Marcin Pilipczuk
Treść zadania, Opracowanie

Bartłomiej Wołowiec
Program

Dostępna pamięć: 64 MB.

OI, Etap I, 19.10–16.11.2009

Gildie

Król Bajtazar ma nie lada problem. Gildia Szwaczek oraz Gildia Krawców jednocześnie poprosiły o pozwolenie na otwarcie swoich filii w każdym z miast królestwa.

W Bajtocji jest n miast. Niektóre z nich są połączone dwukierunkowymi drogami. Każda z gildii wysunęła postulat, aby dla każdego miasta:

- *znajdowała się w nim filia danej gildii, lub*
- *miasto było bezpośrednio połączone drogą z miastem, w którym znajduje się filia tej gildii.*

Z drugiej strony, Król Bajtazar wie, że jeśli pozwoli w jednym mieście otworzyć filie obu gildii, to zapewne doprowadzi to do zmowy gildii i zmonopolizowania rynku odzieżowego. Dlatego też poprosił Cię o pomoc.

Wejście

W pierwszym wierszu standardowego wejścia podane są dwie liczby całkowite n oraz m ($1 \leqslant n \leqslant 200\,000$, $0 \leqslant m \leqslant 500\,000$), oznaczające odpowiednio liczbę miast i liczbę dróg w Bajtocji. Miasta są ponumerowane od 1 do n. W $(i+1)$-szym wierszu wejścia znajduje się opis i-tej drogi; zawiera on liczby a_i oraz b_i ($1 \leqslant a_i, b_i \leqslant n$, $a_i \neq b_i$) oznaczające, że i-ta droga łączy miasta a_i oraz b_i. Każda para miast jest połączona co najwyżej jedną drogą. Drogi nie krzyżują się — jedynie mogą spotykać się w miastach — choć mogą prowadzić tunelami i estakadami.

Wyjście

W pierwszym wierszu standardowego wyjścia Twój program powinien wypisać jedno słowo: `TAK` *— jeśli da się rozmieścić filie gildii w miastach zgodnie z warunkami zadania, lub* `NIE` *— w przeciwnym przypadku. W przypadku odpowiedzi* `TAK`*, w kolejnych n wierszach powinien znaleźć się opis przykładowego rozmieszczenia filii. $(i+1)$-szy wiersz powinien zawierać:*

- *literę* `K`*, jeśli w mieście i ma się znaleźć filia gildii krawców, lub*
- *literę* `S`*, jeśli w mieście i ma się znaleźć filia gildii szwaczek, lub*
- *literę* `N`*, jeśli w mieście i nie ma się znaleźć filia żadnej z dwóch gildii.*

Przykład

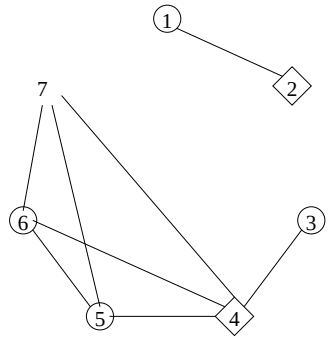
Dla danych wejściowych:

```
7 8
1 2
```

```
3 4
5 4
6 4
7 4
5 6
5 7
6 7
```

poprawnym wynikiem jest:

```
TAK
K
S
K
S
K
K
N
```



Miasta, w których ma zostać otwarta gildia krawców, są zaznaczone kółkami, a te, w których ma zostać otwarta gildia szwaczek, są zaznaczone rombami.

Rozwiązanie

Wprowadzenie

Zadanie *Gildie* było zadaniem prostym, choć dość nietypowym. W przypadku odpowiedzi `TAK`, przykładowe rozmieszczenie gildii można skonstruować na wiele różnych sposobów — intuicyjnie, jest duża swoboda w trakcie konstrukcji. Główną częścią rozwiązania było właśnie wyznaczenie poprawnego rozmieszczenia.

