ciągu $a_1, a_2, \ldots, a_n$ realizującego schemat monotoniczności $s_1, s_2, \ldots, s_k$.

W drugim wierszu Twój program powinien wypisać dowolny przykład takiego podciągu $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$, oddzielając jego wyrazy pojedynczymi odstępami.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`7 3`	`6`
`2 4 3 1 3 5 3`	`2 4 3 3 5 3`
`< > =`	

Rozwiązanie

Szczególny przypadek

Treść zadania niestety nie należy do najprostszych, więc na początek zajmijmy się analizą pewnego szczególnego przypadku. Schemat „`<`” reprezentuje znany problem szukania najdłuższego podciągu rosnącego zadanego ciągu. My natomiast zajmiemy się schematem „`<>`”, od którego zresztą zaczęła się historia tego zadania. Można ten przypadek rozpatrzyć w znacznie prostszy sposób niż oryginalne zadanie, jednak pomoże on dojść do rozwiązania ogólnej wersji.

Mamy zatem znaleźć najdłuższy podciąg, którego drugi wyraz jest większy od pierwszego, a potem każdy następny jest na zmianę mniejszy i większy od poprzedniego. Dla prostoty notacji, nazwijmy każdy taki podciąg *antymonotonicznym*. W rozwiązaniu wykorzystamy technikę programowania dynamicznego. Chcielibyśmy dla każdej pozycji i w ciągu zapamiętać, jako $tab[i]$, maksymalną długość podciągu antymonotonicznego kończącego się na pozycji i. Jest to najczęściej spotykane podejście w programowaniu dynamicznym, jednak w tym wypadku nie widać, jak do obliczenia wartości $tab[i]$ wykorzystać tylko wartości w tablicy tab pod indeksami od 1 do $i-1$.[1] Zilustrujmy to na przykładzie: jeżeli $a_i = 10$, $a_{i-1} = 20$ oraz $tab[i-1] = 5$, to podciągu reprezentowanego przez $tab[i-1]$ nie możemy przedłużyć o wyraz a_i, ponieważ zakłóciłoby to schemat monotoniczności tego podciągu. Jeżeli a_i miałby być antymonotonicznym przedłużeniem podciągu kończącego się na a_{i-1}, to długość tego podciągu musiałaby być parzysta. Jednak wartość $tab[i-1]$ nie mówi nic o maksymalnej **parzystej** długości podciągu antymonotonicznego kończącego się na a_{i-1}.

Jak poradzić sobie z tym ograniczeniem? Otóż zamiast jednej tablicy tab, użyjemy dwóch:

- $tab1$ — reprezentującej maksymalny podciąg antymonotoniczny *nieparzystej* długości kończący się na zadanej pozycji,
- $tab2$ — reprezentującej maksymalny podciąg antymonotoniczny *parzystej* długości kończący się na zadanej pozycji.

Obliczenie wartości dla pozycji i będzie przebiegało następująco[2]:

$$tab1[i] = \max(1,\ 1 + \max\{tab2[j]\ :\ j < i \text{ oraz } a_j > a_i\}), \tag{1a}$$

$$tab2[i] = 1 + \max\{tab1[j]\ :\ j < i \text{ oraz } a_j < a_i\}. \tag{1b}$$

[1] W rzeczywistości (jak okaże się później) do obliczenia $tab[i]$ wystarczy znajomość wartości $tab[1], tab[2], \ldots, tab[i-1]$; jest to jednak wysoce nieoczywiste.

[2] Przyjmujemy standardowo $\max \emptyset = -\infty$.

Po obliczeniu wartości we wszystkich komórkach obydwu tablic szukana optymalna długość podciągu antymonotonicznego będzie równa maksimum z wszystkich elementów tych tablic. Dodatkowo, aby można było odtworzyć optymalny podciąg, potrzebne będą kolejne dwie tablice, w których zapisywać będziemy liczby j, dla których realizowane są maksima w każdym ze wzorów (1ab).

Łatwo widać, że złożoność czasowa takiego algorytmu to $O(n^2)$.

Przypadek ogólny

Będziemy odtąd zakładać, że zapis „m mod k" oznacza $((m-1) \bmod k)+1$. Innymi słowy, będziemy operować na $\{1, 2, \ldots, k\}$ jako na zbiorze reszt modulo k, co jest wygodne, jeśli wszystkie ciągi w zadaniu są numerowane od jedynki. Dodatkowo, przyjmiemy, że zapis s_m (gdzie s to schemat monotoniczności) oznacza $s_{(m \bmod k)}$.

