Mirosław Michalski	Mirosław Michalski, Jakub Łącki	Jakub Łącki
Treść zadania	Opracowanie	Program

Dostępna pamięć: 128 MB. *OI, Etap II, dzień pierwszy, 11.02.2009*

Przechadzka Bajtusia

Bajtuś jest jednym z najmłodszych mieszkańców Bajtogrodu. Dopiero niedawno nauczył się pisać i czytać. Jest jednak na tyle duży, że sam już chodzi do szkoły. Codziennie rano wychodzi z domu, a następnie wstępuje po kolei do wszystkich swoich kolegów; dopiero po dołączeniu się ostatniego kolegi cała grupa idzie na lekcje.

Pewnego dnia nauczyciel Bajtusia poprosił go o sporządzenie listy ulic, którymi Bajtuś przechodzi po drodze z domu do szkoły, i odczytanie tej listy na następnych zajęciach. Szybko jednak okazało się, że taka lista mogłaby pochłonąć ogromne ilości papieru. Ustalono więc, że Bajtuś zapisze jedynie pierwszą literę z nazwy każdej ulicy, po której przejdzie. Każda ulica w Bajtogrodzie jest jednokierunkowa i łączy dwa różne skrzyżowania.

Bajtuś podczas odczytywania opisu trasy robi przerwy jedynie w miejscach, w których wstępował do kolegów. Możemy więc traktować opis każdego fragmentu jego drogi z domu do szkoły jako jedno słowo. Chłopiec wciąż ma problemy z prawidłowym czytaniem, tj. czytaniem od lewej strony do prawej, i zamiast tego zdarza mu się czytać od strony prawej do lewej. Czasem przeczyta wyraz **mleko** *jako* **mleko**, *a czasem jako* **okelm**. *Rodzice Bajtusia wiedzą o tych problemach syna, więc postanowili mu pomóc i znaleźć taką trasę, by opis każdego fragmentu, niezależnie od tego, z której strony będzie czytany, brzmiał tak samo. Jednocześnie chcieliby, żeby długość każdego kawałka tej trasy była jak najmniejsza. Zwrócili się do Ciebie z prośbą o pomoc.*

Napisz program, który:

- *wczyta ze standardowego wejścia opis miasta,*
- *wyznaczy taką trasę z domu do szkoły, w której każdy fragment jest możliwie najkrótszy, a jego opis, niezależnie od kierunku czytania, brzmi dokładnie tak samo,*
- *wypisze wyznaczone opisy fragmentów drogi na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera dwie liczby całkowite n i m oddzielone pojedynczym odstępem ($2 \leqslant n \leqslant 400$, $1 \leqslant m \leqslant 60\,000$). Oznaczają one odpowiednio liczbę skrzyżowań w Bajtogrodzie oraz liczbę łączących je ulic. W kolejnych m wierszach znajdują się opisy ulic. W wierszu $(i+1)$-szym znajdują się trzy wartości x_i, y_i, c_i ($1 \leqslant x_i \leqslant n$, $1 \leqslant y_i \leqslant n$, $x_i \neq y_i$), pooddzielane pojedynczymi odstępami i oznaczające odpowiednio początek ulicy, koniec ulicy oraz pierwszą literę jej nazwy w postaci małej litery alfabetu angielskiego. Między dowolnymi dwoma skrzyżowaniami istnieje maksymalnie jedna ulica w danym kierunku. Kolejny wiersz zawiera jedną liczbę całkowitą d ($2 \leqslant d \leqslant 100$), oznaczającą liczbę skrzyżowań, pomiędzy którymi idzie Bajtuś w drodze do szkoły. Następny wiersz zawiera d liczb całkowitych s_i ($1 \leqslant s_i \leqslant n$), pooddzielanych pojedynczymi odstępami. Oznaczają one, że Bajtuś mieszka przy skrzyżowaniu numer s_1, szkoła znajduje się przy skrzyżowaniu s_d, zaś po drodze Bajtuś idzie **kolejno** *po kolegów mieszkających przy skrzyżowaniach $s_2, s_3, \ldots, s_{d-1}$. Każde dwa następujące po sobie*

numery skrzyżowań na liście są różne. Może się jednak zdarzyć, że pewne numery skrzyżowań na liście będą takie same. Ponadto, jeśli pomiędzy dwoma kolejnymi skrzyżowaniami nie da się przejść, stosując się do ograniczeń podanych w zadaniu, to Bajtuś idzie na przełaj i niczego nie zapisuje na kartce.

