Piotr Niedźwiedź — Treść zadania, Opracowanie
Jakub Pachocki, Juliusz Sompolski — Program

Dostępna pamięć: 128 MB. *OI, Etap I, 18.10–15.11.2010*

Piorunochron

Postępujące zmiany klimatu zmusiły władze Bajtogrodu do wybudowania dużego piorunochronu, który chroniłby wszystkie budynki w mieście. Wszystkie budynki stoją w rzędzie przy jednej prostej ulicy i są ponumerowane kolejno od 1 do n.

Zarówno wysokości budynków, jak i wysokość piorunochronu wyrażają się nieujemnymi liczbami całkowitymi. Bajtogród dysponuje funduszami na wybudowanie tylko jednego piorunochronu. Co więcej, im wyższy ma być piorunochron, tym będzie droższy.

Aby piorunochron o wysokości p umieszczony na dachu budynku i (o wysokości h_i) mógł skutecznie chronić wszystkie budynki, dla każdego innego budynku j (o wysokości h_j) musi zachodzić następująca nierówność:

$$h_j \leqslant h_i + p - \sqrt{|i-j|}.$$

Tutaj $|i-j|$ oznacza wartość bezwzględną różnicy liczb i oraz j.

Bajtazar, burmistrz Bajtogrodu, poprosił Cię o pomoc. Napisz program, który dla każdego budynku i ($i = 1, \ldots, n$) obliczy, jaka jest minimalna wysokość piorunochronu, który umieszczony na budynku i będzie chronił wszystkie budynki.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($1 \leqslant n \leqslant 500\,000$) oznaczająca liczbę budynków w Bajtogrodzie. W każdym z kolejnych n wierszy znajduje się jedna liczba całkowita h_i ($0 \leqslant h_i \leqslant 1\,000\,000\,000$), oznaczająca wysokość i-tego budynku.

Wyjście

Twój program powinien wypisać na standardowe wyjście n wierszy. W i-tym wierszu powinna znaleźć się nieujemna liczba całkowita p_i, oznaczająca minimalną wysokość piorunochronu na i-tym budynku.

Przykład

Dla danych wejściowych:

```
6
5
3
2
4
2
4
```

poprawnym wynikiem jest:

```
2
3
5
3
5
4
```

Rozwiązanie

Wprowadzenie

Uproszczenie

W zadaniach, w których badany obiekt jest symetryczny, często można rozpatrzyć osobno jego lewą i prawą część, po czym na tej podstawie obliczyć wynik dla całości. W naszym przypadku tak właśnie jest. Wystarczy, że zaprojektujemy algorytm obliczający wynik dla prawostronnych piorunochronów — wówczas dla budynku i szukamy takiej minimalnej wysokości p_i, żeby dla każdego $j \geqslant i$ zachodziło:

$$h_j \leqslant h_i + p_i - \sqrt{|i-j|}.$$

Analogicznie rozważamy wysokość l_i lewostronnego piorunochronu, tj. dla $j \leqslant i$. Ostateczna wysokość i-tego piorunochronu to $\max(l_i, p_i)$.

Od tej pory skupimy się tylko na obliczeniu wartości p_i; wartości l_i obliczymy za pomocą tego samego algorytmu zapisanego dla odwróconego ciągu wysokości budynków.

Wzór na p_i

Zgodnie z treścią zadania, aby piorunochron na dachu budynku i chronił budynek j (dla $j \geqslant i$), musi zachodzić nierówność:

$$h_j \leqslant h_i + p_i - \sqrt{j-i}.$$

Jest ona równoważna następującej:

$$p_i \geqslant h_j + \sqrt{j-i} - h_i.$$

Wprowadźmy oznaczenie $\mathit{piorunochron}(i,j) = h_j + \sqrt{j-i} - h_i$. Skoro szukamy możliwie najmniejszego całkowitego p_i, to musi być ono równe:

$$p_i = \max_{j \geqslant i} \left\{ \lceil \mathit{piorunochron}(i,j) \rceil \right\}. \tag{1}$$

W powyższym zapisie wyrażenie $\lceil x \rceil$ oznacza najmniejszą liczbę naturalną nie mniejszą niż x („sufit z x").