Aby uogólnić poprzedni algorytm do oryginalnego zadania, użyjemy k tablic: $tab[1..k][1..n]$, a wzór na wartości w ich komórkach będzie następujący:

$$tab[(m+1) \bmod k][i] = 1 + \max\{tab[m][j] \ : \ j < i \text{ oraz } a_j \langle s_m \rangle a_i\}, \qquad (2)$$

z wyjątkiem przypadku $m = k$, w którym dodatkowo bierzemy maksimum z tej wartości i z jedynki. Użyty powyżej zapis $a_j \langle s_m \rangle a_i$ oznacza:

$a_j < a_i$ jeśli $s_m =$ „<", $\quad a_j = a_i$ jeśli $s_m =$ „=", $\quad a_j > a_i$ jeśli $s_m =$ „>".

W tym przypadku obliczamy zawartość już nie dwóch, lecz k tablic, toteż złożoność czasowa wzrasta do $O(n^2k)$. Przykładowa implementacja powyższego algorytmu znajduje się w plikach `mons1.cpp` oraz `mons2.pas`. Na zawodach podobne rozwiązania otrzymywały ok. 40-50 pkt.

Rozwiązanie wzorcowe

Powyższy algorytm jest już tylko o krok od wzorcowego. Wprawny zawodnik szybko zauważy pole do optymalizacji. Otóż jest nim obliczanie maksimum w powyższych wzorach — można do tego wykorzystać drzewa przedziałowe (zwane także licznikowymi). Jest to struktura danych zbudowana nad ciągiem, pozwalająca (w podstawowej wersji) efektywnie obliczać wyniki dowolnego działania łącznego (na przykład sumy, minimum, maksimum) na zadanym spójnym fragmencie tego ciągu oraz równie szybko aktualizować jego wyrazy. Złożoność pamięciowa drzewa przedziałowego jest liniowa względem długości ciągu, zaś każda operacja zajmuje czas logarytmiczny[3].

Zbudowanie drzew przedziałowych bezpośrednio nad każdą z k tablic $tab[m][1..n]$ niestety nie rozwiązuje naszego problemu. Wówczas bylibyśmy w stanie obliczać maksima ze spójnych fragmentów odpowiedniej tablicy $tab[m]$. My jednak potrzebujemy wyznaczać je dla zbiorów indeksów postaci:

- $\{j$ takich, że $j < i$ oraz $a_j < a_i\}$,

[3]Więcej na temat drzew przedziałowych można znaleźć np. w opracowaniu zadania *Latarnia* w tej książeczce oraz w podanych tam odnośnikach.

- $\{j$ takich, że $j < i$ oraz $a_j = a_i\}$,
- $\{j$ takich, że $j < i$ oraz $a_j > a_i\}$.

Zauważmy jednak, że są to spójne podzbiory zbioru $\{1, 2, \ldots, i-1\}$ w sensie porządku na wartościach ciągu a. Należy więc na początku posortować wszystkie liczby a_i, otrzymując ciąg $a_{r_1}, a_{r_2}, \ldots, a_{r_n}$. Wtedy m-te drzewo przedziałowe będzie budowane nad ciągiem $tab[m][r_1], tab[m][r_2], \ldots, tab[m][r_n]$.

Przykład. Dla ciągu $a = (2, 4, 3, 1, 3, 5, 3)$ z przykładu z treści zadania, ciąg r może mieć postać np. $(4, 1, 3, 5, 7, 2, 6)$ albo $(4, 1, 5, 7, 3, 2, 6)$, czyli indeksy odpowiadające równym elementom ciągu a mogą być ustawione w dowolnej kolejności.