Wyjście

Wyjście powinno składać się z $d-1$ wierszy. W i-tym wierszu powinna znajdować się liczba r_i oraz ciąg znaków w_i, oddzielone pojedynczym odstępem. Liczba r_i oznacza długość najkrótszej drogi, spełniającej wymogi zadania, łączącej skrzyżowania s_i oraz s_{i+1}, zaś w_i to ciąg pierwszych liter nazw ulic na tej drodze. W przypadku gdy pomiędzy danymi dwoma skrzyżowaniami nie istnieje trasa spełniająca warunki zadania, należy wypisać -1. Jeśli zaś istnieje kilka możliwych ciągów znaków w_i zgodnych z warunkami zadania, należy wypisać dowolny z nich.

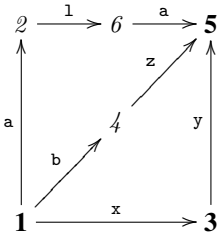
Przykład

Dla danych wejściowych:

```
6 7
1 2 a
1 3 x
1 4 b
2 6 l
3 5 y
4 5 z
6 5 a
3
1 5 3
```

poprawnym wynikiem jest:

```
3 ala
-1
```



Rozwiązanie

Podejście grafowe

Postawiony w zadaniu problem można w naturalny sposób opisać za pomocą grafu, w którym wierzchołkami są skrzyżowania w Bajtogrodzie. Krawędzie w tym grafie są skierowane, łączą pary różnych skrzyżowań i posiadają etykiety oznaczające pierwsze litery nazw

poszczególnych ulic. Oznaczmy tak skonstruowany graf przez $G = (V, E)$ — zgodnie z treścią zadania ma on n wierzchołków i m krawędzi. Naszym zadaniem jest (wielokrotne) wyszukiwanie w grafie G takich ścieżek, by litery na kolejnych krawędziach ścieżki tworzyły palindrom[1].

W pierwszej wersji algorytmu będziemy takie wyszukiwanie wykonywać osobno dla każdego fragmentu trasy Bajtusia. Zauważmy, że jeśli z pierwszego wierzchołka takiego fragmentu wychodzimy krawędzią o etykiecie c, to do ostatniego wierzchołka tego fragmentu musimy również wejść krawędzią etykietowaną tą literą. To rodzi następujący pomysł: wykonujmy jednocześnie ruch z wierzchołka początkowego do końcowego oraz z końcowego w kierunku początku (idąc *pod prąd*), po krawędziach o tej samej etykiecie. Jeśli w którymś momencie oba przemieszczające się wskaźniki spotkają się, będziemy wiedzieć, że znaleźliśmy szukany palindrom.

Mówiąc bardziej formalnie, niech $G' = (V', E')$ będzie grafem par wierzchołków grafu G. Pomiędzy wierzchołkami $(a_1, b_1),\ (a_2, b_2) \in V'$ dodajemy krawędź skierowaną o etykiecie c, jeśli w grafie G istnieją krawędzie $a_1 \xrightarrow{c} a_2$ oraz $b_2 \xrightarrow{c} b_1$ o etykietach c. W tak zdefiniowanym grafie ścieżka $(a_1, b_1) \to (a_2, b_2) \to \ldots \to (a_k, b_k)$ odpowiada sytuacji, w której początkowymi wierzchołkami na trasie Bajtusia są $a_1, a_2, \ldots, a_k$, końcowymi — wierzchołki: $b_k, b_{k-1}, \ldots, b_1$, i dodatkowo wiemy, że pierwsze $k-1$ liter z opisu drogi jest równe ostatnim $k-1$ literom, w odwróconej kolejności.

