Michał Pilipczuk
Treść zadania, Opracowanie

Zbigniew Wojna
Program

Dostępna pamięć: 256 MB.

OI, Etap II, dzień pierwszy, 09.02.2011

Śmieci

Przedsiębiorstwo Oczyszczania Bajtogrodu (POB) podniosło drastycznie ceny za wywóz śmieci. Część mieszkańców zrezygnowała z płacenia za wywóz śmieci i zaczęła wyrzucać je na ulice. W rezultacie wiele ulic Bajtogrodu tonie w śmieciach.

Sieć drogowa Bajtogrodu składa się z n skrzyżowań, z których niektóre połączone są dwukierunkowymi ulicami. Żadne dwie ulice nie łączą tej samej pary skrzyżowań. Niektóre z ulic są zaśmiecone, podczas gdy inne nie.

Burmistrz Bajtogrodu, Bajtazar, zdecydował się na niekonwencjonalną akcję mającą skłonić mieszkańców do płacenia za wywóz śmieci. Postanowił on oczyścić tylko niektóre ulice miasta — te, przy których większość mieszkańców opłaciła wywóz śmieci. Natomiast te ulice, przy których większość mieszkańców nie opłaciła wywozu śmieci, postanowił pozostawić zaśmiecone lub — jeśli to konieczne — zwieźć na nie śmieci z innych ulic! Bajtazar przygotował plan miasta, na którym zaznaczył, które ulice docelowo powinny być czyste, a które zaśmiecone. Niestety, pracownicy POB-u nie są w stanie ogarnąć planu Bajtazara. Są jednak w stanie wykonywać niezbyt skomplikowane zlecenia.

Pojedyncze zlecenie polega na wykonaniu kursu śmieciarką, rozpoczynającego się na dowolnie wybranym skrzyżowaniu, prowadzącego określonymi ulicami i kończącego się na tym samym skrzyżowaniu, na którym zaczął się kurs. Przy tym, każde skrzyżowanie może w jednym kursie zostać odwiedzone co najwyżej raz, z wyjątkiem skrzyżowania, od którego kurs się rozpoczął i na którym się kończy (na którym śmieciarka pojawia się dokładnie dwa razy). Śmieciarka, jadąc zaśmieconą ulicą, sprząta ją, jadąc zaś czystą ulicą, wręcz przeciwnie — zaśmieca ją, wyrzucając śmieci.

Bajtazar zastanawia się, czy może zrealizować swój plan, zlecając ileś kursów śmieciarki. Pomóż mu i napisz program, który wyznaczy zestaw takich kursów lub stwierdzi, że nie jest to możliwe.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite oddzielone pojedynczym odstępem: n i m ($1 \leqslant n \leqslant 100\,000$, $1 \leqslant m \leqslant 1\,000\,000$), oznaczające odpowiednio liczbę skrzyżowań oraz liczbę ulic w Bajtogrodzie. Skrzyżowania są ponumerowane od 1 do n. W kolejnych m wierszach znajdują się opisy kolejnych ulic, po jednej w wierszu. W każdym z tych wierszy znajdują się po cztery liczby całkowite oddzielone pojedynczymi odstępami: a, b, s i t ($1 \leqslant a < b \leqslant n$, $s, t \in \{0, 1\}$). Taka czwórka oznacza, że skrzyżowania a i b są połączone ulicą, przy czym s oznacza obecny stan zaśmiecenia ulicy (0 oznacza czystą, a 1 zaśmieconą), zaś t stan docelowy według planu Bajtazara.

Możesz założyć, że jeśli istnieje zestaw kursów realizujący plan Bajtazara, to istnieje również taki zestaw, w którym łączna liczba ulic, którymi prowadzą kursy śmieciarki, nie przekracza $5 \cdot m$.

W testach wartych 60% punktów zachodzi dodatkowo ograniczenie $m \leqslant 10\,000$.

