Jacek Tomasiewicz
Treść zadania, Opracowanie

Piotr Smulewicz
Program

Dostępna pamięć: 64 MB.

OI, etap III, dzień drugi, 14.04.2016

Parada

Jak co roku na powitanie wiosny ulicami Bajtogrodu przejdzie Wielka Wiosenna Parada Bajtocka. Swoją obecnością uświetni ją sam Król Bajtazar XVI. Sieć drogowa Bajtogrodu składa się z n skrzyżowań połączonych $n-1$ dwukierunkowymi odcinkami ulic (z każdego skrzyżowania da się dojechać do każdego innego).

Dokładna trasa parady nie jest jeszcze znana, ale wiadomo, że zacznie się ona w jednym ze skrzyżowań, będzie biegła pewną liczbą odcinków ulic i zakończy się na **innym** *skrzyżowaniu. Aby nie zanudzić paradujących, trasa przechodzić będzie przez każdy odcinek ulicy co najwyżej raz.*

Z uwagi na bezpieczeństwo uczestników parady, należy zamknąć bramką wlot każdego odcinka ulicy, przez który nie przechodzi parada, a który wchodzi do skrzyżowania, przez które parada przechodzi (włączając początkowe i końcowe skrzyżowanie). Należy wyznaczyć, ile takich bramek może być potrzebnych.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się liczba całkowita n ($n \geqslant 2$) oznaczająca liczbę skrzyżowań w Bajtogrodzie. Skrzyżowania numerujemy liczbami od 1 do n.

Kolejne $n-1$ wierszy opisuje sieć drogową Bajtogrodu. Każdy z nich zawiera dwie liczby całkowite a i b ($1 \leqslant a,b \leqslant n$, $a \neq b$) oddzielone pojedynczym odstępem, oznaczające, że skrzyżowania o numerach a i b są połączone dwukierunkowym odcinkiem ulicy.

Wyjście

W pierwszym i jedynym wierszu standardowego wyjścia należy wypisać jedną liczbę całkowitą, oznaczającą maksymalną liczbę bramek, które mogą być potrzebne do zabezpieczenia parady.

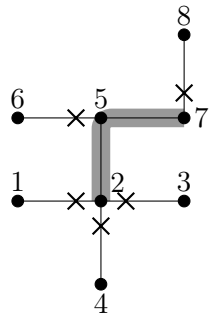
Przykład

Dla danych wejściowych:

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

poprawnym wynikiem jest:

```
5
```



Wyjaśnienie do przykładu: *Jeśli parada ruszy ze skrzyżowania 2 i zakończy się na skrzyżowaniu 7, to potrzebne będzie 5 bramek (3 do zamknięcia wlotów do skrzyżowania 2 i po jednej do zamknięcia wlotów do skrzyżowań 5 i 7).*

Testy „ocen":

`1ocen`: *$n = 20$, ścieżka;*

`2ocen`: *$n = 20$, gwiazda;*

`3ocen`: *$n = 1000$, losowy test o następującej własności: i-ty odcinek ulicy (dla $i = 1, \ldots, n-1$) łączy skrzyżowanie numer $i+1$ z jednym ze skrzyżowań o mniejszych numerach.*

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	Warunki	Liczba punktów
1	$n \leqslant 20$	*15*
2	$n \leqslant 300$	*16*
3	$n \leqslant 3000$	*22*
4	$n \leqslant 200\,000$	*47*

Rozwiązanie

W zadaniu mamy dane drzewo zawierające n węzłów. Szukamy najtrudniejszej w zabezpieczeniu trasy parady, czyli takiej ścieżki w tym drzewie, z którą połączonych jest bezpośrednio jak najwięcej innych węzłów – liczbę tych węzłów nazwiemy *trudnością* trasy.

Rozwiązanie siłowe $O(n^3)$

Sprawdzamy każdą możliwą ścieżkę i liczymy, z iloma węzłami ona sąsiaduje. Jako że wszystkich ścieżek jest $O(n^2)$, a pojedyncze sprawdzenie zajmuje czas liniowy względem długości ścieżki, złożoność czasowa tego rozwiązania to $O(n^3)$.

