Paweł Parys **Karol Cwalina**
Treść zadania, Opracowanie Program

OI, Etap III, dzień 2, 7–04–2005

Prawoskrętny wielbłąd

Bajtocja składa się z N oaz leżących na pustyni, przy czym żadne trzy oazy nie leżą na jednej linii prostej. Bajtazar mieszka w jednej z nich, a w każdej z pozostałych ma po jednym znajomym. Bajtazar chce odwiedzić jak najwięcej swoich znajomych. Zamierza pojechać na swoim wielbłądzie. Wielbłąd ten porusza się niestety w dosyć ograniczony sposób:

- *po wyjściu z oazy wielbłąd porusza się po linii prostej, aż dotrze do innej oazy;*
- *wielbłąd skręca tylko w oazach i tylko w prawo, o kąt z przedziału $[0°; 180°]$ (wielbłąd może tylko raz obrócić się w oazie, tzn. nie jest dozwolony np. obrót o $200°$ w wyniku dwóch kolejnych obrotów o $100°$);*
- *wielbłąd nie chce chodzić po własnych śladach.*

Pomóż Bajtazarowi tak ustalić trasę podróży, aby odwiedził jak najwięcej znajomych. Powinna ona zaczynać i kończyć się w oazie, w której mieszka Bajtazar. Musi składać się z odcinków łączących kolejno odwiedzane oazy. Trasa ta nie może dwa razy przechodzić przez ten sam punkt, z wyjątkiem oazy Bajtazara, gdzie wielbłąd pojawia się dwukrotnie: na początku i na końcu trasy.

Początkowo wielbłąd Bajtazara jest już ustawiony w kierunku konkretnej oazy i musi wyruszyć w tym kierunku. Ustawienie wielbłąda po powrocie z podróży nie ma znaczenia.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia współrzędne oaz oraz ustawienie wielbłąda,*
- *obliczy maksymalną liczbę znajomych, których Bajtazar może odwiedzić zgodnie z powyższymi regułami,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia zapisana jest jedna liczba całkowita N $(3 \leqslant N \leqslant 1\,000)$ — liczba oaz w Bajtocji. Oazy są ponumerowane od 1 do N. Bajtazar mieszka w oazie nr 1, a jego wielbłąd stoi zwrócony w stronę oazy nr 2. W kolejnych N wierszach umieszczone są współrzędne oaz. W $(i+1)$-szym wierszu znajdują się dwie liczby całkowite x_i, y_i — pozioma i pionowa współrzędna i-tej oazy — oddzielone pojedynczym odstępem. Wszystkie współrzędne są z zakresu od $-16\,000$ do $16\,000$.

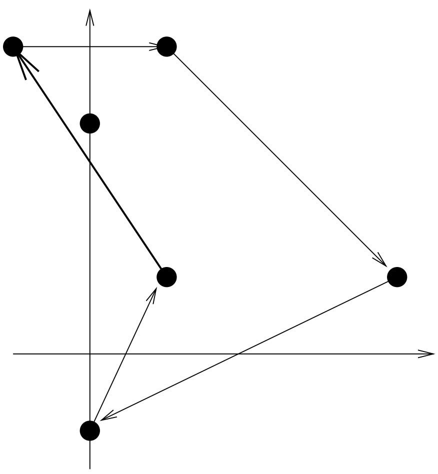
Wyjście

W pierwszym i jedynym wierszu wyjścia Twój program powinien wypisać jedną liczbę całkowitą — największą liczbę znajomych, których może odwiedzić Bajtazar.

Przykład

Dla danych wejściowych:

```
6
1 1
-1 4
0 -1
4 1
0 3
1 4
```



poprawnym wynikiem jest:

```
4
```

Rozwiązanie

Uwagi wstępne

Przedstawimy rozwiązanie zadania oparte na metodzie dynamicznej. Dla każdej pary oaz a,b będziemy liczyć, ilu znajomych może odwiedzić Bajtazar na drodze z początkowej oazy 1 do oazy b, przy założeniu, że do oazy b dotarł bezpośrednio z oazy a.

Zapiszmy oazy w układzie współrzędnych kątowych. Przyjmiemy, że środek układu współrzędnych jest położony w miejscu oazy 1 (od współrzędnych wszystkich oaz odejmujemy współrzędne oazy 1). Następnie ustalimy, że oaza 2 jest pierwsza w porządku kątowym, a pozostałym nadamy numery zgodnie z ruchem wskazówek zegara. Zauważmy, że na każdej poprawnej trasie Bajtazara oazy występują (niekoniecznie wszystkie) zgodnie z opisanym wyżej porządkiem.

