Срещи

Планински район има N села, номерирани с числата от 1 до N, 2 < N < 100000, разположение по склоновете и един неголям град в подножието на планината, номериран с 0. За жителите на района не е никак лесно да комуникират помежду си, защото пътната мрежа е слабо развита. От всяко село излиза само един път, който върви надолу към най-близкото село (или към града), а към всяко село и града се спускат не повече от два пътя. Ако жител на населеното място X и жител на населеното място Y искат да се срещнат, те трябва да тръгнат надолу към града, докато стигнат до населено място Z, което е по пътя от X до града и по пътя от Y до града. Разбира се, колкото по-рано се стигне до мястото на срещата, толкова по-добре. Затова двамата биха искали да знаят предварително мястото на срещата. Напишете програма, която по зададени X и Y намира кое е най-близкото такова населено място Z.

Входа съдържа няколко тестови примера. На първия ред на поредния тестов пример ще бъдат зададени две цели положителни числа N и M, разделени с един интервал, 0 < M < 100. На всеки от следващите N реда ще бъде зададено по едно цяло положително число между 0 и N. На J-ия от тези редове ще бъде зададен номера на най-близкото до J населено място надолу по пътя към града. Всеки от останалите M реда, за поредния тестов пример, ще съдържа по две различни цели числа X и Y (със стойности между 0 и N), разделени с интервал – номерата на населени места, за които се търси най-близкото до тях населено място Z. Край на входа е -1.

За всяка от M-те зададени двойки X и Y, програмата трябва да изведе на отделен ред намереното Z.

Пример:

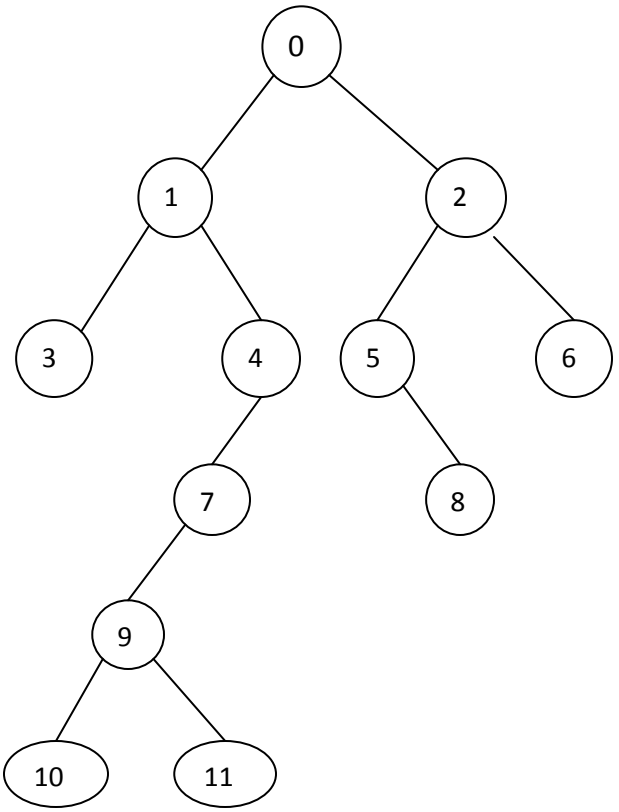
Вход	Изход
5 3 0 5 2 5 0 3 5 1 3 5 3 -1	5 0 5

За решението на тази задача се използва алгоритъма LCA (**L**owest **C**ommon **A**ncestor или „Най-близък общ предшесвеник“). Чистата форма на алгоритъма отговаря на следния въпрос:

Кой е най-близкия общ предшественик на два върха u и v принадлежащи към кореново дърво T?

Има различни методи за намиране на LCA, всички от които имат някаква форма на преизчисляване на таблица от предшественици за всеки връх, която ни позволява след това да отговаряме бързо на запитвания за общия предшественик на произволни два върха. Ние ще разгледаме вариант на алгоритъма, който има сложност O(n log(n)) за преизчисление и O(log(n)) за отговор на запитване. Тук е добре да кажем, че има смисъл от алгоритъма само при достатъчно много запитвания, защото за малък брой, намирането на пътя до корена от всеки от върховете и намиране на първия(най-висока дълбочина в дървото) общ връх от сечението на двата пътя ще даде по-добро време.