Wyjście

Jeżeli za pomocą kursów śmieciarką nie da się zrealizować planu Bajtazara, to pierwszy i jedyny wiersz standardowego wyjścia powinien zawierać słowo „`NIE`*". W przeciwnym razie na wyjściu należy wypisać dowolny zestaw kursów realizujący plan Bajtazara, w którym łączna liczba ulic, którymi prowadzą kursy, nie przekracza $5 \cdot m$. Pierwszy wiersz wyjścia powinien zawierać k: liczbę kursów w zestawie. W kolejnych k wierszach powinny znaleźć się opisy kolejnych kursów, po jednym w wierszu. Wiersz $(i+1)$-szy powinien zaczynać się dodatnią liczbą k_i oznaczającą liczbę ulic, którymi prowadzi i-ty kurs. Po pojedynczym odstępie powinno znaleźć się $k_i + 1$ numerów kolejnych skrzyżowań, przez które prowadzi kurs, pooddzielanych pojedynczymi odstępami.*

Przykład

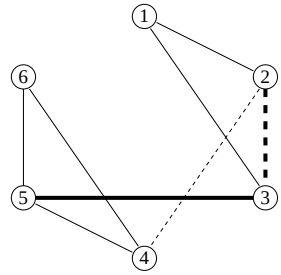
Na rysunkach cienką linią zaznaczono ulice, które są czyste, a grubą te, które są zaśmiecone. Linią przerywaną zaznaczono ulice, które docelowo powinny być czyste, a ciągłą te, które docelowo powinny być zaśmiecone.

Dla danych wejściowych:

```
6 8
1 2 0 1
2 3 1 0
1 3 0 1
2 4 0 0
3 5 1 1
4 5 0 1
5 6 0 1
4 6 0 1
```

jednym z poprawnych wyników jest:

```
2
3 1 2 3 1
3 4 6 5 4
```

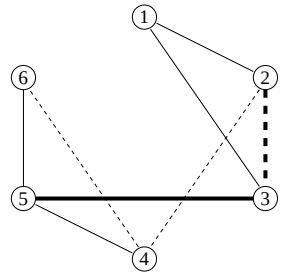


natomiast dla danych wejściowych:

```
6 8
1 2 0 1
2 3 1 0
1 3 0 1
2 4 0 0
3 5 1 1
4 5 0 1
5 6 0 1
4 6 0 0
```

poprawnym wynikiem jest:

```
NIE
```



Rozwiązanie

Analiza problemu

Sformułujmy nasze zadanie w języku teorii grafów. Niech $G = (V, E)$ będzie grafem reprezentującym plan Bajtogrodu: wierzchołki to skrzyżowania, a krawędzie to łączące je ulice. Z treści zadania wiemy, że krawędzie łączą różne wierzchołki i każde dwa wierzchołki łączy co najwyżej jedna krawędź. „Plany zaśmiecenia" możemy reprezentować jako etykietowania krawędzi liczbami 0 (ulica czysta) oraz 1 (ulica zaśmiecona). Kursom śmieciarki odpowiadają natomiast cykle proste, czyli takie cykle w grafie, w których każdy wierzchołek występuje co najwyżej raz. Problem postawiony w zadaniu daje się teraz przetłumaczyć następująco. Mamy dane dwa etykietowania i pytamy, czy da się jedno z nich przekształcić w drugie, używając jedynie operacji zamiany zer na jedynki i jedynek na zera wzdłuż cykli prostych. Jeśli odpowiedź jest twierdząca, chcemy wskazać odpowiedni ciąg operacji. Żądamy przy tym, aby łączna długość cykli nie była zbyt duża.

Na początek, dla wygody zapisu, zmieńmy nieco konwencję. Od teraz ulicę czystą będziemy oznaczać etykietą 1, zaś zaśmieconą — etykietą -1. Ustalmy pewien plan zaśmiecenia, czyli etykietowanie $P : E \to \{-1, 1\}$. Każdemu wierzchołkowi $v \in V$ przypiszmy liczbę $\mu_P(v) = \prod_{vw \in E} P(vw)$, czyli iloczyn etykiet na krawędziach wychodzących z wierzchołka v. Mówiąc prosto, $\mu_P(v)$ jest równe 1, jeśli parzyście wiele krawędzi wychodzących z v jest zaśmieconych, a -1 w przeciwnym przypadku.

