Piotr Zieliński
Treść zadania, Opracowanie

Marek Cygan
Program

OI, Etap II, dzień próbny, 6.02.2008

Blokada

W Bajtocji znajduje się dokładnie n miast. Między niektórymi parami miast biegną dwukierunkowe drogi. Poza miastami nie ma skrzyżowań, lecz mogą istnieć mosty, tunele i estakady. Między każdymi dwoma miastami może istnieć co najwyżej jedna droga. Z każdego miasta da się dojechać — bezpośrednio bądź też pośrednio — do każdego innego.

W każdym mieście mieszka dokładnie jeden mieszkaniec. Mieszkańcom doskwiera samotność. Okazuje się, że każdy z mieszkańców chce odwiedzić każdego innego w jego mieście, i to dokładnie raz. A zatem powinno odbyć się $n \cdot (n-1)$ spotkań. Tak, tak, **powinno**. *Niestety w Bajtocji trwają protesty programistów, domagających się wprowadzenia interwencyjnego skupu oprogramowania. Programiści planują, w ramach protestu, zablokowanie możliwości wjeżdżania do jednego z miast Bajtocji, wyjeżdżania z niego i przejeżdżania przez nie. Zastanawiają się teraz, które miasto wybrać, tak aby jak najbardziej uprzykrzyć życie mieszkańcom Bajtocji.*

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis sieci dróg Bajtocji,*
- *dla każdego miasta obliczy, ile spotkań nie mogłoby się odbyć, gdyby owe miasto zostało zablokowane przez programistów,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby naturalne oddzielone pojedynczym odstępem: n oraz m ($2 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 500\,000$) oznaczające odpowiednio liczbę miast oraz liczbę dróg. Miasta są ponumerowane od 1 do n. W kolejnych m wierszach znajdują się opisy dróg. Każdy opis składa się z dwóch liczb a oraz b ($1 \leqslant a < b \leqslant n$) oddzielonych pojedynczym odstępem i oznacza, że istnieje droga między miastami o numerach a i b.

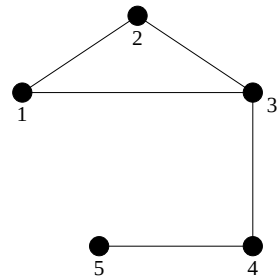
Wyjście

Twój program powinien wypisać dokładnie n liczb naturalnych, po jednej w wierszu. W i-tym wierszu powinna znaleźć się liczba spotkań, które nie odbyłyby się, gdyby programiści zablokowali miasto nr i.

Przykład

Dla danych wejściowych:

```
5 5
1 2
2 3
1 3
3 4
4 5
```



poprawnym wynikiem jest:

```
8
8
16
14
8
```

Rozwiązanie

Analiza problemu

Nawet początkujący uczestnik Olimpiady domyśla się z pewnością, że skoro w zadaniu jest mowa o miastach połączonych drogami, to rozwiązania należy szukać w teorii grafów. Tak jest w istocie — na Bajtocję możemy patrzeć jak na graf nieskierowany G, w którym miasta są wierzchołkami, a drogi krawędziami. Powiemy, że wierzchołek v jest osiągalny z wierzchołka u, jeśli istnieje w grafie G ścieżka zaczynająca się w u i kończąca się w v.

G jest grafem spójnym i nieskierowanym, co oznacza, że wszystkie wierzchołki są wzajemnie osiągalne. Dla każdego wierzchołka x chcemy znaleźć $wynik[x]$ — liczbę takich par (u,v), że po usunięciu z G wierzchołka x (wraz z incydentnymi krawędziami) v nie będzie osiągalne z u.

Bezpośrednia symulacja

Spróbujmy rozwiązać zadanie bezpośrednio. Aby obliczyć wynik dla wierzchołka x, usuniemy go z grafu, a następnie dla wszystkich par (u,v) sprawdzimy, czy są one wzajemnie osiągalne. Takie sprawdzenie możemy wykonać, przeszukując graf dowolną metodą, np. DFS. Złożoność testu dla jednej pary (u,v) będzie wówczas równa $O(n+m)$. Obliczenie wyniku dla jednego wierzchołka wymaga przeprowadzenia $O(n^2)$ sprawdzeń, czyli obliczenia dla całego grafu G zajmą czas $O(n^3 \cdot (n+m))$. Patrząc na rozmiar danych wejściowych, dochodzimy do wniosku, że to stanowczo za wolno.

