Wojciech Rytter, Bartosz Szreder — Treść zadania, Opracowanie
Dawid Dąbrowski — Program

Dostępna pamięć: 128 MB. *OI, Etap III, dzień pierwszy, 06.04.2011*

Inspekcja

Sieć kolejowa Bajtockich Kolei Bitowych (BKB) składa się z dwukierunkowych odcinków torów łączących pewne pary stacji. Każda para stacji jest połączona co najwyżej jednym odcinkiem torów. Ponadto wiadomo, że z każdej stacji kolejowej można dojechać do każdej innej dokładnie jedną trasą. (Trasa może być złożona z kilku odcinków torów, ale nigdy nie przechodzi przez żadną stację więcej niż raz).

Bajtazar jest tajnym inspektorem BKB. W celu przeprowadzenia inspekcji wybiera jedną ze stacji (oznaczmy ją S), w której organizuje sobie centralę, i rozpoczyna podróż po wszystkich innych stacjach. Podróż ma następujący przebieg:

- *Bajtazar zaczyna na stacji S.*
- *Następnie wybiera jedną ze stacji jeszcze nie skontrolowanych i przemieszcza się do niej po najkrótszej ścieżce, dokonuje tam inspekcji i wraca do S.*
- *Nieuczciwi pracownicy BKB ostrzegają się nawzajem o przyjeździe Bajtazara. Aby ich zmylić, następną stację do skontrolowania Bajtazar wybiera w taki sposób, aby wyjechać w inną stronę ze stacji S niż poprzednio, tzn. innym odcinkiem torów wychodzącym z S.*
- *Każda stacja (poza stacją S) jest kontrolowana dokładnie raz.*
- *Po skontrolowaniu ostatniej stacji Bajtazar* **nie wraca** *do S.*

Przejazd każdym odcinkiem torów trwa tyle samo — jedną godzinę.

Bajtazar pragnie rozważyć wszystkie możliwe stacje jako stacje początkowe S. Dla każdej z nich chcemy wyznaczyć kolejność, w jakiej Bajtazar powinien kontrolować pozostałe stacje, tak aby łącznie jak najmniej czasu spędził na przejazdach, o ile dla danej stacji S w ogóle taka kolejność istnieje.

Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 1\,000\,000$), oznaczającą liczbę stacji. Stacje są ponumerowane od 1 do n. Kolejne $n-1$ wierszy opisuje odcinki torów, po jednym w wierszu. W każdym z nich znajdują się dwie liczby całkowite a, b ($1 \leqslant a, b \leqslant n$, $a \neq b$), oddzielone pojedynczym odstępem, oznaczające, że istnieje odcinek torów łączący stacje a i b. Każdy odcinek torów pojawia się w opisie dokładnie raz.

W testach wartych przynajmniej 30% punktów zachodzi dodatkowy warunek $n \leqslant 2\,000$.

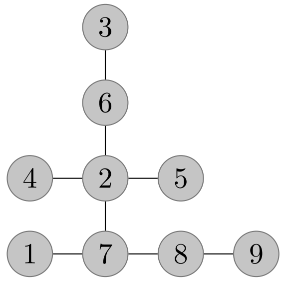
Wyjście

Twój program powinien wypisać na standardowe wyjście n wierszy, a w każdym z nich po jednej liczbie całkowitej. Liczba w i-tym wierszu powinna być równa minimalnej liczbie godzin, jakie Bajtazar musi spędzić na przejazdach, aby skontrolować stacje, dla $S = i$ — o ile dla $S = i$ szukana kolejność stacji istnieje. W przeciwnym przypadku, w i-tym wierszu powinna znaleźć się liczba -1.

Przykład

Dla danych wejściowych:

```
9
3 6
2 4
2 6
2 5
1 7
2 7
8 9
7 8
```



poprawnym wynikiem jest:

```
-1
23
-1
-1
-1
-1
-1
-1
-1
```

Rysunek pokazuje przykładową sieć połączeń. Szukana kolejność, w jakiej Bajtazar powinien kontrolować stacje, istnieje tylko dla $S = 2$. Może to być na przykład: 7, 4, 8, 6, 1, 5, 3, 9. Przy takiej kolejności kontrolowanych stacji Bajtazar spędzi na przejazdach łącznie 23 godziny.