Zobaczmy, co się dzieje, gdy wykonujemy operację zamiany wzdłuż pewnego cyklu. Zauważmy, że dla każdego wierzchołka v z cyklu zmienią się dokładnie dwie etykiety wychodzących z niego krawędzi. Oznacza to, że wartość $\mu_P(v)$ zostanie przemnożona przez -1 dwukrotnie, czyli w ogóle się nie zmieni! Jednocześnie, wartości μ_P dla wierzchołków spoza cyklu oczywiście również nie ulegną zmianie. Wynika stąd, że liczby $\mu_P(v)$ dla $v \in V$ nie zmieniają się w ogóle wskutek wykonywanych operacji.

Mając dane (wejściowe) etykietowania P_1 i P_2, możemy obliczyć wartości $\mu_{P_1}(v)$, $\mu_{P_2}(v)$ dla wszystkich wierzchołków $v \in V$. Jeśli któraś para się nie zgodzi, musimy odpowiedzieć „`NIE`". W dalszych rozważaniach przyjmijmy wobec tego, że dla każdego wierzchołka zachodzi $\mu_{P_1}(v) = \mu_{P_2}(v)$. Okazuje się, że jest to warunek wystarczający na istnienie rozwiązania.

Rozważmy zbiór F złożony z tych krawędzi wyjściowego grafu G, na których etykietowania P_1 oraz P_2 są różne. Pokażemy, że w grafie $H = (V, F)$ utworzonym z tych krawędzi stopnie wszystkich wierzchołków są parzyste. Zauważmy, że ponieważ dla dowolnego ustalonego wierzchołka $v \in V$ mamy $\mu_{P_1}(v) = \mu_{P_2}(v)$, to w grafie H ten wierzchołek ma parzysty stopień. Istotnie, skoro $\mu_{P_1}(v) = \mu_{P_2}(v)$, to równoważnie $\mu_{P_1}(v)\mu_{P_2}(v) = 1$. Wobec tego

$$1 = \mu_{P_1}(v)\mu_{P_2}(v) = \prod_{vw \in E} P_1(vw) \prod_{vw \in E} P_2(vw) = \prod_{vw \in E} \left(P_1(vw)P_2(vw)\right).$$

W ostatnim iloczynie czynniki są równe 1 dla tych krawędzi, dla których P_1 i P_2 się zgadzają, i -1 dla tych, dla których się nie zgadzają. Skoro cały iloczyn ma być równy 1, tych drugich jest parzyście wiele.

Zauważmy, że aby przerobić etykietowanie P_1 na etykietowanie P_2, musimy co najmniej raz zmienić etykietę na każdej krawędzi z F, czyli suma długości cykli musi wynieść co najmniej $|F|$. Okazuje się, że $|F|$ już wystarcza — da się tak zestawić kursy śmieciarki, żeby każdą krawędzią z F przejechać dokładnie raz. Jest to ściślej sformułowane w następującym lemacie.

Lemat 1. Jeśli każdy wierzchołek grafu $H = (V, F)$ ma parzysty stopień, to H da się rozłożyć na krawędziowo rozłączne cykle proste. Formalnie, istnieje taki zbiór cykli prostych $\mathcal{C}$, że każda krawędź $e \in F$ należy do dokładnie jednego cyklu $C \in \mathcal{C}$.

Dowód: Przeprowadzimy dowód indukcyjny (względem liczby krawędzi). Jeśli graf nie ma żadnej krawędzi, teza jest w sposób trywialny prawdziwa. Załóżmy wobec tego, że w H jest krawędź.