Rozwiązanie wolne

Para wierzchołków u, v jest wzajemnie osiągalna w grafie G wtedy i tylko wtedy, gdy oba znajdują się w tej samej spójnej składowej. Gdybyśmy znali rozmiary spójnych składowych, na które rozpada się G po usunięciu z niego wierzchołka x, moglibyśmy w prosty sposób obliczyć dla niego wynik:

$$wynik[x] := 2 \cdot (n-1) + \sum_{i=1}^{k} s_i \cdot (n-1-s_i) \tag{1}$$

gdzie s_i to rozmiary spójnych składowych grafu z usuniętym wierzchołkiem x, a k to liczba owych składowych. Pierwszy składnik w powyższym wzorze to oczywiście liczba „uniemożliwionych" przez blokady spotkań „mieszkańca" wierzchołka x z wszystkimi pozostałymi. Następnie, i-ty składnik sumy to liczba „uniemożliwionych" spotkań „mieszkańców" i-tej składowej z „mieszkańcami" z pozostałych składowych.

Rozmiary spójnych składowych możemy łatwo wyznaczyć, korzystając np. z przeszukiwania grafu w głąb (czyli ponownie DFS). W ten sposób obliczamy odpowiedź dla każdego wierzchołka w czasie $O(n+m)$, co daje łączną złożoność czasową algorytmu $O(n \cdot (n+m))$. Algorytm ten uzyskiwał około połowy wszystkich punktów. Jego implementacja znajduje się w pliku `blos4.cpp`.

Rozwiązanie wzorcowe

Nietrudno zauważyć, że interesują nas tak naprawdę te wierzchołki, po usunięciu których G rozpada się na więcej niż jedną spójną składową — dla pozostałych wierzchołków wynik jest równy $2 \cdot (n-1)$. Wierzchołki takie w teorii grafów nazywa się *punktami artykulacji*. Okazuje się, że wszystkie punkty artykulacji grafu możemy znaleźć w czasie $O(n+m)$, korzystając z procedury przeszukiwania w głąb (znowu DFS).

W tym miejscu warto przypomnieć pewne pojęcia występujące w tej procedurze. Przeszukiwanie rozpoczynamy w wybranym wierzchołku, który nazwiemy *korzeniem*. W trakcie przeszukiwania, gdy po raz pierwszy odwiedzamy jakiś wierzchołek, to wchodzimy do niego z wierzchołka, który nazwiemy jego *ojcem*. Tak zdefiniowany korzeń i pojęcie ojca wyznaczają *drzewo przeszukiwania*, inaczej *drzewo DFS*. W trakcie przeszukiwania, będąc w wierzchołku v, staramy się odwiedzić wszystkich sąsiadów v z wyjątkiem jego ojca. Gdy okazuje się, że sąsiad u był już odwiedzony, to krawędź (v,u) nazywamy *krawędzią powrotną*. Ostatecznie w wyniku przeszukania grafu metodą DFS każda krawędź zostaje sklasyfikowana albo jako drzewowa, albo powrotna, tzn. każda krawędź łączy jakiś wierzchołek z pewnym jego potomkiem w drzewie DFS. To w szczególności oznacza, że nie ma tzw. krawędzi *skośnych*, czyli łączących wierzchołki z różnych poddrzew DFS.

Dodatkowo, w trakcie przeszukiwania możemy ponumerować wierzchołki grafu zgodnie z kolejnością, w jakiej je po raz pierwszy odwiedzamy. Kolejność ta jest zgodna z porządkiem *preorder* w drzewie przeszukiwań, a wartość przypisaną wierzchołkowi v oznaczymy $numer[v]$.

