Michał Włodarczyk
Treść zadania, Opracowanie

Karol Pokorski
Program

Dostępna pamięć: 256 MB. *OI, etap III, dzień pierwszy, 16.04.2015*

Myjnie

Bajtazar ma zamiar wystartować w przetargu na zbudowanie n myjni samochodowych wzdłuż głównej drogi ekspresowej w Bajtocji. Zanim jednak to zrobi, chciałby przeliczyć, czy to przedsięwzięcie mu się opłaci.

W tym celu zlecił profesjonalnej firmie badanie rynku. Z wyników badań wynika, że drogą będzie jeździć m potencjalnych klientów oraz że i-ty będzie jeździł odcinkiem pomiędzy myjniami a_i i b_i (włącznie) oraz jest zainteresowany myciem samochodu, o ile cena w myjni nie będzie wyższa niż c_i bajtalarów. Bajtazar w każdej myjni ustala cenę niezależnie. Zakładając, że każdy klient wybierze najtańszą myjnię na swojej trasie (lub nie wybierze żadnej, jeżeli we wszystkich ceny przekroczą jego budżet), Bajtazar chce dobrać takie ceny w myjniach, by zmaksymalizować swój sumaryczny zysk.

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite n i m ($1 \leqslant n \leqslant 50$, $1 \leqslant m \leqslant 4000$) oddzielone pojedynczym odstępem, oznaczające liczbę myjni i liczbę klientów. Myjnie ponumerowane są liczbami od 1 do n. W kolejnych m wierszach znajdują się opisy klientów: i-ty z tych wierszy zawiera trzy liczby całkowite a_i, b_i i c_i ($1 \leqslant a_i \leqslant b_i \leqslant n$, $1 \leqslant c_i \leqslant 500\,000$) pooddzielane pojedynczymi odstępami, oznaczające, że i-ty klient jeździ pomiędzy myjniami a_i a b_i i posiada budżet c_i bajtalarów.

W testach wartych łącznie 75% punktów zachodzi dodatkowy warunek $m \leqslant 250$.

Wyjście

Pierwszy wiersz standardowego wyjścia powinien zawierać jedną liczbę całkowitą s oznaczającą maksymalny sumaryczny zysk Bajtazara w bajtalarach. Drugi wiersz powinien zawierać przykładowy cennik, pozwalający uzyskać zysk s, a konkretnie ciąg n liczb całkowitych $p_1, p_2, \ldots, p_n$ ($1 \leqslant p_i \leqslant 500\,000$) pooddzielanych pojedynczymi odstępami: liczba p_i ma oznaczać cenę w i-tej myjni. Jeżeli jest więcej niż jedna poprawna odpowiedź, Twój program powinien wypisać dowolną z nich.

Przykład

Dla danych wejściowych:

```
7 5
1 4 7
3 7 13
5 6 20
6 7 1
1 2 5
```

poprawnym wynikiem jest:

```
43
5 5 13 13 20 20 13
```

Ocenianie

Jeśli pierwszy wiersz wyjścia będzie nieprawidłowy, otrzymasz 0 punktów za test. Jeśli pierwszy wiersz wyjścia będzie prawidłowy, jednak reszta wyjścia nie będzie prawidłowa, otrzymasz 60% punktów za test. Stanie się tak, nawet jeśli odpowiedź nie będzie zgodna z formatem wyjścia (np. zostanie wypisany tylko jeden wiersz wyjścia lub zostaną wypisane więcej niż dwa wiersze, lub cennik myjni będzie nieprawidłowy lub w złym formacie).

Testy „ocen":

`1ocen`: *$n = 5$, $m = 2$; pierwszy klient przejeżdża obok wszystkich myjni i ma budżet 10, drugi zaś przejeżdża tylko obok trzeciej myjni i ma budżet 9; w optymalnym rozwiązaniu cena paliwa w trzeciej myjni powinna być równa 9, a w pozostałych myjniach powinna być większa lub równa 9 – wówczas obaj klienci kupią paliwo w cenie 9 bajtalarów i zysk Bajtazara będzie równy $2 \cdot 9 = 18$;*

`2ocen`: *$n = 2$, $m = 8$; trzech klientów przejeżdża obok wszystkich myjni (mają budżet 3), trzech klientów przejeżdża tylko obok pierwszej myjni (mają budżet 1) oraz dwóch klientów przejeżdża tylko obok drugiej myjni (mają budżet 1); w optymalnym rozwiązaniu cena paliwa w każdej myjni powinna być równa 3 bajtalary – wówczas tylko pierwsi trzej klienci kupią paliwo i zysk Bajtazara będzie równy $3 \cdot 3 = 9$;*

