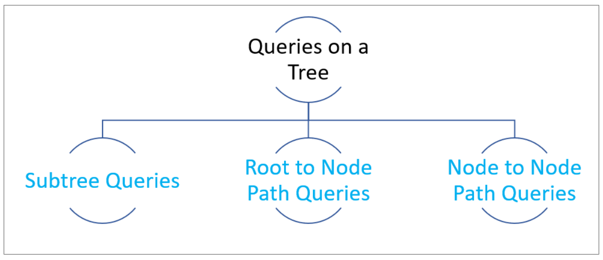
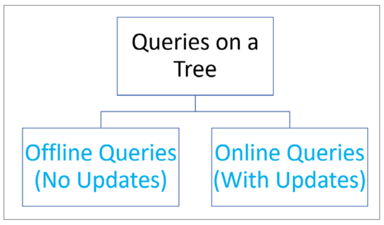
18 Queries on a Tree

Introduction

Problem Types





Some Useful Algorithms and Data Structures to Answer Queries on a Tree

- Depth First Search (DFS)
- Breadth First Search (BFS)
- Euler Tour
- LCA (Least Common Ancestor)
- Ancestor-Descendant Relationship
- Sparse Table
- Mo’s Algorithm
- SQRT Decomposition
- Fenwick Tree (BIT)
- DSU
- Heavy-Light Decomposition (HLD) + Segment Tree (Platinum Level)

Example 1: Offline Subtree Sum Queries

Answer *Q* ($1 <= Q <= 10^5$) subtree sum queries on a tree with *N* ($1 <= N <= 10^5$) nodes. Nodes are conveniently numbered 1 … *N* and node 1 is the root. *Node i* has the value V_i ($1 <= Vi <= 100$). For each `sum(u)` query print sum of node values of the subtree rooted by node *u*.

Sample Input

```
5 3             //5 nodes and 3 queries.
1 3 5 7 9       //Values of the nodes starting with node 1.
1 2             //Node 1 and node 2 are directly connected
1 3
5 3
4 3
3 4
2               //Queries start here
3
1
```

Sample Output

```
3 21 25
```

Solution

Pre-calculate the sum of each subtree in *O(N)* time with DFS and answer each query in *O(1)* time.

Euler Tour on a Tree

Euler Tour is a technique to linearize a tree into an array. Once we get the array representation of the tree, we can use range query methods to answer the tree queries. Euler Tour is the DFS traversal of a tree where we add the nodes into an array every time, we visit them in DFS order. In some cases, we can add them twice: at the first visit (in-time) and at the last visit (out-time). Sometimes may add the leaves only once. The DFS order traversal made it possible to represent any subtree as a contiguous range in an array.

Euler tour tree (ETT) is a method for representing a rooted undirected tree as a number sequence. There are several common ways to build this representation:

The first way is to write down all edges of the tree, directed, in order of DFS.

```
     1
    / \
   2   5
  / \
 3   4

[1-2] [2-3] [3-2] [2-4] [4-2] [2-1] [1-5] [5-1]
```

The second way is to store vertices. Each vertex is added to the array twice: when we descend into it and when we leave it. Each leaf (except maybe root) has two consecutive entries.

```
      1
     / \
    2   5
   / \
  3   4

 1 2 3 3 4 4 2 5 5 1
```

The third way implies storing vertices too, but now each vertex is added every time when we visit it (when descending from parent and when returning from child). Allocate 4*N elements for the Euler Tour array.

```
      1
     / \
    2   5
   / \
  3   4

 1 2 3 2 4 2 1 5 1
```

How these representations can be used?

First, you should notice that a subtree is always mapped to a segment (range). Thus, you can solve subtree queries via range queries. E.g., if you want to add values to nodes and find subtree sum, you just make a BIT on a 2nd type tour. `add(v, x)` becomes `add(first(v), x)` in BIT, and `get_sum(v)` becomes `get_sum(first(v), last(v))` in BIT. There is also a trick to compute sum on path: `add(v, x)` becomes `add(first(v), x), add(last(v), - x)` in BIT, and `get_sum_on_path_from_root(v)` is `get_sum(0, first(v))` (you can check that it works using pencil and paper).

You can also use ETT for finding **LCA**. Use the 3rd type. Make an additional array `h`, where `h[i] = height(v[i])`. Then `lca(u, v) = v[argmin(first(u), first(v)) in h]`. Why? Because `LCA(u, v)` is a highest vertex which is between u and v in DFS traverse order. Sometimes I prefer this method to binary accent (not sure if translated correctly) because it proved to be faster and consumes linear memory.

