Task: DRE
Dream

Olimpiada Informatyczna

CPSPC 2019, Day 3. Source file `dre.*` Available memory: 256 MB. *29. 6. 2019*

Last night, Filip had a dream. In this dream, he saw the most beautiful tree he has ever seen. But when he woke up, he could not remember how the tree looked! Naturally, this made Filip very sad. He wanted to see the tree once more! Filip decided to ask his psychotherapist to help him remember his beautiful dream.

After a few intense sessions of psychotherapy, Filip managed to remember that the tree has n vertices numbered 1 to n and each edge $\{u, v\}$ has a weight $w_{uv} \in \mathbb{N}$. The weights are unique and what is even more interesting, they follow a kind of a heap property: the tree is rooted (but Filip can't remember the root) and for each non-root vertex, the weight of the edge to its parent is lower than the weights to its sons. Formally, if v is a non-root vertex, u is its parent and x is its son, then $w_{uv} < w_{vx}$.

But this wasn't enough for Filip. He wants to know all of the tree's edges and their weights. His psychotherapist agreed to further sessions. In each session, Filip can ask about a pair of vertices (u, v) and through Jungian analysis the psychotherapist can retreive from Filip's subconsciousness the numbers (w_{min}, w_{max}), where

$$\begin{aligned} P &= \{e \mid e \text{ is an edge in the tree on the path between } u \text{ and } v\} \\ w_{min} &= \min_{e \in P} w_e \\ w_{max} &= \max_{e \in P} w_e \end{aligned}$$

Naturally, these sessions are expensive. Filip, being a math student, only has money for k of them. Help Filip reconstruct the tree using at most k psychotherapy sessions.

Communication

Your program should use a library which allows asking questions and announcing the final answer. You should add the line `#include "drelib.h"` at the beginning of your program. The library ('drelib.h') provides the following functions:

- `int get_n();`
  Returns the number of vertices.
- `std::pair<int,int> min_max_on_path(int u, int v);`
  Returns (w_{min}, w_{max}): the minimum and maximum weight from the edges on the path between vertices u and v ($u \neq v$). May be called at most k times.
- `void answer_edge(int u, int v, int w);`
  Answers that there is an edge between u and v with weight w. Must be called exactly $n - 1$ times, once for each of the tree's edges. The order of u, v and the edges does not matter.

In this problem, you are not allowed to read from the standard input or to write to the standard output. Your program should call `answer_edge` exactly $n - 1$ times and then terminate.

In some of the test cases, the library actually does not need to choose the tree at the beginning (it may be adaptive), but you can assume that all answers from the library are consistent and there is always a valid answer.

Example interaction

Consider a tree on $n = 5$ vertices, vertex 3 being the root. Let the edges (denoted (u, v, w)) be:
$(2, 3, 5), (1, 2, 7), (2, 4, 8), (1, 5, 11)$.

The following table shows an example interaction with the library that corresponds to this example.

Call	Returned value
`get_n()`	5
`min_max_on_path(5, 4)`	`{11, 7}`
`min_max_on_path(2, 3)`	`{5, 5}`
`min_max_on_path(1, 3)`	`{5, 7}`
`answer_edge(1, 2, 10)`	-
`answer_edge(2, 3, 20)`	-
`answer_edge(5, 4, 30)`	-
`answer_edge(3, 4, 40)`	-

This interaction is valid as all of the queries are within the correct limits and `answer_edge` is called the correct number of times, however the answer is incorrect because the edges submitted by the program are wrong.

Grading

In all tests $2 \leq n \leq 100$. Also for each edge e, $1 \leq w_e \leq 10^9$.

Subtask	Constraints	Points
1	$k = 5000$	10
2	$k = 600$, the root is always vertex 1 and the tree is a *balanced binary tree*	20
3	$k = 600$, the tree is a *balanced binary tree*	20
4	$k = 600$, the root is always vertex 1	25
5	$k = 600$	25

A *balanced binary tree* is a tree whose every vertex has exactly 0 or 2 sons. Additionally, there is a number d such that every leaf's depth (the distance between it and the root) is d or $d + 1$.

Experiments

In the 'Files' section of SIO there is an example library allowing you to test formal correctness of your solution. The library reads a description of the tree from the standard input in the following format:

- in the first line an integer n, the number of vertices,
- in each of the next $n - 1$ lines: three integers u v w_{uv}, an edge between u and v with weight w_{uv}. The following should hold: $1 \leq u, w \leq n; 1 \leq w \leq 10^9$.

The provided library **does not** make all checks of your solution; it does not check the correctness of the solution or the number of queries used. It also does not check the correctness of the input. It is not the same as the (secret) library on the server.

An example input for the library is given in `dre0.in` file.

After `answer_edge` is called for the $n - 1$-th time, the library prints the given answer and the number of calls to `min_max_on_path` to the standard output.

For compiling the solution with the example library you may use the following command:

- **C++**: `g++ -O2 -static drelib.cpp dre.cpp -lm -std=c++11`

The files with the solution and the library should be in the same directory.

Zadanie: DRE
Sen Filipa

Olimpiada Informatyczna

CPSPC 2019, dzień 3. Plik źródłowy `dre.*` Dostępna pamięć: 256 MB. *29.6.2019*

Ostatnio Filipa nawiedził pewien sen, w którym zobaczył on drzewo – najpiękniejsze, jakie kiedykolwiek widział. Ale kiedy się obudził, nie mógł sobie przypomnieć, jak właściwie drzewo wyglądało. To bardzo zasmuciło Filipa, który chciałby zobaczyć swoje drzewo jeszcze raz.

Aby odtworzyć swój sen, Filip postanowił skorzystać z pomocy psychoterapeuty. Po kilku intensywnych sesjach, Filip zdołał przypomnieć sobie, że jego drzewo miało n wierzchołków (ponumerowanych od 1 do n), a każda jego krawędź $\{u, v\}$ miała wagę w_{uv}. Wszystkie wagi były różne, a dodatkowo drzewo miało własność podobną do kopca: miało korzeń (chociaż Filip nie pamięta, w którym wierzchołku) i dla każdego wierzchołka poza korzeniem waga krawędzi od niego do rodzica jest mniejsza od wag krawędzi do jego dzieci. Formalnie, jeśli v jest wierzchołkiem innym niż korzeń, u rodzicem v, zaś x synem v, to $w_{uv} < w_{vx}$.

Filip chciałby jednak znać wszystkie krawędzie drzewa i ich wagi. Zamierza więc przeprowadzić więcej sesji psychoterapeutycznych – na każdej z nich może wybrać dwa wierzchołki (u, v), a psychoterapeuta, używając analizy Junga, wydobędzie z podświadomości Filipa liczby (w_{min}, w_{max}), będące odpowiednio wagą minimalnej i maksymalnej krawędzi na ścieżce między u i v:

$$\begin{aligned} P &= \{e | e \text{ jest krawędzią na ścieżce między } u \text{ i } v\} \\ w_{min} &= \min_{e \in P} w_e \\ w_{max} &= \max_{e \in P} w_e \end{aligned}$$

Oczywiście takie sesje nie są tanie. Filip, będący biednym studentem matematyki, może sobie pozwolić na co najwyżej k sesji. Pomóż mu zrekonstruować drzewo, mieszcząc się w tym budżecie.

Komunikacja

W tym zadaniu Twój program nie powinien czytać ze standardowego wejścia ani pisać na wyjście. Zamiast tego, powinien użyć komunikować się z dostarczoną biblioteką. Musisz w tym celu umieścić w swoim programie linię:

```
#include "drelib.h"
```

Biblioteka ta umożliwia następujące instrukcje:

- `int get_n();`
  Zwraca liczbę wierzchołków.
- `std::pair<int,int> min_max_on_path(int u, int v);`
  Zwraca parę (w_{min}, w_{max}): minimalną i maksymalną wagę na ścieżce między u i v ($u \neq v$). Funkcję tę można wykonać co najwyżej k razy.
- `void answer_edge(int u, int v, int w);`
  Służy do przekazania Twojej odpowiedzi: komunikuje, że jest krawędź między u a v o wadze w. Musisz tę funkcję wykonać dokładnie $n - 1$ razy, dla każdej krawędzi drzewa. Kolejność krawędzi, jak również porządek wierzchołków u i v, nie mają znaczenia.

Twój program powinien wykonać funkcję `answer_edge` dokładnie $n - 1$ razy, a potem się zakończyć. W niektórych testach może się zdarzyć, że testerka będzie *adaptywna* tzn. drzewo będzie konstruowane w czasie wykonania Twojego programu, możesz jednak założyć, że odpowiedzi biblioteki zawsze opisują poprawne drzewo.

Przykładowe wykonanie

Rozważmy drzewo o $n = 5$ wierzchołkach, przy czym wierzchołek 3 jest korzeniem. Jego krawędzie (dane jako trójki (u, v, w)) to:
$(2, 3, 5), (1, 2, 7), (2, 4, 8), (1, 5, 11)$.
Poniższa tabela pokazuje przykładową konwersację z biblioteką dla powyższego drzewa:

Call	Returned value
`get_n()`	5
`min_max_on_path(5, 4)`	`{11, 7}`
`min_max_on_path(2, 3)`	`{5, 5}`
`min_max_on_path(1, 3)`	`{5, 7}`
`answer_edge(1, 2, 10)`	-
`answer_edge(2, 3, 20)`	-
`answer_edge(5, 4, 30)`	-
`answer_edge(3, 4, 40)`	-

W tej interakcji wszystkie funkcje są wywoływane prawidłowo, a funkcja `answer_edge` jest wywołana właściwą liczbę razy, ale podana odpowiedź nie jest prawidłowa – znalezione wagi krawędzi są błędne.

Ocenianie

We wszystkich testach $2 \leq n \leq 100$. Ponadto dla każdej krawędzi e, $1 \leq w_e \leq 10^9$.

Podzadanie	Ograniczenia	Punkty
1	$k = 5000$	10
2	$k = 600$, korzeń to wierzchołek 1 a drzewo jest *binarne, zrównoważone*	20
3	$k = 600$, *zrównoważone drzewo binarne*	20
4	$k = 600$, korzeń to wierzchołek 1	25
5	$k = 600$	25

Zrównoważone drzewo binarne to takie, w którym każdy wierzchołek ma 0 lub 2 dzieci, a ponadto istnieje d takie, że odległość od korzenia do każdego z liści wynosi d lub $d + 1$.

Przykładowa biblioteka

W sekcji „Pliki" na SIO dostępna jest przykładowa biblioteka, dzięki której możesz przetestować swoje rozwiązanie. Biblioteka pobiera swoje drzewo ze standardowego wejścia, w następującym formacie:

- pierwsza linia zawiera n – liczbę wierzchołków,
- kolejne $n - 1$ linii zawiera po trzy liczby u, v, w_{uv}, oznaczające krawędź między u a v o wadze w_{uv}. Zachodzi $1 \leq u, w \leq n; 1 \leq w \leq 10^9$.

Przykładowa biblioteka nie sprawdza, czy Twój program wykonuje poprawnie funkcje. Nie sprawdza też, czy wczytywane wejście jest prawidłowe. Do oceny Twojego rozwiązania będzie wykorzystywana inna biblioteka. Przykładowe wejście do tej biblioteki podane jest w pliku `dre0.in`.

Po $n - 1$ wywołaniu funkcji `answer_edge` biblioteka wypisuje na standardowe wyjście podaną przez Ciebie odpowiedź i podsumowuje, ile razy wykonane zostało wywołanie `min_max_on_path`.

Aby skompilować swoje rozwiązanie z dostarczoną biblioteką, użyj komendy:

- **C++**: `g++ -O2 -static drelib.cpp dre.cpp -lm -std=c++11`

Twoje rozwiązanie powinno być w tym samym katalogu, co biblioteka.

Úloha: DRE
Dream

Olimpiada Informatyczna

CPSPC 2019, Den 3. Zdrojový soubor `dre.*` Dostupná paměť: 256 MB. *29. 6. 2019*

Minulou noc měl Filip sen. Ve snu se díval na ten nejkrásnější strom, jaký kdy viděl, ale když se probudil, nemohl si ani v nejmenším vzpomenout, jak vypadal. Filip byl z toho samozřejmě velmi smutný a chtěl mít tu příležitost strom ještě aspoň jedinkrát moct vidět. Proto se rozhodnul vyhledat pomoct psychoterapeuta, který by mu pomohl vybavit si ten překrásný sen.

Po několika intenzivních sezeních u psychoterapeuta se Filipovi podařilo vzpomenout si, že strom měl n vrcholů očíslovaných od 1 do n a každá hrana $\{u, v\}$ měla váhu $w_{uv} \in \mathbb{N}$ Hrany mají unikátní váhy, a co je zajímavější, platí pro ně jakási haldová podmínka: strom je zakořeněn (ale Filip si nepamatuje číslo kořene) a pro každý nekořenový vrchol platí, že váha hran vedoucích do jeho rodiče je menší než váha hran vedoucí do jeho potomků. Formálně, pokud vrchol v není kořenem, u je jeho rodič a x je některý jeho potomek, pak platí $w_{uv} < w_{vx}$.

To ale Filipovi nestačilo. Chce znát všechny hrany stromu i jejich váhy. Jeho psychoterapeut tedy souhlasil s dalšími sezeními. Na každém sezení se Filip může zeptat na jednu dvojici vrcholů (u, v). a za pomocí Jungovské analýzy terapeut může získat z Filipova podvědomí hodnoty (w_{min}, w_{max}), kde

$$\begin{aligned} P &= \{e \mid e \text{ je hrana strmu na cestě mezi vrcholy } u \text{ a } v\} \\ w_{min} &= \min_{e \in P} w_e \\ w_{max} &= \max_{e \in P} w_e \end{aligned}$$

Přirozeně, tato sezení jsou velmi drahá. Filip, jakožto student matematiky, si jich může dovolit jen k. Pomozte Filipovi zrekonstruovat strom pomocí nejvýše k sezení.