Rozwiązanie

Wprowadzenie

Przetłumaczmy najpierw treść zadania na język matematyki. Dane jest *drzewo* T o n wierzchołkach, czyli spójny n-wierzchołkowy graf nieskierowany niezawierający żadnych cykli. Definiujemy *przechadzkę* po drzewie T rozpoczynającą się w wierzchołku S jako taką permutację $P = (v_1, v_2, \ldots, v_{n-1})$ pozostałych wierzchołków drzewa, że dla każdych dwóch kolejnych wierzchołków jedyna łącząca je ścieżka przechodzi przez S. Długością przechadzki P nazywamy liczbę

$$w(P) := d(S, v_1) + d(v_1, S) + d(S, v_2) + d(v_2, S) + \ldots + d(S, v_{n-1}).$$

Przez $d(a, b)$ oznaczamy tu odległość między wierzchołkami a i b w drzewie T (odległość, czyli długość najkrótszej, a w wypadku drzewa jedynej, ścieżki). Zwróćmy uwagę, że w definicji nie ma składnika $d(v_{n-1}, S)$.

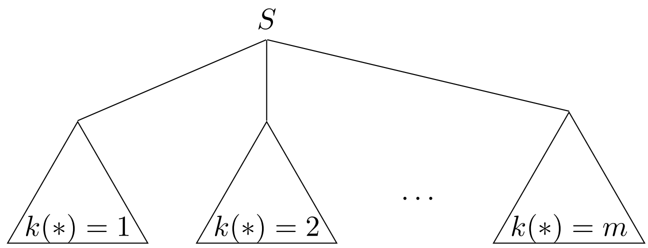
Celem zadania jest stwierdzenie, dla każdego wierzchołka w drzewie T, czy istnieje przechadzka rozpoczynająca się w tym wierzchołku, a jeśli tak, to jaka jest jej minimalna długość. Ograniczenia zadania wymagają rozwiązania działającego w złożoności czasowej liniowej lub ewentualnie nieznacznie gorszej.

Krok po kroku do rozwiązania wzorcowego

Analizujemy prostszy problem, czyli ustalamy S

Rozwiązanie wzorcowe opiszemy, rozpoczynając od przeanalizowania prostszego problemu. Załóżmy, że mamy jako początek przechadzki rozważyć tylko jeden, ustalony wierzchołek S, a przy tym chcemy tylko stwierdzić, czy taka przechadzka istnieje, bez podawania minimalnej długości.

Wierzchołki do odwiedzenia tworzą zbiór $V_S = \{1, \ldots, n\} \setminus \{S\}$. Niech m oznacza liczbę sąsiadów wierzchołka S, ponumerujmy ich liczbami od 1 do m. Niech wreszcie $k(x)$ będzie (dla $x \in V_S$) numerem sąsiada wierzchołka S, przez którego prowadzi ścieżka z S do x, patrz rys. 1.



Rys. 1: Ilustracja drzewa ukorzenionego w S oraz przyporządkowania k.

Weźmy multizbiór (czyli zbiór elementów wraz z dodatnią *krotnością* występowania każdego z nich) $C = \{k(x) : x \in V_S\}$. Przechadzka istnieje wtedy i tylko wtedy, gdy elementy multizbioru C można ustawić w ciąg, tak aby dwie równe liczby nigdy nie stały na sąsiednich pozycjach (takie uporządkowanie nazwiemy *prawidłowym*). Permutacja $(v_1, \ldots, v_{n-1})$ zbioru V_S jest bowiem przechadzką dokładnie wtedy, gdy ciąg $k(v_1), \ldots, k(v_{n-1})$ jest prawidłowym uporządkowaniem C.

Aby zwięźle opisać, kiedy multizbiór da się prawidłowo uporządkować, wprowadźmy pojęcie *lidera*, czyli elementu o maksymalnej krotności. Jeśli istnieje kilka różnych elementów występujących największą liczbę razy, każdy z nich jest liderem.

Twierdzenie 1. *Jeśli $n \geqslant 2$, $A = \{a_1, a_2, \ldots, a_{n-1}\}$ jest multizbiorem liczb, zaś N liczbą wystąpień lidera w tym ciągu, to A da się prawidłowo uporządkować wtedy i tylko wtedy, gdy $2N \leqslant n$.*

Jeśli zachodzi ostra nierówność, owo prawidłowe uporządkowanie może mieć jako ostatni wyraz dowolny element $a \in A$. W przeciwnym razie ostatni wyraz musi być liderem.

Dowód: Na razie wykażemy indukcyjnie pozytywną część twierdzenia, tzn. istnienie odpowiednich ciągów. Dla $n = 2$ teza jest jasna.