Wpierw pokażmy, że w H jest wówczas pewien cykl prosty. Weźmy dowolny wierzchołek v_0, z którego wychodzi jakaś krawędź. Przejdźmy tą krawędzią do jego sąsiada v_1. Skoro v_1 ma stopień parzysty, to musi z niego wychodzić jeszcze jakaś inna krawędź, prowadząca, powiedzmy, do v_2. Przejdźmy więc wzdłuż tej krawędzi. Teraz z kolei v_2 ma stopień parzysty, więc wychodzi z niego jakaś krawędź różna od tej, którą przyszliśmy. Kontynuujemy tak budowę ścieżki $v_0, v_1, v_2, \ldots$ aż do momentu, gdy dla któregoś z wierzchołków v_i okaże się, że leży on na już zbudowanej ścieżce — powtórzyliśmy wierzchołek, czyli $v_i = v_j$ dla pewnego $j < i$. Wierzchołków jest skończenie wiele, więc w końcu musi tak się stać. Wówczas krawędzie łączące kolejno wierzchołki $v_j, v_{j+1}, \ldots, v_i = v_j$ tworzą oczywiście cykl. Co więcej, jest to cykl prosty, gdyż cały czas budowaliśmy ścieżkę prostą.

Mając już cykl w garści, możemy usunąć go z grafu H. Zauważmy, że po usunięciu cyklu stopień każdego wierzchołka nie zmienił się lub spadł o 2, więc w szczególności pozostał parzysty. Wobec tego nasz cykl możemy dołączyć do rozkładu, którego istnienie wynika z założenia indukcyjnego. ■

Widzimy zatem, że po obliczeniu wartości $\mu_{P_1}(v)$, $\mu_{P_2}(v)$ i sprawdzeniu ich równości dla każdego v, wystarczy efektywnie zaimplementować znajdowanie rozkładu na cykle, którego istnienie gwarantuje Lemat 1. Przeprowadzony dowód daje już pewien algorytm. Zastanówmy się nad jego złożonością. Pesymistycznie może okazać się, że w każdym kroku z bardzo długiej ścieżki odcinamy bardzo krótki cykl. Oznacza to, że aż $\Theta(m)$ razy możemy budować ścieżkę długości $\Theta(n)$, czyli czas działania całego algorytmu możemy oszacować jedynie przez $O(mn)$. Programy implementujące taki algorytm, jak na przykład `smis2.cpp` i `smis4.pas`, otrzymywały około 20 punktów.

Rozwiązanie wzorcowe

Zauważmy, że w poprzednim rozwiązaniu zupełnie niepotrzebnie po odcięciu cyklu wyrzucaliśmy już zbudowany odcinek ścieżki. Zamiast tego, możemy budować kolejny cykl, rozpoczynając nie od pustej ścieżki, ale od fragmentu już zbudowanego, powstałego po odcięciu odcinka pomiędzy v_i a jego poprzednim wystąpieniem v_j. W ten sposób każda krawędź grafu H jest dokładana do ścieżki dokładnie raz — ponieważ gdy jest usuwana ze ścieżki, to razem z całym cyklem jest też wyrzucana w ogóle z grafu H. Możemy zresztą usuwać krawędź z grafu już przy dodawaniu jej do ścieżki

— wtedy nie musimy dodatkowo sprawdzać, czy przypadkiem nie cofnęliśmy się na ścieżce. Cały algorytm wygląda teraz następująco:

```
1:  procedure Śmieci
2:  begin
3:    Wczytaj graf G oraz etykietowania P_1, P_2;
4:    Dla każdego v ∈ V oblicz wartości μ_{P_1}(v), μ_{P_2}(v);
5:    if dla pewnego v ∈ V zachodzi μ_{P_1}(v) ≠ μ_{P_2}(v) then return NIE;
6:    Zbuduj graf H;
7:    foreach v in V do begin
8:      S := v;
9:      while deg_H(v) > 0 do begin
10:       w := sąsiad końca ścieżki S;
11:       Usuń użytą krawędź z grafu;
12:       S := S.append(w);
13:       if w zaznaczony jako odwiedzony then begin
14:           Przejrzyj S od końca, znajdując poprzednie wystąpienie w;
15:           Odetnij fragment późniejszy niż to wystąpienie;
16:           Odznacz odwiedzenie wierzchołków z fragmentu;
17:           Wypisz fragment jako kolejny cykl;
18:       end else Zaznacz w jako odwiedzony;
19:     end
20:   end
21: end
```

Oczywiście, skoro na wyjściu wpierw musimy wypisać liczbę uzyskanych cykli, to algorytm tak naprawdę musi wpisywać znajdowane cykle do pomocniczej tablicy, potem wyznaczyć ich liczbę i wypisać wszystkie naraz.

Przejdźmy do analizy złożoności. Linie 3–6 oczywiście możemy wykonać w czasie $O(n+m)$. Zgodnie z poprzednimi obserwacjami, w pętli rozpoczynającej się w linii 7 każda krawędź grafu jest dokładana do ścieżki S dokładnie raz oraz usuwana dokładnie raz. Oznacza to, że cała pętla działa w czasie $O(n+m)$. Stąd wniosek, że cały algorytm działa również w czasie $O(n+m)$.

Jak usuwać krawędzie?

Wnikliwy Czytelnik na pewno zauważył, że w opisie algorytmu prześlizgnęliśmy się nad kilkoma szczegółami implementacyjnymi. Najważniejszym z nich jest usuwanie krawędzi grafu.

Naiwna implementacja wymaga przejrzenia wszystkich krawędzi znajdujących się na liście sąsiedztwa jednego z końców usuwanej krawędzi, co prowadzi do rozwiązania zadania w czasie $O(n^2)$. Takie rozwiązania dostawały ok. 60 punktów, przykładowe zaimplementowano w plikach `smis1.cpp` i `smis3.pas`. Nieco lepszym pomysłem jest przechowywanie sąsiadów każdego wierzchołka w strukturze słownikowej (dowolne zrównoważone drzewo poszukiwań binarnych, np. kontener `set` z biblioteki STL w C++). Taki algorytm działa w czasie $O((n+m)\log n)$ i pozwalał zdobyć ponad 90 punktów.

Nas jednak interesuje rozwiązanie liniowe, a więc chcielibyśmy pojedynczą operację usunięcia krawędzi wykonać w czasie stałym. Można to uczynić na kilka sposobów. Jednym z nich jest trzymanie sąsiadów każdego wierzchołka na liście dwukierunkowej (o tej strukturze danych można dowiedzieć się więcej m.in. w książce [20] lub [22]). Jeśli mamy krawędź vw, to zarówno na liście wierzchołka v jak i wierzchołka w trzymamy informację o istnieniu krawędzi vw, jako pojedynczy element listy. Dodatkowo, przy tworzeniu tej krawędzi dbamy o zapamiętanie tzw. *dowiązań krzyżowych*: przy informacji o vw na liście v trzymamy wskaźnik na odpowiadający element na liście w i vice versa. Teraz, chcąc usunąć krawędź vw, mając ją daną jako element listy wierzchołka v, nie tylko umiemy usunąć ten element z listy v, lecz także dzięki dowiązaniu krzyżowemu wiemy, który element z listy w usunąć, bez potrzeby przeglądania całej listy wierzchołka w.

Istnieje jeszcze jedno warte uwagi asymptotycznie optymalne rozwiązanie. Krawędzie wychodzące z każdego wierzchołka przechowywane są w (dynamicznie alokowanej) tablicy[1]. Element tej tablicy zawiera trzy wartości — numer wierzchołka w będącego drugim końcem krawędzi, indeks, pod którym ta krawędź jest przechowywana w tablicy odpowiadającej wierzchołkowi w, a także flaga, która informuje, czy krawędź została usunięta.

Samo usuwanie jest proste, ponieważ przechowujemy coś na kształt dowiązań krzyżowych i tylko zaznaczamy odpowiednie flagi. Jednakże gdy chcemy znaleźć kolejną krawędź, którą możemy wyjść z wierzchołka, musimy przebijać się przez potencjalnie wiele „usuniętych" krawędzi. Zauważmy jednak, że gdybyśmy za każdym razem szukali wolnej krawędzi, sprawdzając wszystkie po kolei od początku tablicy, znajdowalibyśmy krawędzie o coraz większych indeksach w tej tablicy, ponieważ raz usunięta krawędź nigdy nie jest wstawiana ponownie. Można zatem za każdym razem zaczynać w miejscu, w którym zatrzymaliśmy się poprzednio. Polecamy Czytelnikowi zastanowić się, dlaczego to już wystarcza do osiągnięcia liniowej złożoności czasowej całego rozwiązania.