Zauważmy, że spójne fragmenty ciągu $r_1, r_2, \ldots, r_n$ odpowiadają spójnym przedziałom wartości w ciągu a_j. Dla każdego elementu a_i ciągu a, będziemy pamiętali indeks p_i jego wystąpienia w ciągu r (tzn. $r_{p_i} = i$), a także dwa indeksy l_i i u_i, oznaczające indeksy skrajnych elementów ciągu r odpowiadających elementom ciągu a równym a_i, tzn.:

$$a_i = a_{r_{l_i}} = a_{r_{l_i+1}} = \ldots = a_{r_{u_i}},$$

natomiast elementy $a_{r_{l_i-1}}$ oraz $a_{r_{u_i+1}}$ nie istnieją lub są odpowiednio mniejszy i większy niż a_i. Dzięki temu będziemy w stanie wyznaczać maksimum w równaniu (2) w czasie $O(\log n)$, obliczając, na drzewie przedziałowym, wartość ze wzoru[4]:

$$tab[(m+1) \bmod k][i] = \begin{cases} 1 + \max\{tab[m][r_j] \ : \ j \in [1,\ l_i - 1]\} & \text{dla } s_m = \text{„<”}, \\ 1 + \max\{tab[m][r_j] \ : \ j \in [u_i + 1,\ n]\} & \text{dla } s_m = \text{„>”}, \\ 1 + \max\{tab[m][r_j] \ : \ j \in [l_i,\ u_i] \setminus \{p_i\}\} & \text{dla } s_m = \text{„=”}. \end{cases}$$

Aby obliczanie maksimów było poprawne, wartości *tab* muszą być inicjowane np. na $-\infty$. W ten sposób egzekwujemy warunek $j < i$ z równania (2).

Również w czasie $O(\log n)$ możemy aktualizować odpowiednie drzewo przedziałowe po nadpisaniu wartości $tab[m][i]$. Przeprowadzona optymalizacja pozwala zatem otrzymać algorytm o sumarycznej złożoności czasowej $O(kn \log n)$ i pamięciowej $O(kn)$. Jest to algorytm wzorcowy, jego przykładowa implementacja znajduje się w plikach `mon.cpp` oraz `mon1.pas`.

Rozwiązanie alternatywne

Uważny Czytelnik mógł dostrzec, że rozwiązanie wzorcowe używa drzew przedziałowych podobnie, jak to ma miejsce w znanym algorytmie wyznaczania najdłuższego podciągu rosnącego. Okazuje się, że bazując na innym, klasycznym algorytmie rozwiązującym ten problem, także można skonstruować dość efektywne rozwiązanie zadania. Przedstawmy najpierw ów algorytm[5]. Przeglądamy kolejne wyrazy ciągu, przechowując przy tym odpowiedzi na pytania postaci: ile wynosi najmniejsza liczba, jaką może

[4] W trzecim z poniższych przypadków należy zauważyć, że $[l_i,\ u_i] \setminus \{p_i\} = [l_i,\ p_i - 1] \cup [p_i + 1,\ u_i]$.

[5] Nieco dokładniejszy opis tego algorytmu można znaleźć w opracowaniu zadania *Egzamin na prawo jazdy* z XIV Olimpiady Informatycznej, patrz książeczka [14].

kończyć się podciąg rosnący długości m (oczywiście chodzi o podciągi dotychczas przejrzanego fragmentu ciągu). Uznajemy, że jeśli nie ma podciągów długości m, to szukana liczba wynosi $+\infty$. Wartości te przechowujemy w tablicy $\ell[1..n]$, wygodnie nam będzie przyjąć $\ell[0] = 0$.

Zastanówmy się, jak aktualizować tę tablicę. Zauważmy, że liczby w niej zawarte tworzą ciąg niemalejący. Każdy koniec podciągu rosnącego długości m jest też bowiem końcem krótszych podciągów rosnących. Aby pewien ciąg rosnący długości m mógł kończyć się liczbą a_i, musi zachodzić $\ell[m-1] < a_i$. Wówczas będziemy chcieli dokonać przypisania $\ell[m] := \min(\ell[m], a_i)$. Aby ta operacja cokolwiek zmieniła, musimy mieć $a_i < \ell[m]$. Podsumowując, wystarczy znaleźć wszystkie takie m, że $\ell[m-1] < a_i < \ell[m]$. Ciąg ℓ jest niemalejący, więc szukana wartość będzie co najwyżej jedna, a można ją znaleźć np. przez wyszukiwanie binarne.