Aby znaleźć drogę biegnącą w grafie G z wierzchołka *start* do wierzchołka *meta*, której opis jest palindromem parzystej długości, wystarczy w G' znaleźć ścieżkę postaci:

$$(start, meta) \to (a_2, b_2) \to \ldots \to (v, v)$$

dla pewnego $v \in V$. Natomiast trasie w G, którą da się opisać palindromem nieparzystej długości, w G' odpowiada ścieżka postaci:

$$(start, meta) \to (a_2, b_2) \to \ldots \to (v, w),$$

przy czym w grafie G musi istnieć krawędź $v \to w$. Wierzchołki postaci (v, v) oraz takie wierzchołki (v, w), że w G istnieje krawędź $v \to w$, będziemy odtąd nazywać *wierzchołkami końcowymi* grafu G'.

Ponieważ każdorazowo szukamy trasy o najmniejszej możliwej długości, do wyszukiwania ścieżek możemy użyć algorytmu przeszukiwania grafu wszerz, czyli BFS[2]. Przypomnijmy, że algorytm ten pozwala na znalezienie nie tylko długości najkrótszej ścieżki, ale także, na podstawie tablicy przodków (przodkiem wierzchołka v nazywamy wierzchołek, z którego do v przyszliśmy), na odtworzenie samej tej ścieżki, co jest wymagane w zadaniu. Pozostaje obliczyć rozmiar grafu G', co pozwoli nam ocenić złożoność algorytmu. Każdy wierzchołek G' powstaje z pary wierzchołków grafu G, więc sumaryczna liczba wierzchołków w G' to n^2. Podobnie, z każdej pary krawędzi w G o takiej samej etykiecie powstaje jedna krawędź w G', więc pesymistycznie — tj. w przypadku, gdy dużo krawędzi ma takie same etykiety — liczba krawędzi w G' może być rzędu m^2. Ponieważ dla każdego fragmentu podróży Bajtusia musimy wykonać od nowa przeszukiwanie grafu, więc złożoność czasowa takiego algorytmu to $O(d \cdot (n^2 + m^2))$, zaś pamięciowa — $O(n^2 + m^2)$. Jest on zaimplementowany w pliku `przb2.cpp` — takie rozwiązanie zdobywało na zawodach 40% punktów.

[1] *Palindrom* — słowo wyglądające tak samo czytane zarówno normalnie, jak i od tyłu, np. `kajak`.

[2] Opis algorytmu BFS można znaleźć w większości książek poświęconych algorytmice, np. [20].

Złożoność pamięciową można w łatwy sposób poprawić do $O(n^2+m) = O(n^2)$, jeśli zauważymy, że nie musimy trzymać w pamięci całego grafu — wszystkie potrzebne krawędzie możemy generować na bieżąco na podstawie tych z grafu G (składnik n^2 w oszacowaniu złożoności wynika z rozmiaru struktur danych wykorzystywanych w algorytmie BFS). Implementacje tak poprawionego algorytmu można znaleźć w plikach `przs3.cpp`, `przs13.pas` oraz `przs23.java`; takie rozwiązania zdobywały na zawodach już 50% punktów.