Obliczanie pierwiastków

Niezależnie od tego, jaką metodą będziemy próbowali rozwiązać to zadanie, nie unikniemy konieczności obliczania pierwiastków liczb całkowitych (lub tych pierwiastków zaokrąglonych w górę, zależnie od rozwiązania). Języki programowania wykorzystywane na zawodach (C/C++, Pascal) mają wbudowaną funkcję `sqrt`, która pozwala obliczać pierwiastki z liczb zmiennopozycyjnych. Warto jednak pamiętać o tym, że funkcja ta jest dosyć powolna (zachęcamy Czytelników do poeksperymentowania, ile wywołań tej funkcji można wykonać w ciągu, powiedzmy, sekundy). Dlatego dobrym pomysłem w każdym, właściwie, rozwiązaniu było spamiętanie na początku pierwiastków wszystkich liczb całkowitych od 0 do $n-1$ (lub sufitów z nich), a potem korzystanie już tylko z takich stablicowanych wartości. W dalszej części opisu pojawiają się określenia typu „wersja zoptymalizowana rozwiązania" — oznaczają one, między innymi, zastosowanie podanego tu usprawnienia.

Rozwiązanie siłowe

To rozwiązanie opiera się na bezpośrednim zastosowaniu wzoru (1). Złożoność czasowa to $O(n^2)$, a stosowne implementacje można znaleźć w plikach `pios1.cpp`, `pios2.pas`, `pios3.cpp` i `pios4.pas`. Można było za nie uzyskać, w zależności od optymalizacji programu (w tym użycia pewnych sprytnych heurystyk, szybko wykluczających niektóre budynki z rozważań), od 27 do nawet 45 punktów.

Rozwiązania $O(n\sqrt{n})$

Zauważmy, że wyrażenie $\lceil\sqrt{j-i}\rceil$ przyjmuje wartości od 0 do $\lceil\sqrt{n}\rceil$. Dla danego budynku i, budynki o numerach $j \geqslant i$ można pogrupować w przedziały, w których zachodzi $\lceil\sqrt{j-i}\rceil = k$, a dokładniej:

$$L_{i,k} \stackrel{\text{def}}{=} i+(k-1)^2+1 \quad \leqslant \quad j \quad \leqslant \quad i+k^2 \stackrel{\text{def}}{=} R_{i,k}.$$

Możemy w takim razie zapisać wzór (1) w nowej postaci:

$$p_i = \max_{k\in 0..\lceil\sqrt{n}\rceil} \left\{ k - h_i + \max_{L_{i,k}\leqslant j\leqslant R_{i,k}} h_j \right\}.$$

Zakładając, że dla danego przedziału potrafilibyśmy w miarę szybko znaleźć największą wartość, która w nim występuje, to wartość p_i moglibyśmy obliczyć, badając jedynie $O(\sqrt{n})$ takich przedziałów. Znajdowanie maksimum na przedziale jest już klasycznym problemem, który można rozwiązać na wiele sposobów. Przedstawimy w skrócie dwa z nich.

Drzewo przedziałowe

Drzewo przedziałowe to struktura danych, która zajmuje $O(n)$ pamięci, można ją skonstruować w czasie $O(n)$ i umożliwia ona wyszukiwanie maksimum na przedziale w złożoności czasowej $O(\log n)$. Więcej na temat drzew przedziałowych można przeczytać np. w opracowaniu zadania *Tetris 3D* z XIII Olimpiady Informatycznej [13] albo na stronie `http://was.zaa.mimuw.edu.pl/?q=node/8`.

Złożoność czasowa rozwiązania używającego drzew przedziałowych to $O(n\sqrt{n}\log n)$, a pamięciowa to $O(n)$. Implementacje znajdują się w plikach `pios5.cpp` i `pios6.pas`.

Struktura danych podobna do słownika podsłów bazowych[1]

Jest to dwuwymiarowa tablica postaci:

$$t[k][i] = \max_{i \leqslant j < i+2^k} h_j \qquad \text{dla } k = 0, 1, \ldots, \lfloor \log n \rfloor,\ i = 1, 2, \ldots, n - 2^k + 1.$$

