

Rekonstrukcja

Limit czasu: 4 sekundy

Limit pamięci: 1024 MiB

To jest zadanie interaktywne.

Michał za chwilę przystąpi do gry orientacyjnej *Meet Kaunas*, w której delegacje EJOI będą konkurować poprzez odwiedzanie miejsc w mieście. Ale Michał bardziej interesuje się tym, *jak* gra została zaprojektowana, niż zwycięstwem.

Jest N miejsc na mapie oraz N zespołów. Miejsca oraz zespoły są oznaczone kolejnymi liczbami od 0 do $N - 1$. Każdy zespół musi odwiedzić wszystkie miejsca i każdy zespół ma unikatowe miejsce startowe — zespół i rozpoczyna od miejsca i . Każdy zespół ma przydzieloną trasę.

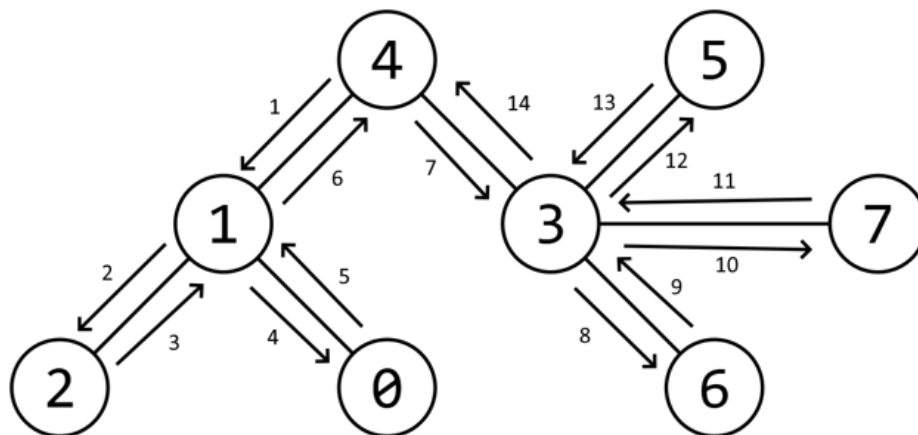
Michał ma również kilka przecieków – trasy opierają się na ukrytej strukturze szybkich linii transportu publicznego. Jury konkurencji wybrało $N - 1$ linii, z których każda łączy parę miejsc na mapie, w taki sposób, że każde miejsce można osiągnąć z każdego innego za pomocą tych linii. Taką strukturę nazywa się drzewem. Następnie, dla każdego miejsca startowego i , jury wygenerowało **pewien** spacer DFS (zdefiniowany poniżej) i przydzieliło ten spacer jako trasę dla zespołu i .

Michał chce odkryć drzewo transportu publicznego, które wybrało jury. Może on zadawać następujące pytania:

- Jakie jest j -te miejsce na trasie zespołu i ?

Pomóż Michałowi, napisz program `find_tree`, który znajduje ukryte drzewo.

Spacer DFS drzewa to kolejność, w której jego wierzchołki są odwiedzane po raz pierwszy przez procedurę przeszukiwania w głąb. Taka procedura rozpoczyna się w danym wierzchołku v i rekurencyjnie przechodzi do jednego z jego nieodwiedzonych sąsiadów, dopóki taki sąsiad istnieje – wówczas powraca do poprzednio odwiedzonego wierzchołka i kontynuuje z niego. Przykład:



Tutaj $v = 4$ i strzałki wzdłuż krawędzi odpowiadają krokom pewnego przeszukiwania w głąb. Uzyskany spacer DFS to $[4, 1, 2, 0, 3, 6, 7, 5]$. Zauważ, że kolejność, w której odwiedzamy sąsiadów podczas procedury, ma znaczenie; innym spacerem DFS mógłby być $[4, 3, 5, 7, 6, 1, 0, 2]$.

Struktura drzewa oraz trasy zespołów zostały ustalone przed uruchomieniem Twojego programu i nie zmieniają się w odpowiedzi na Twoje pytania. Kolejność odwiedzania sąsiadów wierzchołka podczas różnych procedur DFS może być różna.

Szczegóły implementacji

Zaimplementuj następującą funkcję:

```
std::vector> find_tree(int N)
```

gdzie

- N to liczba miejsc;
- wartość zwracana to lista krawędzi drzewa – kolejność krawędzi i ich punktów końcowych nie ma znaczenia.

Dla każdego testu funkcja będzie wywołana maksymalnie T razy.