Odwracamy krawędzie

Dla danych o maksymalnych rozmiarach powyższe rozwiązanie jest zdecydowanie zbyt wolne. Jego koncepcja jest jednak słuszna — wystarczy kilka usprawnień, aby dojść do rozwiązania wzorcowego. W pierwszej kolejności zmniejszymy wpływ czynnika d na złożoność algorytmu. Każdorazowe uruchomienie algorytmu BFS jest dosyć podobne: zawsze szukamy najkrótszej ścieżki ze wskazanego wierzchołka do jednego z wierzchołków końcowych. Co ważne, zbiór wierzchołków końcowych jest zawsze taki sam. Odwróćmy więc krawędzie naszego grafu i rozpocznijmy przeszukiwanie wszerz od wierzchołków końcowych. Wystarczy jedynie wstawić je wszystkie do kolejki na początku przeszukiwania BFS, a wyznaczymy odległości od każdego wierzchołka w grafie do wierzchołka końcowego (w grafie nieodwróconym)[3]. Ponadto, jak łatwo zauważyć, także w tym przypadku na podstawie tablicy przodków obliczonej podczas przeszukiwania można skonstruować wszystkie wynikowe palindromy. Takie rozwiązanie działa w złożoności czasowej $O(dn^2+m^2)$, gdyż rozmiar każdego z d palindromów jest w najgorszym przypadku rzędu $O(n^2)$. Skąd to wiadomo? Trzeba zauważyć, że wynik jest najkrótszą ścieżką (oczywiście w przypadku, gdy takowa istnieje) pomiędzy pewnymi dwoma wierzchołkami w grafie o n^2 wierzchołkach. Takie rozwiązanie, zaimplementowane w plikach `przs2.cpp`, `przs12.pas` oraz `przs22.java`, na zawodach zdobywało 70% punktów.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe opiera się na pomyśle podobnym do przedstawionego powyżej, jednak realizuje go nieco efektywniej. W tym celu stosuje sztuczkę, która została zaproponowana w rozwiązaniu zadania *Agenci* z VII Olimpiady Informatycznej (więcej w [7]). Zauważmy, że w grafie G' przejście po jednej krawędzi odpowiada dwóm przejściom w grafie G: wykonujemy krok naprzód z początku oraz krok wstecz z końca drogi. Nieco zaskakujący może wydawać się fakt, że jeśli kroki te będziemy wykonywać osobno, uzyskamy algorytm o istotnie lepszym czasie działania.

Skonstruujmy graf $G'' = (V'', E'')$, który będzie modelował takie podejście. Wierzchołkami G'' są trójki (u,v,c), przy czym $u,v \in V$, natomiast $c \in \{\mathtt{a},\mathtt{b},\ldots,\mathtt{z},\#\} \stackrel{\text{def}}{=} A$. Trójka

[3]Przeszukiwanie rozpoczęte z wielu wierzchołków naraz można wyobrażać sobie tak, że dokładamy do grafu jedno sztuczne źródło, które łączymy krawędziami ze wszystkimi wierzchołkami końcowymi, i to od niego rozpoczynamy przeszukiwanie. Wówczas tuż po przetworzeniu owego źródła kolejka wykorzystywana w algorytmie będzie zawierała właśnie wszystkie wierzchołki końcowe. Taka interpretacja stanowi więc zarazem uzasadnienie poprawności zmodyfikowanego algorytmu BFS.

$(u,v,\#)$ odpowiada „zwykłemu" wierzchołkowi $(u,v) \in V'$; w szczególności, taki wierzchołek nazwiemy wierzchołkiem końcowym grafu G'', jeżeli (u,v) był wierzchołkiem końcowym grafu G'. Wszystkie pozostałe wierzchołki grafu G'', tj. wierzchołki postaci (u,v,c) dla $c \neq \#$, reprezentują pozycje pośrednie, osiągane za pomocą pojedynczych kroków z tych „zwykłych" wierzchołków.

Dokładniej, osiągnięcie wierzchołka $(u,v,\#)$ w przeszukiwaniu z wierzchołków końcowych będzie odpowiadało znalezieniu ścieżki z u do v w grafie G, której opis jest palindromem (czyli dokładnie tak samo, jak w przypadku wierzchołka (u,v) w G'). Natomiast trójka (u,v,c) dla $c \in \{\mathtt{a},\mathtt{b},\ldots,\mathtt{z}\}$, napotkana podczas przeszukiwania, będzie oznaczać znalezienie takiej ścieżki z u do v w G, która po doklejeniu na samym początku litery c staje się palindromem. W tym celu, krawędzie w G'' są zdefiniowane następująco:

$$\begin{aligned} (u,v,\#) &\longrightarrow (u',v,c) \qquad \text{jeżeli w } G \text{ jest krawędź } u \xrightarrow{c} u' \\ (u,v,c) &\longrightarrow (u,v',\#) \qquad \text{jeżeli w } G \text{ jest krawędź } v' \xrightarrow{c} v. \end{aligned}$$