Komunikace

Váš program bude využívat knihovnu, která umožňuje pokládání dotazů a oznamování finální odpovědi. Na začátko svého programu přidejte řádek `#include "drelib.h"`. Knihovna `drelib.h` poskytuje tyto funkce:

- `int get_n();`
  Navrátí počet vrcholů stromu n.
- `std::pair<int,int> min_max_on_path(int u, int v);`
  Navrátí hodnoty (w_{min}, w_{max}): minimální a maximální váhy hran na cestě mezi vrcholy u a v ($u \neq v$). Smí být zavolána nejvýše k krát.
- `void answer_edge(int u, int v, int w);`
  Odpověď systému, že ve stromu se nachází hrana mezi vrcholu u a v s vahou w. Tato funkce musí být zavolána právě $n - 1$ krát, jednou za každou hranu stromu. Na pořadí odpovědí nezáleží.

V této úloze vám není povoleno číst ze standardního vstupu nebo zapisovat do standardního výstupu. Váš program by měl zavolat funkci `answer_edge` přesně $n - 1$ krát a následně skončit.

V některých testovacích vstupech se může stát, že si knihovna nezvolí pevný strom (může své odpovědi přizpůsobovat dotazům), ale můžete předpokládat, že všechny odpovědi jsou konzistentní a vždy existuje validní odpověď.

Příklad interakce

Uvažme strom na $n = 5$ vrcholech s kořenem 3. Dále nechť hrany tohoto stromu ve formátu (u, v, w) jsou: $(2, 3, 5), (1, 2, 7), (2, 4, 8), (1, 5, 11)$.

Následující tabulka popisuje interakci s knihovnou která odpovídá tomuto příkladu.

Volání	Vrácená hodnota
`get_n()`	5
`min_max_on_path(5, 4)`	`{11, 7}`
`min_max_on_path(2, 3)`	`{5, 5}`
`min_max_on_path(1, 3)`	`{5, 7}`
`answer_edge(1, 2, 10)`	-
`answer_edge(2, 3, 20)`	-
`answer_edge(5, 4, 30)`	-
`answer_edge(3, 4, 40)`	-

Tato interakce je korektní, protože všechny dotazy jsou ve správném rozmezí hodnot a funkce `answer_edge` je zavolána právě čtyřikrát. Odpověď je bohužel vyhodnocena jako špatná, protože program odelal nesprávnou podobu stromu.

Limity

Ve všech sadách $2 \leq n \leq 100$. Pro každou hranu e platí $1 \leq w_e \leq 10^9$.

Subtask	Omezení	Body
1	$k = 5000$	10
2	$k = 600$, kořen je vždy vrchol 1 a strom je *vyvážený binární strom*	20
3	$k = 600$, strom je *vyvážený binární strom*	20
4	$k = 600$, kořen je vždy vrchol 1	25
5	$k = 600$	25

Vyvážený binární strom je strom ve kterém každý vrchol má přesně 0 nebo 2 syny a zároveň existuje číslo d takové, že hloubka každého listu (vzdálenost mezi ním a kořenem) je d nebo $d + 1$.

Experimenty

V sekci 'Files' v SIO je ukázková knihovna pomocí které můžete testovat, zda je vaše řešení formálně správně. Knihovna čte popis stromu ze standardního vstupu v tomto formátu:

- na první řádce číslo n, počet vrcholů,
- na každé z dalších $n - 1$ řádek: tři čísla u v w_{uv}, hrana mezi u a v s vahou w_{uv}. Mělo by platit: $1 \leq u, w \leq n; 1 \leq w \leq 10^9$.

Poskytnutá knihovna **nekontroluje** vaše řešení všemi způsoby; nekontroluje, zda je odpověď správná, ani počet použitých dotazů. Nekontroluje ani správnost vstupu. Není stejná jako (tajná) knihovna na serveru.

Příklad vstupu pro tuto knihovnu je v souboru `dre0.in`.

Poté, co zavoláte `answer_edge` po $n - 1$-té, knihovna vypíše vaší odpověď a počet použitých dotazů na standardní výstup.

Na kompilování řešení s ukázkovou knihovnou můžete použít tento příkaz:

- **C++**: `g++ -O2 -static drelib.cpp dre.cpp -lm -std=c++11`

Soubory s řešením a knihovnou by měly být ve stejné složce.