Niech więc $n \geqslant 3$. Udowodnimy tezę dla n, korzystając z jej prawdziwości dla $n' = n - 1$. Jeśli $2N < n$, niech $a \in A$ będzie dowolne. Wówczas multizbiór

$A' = A \setminus \{a\}$ ma $n' - 1$ elementów, zaś jego lider występuje $N' \leqslant N$ razy. Wobec tego $2N' \leqslant 2N \leqslant n - 1 = n'$, a więc A' da się prawidłowo uporządkować.

Przy tym, jeśli a był liderem A, to albo $N' < N$ i uporządkowanie A' może mieć dowolny ostatni wyraz, albo $N' = N$, lecz a nie jest liderem A'. Jeśli zaś a nie był liderem A, to nie jest też liderem A' i $N' = N$. W obydwu przypadkach można prawidłowo uporządkować A', tak aby ostatni wyraz był różny od a, co pozwala stworzyć prawidłowe uporządkowanie A kończące się na a (które wybraliśmy jako dowolny element A).

Jeśli $2N = n$, niech a będzie liderem. Łatwo widać, że jest to wówczas jedyny lider, czyli multizbiór $A' = A \setminus \{a\}$ ma $n' - 1$ elementów, zaś jego lider występuje $N' = N - 1$ razy. Stąd $2N' < 2N - 1 = n - 1 = n'$, czyli A' można prawidłowo uporządkować tak, aby ostatni wyraz był dowolny, w szczególności różny od a. Zatem istnieje prawidłowe uporządkowanie A, którego ostatnim wyrazem jest lider.

Przejdźmy teraz do dowodu części negatywnej. Niech ℓ będzie (dowolnym) liderem A. Rozważmy hipotetyczne prawidłowe uporządkowanie A. Po każdym wystąpieniu ℓ, być może z wyjątkiem jednego, na ostatniej pozycji, musi być inny wyraz ciągu. Stąd liczba wszystkich wyrazów (czyli $n - 1$) musi wynosić przynajmniej $N + N - 1$, czyli $2N \leqslant n$. Jeśli zaś ℓ nie występuje na ostatniej pozycji, to przynajmniej $N + N$, skąd $2N < n$. ■

Mając tak udowodnione twierdzenie, możemy śmiało stwierdzić, że przechadzka z wierzchołka S istnieje wtedy i tylko wtedy, gdy rozmiar największego poddrzewa zaczepionego w pewnym synu S nie przekracza $\frac{n}{2}$.

Możemy ukorzenić graf w wierzchołku S i przeszukać go (np. w głąb), licząc wielkości poddrzew zaczepionych we wszystkich wierzchołkach w czasie $O(n)$, a następnie sprawdzić, czy największe z nich spełnia podane ograniczenie.

Utrudniamy problem – zbliżamy się do rozwiązania

Rozwiązaliśmy prostszy problem, teraz pora go utrudnić. Zapytajmy o minimalną możliwą długość przechadzki przy ustalonym S i założeniu, że taka przechadzka istnieje. Spójrzmy na definicję długości przechadzki. Możemy ją zapisać prościej:

$$w(P) = 2D - d(S, v_{n-1}),$$

gdzie D jest sumą odległości od wierzchołka S do pozostałych wierzchołków drzewa.

Zauważmy, że dla ustalonego punktu początkowego przechadzki wartość D jest stała. Widzimy więc, że chcielibyśmy skończyć przechadzkę w wierzchołku położonym jak najdalej od S.

Zastanówmy się, w jakich wierzchołkach możemy skończyć przechadzkę. Z pomocą przychodzi udowodnione twierdzenie, które w naszym języku mówi, że ostatni wierzchołek przechadzki może być dowolny, o ile tylko nie zachodzi równość w ograniczeniu. W przeciwnym przypadku jesteśmy zobligowani skończyć w największym poddrzewie.

Możemy więc sprawdzić, która z sytuacji ma miejsce, uruchomić ponownie przeszukiwanie grafu i obliczyć odległość od S do najdalszego wierzchołka w drzewie bądź do najdalszego wierzchołka w największym poddrzewie.

Ten algorytm działa w czasie $O(n)$, a zatem na jego podstawie jesteśmy w stanie rozwiązać całe zadanie w czasie $O(n^2)$ (po kolei ustalając $S = 1, 2, \ldots, n$). Takie rozwiązanie zdobywało około 30 punktów. Jego implementacja znajduje się w plikach `inss0.cpp` oraz `inss1.pas`.

Rozwiązania wzorcowe

Przyspieszamy rozwiązanie nieoptymalne

Rozwiązanie wzorcowe opiera się na tych samych spostrzeżeniach, jednak istotnie szybciej oblicza dla każdego wierzchołka wielkości poddrzew oraz najdalsze wierzchołki w odpowiednich poddrzewach.

