Jacek Tomasiewicz	Marek Sommer	Krzysztof Kiljan
Treść zadania	Opracowanie	Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień próbny, 1.04.2014*

FarmerCraft

W wiosce o nazwie Bajtowice znajduje się n domów połączonych za pomocą $n-1$ dróg. Z każdego domu można dojechać do każdego innego na dokładnie jeden sposób. Domy są ponumerowane liczbami od 1 do n. W domu o numerze 1 mieszka sołtys Bajtazar. W ramach programu zapewniania mieszkańcom wsi dostępu do najnowszych technologii, do domu Bajtazara dostarczona została paczka zawierająca n komputerów. Do każdego domu we wsi ma trafić jeden z komputerów. Mieszkańcy Bajtowic zgodnie postanowili, że jak tylko otrzymają komputery, zagrają razem przez sieć w najnowszą wersję gry „FarmerCraft".

Bajtazar załadował paczkę do swojego samochodu z kratką i za chwilę wyruszy, aby rozwieźć komputery po domach. Benzyny starczy mu tylko na przejechanie każdą drogą co najwyżej dwukrotnie. W każdym domu Bajtazar zostawia jeden komputer i od razu rusza w dalszą drogę. Mieszkańcy każdego domu, natychmiast po otrzymaniu komputera, zabierają się za instalację gry FarmerCraft. Dla każdego domu znany jest czas, jaki jest na to konieczny (zależny od stopnia sprawności informatycznej jego mieszkańców). Po rozwiezieniu wszystkich komputerów Bajtazar wraca do siebie i również zabiera się za instalację gry. Czas przejechania każdą drogą bezpośrednio łączącą dwa domy wynosi dokładnie 1 minutę, a czas wypakowywania komputerów jest pomijalny.

Pomóż Bajtazarowi ustalić taką kolejność rozwożenia komputerów, aby wszyscy mieszkańcy wsi (wraz z Bajtazarem) mogli jak najszybciej rozpocząć grę. Dokładniej, interesuje nas najwcześniejszy moment, w którym wszyscy będą mieli zainstalowaną grę FarmerCraft.

Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną liczbę całkowitą n ($2 \leqslant n \leqslant 500\,000$), oznaczającą liczbę domów w Bajtowicach. Drugi wiersz zawiera n liczb całkowitych $c_1, c_2, \ldots, c_n$ ($1 \leqslant c_i \leqslant 10^9$) pooddzielanych pojedynczymi odstępami; c_i oznacza czas instalacji gry (w minutach) przez mieszkańców domu o numerze i.

Kolejne $n-1$ wierszy opisuje drogi łączące domy. Każdy z tych wierszy zawiera dwie dodatnie liczby całkowite a i b ($1 \leqslant a < b \leqslant n$) oddzielone pojedynczym odstępem. Oznaczają one, że istnieje bezpośrednia droga pomiędzy domami o numerach a i b.

Możesz założyć, że w testach wartych łącznie 40% punktów zachodzi dodatkowy warunek $n \leqslant 7000$.

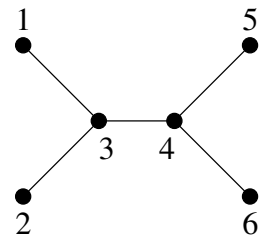
Wyjście

Pierwszy i jedyny wiersz standardowego wyjścia powinien zawierać jedną liczbę całkowitą, równą minimalnemu czasowi (w minutach), po którym wszyscy mieszkańcy będą mogli rozpocząć wspólną grę w FarmerCrafta.

Przykład

Dla danych wejściowych:

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



poprawnym wynikiem jest:

```
11
```

Wyjaśnienie do przykładu: *Bajtazar powinien zawieźć komputery kolejno do domów o numerach: 3, 2, 4, 5, 6 i 1. Gra będzie zainstalowana na komputerach odpowiednio (w kolejności numeracji domów) po: 11, 10, 10, 10, 8 i 9 minutach. Wszyscy mogą rozpocząć grę po 11 minutach.*

Gdyby Bajtazar zawiózł komputery kolejno do domów: 3, 4, 5, 6, 2 i 1, to gra byłaby na nich zainstalowana odpowiednio po: 11, 16, 10, 8, 6 i 7 minutach, a wszyscy mogliby rozpocząć grę dopiero po 16 minutach.

