Wojciech Rytter
Treść zadania, Opracowanie

Marek Cygan
Program

OI, Etap III, dzień 2, 7–04–2005

Autobus

Ulice Bajtogrodu tworzą szachownicę — prowadzą z północy na południe lub ze wschodu na zachód. Ponadto każda ulica prowadzi na przestrzał przez całe miasto — każda ulica biegnąca z północy na południe krzyżuje się z każdą ulicą biegnącą ze wschodu na zachód i vice versa. Ulice prowadzące z północy na południe są ponumerowane od 1 do n, w kolejności z zachodu na wschód. Ulice prowadzące ze wschodu na zachód są ponumerowane od 1 do m, w kolejności z południa na północ. Każde skrzyżowanie i-tej ulicy biegnącej z północy na południe i j-tej ulicy biegnącej ze wschodu na zachód oznaczamy parą liczb (i,j) (dla $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$).

Po ulicach Bajtogrodu kursuje autobus. Zaczyna on trasę przy skrzyżowaniu $(1,1)$, a kończy przy skrzyżowaniu (n,m). Ponadto autobus może jechać ulicami tylko w kierunku wschodnim i/lub północnym.

Przy pewnych skrzyżowaniach oczekują na autobus pasażerowie. Kierowca autobusu chce tak wybrać trasę przejazdu autobusu, aby zabrać jak najwięcej pasażerów. (Zakładamy, że bez względu na wybór trasy i tak wszyscy pasażerowie zmieszczą się w autobusie.)

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis siatki ulic oraz liczbę pasażerów czekających przy poszczególnych skrzyżowaniach,*
- *obliczy, ilu maksymalnie pasażerów może zabrać autobus,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu pliku wejściowego zapisane są trzy dodatnie liczby całkowite n, m i k — odpowiednio: liczba ulic biegnących z północy na południe, liczba ulic biegnących ze wschodu na zachód i liczba skrzyżowań, przy których pasażerowie czekają na autobus ($1 \leqslant n \leqslant 10^9$, $1 \leqslant m \leqslant 10^9$, $1 \leqslant k \leqslant 10^5$).

Kolejne k wierszy opisuje rozmieszczenie pasażerów czekających na autobus, w jednym wierszu opisani są pasażerowie czekający na jednym ze skrzyżowań. W wierszu $(i+1)$-szym znajdują się trzy dodatnie liczby całkowite x_i, y_i i p_i, oddzielone pojedynczymi odstępami, $1 \leqslant x_i \leqslant n$, $1 \leqslant y_i \leqslant m$, $1 \leqslant p_i \leqslant 10^6$. Taka trójka liczb oznacza, że przy skrzyżowaniu (x_i, y_i) oczekuje p_i pasażerów. Każde skrzyżowanie pojawia się w danych wejściowych co najwyżej raz. Łączna liczba oczekujących pasażerów nie przekracza 1 000 000 000.

Wyjście

Twój program powinien na wyjściu wypisać jeden wiersz zawierający jedną liczbę całkowitą — maksymalną liczbę pasażerów, których może zabrać autobus.

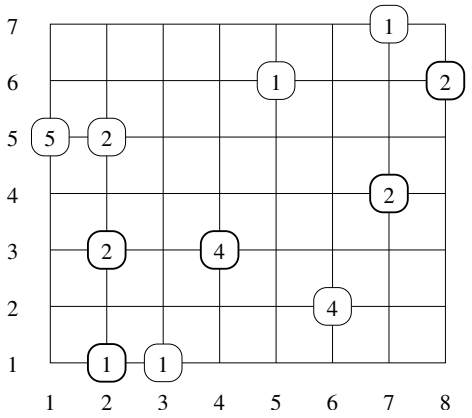
Przykład

Dla danych wejściowych:

```
8 7 11
4 3 4
6 2 4
2 3 2
5 6 1
2 5 2
1 5 5
2 1 1
3 1 1
7 7 1
7 4 2
8 6 2
```

poprawnym wynikiem jest:

```
11
```



W tym przypadku Twój program powinien podać wynik 11. Na rysunku zaznaczono skrzyżowania, przez które przejedzie autobus zabierając 11 pasażerów.