Zacznijmy od prostej obserwacji. Załóżmy, że mamy dane przykładowe poprawne rozmieszczenie filii (takie, w którym postulaty obu gildii są spełnione). Jeśli w każdym mieście, w którym nie postawiliśmy żadnej gildii (literka `N` na wyjściu), postawimy dowolną z gildii, rozmieszczenie wciąż pozostanie poprawne. Dlatego też wystarczy rozważać tylko te rozmieszczenia, w których w każdym mieście jest filia jednej z gildii.

Przetłumaczmy teraz treść zadania na język teorii grafów. Mamy dany nieskierowany graf G, w którym miasta Bajtocji tworzą zbiór wierzchołków V, a drogi łączące miasta — zbiór krawędzi E. Naszym zadaniem jest podzielić zbiór wierzchołków V na dwie rozłączne części V_K i V_S — lokalizacje dla filii odpowiednio gildii krawców i szwaczek — tak, by każdy wierzchołek $v \in V$ spełniał dwa warunki:

- $v \in V_K$ lub jakiś sąsiad v należy do V_K,
- $v \in V_S$ lub jakiś sąsiad v należy do V_S.

Wprowadźmy teraz kilka pojęć z teorii grafów. Powiemy, że wierzchołek v *dominuje* wierzchołek w, jeśli $v = w$ lub v jest sąsiadem w. Dla danego zbioru wierzchołków S powiemy, że wierzchołek w jest *zdominowany* przez S, jeśli istnieje w S wierzchołek dominujący w. Zbiór wierzchołków $D \subseteq V$ nazwiemy *zbiorem dominującym*, jeśli

dominuje on wszystkie wierzchołki w grafie. Wyposażeni w nową definicję, możemy przeformułować nasze zadanie tak: mamy podzielić zbiór wierzchołków V na dwa rozłączne zbiory dominujące V_K i V_S.

Liczba domatyczna (ang. *domatic number*) grafu G to największa liczba całkowita k taka, że zbiór wierzchołków V da się podzielić na k parami rozłącznych zbiorów dominujących. W zadaniu mamy więc sprawdzić, czy liczba domatyczna podanego grafu (miast i dróg w Bajtocji) wynosi co najmniej 2. W ogólności, wyznaczenie liczby domatycznej grafu jest problemem NP-trudnym, to znaczy, że prawdopodobnie nie da się jej obliczyć w czasie wielomianowym ze względu na wielkość grafu. Jednak, jak pokażemy, rozstrzygnięcie, czy liczba domatyczna wynosi 1, czy też jest nie mniejsza niż 2, jest dość proste.

Zauważmy, iż jeśli w grafie istnieje wierzchołek, z którego nie wychodzą żadne krawędzie (tzw. wierzchołek izolowany), to odpowiedź brzmi `NIE`. Istotnie, jeśli w tym wierzchołku postawimy filię jednej z gildii, to nie mamy gdzie postawić filii drugiej gildii tak, by zdominowała ten wierzchołek. Nietrudno dostrzec, że sprawdzenie, czy w grafie jest wierzchołek izolowany, możemy wykonać w czasie $O(|V|+|E|)$. Wystarczy zliczyć stopnie wierzchołków.

Pokażemy, że jeśli w grafie nie ma wierzchołków izolowanych, to odpowiedź w zadaniu brzmi `TAK`. Dowodząc tego, opiszemy trzy różne sposoby konstrukcji podziału zbioru wierzchołków na dwa zbiory dominujące V_K i V_S. Wszystkie konstrukcje będą działały w czasie $O(|V| + |E|)$. Otrzymamy więc trzy rozwiązania o liniowej złożoności czasowej. Istnieje też mnóstwo innych konstrukcji działających w czasie liniowym. Uznajmy jednak, że na potrzeby tego opracowania wystarczy nam opis trzech.