Spróbujmy zaadaptować przedstawiony algorytm do potrzeb zadania. Na razie załóżmy, że w schemacie s występują tylko znaki „<” i „>”. Aby uprościć dalszy opis, podciąg długości m dotychczas przejrzanego fragmentu ciągu a, kończący się wyrazem $a_i = p$ i realizujący schemat s nazwijmy *dobrym m-podciągiem o końcu p*. Jeśli $s_m =$ „<”, to w polu $\ell[m]$ będziemy przechowywać najmniejszy możliwy koniec dobrego m-podciągu, jeśli zaś $s_m =$ „>”, to największy. Pola tablicy ℓ inicjujemy wówczas odpowiednio na $+\infty$ i $-\infty$. Warunek aktualizacji wartości $\ell[m]$ jest podobny jak poprzednio (dla $m = 1$ przyjmujemy pierwszą część za trywialnie prawdziwą):

$$\ell[m-1] \,\langle s_{m-1}\rangle\, a_i \quad \text{oraz} \quad a_i \,\langle s_m\rangle\, \ell[m]. \tag{3}$$

Jego uzasadnienie także nie ulega zmianie. Tym razem liczby w tablicy ℓ nie tworzą ciągu monotonicznego, więc będziemy po prostu sprawdzać wszystkie możliwe wartości m.

Pozostaje jeszcze do rozwiązania problem znaków „=” w schemacie s. Tu użyjemy rozwiązania podobnego jak w algorytmie wzorcowym: dla każdej możliwej wartości v w ciągu a_i i każdego $j \in \{1, 2, \ldots, k\}$, takiego że $s_j =$ „=”, będziemy pamiętać długość najdłuższego dobrego podciągu o końcu v i o długości przystającej do j modulo k. Jeśli na samym początku przenumerujemy wartości ciągu a_i, wystarczy nam do tego zwykła tablica $T[j][v]$.

Teraz aktualizacja naszych struktur dla danego a_i, gdzie $i > 1$, będzie przebiegać następująco.

```
1:  for m := i downto 1 do begin
2:    { war określa, czy istnieje dobry m-podciąg o końcu a_i }
3:    if s_{m-1} ≠ „=” then war := ( ℓ[m-1] ⟨s_{m-1}⟩ a_i )
4:    else war := ( T[(m-1) mod k][a_i] ⩾ m-1 );
5:    if not war then continue;
6:    if (s_m ≠ „=”) and a_i ⟨s_m⟩ ℓ[m] then
7:      ℓ[m] := a_i
8:    else if s_m = „=” then
9:      T[m mod k][a_i] := max(T[m mod k][a_i], m);
10: end
```

Przypomnijmy, że dla $m = 1$ pomijamy sprawdzenie warunku *war*, wiedząc, że jest spełniony. Dokładne uzasadnienie poprawności przedstawionego algorytmu pozosta-

wiamy Czytelnikowi jako sympatyczne ćwiczenie (w szczególności, dlaczego indeks m w pętli **for** musi maleć, a nie może np. rosnąć?).

Umiemy już wyznaczyć długość najdłuższego dobrego podciągu. Jest to większa z dwóch wartości: największego m, dla którego $\ell[m] \neq \pm\infty$, oraz największej z wartości w tablicy T. Zastanówmy się teraz, jak odzyskać szukany podciąg. W rozwiązaniu wzorcowym było to łatwe — wystarczyło w tablicy (nazwijmy ją F) pamiętać indeks j realizujący maksimum we wzorze (2). Tu zapamiętamy dokładnie te same wartości. Aby było to możliwe, musimy jeszcze nieco wzbogacić nasze informacje. Elementy tablic ℓ i T łączy fakt, że informują o istnieniu pewnego dobrego podciągu. Dołączymy zatem do nich dodatkowe tablice przechowujące *indeks* ostatniego wyrazu tego podciągu.

Jak przy ich pomocy obliczyć $F[m \bmod k][i]$? Otóż w przedstawionym wyżej pseudokodzie dla danego i i każdego m stwierdzaliśmy, czy istnieje dobry m-podciąg o końcu a_i. Korzystając z naszych pomocniczych tablic, umiemy wskazać jego poprzedni wyraz. Dzięki temu możemy bezpośrednio wskazać poprzedni wyraz najdłuższego spośród takich podciągów dla m dającego ustaloną resztę z dzielenia przez k, a właśnie tę wartość chcemy zapamiętać w tablicy F.

Złożoność czasowa powyższego algorytmu wynosi $O(n^2)$, zaś pamięciowa — $O(nk)$. W plikach `mon2.cpp` i `mon3.pas` znajdują się programy, które są nieco usprawnionymi wersjami przedstawionego algorytmu. W praktyce są nawet 10 razy szybsze niż rozwiązanie wzorcowe, jako że stała ukryta w notacji O jest tu stosunkowo nieduża.

