Jacek Tomasiewicz — Treść zadania, Opracowanie
Karol Pokorski — Program

Dostępna pamięć: 128 MB. *OI, etap II, dzień pierwszy, 13.02.2013*

Łuk triumfalny

Król Bajtocji, Bajtazar, wraca po zwycięskiej bitwie do swojego kraju. W Bajtocji jest n miast połączonych zaledwie $n-1$ drogami. Wiadomo, że z każdego miasta da się dojechać do każdego innego dokładnie jedną trasą, złożoną z jednej lub większej liczby dróg. (Inaczej mówiąc, sieć dróg tworzy **drzewo***).*

Król właśnie wkroczył do stolicy Bajtocji. W mieście tym postawiono łuk triumfalny, czyli bramę, pod którą przejeżdża król po odniesieniu zwycięstwa. Bajtazar, zachwycony przyjęciem przez poddanych, zaplanował pochód triumfalny, w którym odwiedzi wszystkie miasta Bajtocji, zaczynając z miasta, w którym aktualnie przebywa.

Pozostałe miasta nie są przygotowane na przyjazd króla – nie zostały w nich jeszcze postawione łuki triumfalne. Nad budową łuków czuwa doradca Bajtazara. Chce on zatrudnić pewną liczbę ekip budowlanych. Każda ekipa codziennie może wybudować jeden łuk, w dowolnie wybranym mieście. Niestety nikt nie wie, w jakiej kolejności król będzie odwiedzał miasta. Wiadomo jedynie, że każdego dnia król przemieści się z miasta, w którym się aktualnie znajduje, do sąsiedniego miasta. Król może odwiedzać poszczególne miasta dowolnie wiele razy (jednak wystarczy mu jeden łuk triumfalny w każdym mieście).

Doradca Bajtazara musi zapłacić każdej ekipie budowlanej tyle samo, bez względu na to, ile zbuduje łuków triumfalnych. Z jednej strony, doradca chce mieć pewność, że każde miasto aktualnie odwiedzane przez króla będzie miało łuk triumfalny. Z drugiej jednak strony chciałby zatrudnić jak najmniej ekip budowlanych. Pomóż mu i napisz program, który pomoże wyznaczyć minimalną konieczną liczbę ekip budowlanych.

Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 300\,000$) oznaczającą liczbę miast w Bajtocji. Miasta są ponumerowane od 1 do n, przy czym stolica Bajtocji ma numer 1.

Sieć dróg Bajtocji jest opisana w kolejnych $n-1$ wierszach. Każdy z tych wierszy zawiera dwie liczby całkowite a, b ($1 \leqslant a, b \leqslant n$) oddzielone pojedynczym odstępem, oznaczające, że miasta a i b są połączone bezpośrednio dwukierunkową drogą.

W testach wartych łącznie 50% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$.

Wyjście

Pierwszy i zarazem jedyny wiersz standardowego wyjścia powinien zawierać jedną liczbę całkowitą równą minimalnej liczbie ekip budowlanych, które powinien zatrudnić doradca Bajtazara.

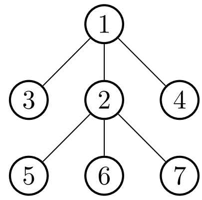
Przykład

Dla danych wejściowych:

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

poprawnym wynikiem jest:

```
3
```



Wyjaśnienie do przykładu: *W pierwszym dniu należy wybudować łuki triumfalne w miastach 2, 3, 4. Później wystarczy je zbudować w miastach 5, 6, 7.*

Testy „ocen":

`1ocen`: $n = 8\,191$, *miasta tworzą pełne drzewo binarne o wysokości 12;*

`2ocen`: $n = 300\,000$, *miasta* $1, \ldots, n$ *są ułożone wzdłuż jednej ścieżki;*

`3ocen`: $n = 5$, *prosty test poprawnościowy;*

`4ocen`: $n = 6$, *prosty test poprawnościowy;*

`5ocen`: $n = 10\,000$, *dziewięć długich ścieżek zwisających z korzenia.*

Rozwiązanie

Zadanie można rozpatrywać jako rodzaj gry rozgrywającej się na drzewie, którego wierzchołki oznaczać będą miasta Bajtocji, zaś krawędzie – drogi pomiędzy miastami. W grze tej pierwszym graczem jest król, a drugim – ekipy budujące łuki triumfalne.

