Alan Kutniewski
Treść zadania, Opracowanie

Zbigniew Wojna
Program

Dostępna pamięć: 128 MB.

OI, etap I, 17.10–14.11.2011

Randka

Bajtazar jest strażnikiem przyrody i pracuje w Jaskini Strzałkowej — znanym miejscu schadzek zakochanych par. Jaskinia ta składa się z n komór połączonych jednokierunkowymi korytarzami. W każdej komorze dokładnie jeden wychodzący z niej korytarz jest oznaczony strzałką. Każdy korytarz prowadzi bezpośrednio z jednej komory do pewnej (niekoniecznie innej) komory.

Notorycznie zdarza się, że zakochane pary, które umawiają się na randki w Jaskini Strzałkowej, zapominają dokładnie ustalić miejsce spotkania i nie mogą się odnaleźć. W przeszłości prowadziło to do wielu nieporozumień i pomyłek... Od czasu, gdy w każdej komorze zainstalowano telefon alarmowy łączący z dyżurnym strażnikiem przyrody, głównym zajęciem strażników stało się pomaganie zakochanym parom w odnajdowaniu się.

Strażnicy wypracowali następującą metodę. Wiedząc, w których komorach znajdują się zakochani, mówią każdemu z nich, ile razy, odpowiednio, powinien przejść z komory do komory korytarzem oznaczonym strzałką, aby oboje mogli się spotkać. Przy tym, zakochani bardzo chcą spotkać się jak najszybciej — zależy im przecież, aby razem miło spędzać czas, a nie samotnie przemierzać korytarze. Strażnicy starają się podawać zakochanym parom takie liczby, aby ich maksima były możliwie jak najmniejsze.

Bajtazar jest już zmęczony ciągłym pomaganiem zakochanym i poprosił Cię o napisanie programu, który by to usprawnił. Program ten, na podstawie opisu Jaskini Strzałkowej oraz aktualnego położenia k zakochanych par, powinien wyznaczyć k par liczb x_i i y_i, takich że:

- *jeżeli i-ta para zakochanych przejdzie odpowiednio: on x_i, a ona y_i korytarzami oznaczonymi strzałkami, to spotkają się w jednej komorze jaskini,*
- *$max(x_i, y_i)$ jest jak najmniejsze,*
- *w drugiej kolejności $min(x_i, y_i)$ jest jak najmniejsze,*
- *jeżeli rozwiązanie wciąż nie jest jednoznaczne, to kobieta powinna pokonywać mniejszy dystans.*

Może się tak zdarzyć, że takie liczby x_i i y_i nie istnieją — wówczas przyjmujemy, że $x_i = y_i = -1$.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie dodatnie liczby całkowite n i k ($1 \leqslant n \leqslant 500\,000$, $1 \leqslant k \leqslant 500\,000$), oddzielone pojedynczym odstępem i określające liczbę komór w Jaskini Strzałkowej i liczbę zakochanych par, które chcą się odnaleźć. Komory są ponumerowane od 1 do n, natomiast zakochane pary są ponumerowane od 1 do k.

W drugim wierszu wejścia znajduje się n liczb całkowitych dodatnich, pooddzielanych pojedynczymi odstępami: i-ta liczba w tym wierszu określa numer komory, do której prowadzi korytarz oznaczony strzałką wychodzący z komory numer i.

W kolejnych k wierszach znajdują się kolejne zapytania zakochanych par. Każde zapytanie składa się z dwóch liczb całkowitych dodatnich oddzielonych pojedynczym odstępem — oznaczają one numery komór, w których znajduje się dana para zakochanych — najpierw on, a potem ona.