Tablica ta ma rozmiar $O(n \log n)$ i konstruuje się ją w tej samej złożoności czasowej. Najpierw wyznaczamy $t[0][i] = h_i$. Następnie dla kolejnych k mamy:

$$t[k][i] = \max(t[k-1][i],\ t[k-1][i + 2^{k-1}]).$$

Po wypełnieniu wszystkich komórek takiej tablicy maksima na przedziałach możemy już wyznaczać w czasie stałym:

$$\max_{l \leqslant j \leqslant r} h_j \quad = \quad \max\left(t[k_{r-l+1}][l],\ t[k_{r-l+1}][r - 2^{k_{r-l+1}} + 1]\right),$$

przy czym $k_d \stackrel{\text{def}}{=} \lfloor \log_2 d \rfloor$. Wszystkie wartości k_d możemy obliczyć w złożoności czasowej $O(n)$, a następnie spamiętać w tablicy. Używając tej struktury danych do wyznaczania maksimów na przedziale, uzyskamy algorytm o łącznej złożoności czasowej $O(n \log n + n\sqrt{n}) = O(n\sqrt{n})$ i pamięciowej $O(n \log n)$. Takie rozwiązanie zostało zaimplementowane w plikach `piob3.cpp` i `piob4.pas`.

Za rozwiązania z opisywanych grup można było uzyskać od 27 do ok. 50 punktów.

Alternatywne rozwiązanie $O(n\sqrt{n})$

Opis tego rozwiązania rozpocznijmy od dwóch prostych spostrzeżeń.

Obserwacja 1. Niech M oznacza numer najwyższego budynku. Wtedy dla każdego budynku j, dla którego $h_j < h_M - \lceil \sqrt{n} \rceil$, oraz każdego budynku i zachodzi nierówność:

$$\lceil \mathit{piorunochron}(i,j) \rceil = h_j + \left\lceil \sqrt{|j-i|} \right\rceil - h_i <$$
$$< h_M + \left\lceil \sqrt{|M-i|} \right\rceil - h_i = \lceil \mathit{piorunochron}(i,M) \rceil .$$

Stąd, przy wyznaczaniu wysokości piorunochronu możemy ograniczyć się do rozpatrywania budynków j o wysokościach $h_M - \lceil \sqrt{n} \rceil \leqslant h_j \leqslant h_M$.

Obserwacja 2. Rozważmy trzy budynki i, a, b, takie że $i \leqslant a < b$ oraz $h_a \leqslant h_b$. Wówczas:

$$\lceil \mathit{piorunochron}(i,a) \rceil = h_a + \left\lceil \sqrt{a-i} \right\rceil - h_i$$
$$\leqslant h_b + \left\lceil \sqrt{b-i} \right\rceil - h_i = \lceil \mathit{piorunochron}(i,b) \rceil .$$

[1]Słownik podsłów bazowych to tekstowa struktura danych służąca do przypisywania podsłowom wyjściowego tekstu jednoznacznych identyfikatorów. Więcej o tej strukturze można przeczytać w książce [20] (algorytm Karpa-Millera-Rosenberga, KMR) lub w opisie rozwiązania zadania *Powtórzenia* z VII Olimpiady Informatycznej [7].

Innymi słowy, wyznaczając maksimum we wzorze (1), możemy nie rozpatrywać budynku a, jeśli położony na prawo od niego budynek b jest co najmniej tak samo wysoki jak a.

Wykorzystując podane obserwacje, możemy dojść do efektywnego, a zarazem dosyć prostego rozwiązania. Przechodzimy budynki od prawej do lewej, zapamiętując dla każdej wysokości od $h_M - \lceil\sqrt{n}\rceil$ do h_M największy numer budynku o tej wysokości. Są to tak zwane *prawostronne maksima.* Aby wyznaczyć wysokości prawostronnych piorunochronów, dla każdego budynku i sprawdzamy $O(\sqrt{n})$ kandydatów na maksimum i wybieramy największego z nich. Być może nie znajdziemy żadnego kandydata, np. dla budynku o numerze n — wtedy możemy uznać, że wysokość piorunochronu to 0. Zauważmy, że nasz algorytm może nie obliczyć prawdziwego p_i, gdyż obliczana przez nas wartość to tak naprawdę:

$$p'_i \stackrel{\text{def}}{=} \max_{j \geqslant i,\ h_M - \lceil\sqrt{n}\rceil \leqslant h_j} \{\lceil \mathit{piorunochron}(i,j) \rceil\}. \tag{2}$$

Jednak, na mocy Obserwacji 1, to nam wystarcza. Innymi słowy, jeśli $p'_i < p_i$, to ostateczna wysokość piorunochronu będzie poprawna po uwzględnieniu lewostronnego piorunochronu. Sumaryczna złożoność czasowa powyższego algorytmu to $O(n\sqrt{n})$, a pamięciowa to $O(n)$.

Implementacje tego rozwiązania można znaleźć w plikach `pios7.cpp`, `pios8.pas`, `pios9.cpp` i `pios10.pas`. Na zawodach takie rozwiązania, w zależności od jakości implementacji, uzyskiwały od 50 do 75 punktów.

Rozwiązanie wzorcowe

Aby uzyskać rozwiązanie wzorcowe, musimy poczynić jeszcze jedną obserwację, opisującą „podział strefy wpływów” dwóch budynków.

Lemat 1. Jeśli mamy budynki $a < b$ i $h_a > h_b$, to istnieje takie $q_{a,b}$, $0 \leqslant q_{a,b} \leqslant a-1$, że dla $i \in [1,\ q_{a,b}]$:

$$\mathit{piorunochron}(i,a) \geqslant \mathit{piorunochron}(i,b)$$

a dla $i \in [q_{a,b}+1,\ a-1]$:

$$\mathit{piorunochron}(i,a) < \mathit{piorunochron}(i,b).$$

Dowód: Zauważmy, że funkcja

$$f(x) = \sqrt{x+c} - \sqrt{x}$$

dla dowolnej stałej $c > 0$ jest malejąca[2] na przedziale $(0, \infty)$.

[2]Co ciekawe, funkcja zdefiniowana podobnym wzorem $\lceil\sqrt{x+c}\rceil - \lceil\sqrt{x}\rceil$ nie musi już być nierosnąca (dlaczego?). To uzasadnia, dlaczego w sformułowaniu Lematu 1 wyjątkowo porzuciliśmy wszechobecne w tym opracowaniu sufity z *piorunochronów* (czyli sufity z pierwiastków).

Aby się o tym przekonać, zapiszmy ją w nieco inny sposób:

$$f(x) = \sqrt{x+c} - \sqrt{x} = \frac{\left(\sqrt{x+c} - \sqrt{x}\right)\left(\sqrt{x+c} + \sqrt{x}\right)}{\sqrt{x+c} + \sqrt{x}} = \frac{c}{\sqrt{x+c} + \sqrt{x}}$$

Nietrudno zauważyć, że mianownik jest funkcją rosnącą, skąd f jest malejąca.

Wreszcie z faktu, że f jest malejąca, wnosimy, że im większe i, tym mniejsza wartość różnicy $\sqrt{a-i} - \sqrt{b-i}$, co wobec zależności

$$piorunochron(i,a) - piorunochron(i,b) = \sqrt{a-i} - \sqrt{b-i} + h_a - h_b$$

uzasadnia istnienie parametru $q_{a,b}$, dla którego nierówności z lematu są spełnione. ■

Warto zauważyć, że wartości $q_{a,b}$ możemy wyznaczać w czasie $O(\log n)$ za pomocą wyszukiwania binarnego.

Możemy już teraz przejść do opisu algorytmu wzorcowego. Zaczynamy od znalezienia wszystkich prawostronnie maksymalnych budynków o wysokościach nie mniejszych niż $h_M - \lceil\sqrt{n}\rceil$, podobnie jak w poprzednim rozwiązaniu powolnym. Niech $a_1 < a_2 < \ldots < a_d$ będą indeksami kolejnych takich budynków, oczywiście $h_{a_i} > h_{a_{i+1}}$. Przypomnijmy, że na mocy Obserwacji 1 i 2, są to jedyne budynki potrzebne przy obliczaniu wysokości prawostronnych piorunochronów (a dokładniej, przy obliczaniu wartości p'_i ze wzoru (2)).