Rozwiązanie o złożoności $O(n \log n)$

Okazuje się, że nasze zadanie można rozwiązać jeszcze szybciej. Poniższe rozwiązanie jest dosyć zaskakujące: otóż algorytm wzorcowy pozostaje poprawny, jeśli zamiast k tablic $tab[1][1..n], tab[2][1..n], \ldots, tab[k][1..n]$ użyjemy jednej — $f[1..n]$, reprezentującej ich maksimum. Krok obliczenia wartości $f[i]$ wygląda wtedy następująco:

$$f[i] = 1 + \max\{f[j] \text{ dla } j < i \text{ takich, że } a_j \langle s_{f[j]} \rangle a_i\}. \tag{4}$$

Okazuje się, że wartość $f[i]$ reprezentuje wówczas długość najdłuższego podciągu zgodnego ze schematem monotoniczności, kończącego się na pozycji i.

Twierdzenie 1. *Każdy najdłuższy podciąg zadanego ciągu $(a_i)_{i=1}^n$, spełniający zadany schemat monotoniczności s oraz kończący się na pozycji j (czyli na elemencie a_j), nazwiemy* optymalnym podciągiem dla pozycji j. *Niech $f(x)$ oznacza długość najdłuższego optymalnego podciągu dla pozycji $x \in \{1, \ldots, n\}$. Wtedy*

$$f(x) = 1 + \max\{f(y) \ : \ y < x \ \wedge \ a_y \langle s_{f(y)} \rangle a_x\}.$$

Dowód: Dowód przeprowadzimy nie wprost. Załóżmy, że $a_1, a_2, \ldots, a_n$ jest najkrótszym takim ciągiem, dla którego teza twierdzenia nie zachodzi. Wówczas zachodzi ono dla ciągu $a_1, a_2, \ldots, a_{n-1}$ (bo inaczej wybrany ciąg nie byłby najkrótszy). To znaczy, że pozycja n jest jedyną, dla której optymalny podciąg (odpowiadający $f(n)$) nie jest przedłużeniem o pozycję n jakiegoś innego optymalnego podciągu.

Niech $f(n) = k$. Istnieje zatem optymalny podciąg o długości k dla pozycji n. Ponadto, skoro twierdzenie nie zachodzi dla n, to $k > 1$. Niech więc $m < n$ oznacza pozycję $(k-1)$-szego wyrazu tego optymalnego podciągu. Wiemy, że $f(m) \geqslant k-1$, a gdyby $f(m) = k-1$, to twierdzenie zachodziłoby dla pozycji n. Zatem $f(m) > k-1$. Spójrzmy na pewien optymalny podciąg dla pozycji m. Jest on wyznaczony przez ciąg pozycji $b_1, b_2, \ldots, b_{f(m)}$, gdzie $b_{f(m)} = m$. Należy zwrócić uwagę, że $f(b_i) = i$ dla każdego $i = 1, 2, \ldots, m$ (ponieważ twierdzenie zachodzi dla wszystkich $x < n$).

Pokażemy, że można usunąć pewną liczbę wyrazów z końca ciągu b_i, a w zamian dostawić liczbę n, tak aby wyrazy otrzymanego ciągu spełniały schemat monotoniczności oraz aby jego długość była nie mniejsza niż k, co będzie stanowić żądaną sprzeczność.

W tym celu rozważmy kilka przypadków:

- $a_m = a_n$, czyli $s_{k-1} =$ „=”. Nowy ciąg ma postać $b_1, b_2, \ldots, b_{f(m)-1}, n$ — zastępując a_m przez a_n, nie zmieniamy podciągu, więc ten pozostanie dobry.
- $a_m < a_n$, czyli $s_{k-1} =$ „<”. Podzielmy ten przypadek na dwa podprzypadki:
  - $a_{b_{k-1}} < a_n$. Nowy ciąg ma postać $b_1, \ldots, b_{k-1}, n$ — dalej $(k-1)$-szy wyraz jest mniejszy od k-tego.
  - $a_{b_{k-1}} \geqslant a_n$. Tu jest nieco trudniej. Wiemy, że $a_{b_{k-1}} < a_{b_k}$. Stąd wnioskujemy, że musi istnieć takie $w \geqslant k$, że $a_{b_w} > a_{b_{w+1}}$ (czyli $s_w =$ „>”) i $a_{b_w} > a_n$. Wtedy nowy ciąg ma następującą postać: $b_1, \ldots, b_{k-1}, \ldots, b_w, n$.
- $a_m > a_n$, czyli $s_{k-1} =$ „>”. Ten przypadek jest symetryczny względem poprzedniego.