W testach wartych łącznie 40% punktów zachodzą dodatkowe warunki $n \leqslant 2\,000$ *oraz* $k \leqslant 2\,000$*.*

Wyjście

Twój program powinien wypisać na standardowe wyjście dokładnie k wierszy, po jednym dla każdej pary zakochanych z wejścia: i-ty wiersz wyjścia powinien opisywać wynik dla i-tej pary z wejścia. Każdy wiersz wyjścia powinien składać się z dwóch liczb całkowitych x_i, y_i, *oddzielonych pojedynczym odstępem.*

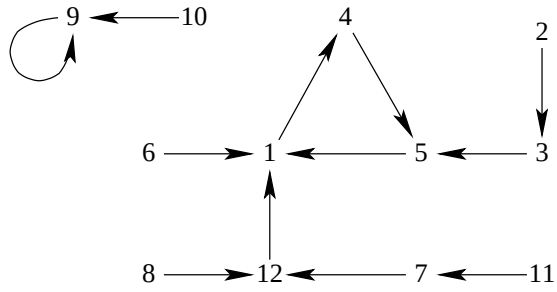
Przykład

Dla danych wejściowych:

```
12 5
4 3 5 5 1 1 12 12 9 9 7 1
7 2
8 11
1 2
9 10
10 5
```

poprawnym wynikiem jest:

```
2 3
1 2
2 2
0 1
-1 -1
```



Rozwiązanie

Analiza problemu

Na początku zastanówmy się, jaką postać ma graf opisany w tym zadaniu. Jest to graf skierowany, w którym z każdego wierzchołka wychodzi dokładnie jedna krawędź. Nasz graf składa się zatem z pewnej liczby cykli, do których mogą być dołączone drzewa[1].

Powiemy, że dwa wierzchołki należą do tej samej *słabo spójnej składowej* grafu, jeśli można przejść z jednego z nich do drugiego po strzałkach, ignorując ich skierowanie (tzn. możemy iść w przód lub w tył). W każdej składowej znajduje się dokładnie

[1]Ciekawe własności takich grafów były niejednokrotnie wykorzystywane w rozwiązaniach zadań olimpijskich, np. w zadaniu *Mafia* z XV Olimpiady Informatycznej [15] i w zadaniu *Szpiedzy* z XI Olimpiady Informatycznej [11].

jeden cykl. W dalszym opisie przyjmiemy, że każdy wierzchołek cyklu stanowi korzeń pewnego drzewa dołączonego do cyklu — w niektórych przypadkach drzewo to składa się tylko z tego wierzchołka.

Następnie mamy daną listę par wierzchołków w grafie. Dla każdej pary wierzchołków u, v musimy znaleźć taką *optymalną* parę liczb x, y, że po przejściu x kroków z wierzchołka u i y kroków z wierzchołka v dojdziemy do tego samego wierzchołka. Szukana optymalna para x, y powinna przede wszystkim minimalizować $\max(x, y)$, w drugiej kolejności — $\min(x, y)$, a w trzeciej kolejności — liczbę y.

Pomyślmy teraz, gdzie może leżeć wierzchołek, w którym nastąpi spotkanie. Możliwe są trzy przypadki, w zależności od wzajemnego położenia u i v:

1. Wierzchołki leżą na jednym drzewie i na nim też nastąpi spotkanie.
2. Wierzchołki leżą na różnych drzewach, których korzenie znajdują się na jednym cyklu. Wówczas spotkanie nastąpi na tym cyklu.
3. Wierzchołki leżą w różnych składowych grafu, zatem spotkanie jest niemożliwe.

Zauważmy, że łatwo stwierdzić, czy mamy do czynienia z przypadkiem 3, jeśli dla każdego wierzchołka znamy numer składowej grafu, w której się ten wierzchołek znajduje. Takie numery dla wszystkich wierzchołków można wyznaczyć w czasie $O(n)$, przeszukując graf i ignorując skierowanie krawędzi (opisujemy to także w sekcji „Rozwiązanie wzorcowe"). W dalszej części opisu będziemy zatem zakładać, że mamy do czynienia z przypadkiem 1 lub 2.

Rozwiązanie siłowe

Na podstawie powyższych obserwacji możemy skonstruować pierwsze rozwiązanie. Zaznaczamy wszystkie wierzchołki, do których można dojść z wierzchołka u. Następnie, startując z wierzchołka v, idziemy po strzałkach, aż dojdziemy do jednego z wierzchołków, które zaznaczyliśmy. Później robimy to samo, odwracając role. Jako wynik podajemy lepsze z dwóch znalezionych miejsc spotkania (używając zadanego kryterium porównywania wyników).

Dlaczego otrzymany w ten sposób wynik jest optymalny? Wróćmy do określonych wcześniej przypadków.

