Codechef Learn, Episode 1
Lecture-1 Edge Decomposition Tree

https://youtu.be/D63l9u1-nBI

tanujkhattar@

TULIPS - Problem Statement

- https://www.codechef.com/problems/TULIPS
- Need to support queries / updates on set of nodes which are reachable from a given node x by only traversing edges with lengths <= k.

General approach to solve a Query on Tree problem

- Find a way to linearize the tree (eg: by ETT, HLD etc.) such that Query / Updates on a tree reduces to Query / Updates on an array.
- Use one of the standard data structure techniques (eg: Segment Trees, Square Root Decomposition etc.) to solve the Query on Array problem.

How to reduce a problem to a Query on Array problem?

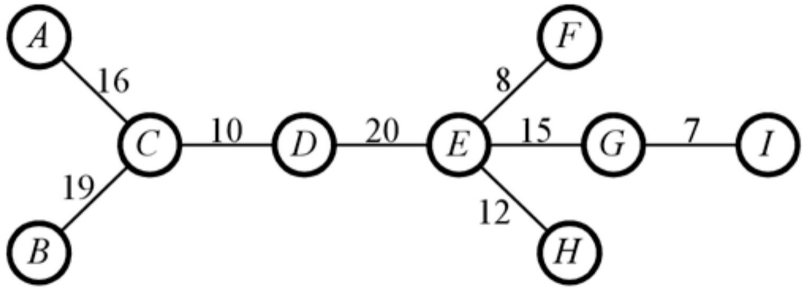
- HLD - Heavy Light Decomposition
  - Linearizes the tree into an HLD array such that a path query / update on the tree is reduced to O(logN) different range query / updates on a linear array.
- ETT - Euler Tour Technique
  - Linearizes the tree into an ETT array such that a subtree query / update on the tree is reduced to a single range query / update on the linear array.
- Combining ETT & HLD
  - Since both are done via DFS, there's a smart technique to combine ETT & HLD into a single linear array which supports both path and subtree query / updates.

How to reduce a problem to a Query on Array problem?

- Need to support queries / updates on set of nodes which are reachable from a given node x by only traversing edges with lengths <= k.
  - How we do this??

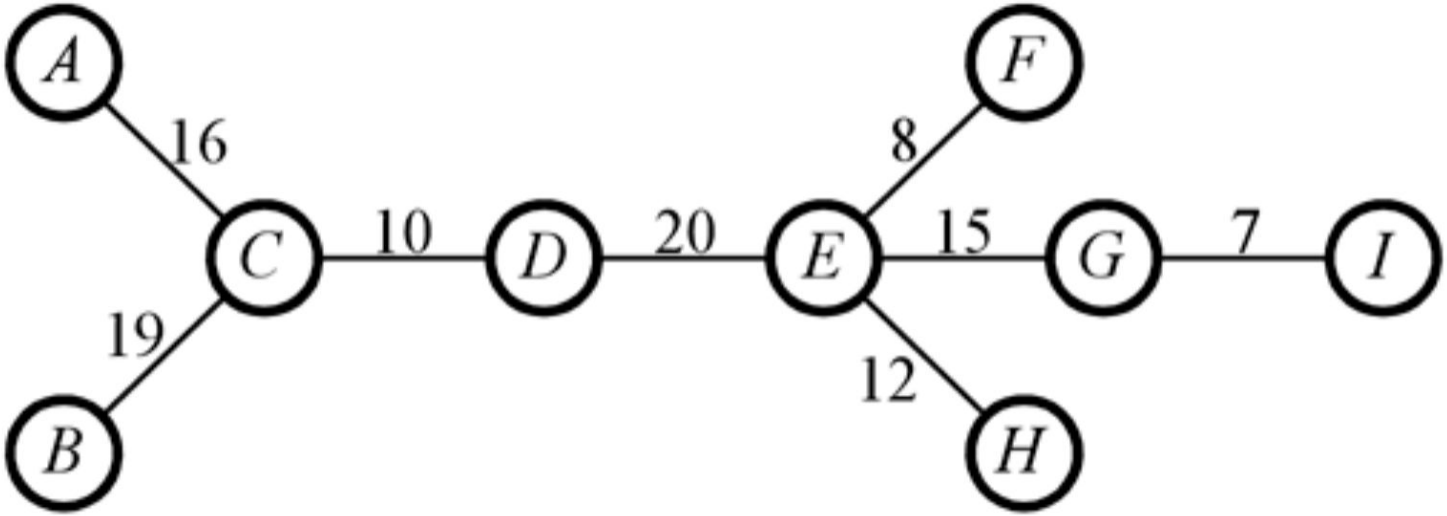
Motivating Problem

- **Given a weighted tree with N nodes, support Q operations of the form:**
  - **Q u k : Tell the sum of values of all nodes reachable from node u by only traversing edges with weights <= k.**
  - **U u add : Add “add” to values of all nodes reachable from node u by only traversing edges with weights <= k.**