Ukorzeńmy drzewo w dowolnym wybranym wierzchołku, np. tym o numerze 1. Na początek obliczymy dla każdego wierzchołka rozmiary poddrzew.

Możemy rekurencyjnie wyznaczyć dla każdego u rozmiary poddrzew zaczepionych w *synach* u (według hierarchii ustalonej względem wierzchołka wybranego na samym początku). Najniższym poziomem rekursji będzie liść drzewa, który nie ma żadnych synów.

Dla każdego wierzchołka u zostaje zatem do wyznaczenia tylko rozmiar poddrzewa, które, gdy spojrzymy od strony u, jest zaczepione w rodzicu u. Jeśli suma rozmiarów poddrzew zaczepionych w synach u wynosi $t(u)$, szukany rozmiar to oczywiście $n-t(u)-1$. Daje to liniowy algorytm obliczania rozmiarów poddrzew wszystkich wierzchołków w drzewie.

Teraz zajmijmy się wyznaczaniem najdalszego wierzchołka dla każdego z wierzchołków drzewa. Podzielimy tę procedurę na dwie fazy. W pierwszej znajdziemy najdalszego *potomka* każdego wierzchołka. Można to zrealizować w czasie $O(n)$ dla całego drzewa, w podobny sposób, w jaki wyznaczaliśmy wielkości poddrzew. Oznaczmy najdalszego potomka wierzchołka u przez $A(u)$.

Najbardziej odległy punkt od danego wierzchołka u nie musi być jednak jego potomkiem. W tym przypadku ścieżka od u do tego wierzchołka składa się z kilku krawędzi w górę drzewa (co najmniej jednej), a następnie z pewnej ścieżki w dół drzewa, rozłącznej krawędziowo ze ścieżką w górę drzewa. Taki najdalszy wierzchołek oznaczmy przez $B(u)$.

Uruchomimy drugie przeszukiwanie w głąb, które będzie korzystało z wyników pierwszego. Dla korzenia znamy wynik, gdyż nie ma wierzchołków wyżej od niego. Załóżmy teraz, że jesteśmy w pewnym wierzchołku u oraz że znamy wynik dla niego, natomiast chcemy poznać wynik dla pewnego syna v. Możliwe są dwa przypadki.

1. Ścieżka z v do $B(v)$ biegnie przez u dalej *w górę*. W tym wypadku $B(v) = B(u)$.

2. Ścieżka z v do $B(v)$ prowadzi krawędzią z v do u, a następnie *w dół* z u do $B(v)$. W tym wypadku $B(v) = A(u)$, chyba że v oraz $A(u)$ leżą w tym samym poddrzewie, wówczas musimy sprawdzić wierzchołki $A(w)$ dla w będących synami u różnymi od v i wybrać najgłębszy z nich. Oczywiście, taką bardziej kosztowną operację trzeba wykonać tylko dla dokładnie jednego z synów u.

Korzystając z tych spostrzeżeń, wartości B możemy policzyć w czasie $O(n)$ dla wszystkich wierzchołków drzewa, o ile będziemy także pamiętali odległości każdego wierzchołka u od wierzchołków $A(u)$ i $B(u)$.

Brakuje nam jeszcze tylko wartości $D(u)$, czyli sum odległości. Oznaczmy przez T_u drzewo ukorzenione w wierzchołku u. W przeszukiwaniu w głąb możemy posłużyć się następującym spostrzeżeniem: przechodząc krawędzią między wierzchołkami u oraz v, oddalamy się o 1 od wszystkich wierzchołków z poddrzewa u w drzewie T_v oraz przybliżamy o 1 do wszystkich z poddrzewa v w drzewie T_u. Możemy więc dla wierzchołka numer 1 obliczyć sumę odległości prostym przeszukiwaniem drzewa, a następnie uruchomić kolejne przeszukiwanie, które będzie, korzystając z powyższego spostrzeżenia, liczyć wartości $D(u)$ dla wszystkich wierzchołków grafu w czasie $O(n)$.

Mając tak policzone wartości dla wierzchołków drzewa, możemy prosto odpowiadać na pytania o istnienie przechadzki oraz jej minimalną długość. Rozwiązanie to zostało zaimplementowane w plikach `ins.cpp` oraz `ins2.pas`.

Po co się męczyć?