`3ocen`: *$n = 50$, $m = 1000$; i-ty klient jedzie od pierwszej do ostatniej myjni i ma budżet $500 \cdot i$; w optymalnym rozwiązaniu cena paliwa w każdej myjni powinna być równa 250 000 bajtalarów – wówczas tylko klienci o numerach od 500 do 1000 kupią paliwo i zysk Bajtazara będzie równy $501 \cdot 250\,000 = 125\,250\,000$.*

Rozwiązanie

Często pierwszym krokiem w pracy nad zadaniem algorytmicznym jest postawienie sobie pytania, „od której strony" będziemy konstruować rozwiązanie. Przykładowe pytania, które powinniśmy sobie zadać, to:

- „Kto będzie korzystać z myjni znajdującej się na początku drogi?"
- „Gdzie będzie mył samochód najbogatszy klient?"
- „Która myjnia powinna być najtańsza?"

I to ostatnie pytanie okazuje się kluczem do rozwiązania zadania.

Przypuśćmy, że myjnia o numerze x (w skrócie: myjnia x) jest najtańsza i można z niej skorzystać za jedyne s bajtalarów. Daje to nam informację o zachowaniach wszystkich kierowców, których trasa przebiega obok myjni x, czyli takich, dla których $a_i \leqslant x \leqslant b_i$. Jeśli $c_i \geqslant s$, to taki klient umyje samochód w myjni x (lub innej z taką samą ceną), zaś pozostałych nie będzie stać na skorzystanie z usług żadnej myjni. Dla kierowców nieprzejeżdżających obok myjni x odcinek $[a_i, b_i]$ musi być całkowicie zawarty w przedziale $[1, x-1]$ bądź $[x+1, n]$. Sytuacje w tych przedziałach możemy rozpatrywać niezależnie, pamiętając jedynie o założeniu, że ceny we wszystkich

myjniach wynoszą przynajmniej s. Powyższa obserwacja pozwala sprowadzić nasz problem do badania mniejszych podproblemów, czyli do techniki zwanej programowaniem dynamicznym.

Pierwsze podejście

Chcielibyśmy oznaczyć przez $t[a][b][s]$ maksymalny zysk, jaki możemy osiągnąć uwzględniając tylko klientów podróżujących na odcinku od myjni a do myjni b (w skrócie: na odcinku $[a, b]$), przy założeniu, że we wszystkich myjniach na tym odcinku ustalimy ceny nie mniejsze niż s. Jako że potencjalnie będziemy obliczać wszystkie możliwe wartości tablicy t, musimy najpierw uporać się z pewną subtelnością. Zakres cen w zadaniu to $[1, 500\,000]$ i uwzględnienie ich wszystkich prowadziłoby do ogromnej ilości obliczeń. Mało tego, tablica t musiałaby mieć co najmniej $\binom{50}{2} \cdot 500\,000 \geqslant 6 \cdot 10^8$ pól, co przekroczyłoby dozwolony limit pamięci. Na szczęście możemy wykorzystać fakt, że liczba klientów jest ograniczona przez 4000.

Lemat 1. Wśród rozwiązań optymalnych istnieje takie, w którym zbiór cen pojawiających się w myjniach jest zawarty w zbiorze budżetów.

Dowód: Niech B oznacza zbiór budżetów. Na początek zauważmy, że nie ma sensu ustalać cen wyższych niż $\max B$, toteż możemy ustalić pewne rozwiązanie optymalne, w którym ceny zawarte są w przedziale $[1, \max B]$.

Każdą cenę s występującą w tym rozwiązaniu zastąpmy przez $s' = \min\{c \in B : c \geqslant s\}$. Otrzymujemy w ten sposób pewne nowe rozwiązanie. Klient o numerze i wybierał poprzednio najtańszą myjnię z przedziału $[a_i, b_i]$, o ile wystarczał mu budżet c_i bajtalarów. Nowa cena na tej myjni nie przekroczy c_i, zatem i-ty klient nie zrezygnuje z mycia samochodu. Z drugiej strony zapłaci on przynajmniej tyle samo, ponieważ cena na żadnej myjni nie została obniżona. Nowe rozwiązanie spełnia założenia lematu, a ponadto gwarantuje nie mniejszy zysk od wyjściowego, zatem również jest optymalne. ■

Niech (s_j) oznacza posortowany rosnąco ciąg budżetów wszystkich klientów. Możemy teraz poprawić definicję tablicy t.