Rozwiązania wzorcowe

Jak już zauważyliśmy we wprowadzeniu, jeśli w Bajtocji istnieje miasto, z którego nie wychodzi żadna droga, odpowiedź brzmi `NIE`. Pokażemy trzy różne konstrukcje rozmieszczenia filii gildii, przy założeniu, że z każdego miasta wychodzi choć jedna droga. Każda konstrukcja będzie wypełniała tablicę *gildia*, przypisując gildie kolejnym miastom. Początkowo tablica *gildia* jest zainicjowana na wartości N (brak przypisania), w miarę działania algorytmu będziemy przypisywać miastom wartości K (gildia krawców) lub S (gildia szwaczek).

Konstrukcja pierwsza

W tym algorytmie przeglądamy listę krawędzi w dowolnej kolejności. Dla każdej krawędzi:

1. jeśli oba końce krawędzi mają już przypisane gildie, nic nie robimy,
2. jeśli jeden koniec krawędzi ma przypisaną gildię, przypisujemy inną gildię drugiemu końcowi tej krawędzi,
3. jeśli żaden z końców krawędzi nie ma przypisanej gildii, przypisujemy w dowolny sposób obu końcom różne gildie.

Oto pseudokod powyższego rozwiązania:

```
1:  procedure UstawPrzeciwna(v, w)
2:  begin
3:    if gildia[v] = N then begin
4:      if gildia[w] = S then gildia[v] := K
5:      else gildia[v] := S;
6:    end
7:  end
8:
9:  begin
10:   for v := 1 to n do gildia[v] := N;
11:   foreach (v, w) ∈ E do begin
12:     UstawPrzeciwna(v, w);
13:     UstawPrzeciwna(w, v);
14:   end
15: end
```

Spróbujmy teraz wykazać, że powyższe rozwiązanie konstruuje poprawne przypisanie gildii miastom. Wpierw zauważmy, że każde miasto będzie miało przypisaną jakąś gildię: skoro nie ma wierzchołków izolowanych, każdy wierzchołek jest incydentny z jakąś krawędzią, a zatem w momencie przeglądania tej krawędzi przypiszemy wierzchołkowi gildię, jeśli nie była przypisana wcześniej.

By dokończyć dowód poprawności tej konstrukcji, wystarczy zauważyć jedno: w momencie, gdy przypisujemy wierzchołkowi v jakąś gildię (dla ustalenia uwagi: gildię krawców), wierzchołek v ma już pewnego sąsiada w, któremu przypisana jest gildia szwaczek (być może właśnie ją przypisujemy). Istotnie, w chwili, gdy przeglądamy krawędź (v, w) i przypisujemy w wyniku tego przeglądania wierzchołkowi v gildię krawców, to albo do w już była przypisana gildia szwaczek (przypadek drugi algorytmu), albo przypiszemy ją w bieżącym kroku (przypadek trzeci algorytmu). Tą sprytną obserwacją zakończyliśmy dowód poprawności pierwszej konstrukcji.

Implementację takiego rozwiązania można znaleźć w pliku `gil1.cpp`.

Konstrukcja druga

W tym algorytmie przeglądamy listę wierzchołków w dowolnej kolejności. Dla każdego wierzchołka, jeśli nie ma on jeszcze przypisanej gildii, przypisujemy mu gildię krawców, zaś wszystkim jego sąsiadom, którym jeszcze nie przypisaliśmy gildii, przypisujemy gildię szwaczek. Oto pseudokod:

```
1:  begin
2:    for v := 1 to n do gildia[v] := N;
3:    for v := 1 to n do
4:      if gildia[v] = N then begin
5:        gildia[v] := K;
6:        foreach w : (v, w) ∈ E do
7:          if gildia[w] = N then
```

```
8:              gildia[w] := S;
9:      end
10: end
```

Przejdźmy do dowodu poprawności algorytmu. Oczywiście, każde miasto będzie miało przypisaną jakąś gildię. Spójrzmy teraz na dowolny wierzchołek w, któremu przypisaliśmy gildię szwaczek. Skoro to uczyniliśmy, musiał on być sąsiadem jakiegoś wierzchołka v, któremu przypisaliśmy gildię krawców. Zatem zbiór wierzchołków, którym przypisaliśmy gildię krawców, jest zbiorem dominującym.