Celem pierwszego gracza jest wejście do wierzchołka drzewa, w którym nie ma łuku triumfalnego, natomiast celem drugiego gracza jest niedopuszczenie do takiej sytuacji. Król może w ciągu jednego ruchu przemieścić się do sąsiedniego wierzchołka, natomiast każda ekipa budowlana może w jednym ruchu wybudować łuk triumfalny w jednym, dowolnym wierzchołku.

Zadanie polega na wyznaczeniu minimalnej liczby ekip budowlanych, które mają strategię wygrywającą w tej grze (czyli taką, że król nie jest w stanie wejść do wierzchołka bez łuku triumfalnego).

Strategie graczy

Ukorzeńmy drzewo w wierzchołku reprezentującym stolicę Bajtocji. Zauważmy, że jeśli król może wygrać – tzn. niezależnie od działań ekip budowlanych istnieje trasa króla kończąca się w pewnym wierzchołku v, taka że w chwili wejścia króla do v nie stoi tam łuk triumfalny – to może to równie dobrze zrobić, poruszając się jedynie w dół drzewa – czyli wybierając najkrótszą ścieżkę z korzenia do wierzchołka v. Faktycznie, gdyby na ścieżce króla z drzewa do v jakiś wierzchołek w występował dwukrotnie, to po usunięciu fragmentu ścieżki między tymi dwoma wystąpieniami król byłby tylko w lepszej sytuacji (bo ekipy budowlane miałyby mniej czasu na budowanie łuków). Zatem jeśli stwierdzimy, że ekipy budowlane są w stanie wygrywać do momentu wejścia króla do liścia drzewa, to będziemy wiedzieli, że są już w stanie wygrywać do końca gry.

Zauważmy, że jeśli król znajduje się w pewnym wierzchołku v, to przed jego kolejnym ruchem we wszystkich synach wierzchołka v muszą zostać wybudowane łuki triumfalne. To sugeruje następującą strategię wygrywającą dla ekip budowlanych: po ruchu króla w dół drzewa do wierzchołka v, budowane są łuki triumfalne we wszystkich synach wierzchołka v. Jeśli K jest maksymalną liczbą synów, które posiada pewien wierzchołek drzewa, to ta strategia wymaga K ekip budowlanych.

Niestety, K nie musi być minimalną liczbą potrzebnych ekip budowlanych. Jeśli w teście przykładowym doczepilibyśmy dwa nowe wierzchołki do wierzchołka 2, to posiadałby on $K = 5$ synów. Do wygrania z królem wystarczą natomiast 4 ekipy (w pierwszym dniu budują one łuki triumfalne w wierzchołkach 2, 3, 4 i 5; w drugim dniu w pozostałych wierzchołkach).

Rozwiązanie stosujące tę strategię znajduje się w pliku `lukb1.cpp`. Nie otrzymuje ono żadnych punktów.

Rozwiązanie wzorcowe

Podstawowa obserwacja jest następująca: jeśli k ekip budowlanych może wygrać z królem, to tym bardziej uda się to $k+1$ ekipom. Dzięki tej obserwacji, wartość k możemy wyszukiwać binarnie. Jedyne, co pozostaje, to odpowiadać na pytania: czy mając do dyspozycji ustaloną liczbę k ekip budowlanych, można wygrać niezależnie od ruchów króla.

W rozwiązaniu wykorzystamy metodę programowania dynamicznego. Dla każdego wierzchołka v obliczymy następującą wartość:

$dp[v]$ – minimalna, niezbędna liczba wcześniej wybudowanych łuków triumfalnych w poddrzewie zaczepionym w wierzchołku v, która zapewnia strategię wygrywającą, jeśli król właśnie wchodzi do wierzchołka v, w którym jest już łuk triumfalny.

