Jacek Tomasiewicz — Treść zadania, Opracowanie
Aleksander Łukasiewicz, Karol Pokorski — Program

Dostępna pamięć: 128 MB. *OI, etap II, dzień drugi, 13.02.2014*

Ptaszek

W Bajtockim Lesie rośnie w rzędzie n drzew. Na czubku pierwszego drzewa siedzi ptaszek, który chciałby dolecieć na czubek ostatniego drzewa. Ponieważ ptaszek jest jeszcze mały, więc niekoniecznie będzie miał siłę pokonać całą drogę bez zatrzymywania się. Jeśli ptaszek siedzi na czubku drzewa o numerze i, to podczas pojedynczego lotu może on dolecieć do dowolnego z drzew o numerach $i+1$, $i+2$, $\ldots$, $i+k$, po czym musi odpocząć.

Ponadto lot w górę jest dla niego o wiele trudniejszy niż lot w dół. Innymi słowy, ptaszek się męczy, jeśli leci z niższego na wyższe, bądź równe drzewo. W przeciwnym przypadku ptaszek się nie męczy.

Należy tak wybrać drzewa, na których będzie lądował ptaszek, aby zmęczył się jak najmniejszą liczbę razy. Dodatkowo, ptaszek ma kilku kolegów, którzy także chcieliby przedostać się z pierwszego do ostatniego drzewa – mogą być oni bardziej lub mniej wytrzymali od niego (a zatem mogą mieć oni różne wartości k). Pomóż także kolegom ptaszka.

Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną liczbę całkowitą n ($2 \leqslant n \leqslant 1\,000\,000$), oznaczającą liczbę drzew w Bajtockim Lesie. Drugi wiersz wejścia zawiera n liczb całkowitych $d_1, d_2, \ldots, d_n$ ($1 \leqslant d_i \leqslant 10^9$) pooddzielanych pojedynczymi odstępami: d_i oznacza wysokość i-tego drzewa.

Trzeci wiersz wejścia zawiera jedną liczbę całkowitą q ($1 \leqslant q \leqslant 25$), określającą liczbę ptaszków, dla których należy rozważyć przelot. Kolejne q wierszy zawierają opisy ptaszków: w i-tym z tych wierszy znajduje się liczba całkowita k_i ($1 \leqslant k_i \leqslant n-1$), oznaczająca wytrzymałość i-tego ptaszka. Innymi słowy, maksymalna liczba drzew, które może minąć i-ty ptaszek, zanim będzie zmuszony odpocząć, wynosi $k_i - 1$.

W testach wartych łącznie 70% punktów zachodzi dodatkowy warunek: $n \leqslant 100\,000$. W podzbiorze tych testów wartym łącznie 30% punktów zachodzi dodatkowy warunek: $n \leqslant 1000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście dokładnie q wierszy. W i-tym wierszu powinna się znaleźć odpowiedź dla i-tego ptaszka z wejścia. Odpowiedź dla każdego ptaszka składa się z jednej liczby całkowitej, równej minimalnej liczbie razy, w których ptaszek musi podlecieć z niższego na wyższe, bądź równe drzewo.

Przykład

Dla danych wejściowych:

```
9
4 6 3 6 3 7 2 6 5
```

```
2
2
5
```
poprawnym wynikiem jest:
```
2
1
```

Wyjaśnienie do przykładu: *Pierwszy ptaszek może zatrzymać się kolejno na drzewach o numerach 1, 3, 5, 7, 8, 9. Ptaszek będzie się męczył, lecąc z trzeciego drzewa na piąte i z siódmego na ósme.*

Testy „ocen”:

`1ocen`: $n = 11$, $q = 1$, $k_1 = 5$, *wysokości wszystkich drzew są równe, odpowiedzią jest 2 (wystarczy międzylądowanie na szóstym drzewie);*

`2ocen`: $n = 100$, $q = 2$, $k_1 = 5$, $k_2 = 6$, *wysokości drzew są na przemian 1 i 2 – dla obu ptaszków optymalna strategia wybiera odpoczynek co 5 drzew, co daje 11-krotne zmęczenie obu ptaszków;*

`3ocen`: $n = 100$, $q = 1$, $k_1 = 10$, *ciąg wysokości drzew to:* $100, 99, \ldots, 1$, *odpowiedzią jest 0;*

`4ocen`: $n = 1\,000\,000$, $q = 25$, $k_i = i$, $d_{1000} = d_{2000} = d_{3000} = \ldots = d_{1\,000\,000} = 2$, *zaś pozostałe* $d_i = 1$.