Z drugiej strony zauważmy, że żadne dwa wierzchołki, którym przypisaliśmy gildię krawców, nie mogą być sąsiadujące. Jeśli bowiem jakiemuś wierzchołkowi przypisaliśmy gildię krawców, automatycznie wszyscy jego nieprzypisani sąsiedzi otrzymali filię gildii szwaczek. Innymi słowy, wszyscy sąsiedzi wierzchołków z filią gildii krawców mają przypisaną gildię szwaczek. Skoro każdy wierzchołek ma co najmniej jednego sąsiada, miasta z gildią szwaczek tworzą zbiór dominujący.

Można powyższą konstrukcję wyrazić trochę inaczej. *Zbiorem niezależnym* w grafie G nazwiemy taki zbiór wierzchołków I, że żadne dwa wierzchołki z tego zbioru nie są połączone krawędzią. Powyższy algorytm można zapisać następująco:

1. Znajdź dowolny maksymalny w sensie zawierania zbiór niezależny I.
2. Przypisz wszystkim wierzchołkom z I gildię krawców, a pozostałym wierzchołkom gildię szwaczek.

Implementację tej wersji rozwiązania wzorcowego można znaleźć w pliku `gil3.cpp`.

Konstrukcja trzecia

W tym algorytmie przeglądamy listę wierzchołków w dowolnej kolejności. Załóżmy, że rozpatrujemy wierzchołek v, który jeszcze nie ma przypisanej gildii. Wówczas będziemy przypisywać gildie wszystkim wierzchołkom w całej spójnej składowej, do której należy v. Innymi słowy, przeglądamy wszystkie spójne składowe grafu G i dla każdej z nich niezależnie wybieramy przypisanie miastom gildii.

Załóżmy więc, że analizujemy wierzchołek v należący do pewnej spójnej składowej H. Konstruujemy dowolne drzewo rozpinające H (np. drzewo przeszukiwania w głąb) i ukorzeniamy je w v. Wierzchołkowi w przyporządkowujemy gildię krawców, jeśli jego odległość od v w drzewie rozpinającym jest parzysta, zaś gildię szwaczek, jeśli nieparzysta. Oto pseudokod tego rozwiązania, wykorzystujący drzewo przeszukiwania w głąb:

```
1: procedure DFS(v, g)
2: begin
3:   if gildie[v] = N then begin
4:     gildie[v] := g;
5:     if g = S then g := K else g := S;
6:     foreach w : (v, w) ∈ E do
7:       DFS(w, g);
8:   end
```

```
 9: end
10:
11: begin
12:    for v := 1 to n do gildia[v] := N;
13:    for v := 1 to n do DFS(v, K);
14: end
```

Uzasadnijmy poprawność przedstawionej konstrukcji. Oczywiście, każdemu wierzchołkowi przypiszemy jakąś gildię. Jeśli wierzchołek v nie był korzeniem drzewa rozpinającego, to jego ojcu w tym drzewie przypiszemy inną gildię niż v, czyli wierzchołek v będzie zdominowany przez obie gildie. Z drugiej strony, jeśli v jest korzeniem drzewa rozpinającego, to ma co najmniej jednego syna w tym drzewie (v nie jest izolowany), czyli ma sąsiada o innej gildii niż on sam. Tym samym, opisany algorytm konstruuje poprawne przyporządkowanie miastom gildii.

Implementacje takiego rozwiązania można znaleźć w plikach `gil.cpp` i `gil2.pas`.