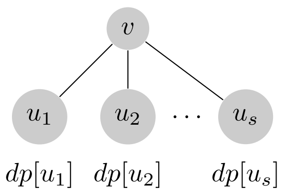
Nietrudno zauważyć, że po wyznaczeniu tych wartości, będzie można bezpośrednio odpowiedzieć na pytanie o strategię wygrywającą. Wystarczy bowiem odczytać wartość z wierzchołka o numerze 1, w którym znajduje się początkowo król:

- jeśli $dp[1] > 0$, to należałoby mieć wcześniej jakieś wybudowane łuki, a takich nie mamy, więc wtedy ekipy budowlane nie mają strategii wygrywającej,
- jeśli $dp[1] = 0$, to ekipy budowlane mają strategię wygrywającą.

Wyliczając wartości, poruszamy się od liści w kierunku korzenia. Jeśli wierzchołek v jest liściem, to nie ma żadnych innych wierzchołków w jego poddrzewie, zatem nie potrzeba tam budować innych łuków triumfalnych i $dp[v] = 0$.

Dla wierzchołka wewnętrznego v wyznaczamy $dp[v]$ na podstawie wartości w jego synach, których zbiór oznaczamy przez $synowie[v] = \{u_1, u_2, \ldots, u_s\}$ (patrz rys. 1). Dla każdego syna u_i musimy wybudować $dp[u_i]$ łuków triumfalnych w jego poddrzewie, a oprócz tego również łuk w samym wierzchołku u_i. Potrzebujemy zatem w sumie wybudować $\sum_{i=1}^{s}(dp[u_i] + 1)$ łuków triumfalnych. W najbliższym ruchu k ekip budowlanych może wybudować co najwyżej k łuków triumfalnych, zatem liczba łuków, które muszą być zbudowane przed tym ruchem, jest równa:

$$dp[v] = \max\Big(0,\ \sum_{i=1}^{s}(dp[u_i] + 1) - k\Big).$$



Rys. 1: Wyznaczanie wartości dp na podstawie wartości w synach $S = \{u_1, u_2, \ldots, u_s\}$.

Gdyby liczba wybudowanych łuków była mniejsza niż $dp[v]$, oznaczałoby to, że w którymś z synów wierzchołka v nie wybudowaliśmy łuku triumfalnego lub wchodząc do któregoś z synów, nie mamy wystarczająco dużo wybudowanych wcześniej łuków i tam król ma strategię wygrywającą. Król wybierze taki wierzchołek, jeśli tylko będzie mógł.

Implementacja

Rozwiązanie najprościej zaimplementować rekurencyjnie. Oto schematyczny zapis funkcji sprawdzającej:

```
function sprawdź(v, k)
begin
  suma := 0;
  foreach u ∈ synowie[v] do
    suma := suma + sprawdź(u, k) + 1;
  return max(0, suma − k);
end
```

W czasie implementacji zwykle nie przechowujemy oddzielnie wszystkich synów danego wierzchołka, a trzymamy zbiór wszystkich jego sąsiadów. Zawiera on także ojca tego wierzchołka w drzewie, powinniśmy więc umieć stwierdzać, czy analizowany sąsiad nie jest ojcem w drzewie. Najłatwiej to zrobić, przekazując do funkcji numer ojca jako argument:

```
function sprawdź(v, k, ojciec)
begin
  suma := 0;
  foreach u ∈ sasiedzi[v] do
    if u ≠ ojciec then
      suma := suma + sprawdź(u, k, v) + 1;
  return max(0, suma − k);
end
```

Złożoność czasowa takiej funkcji sprawdzającej jest liniowa względem liczby wierzchołków drzewa. Wobec tego cały algorytm, korzystający z wyszukiwania binarnego, działa w czasie $O(n \log n)$. Na tej zasadzie zostało zaimplementowane rozwiązanie wzorcowe, znajdujące się w pliku `luk.cpp` oraz `luk1.pas`.

Rozwiązanie wolne

Można było nie zauważyć możliwości wyszukiwania binarnego i uzyskać algorytm $O(n^2)$, sprawdzając kolejno każdą możliwą liczbę ekip budowlanych. Rozwiązanie takie otrzymywało około 50% punktów, a jego implementacja znajduje się w pliku `luks1.cpp` oraz `luks2.pas`.

Testy

W zestawie były cztery typy testów: małe testy poprawnościowe wygenerowane ręcznie; testy generowane losowo, w których wyniki są często bardzo małe; testy z długimi ścieżkami z małą częścią losową oraz testy przypominające drzewa *d*–arne (każdy wierzchołek albo jest liściem, albo ma kilku synów).