Rozwiązanie to zaimplementowane jest w pliku `pars4.cpp`. Za poprawne zaprogramowanie takiego rozwiązania na zawodach można było uzyskać około 30% punktów.

Rozwiązanie wolne $O(n^2)$

Sprawdzamy wszystkie możliwe początki ścieżki, ukorzeniając drzewo w każdym z węzłów. Zakładamy, że korzeń jest początkiem ścieżki, która będzie prowadziła w dół drzewa. Wykonując przeszukiwanie drzewa w głąb (DFS), w czasie stałym aktualizujemy liczbę węzłów sąsiadujących ze ścieżką prowadzącą do aktualnie odwiedzanego

węzła. W ten sposób wszystkie ścieżki o ustalonym początku rozpatrujemy w łącznym czasie $O(n)$. Jako że wszystkich początków jest $O(n)$, to złożoność czasowa tego rozwiązania to $O(n^2)$.

Implementacja takiego rozwiązania znajduje się w pliku `pars1.cpp`. Rozwiązanie tego typu otrzymywało na zawodach około 50% punktów.

Rozwiązanie wzorcowe $O(n)$

Ukorzeniamy drzewo w dowolnym węźle. Następnie, zaczynając od liści i poruszając się w górę drzewa, dla każdego węzła v wyznaczamy dwie wartości:

$h[v]$ – trudność najtrudniejszej trasy parady zaczynającej się w węźle v i prowadzącej w dół poddrzewa węzła v,

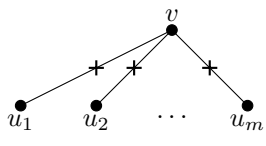
$d[v]$ – trudność najtrudniejszej trasy parady przechodzącej przez v i biegnącej w poddrzewie węzła v.

Jeśli v jest liściem, to oczywiście $h[v] = d[v] = 0$. W ogólnym przypadku, gdy węzeł v ma m synów u_1, u_2, $\ldots$, u_m, dla których obliczyliśmy już wartości $d[u_i]$ i $h[u_i]$, wartości dla węzła v obliczamy z następującej rekurencji:

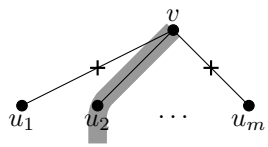
$$h[v] = \max(m, \max_{1 \leqslant i \leqslant m} (h[u_i]) + m - 1)$$

$$d[v] = \max(h[v], \max_{1 \leqslant i < j \leqslant m} (h[u_i] + h[u_j]) + m - 2)$$

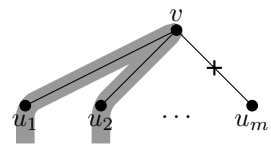
Gdy obliczamy wartość $h[v]$, bierzemy pod uwagę dwie możliwości: albo blokujemy wszystkich synów węzła v (rys. `A`) i trasa parady kończy się w węźle v, albo wybieramy najtrudniejszą trasę parady przechodzącą przez jednego z synów węzła v, blokując przy tym pozostałych synów (rys. `B`). Gdy obliczamy wartość $d[v]$, wybieramy albo trasę parady zaczynającą się w węźle v (co odpowiada wartości $h[v]$), albo połączone dwie trasy parady zaczynające się w synach węzła v (rys. `C`). Na poniższych rysunkach trasę parady zaznaczono kolorem szarym:



rys. `A`



rys. `B`



rys. `C`

Zauważmy, że ostateczny wynik to maksymalna wartość $d[v]$ powiększona o 1, jako że powinniśmy zablokować jeszcze ojca węzła v. Należy tutaj pamiętać o szczególnym przypadku, gdy węzeł v jest korzeniem (wtedy nie dodajemy jedynki).

Wyznaczenie wartości $h[v]$ i $d[v]$ możemy zaimplementować w czasie liniowym od liczby synów m. Faktycznie, wystarczy wyznaczyć maksymalną i drugą co do wielkości wartość $h[u_i]$. Ostatecznie otrzymujemy rozwiązanie działające w czasie liniowym. Przykładową implementację można znaleźć w pliku `par.cpp`.

XXVIII Międzynarodowa Olimpiada Informatyczna,

Kazań, Rosja 2016