Rozwiązania alternatywne

Rozwiązanie polegające na rozpatrzeniu wszystkich możliwych przyporządkowań miastom gildii działa w czasie $O\left((|V|+|E|)\cdot 3^{|V|}\right)$ lub $O\left((|V|+|E|)\cdot 2^{|V|}\right)$, jeśli uwzględnimy obserwację, że możemy w każdym mieście umieścić filię którejś gildii. Takie rozwiązania otrzymywały w tym zadaniu pojedyncze punkty.

Nie znamy żadnych ciekawych rozwiązań *pośrednich*. Wszystkie rozwiązania działające istotnie szybciej niż rozwiązania przeszukujące wszystkie możliwe odpowiedzi, lecz istotnie wolniej niż wzorcowe, okazywały się niepotrzebnie przekomplikowanymi rozwiązaniami wzorcowymi.

W tym zadaniu można też było wymyślić różnorakie rozwiązania niepoprawne. Dla przykładu, rozważmy następującą heurystykę. Przeglądamy wszystkie wierzchołki w dowolnej kolejności. Jeśli dany wierzchołek v ma jakiegoś sąsiada z przypisaną gildią, przyporządkowujemy v inną gildię niż ma ów sąsiad. W przeciwnym wypadku przyporządkowujemy v gildię inną niż ostatnio przypisana. To rozwiązanie może działać niepoprawnie już dla bardzo małego grafu: ścieżki złożonej z trzech wierzchołków. Jeśli wpierw rozważymy oba końce tej ścieżki, przypiszemy im różne gildie. Wówczas cokolwiek przypiszemy środkowemu wierzchołkowi, nie otrzymamy poprawnego przyporządkowania.

Testy

Zadanie było sprawdzane na 22 zestawach danych testowych, pogrupowanych w 12 grup testów.

Nazwa	n	m	Opis
gil1a.in	100	95	drzewa i ścieżki po 10 miast, dodatkowe losowe krawędzie

Nazwa	n	m	Opis
gil1b.in	1	0	najmniejszy test
gil2a.in	972	946	drzewa i ścieżki po 2-50 miast, dodatkowe losowe krawędzie
gil2b.in	982	946	test *2a* plus 10 wierzchołków izolowanych
gil3a.in	1 073	1 827	drzewa po 2-30 miast, ponad 1800 losowych krawędzi
gil3b.in	1 074	1 827	test *3a* plus jeden wierzchołek izolowany
gil4a.in	183 736	173 226	drzewa 5-30 miast
gil4b.in	200 000	0	duży test bez krawędzi
gil5a.in	191 726	497 272	ścieżka łącząca wszystkie miasta, dodatkowo dużo krawędzi losowych
gil5b.in	191 727	497 272	test *5a* plus jeden wierzchołek izolowany
gil6.in	192 837	499 231	duży losowy test
gil7a.in	58 263	291 314	losowy test, odpowiedź `NIE`
gil7b.in	59 273	49 422	dużo małych drzew po 2-10 miast
gil8a.in	196 372	196 371	ścieżka łącząca wszystkie miasta
gil8b.in	197 253	197 335	ścieżka łącząca wszystkie miasta, dodatkowo losowe krawędzie
gil9.in	198 372	198 356	ścieżki po 1 000-10 000 miast
gil10a.in	196 382	196 446	ścieżki po 1 000-30 000 miast, dodatkowe losowe krawędzie
gil10b.in	200 000	500 000	cykl z wszystkimi miastami, dodatkowo drogi między miastami odległymi o 2 i między co drugą parą miast odległych o 3
gil11a.in	197 332	197 318	ścieżki i drzewa po 2-50 000 miast
gil11b.in	197 382	197 318	test *11a* plus 50 wierzchołków izolowanych
gil12a.in	199 233	365 871	zawiera pełną klikę 600 wierzchołków, a poza tym losowe ścieżki i drzewa po 2-30 wierzchołków
gil12b.in	199 234	365 871	test *12a* plus jeden wierzchołek izolowany