Definicja 1. Przez $t[a][b][j]$ oznaczamy maksymalny zysk, jaki możemy osiągnąć uwzględniając tylko klientów podróżujących na odcinku $[a, b]$, przy założeniu, że we wszystkich myjniach na tym odcinku ustalimy ceny nie mniejsze niż s_j. Dla wygody przyjmujemy $t[a][b][j] = 0$, jeśli $a > b$.

Wykorzystując obserwację ze wstępu, możemy zaproponować algorytm obliczania wartości $t[a][b][j]$. Na początek rozważmy przypadek, w którym wizyta w najtańszej myjni na odcinku $[a, b]$ kosztuje dokładnie s_j. Przeglądamy wszystkie pozycje $a \leqslant x \leqslant b$, w których może znajdować się ta myjnia, i maksymalizujemy wartość

$$s_j \cdot K(a, x, b, j) + t[a][x-1][j] + t[x+1][b][j], \tag{1}$$

gdzie $K(a, x, b, j)$ to liczba kierowców spełniających następujące warunki:

1. ich trasa jest zawarta w przedziale $[a, b]$ ($a \leqslant a_i$ oraz $b_i \leqslant b$),
2. przejeżdżają obok myjni x ($a_i \leqslant x \leqslant b_i$),
3. są gotowi wydać s_j bajtalarów na mycie samochodu ($c_i \geqslant s_j$).

Jeśli zaś cena w najtańszej myjni jest wyższa niż s_j, to $t[a][b][j] = t[a][b][j+1]$. Jesteśmy gotowi, aby zaprezentować pseudokod pierwszego rozwiązania. Wartości $K(a, x, b, j)$ możemy obliczać, przeglądając każdorazowo wszystkich klientów. Musimy jedynie uważać, aby uzupełniać tablicę t w kolejności rosnących długości przedziałów (*len*) i malejących cen, aby móc odwołać się do wyników wcześniejszych obliczeń. Dla uproszczenia przyjmujemy, że ciąg (s_j) jest długości m (tzn. budżety klientów nie powtarzają się), oraz zakładamy, że tablica t jest wyzerowana. Ze względu na przypisanie w linii 5, aby uniknąć rozpatrywania przypadku brzegowego, wygodnie dodatkowo ustalić $t[a][b][m+1] = 0$.

```
 1: begin
 2:   for len := 1 to n do
 3:     for a := 1 to n − len + 1 do begin
 4:       b := a + len − 1;
 5:       for j := m downto 1 do begin
 6:         t[a][b][j] := t[a][b][j + 1];   { drugi przypadek }
 7:         for x := a to b do begin   { pierwszy przypadek }
 8:           k := 0;
 9:           for i := 1 to m do   { obliczamy k = K(a, x, b, j) }
10:             if a ⩽ a_i ⩽ x and x ⩽ b_i ⩽ b and c_i ⩾ s_j then
11:               k := k + 1;
12:           t[a][b][j] := max(t[a][b][j], s_j · k + t[a][x − 1][j] + t[x + 1][b][j]);
13:         end
14:       end
15: end
```

Pełną implementację można znaleźć w plikach `myjs5.cpp` oraz `myjs6.pas`. Prostota powyższego pseudokodu pozwala na łatwe oszacowanie jego złożoności obliczeniowej: $O(n^3m^2)$. Takie rozwiązanie mogło otrzymać na zawodach 50 punktów.

Usprawnienie

Zastanówmy się teraz, które fazy algorytmu można przyspieszyć. Na początek spróbujemy sprytniejszego podejścia do wyliczania $K(a, x, b, j)$ poprzez usunięcie z funkcji K jej drugiego argumentu. Oznaczmy przez $K(a, b, j)$ liczbę tych klientów, dla których przedział mijanych myjni zawiera się w $[a, b]$ oraz ich budżet jest nie mniejszy niż s_j. Takich klientów można podzielić na trzy grupy: przejeżdżających obok myjni x, podróżujących tylko na lewo od niej oraz tylko na prawo. Zapisanie powyższej obserwacji w postaci równania prowadzi do bardzo pomocnego wzoru:

$$K(a, x, b, j) = K(a, b, j) - K(a, x-1, j) - K(x+1, b, j). \tag{2}$$

Wprowadźmy tablicę k w celu spamiętania wszystkich wartości $K(a, b, j)$. Dzięki temu będziemy mogli obliczać $K(a, x, b, j)$ w czasie stałym, co daje nadzieję na algorytm o złożoności obliczeniowej $O(n^3m)$. Niestety każdorazowe przeglądanie wszystkich klientów w celu obliczenia $k[a][b][j]$ wymaga w sumie $O(n^2m^2)$ operacji, co daje jedynie niewielką poprawę w stosunku do poprzedniego podejścia. Rozwiązania o takiej złożoności ($O(n^3m + n^2m^2)$) mogły liczyć na 80 punktów. Przykłady implementacji można znaleźć w plikach `myjs3.cpp` oraz `myjs4.pas`.