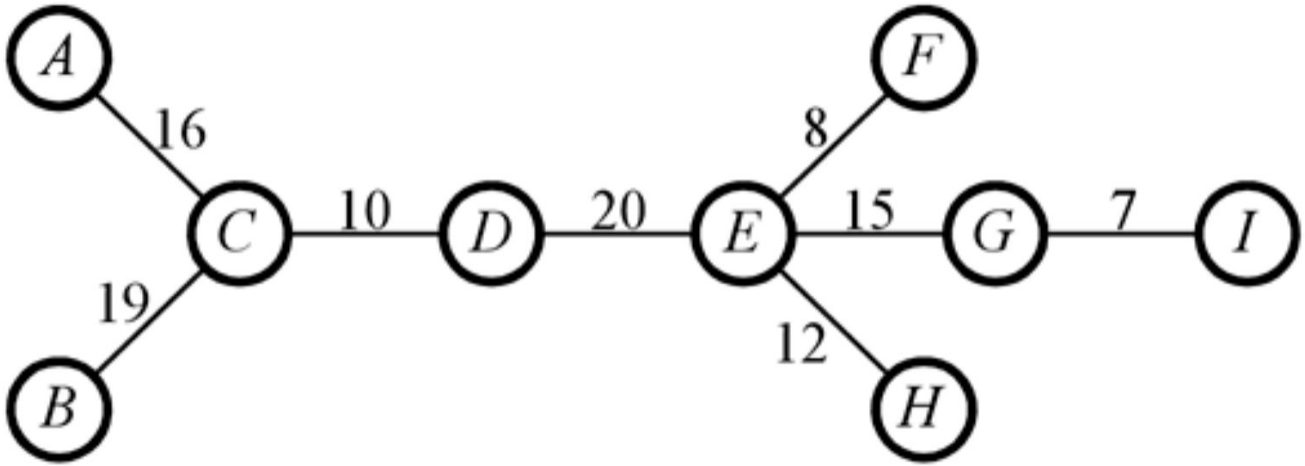
Introduction & Construction

- Find the maximum / minimum edge in the given tree and remove it to get two disconnected subtrees T1 and T2.



Introduction & Construction

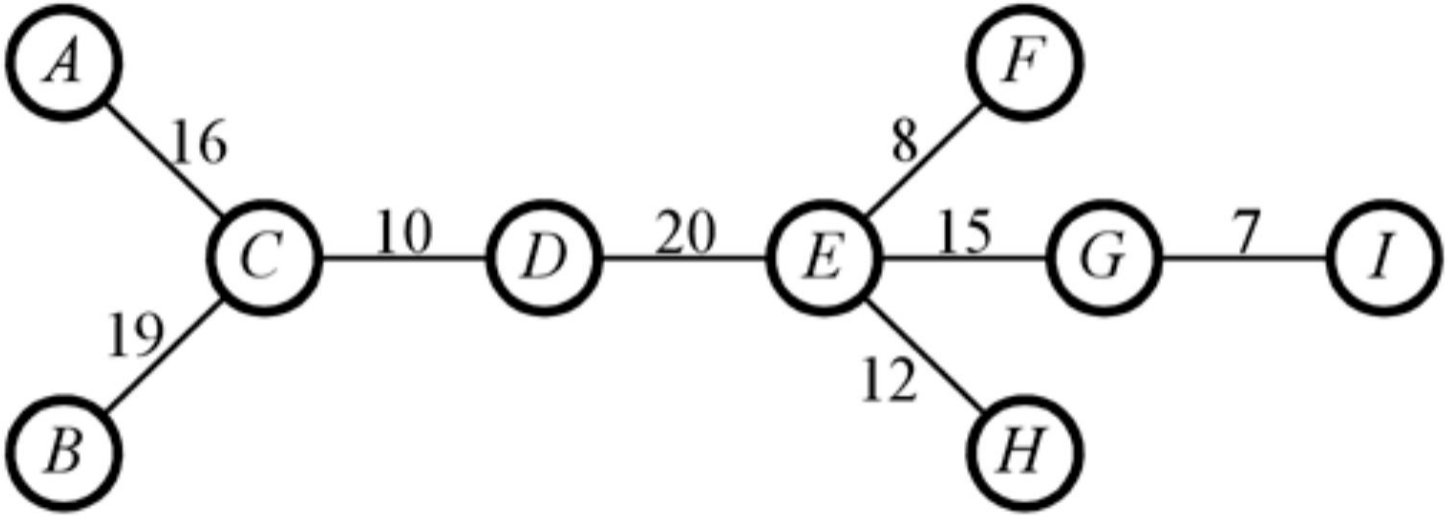
- Find the maximum / minimum edge in the given tree and remove it to get two disconnected subtrees T1 and T2.
- Create a new node N corresponding to the above edge in the new reachability tree / dsu tree / edge decomposition tree.



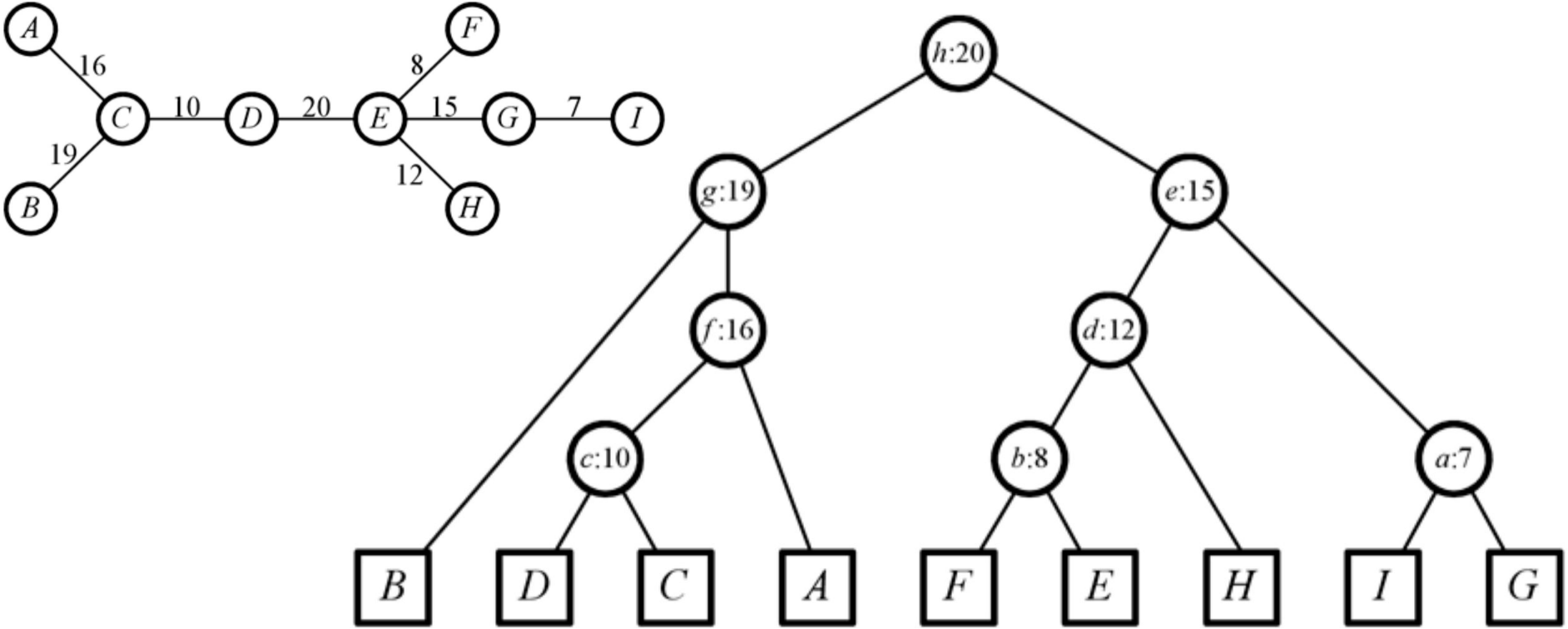
Introduction & Construction

- Find the maximum / minimum edge in the given tree and remove it to get two disconnected subtrees T1 and T2.
- Create a new node N corresponding to the above edge in the new reachability tree / dsu tree / edge decomposition tree.
- Call the procedure recursively on the two resulting subtree's T1 & T2 and attach the resulting reachability tree's as left & right child of the node N.
- Stop when there are no remaining edges to decompose

Find the Reachability Tree of the following given Tree



Find the Reachability Tree of the following given Tree



Implementation Ideas?

Implementation - Graph representation

- Since the final reachability tree will have a new node corresponding to every input edge, index the original nodes from 1...n and edges from n+1...2*n-1.
- Therefore, final reachability tree will have nodes indexed from 1..2*n-1.

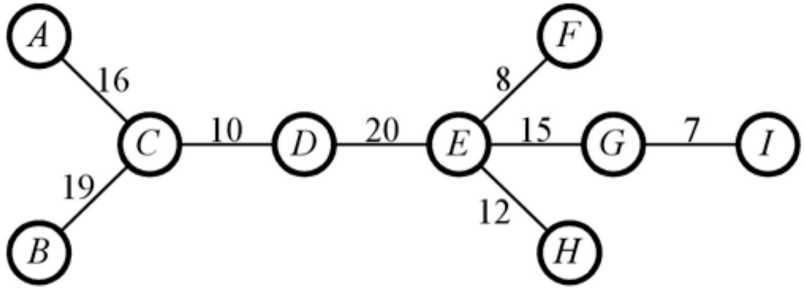
```
int n; scanf("%d", &n);
for (int i = n + 1; i < 2 * n; i++) {
  scanf("%d %d %d", U + i, V + i, W + i);
  E[i - n] = i;
}
int tree_root = build_tree(n);
```

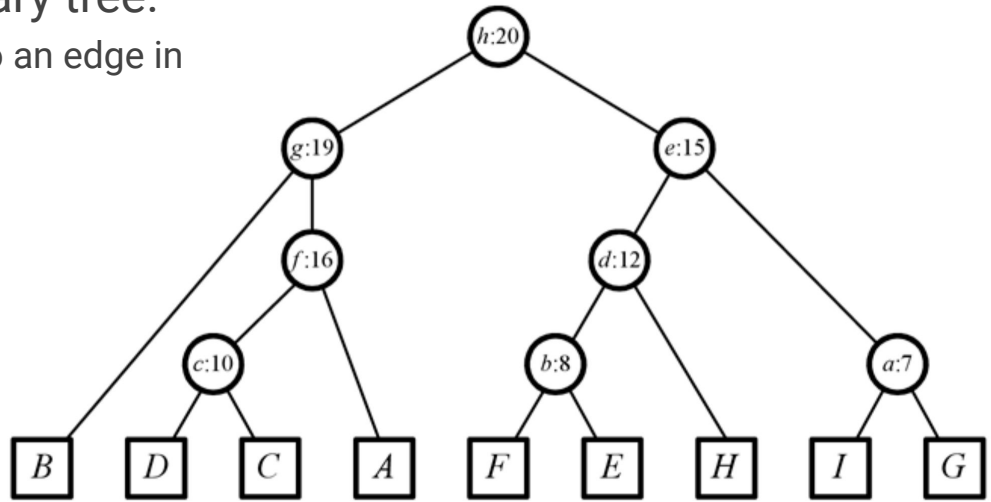
Implementation - Tree Construction

```
int build_tree(int n) {
  // sort edges in increasing order for max tree.
  sort(&E[1], &E[n], [](int e1, int e2) { return W[e1] < W[e2]; });
  function<int(int)> f = [&](int x) {
    return dsu[x] = (x == dsu[x] ? x : f(dsu[x]));
  };
  // Assumes nodes are numbered from 1..n and edges from n+1..2n-1
  for (int i = 1; i <= n; i++) { dsu[i] = root[i] = i;}
  // Builds the tree bottom-up.
  for (int i = 1; i < n; i++) {
    int e = E[i], x = f(U[e]), y = f(V[e]);
    // Attach root[x], root[y] as left & right children of e.
    tree[e][0] = root[x]; tree[e][1] = root[y];
    // Merge components of x & y and make e the new root.
    dsu[x] = y; root[y] = e;
  }
  return E[n - 1];
}
```

Properties of the Reachability Tree

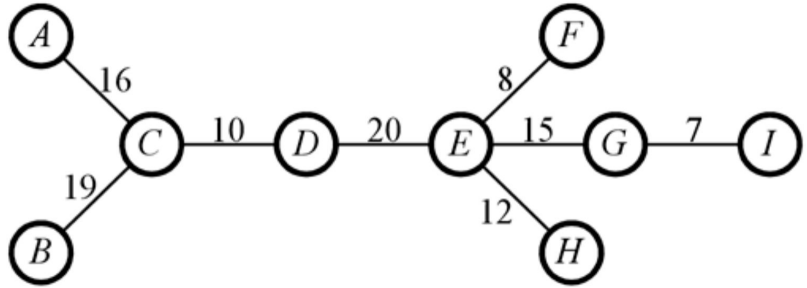
- No. of nodes = 2 * n - 1
  - N leaf nodes → Original Tree nodes
  - N - 1 internal nodes → Original Tree Edges
- The Reachability Tree is a full binary tree.
  - Every internal node (corresponding to an edge in OT) has exactly two children

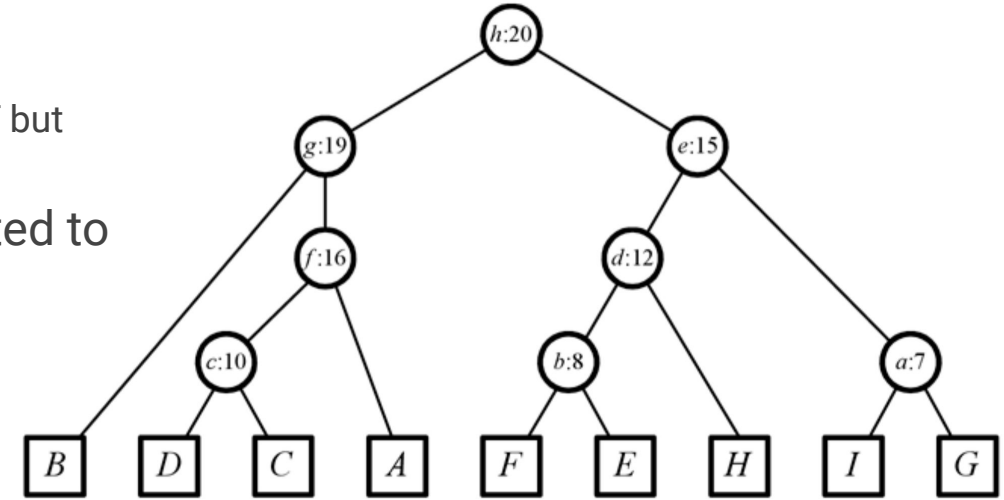




Properties of the Reachability Tree

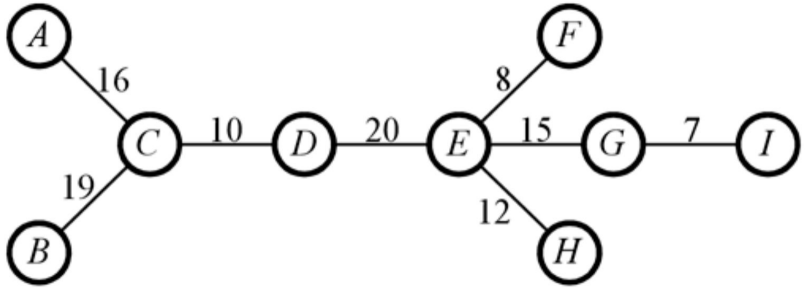
- Paths in Original Tree (OT) don't necessarily correspond to paths in Reachability Tree (RT)
  - B → D in OT has 2 edges (19, 10)
    B → D in RT has 3 edges (19, 10, 16)
  - B → F has edge 10 on the path in OT but
    B → F doesn't have edge 10 in RT.
- Therefore, RT is not very well suited to answer all types of path queries.

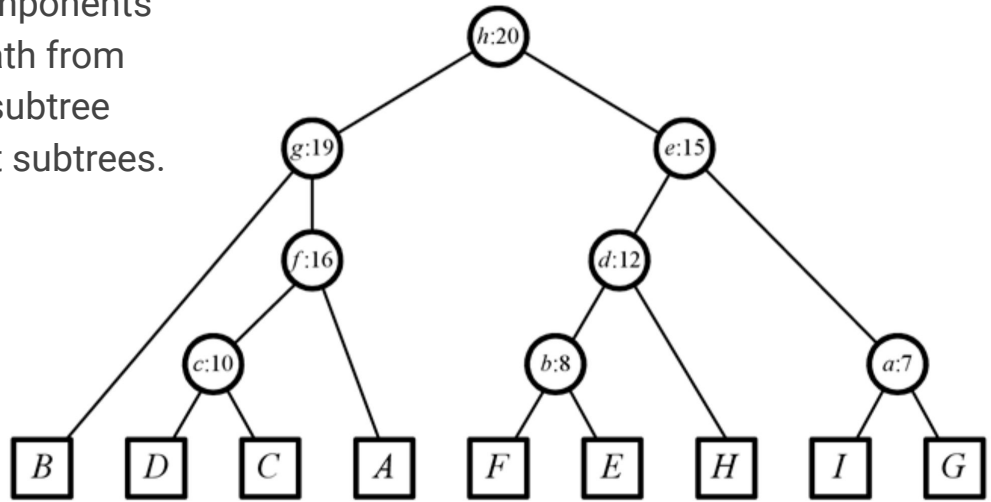




Properties of the Reachability Tree

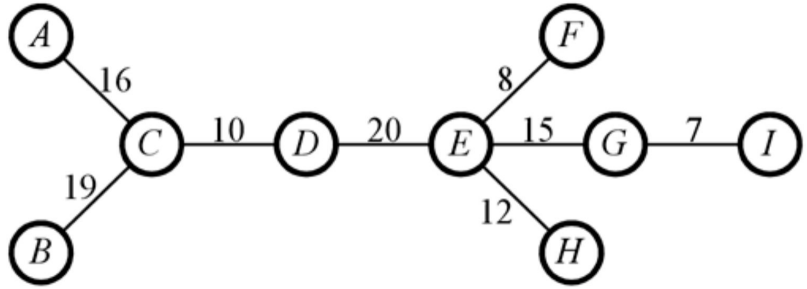
- The max edge on the path from u → v in OT corresponds to the LCA(u, v) in RT.
  - This is true because, by construction, the first u and v got separated into different components is when the maximum edge on the path from u→ v got selected as the root of the subtree and was removed to get two different subtrees.

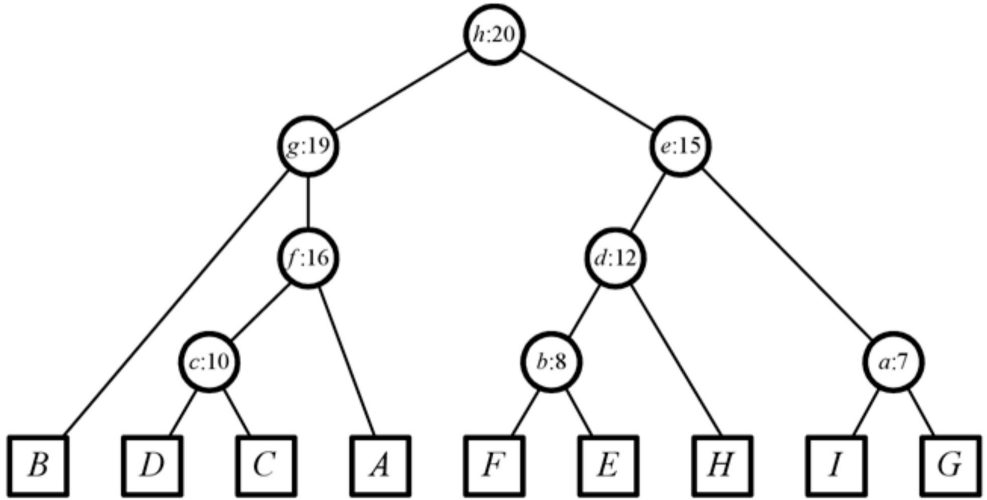




Properties of the Reachability Tree

- All nodes in the subtree of an internal node N_e form a connected subgraph in the original tree where weight of every edge in the subgraph <= W[e].

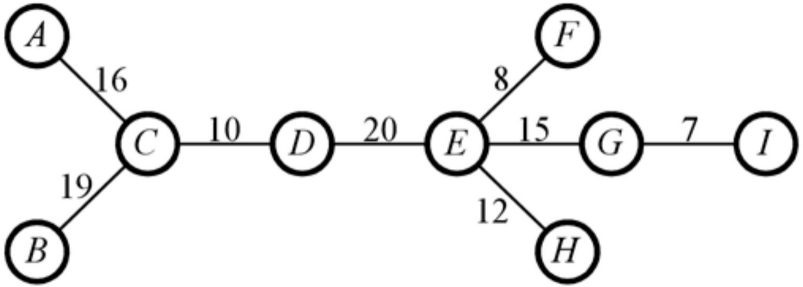


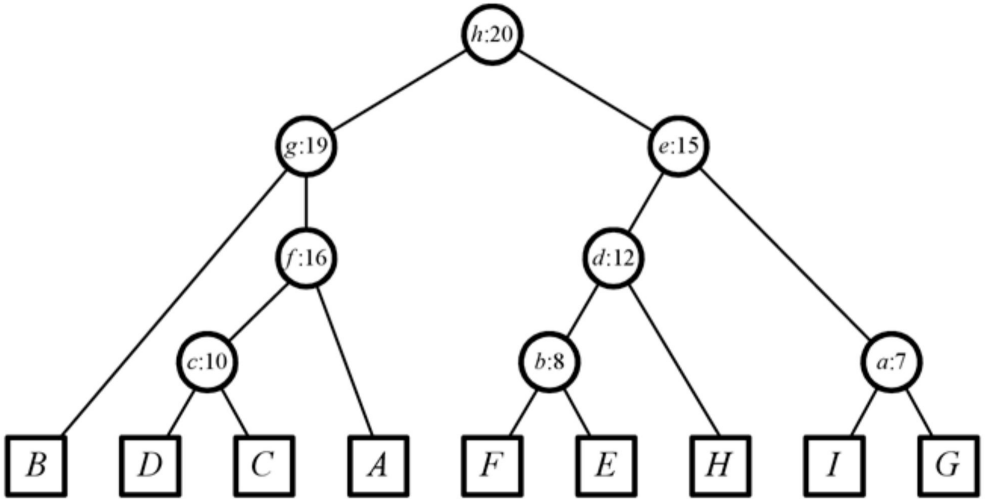


Problem - 1

- **Given a tree with N nodes, answer min / max edge on the path queries.**

https://www.spoj.com/problems/DISQUERY/

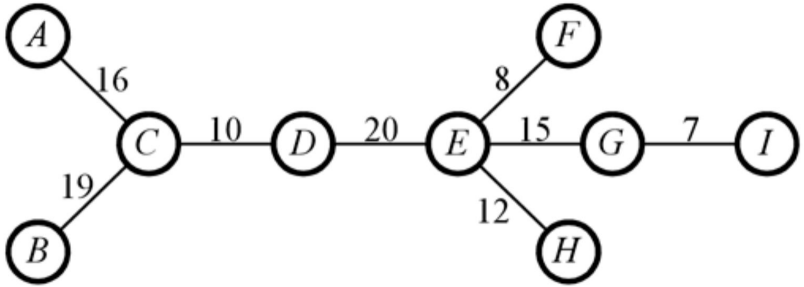


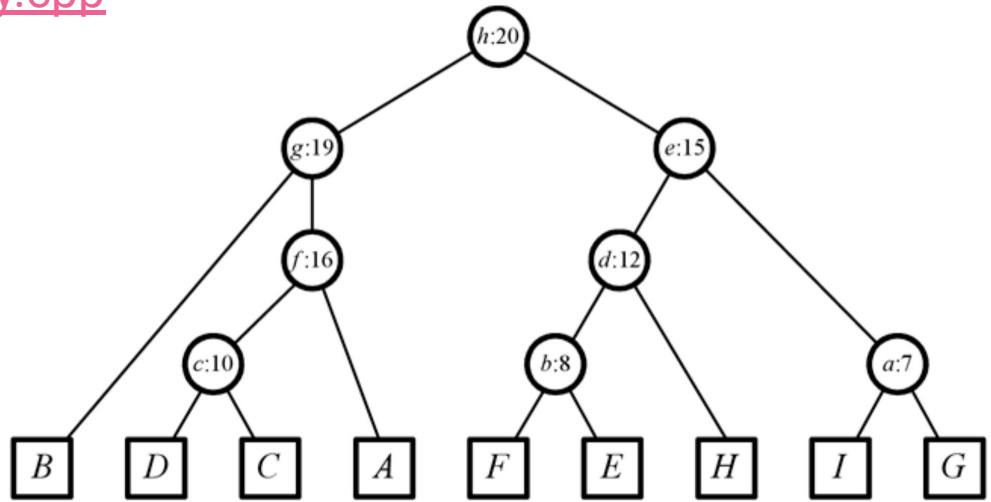


Solution - 1

- Min/Max edge on path from u to v = LCA(u, v) in Min/Max RT

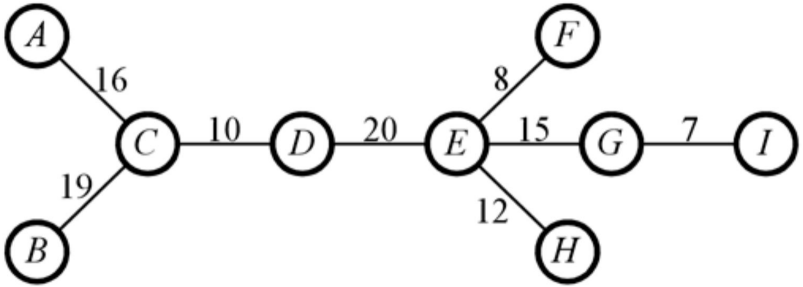
https://github.com/tanujkhattar/cp-teaching/blob/master/CWC/Ep01:%20Edge%20Decomposition%20Tree/disquery.cpp

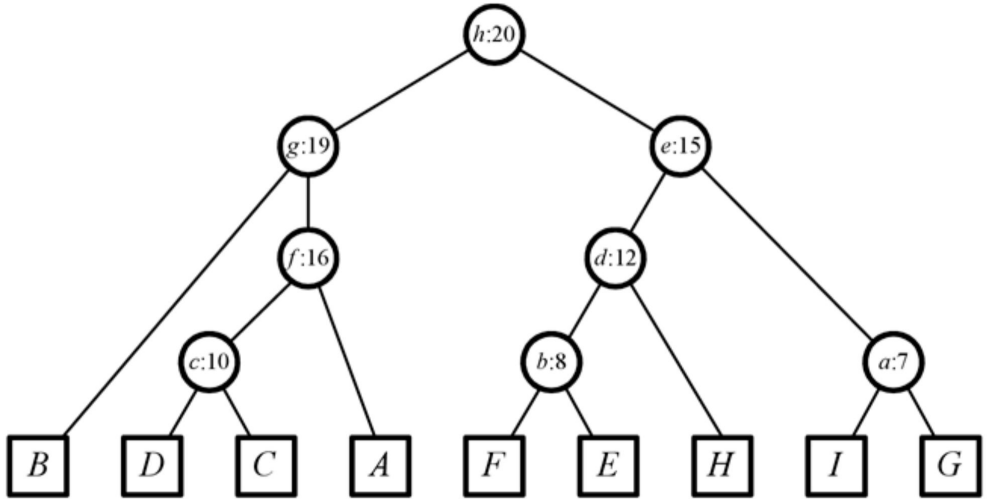




Problem - 2

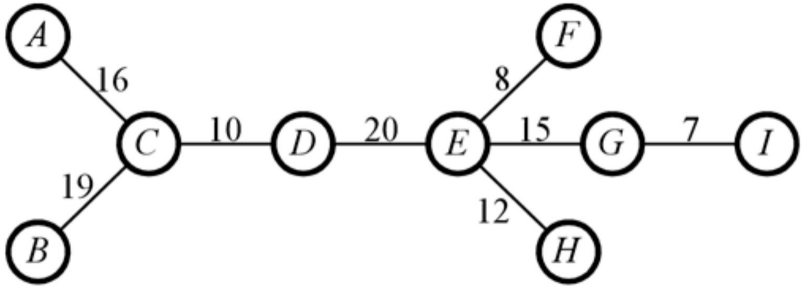
- **Given a weighted tree with N nodes, support Q operations of the form:**
  - **Q u k : Tell the sum of values of all nodes reachable from node u by only traversing edges with weights <= k.**
  - **U u add : Add “add” to values of all nodes reachable from node u by only traversing edges with weights <= k.**

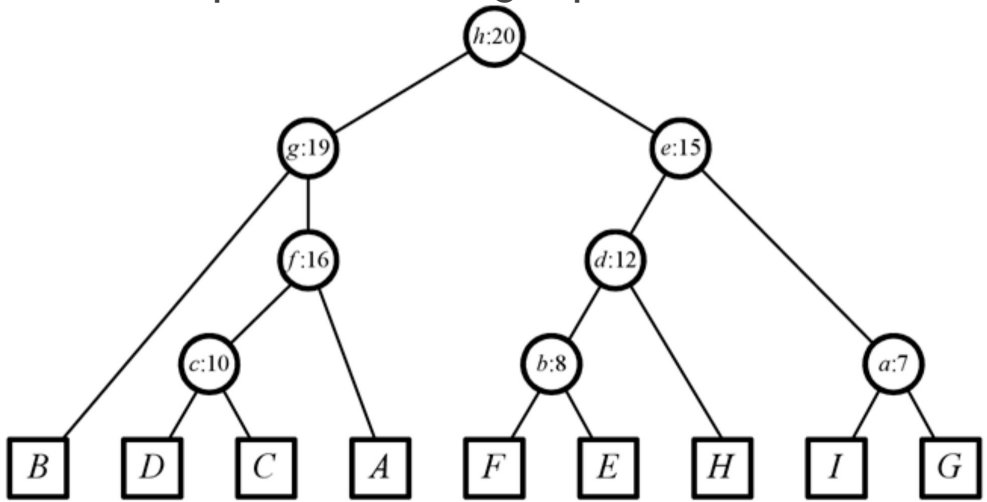




Solution - 2

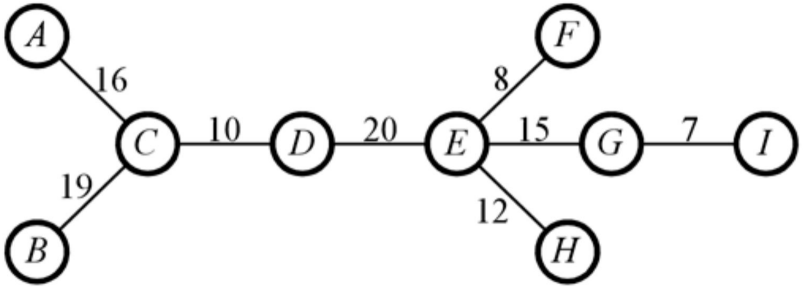
- Construct the RT and find the topmost ancestor N_e of u in the RT s.t. W[e] <= k.
- All nodes in the subtree of N_e need to be queried / updated.
- Construct ETT of RT and reduce queries & updates to range queries / updates.