Zdefiniujmy tablicę dwuwymiarową, w której będziemy zapisywać obliczone wyniki częściowe. Dla $1 \leqslant a < b \leqslant N$ niech $ile[a,b]$ oznacza największą możliwą liczbę znajomych, których może odwiedzić Bajtazar na drodze z oazy 1 do oazy b, przy założeniu, że do oazy b przybył bezpośrednio z oazy a. Rozwiązaniem zadania jest maksymalna z wartości $\{ile[a,b] : 1 < a < b \leqslant N\}$, ponieważ z każdej oazy b, do której Bajtazar doszedł niebezpośrednio z oazy początkowej, można powrócić do oazy początkowej zgodnie z zasadami podróżowania na prawoskrętnym wielbłądzie.

Obliczanie wartości ile[a,b]

Prawdziwe są następujące zależności:

(i) $ile[1,2] = 1$

(ii) $ile[1,c] = -\infty$ dla $c > 2$

(iii) $ile[b,c] = 1 + \max\{ile[a,b] : 1 \leqslant a < b$ oraz (a,b,c) tworzą zakręt w prawo$\}$

Aby wykorzystać powyższe zależności do obliczenia wartości tablicy *ile*, musimy potrafić szybko określać, czy droga pomiędzy trzema oazami tworzy zakręt w prawo.

Definicja 1 Powiemy, że wektor $VEC(a,b)$ jest *mniejszy* od $VEC(b,c)$ (zapiszemy to $VEC(a,b) \leqslant VEC(b,c)$), gdy droga z a do b i dalej do c skręca w prawo (kąt zorientowany pomiędzy odcinkami *bc* i *ab* ma wartość z przedziału $(0,180)$).

Dla trzech różnych oaz a, b i c zdefiniujemy także warunek $wPrawo(a,b,c)$ oznaczający, że prawoskrętny wielbłąd może z oazy a przejść bezpośrednio do oazy b i dalej (także bezpośrednio) do oazy c, a następnie powrócić (niekoniecznie bezpośrednio) do oazy 1:

$$VEC(a,b) \quad \leqslant \quad VEC(b,c) \tag{1}$$

oraz dla $a > 1$

$$VEC(a,b) \leqslant VEC(b,1) \quad \text{i} \quad VEC(b,c) \leqslant VEC(c,1) \tag{2}$$

Warunek (2) gwarantuje, że wielbłąd nie przecina trasy, po której już przeszedł (żaden z odcinków nie przecina półprostej z 1 do 2, czyli oaza 1 jest cały czas na prawo od trasy).

Aby określić wzajemne położenie dwóch wektorów należy obliczyć ich iloczyn wektorowy. Niech $\mathbf{v} = (x_1, y_1)$ oraz $\mathbf{u} = (x_2, y_2)$ będą dwoma wektorami — ich iloczyn wektorowy wynosi $\mathbf{v} \times \mathbf{u} = x_1 \cdot y_2 - x_2 \cdot y_1$. Jeśli jest on dodatni, to drugi wektor **u** jest skierowany w prawo od pierwszego; jeśli iloczyn jest ujemy, to wektor **u** jest skierowany na lewo od wektora **v**; jeśli iloczyn jest zerem, to wektory są równoległe.

Algorytm

Teraz możemy zapisać algorytm obliczania wartości $ile[a,b]$. Trzy zagnieżdżone pętle sprawiają, że działa on w czasie $O(n^3)$.

```
zainicjalizuj wszystkie ile[a,b]= −∞;
ile[1,2]:=1;
for b:=2 to n do
   for c:=b+1 to n do
      for a:=1 to b−1 do
         if wPrawo(a,b,c) then ile[b,c]:=max(ile[b,c], 1+ile[a,b]);
return max{ile[a,b] : 1 < a < b ⩽ n};
```

Szybszy algorytm

Zauważmy, że w przedstawionej procedurze dla ustalonej oazy b musimy sprawdzić $\Theta(n^2)$ trójek (a,b,c), by znaleźć maksimum. Możemy jednak spróbować usprawnić przeglądanie oaz rozważając oazy c w takiej kolejności, by zbiór dopuszczalnych wartości a wzrastał.

W tym celu dla ustalonego b sortujemy oazy $1,2,\ldots,b-1$ (oazy a w algorytmie) w kolejności współrzędnych kątowych obliczonych względem oazy b — oznaczmy posortowane oazy $a_1,a_2,\ldots,a_{b-1}$. Podobnie sortujemy oazy o numerach $b+1,b+2,\ldots,n$ (oazy c w algorytmie) i podobnie posortowane oazy oznaczamy $c_{b+1},c_{b+2},\ldots,c_n$. Zaważmy, że dla każdej oazy c_i zbiór oaz a, które musimy dla niej sprawdzić, to początkowy fragment ciągu $a_1,a_2,\ldots,a_{b-1}$ i dodatkowo zbiory te rosną wraz z wzrostem indeksu i, czyli mamy następujące zależności.

Dla oazy c_i istnieje liczba $l_i \geqslant 0$ taka, że

$$\{a_j \;:\; wPrawo(a_j,b,c_i)\} \;=\; \{a_j \;:\; 1 \leqslant j \leqslant l_i\}.$$

Dodatkowo dla dowolnej oazy $a \in \{a_1,a_2,\ldots,a_{b-1}\}$ oraz liczb $b < j < k \leqslant n$ zachodzi implikacja

$$\text{jeśli } wPrawo(a,b,c_j) \text{ to } wPrawo(a,b,c_k),$$

stąd także $l_j < l_k$ oraz

$$ile[b,c_j] \leqslant ile[b,c_k].$$

Po zastosowaniu powyższych spostrzeżeń mamy następujący algorytm.

```
zainicjalizuj wszystkie ile[a,b]= −∞;
ile[1,2]:=1;
for b:=2 to n do
   (a_1,...,a_{b−1}):= oazy 1...b−1  posortowane kątowo względem  b;
   (c_{b+1},...,c_n):= oazy b+1...n  posortowane kątowo względem  b;
   a:=a_1;
   for c:=c_{b+1} to c_n do
      while wPrawo(a,b,c) do
         ile[b,c]:=max(ile[b,c], 1+ile[a,b]);
         a:=next(a);
      ile[b,c+1]:=ile[b,c];
return max{ile[a,b] : 1 < a < b ⩽ n};
```

Po wprowadzonych zmianach algorytm dla jednej wartości b działa w czasie $O(n)+czas\_sortowania = O(n\log n)$. Stąd całość działa w czasie $O(n^2\log n)+czas\_sortowania$ (wstępne sortowanie potrzebne do poprawnego, wstępnego ponumerowania oaz) i wymaga $O(n^2)$ pamięci.

Dalsze ulepszenia

Rozwiązanie można jeszcze nieco przyśpieszyć, ograniczając dla każdego b zbiory przeglądanych a i c do takich, że $VEC(a,b) \leqslant VEC(b,1)$ i $VEC(b,c) \leqslant VEC(c,1)$. Pozwala to sortować nieco mniejsze zbiory i jednocześnie upraszcza obliczanie funkcji *wPrawo* sprowadzając je do sprawdzenia warunku $VEC(a,b) \leqslant VEC(b,c)$.

Kolejne ulepszenie jest związane z efektywniejszym wykonywaniem porównań dla kątów. W każdej wersji rozwiązania pojawia się sortowanie punktów według współrzędnych kątowych, a tym samym konieczność wykonywania licznych porównań kątów pomiędzy wektorami, realizowanych poprzez operacje iloczynu skalarnego i wektorowego. Wykonywanie takich porównań siłą rzeczy jest trochę wolniejsze od zwykłego porównywania liczb — możemy je jednak zastąpić porównywaniem liczb. W tym celu dla wektorów zdefiniujemy pewną funkcję rosnącą wraz z kątem sortowania, obliczymy jej wartości dla wszystkich rozważanych wektorów i będziemy sortować wektory według jej wartości.

Przykładem funkcji odpowiedniej dla kąta nachylenia względem wektora $[0,1]$ (tzn. funkcji, którą możemy zastosować, gdy oaza 2 ma współrzędne $(0,1)$) jest:

$$f([x,y]) = \begin{cases} \frac{x}{x+y} & \text{gdy } x>0,\ y>0 \\ 2-\frac{x}{x-y} & \text{gdy } x>0,\ y<0 \\ 2+\frac{x}{x+y} & \text{gdy } x<0,\ y<0 \\ 4-\frac{x}{x-y} & \text{gdy } x<0,\ y>0 \end{cases}$$

Wartości powyższej funkcji są liczbami wymiernymi i ich reprezentacje w komputerze mogą być obarczone błędami zaokrągleń. Aby zmniejszyć błędy możemy wszystkie wartości przemnożyć przez odpowiednio dużą liczbę naturalną i zaokrąglić do wartości całkowitych (za mnożnik przyjmujemy wartość większą niż $(4 \cdot \mathit{MaxWsp\acute{o}\l rz\k{e}dna})^2$).

Przedstawiona modyfikacja przyśpiesza działanie algorytmu około czterokrotnie. Jako program wzorcowy został zaimplementowany powyższy algorytm z wszystkimi usprawnieniami.

Uwagi końcowe

Zadanie jest dość trudne. Co prawda nie są w rozwiązaniu potrzebne żadne skomplikowane algorytmy, ale nagromadzenie prostych algorytmów i różnych przypadków jest na tyle duże, że kompletna implementacja nie jest sprawą prostą.

Testy

Zadanie było testowane na 11 danych testowych.

Nazwa	n	wynik	Opis
pra1.in	11	6	mały prosty test
pra2.in	15	6	mały test poprawnościowy
pra3.in	19	7	mały test poprawnościowy
pra4.in	40	9	losowy test poprawnościowy
pra5.in	106	52	test ze względnie dużą odpowiedzią
pra6.in	200	31	test losowy
pra7.in	500	43	test losowy
pra8.in	500	23	test losowy
pra9.in	743	316	test ze stosunkowo dużą odpowiedzią
pra10.in	997	31	test losowy
pra11.in	1 000	61	test losowy