Rozwiązanie wzorcowe

Jedyna rzecz, której nam brakuje, to efektywna metoda wypełniania tablicy k. Aby obliczyć $k[a][b][j]$ wystarczy znać $k[a][b][j+1]$ oraz rozmieszczenie klientów o budżecie s_j. Możemy przeglądać wszystkich klientów i o budżecie s_j i za każdym razem dodawać 1 do $k[a][b][j]$ dla wszystkich par (a, b), takich że $a \leqslant a_i$ oraz $b_i \leqslant b$. W ten sposób wykonamy $O(n^2)$ operacji dla każdego kierowcy oraz stałą liczbę dodatkowych operacji dla każdego pola w tablicy k. Poniżej przedstawiamy pseudokod tej procedury. W dalszym ciągu zakładamy, że ciąg (s_j) jest długości m, tablice są wyzerowane, a ponadto przyjmujemy, że klienci zostali posortowani według niemalejących budżetów.

```
1:  begin
2:    i := m;
3:    for j := m downto 1 do begin
4:      for a := 1 to n do
5:        for b := a to n do
6:          k[a][b][j] := k[a][b][j + 1];
7:      while i > 0 and c_i = s_j do begin
8:        for a := 1 to a_i do
9:          for b := b_i to n do
10:           k[a][b][j] := k[a][b][j] + 1;
11:       i := i − 1;
12:     end
13:   end
14: end
```

Tym razem obliczanie obu tablic k oraz t wymaga $O(n^3m)$ operacji, podczas gdy złożoność pamięciowa algorytmu wynosi $O(n^2m)$. Rozwiązanie wzorcowe zostało zaimplementowane w pliku `myj3.cpp`. W plikach `myj.cpp` oraz `myj1.pas` użyto nieco innej implementacji: zamiast zastanawiać się, w jakiej kolejności przeglądać tablicę t, można wykorzystać rekurencję ze spamiętywaniem i po prostu wyliczać kolejne pola tablicy t, kiedy będą potrzebne. Należy jednak uważać – błąd przy spamiętywaniu wyników może prowadzić do wykładniczego czasu działania.

Odzyskiwanie cen

Pominęliśmy w omówieniu jedną istotną kwestię, jaką jest odtwarzanie cen w myjniach dla optymalnego rozwiązania. W tym celu dla każdego stanu (a, b, j) należy zapamiętać nie tylko maksymalny zysk, ale także miejsce najtańszej myjni na przedziale $[a, b]$. Następnie możemy rekurencyjnie odtworzyć znalezione rozwiązanie optymalne na coraz mniejszych przedziałach. Nietrudno jednak o pomyłkę w implementacji tej pozornie prostej procedury i dlatego uczestnicy, którzy nie zaprogramowali drugiej części zadania poprawnie, mogli dalej liczyć na 60% punktów za każdy test.

Przypomnijmy, że dla każdej trójki (a, b, j) obliczaliśmy maksymalny zysk, jaki można osiągnąć uwzględniając klientów podróżujących wewnątrz przedziału $[a, b]$, przy założeniu, że wszystkie ceny na tym przedziale wynoszą przynajmniej s_j. Niech $opt[a][b][j]$ równa się pozycji myjni $x \in [a, b]$, w której należy ustalić cenę s_j, aby zmaksymalizować zysk. Jeśli optymalnie jest ustalić cenę wyższą od s_j (tj. $t[a][b][j] = t[a][b][j+1]$), to przyjmujemy $opt[a][b][j] = -1$. Opracowanie kończymy pseudokodem funkcji rekurencyjnej, która znajduje optymalny przydział cen, o ile tablica opt została obliczona wcześniej (najwygodniej obliczyć ją jednocześnie z tablicą t).

```
1:  function Odtworz(a, b, j)
2:  begin
3:    if b < a then return;
4:    x := opt[a][b][j];
5:    if x = -1 then
6:      Odtworz(a, b, j + 1);
7:    else begin
8:      cena[x] := s_j;
9:      Odtworz(a, x - 1, j);
10:     Odtworz(x + 1, b, j);
11:   end
12: end
```