Rozwiązanie

W zadaniu mamy dane drzewo o n wierzchołkach, ukorzenione w wierzchołku nr 1. Każdy wierzchołek i ma przypisaną pewną wartość c_i – czas instalacji gry. Przejazd dowolną krawędzią trwa jedną minutę.

Szukamy trasy o najmniejszym koszcie, która zaczyna się w korzeniu, odwiedza każdy wierzchołek i kończy się w korzeniu. Trasa nie może przechodzić żadną krawędzią więcej niż dwa razy. *Kosztem* trasy nazywamy wartość:

$$\max\left(t_1 + c_1, t_2 + c_2, \ldots, t_n + c_n\right),$$

gdzie t_i oznacza czas rozpoczęcia instalacji gry w i-tym wierzchołku – dla korzenia jest to czas przejścia całej trasy, a dla pozostałych wierzchołków jest to czas, po którym zostaną one odwiedzone po raz pierwszy.

Sposób obchodzenia drzewa

Najistotniejszą informacją w zadaniu jest to, że przez każdą krawędź możemy przejść co najwyżej dwukrotnie. To oznacza, że gdy wejdziemy do pewnego poddrzewa, to nie będziemy mogli z niego wyjść, dopóki nie odwiedzimy wszystkich wierzchołków, które się w tym poddrzewie znajdują. Gdybyśmy wyszli z tego poddrzewa wcześniej, to nigdy byśmy do niego nie wrócili, ponieważ jedyna krawędź łącząca to poddrzewo z resztą drzewa zostałaby już odwiedzona dwukrotnie (podczas wejścia i podczas wyjścia). Trasa, której szukamy, jest zatem obejściem drzewa algorytmem DFS. Dowolny

porządek DFS odpowiada poprawnej trasie, jednak w zadaniu trzeba jeszcze zminimalizować koszt trasy. W tym celu należy w każdym wierzchołku wybrać optymalną kolejność, w jakiej będziemy odwiedzali jego synów.

Instalacja gry w korzeniu

Sposób instalacji gry w korzeniu drzewa (w domu Bajtazara) jest niepotrzebnie wyróżniony. Czas, po którym Bajtazar będzie miał zainstalowaną grę, jest równy czasowi obejścia całego drzewa (ten czas jest równy dwukrotności liczby krawędzi), powiększonemu o c_1. Możemy wyznaczyć tę wartość na samym początku, po czym obliczyć odpowiedź dla uproszczonej wersji zadania, w której w każdym wierzchołku gra jest instalowana w chwili pierwszego odwiedzenia tego wierzchołka (czyli w korzeniu jest instalowana od razu). Końcową odpowiedzią będzie większy z tych dwóch czasów.

Programowanie dynamiczne

Wszystkie podejścia do rozwiązania tego zadania opierają się na metodzie programowania dynamicznego. Aby móc skorzystać z tej metody, niezbędna jest następująca obserwacja:

Obserwacja 1. Optymalna kolejność odwiedzania synów wierzchołka v nie zależy od tego, w jakiej kolejności odwiedzane były inne poddrzewa przed dojściem do v.

Powyższe stwierdzenie jest raczej naturalne – kolejność odwiedzania wierzchołków przed dotarciem do v wpływa tylko na to, po jakim czasie Bajtazar dotrze do poddrzewa v. Niech t będzie czasem instalacji ostatniej gry w poddrzewie v, przy założeniu, że Bajtazar dociera do wierzchołka v w czasie t_1 i przechodzi dane poddrzewo optymalną trasą. Gdyby Bajtazar odwiedził wierzchołek v w czasie t_2, to sposób obejścia tego poddrzewa nie powinien się zmienić, a czas, po którym zostałaby zainstalowana ostatnia gra, byłby równy $t - t_1 + t_2$.

Spróbujmy zatem dla każdego poddrzewa v obliczyć minimalny czas instalacji ostatniej gry w tym poddrzewie, gdyby Bajtazar dotarł do wierzchołka v po czasie 0. Oznaczmy tę wartość przez T_v. Na mocy tego, co powiedzieliśmy do tej pory, taka wartość pozwala obliczyć minimalny czas instalacji, jeśli Bajtazar odwiedzi poddrzewo v w dowolnej innej chwili.