W trakcie procedury DFS możemy wyznaczyć sporo innych ciekawych własności drzewa przeszukiwań i grafu, w szczególności dla każdego wierzchołka v możemy obliczyć wartość funkcji *low* zdefiniowanej jako:

$$low[v] = \min\{numer(u) \mid \text{z } v \text{ da się dojść do } u, \text{ idąc krawędziami drzewa przeszukiwania w dół, a następnie co najwyżej raz krawędzią powrotną}\}.$$

Aby trochę lepiej zrozumieć sens powyższej definicji, przyjrzyjmy się pseudokodowi zmodyfikowanej procedury DFS, za pomocą której można wyznaczyć $low[v]$ (i oczywiście także $numer[v]$) dla każdego wierzchołka v grafu G.

```
1:  nr := 0; { zmienna pomocnicza do numeracji preorder }
2:  odwiedzony :=(false dla każdego wierzchołka);
3:  procedure DFS(v, ojciec)
4:  { Wywołanie: DFS(korzeń, nieistniejący_wierzchołek). }
5:  begin
6:    odwiedzony[v] :=true;
7:    nr := nr + 1; numer[v] := nr;
8:    low[v] := numer[v];
9:    for u ∈ Sąsiedzi(v) do
10:     if u ≠ ojciec then begin
11:       if not odwiedzony[u] then begin
12:         DFS(u, v);
13:         low[v] := min(low[v], low[u]);
14:       end else
15:         low[v] := min(low[v], numer[u]);
16:     end
17: end
```

Dla każdego wierzchołka v, $low[v]$ wyznaczamy w wywołaniu rekurencyjnym odpowiadającym v. Zaczynamy od $low[v] = numer[v]$ (wiersz 8.), a następnie próbujemy tę wartość jak najbardziej zmniejszyć. W tym celu analizujemy wszystkie wierzchołki u sąsiadujące z v różne od jego ojca i dla każdego z nich sprawdzamy, czy można poprowadzić przez niego ścieżkę z v typu *najpierw w dół drzewa, a potem co najwyżej jedna krawędź powrotna* do wierzchołka o mniejszym niż dotychczas znalezione numerze preorder. Jeżeli analizowany wierzchołek u jest już odwiedzony (wiersz 15.), to mamy krawędź powrotną, a zatem jedyną ścieżką spełniającą opisane warunki jest $v \to u$, prowadząca do wierzchołka o numerze $numer[u]$. Jeżeli zaś u nie jest odwiedzony (wiersze 12.–13.), to u znajdzie się w poddrzewie drzewa DFS ukorzenionym w v, czyli krawędź (v,u) prowadzi w dół drzewa. To oznacza, że szukaną ścieżkę z v możemy otrzymać, przedłużając o krawędź (v,u) dowolną ścieżkę spełniającą nasze kryterium wychodzącą z u — najlepsza z tych ścieżek prowadzi do wierzchołka o numerze $low[u]$. W momencie odwiedzenia v wierzchołek u nie jest jeszcze odwiedzony, więc musimy najpierw wykonać z niego rekurencyjnie przeszukiwanie (wiersz 12.), aby wartość $low[u]$ została policzona.

Jaki jest związek między funkcją *low* a punktami artykulacji? Okazuje się, że v jest punktem artykulacji G wtedy i tylko wtedy, gdy:

- jeżeli v nie jest korzeniem, to istnieje syn u wierzchołka v w drzewie DFS, dla którego zachodzi nierówność $low[u] \geqslant numer[v]$;
- jeżeli v jest korzeniem, to v ma co najmniej dwóch synów w drzewie DFS.

Zastanówmy się najpierw nad pierwszym z powyższych podpunktów. Jeśli zachodzi nierówność $low[u] \geqslant numer[v]$, to w wyniku usunięcia z grafu G wierzchołka v, u znajdzie się w odrębnej składowej wraz z ukorzenionym w nim poddrzewem DFS. Dlaczego? Ponieważ z żadnego wierzchołka w poddrzewie o korzeniu u nie wychodzi krawędź powrotna do żadnego przodka wierzchołka v (oraz dlatego, że w drzewie DFS nie ma krawędzi skośnych). Faktycznie, gdyby jakaś krawędź powrotna istniała, to wówczas zachodziłaby nierówność $low[u] < numer[v]$. Rozumując podobnie dla wszystkich wierzchołków u spełniających nierówność $low[u] \geqslant numer[v]$, otrzymamy, że usunięcie v spowoduje, że znajdą się one wraz z ukorzenionymi w nich poddrzewami DFS w osobnych spójnych składowych, a reszta grafu znajdzie się w jednej, osobnej składowej. To w szczególności oznacza, że jeżeli nie istnieje żaden u spełniający opisaną nierówność, to usunięcie v nie rozspójni G.

