Jakub Wojtaszczyk
Treść zadania, Opracowanie

Paweł Parys
Program

Dostępna pamięć: 32 MB.

OI, Etap III, dzień pierwszy, 01.04.2009

Wyspa

Bajtazar jest królem Bajtocji, wyspy na Oceanie Szczęśliwości. Bajtocja ma kształt figury wypukłej, a wszystkie miasta w niej są położone nad brzegiem oceanu. Jedno z tych miast to Bajtogród, stolica Bajtocji. Każde dwa miasta są połączone drogą biegnącą przez wyspę po prostej łączącej oba miasta. Drogi łączące różne pary miast przecinają się, tworząc skrzyżowania.

Bitocy, konkurent Bajtazara do tronu, zaplanował nikczemny spisek. W trakcie podróży Bajtazara ze stolicy do **sąsiedniego** *miasta opanował Bajtogród. Bajtazar musi jak najszybciej powrócić do Bajtogrodu, by odzyskać władzę. Niestety niektóre z dróg są patrolowane przez zbrojne bojówki Bitocego. Bajtazar nie może poruszać się takimi drogami, choć może przekraczać je na skrzyżowaniach. Na trasie może on skręcać wyłącznie na skrzyżowaniach.*

Bajtazar wie, które drogi są patrolowane przez bojówki Bitocego. Poprosił Cię o wyznaczenie najkrótszej bezpiecznej trasy z miasta, w którym się aktualnie znajduje, do Bajtogrodu.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n i m, oddzielone pojedynczym odstępem ($3 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 1\,000\,000$), oznaczające odpowiednio: liczbę miast oraz liczbę dróg patrolowanych przez bojówki Bitocego. Ponumerujmy miasta od 1 do n, zaczynając od Bajtogrodu i idąc wzdłuż brzegu zgodnie z ruchem wskazówek zegara. Bajtazar znajduje się w mieście nr n. W każdym z kolejnych n wierszy znajduje się para liczb x_i i y_i ($-1\,000\,000 \leqslant x_i, y_i \leqslant 1\,000\,000$), oddzielonych pojedynczym odstępem, oznaczających współrzędne miasta nr i na wyspie.

W każdym z kolejnych m wierszy znajduje się para liczb a_j i b_j ($1 \leqslant a_j < b_j \leqslant n$). Para taka oznacza, że droga łącząca miasta a_j i b_j jest patrolowana przez bojówki Bitocego. Pary liczb opisujące patrolowane drogi nie powtarzają się. Możesz założyć, że dla każdych danych testowych Bajtazar może dostać się do Bajtogrodu.

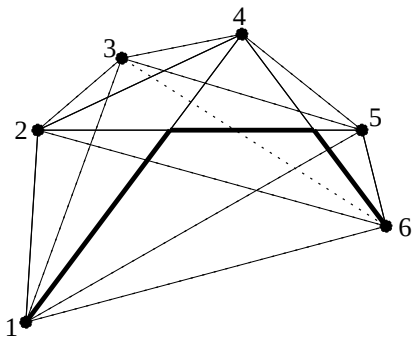
Wyjście

Twój program powinien wypisać na standardowe wyjście jedną liczbę zmiennopozycyjną: długość najkrótszej bezpiecznej trasy prowadzącej z miasta n do Bajtogrodu. Wynik Twojego programu może różnić się od poprawnego o co najwyżej 10^{-5}.

Przykład

Dla danych wejściowych:

```
6 9
-12 -10
-11 6
-4 12
6 14
16 6
18 -2
3 4
1 5
2 6
2 3
4 5
3 5
1 3
3 6
1 6
```



poprawnym wynikiem jest:

```
42.0000000000
```

Trasa, którą powinien podążać Bajtazar, wychodzi z miasta 6 w kierunku miasta 4, następnie biegnie drogą łączącą miasta 2 i 5, a na koniec drogą łączącą Bajtogród z miastem 4.

Rozwiązanie

Wprowadzenie

Przez M_i będziemy oznaczać miasto o numerze i. Jako że wszystkie drogi są dwukierunkowe, znalezienie najkrótszej trasy z M_n do Bajtogrodu (czyli M_1) jest równoważne znalezieniu najkrótszej drogi z M_1 do M_n. Drogę, na której nie ma bojówek Bitocego (czyli taką, którą może poruszać się Bajtazar), będziemy nazywać *przejezdną*.

Zaczniemy od zrozumienia, jak wygląda szukana najkrótsza trasa.

Lemat 1. Najkrótsza bezpieczna trasa z M_1 do M_n wraz z odcinkiem $[M_n, M_1]$ stanowi brzeg wielokąta wypukłego (być może zdegenerowanego do odcinka, jeśli droga $(1,n)$ jest przejezdna), którego wnętrza nie przecina żadna przejezdna droga.

Dowód: Najkrótsza bezpieczna trasa oczywiście nie może mieć samoprzecięć, a zatem wraz z odcinkiem $[M_1, M_n]$ ogranicza pewien wielokąt prosty. Żadna droga nie przecina odcinka $[M_1, M_n]$ poza końcami (bo pomiędzy M_1 a M_n na brzegu wyspy nie ma miast), zatem jeśli jakakolwiek przejezdna droga D przecina wnętrze tego wielokąta, to przynajmniej dwukrotnie przecina najkrótszą trasę. Wobec tego, zastępując odcinek trasy pomiędzy pierwszym i ostatnim punktem przecięcia odpowiednim fragmentem drogi D, otrzymamy drogę krótszą. To pokazuje, że żadna przejezdna droga nie przecina naszego wielokąta. Gdyby ten wielokąt nie był wypukły, to wierzchołek z kątem wklęsłym znajdowałby się na

pewnym skrzyżowaniu dróg (bo wszystkie drogi wychodzące z jednego miasta są zawarte w kącie wypukłym). Ale wtedy pozostałe fragmenty dwóch dróg krzyżujących się na tym skrzyżowaniu znajdowałyby się wewnątrz wielokąta, a więc przecinałyby go — wbrew temu, co już udowodniliśmy. ■

Lemat 1 charakteryzuje najkrótszą trasę w sposób jednoznaczny, bowiem dla dwóch różnych wielokątów wypukłych, które mają wspólny odcinek brzegowy i leżą po tej samej stronie prostej zawierającej ten odcinek, jeden z nich musi zawierać we wnętrzu fragment brzegu drugiego (rysunkowe uzasadnienie tego stwierdzenia pozostawiamy Czytelnikowi).

Rozwiązania nieoptymalne

Zacznijmy od rozważenia najprostszego rozwiązania. W zadaniu występuje problem znalezienia najkrótszej ścieżki między dwoma wierzchołkami w grafie, co przywodzi na myśl algorytm Dijkstry (patrz np. [20]). Zastanówmy się jednak, jaki graf musielibyśmy rozważyć. Otóż wierzchołkami naszego grafu musiałyby być nie tylko miasta, ale także skrzyżowania dróg (bo tam również możemy skręcać). Jednak skrzyżowań przekątnych w wielokącie jest $O(n^4)$ (każda czwórka wierzchołków wielokąta wyznacza jedno przecięcie przekątnych), czyli stanowczo za dużo, by stosować taki algorytm — niezależnie od złożoności czasowej, brak nam pamięci komputera, by choćby przechowywać wierzchołki takiego grafu.

W następnych podejściach spróbujemy skorzystać z udowodnionego wyżej lematu. Jeżeli droga $(1,n)$ jest przejezdna, to oczywiście jest najkrótszą trasą pomiędzy miastami 1 i n; dalej będziemy zakładać, że Bitocy (całkiem rozsądnie) postanowił jednak patrolować tę drogę. Rozważmy dowolną inną przejezdną drogę D. Prosta k zawierająca D nie przecina wnętrza odcinka (M_1,M_n), bo pomiędzy M_1 a M_n nie leży żadne miasto. Zatem miasta M_1 i M_n leżą po tej samej stronie k i przynajmniej jedno z nich nie leży na k (bo D jest różna od $(1,n)$). Zauważmy, że z lematu wynika, iż optymalna trasa cała przebiega przez tę półpłaszczyznę ograniczoną przez k, w której leżą miasta M_1 i M_n — optymalna trasa nie może przecinać (tzn. przechodzić na drugą stronę) odcinka D na mocy lematu oraz nie może przecinać k poza D, gdyż nie może wyjść poza wyspę. Z drugiej strony, jeżeli rozważymy przecięcie wszystkich półpłaszczyzn wyznaczonych przez wszystkie przejezdne drogi D, to otrzymamy wielokąt wypukły (bo przecinamy zbiory wypukłe), którego wnętrza nie przecina żadna przejezdna droga (bo każda przejezdna droga leży w brzegu pewnej półpłaszczyzny), a zatem jego brzeg będzie szukaną optymalną trasą.

Zadanie znalezienia obszaru będącego przecięciem zbioru zadanych półpłaszczyzn jest bardzo zbliżone do problemu znalezienia otoczki wypukłej zbioru punktów (o tym problemie można poczytać w większości książek o algorytmach, w szczególności w [20]). Jeśli przez q oznaczymy liczbę półpłaszczyzn, to algorytm rozwiązujący ten problem (opisany dokładniej dalej) działa w złożoności czasowej $O(q \log q)$ — czyli w naszym wypadku $O(n^2 \log n)$, tzn. dużo lepszej, ale ciągle niesatysfakcjonującej. Aby poprawić złożoność, będziemy musieli podjąć wysiłek rozważania tylko części dróg i odrzucenia tych, o których umiemy zagwarantować, że nie przebiega nimi optymalna trasa.

Rozwiązanie wzorcowe

Stoi przed nami zadanie efektywnego wybrania części przejezdnych dróg tak, by nie rozpatrywać ich wszystkich. Zauważmy jednak, że jeśli dla ustalonej przejezdnej drogi (a,b), przy czym $a < b$, istnieje taka przejezdna droga (a,b'), że $b < b'$, to dowolna trasa idąca kawałek drogą (a,b) przecina drogę (a,b'), a zatem na mocy lematu nie może być najkrótsza. Wystarczy zatem rozważać takie przejezdne drogi (a,b), że dla dowolnego $b' > b$ droga (a,b') jest patrolowana. Drogi o tej własności nazwiemy *potrzebnymi*. Oczywiście z każdego miasta wychodzi co najwyżej jedna droga potrzebna, a zatem dróg potrzebnych jest w sumie $O(n)$.

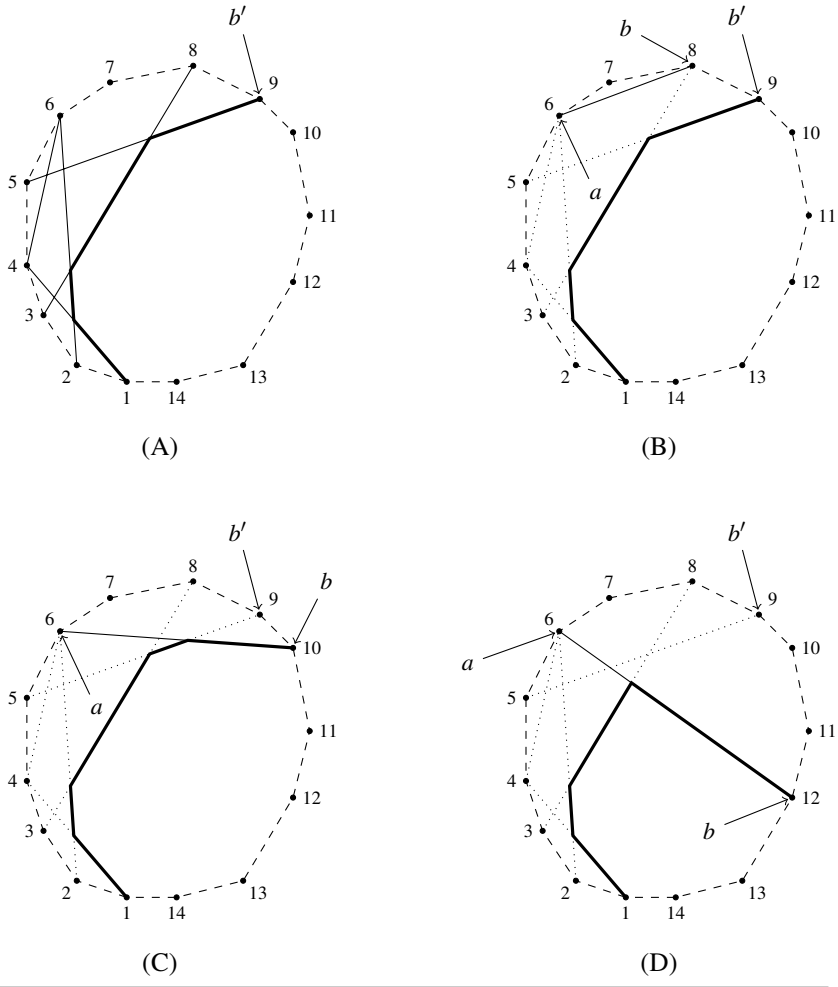
Zastanówmy się nad efektywną implementacją wyszukiwania dróg potrzebnych. Musimy stworzyć tablicę, w której dla każdego miasta a będzie podany numer miasta b, do którego prowadzi potrzebna droga z a (a jeżeli z a nie prowadzi żadna przejezdna droga, to droga potrzebna z a nie istnieje). Chcemy dla każdego miasta a przeglądać wszystkie drogi (a,b) w kolejności od największego b i znaleźć pierwszą, która jest przejezdna. Wystarczy więc wrzucić wszystkie drogi patrolowane wychodzące z miasta a do miast o większym numerze do tablicy, posortować w porządku malejącym numerów miast docelowych i przeglądać w poszukiwaniu pierwszej drogi, której brakuje. Wszystkich dróg patrolowanych jest co najwyżej m, zatem łączny czas wyszukiwania dróg potrzebnych to $O((m+n)\log n)$.

Należy tu zwrócić uwagę na sposób przechowywania dróg patrolowanych. Jako że z każdego konkretnego miasta może wychodzić $O(n)$ dróg patrolowanych, to tablica rozmiaru $O(n^2)$ nie zmieści się w pamięci. Można użyć list (które się trudniej sortuje) albo, w Javie i w C++, tablic dynamicznych o zmiennym rozmiarze (vectorów). Można też po prostu wszystkie drogi patrolowane wczytać do jednej tablicy rozmiaru $O(m)$, ewentualnie zamieniając końce tak, by numer miasta startowego był mniejszy niż numer docelowego, a następnie posortować tę tablicę leksykograficznie — w kolejności rosnącej numerów miast startowych, rozstrzygając remisy na korzyść większych numerów miast docelowych. Wtedy przeglądając tę tablicę, wyznaczymy drogi potrzebne dla wszystkich miast po kolei.

Mając tablicę dróg potrzebnych, będziemy konstruować trasę, której nie przecinają żadne inne drogi. Opiszemy tu precyzyjnie stosowany algorytm. Będziemy rozważać drogi potrzebne w kolejności wyznaczonej przez ich miasta początkowe: najpierw zaczynająca się w mieście 1, potem w 2, itd. Tuż po przetworzeniu drogi (a,b) chcemy zawsze mieć obliczoną trasę z miasta 1 do pewnego miasta $b' \geqslant b$, której nie przecina żadna z dotychczas przetworzonych potrzebnych dróg i która jest wypukła (formalnie: po dołożeniu drogi $(b',1)$ stanowi brzeg wielokąta wypukłego). Przynajmniej jedna potrzebna droga musi kończyć się w mieście n (w przeciwnym wypadku nie da się w ogóle dojść do miasta n), zatem po przetworzeniu wszystkich potrzebnych dróg dostajemy rozwiązanie. Pierwszą potrzebną drogę bardzo łatwo przetworzyć, musi to być droga $(1,b)$ (taka musi istnieć, bo jeśli nie istnieje droga potrzebna z miasta 1, to wszystkie drogi z niego są patrolowane i wbrew założeniom zadania nie istnieje trasa, którą może przejść Bajtazar), więc na początku trasa składa się tylko z niej.

Przypuśćmy, że mamy skonstruowaną trasę do miasta b' i chcemy przetworzyć kolejną potrzebną drogę (a,b). Wiemy, że $a > a'$, przy czym (a',b') jest drogą, której fragment został wykorzystany, by dojść do b'. Jeśli $b \leqslant b'$, to drogę (a,b) możemy odrzucić (gdyż nie przecina drogi (a',b'), a zatem z wypukłości nie przecina też dotychczas skonstruowanej trasy). W przeciwnym razie szukamy punktu jej przecięcia C z dotychczasową trasą. Do

punktu przecięcia C idziemy dotychczasową trasą, a dalej drogą (a,b). Otrzymany wielokąt jest wypukły (jest przecięciem figur wypukłych) i nie przecina go żadna z dotychczas skonstruowanych dróg (wykorzystany fragment drogi (a,b) przebiega wewnątrz starego wielokąta, a więc żadna z uprzednio przejrzanych dróg go nie przecina). Punkt przecięcia efektywnie znajdujemy poprzez umieszczenie fragmentów dróg, z których składa się dotychczasowa trasa, na stosie. Gdy przetwarzamy nową drogę (a,b), zdejmujemy od wierzchołka stosu odcinki do momentu trafienia na taki odcinek $[P,Q]$, który przecina się z $[M_a,M_b]$ w pewnym punkcie C, a następnie umieszczamy na stosie odcinki $[P,C]$ i $[C,M_b]$.



Rys. 1: Kilka początkowych kroków algorytmu konstrukcji trasy Bajtazara. (A) Ciągłe kreski reprezentują drogi potrzebne wychodzące z miast od 1 do 5, natomiast skonstruowany fragment trasy, prowadzący do miasta $b' = 9$, jest oznaczony pogrubioną linią. (B) Jeżeli droga potrzebna z miasta $a = 6$ prowadzi do miasta $b = 8$, to trasa nie ulega zmianie (jako że $b \leqslant b'$). (C) Jeżeli drogą potrzebną jest (6,10), to przecina ona dotychczasową trasę w jej ostatnim odcinku. (D) Jeżeli natomiast jest nią (6,12), to przecięcie następuje na przedostatnim odcinku dotychczasowej trasy, więc ostatni odcinek trasy zostaje zdjęty ze stosu.

Żeby sprawdzić, czy dane dwa odcinki $[P,Q]$ i $[M_a,M_b]$ przecinają się, a jeżeli tak, wyznaczyć ich punkt przecięcia, można np. zapisać je w postaci parametrycznej:

$$\{(1-s)\cdot P+s\cdot Q:\ s\in[0,1]\}\quad\text{oraz}\quad\{(1-t)\cdot M_a+t\cdot M_b:\ t\in[0,1]\}$$

i rozwiązać układ dwóch równań liniowych (osobno po współrzędnej x i y) z niewiadomymi s i t:

$$(1-s)\cdot P+s\cdot Q=(1-t)\cdot M_a+t\cdot M_b.$$

Przy przetwarzaniu pojedynczej drogi włożymy na stos co najwyżej jeden nowy odcinek, więc w czasie działania całego algorytmu włożymy $O(n)$ odcinków, a zatem i zdejmiemy najwyżej $O(n)$ odcinków. Zatem ta część algorytmu działa w czasie $O(n)$. Rozwiązanie takie zostało zaimplementowane w plikach `wys.cpp`, `wys1.c`, `wys5.pas`, `wys6.java`, `wys7.ml` (działające na liczbach zmiennoprzecinkowych) oraz w pliku `wys3.cpp` (implementacja używająca wyłącznie liczb całkowitych, z własną arytmetyką ułamków).

Rozwiązanie alternatywne

Rozwiązanie to opiera się na spostrzeżeniu, że możemy jeszcze bardziej ograniczyć liczbę rozważanych dróg. Zauważmy, że jeśli potrzebna droga (a,b) ma zostać wykorzystana w trasie docelowej, to w szczególności wszystkie drogi (a',b) dla $a'<a$ muszą być patrolowane — czyli wykorzystamy co najwyżej jedną drogę wiodącą do danego b (tę z miasta a o najmniejszym numerze). Możemy zatem, konstruując drogi potrzebne z każdego a, trzymać dodatkowo tablicę indeksowaną miastami docelowymi, wskazującą, czy już jakaś potrzebna droga prowadzi do tego miasta. Jeżeli tak, to droga, którą właśnie skonstruowaliśmy, na pewno nie będzie nigdy wykorzystana (bo przeglądamy miasta wyjściowe w kolejności rosnącej), czyli nie trzeba jej dodawać.

Zauważmy, że żeby została skonstruowana droga potrzebna z miasta a do miasta b, na wejściu muszą być podane (patrolowane) drogi (a,b') dla wszystkich $b'>b$. Co więcej, każda skonstruowana droga potrzebna prowadzi do innego miasta b. Wobec tego, aby skonstruować k dróg potrzebnych, na wejściu musi znaleźć się co najmniej $0+1+\ldots+(k-1)=k\cdot(k-1)/2=\Omega(k^2)$ dróg. Wobec tego skonstruujemy co najwyżej $O(\sqrt{m})$ dróg potrzebnych.

W takim razie przy konstrukcji najkrótszej trasy nie musimy działać sprytnie, wystarczy stworzyć ją w złożoności czasowej $O(k^2)$. Dla każdej pary skonstruowanych dróg wyznaczamy ich punkt przecięcia (o ile się przecinają). Następnie dla aktualnej drogi wybieramy punkt przecięcia z tą z kolejnych skonstruowanych dróg, który znajduje się najbliżej jej początku, i w tę drogę potrzebną skręcamy. Rozwiązanie to zostało zaimplementowane w pliku `wys2.cpp`.

Testy

Poprawność rozwiązań badało 14 testów połączonych w 10 zestawów.

Pierwsze pięć testów to testy pseudolosowe. Zawierają stosunkowo małe wielokąty (tak by Bitocy mógł patrolować nawet wszystkie drogi, mieszcząc się w górnym limicie). W tych testach drogi potrzebne są stosunkowo krótkie, a wynikowa trasa zawiera stosunkowo wiele

dróg. W tym celu jako przejezdne wybrane zostały losowe krótkie drogi, wszystkie długie drogi są patrolowane.

Druga grupa testów, również pseudolosowych, zawiera duże wielokąty, w których patrolowane są losowe drogi wychodzące z miast o stosunkowo małych numerach i prowadzące do miast o stosunkowo dużych numerach (ze względu na ograniczenia z zadania, siłą rzeczy, procent dróg patrolowanych musi tu być niewielki).

W zestawie znajdują się dwa testy w pełni losowe (tzn. takie, w których każda droga jest patrolowana z równym prawdopodobieństwem), co dla dużych wielokątów powoduje, że najkrótsza trasa jest zdecydowanie nieskomplikowana.

Do tego jest jeden test badający dokładność obliczeń, w którym występują dwie drogi prawie równoległe (a więc stosunkowo niewielki kątowo błąd w wyznaczaniu punktu przecięcia istotnie zmienia długość trasy), oraz test skrajny, w którym dozwolone jest poruszanie się wyłącznie po brzegu wielokąta.

Poniższa tabelka zawiera statystyki poszczególnych testów, w których k to liczba dróg potrzebnych skonstruowanych przez algorytm alternatywny, a l to liczba dróg na najkrótszej trasie.

Nazwa	n	m	k	l	Opis
wys1.in	100	4 772	41	26	mały test pseudolosowy
wys2.in	200	18 472	120	49	mały test pseudolosowy
wys3.in	500	120 793	289	99	mały test pseudolosowy
wys4.in	1 000	494 995	920	518	mały test pseudolosowy
wys5.in	1 412	982 600	420	108	mały test pseudolosowy
wys6.in	10 000	1 000 000	1 185	26	duży test pseudolosowy
wys7a.in	20 000	1 000 000	1 299	23	duży test pseudolosowy
wys7b.in	4	4	2	2	test dokładnościowy
wys8a.in	30 000	1 000 000	1 287	22	duży test pseudolosowy
wys8b.in	1 416	1 000 000	6	4	test w pełni losowy
wys9a.in	40 000	1 000 000	1 235	14	duży test pseudolosowy
wys9b.in	55 944	1 000 000	1	1	test w pełni losowy
wys10a.in	50 000	1 000 000	584	2	duży test pseudolosowy
wys10b.in	1 412	994 755	1 411	1 411	test skrajny