Week 6: Lecture 11 - Lecture 12

#tree

Lecture 11

Storage of trees

(1) adjacency list: vector <vector <int>> neighbors

DFS on a tree: to avoid a loop, pass the parent in recursion to avoid re-visiting the parent. DFS(child, cur).

Example problems

(1) compute the depth of every node. The root of the entire tree should have depth = 0

(2) CSES problem: subordinates

(3) Linova and Kingdom Problem - 1336A - Codeforces

Idea of mentor:

Red: tourism city; yellow: industry city

Observation: if a node is a tourism city, the parent of the node is also a tourism city. (Greedy property). *Otherwise, by switching the order of child node and parent node, a more optimal result can be achieved.*

Algo:

- Find out the change (delta) of marking each node red. Delta = +contribution - loss. Contribution = benefits all subtrees by 1, loss = reduces current node's contribution by depth[node]

- Sort the changes and take K of them

My idea:

Traverse the tree, store by path length, put node in a priority queue sort by path length. (*Actually this doesn't work, because the parent of a leaf node could be added later so that will change the length of the path*).

Pop the priority queue K times, sum the result contributed from each node

Lecture 12

Problem 1

CSES - Company Queries I

Brute force approach: O(n^2)

If each node's jump of powers of 2 is known, complexity can be reduced to logarithmic.

(~binary exponentiation)

Ans: for each node, store the powers of 2 levels (node at 2^0, 2^1, 2^2, etc), find parent of parent through recursive function

f(x) = f(f(x))

Then update the node through binary lifting, this cuts runtime to O(n lgn)

code: cp24/cq1.cpp at master · TheSithPadawan/cp24 · GitHub

Problem 2

CSES - Company Queries II

for nodes a, b:

First move them to the same depth d through binary lifting as problem 1~ O(lg d)

Then apply binary search on distance of LCA , check after jump of distance if a, b have reached the same node. If so, lower the range; else, up the range. This approach takes O(q * lg n * lg n)

// first not so optimized version

```
step 1: index each node by depth, this is done by DFS
step 2: for each query LCA (a, b)
LCA distance range is [0, current depth-1] where 0 is the overall root
```

code: cp24/cq2.cpp at master · TheSithPadawan/cp24 · GitHub

Optimized method. Takes O(q * lg n)

Use the table created by binary lifting to find the children of LCA

```
step 1: move node a and node b to the same depth
step 2: for each query LCA, let up(x, p) represents x's ancester at p levels
above. Iterate at p from LOG -> 0 If up(a, p) != up (b, p) move a and b up.
Eventually up (a, p*) == up(b, p*)
step 3: return up(a, 0). The parent of LCA child is the LCA.
```

code: cp24/cq2.2.cpp at master · TheSithPadawan/cp24 · GitHub