Przedstawiona argumentacja nie działa oczywiście, jeżeli v jest korzeniem — stąd druga część powyższego kryterium. Jej uzasadnienie jest całkiem proste. Jeżeli korzeń ma tylko jednego syna w drzewie DFS, to oczywiście jego usunięcie nie rozspójni grafu. Z kolei jeżeli synów korzenia jest więcej, to poddrzewa DFS w nich ukorzenione nie są połączone żadnymi krawędziami, gdyż byłyby to krawędzie skośne, a takich w przeszukiwaniu DFS nie ma. To oznacza, że usunięcie korzenia powoduje w tym przypadku rozkład grafu na składowe odpowiadające właśnie tym poddrzewom.

Dzięki zastosowaniu funkcji *low* potrafimy zatem rozpoznać w grafie punkty artykulacji i obliczać wynik zgodnie ze wzorem (1) jedynie dla nich. Nie polepsza to niestety złożoności pesymistycznej rozwiązania, która wynosi nadal $O(n \cdot (n+m))$, ale pozwala w niektórych przypadkach przyspieszyć rozwiązanie i wskutek tego zdobyć nieco więcej punktów. Implementację tego pomysłu można znaleźć w pliku `blos3.cpp`.

Maksymalną liczbę punktów można zdobyć, jeśli poczyni się jeszcze jedno spostrzeżenie. Już wcześniej zauważyliśmy, że spójne składowe grafu G powstałe po usunięciu z niego pewnego wierzchołka to (poza ewentualnie jedną) pewne poddrzewa DFS. Procedurę DFS można łatwo zmodyfikować tak, by przy okazji przechodzenia grafu obliczać jednocześnie dla każdego wierzchołka rozmiar poddrzewa DFS w nim ukorzenionego (przeprowadzenie tej modyfikacji pozostawiamy Czytelnikowi jako ćwiczenie). Jeśli mamy już tę informację (oraz policzoną funkcję *low*), to bez trudu możemy dla każdego punktu artykulacji v obliczyć wynik za pomocą wzoru (1), założywszy na przykład, że $s_1, \ldots, s_{k-1}$ ze wzoru odpowiadają pewnym poddrzewom DFS ukorzenionym w synach wierzchołka v, natomiast

$$s_k = n - 1 - s_1 - \ldots - s_{k-1}$$

(jeżeli v jest korzeniem, to $s_k = 0$). Rozwiązanie takie przechodzi wszystkie testy i można je znaleźć: zaimplementowane w C++ w pliku `blo.cpp`, w Pascalu — w pliku `blo2.pas` oraz w Javie — w pliku `blo3.java`. Działa oczywiście w czasie $O(n+m)$, gdyż wymaga zaledwie jednego przeszukania DFS grafu.

Wszystkie przedstawione w tym opracowaniu rozwiązania mają złożoność pamięciową $O(n+m)$, jako że przechowują graf w pamięci w postaci list sąsiedztwa. Podczas implementacji rozwiązania należało zwrócić uwagę na to, że obliczenia należy przeprowadzać na liczbach 64-bitowych, gdyż wynik dla pojedynczego wierzchołka może być nawet rzędu $O(n^2)$.

Testy

Testy zostały wygenerowane losowo. W poniższym opisie n oznacza liczbę wierzchołków w grafie, m — liczbę krawędzi, natomiast a — liczbę punktów artykulacji w grafie. Rozwiązania wolniejsze o złożoności $O(n \cdot (n+m))$ przechodziły testy 1–5. Dzięki zastosowaniu usprawnienia polegającego na przeszukiwaniu grafu jedynie dla punktów artykulacji możliwe było również zdobycie punktów za test 6.

Nazwa	n	m	a
blo1.in	8	12	2
blo2.in	30	50	6
blo3.in	100	150	26
blo4.in	250	800	24
blo5.in	500	10 000	22
blo6.in	10 000	90 000	774
blo7.in	40 000	100 000	2 937
blo8.in	100 000	200 000	6 762
blo9.in	100 000	350 000	7 260
blo10.in	100 000	500 000	6 946