Rozwiązanie

Najbardziej naturalną techniką algorytmiczną, którą można zastosować do rozwiązania zadania, jest *programowanie dynamiczne*. Metoda ta polega na wyznaczeniu dość licznego *zbioru podproblemów*, które następnie rozwiązujemy w *odpowiedniej kolejności*, tak by w trakcie rozwiązywania kolejnego z nich móc wykorzystać uzyskane wcześniej wyniki.

W naszej sytuacji podproblemem będzie wyznaczenie maksymalnej liczby pasażerów, których autobus może dowieźć do konkretnego przystanku. Niech $\mathcal{X}$ oznacza zbiór przystanków (punktów) i niech wartością $W[v]$ dla punktu $v \in \mathcal{X}$ będzie maksymalna liczba pasażerów, których może zabrać autobus dojeżdżając do punktu v (włącznie z pasażerami stojącymi na przystanku v). Oznaczymy również porządek, w jakim autobus może dojeżdżać do przystanków, tzn. $u \lhd v$, dla $u, v \in \mathcal{X}$, jeśli $u \neq v$ i autobus może dojechać z przystanku u do przystanku v zgodnie z zasadami podanymi w zadaniu.

Niech $\{(x_i, y_i, p_i) : 1 \leqslant i \leqslant k\}$ będzie zbiorem danych wejściowych, gdzie x_i jest numerem kolumny, y_i jest numerem wiersza, p_i jest wartością przypisaną do i-tego punktu (liczbą pasażerów na i-tym przystanku). Załóżmy, że punkt (n, m) (skrajnie północno-wschodnie skrzyżowanie) też należy do zbioru wejściowego. Jeśli nie, to uzupełnimy dane o trójkę $(n, m, 0)$ (zwiększając także odpowiednio wartość k), co nie zmieni rozwiązania.

Pierwszy algorytm

Teraz wystarczy przeglądać podproblemy w dowolnym porządku zgodnym z porządkiem $\lhd$ i mamy pierwszy algorytm (zauważmy, że takim porządkiem może być na przykład przeglądanie punktów kolumnami od lewej do prawej, a w każdej kolumnie wierszami od najniższego do najwyższego; równie dobra jest kolejność od najniższego do najwyższego wiersza, a w każdym wierszu od lewej do prawej). Zakładamy, że początkowo wartością $W[v]$ jest liczba pasażerów czekających na przystanku v.

```
1:  procedure Algorytm „brutalny”
2:    begin
3:      forall v ∈ X w jakimkolwiek porządku zgodnym z ◁
4:        do
5:          W[v]:=W[v]+max{W[w]: w ◁ v };
6:      return W[(n,m)];
7:    end
```

Przedstawiony algorytm działa w czasie $O(k^2)$ (k iteracji pętli (3), a znalezienie każdego maksimum w wierszu (5) wymaga czasu $O(k)$). Pokażemy teraz, że możliwe jest przyśpieszenie jego działania.

Algorytm sprytniejszy

Zdefiniujmy dwa dodatkowe porządki na zbiorze punktów (x_i, y_i).

Definicja 1 Porządek *wierszowo-kolumnowy*:

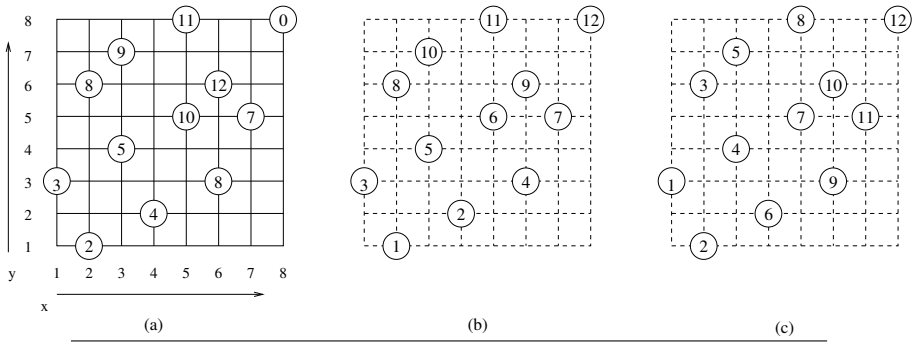
$$(a,b) \overset{w}{\prec} (c,d) \equiv [(b < d) \text{ lub } (b = d \text{ i } a < c)]$$

oraz porządek *kolumnowo-wierszowy*:

$$(a,b) \overset{k}{\prec} (c,d) \equiv [(a < c) \text{ lub } (a = c \text{ i } b < d)]$$