Example 2: Online Subtree Sum Queries

Handle Q ($1 <= Q <= 10^5$) subtree sum queries and point update queries on a tree with N ($1 <= N <= 10^5$) nodes. Nodes are conveniently numbered 1 … N and node 1 is the root. In an update query, add the given value x ($-100 <= x <= 100$) to the target node and in a sum query, print the sum of the node values in the target subtree. At the beginning *Node i* has the value V_i ($-100 <= Vi <= 100$).

Sample Input

```
5 3                   //5 nodes and 3 queries.
-1 3 -2 5 1           //Values of the nodes starting with node 1.
```

```
1 2                     //Node 1 and node 2 are directly connected
1 3
5 3
4 3
3 4
5 3                     //update query: add 3 to the node 5
3                       //sum query: print sum of the subtree 3
2
```

Sample Output

```
7 3
```

Solution 1: Brute Force 1

- Update each nod in O(1) time.
- For each sum query, calculate the sum of the target subtree in *O(N)* time.

Solution 1: Brute Force 2

- Pre-calculate the sum of each subtree in *O(N)* time with DFS.
- For each update query, update the target node and all its parents up to the root in *O(N)* time.
- Answer sum queries in O(1) time.

Solution 3: Euler Tour + BIT

- Create the Euler Tour of the tree starting with the root in *O(N)* time. You should have two copies of each node in the Euler Tour array: the first visit (in-time) and the last visit (out-time).
- Construct a BIT for 2*N elements and initialize in-time positions with the initial values of the nodes. Set all out-time positions to zeros. It takes *O(N)* or *O(NlgN)* time.
- For any update query, update the target node in in-time position in *O(lgN)* time.
- For any sum query, calculate the range sum in the BIT array between the in-time and out-time positions of the target node in *O(lgN)* time.

Exercise 1: Online Subtree Min Queries

Handle *Q* ($1 <= Q <= 10^5$) subtree min queries and point update queries on a tree with *N* ($1 <= N <= 10^5$) nodes. Nodes are conveniently numbered 1 … *N* and node 1 is the root. In an update query, add the given value *x* ($-100 <= x <= 100$) to the target node and in a min query, print the minimum value in the target subtree. At the beginning *Node i* has the value V_i ($-100 <= Vi <= 100$).

Hint: *Euler Tour + SQRT decomposition*

Sample Input

```
5 3                     //5 nodes and 3 queries.
-1 3 -2 5 1             //Values of the nodes starting with node 1.
1 2                     //Node 1 and node 2 are directly connected
```

```
1 3
5 3
4 3
3 4
5 3                     //update query: add 3 to the node 5
3                       //min query: print min value of the subtree 3
2
```

Sample Output

```
-2 3
```

Exercise 2: Offline Subtree Distinct Values Queries

Answer *Q* ($1 <= Q <= 10^5$) subtree distinct value queries on a tree with *N* ($1 <= N <= 10^5$) nodes. Nodes are conveniently numbered 1 … *N* and node 1 is the root. *Node i* has the value V_i ($0 <= V_i <= 10{,}000$).

Solve this problem in two ways:

Sol 1: Construct a set of every subtree in $O(N\lg^2 N)$ time and answer each query in *O(1)* time. You should always merge a smaller set into a larger set.

Sol 2: Euler Tour + Mo's Algorithm in *O(Nsqrt(N))* time.

Sample Input

```
5 3                     //5 nodes and 3 queries.
2 1 3 2 3               //Values of the nodes starting with node 1.
1 2                     //Node 1 and node 2 are directly connected
1 3
5 3
4 3
3 4
1
2
3
```

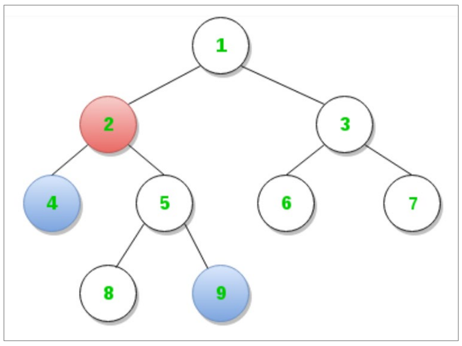
Sample Output

```
3 1 2
```

Example 3: Least Common Ancestor (LCA) Queries

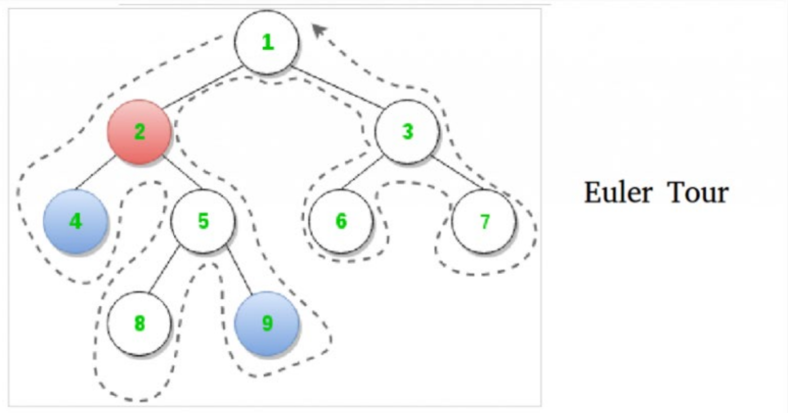
Lowest Common Ancestor (LCA) of two nodes u and v in a rooted tree T is defined as the node located farthest from the root that has both u and v as descendants.