Przedstawione właśnie podejście zostało zaimplementowane w pliku `smi3.cpp`. W praktyce jest nieco oszczędniejsze zarówno pod względem zużycia czasu, jak i pamięci.

Rozwiązanie alternatywne – cykl Eulera

Czytelnik, który zapoznał się bliżej z pojęciem cyklu Eulera, zapewne zauważył jego silny związek z zadaniem. Przypomnijmy, że cyklem Eulera w grafie nazywamy taki cykl, który przechodzi przez każdą krawędź dokładnie raz (zwykle nie jest to cykl prosty). Oczywiście warunkiem koniecznym na istnienie cyklu Eulera jest, aby graf był krawędziowo spójny oraz aby każdy wierzchołek miał stopień parzysty: za każdym razem, gdy cykl wchodzi do wierzchołka, musi też wyjść. Okazuje się, że jest to również warunek wystarczający. Co więcej, mając dany spójny graf o parzystych stopniach wszystkich wierzchołków, cykl Eulera można znaleźć w czasie $O(n+m)$ (można o tym przeczytać m.in. w książce [27] lub [22]).

[1] W bibliotece STL używa się do tego kontenera `vector`, gdyż dzięki niemu można łatwiej (w szczególności jednoprzebiegowo) skonstruować taką reprezentację grafu.

Rozwiązanie alternatywne wygląda zatem następująco. Po upewnieniu się, że warunek konieczny na istnienie rozwiązania jest spełniony, i zbudowaniu grafu H, w każdej spójnej składowej H znajdujemy cykl Eulera. Teraz każdy z tych cykli trzeba rozłożyć na cykle proste — kursy śmieciarki. Robimy to podobnie jak w rozwiązaniu wzorcowym. Wziąwszy jeden z tych cykli, ustalamy wierzchołek początkowy i po kolei przeglądamy wierzchołki zgodnie z kolejnością na cyklu. Gdy któryś z nich się powtórzy, odcinamy fragment pomiędzy powtórzeniami i wypisujemy jako kolejny znaleziony cykl prosty. Postępujemy tak, aż wytniemy cały cykl Eulera.

Rozwiązanie to również działa w czasie $O(n+m)$ i tak naprawdę jest w pewnym sensie równoważne rozwiązaniu wzorcowemu. Zostało zaimplementowane w plikach `smi.cpp` oraz `smi2.pas`.

Testy

Rozwiązania zostały sprawdzone na 10 grupach testów, każda z nich obejmowała jeden lub dwa testy. W testach 7*b*, 8*b*, 9*b* i 10*b* graf H jest silnie niezrównoważonym grafem dwudzielnym. Takie grafy są bowiem szczególnie trudne dla rozwiązań używających nieoptymalnych algorytmów usuwania krawędzi. Wszystkie testy, dla których nie zaznaczono inaczej, mają odpowiedź pozytywną.

Nazwa	n	m	Opis
smi1.in	8	9	mały test stworzony ręcznie
smi2.in	20	23	mały test stworzony ręcznie
smi3a.in	500	1 500	test losowy
smi3b.in	500	1 500	test losowy z odpowiedzią negatywną
smi4a.in	1 200	5 000	test losowy
smi4b.in	1 200	5 000	test losowy z odpowiedzią negatywną
smi5a.in	200	3 405	test losowy
smi5b.in	200	3 468	test losowy z odpowiedzią negatywną
smi6a.in	2 000	9 921	test losowy
smi6b.in	1 000	9 522	test losowy z odpowiedzią negatywną
smi7a.in	10 000	150 000	test losowy
smi7b.in	100 000	1 000 000	graf dwudzielny
smi8a.in	50 000	550 000	test losowy
smi8b.in	100 000	1 000 000	graf dwudzielny
smi9a.in	75 000	850 000	test losowy
smi9b.in	100 000	999 900	graf dwudzielny
smi10a.in	100 000	1 000 000	test losowy
smi10b.in	100 000	999 900	graf dwudzielny