Tak więc twierdzenie zachodzi również dla $x = n$, co jest sprzeczne z założeniem. Otrzymujemy więc, że teza twierdzenia jest prawdziwa dla dowolnego ciągu a, schematu monotoniczności s oraz pozycji x. ■

Jak możemy zaimplementować niniejsze rozwiązanie? Wyznaczanie tablicy f z definicji, z liniowym obliczaniem maksimów, zajmie nam czas $\Theta(n^2)$. Chcielibyśmy znów skorzystać z drzew przedziałowych. Możemy, podobnie jak w rozwiązaniu wzorcowym, zbudować takie drzewo nad tablicą $f[r_i]$. Jednak tym razem nasz problem wcale nie sprowadza się do szukania maksimum w przedziale — to, czy szukamy liczb a_j większych, równych, czy też mniejszych od a_i, zależy od j. Jednak możemy zbudować trzy tablice: $f_<$, $f_=$ i $f_>$ zdefiniowane następująco (dla $q \in \{$„<”, „=”, „>”$\}$):

$$f_q[j] = \begin{cases} f[j] & \text{jeżeli } s_{f[j]} = q \\ 0 & \text{w przeciwnym przypadku.} \end{cases}$$

Dzięki temu możemy nad każdą z tablic $f_q[r_j]$ zbudować drzewo przedziałowe, w których to drzewach będziemy szukać maksimów w spójnych przedziałach. Złożoność czasowa algorytmu wyniesie $O(n \log n)$, zaś pamięciowa $O(n)$. Jest on więc równie szybki, co najlepsze znane rozwiązanie problemu obliczania najdłuższego podciągu rosnącego, który to problem jest, rzecz jasna, szczególnym przypadkiem rozważanego zadania. Implementację tego rozwiązania można znaleźć w pliku `mon5.cpp`.

Rozwiązania powolne

Najprostszym i zarazem najmniej efektywnym rozwiązaniem zadania jest metoda przeszukiwania z nawrotami (brutalna), pesymistycznie wymagająca zbadania wszystkich $\Theta(2^n)$ możliwych podciągów. Takie rozwiązanie zaimplementowano w pliku `monb2.cpp`. Na zawodach otrzymywało ok. 20 punktów.

Innym rozwiązaniem poprawnym, ale zużywającym dużą ilość pamięci, jest algorytm, który nie wykorzystuje przenumerowania indeksów przy budowaniu drzewa przedziałowego i w związku z tym buduje je nad indeksami z przedziału od 1 do $\max_i a_i$. Zaimplementowano je w pliku `monb3.cpp`, otrzymywało ono ok. 30 punktów.

Testy

Zadanie było sprawdzane na 10 zestawach danych testowych. Testy dzielą się na trzy rodzaje: testy *losowe*, testy zawierające tylko jeden rodzaj monotoniczności (*monotoniczne*) oraz testy *skokowe*, czyli składające się z większych, monotonicznych grup.

Nazwa	n	k	Wynik	Opis
mon1a.in	20	5	12	test losowy
mon1b.in	1	1	1	test losowy
mon2.in	50	3	25	test losowy
mon3.in	100	10	100	test losowy
mon4.in	500	30	300	test losowy
mon5a.in	4 000	50	2 000	test losowy
mon5b.in	4 000	30	7	test losowy
mon6a.in	10 000	100	17	test losowy
mon6b.in	3 000	2	1 522	test monotoniczny
mon7a.in	12 000	100	11	test losowy
mon7b.in	5 000	3	2 575	test monotoniczny
mon8a.in	15 000	100	12 346	test losowy
mon8b.in	10 000	2	5 072	test monotoniczny
mon8c.in	15 000	100	17	test skokowy
mon9a.in	18 000	100	15 320	test losowy
mon9b.in	20 000	3	10 074	test monotoniczny
mon9c.in	20 000	100	17	test skokowy
mon10a.in	20 000	100	15 700	test losowy
mon10b.in	20 000	1	10 120	test monotoniczny
mon10c.in	20 000	100	15	test skokowy