Przykład. Rozważmy dane: (2,1,2), (4,2,4), (1,3,3), (6,3,8), (3,4,5), (5,5,10), (7,5,7), (2,6,8), (6,6,12), (3,7,9), (5,8,11), (8,8,0).

Na rysunku 1 przedstawione są liczby pasażerów czekających na przystankach oraz numeracja wierszowo-kolumnowa i kolumnowo-wierszowa przystanków.



Rys. 1: (a) Dane wejściowe. (b) Porządek wierszowo-kolumnowy przystanków. (c) Porządek kolumnowo-wierszowy przystanków.

Redukcja rozmiaru współrzędnych

Teraz możemy nieco uprościć dane wejściowe. Niech $k_1 < k_2 < \ldots < k_r$ będą współrzędnymi kolumn, w których znajduje się przynajmniej jeden punkt. Podobnie niech $w_1 < w_2 < \ldots < w_s$ będą numerami wierszy, w których znajduje się przynajmniej jeden punkt. Zastąpmy teraz punkt (x_i, y_i) znajdujący się w kolumnie k_j i w wierszu w_l przez punkt (j,l). Zauważmy, że punkty po transformacji pozostają w takim samym porządku względem relacji $\lhd$, relacji $\overset{k}{\prec}$ oraz relacji $\overset{w}{\prec}$. Mamy natomiast gwarancje, że wszystkie współrzędne mieszczą się w przedziale $[1,k]$.

Algorytm

Uporządkujmy dane punkty (x_i, y_i) zgodnie z porządkiem wierszowo-kolumnowym, tzn.

$$(x_j, y_j) \overset{w}{\prec} (x_l, y_l), \text{ dla } j < l.$$

Następnie przyjmijmy, że $(i_1, i_2, \ldots i_k)$ to numery punktów wypisane w porządku kolumnowo-wierszowym, tzn.

$$(x_{i_j}, y_{i_j}) \overset{k}{\prec} (x_{i_l}, y_{i_l}), \text{ dla } j < l.$$

Oba porządki możemy wyznaczyć w czasie liniowym korzystając z algorytmu *radix-sort* (możemy go zastosować, bo po redukcji wartości, według których sortujemy, są niewielkie).

Załóżmy, że początkowo mamy tablicę zawierającą liczby pasażerów na kolejnych przystankach według porządku wierszowo-kolumnowego — oznaczmy ją tym razem $P = [p_1, p_2, \ldots p_k]$. Niech natomiast W będzie, jak poprzednio, tablicą, w której będziemy zapisywać wyliczane kolejno wyniki.

```
1:  procedure Algorytm sprytniejszy
2:    begin
3:      Wypełnij tablicę W zerami;
4:      Oblicz porządek kolumnowo-wierszowy punktów N=[i_1, i_2, ... i_k];
5:      for j:=1 to k do
6:        W[N[j]]=P[N[j]]+max{W[t] : t < N[j]};
7:      return W[k]
8:    end
```

Przykład (c.d.) W naszym przykładzie mamy początkowo:

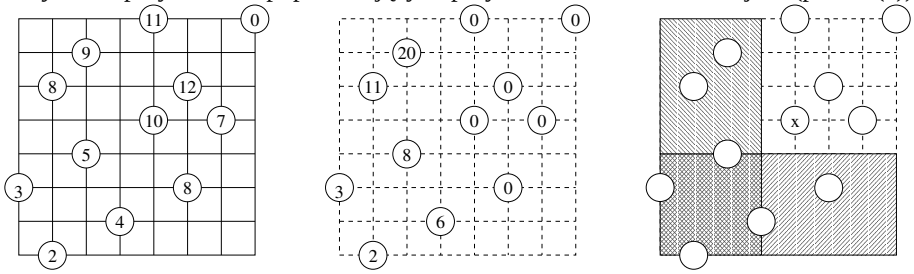
$$P = [2,4,3,8,5,10,7,8,12,9,11,0], \text{ oraz } N = [3,1,8,5,10,2,6,11,4,9,7,12].$$