За да осигурим бързи отговори за запитванията ще извършим едно обхождане в ширина на дървото тръгвайки от корена. При посещението на всеки връх ще пресмятаме неговата дълбочина и ще пазим списък от предшествениците му, чиято дълбочина се дели точно на степените на двойката от 0 до log(n-1). Нека илюстрираме с 1 пример. Имаме следното дърво:



parents[u][i]	2^0	2^1	2^2	2^3
1	0	0	0	0
2	0	0	0	0
3	1	0	0	0
4	1	0	0	0
5	2	0	0	0
6	2	0	0	0
7	4	4	0	0
8	5	5	0	0
9	7	4	0	0
10	9	9	9	0
11	9	9	9	0

Таблицата level се попълва със следната формула при обхождането: level[child] = 1 + level[parent], като за корена се слага 0. След като вече имаме таблиците parents и level попълнени можем да продължим към същинската част на алгоритъма – заявките.

За да работи правилно алгоритъма е нужно първо да изравним нивата на върховете, чиито общ предшественик търсим. Тази стъпка е необходима, за да работи втората част от алгоритъма за търсене. Ако двата върха не са на едно и също ниво, то тогава от нивото на по-ниския връх до нивото на по-високия ще има междинни върхове по пътя към корена и е възможно съответните прародители да не съвпадат.

След като вече имаме два върха, които са на едно и също ниво в дървото, търсим техния общ прародител. Ако върховете съвпадат, задачата е решена. Иначе търсим първия им необщ прародител по аналогичен на вдигането на по-ниския връх начин. Така стигаме до два върха, чийто пряк родител е най-близкия общ предшественик на първоначалните върхове. Той е върхът, който търсим. Ако не ви стане ясно от тук, може да погледнете кода и да се върнете.

Примерно проиграване за lca(u: 9, v: 8, p: 3) би изглеждало така:

Изпълнение на lift(u, v, p)
    for i = 3 … 0 :
        candidate = parents[u][i]
        is level[candidate] == level[v] → yes → return 7
Изпълнение на lca(9, 8, 3)
    Level[9] = 4, level[8] = 3 => lift(u, v, p)

    for i = 3 … 0 :
        i = 3 is parents[7][3] != parents[7][3] → no

        I = 2 is parents[7][2] != parents[7][2] → no

        I = 1 is parents[7][1] != parents[7][1] → yes

            U(7) = 4, v(8) = 5

        I = 0 is parents[7][0] != parents[7][0] → yes

            U(4) = 1, v(5) = 2

    Lca(9, 8, 4) = parents[u][0] = 0

Друга задача, която може да се реши със LCA без почти никаква промяна е да кажем не кой е най-близкият общ предшественик, а колко е разстоянието между всеки два върха в дървото. В случая когато нямаме тегла, т.е. теглата са равни на 1, тогава се използва разликата в дълбочината в дървото. Когато имаме тегла в стъпката „преизчисляване“ на алгоритъма ще трябва да включим и преизчисляването на таблица dist, в която ще пазим разстоянието от всеки връх до корена на дървото. То се изчислява толкова тривиално, колкото дълбочината, а именно разстоянието от корена до родителя на текущия връх плюс разстоянието от текущия връх до родителя(w):

```
dist[c] = dist[parent] + w
```

След като преизчислим тази таблицата отговора на въроса за разстоянието между u и v е равен на:

```
ans = dist[u] + dist[v] - 2*dist[lca]
```

Реализациите на 2те задачи ги има в zip файла заедно с тестовете за “Срещи“, а другата с пътя между върховете може да я решите в Тимус.

Bibliography

LCA. (n.d.). Retrieved from Състезателено програмиране ФМИ:
http://judge.openfmi.net:9080/mediawiki/index.php/LCA

Range Minimum Query and Lowest Common Ancestor. (n.d.). Retrieved from Top Coder:
http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor