Paweł Parys Treść zadania, Opracowanie

Arkadiusz Paterek Program

Jaskinia

W Bajtocji znajduje się jaskinia, która składa się z n komór oraz z łączących je korytarzy. Korytarze są tak ułożone, że między każdymi dwiema komorami można przejść tylko w jeden sposób. W jednej z komór Jaś ukrył skarb, ale nie chce powiedzieć w której. Małgosia koniecznie chce się tego dowiedzieć. Pyta więc Jasia kolejno o różne komory. Jeśli Małgosia trafi, to Jaś mówi jej o tym, a jeśli nie trafi, to mówi jej, w którą stronę trzeba iść z danej komory w kierunku skarbu.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis jaskini,*
- *obliczy minimalną liczbę pytań, które w pesymistycznym przypadku musi zadać Małgosia, żeby wiedzieć, w której komorze znajduje się skarb,*
- *wypisze obliczony wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna dodatnia liczba całkowita n, $1 \leqslant n \leqslant 50\,000$. Jest to liczba komór w jaskini. Komory jaskini są ponumerowane od 1 do n. W kolejnych $n-1$ wierszach opisane są korytarze łączące komory, po jednym w wierszu. W każdym z tych wierszy znajduje się para różnych dodatnich liczb całkowitych a i b ($1 \leqslant a, b \leqslant n$), oddzielonych pojedynczym odstępem. Para taka oznacza, że komory a i b są połączone korytarzem.

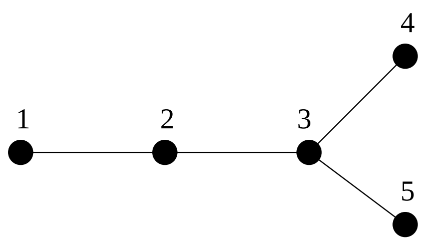
Wyjście

Na standardowe wyjście powinna być wypisana jedna liczba całkowita, minimalna liczba pytań, jakie musi zadać Małgosia w pesymistycznym przypadku (tzn. zakładamy, że Małgosia zadaje pytania najlepiej jak można, ale skarb jest umieszczony w komorze, która wymaga zadania największej liczby pytań).

Przykład

Dla danych wejściowych:

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



poprawnym wynikiem jest:

```
2
```

Rozwiązanie

Plan jaskini jest drzewem. Pytając o dany wierzchołek, dzielimy to drzewo na pewną liczbę części, które powstają przez usunięcie tego wierzchołka. W odpowiedzi dostajemy jedną z tych części i teraz w niej musimy znaleźć skarb. Możliwe są jednak różne odpowiedzi, więc trzeba rozważyć każdą z nich. Można więc spojrzeć na to tak: W drzewie usuwamy wybrany jeden wierzchołek. W następnym kroku z każdej z powstałych części wybieramy znowu po jednym wierzchołku i je usuwamy. Tak postępujemy, aż uzyskane części będą jednowierzchołkowe.

W zadaniu chodzi o zminimalizowanie liczby tych kroków. Chcemy więc, aby powstające w każdym kroku części (największa z nich) były jak najmniejsze.

W przypadku, gdy w jaskini nie ma rozgałęzień, możemy po prostu zastosować wyszukiwanie binarne — pytamy zawsze o wierzchołek znajdujący się w samym środku jaskini. W ogólnej sytuacji też widać, że trzeba pytać mniej więcej w środku jaskini. Nie wiadomo jednak, w jakim sensie ma to być środek. Widać też, że bardziej opłaca się wybierać wierzchołki, które mają więcej sąsiadów, wtedy podzielimy drzewa na więcej mniejszych części. Można próbować w wierzchołku, z którego odległość do pozostałych wierzchołków jest minimalna. Albo tak, aby powstałe części miały jak najmniej wierzchołków. Można też strzelać, że wynik jest logarytmem z liczby wierzchołków lub ze średnicy drzewa (tzn. odległości między najdalszymi wierzchołkami). Łatwo jednak sprawdzić, że żadna z tych heurystyk nie jest poprawna.

Idea rozwiązania

Najpierw w dowolny sposób wybierzmy sobie korzeń naszego drzewa. Chodzi tylko o to, aby ustalić kolejność, w której będziemy przeglądać wierzchołki przy poszukiwaniu optymalnego rozwiązania.

Rozwiązanie opiera się na następującym pomyśle: Będziemy rozważać nasze drzewo, poczynając od liści i idąc w stronę korzenia. Najpierw wywołamy obliczenia dla poddrzew danego wierzchołka. Następnie na podstawie wyników dla poddrzew obliczymy wynik dla drzewa mającego korzeń w tym wierzchołku. A więc na podstawie wyników dla mniejszych drzew liczymy wynik dla większego.

Znamy więc minimalną liczbę pytań w poddrzewie. Chcemy jednak, aby (w ramach ustalonej liczby pytań) pytania zadawać jak najwyżej. Na przykład im wyżej zadamy pierwsze pytanie, tym mniejsza będzie część znajdująca się nad wierzchołkiem o który pytamy, więc tym więcej wierzchołków można jeszcze dołożyć z góry do drzewa, aby liczba potrzebnych pytań nie zmieniła się.

Można się przekonać, że znając tylko liczbę pytań potrzebną w poddrzewach, nie jesteśmy w stanie obliczyć liczby pytań w większym drzewie. Trzeba również wiedzieć coś o tym, jak duża jest część nad wierzchołkiem, w którym zadajemy pierwsze pytanie. W szczególności chcielibyśmy wiedzieć, ile pytań potrzeba na zbadanie tej górnej części, ale też coś na temat części ponad pierwszym pytaniem w tej części, itd.

Rozwiązanie wzorcowe

Sposób na zadawanie pytań będziemy zapisywać za pomocą funkcji f. Każdemu wierzchołkowi w będziemy chcieli przypisać nieujemną liczbę $f(w)$. Chcemy przy tym zachować następujący warunek $(*)$, że jeśli dla dwóch wierzchołków u i v jest $f(u) = f(v)$, to na drodze z u do v znajduje się wierzchołek w taki, że $f(w) > f(u)$. Intuicyjnie $f(w)$ oznacza numer pytania, licząc od końca, które będziemy zadawać w wierzchołku w.

Dla zadanej funkcji f pytania będziemy zadawać w następujący sposób: Pytamy o wierzchołek w, dla którego $f(w)$ jest największe. W odpowiedzi otrzymujemy pewną część drzewa. Znowu pytamy o wierzchołek w, dla którego $f(w)$ jest największa w tej części drzewa, itd. Zawsze w otrzymanej części drzewa mamy dokładnie jeden wierzchołek o największym $f(w)$ (wynika to z warunku $(*)$). Albo w którymś momencie trafimy, albo na koniec otrzymamy drzewo składające się z jednego wierzchołka i będziemy wiedzieli, że tam jest skarb. W ten sposób zadajemy co najwyżej $K = \max_w f(w)$ pytań. Chcemy znaleźć taką funkcję f, dla której K jest najmniejsze.

Jak już było powiedziane, najlepszej funkcji $f(w)$ będziemy szukać w drzewie ukorzenionym. A więc wybierzmy w dowolny sposób korzeń. Niech $T(w)$ oznacza poddrzewo o korzeniu w w. Oprócz funkcji $f(w)$ będziemy obliczać zbiory $S(w)$. Zbiór $S(w)$ jest zbiorem wartości funkcji, które „widać” w drzewie $T(w)$, patrząc od ojca w. Oznacza to, że liczba a należy do $S(w)$ wtedy i tylko wtedy, gdy w $T(w)$ jest wierzchołek u taki, że $f(u) = a$ i na drodze z u do ojca w nie występuje liczba większa niż $f(u)$. Inaczej mówiąc: Liczba pytań, które trzeba zadać w drzewie $T(w)$ należy do $S(w)$. Liczba pytań, które trzeba zadać w części ponad wierzchołkiem, w którym zadamy pierwsze pytanie, również należy do $S(w)$, itd.

Postępujemy w następujący sposób: Przed policzeniem f i S dla w obliczamy f i S dla wszystkich synów w. Niech R oznacza $\bigcup_u S(u)$, gdzie u to wszyscy synowie w. Niech m będzie największą liczbą taką, że $m \in S(u)$ dla więcej niż jednego u. (Jeśli takich liczb nie ma, to niech $m = -1$.) Jako $f(w)$ bierzemy najmniejszą liczbę większą od m i nie należącą do R. Zauważmy, że $S(w) = R \cup \{f(w)\} \setminus \{0, 1, \ldots, f(w)-1\}$. Łatwo sprawdzić, że powstała funkcja f spełnia warunek $(*)$. Przyglądając się dokładniej temu warunkowi widzimy, że tak zdefiniowane $f(w)$ jest najmniejszą wartością, przy której $(*)$ jest spełniony (dla już ustalonych wartości f w poddrzewach).

Trzeba teraz udowodnić, że otrzymana w ten sposób funkcja jest najlepsza. Będziemy to robić indukcyjnie ze względu na minimalną liczbę pytań K. Teza indukcyjna jest taka, że jeśli powyższy algorytm dla pewnego drzewa zwrócił K, to rzeczywiście w tym drzewie potrzeba co najmniej K pytań. Dla $K = 0$ jest to oczywiste. Dla $K = 1$ nasze drzewo składa się z co najmniej dwóch wierzchołków, czyli nie można znaleźć skarbu w 0 ruchach. Teraz chcemy udowodnić poprawność powyższego rozwiązania dla pewnego K, wiedząc, że dla wszystkich mniejszych K jest ono prawdziwe. Przyjmijmy, że dla pewnego drzewa T można znaleźć skarb w $K-1$ ruchach, zadając pierwsze pytanie w wierzchołku u. Niech v_1 będzie wierzchołkiem, dla którego $f(v_1) = K$. W drzewie istnieje również pewien potomek v_2 (co najmniej jeden) wierzchołka v_1, dla którego $f(v_2) = K-1$. Niech T_1 oznacza drzewo $T(v_1) - T(v_2)$ (czyli drzewo zawierające wierzchołki $T(v_1)$ za wyjątkiem wierzchołków $T(v_2)$). Rozważmy dwa przypadki:

- u należy do $T(v_2)$ — Po zadaniu pytania w u skarb może znajdować się w tej części drzewa T, która zawiera v_1. Ma więc istnieć sposób na znalezienie tam skarbu w $K-2$ pytaniach. Ta część zawiera drzewo T_1, czyli w drzewie T_1 również można znaleźć skarb w $K-2$ pytaniach. Popatrzmy jednak, co się stanie z obliczoną przez program funkcją f, gdy z drzewa T usuniemy drzewo $T(v_2)$. Ponieważ $S(v_2) = \{K-1\}$, to wartość $f(v_1)$ zmniejszy się najwyżej o 1, do wartości $K-1$. Zatem na mocy założenia indukcyjnego na drzewo T_1 potrzeba $K-1$ pytań. Sprzeczność.

- u nie należy do $T(v_2)$ — Po zadaniu pytania w u skarb może znajdować się w tej części drzewa T, która zawiera v_2. Ma więc istnieć sposób na znalezienie tam skarbu w $K-2$ pytaniach. Ta część zawiera drzewo $T(v_2)$, czyli w drzewie $T(v_2)$ też można znaleźć skarb w $K-2$ pytaniach. Ale ponieważ $f(v_2) = K-1$, to na mocy założenia indukcyjnego na drzewo $T(v_2)$ potrzeba co najmniej $K-1$ pytań. Sprzeczność.

W obu przypadkach dochodzimy do sprzeczności, nie można znaleźć skarbu w drzewie T w mniej niż K pytaniach. Krok indukcyjny jest prawdziwy. To kończy dowód.

Podany algorytm działa w czasie $O(n \log n)$.

Rozwiązanie w czasie liniowym

Powyższe rozwiązanie można łatwo zmodyfikować tak, aby działało w czasie liniowym. Wystarczy zbiory $S(w)$ reprezentować za pomocą kolejnych bitów liczby. Największą liczbą znajdującą się w $S(w)$ może być wynik K, który jest nie większy niż logarytm z liczby wierzchołków drzewa. Zatem do reprezentacji tych zbiorów wystarczą liczby długości takiej samej, jak do reprezentacji numerów wierzchołków, czyli uwzględniając ograniczenia zadania, od 0 do $2^{17}-1$. Pozostaje jeszcze sprawa wykonywania operacji na tych zbiorach. Chcielibyśmy to robić w czasie stałym.

- Sumę zbiorów można policzyć wykonując OR na reprezentacjach.

- Chcemy też znajdować najmniejszą lub największą wartość należącą do danego zbioru. Nie ma instrukcji procesora obliczającej tę wartość w czasie stałym. Możemy jednak obliczyć odpowiedzi na początku — dla każdej reprezentacji zbioru będziemy pamiętać wynik. Korzystamy z tego, że reprezentacja zbioru jest niewielką liczbą — różnych reprezentacji jest mniej więcej tyle, co wierzchołków drzewa.

- Wstawianie danej wartości do zbioru i usuwanie ze zbioru wartości mniejszych niż dana można zrealizować za pomocą AND, OR i przesunięć bitowych.

- Znajdowanie zbioru zawierającego liczby należące do co najmniej dwóch spośród danych zbiorów S_1, S_2, $\ldots$, S_n również można zrealizować za pomocą AND i OR. Liczymy najpierw $A_0 = \phi$, $A_k = S_k \cup A_{k-1}$, dla $k = 1, \ldots, n$. Zbiór A_k zawiera liczby należące do S_1, S_2, $\ldots$, S_k. Następnie liczymy $B_0 = \phi$, $B_k = (S_k \cap A_{k-1}) \cup B_{k-1}$, dla $k = 1, \ldots, n$. Zbiór B_k zawiera liczby należące do co najmniej dwóch zbiorów spośród $S_1, S_2, \ldots, S_k$, a zatem B_n jest szukanym zbiorem.

Testy

Zadanie testowane było na zestawie 13 danych testowych. Testy były generowane w dużym stopniu losowo, przy zadanych różnych prawdopodobieństwach otrzymania wierzchołków danego stopnia. Rozmiary testów zawiera poniższa tabela.

nr testu	n	wynik
1	17	3
2	16	3
3	32	3
4	100	5
5	1000	8
6	5000	11
7	10000	10
8	22000	12
9	24000	12
10	30000	11
11	40000	13
12	40063	7
13	50000	11