W pierwszym przypadku, wykonując powyższy algorytm, znajdziemy najbliższego wspólnego przodka wierzchołków u i v w drzewie. Jedynymi innymi potencjalnymi miejscami spotkań są wierzchołki bliższe korzenia lub wierzchołki leżące na cyklu. Jednak żeby do nich dojść, trzeba zwiększyć zarówno x, jak i y, co z oczywistych względów nie da nam lepszego wyniku.

Drugi przypadek jest trochę bardziej skomplikowany. Potencjalne miejsca spotkań leżą na cyklu. Jeśli więc któryś wierzchołek z pary nie leży na cyklu, to musimy przejść z niego w górę drzewa, aż dojdziemy do cyklu. Kiedy już oba wierzchołki znajdą się na cyklu, mamy dwie możliwości: albo przejdziemy po cyklu z pierwszego wierzchołka do drugiego, albo na odwrót. Tym dwóm przypadkom odpowiadają dwa przebiegi naszego algorytmu. Zatem bierzemy pod uwagę dwa potencjalne miejsca spotkań: pierwszy wierzchołek leżący na cyklu należący do ścieżki wychodzącej z u oraz analogiczny wierzchołek należący do ścieżki wychodzącej z v. Każdy inny wierzchołek

cyklu może być tylko gorszym kandydatem na miejsce spotkania, ponieważ ścieżki z u oraz z v do dowolnego niewybranego przez nas wierzchołka na cyklu prowadzą, odpowiednio, przez wybrane przez nas miejsca.

Złożoność czasowa tego rozwiązania to $O(n \cdot k)$. Przykładowe implementacje można znaleźć plikach `rans1.cpp` i `rans2.pas`.

Wnioski z rozwiązania siłowego

Konstruując rozwiązanie siłowe, poczyniliśmy bardzo ważne spostrzeżenia.

1. Jeśli wierzchołki u, v leżą na jednym drzewie, to wynikiem jest ich najniższy wspólny przodek (tzw. *LCA*, ang. *lowest common ancestor*).

2. W przeciwnym razie jedynymi kandydatami na optymalne miejsce spotkania są pierwsze wierzchołki na ścieżkach wychodzących z u i v, które leżą na cyklu.

Rozwiązanie wzorcowe

Optymalne miejsca spotkań zależą od przypadku, z którym mamy do czynienia. Musimy więc umieć szybko go określać. Dla każdego wierzchołka chcemy znaleźć cykl, do którego można z niego dojść, oraz zidentyfikować drzewo, w którym się znajduje. Aby wyznaczyć te wartości dla wszystkich wierzchołków, będziemy w grafie naprzemiennie wyszukiwali cykle i dołączone do nich drzewa.

W pierwszym kroku chcemy znaleźć jakiś cykl. W tym celu wybieramy dowolny wierzchołek w i idziemy po strzałkach. Musimy w końcu dojść do odwiedzonego wcześniej wierzchołka. To będzie oznaczało, że znaleźliśmy cykl; teraz wystarczy przejść po nim, zaznaczając go. Można to zrobić przy pomocy algorytmu przedstawionego poniżej (początkowo dla każdego wierzchołka i mamy $odwiedzony[i] =$ **false**, $cykl[i] = -1$).

```
1:  while not odwiedzony[w] do begin
2:      odwiedzony[w] := true;
3:      w := następny[w];
4:  end
5:  { W tym miejscu w leży na cyklu }
6:  ostatni := w;
7:  do
8:      cykl[w] := nr_cyklu;
9:      w := następny[w];
10: while w ≠ ostatni;
```

W drugim kroku zaznaczamy drzewa, które wchodzą do cyklu znalezionego w pierwszym kroku. Przechodzimy po wszystkich wierzchołkach należących do cyklu. Z każdego z nich idziemy w przeciwną stronę do tej, którą pokazują strzałki, do wierzchołków, które nie mają jeszcze przypisanego cyklu. Innymi słowy, przeszukujemy wszerz (algorytm BFS) graf transponowany. Każdemu wierzchołkowi zapisujemy

numer wierzchołka cyklu, z którego przyszliśmy do tego wierzchołka, jako numer jego drzewa. Wierzchołki leżące na cyklu są korzeniami tych drzew.