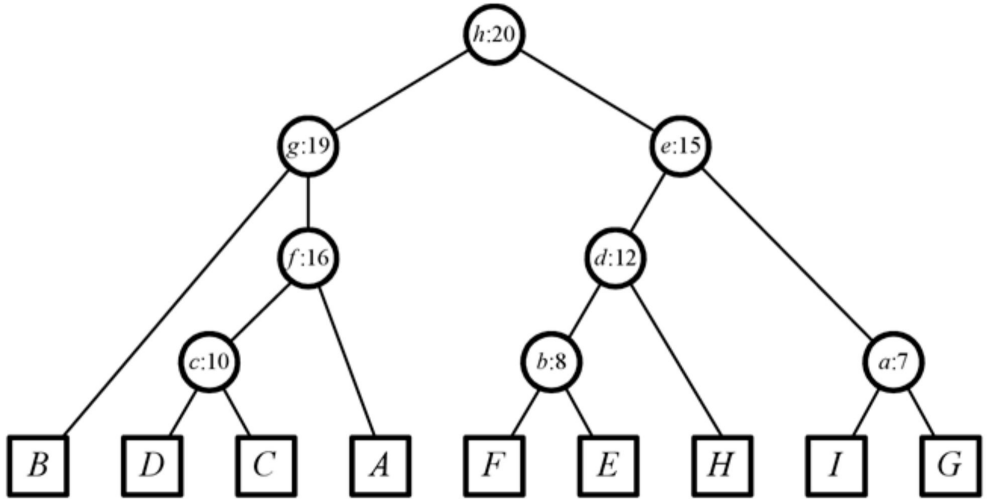




Problem - 3 : TULIPS

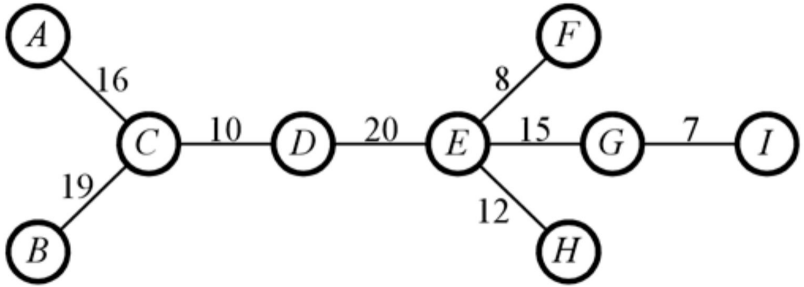
- https://www.codechef.com/problems/TULIPS

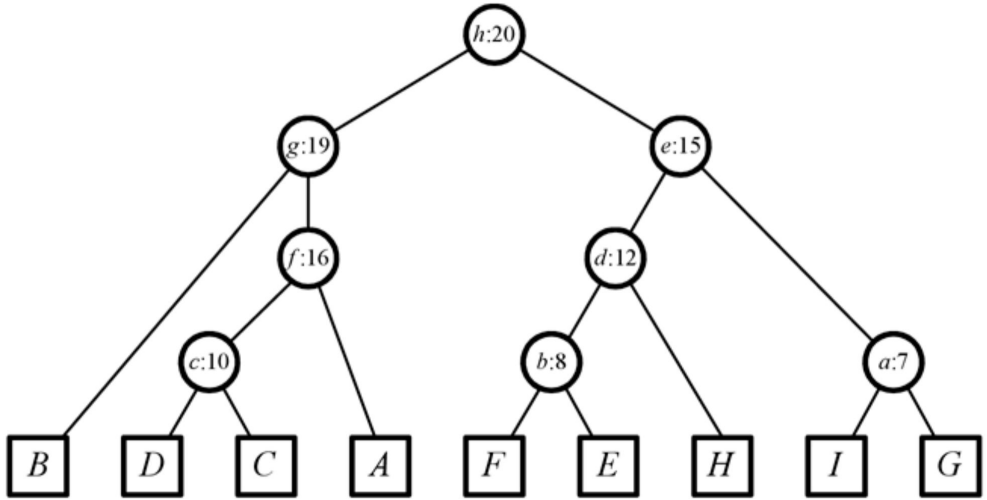




Solution - 3 : TULIPS

- Reduce queries & updates to range queries & updates in ETT
- Use a range query / update structure on ETT to solve the problem
- More on this in Part-2 of this episode.





Conclusion

- Reachability tree (RT) can be easily built in O(N logN) preprocessing.
- It’s a complete binary tree with N leaf nodes corresponding to the N nodes of the original tree and N - 1 internal nodes corresponding the N - 1 edges in the original tree.
- RT has a nice property that every subtree represents a connected component in the original tree and the edges in the connected component satisfy a certain property (eg: all edge weights <= K).
- We will continue the discussion tomorrow in Part-2 to look at interesting applications of RT.