Dla tak zdefiniowanego ciągu (a_i) chcielibyśmy wyznaczyć punkty podziału ciągu wszystkich budynków na fragmenty, których prawostronny piorunochron jest wyznaczony przez poszczególne a_i. Jako stosowne punkty podziału należałoby przyjąć kolejne wartości q_{a_1,a_2}, q_{a_2,a_3}, $\ldots$ z Lematu 1. Taki podział jest poprawny, o ile dla każdego j zachodzi $q_{a_j,a_{j+1}} < q_{a_{j+1},a_{j+2}}$. Jeśli jednak w pewnym momencie mamy $q_{a_j,a_{j+1}} \geqslant q_{a_{j+1},a_{j+2}}$, to to po prostu oznacza, że budynek a_{j+1} nie ma wpływu na wysokość prawostronnego piorunochronu żadnego innego budynku, czyli możemy go wyeliminować z ciągu (a_i), zastępując dwa dotychczasowe punkty podziału punktem $q_{a_j,a_{j+2}}$. To postępowanie powtarzamy do momentu, gdy lista punktów podziału jest rosnąca.

W ten sposób uzyskujemy, potencjalnie krótszy, ciąg prawostronnych maksimów $a'_1, \ldots, a'_g$, z odpowiadającym mu ciągiem punktów podziału:

$$s_i = q_{a'_i,a'_{i+1}} \quad \text{dla } 1 \leqslant i \leqslant g-1, \quad \text{a dodatkowo } s_0 = 0,\ s_g = a'_g.$$

Wówczas dla każdego $m \in [0, g-1]$ oraz $i \in [s_m + 1, s_{m+1}]$ zachodzi:

$$p'_i = \left\lceil piorunochron(i, a'_{m+1}) \right\rceil. \tag{3}$$

To oznacza, że znając ciągi (a'_i) i (s_i), wartości p'_i możemy obliczyć w czasie $O(n)$.

Poniższy pseudokod stanowi ilustrację tego, jak można wyznaczyć ciągi (a'_i) i (s_i) za jednym przejściem przez ciąg (tablicę) h. Elementy ciągów odkładane są na dwa stosy, A oraz S, oba początkowo puste. Zakładamy, że mamy zaimplementowane wyszukiwanie binarne *obliczQ*, które dla dwóch budynków a, b znajduje $q_{a,b}$.

```
A, S := puste stosy;
for i := n downto 1 do begin
  if (h[i] ⩾ h[M] − ⌈√n⌉) and (A.empty() or (h[i] > h[A.top()])) then begin
    while (not A.empty()) and (S.top() ⩽ obliczQ(i, A.top())) do begin
      A.pop(); S.pop();
    end
    if not A.empty() then begin
      A.push(i); S.push(obliczQ(i, A.top()));
    end else begin
      A.push(i); S.push(i);
    end
  end
end
```

Widzimy, że wyznaczenie naszych podciągów wymaga $O(\sqrt{n})$ wywołań funkcji *obliczQ*, co sumarycznie daje złożoność $O(\sqrt{n}\log n)$. Mając ciągi (a'_i) oraz (s_i), wszystkie wartości p'_i możemy obliczyć w czasie $O(n)$, korzystając ze wzoru (3). W ten sposób uzyskujemy algorytm działający w czasie $O(n+\sqrt{n}\log n) = O(n)$.

Rozwiązanie to można znaleźć w plikach `pio.cpp` i `pio1.pas`. Na zawodach otrzymywało, rzecz jasna, maksymalną punktację.

Testy

Rozwiązania zawodników były sprawdzane na 11 zestawach danych wejściowych. Zestawy o numerach od 6 do 11 składały się z trzech pojedynczych testów, przy czym test *a* w każdej grupie to duży test losowy (testy *9a* i *10a* wymuszają maksymalne możliwe odpowiedzi dla budynku 1), test *b* to duży test losowy o bardzo małej liczbie różnych największych wysokości, a test *c* to test wymuszający na wolniejszych rozwiązaniach wykonanie $\Omega(n\sqrt{n})$ operacji. Poniżej tabela z zestawieniem wszystkich testów.

Nazwa	n	Opis
pio1.in	100	niewielki test poprawnościowy
pio2.in	200	niewielki test poprawnościowy
pio3.in	1 000	niewielki test poprawnościowy
pio4.in	30 000	większy test poprawnościowo-wydajnościowy
pio5.in	50 000	większy test poprawnościowo-wydajnościowy
pio6[abc].in	64 915	grupa dużych testów
pio7[abc].in	98 117	grupa dużych testów
pio8[abc].in	228 423	grupa dużych testów
pio9[abc].in	351 234	grupa dużych testów
pio10[abc].in	500 000	grupa dużych testów
pio11[abc].in	500 000	grupa dużych testów