Po wykonaniu tych dwóch kroków mamy oznaczony jeden cykl oraz wszystkie jego drzewa. Musimy powtarzać ten algorytm, dopóki istnieją nieoznaczone wierzchołki, ignorując wierzchołki, które już zostały oznaczone.

Wszystkie wstępne obliczenia działają w złożoności $O(n)$. Obliczone za ich pomocą informacje pozwalają łatwo określać, dla każdego zapytania, z którym przypadkiem mamy do czynienia. Odtąd skupimy się na rozwiązywaniu obu przypadków osobno.

Pierwszy przypadek: LCA

Zakładamy, że u i v leżą na jednym drzewie, i szukamy ich najniższego wspólnego przodka (czyli wspomnianego wcześniej LCA). To już jest standardowy problem. Algorytm znajdujący LCA w czasie $O(\log n)$ (po wstępnych obliczeniach o złożoności czasowej i pamięciowej $O(n \log n)$) można znaleźć, na przykład, w opisie rozwiązania zadania *Komiwojażer Bajtazar* z IX Olimpiady Informatycznej [9].

Drugi przypadek: wynik leży na cyklu

W tym przypadku mamy dwóch kandydatów na optymalne miejsce spotkania. Musi nim być korzeń drzewa, na którym leży jeden z danych wierzchołków u, v.

Przede wszystkim, obydwa wierzchołki muszą znać swoją odległość od korzenia. Takie wartości możemy łatwo zapamiętywać podczas wstępnych obliczeń (algorytm BFS).

Teraz musimy jeszcze obliczyć odległość na cyklu z pierwszego korzenia do drugiego, a także z drugiego do pierwszego. Żeby to zrobić, zapamiętujemy długość cyklu oraz numerujemy kolejno jego wierzchołki. Odległość między dwoma korzeniami na cyklu to różnica ich numerów, gdy jest dodatnia, lub długość cyklu pomniejszona o tę różnicę w przeciwnym przypadku.

Podsumowanie

W celu określenia przypadku wykonujemy wstępne obliczenia działające w czasie $O(n)$. Potrzebujemy także czasu $O(n \log n)$ na wstępne obliczenia związane z wyznaczaniem LCA par wierzchołków. Następnie k razy znajdujemy optymalne miejsce spotkania. Obliczanie LCA działa w czasie $O(\log n)$, a obliczanie wyniku na cyklu — w czasie $O(1)$.

Cały algorytm działa więc w czasie $O((n + k) \log n)$ i pamięci $O(n \log n)$. Implementacje rozwiązania wzorcowego można znaleźć w plikach `ran.cpp`, `ran2.pas` i `ran3.cpp`. Natomiast w plikach `rans3.cpp`, `rans4.pas`, `rans5.cpp` i `rans6.pas` znajdują się implementacje rozwiązań wolniejszych, w których poszczególne fazy rozwiązania wzorcowego (obliczanie LCA, obsługa przypadku na cyklu) działają w czasie liniowym.

Testy

Rozwiązania zawodników sprawdzano za pomocą 10 zestawów testowych. Testy 1 i 2 to proste testy generowane ręcznie. Każdy z zestawów 3–10 składał się z trzech testów następujących typów:

- test *a*: jedna bardzo długa gałąź z losowymi zaburzeniami. Na takich testach wolno działają te rozwiązania, w których wyznaczanie LCA jest liniowe.
- test *b*: jeden bardzo duży cykl z losowymi zaburzeniami. Na takich testach wolno działają te rozwiązania, w których wyznaczanie najkrótszej ścieżki na cyklu jest liniowe.
- test *c*: test zawierający różne trudne przypadki. Mały graf i kilka zapytań są wszędzie takie same, reszta wierzchołków i zapytań jest wybrana losowo.

Poniżej znajduje się tabela zawierająca parametry poszczególnych testów.

Nazwa	**n**	**k**
ran1.in	3	5
ran2.in	9	4
ran3abc.in	1 000	1 000
ran4abc.in	2 000	2 000
ran5abc.in	50 000	50 000
ran6abc.in	100 000	100 000
ran7abc.in	200 000	200 000
ran8abc.in	500 000	250 000
ran9abc.in	500 000	500 000
ran10abc.in	500 000	500 000