Oprócz czasów T_v, przydadzą nam się jeszcze wartości K_v, które będą oznaczać czas obejścia całego poddrzewa v. Jest to zawsze dokładnie dwukrotność liczby krawędzi, które znajdują się w poddrzewie v.

We wszystkich rozwiązaniach będziemy wyznaczać wartości T_v i K_v dla każdego poddrzewa, zaczynając od liści i kończąc w korzeniu. Po zakończeniu obliczeń, minimalny czas instalacji ostatniej gry w całym drzewie będzie można odczytać z T_1.

Dynamiczne obliczanie wartości T_v i K_v

Niech v będzie liściem. Jeśli wejdziemy do niego po czasie 0, to gra zostanie tam zainstalowana po czasie c_v. Zatem $T_v = c_v$. Skoro v jest liściem, to nie wychodzą z niego żadne krawędzie, więc $K_v = 0$.

Niech teraz v nie będzie liściem. Oznaczmy jego synów przez s_1, s_2, …, s_k. Czas obejścia poddrzewa v jest sumą czasów obejścia wszystkich poddrzew, które z niego wychodzą. Należy do tego doliczyć czas na wejście do syna (1 minuta) i czas na powrót do ojca (1 minuta):

$$K_v = \sum_{i=1}^{k} (1 + K_{s_i} + 1)\,.$$

Aby obliczyć wartość T_v należy ustalić optymalną kolejność odwiedzania synów. Załóżmy na razie, że udało się ją znaleźć: s_{p_1}, s_{p_2}, …, s_{p_k}. Niech $X_{s_{p_i}}$ będzie czasem, po którym Bajtazar wejdzie do poddrzewa s_{p_i}:

$$X_{s_{p_i}} = \sum_{j=1}^{i-1} \left(1 + K_{s_{p_j}} + 1\right) + 1.$$

Na powyższą sumę składają się czasy obejścia wszystkich wcześniejszych poddrzew i czas na przejście krawędzią od wierzchołka v do wierzchołka s_{p_i}.

Na podstawie wartości $X_{s_{p_i}}$ można prosto wyznaczyć wartość T_v:

$$T_v = \max\left(X_{s_{p_1}} + T_{s_{p_1}}, X_{s_{p_2}} + T_{s_{p_2}}, \ldots, X_{s_{p_k}} + T_{s_{p_k}}\right).$$

Powyższe wzory pozwalają w czasie $O(k)$ obliczyć wartości K_v i T_v na podstawie wyników dla poszczególnych synów v. Brakuje jeszcze tylko sposobu na znalezienie optymalnego uporządkowania poddrzew.

Optymalna kolejność odwiedzania poddrzew

Najprostszym rozwiązaniem byłoby sprawdzenie wszystkich możliwych permutacji synów, a spośród nich wybranie tej najtańszej. To rozwiązanie działa w złożoności $O(k!)$ dla pojedynczego wierzchołka, dając w rezultacie algorytm o złożoności $O(n!)$, co dla danych z wejścia (n jest rzędu 500 000) jest zdecydowanie za dużo. Implementacja tego algorytmu znajduje się w pliku `fars2.cpp`. Podczas zawodów za takie rozwiązanie można było otrzymać 20 punktów.

Zastanówmy się, który z synów powinien zostać odwiedzony jako ostatni. Niech G_{s_i} oznacza czas, po którym ostatnia gra zostałaby zainstalowana w poddrzewie s_i, gdyby syn s_i został odwiedzony na samym końcu. Wówczas:

$$\begin{aligned} G_{s_i} &= \sum_{j \in \{1,2,\ldots,k\} \setminus \{i\}} \left(1 + K_{s_j} + 1\right) + 1 + T_{s_i} = \\ &= \sum_{j \in \{1,2,\ldots,k\}} \left(1 + K_{s_j} + 1\right) - (1 + K_{s_i} + 1) + 1 + T_{s_i} = \\ &= K_v - 1 + T_{s_i} - K_{s_i}. \end{aligned}$$

Lemat 1. Istnieje optymalna kolejność przechodzenia poddrzew, w której poddrzewo o najmniejszej wartości G_{s_i} jest odwiedzane jako ostatnie.