Jeśli przez $|A|$ oznaczymy rozmiar alfabetu A, to tak skonstruowany graf ma, jak łatwo sprawdzić, $O(n^2 \cdot |A|)$ wierzchołków oraz $O(nm)$ krawędzi. W rozwiązaniu wzorcowym wszystkie kroki wcześniej opisanego rozwiązania wykonujemy na grafie G'' zamiast G', tzn.: odwracamy krawędzie G'', wykonujemy przeszukiwanie BFS z wierzchołków końcowych i odczytujemy wyniki z wierzchołków postaci $(start, meta, \#)$. Otrzymany w ten sposób algorytm działa w czasie $O\big(n^2 \cdot (|A|+d) + mn\big)$ i pamięci rzędu $O(n^2 \cdot |A|)$, ponieważ krawędzi G'', podobnie jak w przypadku G', nie musimy pamiętać bezpośrednio, tylko konstruujemy je na bieżąco na podstawie krawędzi G. Implementacje rozwiązania wzorcowego można znaleźć w plikach `prz.cpp`, `prz1.pas` oraz `prz2.java`.

Testy

Do oceny rozwiązań zawodników użyto 10 zestawów testowych. Przygotowane testy można podzielić na pięć kategorii:

Graf z losowymi palindromami Generujemy pewną liczbę losowych palindromów, a następnie dla każdego palindromu dodajemy do grafu losowo umiejscowioną ścieżkę o etykietach zgodnych z kolejnymi literami palindromu.

Graf z długim wynikiem Łączymy wierzchołek 1 z wierzchołkiem 2 krawędzią o etykiecie `a` oraz wierzchołek 2 z wierzchołkiem 3 krawędzią o etykiecie `z`. Dodatkowo wierzchołki 2 i 3 wpinamy w cykle długości odpowiednio c_1 i c_2, w których wszystkie krawędzie mają etykietę `a`. Na koniec dokładamy trochę losowych krawędzi z drugiego cyklu do pierwszego. W takim grafie, jeśli $\mathrm{NWD}(c_1,c_2) = 1$, najkrótszy palindrom z 1 do 3 ma długość zbliżoną do $c_1 \cdot c_2$.

Graf z cyklem Cykl złożony z krawędzi o ustalonej etykiecie, z dodanymi losowymi krawędziami.

Graf gęsty Dwa bardzo gęste losowe grafy połączone jedną krawędzią w każdą stronę.

Graf długi Graf, w którym dodajemy losowe krawędzie, ale tylko między wierzchołkami, których indeksy różnią się co najwyżej o ustaloną wartość.

W poniższej tabelce znajdują się opisy poszczególnych testów, przy czym n oznacza liczbę skrzyżowań, m — liczbę ulic, a d — liczbę punktów występujących na trasie Bajtusia z domu do szkoły.

Nazwa	n	m	d	Opis
prz1.in	5	12	4	graf z losowymi palindromami
prz2.in	7	12	25	graf z losowymi palindromami
prz3.in	60	244	40	graf z losowymi palindromami
prz4.in	100	200	100	graf z cyklem
prz5a.in	87	315	30	graf długi
prz5b.in	104	4030	96	graf gęsty
prz6a.in	200	10408	41	graf z losowymi palindromami
prz6b.in	277	3031	9	graf z długim wynikiem
prz7a.in	398	11500	5	graf długi
prz7b.in	400	400	3	graf z cyklem
prz8a.in	399	53472	94	graf długi
prz8b.in	378	13532	94	graf z losowymi palindromami
prz9a.in	400	401	100	graf z długim wynikiem
prz9b.in	398	60000	97	graf gęsty
prz10a.in	394	12532	94	graf długi
prz10b.in	400	60000	97	graf gęsty