Udostępniona jest następująca funkcja:

```
int guess(int i, int j)
```

Zwraca j -te miejsce na trasie zespołu i . Zauważ, że `guess(i, 0)` zwróci i . Możesz założyć, że funkcja odpowiada w czasie stałym $O(1)$ dla wszystkich podzadań do 6 włącznie, oraz w czasie logarytmicznym $O(\log N)$ dla podzadania 7. Twoje wywołania funkcji powinny spełniać $0 \leq i, j \leq N - 1$, w przeciwnym razie Twoje rozwiązanie otrzyma werdykt `Output isn't correct: Invalid call`.

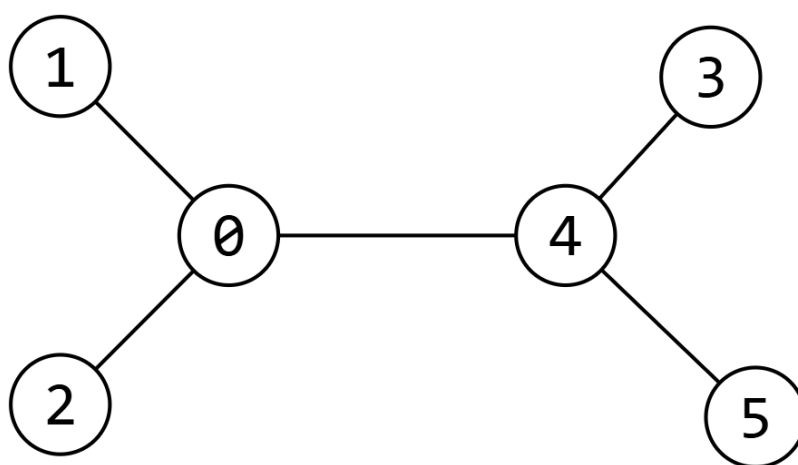
Ograniczenia

- $2 \leq N \leq 2^{16} + 1$
- T podlega ograniczeniom opartym na maksymalnym N spośród wywołań funkcji w jednym teście, oznaczonym przez N_{max} :
 - jeśli $N_{max} \leq 9$, to $1 \leq T \leq 100$
 - jeśli $N_{max} \leq 2^{10} + 1$, to $1 \leq T \leq 10$
 - jeśli $N_{max} \leq 2^{16} + 1$, to $1 \leq T \leq 3$

Program oceniający może użyć do 280 MiB, co wlicza się do zużycia pamięci przez Twoje rozwiązanie.

Przykład

Niech struktura szybkich linii transportu publicznego wygląda następująco:



Niech dla miejsca startowego 0 spacer to $[0, 1, 2, 4, 3, 5]$. Poniżej znajduje się przykład interakcji:

Program uczestnika	Program jury
	<code>find_tree(6)</code>
<code>guess(0, 0)</code>	0
<code>guess(0, 1)</code>	1
<code>guess(0, 2)</code>	2
<code>guess(0, 3)</code>	4
<code>guess(0, 4)</code>	3
<code>guess(0, 5)</code>	5
<code>return {{0,1},{0,2},{4,0},{5,4},{3,4}};</code>	

Zauważ, że informacje z zapytań nie wystarczają do jednoznacznego wyznaczenia drzewa, ale jest to poprawna odpowiedź.

Podzadania

Podzadanie	Punkty	N	Dodatkowe ograniczenia
0	0	-	Przykład z zadania.
1	11	≤ 9	-
2	6	≤ 100	Każdy wierzchołek jest połączony z co najwyżej dwoma innymi.
3	13	≤ 100	Każdy spacer jest uzyskany przez przeszukiwanie w głąb, które na każdym kroku priorytetyzuje oddalanie się od wierzchołka 0. Jeśli istnieje kilka opcji oddalania się od wierzchołka 0, wybierana jest dowolna z nich.
4	11	≤ 100	Każdy wierzchołek inny niż 0 ma co najwyżej dwóch sąsiadów.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Punktacja

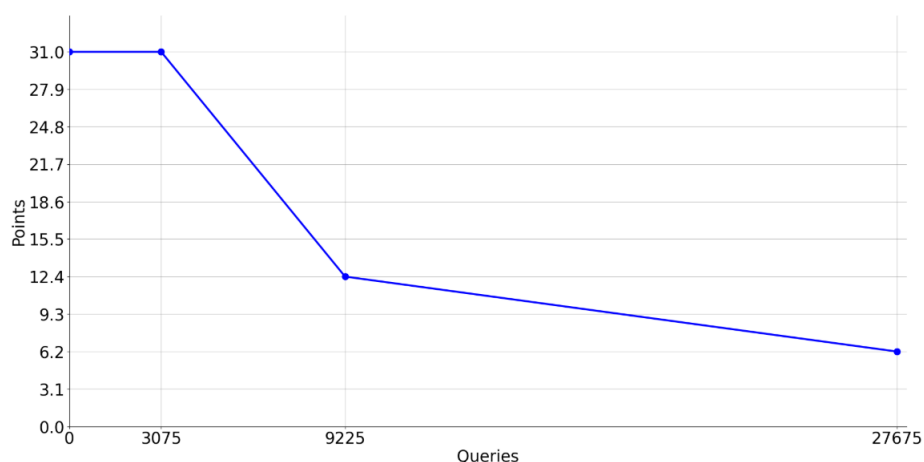
Dla podzadań od 1 do 5 otrzymujesz pełne punkty, jeśli poprawnie znajdziesz drzewo w ramach ograniczeń zadania. Dla podzadań 6 i 7 część S puli punktów dla danego testu zależy od **maksymalnej** liczby zapytań, które zadajesz w jednym z jego podtestów, Q_{max} . Punktacja dla tych podzadań wygląda następująco:

- jeśli $Q_{max} \leq L_1$, to $S = 1.0$;
- jeśli $L_1 < Q_{max} \leq L_2$, to $S = 0.4 + 0.6 \times \frac{L_2 - Q_{max}}{L_2 - L_1}$;
- jeśli $L_2 < Q_{max} \leq 3L_2$, to $S = 0.2 + 0.2 \times \frac{3L_2 - Q_{max}}{2L_2}$;
- jeśli $3L_2 < Q_{max}$, to $S = 0.2$;

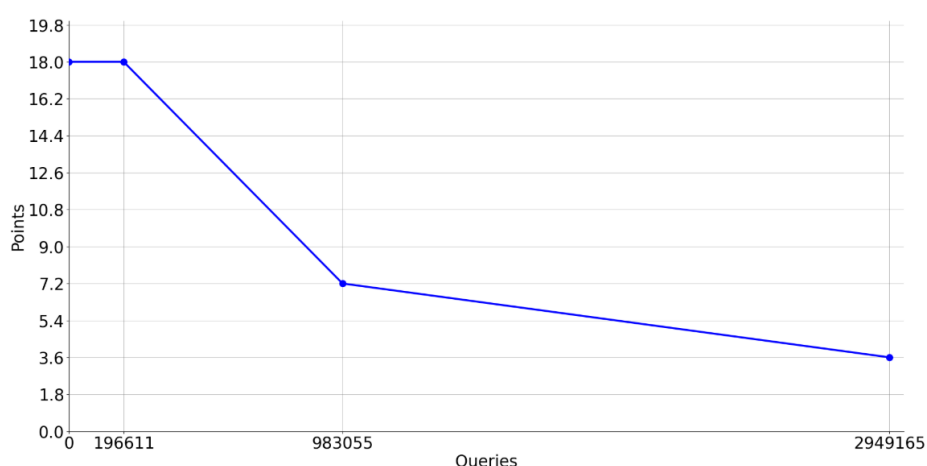
gdzie $L_1 = 3 \times (2^{10} + 1) = 3075$ i $L_2 = 9 \times (2^{10} + 1) = 9225$ dla podzadania 6, oraz $L_1 = 3 \times (2^{16} + 1) = 196611$ i $L_2 = 15 \times (2^{16} + 1) = 983055$ dla podzadania 7. Część punktów, które otrzymasz z danego podzadania, to najmniejsze S spośród wszystkich testów w tym podzadaniu.

Poniższe wykresy pokazują funkcję punktacji dla ostatnich dwóch podzadań:

Podzadanie 6



Podzadanie 7



Przykładowy program oceniający

Dostarczane są dwa przykładowe programy oceniające.

Do testowania lokalnego: możesz skompilować `Lgrader.cpp` wraz z Twoim rozwiązaniem, aby otrzymać narzędzie do testowania. Program czyta liczbę T przypadków testowych i dla każdego z nich wczytuje liczbę miejsc N , a następnie $N - 1$ wierszy, każdy zawierający parę liczb całkowitych opisującą linię. Po tym następuje N wierszy, każdy po N liczb całkowitych – spacer DFS. Zauważ, że spacer i musi rozpoczynać się od wierzchołka i . Możesz pozwolić programowi oceniającemu na wygenerowanie spacerów dla Ciebie, ustawiając stałą `AUTO_GENERATE` na `true`. Na wyjściu program oceniający raportuje komunikat o błędzie, jeśli wynik jest niepoprawny, lub liczbę zapytań zadanych dla każdego przypadku testowego oraz maksimum z nich. Zwróć również uwagę, że program oceniający nie działa dla dużych N (dokładniej dla ograniczeń podzadania 7). W tym przypadku jesteś zdany na własną kreatywność.

Do uruchomień próbnych: możesz użyć programu `stub.cpp` wraz ze swoim rozwiązaniem do uruchomienia próbnego. Format wejścia jest taki sam jak dla lokalnego programu oceniającego.

Dla uruchomień próbnych nie ma wbudowanej opcji automatycznego generowania spacerów.