Dowód: Ustalmy dowolną optymalną kolejność przechodzenia poddrzew: s_{p_1}, s_{p_2}, ..., s_{p_k}. Jeśli $G_{s_{p_k}}$ jest najmniejsze, to teza jest spełniona. Załóżmy przeciwnie: niech h będzie takim indeksem, że $G_{s_{p_h}}$ przyjmuje najmniejszą wartość spośród wszystkich G_{s_i} (wówczas $G_{s_{p_h}} < G_{s_{p_k}}$).

Wykażemy, że jeśli przeniesiemy syna s_{p_h} z h-tej pozycji na sam koniec, to kolejność ciągle będzie optymalna.

Spójrzmy, jak zmienią się największe czasy instalacji gier w poszczególnych poddrzewach. W poddrzewach od s_{p_1} do $s_{p_{h-1}}$ nic się nie zmieni. Poddrzewa od $s_{p_{h+1}}$ do s_{p_k} zostaną odwiedzone o $(1 + K_{s_{p_h}} + 1)$ minut wcześniej, więc czasy instalacji mogą się tylko zmniejszyć. Natomiast w poddrzewie s_{p_h}, które zostało przeniesione na sam koniec, największy czas instalacji mógł się zwiększyć. Wiemy jednak, że ten czas jest równy dokładnie $G_{s_{p_h}}$, zatem przeniesienie nie mogło pogorszyć wyniku, który był równy co najmniej $G_{s_{p_k}}$.

Po wykonaniu przeniesienia h-tego elementu na koniec, koszt trasy okazał się nie większy niż przed modyfikacją. Zatem kolejność, w której poddrzewo s_{p_h} jest na końcu, również jest optymalna. ■

Korzystając z lematu 1, można skonstruować algorytm działający w złożoności $O(k^2)$. Algorytm ten będzie obliczał wartości G_{s_i} dla wszystkich synów, wybierał tego z najmniejszym wynikiem i ustawiał go na końcu, a następnie powtarzał całą procedurę dla pozostałych synów (wybierając przedostatniego syna, potem przedprzedostatniego, itd.). Złożoność całego rozwiązania to $O(n^2)$. Implementacja znajduje się w pliku `fars1.cpp`. Na zawodach za takie rozwiązanie można było otrzymać 40 punktów, obiecanych w treści zadania.

Do otrzymania rozwiązania wzorcowego wystarczy już tylko jedna drobna obserwacja. Zamiast k razy wybierać kandydata na ostatni element, możemy posortować wszystkich synów względem różnicy $T_{s_i} - K_{s_i}$. Syn, dla którego wartość $T_{s_i} - K_{s_i}$ jest najmniejsza, ma również najmniejszą wartość G_{s_i}, niezależnie od wartości K_v, która pojawia się we wzorze na G_{s_i}. Nie trzeba zatem przeliczać wartości G_{s_i} na nowo za każdym razem gdy wybierzemy ostatnie poddrzewo – kolejność poddrzew się nie zmieni. W każdym wierzchołku v można więc znaleźć optymalną kolejność w czasie $O(k \log k)$. Całkowita złożoność tego rozwiązania jest równa $O(n \log n)$. Jego implementacje znajdują się w plikach `far.cpp`, `far2.cpp` i `far3.cpp`.

Testy

Do oceny rozwiązań przygotowano 10 grup testów. Występowały następujące rodzaje testów:

- *1a-10a* – testy, w których to Bajtazar ma największy czas instalacji gry,
- *1b-10b*, *1f-6f* – testy obalające rozwiązania zachłanne: istnieje poddrzewo, w którym opłaca się najpierw iść do liścia o mniejszym czasie instalacji,
- *1c-10c* – testy, w których Bajtazar ma stosunkowo mały czas instalacji w porównaniu do innych,

- *1d-10d* – testy, w których z korzenia wyrasta $\sqrt{n}$ poddrzew, z których każde ma po $\sqrt{n}$ węzłów,
- *1e-10e* – testy całkowicie losowe,
- *3g*, *5g*, *6g*, *7f-10f* – testy, w których drzewo ma strukturę gwiazdy.