Rozwiązanie

We wszystkich rozwiązaniach wynik będziemy obliczać oddzielnie dla każdego ptaszka. Na potrzeby opisu przyjmijmy więc, że rozważamy jednego ptaszka o wytrzymałości k. Dodatkowo, oznaczmy przez w_i minimalną liczbę zmęczeń ptaszka, zakładając, że kończy on podróż na i-tym drzewie.

Rozwiązanie wolne $O(q \cdot n \cdot k)$

Spróbujemy obliczyć wartości $w_1, w_2, \ldots, w_n$ za pomocą programowania dynamicznego. Załóżmy, że obliczamy wartość w_i oraz że mamy już obliczone wyniki dla wszystkich wcześniejszych drzew, czyli wartości $w_1, w_2, \ldots, w_{i-1}$.

Obliczając liczbę zmęczeń potrzebnych na dostanie się do i-tego drzewa, możemy przejrzeć wszystkie drzewa, z których mógł bezpośrednio przylecieć do niego ptaszek. Będzie to tylko k poprzednich drzew, ponieważ tylko tyle może przelecieć ptaszek bez robienia odpoczynku.

Załóżmy, że ptaszek leci z j-tego drzewa na i-te. Jeśli leci z niższego drzewa na wyższe bądź równe ($d_j \leqslant d_i$), to się zmęczy. W związku z tym, łączna liczba zmęczeń wzrośnie o jeden, czyli $w_i = w_j + 1$. W przeciwnym przypadku, gdy ptaszek leci z wyższego drzewa na niższe ($d_j > d_i$), to się nie zmęczy. Łączna liczba zmęczeń pozostanie bez zmian, czyli $w_i = w_j$. Ostatecznie powinniśmy wybrać minimalną liczbę zmęczeń, jaką możemy uzyskać.

```
function obliczZmeczenie(d[1..n])
begin
  w[1] := 0;
  for i := 2 to n do begin
    w[i] := ∞;
    for j := max(1, i − k) to i − 1 do begin
      zmeczenie := 0;
      if d[j] ⩽ d[i] then
        zmeczenie := 1;
      w[i] := min(w[i], w[j] + zmeczenie);
    end
  end
  return w[n];
end
```

Złożoność takiego rozwiązania dla pojedynczego ptaszka wynosi $O(n \cdot k)$, ponieważ dla każdego drzewa przeglądamy k poprzednich drzew. Powtarzając to dla wszystkich ptaszków, uzyskamy złożoność czasową $O(q \cdot n \cdot k)$.

Za takie rozwiązanie można było uzyskać około 30% punktów. Implementacje znajdują się w plikach `ptas1.cpp` i `ptas5.pas`.

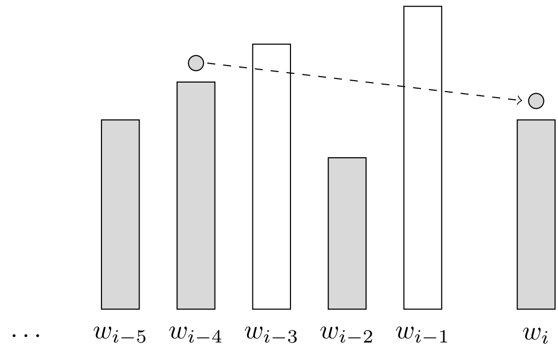
Rozwiązanie wzorcowe $O(q \cdot n)$

Zauważmy, że jeśli mamy dwa drzewa $j < i$ oddalone maksymalnie o k, to liczba zmęczeń ptaszka dla tych drzew nie różni się o więcej niż jeden, czyli $|w_i - w_j| \leqslant 1$.

Załóżmy, że tak nie jest. Jeśli $w_i > w_j$, to moglibyśmy z j-tego drzewa polecieć do i-tego i w najgorszym przypadku uzyskalibyśmy $w_i = w_j + 1$. Jeśli zaś $w_j > w_i$, to moglibyśmy znaleźć takie drzewo $p \leqslant j$, że $w_p \leqslant w_i$ i z p-tego drzewa dałoby się dolecieć do j-tego. W ten sposób uzyskalibyśmy $w_j = w_p + 1 \leqslant w_i + 1$, więc obserwacja jest prawdziwa.

Z powyższą, kluczową obserwacją możemy przejść do właściwego rozwiązania. Załóżmy, że obliczamy wartość w_i oraz że mamy już obliczone wyniki dla wszystkich wcześniejszych drzew. Tak naprawdę, interesuje nas tyko k poprzednich drzew, czyli tych, z których można bezpośrednio dolecieć do i-tego drzewa. Dodatkowo, drzewa te możemy podzielić na dwa zbiory. Niech A będzie zbiorem drzew o najmniejszej wartości w – dokładniej, o wartości $x = \min(w_{i-1}, w_{i-2}, \ldots, w_{i-k})$. Natomiast B to zbiór pozostałych drzew, czyli takich, w których liczba zmęczeń ptaszka będzie o jeden większa (zauważmy, że zbiór B może być pusty, natomiast zbiór A będzie zawsze zawierał co najmniej jedno drzewo).

Chcielibyśmy teraz znaleźć poprzednika i, czyli takie drzewo, z którego opłaca się bezpośrednio dolecieć do i-tego drzewa. Najlepszym miejscem będzie najwyższe drzewo ze zbioru A lub ze zbioru B, ponieważ każde niższe drzewo może dać tylko nie lepszy wynik. Dodatkowo zauważmy, że jeżeli miałoby to być drzewo ze zbioru B, to moglibyśmy równie dobrze wybrać dowolne drzewo ze zbioru A i taki przelot byłby



Rys. 1: Obliczanie wartości w_i (dla $k = 5$). Szarym kolorem oznaczono zbiór A, czyli drzewa z liczbą zmęczeń x. Białym kolorem oznaczono zbiór B, czyli drzewa z liczbą zmęczeń $x + 1$.

nie gorszy. Podsumowując, poprzednikiem i będzie najwyższe drzewo ze zbioru A, patrz rys. 1.

W naszym algorytmie powinniśmy przechowywać oddzielnie oba zbiory. Gdy wraz ze zwiększaniem i będziemy przesuwać się do kolejnych drzew, za każdym razem będziemy usuwać jeden element z któregoś ze zbiorów A, B i wstawiać jeden element do któregoś ze zbiorów. Dodatkowo, gdy napotkamy drzewo o największej dotąd wartości w, wszystkie elementy ze zbioru B trafią do zbioru A. Chcemy więc dobrać strukturę danych z wyróżnionym początkiem i końcem, która umożliwi operacje: wstawiania na koniec, zdejmowania z początku oraz odczytywania maksimum.

Obliczanie maksimów

Najtrudniejszym elementem jest efektywne znajdywanie najwyższego drzewa. Możemy użyć do tego kolejki priorytetowej, struktury `set` z biblioteki STL lub drzewa przedziałowego, opisywanych wielokrotnie w opracowaniach poprzednich zadań olimpijskich. Takie programy będą miały złożoność $O(q \cdot n \log n)$. Za takie rozwiązania można było uzyskać 70% punktów, a przykładowe implementacje trzech wariantów zawarte są w plikach `ptas3.cpp`, `ptas6.cpp` i `ptas7.cpp`.

Najwyższe drzewa możemy znajdywać szybciej, w zamortyzowanym czasie $O(n)$. Taka struktura danych, zwana *K-max* kolejką, wystąpiła już w rozwiązaniach zadań olimpijskich: w zadaniu *Temperatura* z II etapu XVIII Olimpiady [18] lub w zadaniu *Piloci* z III etapu XVII Olimpiady [17].

To kończy opis rozwiązania wzorcowego, działającego w czasie $O(q \cdot n)$. Implementacje znajdują się w plikach `pta.cpp`, `pta1.cpp`, `pta2.cpp` oraz `pta3.pas`, `pta4.pas`.

Testy

Testy zostały podzielone na cztery główne grupy: *losowe* – wygenerowane losowo, zarówno małe testy poprawnościowe, jak i duże testy wydajnościowe; *niskie drzewa* – losowe z małymi wysokościami drzew; *ustalona wytrzymałość* – małe lub duże wytrzymałości ptaszków; *rosnące wysokości* – przedziały drzew tworzą rosnące wysokości.