Uruchamiając rozwiązanie (chociażby siłowe) na różnych testach, można dostrzec pewną prawidłowość w generowanych odpowiedziach. Zazwyczaj tylko jeden wierzchołek drzewa spełnia nierówność przedstawioną we wcześniejszym twierdzeniu, a w niektórych przypadkach są dwa takie wierzchołki. Okazuje się, że dzieje się tak nieprzypadkowo.

Twierdzenie 2. *Niech T będzie dowolnym drzewem. Każde dwa wierzchołki T, które są początkami pewnych przechadzek, są sąsiadami. W szczególności, przechadzka w T istnieje dla co najwyżej dwóch wierzchołków początkowych.*

Dowód: Niech u i v będą dwoma różnymi wierzchołkami T, które stanowią początki pewnych przechadzek.

Niech T_1 oznacza zbiór wierzchołków, z których ścieżka do u *nie* wiedzie przez v. Zauważmy, że po ukorzenieniu drzewa w wierzchołku v zbiór T_1 stanowi zbiór wierzchołków poddrzewa zaczepionego w u. Wobec tego $|T_1| \leqslant \frac{n}{2}$. Analogicznie niech T_2 będzie zbiorem wierzchołków, z których ścieżka do v nie wiedzie przez u. Podobnie jak poprzednio, mamy $|T_2| \leqslant \frac{n}{2}$.

Zauważmy, że w drzewie nie jest możliwe, aby z pewnego wierzchołka ścieżka do u prowadziła przez v, a ścieżka do v przez u. Oznacza to, że każdy wierzchołek drzewa T leży w przynajmniej jednym ze zbiorów T_1, T_2. Ponieważ oba liczą po co najwyżej $\frac{n}{2}$ elementów, więc musi zachodzić $|T_1| = \frac{n}{2}$ i $|T_2| = \frac{n}{2}$, a ich część wspólna musi być pusta. Do tej części wspólnej należą jednak wszystkie, poza końcami, wierzchołki ścieżki z u do v. Stąd wniosek, że u i v to sąsiedzi.

Każdy co najmniej trójelementowy zbiór wierzchołków drzewa zawiera parę niesąsiadujących wierzchołków, więc druga część twierdzenia wynika bezpośrednio z pierwszej. ■

Mając takie twierdzenie, widzimy, że dla wszystkich wierzchołków wystarczy wyliczyć jedynie wielkości poddrzew. Jeśli na podstawie tych rozmiarów stwierdzimy, że że dla pewnego wierzchołka u istnieje przechadzka, możemy uruchomić przeszukiwanie w głąb w drzewie w nim ukorzenionym i łatwo policzyć potrzebne wartości $A(u)$, $B(u)$ i $D(u)$. Bardzo upraszcza to strukturę programu, o czym można się przekonać, spoglądając na rozwiązania zaimplementowane w plikach `ins1.cpp`, `ins3.pas` oraz `ins4.cpp`.

Testy

Większość zestawów składała się z testów należących do czterech następujących kategorii:

(a) Testy obalające rozwiązania korzystające z błędnych warunków na istnienie przechadzki. Składają się z korzenia, poddrzewa o wielkości $\frac{n}{2}$ oraz losowej reszty grafu.

(b) Testy obalające rozwiązanie zakładające, że odpowiedź dla jednego tylko wierzchołka jest niezerowa. Składają się z dwóch wierzchołków połączonych krawędzią oraz dwóch poddrzew tej samej wielkości doczepionych do każdego z nich.

(c) Testy obalające rozwiązanie zakładające, że zawsze kończymy w najgłębszym wierzchołku. Składają się z płytkiego poddrzewa o wielkości $\frac{n}{2}$ oraz długiej ścieżki w drugim poddrzewie.

(d) Losowo generowane drzewa, w których najdłuższe ścieżki są rzędu $\sqrt{n}$.

Nazwa	**n**	**Opis**
ins1[abcd].in	20	testy typu (a)-(d)
ins1e.in	1	przypadek brzegowy
ins1f.in	2	przypadek brzegowy
ins2[abcd].in	80	testy typu (a)-(d)
ins3[abcd].in	2 000	testy typu (a)-(d)
ins4[abcd].in	20 000	testy typu (a)-(d)
ins5[abcd].in	39 514	testy typu (a)-(d)
ins6[abcd].in	100 000	testy typu (a)-(d)
ins7[abcd].in	500 000	testy typu (a)-(d)
ins8[abcd].in	1 000 000	testy typu (a)-(d)
ins9[abcd].in	1 000 000	testy typu (a)-(d)
ins10[abcd].in	1 000 000	testy typu (a)-(d)