For example, in below diagram, LCA of node 4 and node 9 is node 2.



Algorithm to answer several LCA queries of pairs of nodes of a tree:

- Do a Euler tour on the tree, and fill the *Euler*, *level* and *first occurre*nce arrays.
- Using the *first occurrence* array, get the indices corresponding to the two nodes which will be the corners of the range in the *level* array that is fed to the RMQ algorithm for the minimum value.
- Once the algorithm returns the index of the minimum level in the range, we use it to determine the LCA using Euler tour array.



An euler tour of the tree starting from node 1 will yield:

1	2	4	2	5	8	5	9	5	2	1	3	6	3	7	3	1

The corresponding levels for every node in Euler tour:

0	1	2	1	2	3	2	3	2	1	0	1	2	1	2	1	0

The first occurrences corresponding to every node in Euler tour of T:

Node	1	2	3	4	5	6	7	8	9
First Occurrence	0	1	11	2	4	12	14	5	7

Constructing the arrays takes *O(N)* time. Building a sparse table to answer the range minimum queries takes *O(NlgN)* time. Answering each query with the sparse table takes *O(1)* time. Total time complexity for N nodes and Q queries is *O(NlgN + Q)*.

Exercise 3: Ancestor-Descendant Relationship Queries

An ancestor node of a node is any node in the path from that node to the root node (including the root node). The immediate ancestor of a node is the "parent" node.

A descendant node of a node is any node in the path from that node to the leaf node (including the leaf node). The immediate descendant of a node is the "child" node.

Answer Q ancestor-descendant relationship queries on a tree with N nodes. Nodes are conveniently numbered 1 … N and node 1 is the root. For each query (u, v) print "YES" if u is an ancestor of v, print "NO" otherwise. N=4 and Q=4 in the sample input.

INPUT FORMAT

Line 1: N and Q

Line 2 ... N+1: Each line denotes an edge with two integers u and v (1 <= u, v <= N and u != v).

OUTPUT FORMAT

"YES" or "NO" for each query in the same order with the input.

SAMPLE INPUT

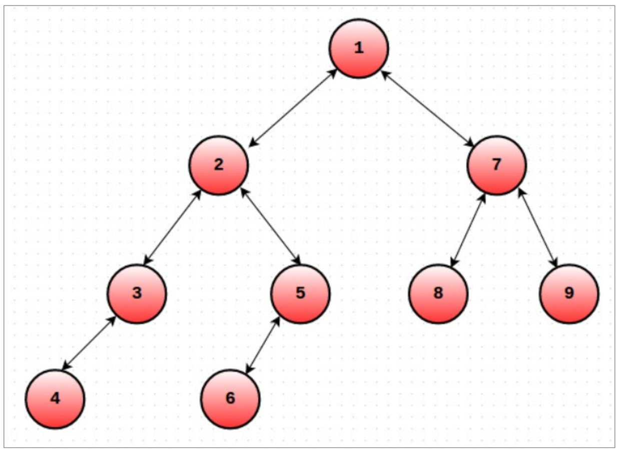
```
5 4
1 2
1 3
5 3
4 3
3 4
2 4
5 1
1 5
```

SAMPLE OUTPUT

```
YES
NO
NO
YES
```

Answering the Offline Path Sum Queries

Answer Q ($1 <= Q <= 10^5$) path sum queries on a tree with N ($1 <= N <= 10^5$) nodes. Nodes are conveniently numbered 1 … N and node 1 is the root. Node i has the value Vi (-100 <= Vi <= 1000).



Algorithm:

- Construct the Euler Tour Traversal array where you store the in-time and out-time position of each node.
- Write values of nodes in in-time positions and -values of nodes in out-time positions.
- The sum of the values from the root to node u: sum(u) is the prefix sum of the TTT array to the in-time position of node u in the TTT array. `sum(u) = presum[intime[u]]`
- The path sum query between the node u and v: `sum(u, v) = sum(u) + sum(v) – sum(lca(u,v)) + value[lca(u,v)].` We just add two paths from the root the nodes *u* and *v*, and subtract the common part.

Useful Links and References

https://codeforces.com/blog/entry/18369
https://codeforces.com/blog/entry/43230
https://www.geeksforgeeks.org/find-lca-in-binary-tree-using-rmq/