

Reconstruct

Временско ограничење: 4 секунде

Меморијско ограничење: 1024 MiB

Ово је интерактивни задатак.

Бисерка улази у игру оријентиринга организовану од стране волонтера *Крајња дестинација*, у којој ће се делегације на ЕЈОИ-у такмичити у посећивању различитих локација у граду. Ипак, Бисерку више интересује *како* је игра направљена него да у њој победи.

На мапи постоји N локација и N тимова. Сваки тим мора да посети све локације и сваки тим има засебну почетну локацију — тим i креће са локације i . Сваки тим има своју руту.

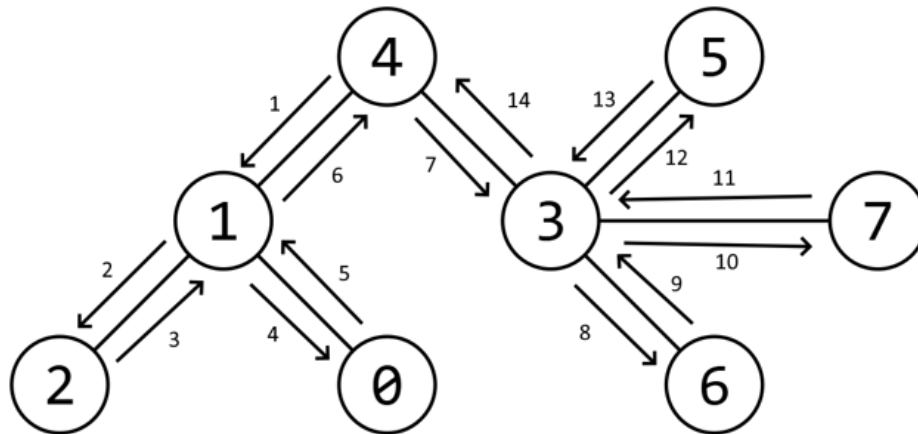
Бисерка такође располаже инсајдерским информацијама — руте су засноване на скривеној структури брзих линија јавног превоза. Прецизније, жири такмичења је изабрао $N - 1$ линију, од којих свака повезује две локације тако да се од сваке локације може стићи до било које друге коришћењем ових линија. Оваква структура се још назива и стабло. Потом, за сваку почетну локацију i , жири је генерисао **произвољну** *DFS шетњу* (дефинисано у наставку) и поставио ову шетњу за руту тима i .

Бисерка жели да пронађе стабло јавног превоза које је жири изабрао. Она поставља само кратка питања следеће врсте:

- Која је j -та дестинација i -тог тима?

Помозите Бисерки да напише програм `find_tree` који проналази скривено стабло.

DFS шетња у стаблу представља редослед по којим се по први пут посећују чворови током извршавања алгоритма DFS. Овај алгоритам почиње у датом чвору v и рекурзивно обилази једног од суседа чвора који није раније био посећен све док такви суседи постоје — потом се враћа у претходни чвор и наставља претрагу из њега. Пример:



Овде је $v = 4$ и стрелице поред грана одговарају корацима у DFS-у. Генерисана шетња је $[4, 1, 2, 0, 3, 6, 7, 5]$. Приметимо да је редослед по којим се обилазе суседи битан; још једна шетња може да буде $[4, 3, 5, 7, 6, 1, 0, 2]$.

Стабло и N DFS шетњи су фиксирани пре него што твој програм крене да се извршава и не мењају се током одговарања на питања. Редоследи суседа чворова коришћени током ових DFS-ова могу да се разликују.

Детаљи имплементације

Треба да имплементираш следећу функцију:

```
std::vector<std::pair<int, int>> find_tree(int N)
```

- овде је N број локација;
- тип повратне вредности је листа грана из стабла — редослед грана и њихових крајева није битан.

За сваки тест ова функција може да се позове до T пута.

Да би интераговао са жиријем, можеш да позиваш следећу функцију:

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

Она враћа j -ту локацију у i -тој шетњи стаблом. Приметимо да `guess(i, 0)` враћа i . Можеш да претпоставиш да функција одговара у константном времену $O(1)$ за све подзадатке до 6, и у логаритамском времену $O(\log N)$ за подзатак 7. Треба да важи $0 \leq i, j \leq N - 1$, иначе ће твоје решење добити одговор `Output isn't correct: Invalid call`.

Ограничења

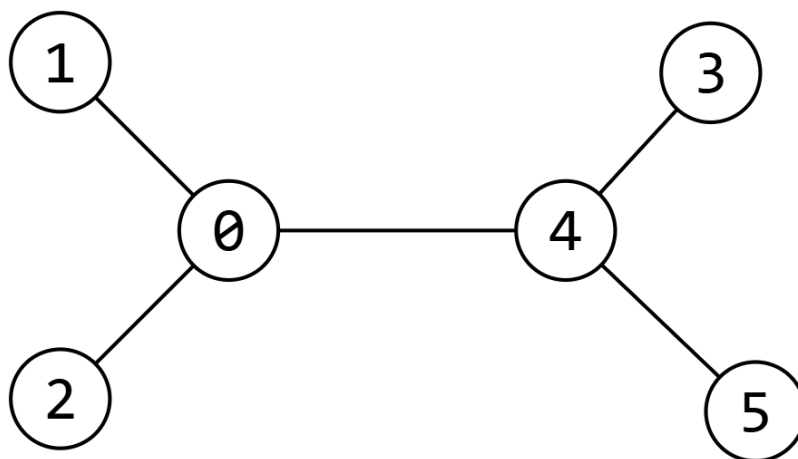
- $2 \leq N \leq 2^{16} + 1$

- T прати ограничења у зависности од максималне вредности N (у ознаци N_{max}) у свим позивима функције у једном тесту:
 - ако је $N_{max} \leq 9$, онда је $1 \leq T \leq 100$
 - ако је $N_{max} \leq 2^{10} + 1$, онда је $1 \leq T \leq 10$
 - ако је $N_{max} \leq 2^{16} + 1$, онда је $1 \leq T \leq 3$

Оцењивач може да користи до 280 MB, што се рачуна у меморију твог решења.

Пример

Претпоставимо да је скривена структура брзих линија јавног превоза следећа:



Претпоставимо да је за почетну локацију 0 шетња $[0, 1, 2, 4, 3, 5]$. Тада је пример интеракције следећи:

Учесников програм	Жиријев програм
	<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>	

Приметимо да информације из упита нису довољне да би се стабло једнозначно одредило, али је ово заправо решење примера из подзадатка 0.

Подзадаци

Подзадатак	Поени	N	Додатна ограничења
0	0	-	Пример.
1	11	≤ 9	-
2	6	≤ 100	Сваки чвор је повезан са још највише 2 чвора.
3	13	≤ 100	Свака шетња је генерисана DFS-ом који у сваком кораку даје приоритет удаљавању од чвора 0. Ако постоји више могућности да се удаљи од чвора 0, једна од њих се бира произвољно.
4	11	≤ 100	Сви чворови осим чвора 0 су повезани са још највише 2 чвора.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Бодовање