Załóżmy, że wykonaliśmy już sześć iteracji pętli (4). Wówczas zawartość tablicy W jest następująca (patrz też rysunek 2(b)):

$$W = [2,6,3,0,8,0,0,11,0,20,0,0].$$

Dla sześciu pierwszych punktów w porządku kolumnowo-wierszowym jest to już poprawny wynik, a pozostałe wartości są zerami. Podczas obliczania kolejnej wartości $W[N[7]] = W[6]$ wybieramy maksimum ze zbioru wartości $\{W[1],\ldots,W[5]\}$. Wśród nich są dobrze policzone wartości $\{W[1],W[2],W[3],W[5]\}$ oraz nieistotna (zerowa) wartość $\{W[4]\}$.

Co najważniejsze, dobrze wyliczone wartości $\{W[1],W[2],W[3],W[5]\}$, to są wartości dla *wszystkich* przystanków poprzedzających przystanek numer 6 w relacji $\lhd$ (patrz 2(c)).



(a) (b) (c)

Rys. 2: (a) Dane wejściowe — tablica P. (b) Stan tablicy W po sześciu iteracjach pętli (4). (c) Zaznaczony obszar, w którym znajdują się przystanki wcześniejsze od x według porządku wierszowo-kolumnowego (4 dolne wiersze), przystanki wcześniejsze od x według porządku kolumnowo-wierszowego (4 kolumny z lewej) oraz przystanki wcześniejsze według relacji $\lhd$ (część wspólna powyższych obszarów).

Złożoność i poprawność algorytmu

Algorytm nazwaliśmy sprytniejszym, ponieważ teraz maksimum obliczamy dla łatwiejszego obliczeniowo zbioru — początkowego segmentu tablicy W. Operację $\max\{W[t] : t < N[j]\}$

można wykonać w czasie logarytmicznym, jeśli umieścimy elementy tablicy W w liściach regularnego *kopca*, patrz [14]. W każdym wewnętrznym węźle v kopca jest zawarta maksymalna wartość liścia z poddrzewa o korzeniu v.

Zamiast kopca można zastosować także zbalansowane drzewa uporządkowane (na przykład drzewo AVL), ale kopiec jest znacznie prostszy w implementacji. Można go zapisać w tablicy, a operacje przechodzenia po drzewie realizować za pomocą prostych operacji arytmetycznych.

Czas działania algorytmu „sprytnego" wynosi $O(k \log k)$.

Poprawność algorytmu wynika stąd, że obliczamy wartości w pewnej kolejności zgodnej z porządkiem $\lhd$, a jeśli licząc wartość dla pewnego punktu v, sięgamy po wartość pewnego punktu w (licząc max), to albo $w \lhd v$ (wtedy wartość $W[w]$ jest już dobrze policzona), albo $W[w] = 0$ (co nie ma wpływu na wynik). Mówiąc w sposób *nieprzyjemnie formalny* relacja częściowego porządku $\lhd$ jest częścią wspólną liniowego (pełnego) porządku wierszowo-kolumnowego i kolumnowo-wierszowego (patrz także raz jeszcze rysunek 2(c)).

Testy

Rozwiązania zawodników były sprawdzane na 10 testach. Zostały one w większości wygenerowane losowo.
W poniższej tabeli liczby n, m to rozmiary siatki ulic, natomiast k to liczba przystanków.

Nazwa	n	m	k
aut1.in	50	50	100
aut2.in	500	500	1000
aut3.in	5000	5000	2000
aut4.in	50000	50000	2000
aut5.in	300000	100000	10000
aut6.in	500000	500000	30000
aut7.in	1000000	100000	50000
aut8.in	10000000	20000000	70000
aut9.in	50000000	400000000	90000
aut10.in	1000000000	1000000000	100000