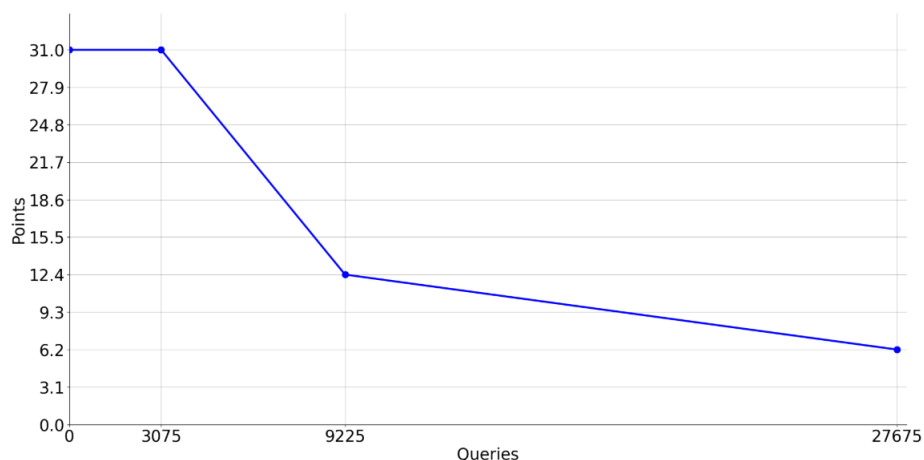
За сваки подзадатак до подзадатка 5, добијаш све поене ако успешно пронађеш стабло у задатим временским ограничењима. За подзадатке 6 и 7, део од укупног броја поена S за дати тест зависи од **максимума** броја упита које поставиш на једном или више подтестова, Q_{max} . Бодовање за ове подзадатке је следеће:

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

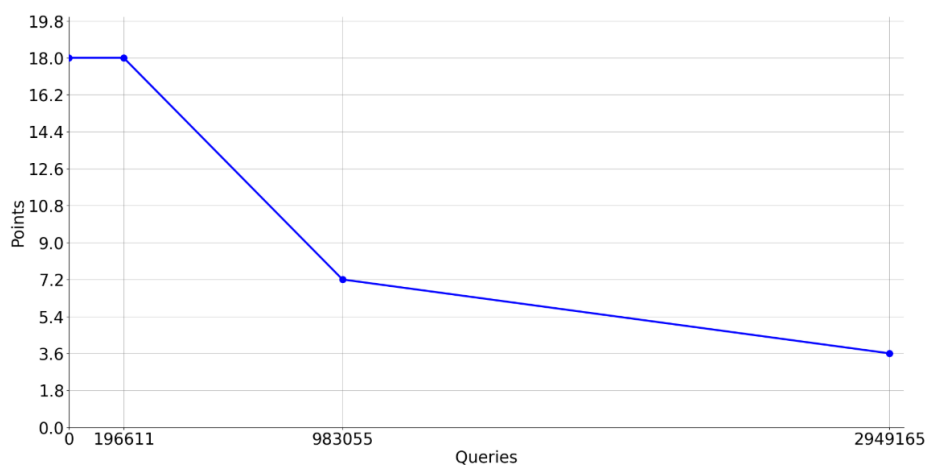
где је $L_1 = 3 \times (2^{10} + 1) = 3075$ и $L_2 = 9 \times (2^{10} + 1) = 9225$ за подзадатак 6, и $L_1 = 3 \times (2^{16} + 1) = 196611$ и $L_2 = 15 \times (2^{16} + 1) = 983055$ за подзадатак 7. Део поена које добијаш на подзадатку је минимално S на свим тестовима.

Графици испод показују функцију оцењивања за последња два подзадатка:

Подзадатак 6



Подзадатак 7



Пример грејдера

Дата су два примера грејдера.

За локално тестирање: `Lgrader.cpp` може да се компајлује заједно са твојим програмом као алат за тестирање. Програм учитава број T тест примера и за сваки од њих очекује број локација N праћен са $N - 1$ линијом, где се у свакој од њих налази пар целих бројева који описују једну грану. Након тога следи N линија са N целих бројева – DFS шетње. Приметимо да шетња i мора да крене из чвора i . Грејдер може да генерише шетње за тебе тако што ћеш поставити константу `AUTO_GENERATE` на `true`. Грејдер на излазу или пријављује грешку ако резултат није тачан или број упита за сваки тест пример и максимални број упита. Приметимо да грејдер не ради за довољно велико N (прецизније за ограничења у подзадатку 7), слободно користи своју машту за тај случај.

За системско тестирање: `stub.cpp` може да се користи заједно са твојим програмом за корисничке тестове на систему. Формат улаза је исти као и за претходни грејдер. Приметимо да за корисничке тестове не постоји уграђена опција за аутоматско генерисање шетњи.