Trees

Junhua Chen

Table of contents

Basic definitions

- A tree is an acyclic connected graph: N nodes and $N - 1$ edges
- A rooted tree dictates that every node, except the root, have a parent
  - It is common to have $p_i < i$
  - Often nice for dp problems
- Diameter is the longest path in tree
- Subtrees are defined for rooted trees

Problems I assume you know about

- Maximum independent set on tree (i.e any basic tree dp)
- Jim Thomas (i.e basic LCA)

Ordering a tree

- The tree is nice as you can order its nodes in a variety of ways
- Ordering is useful as more linear structures are better for things like slapping segtrees and/or dp

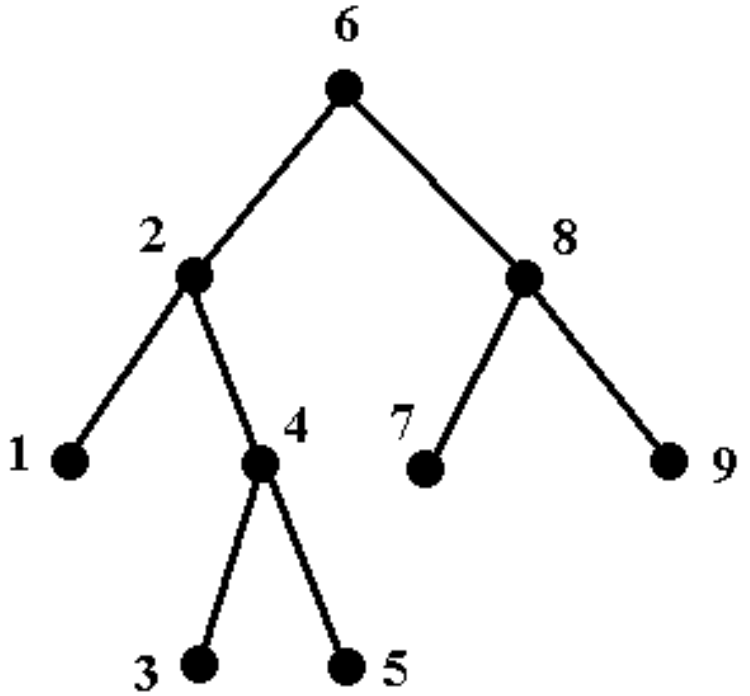
Orderings we will look at:

- Euler tour
- Preorder
- Subtree

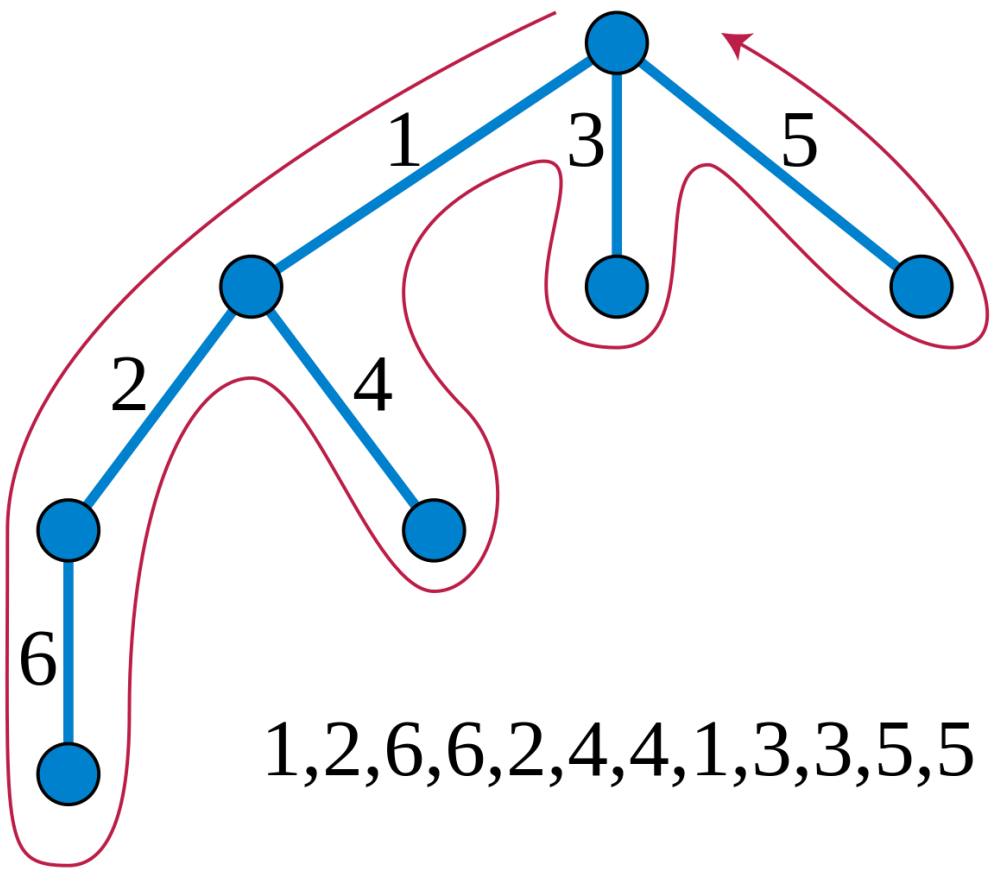
Euler tour

- Flatten the tree into a line

```
1
2   //EULER TOUR
3   int id = 0
4   int ord[N]
5   vector ETseq
6   function ET(u):
7       ord[u] = id
8       id += 1
9       for each child c DO
10          //INCLUDE BELOW LINE FOR ET BUT NOT PREORDER
11          ETseq.pushback(u)
12          ET(c)
13      endfor
```



What the ET basically does

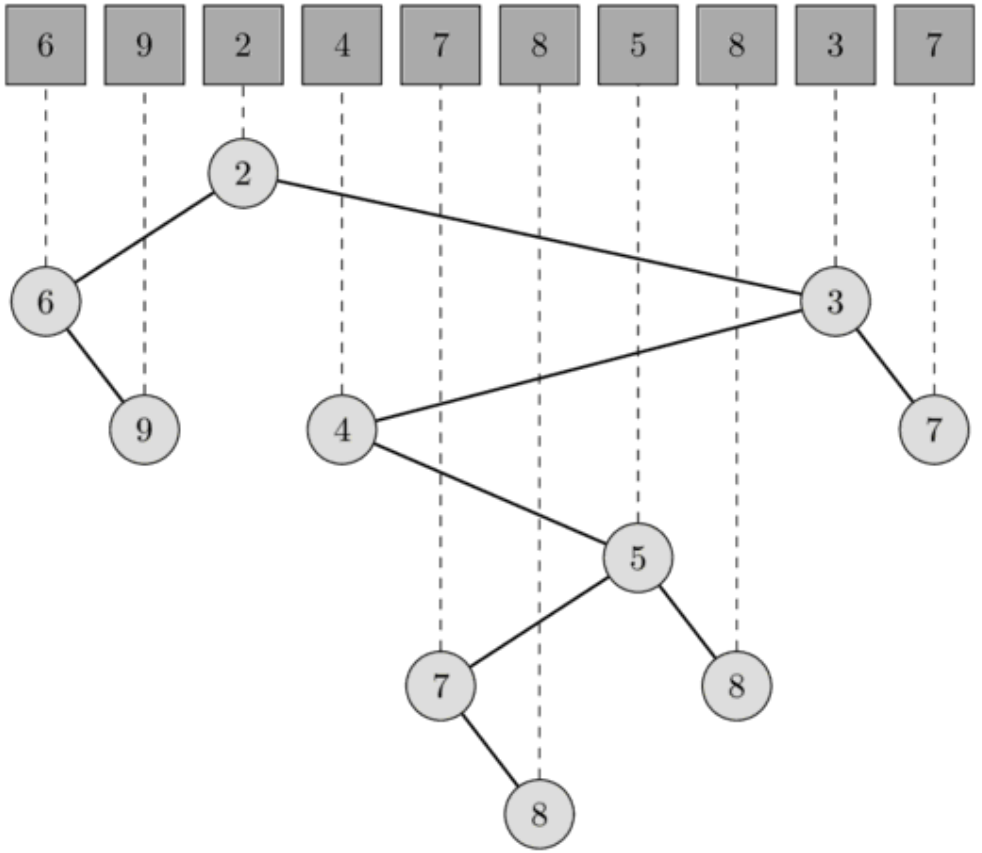


Why is this useful?

- Maps each subtree into a interval [l,r]
- So it maps the tree into a number line and nested intervalswhile preserving some structure of the tree

- Motivating problem:
  - Make data structure on tree supporting following operation:
    - And x to the value of each node
    - Query maximum node label in subtree

Euler tour and segment tree

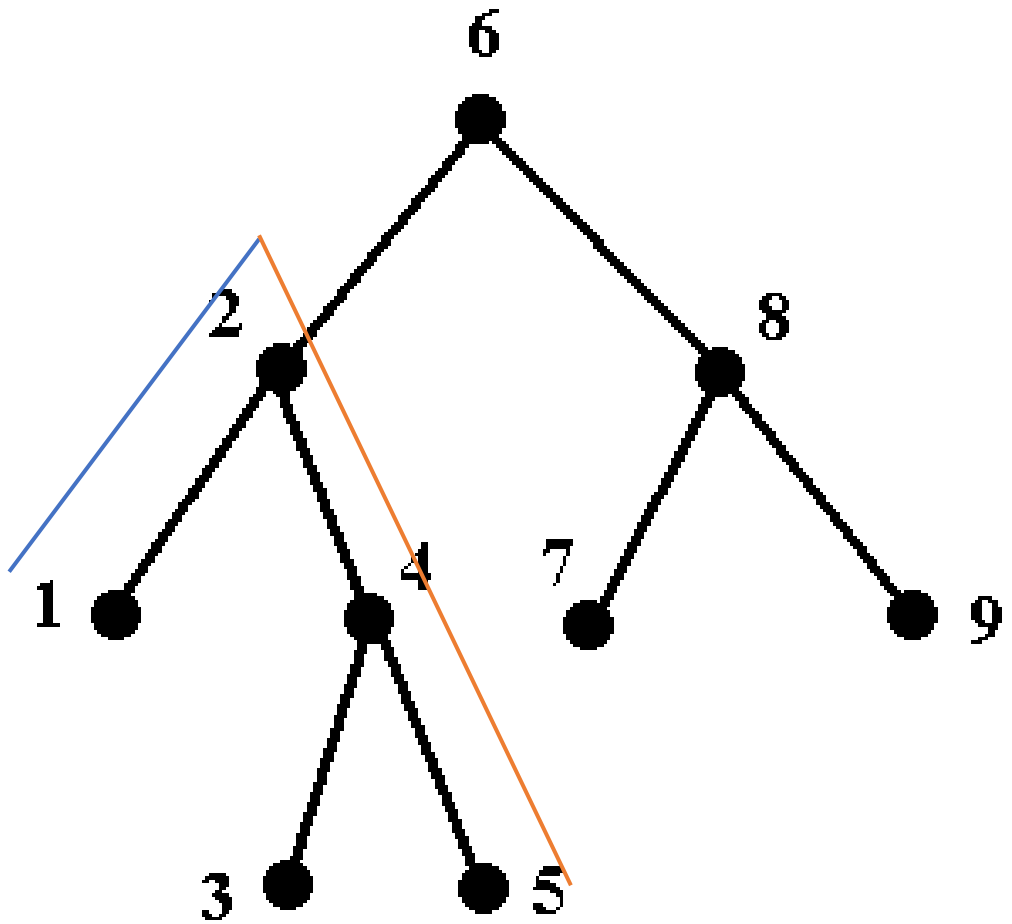


The preorder sweep

- Order is also good for sweeping. Preorder sweeps are nice as they eventually walk all the paths on a tree

- Consider the following problem:
  - Given tree of $N \leq 10^5$ nodes and $M \leq 10^5$ paths, where each path has a score. For each edge determine the maximum score path that uses the edge.

The How to: split the paths at LCA



Now that the paths are “nice”

- Pre-order walk:
  - Entering the subtree of a node: Activate the paths that start at this node
  - Leave the subtree of a node: Deactivate the paths starting at this node

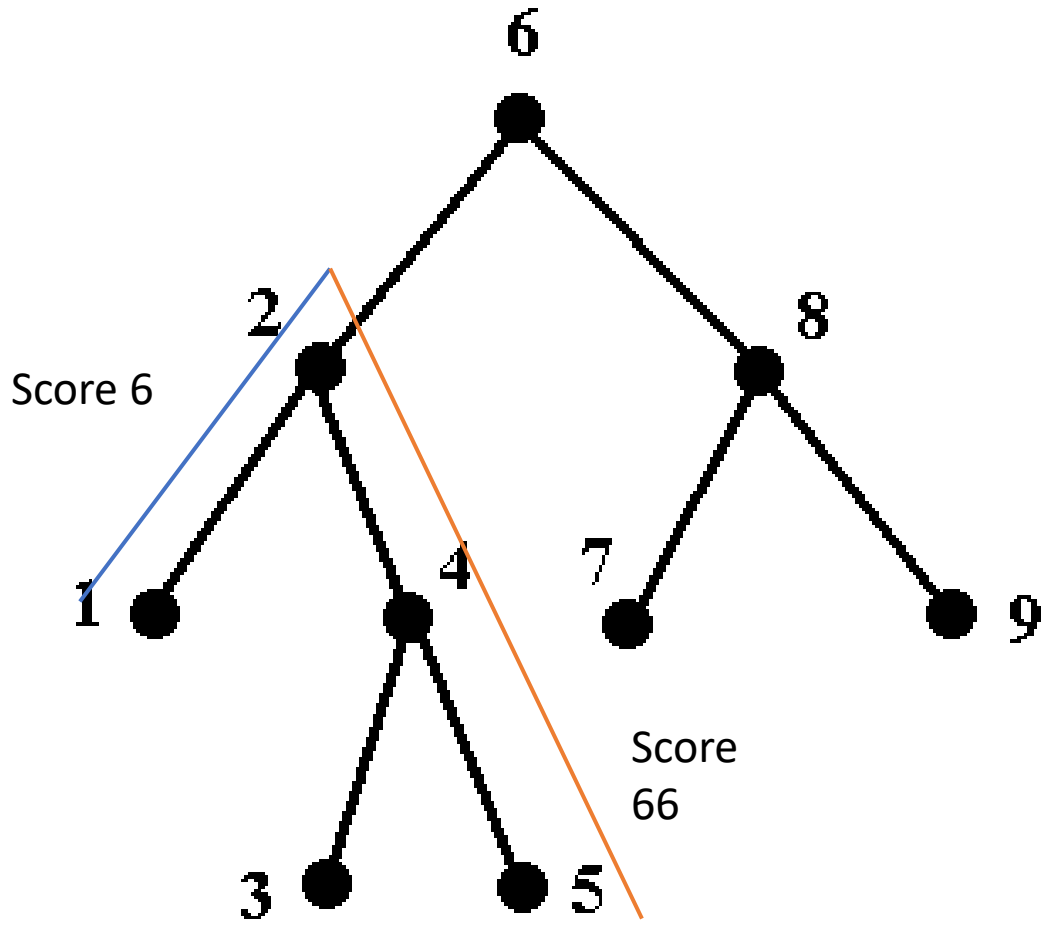
To discuss activation/deactivation, we use our euler-tour segment tree from before.

Activate: place the endpoint of path in the segment tree

Deactivate: remove the endpoint of path in segment tree

Query an edge: Use RMQ to determine which active endpoint in subtree is best

Example: Adding this edge



6	2	1	4	3	5	8	7	9
		6			66			

But the most important property of the tree is …

- That it’s a tree
- The subtree structure of the tree allows all sorts of dps, and so problems that are hard (NP-H) in general graph are doable in O(poly) on trees e.g Max independent set

DP on tree + small merge

- So suppose you have a tree on $N \leq 10^5$ nodes, each node has a value $W(i) \leq N$. You wish to find a series of nodes $v_1 \ldots v_k$ such that:
  - These nodes lie on one path
  - $W(v_i) < W(v_{i+1})$

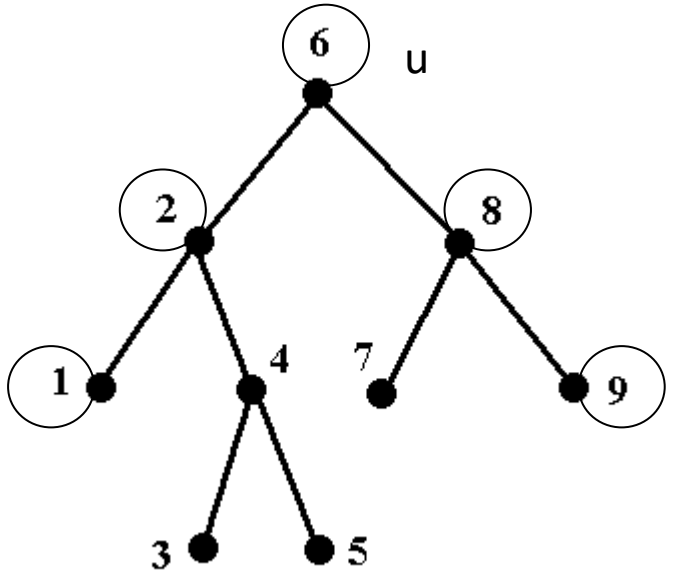
  - TLDR: Longest increasing subsequence on tree

Basic observations: Root tree arbitarily

- Consider the path $v_1 \rightarrow v_k$ of a valid subsequence
- The path goes up some LCA node u then down again

  -going up part: the deeper the node the less the label

  -going down part: the deeper the node the greater the label



Iterate over all choices of u:

- Naïve solution $dp[i][j]$= “In the subtree of I, what is the best ‘upwards path’ that ends at node with value at most j”
- $dp[i][j] = \max_{k\ child\ of\ i} dp[k][j-1] + 1$

- This is $O(N^2)$. Can we do better?

Revision: LIS in 1 dimension

```
//maintain set of changepoints c s.t dp[i][c] > dp[i][c-1]
Set S
Int A[N]

for i = 1 … N do
      erase first value in S greater or equal to i, if any
      insert I to S

Len(S) = answer
```

Q: Why does this work?

Maintain $dp[i][j]$ in a set

- Rather than maintaining $dp[i][j]$ explicitly we maintain a set $dp[i]$ for its changepoints j
- To combine children we small merge the sets of the children

- Q: How do we small merge?

  e.g Merge changepoints [2,3,4] into [1,3,5] & amortise

- Complexity: $O(Nlog^2N)$

Summary

- When the **state transitions** are very simple, using **data structures** to update many entries of the dp at once (in our case implicitly) + **small merge** in tree dps can allow us to accelerate the dp while maintaining effectively the same **dimensionaity** of the dp

- Same idea possible with many tree problems (Kevin would have explained them)

LCA and virtual tree

Motivating problem:

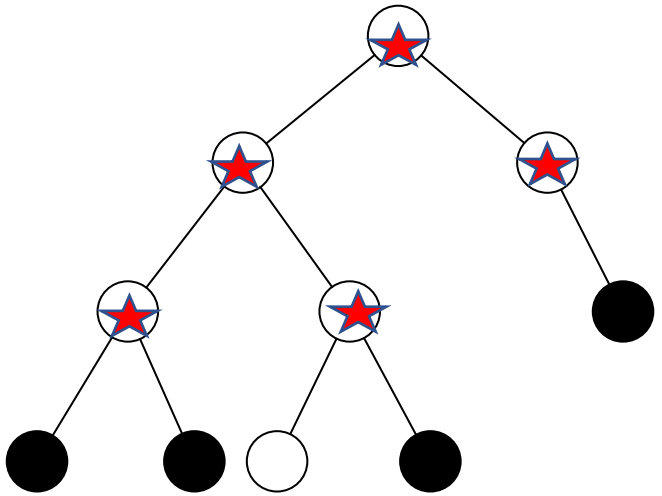
Given tree on $N \leq 10^5$ nodes answer $Q \leq 10^5$ queries of form:

`query` $v_1 \ldots v_{n_i}$: what is the maximum distance between any two nodes?

$\sum n_i \leq 3 \times 10^5$

How to do it with just one query?

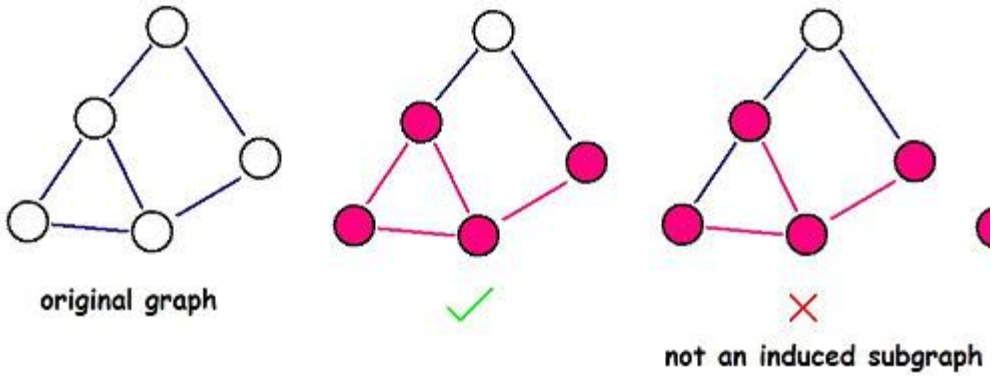
- Lets call an edge or node *important* if it lies on the path between any two nodes that are given
- Consider all important nodes and edges, they form a ***subgraph*** of the original tree, call it the “important forest/tree”
- Then just find diameter of this subgraph



Aside on terminology

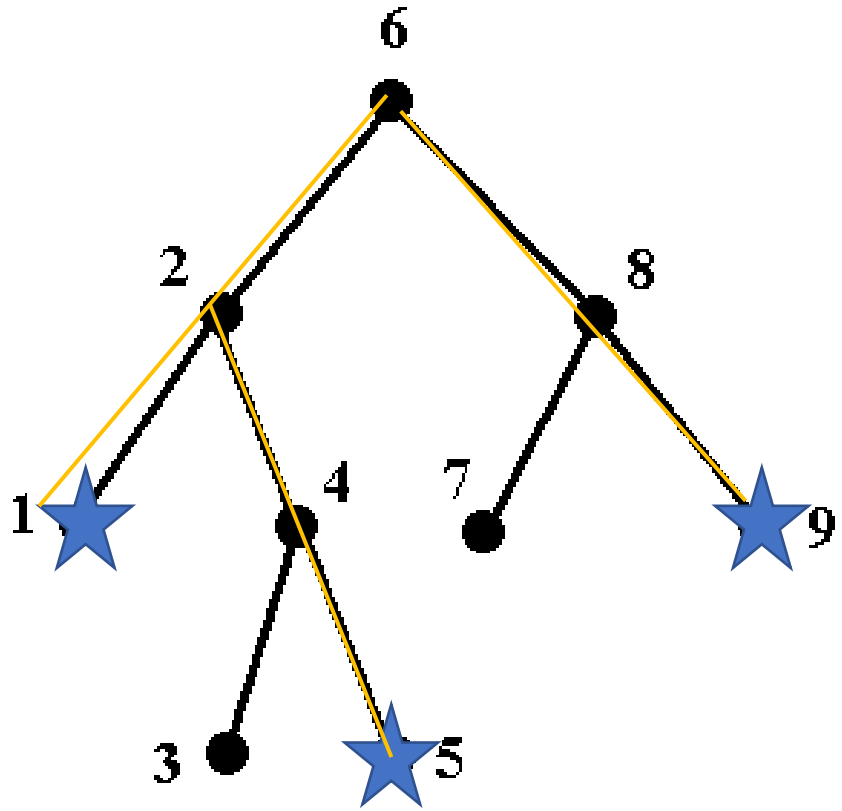
- **Subgraph** of a graph is made by considering a subset of the nodes and edges of a graph

- A vertex **induced subgraph** is a subgraph based on a set of nodes, which contain all relevant edges



How can we speed things up

- Consider a query of 3 nodes
- Even though the number of nodes is “big” the structure is simple
- Seems sus



Lemma: The "important tree" of N nodes has O(N) nodes with degree exceeding 2

- **Proof:** Let $S = \{lca(a,b) | a,b \in V\}$ where V is the set of query nodes
- We show $|S| = O(N)$ as S is the set of all deg > 2 nodes

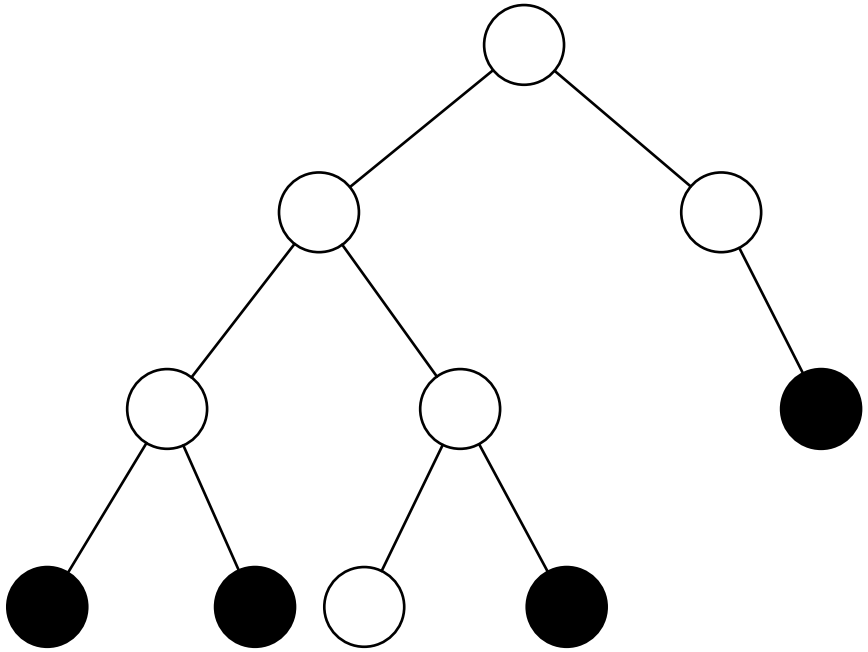
- **Claim**: sort V by preorder index, then $S = \{lca(V_i, V_{i+1}), 1 \leq i < N\}$
- **Proof:** Suppose $x = lca(V_i, V_j)$ and $x \neq lca(V_k, V_{k+1})$ for some k.

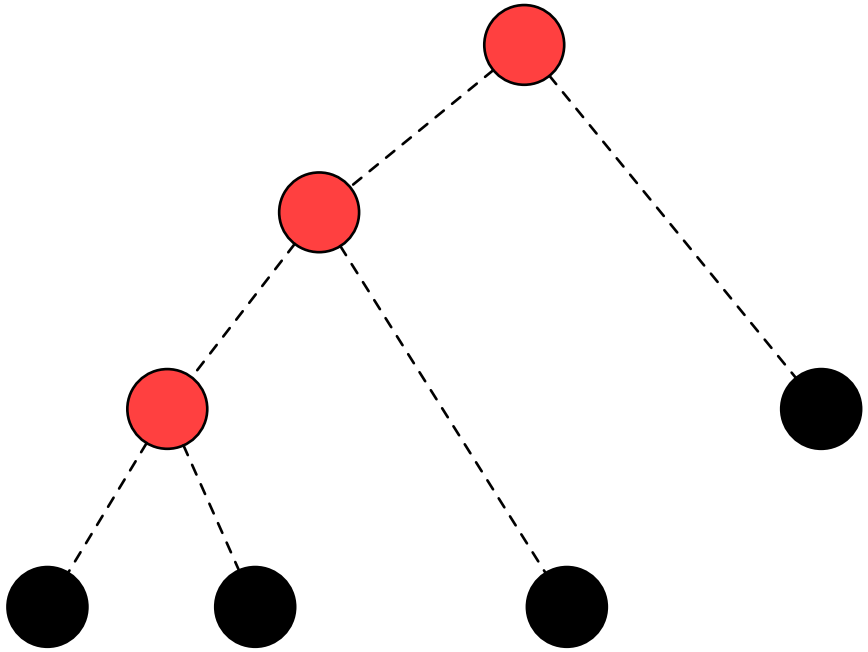
Consider the euler tour ordering, and let l be the first occurrence of V_i and r the last instance of V_j. Then $x = argmin_{l \leq i \leq r}\{depth(i)\}$

Consider an vertex $i < \lambda < j$. Then either $x = lca(V_i, V_\lambda)$ or $x = lca(V_\lambda, V_j)$. But then we can basically "binary search" for k, contradiction

Great … we got a proof but how do we build?

- The critical nodes of the tree (with degree != 2) are kept in our compressed “virtual tree”, while the rest with degree 2 are compressed into paths
- The critical set $C = V \cup S$





Then sweep by depth high to low

- Ha … another way to order the nodes (also can think of as BFS order)
- Use map to maintain current root nodes in euler tour position, then at each node find the nodes in its subtree, mark them as children and delete them from the map
- Then insert current node into map
- This builds tree in $O(NlogN)$ which is good enough

Takeaways: Virtual Tree

- When it comes to studying lca structures, the euler tour is a great tool
- Dumb sqrt solution exists to this problem can you find it
- While this technique doesn’t appear that often, the ideas behind it regarding LCA & its interplay with euler tour is why I chose to teach it

Greedy algorithms on tree

- Thinking about extremal properties of trees help
- Combinatorial thinking also pays off handsomely often

- Leaves
- Forced moves

- Oh, and solve a smaller problem: subtrees

Detailed study: Tokens (ACIO contest 2)

- https://acio-olympiad.github.io/2020/snap.pdf
- Few minutes to read (simple problem)

- Shameless ACIO plug
- We will look at tree version of the problem
- Vortex = Cycle with trees hanging off it = screw it (actually a lot of graph types look like trees and much of the time reducing a problem to trees is nice)

Basic Observations: Reducing tree structure

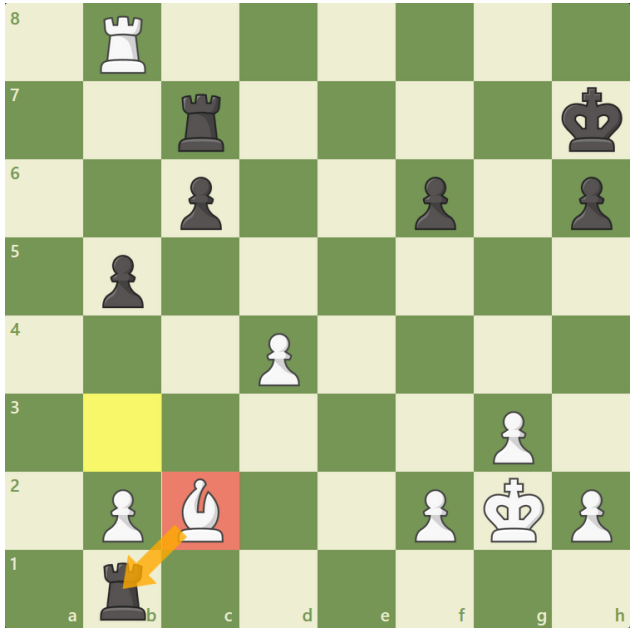
- Often a good bit of a tree in these problems is useless
- E.g a leaf node with nothing on it (why would you go there)

- Exterminate them
- Okay, so now every leaf has tokens on them

Forced moves: Moves that are evidently optimal

- If leaf has tokens on them what can you do but move them upwards?

- Eliminate tokens if possible, then rinse and repeat from reducing part



Sure, we can implement this

- But this is messy
- Now think combinatorial
- Ask question: Under what circumstances is a token moved through the edge; we no longer just think of trees as a graph

Answer

- Parity of number of tokens in a subtree
- Easy, O(N) solution

Summary of things we've done to solve

- Reduce the tree
- Forced Moves
- Think combinatorically
- Do it by subtree

Other things that are helpful

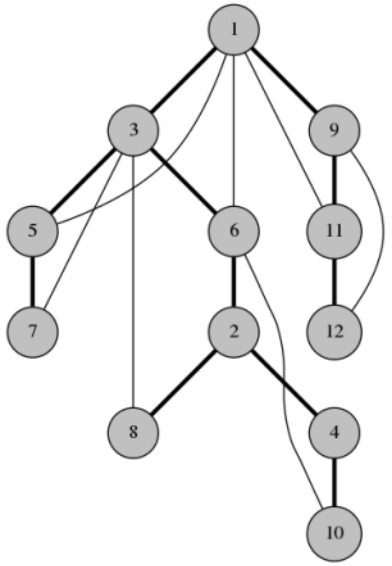
- Sorting (duh)

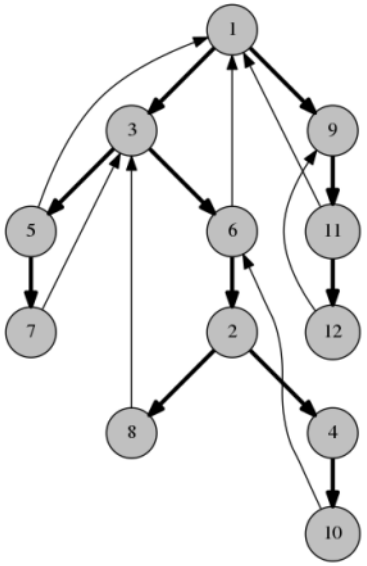
DFS trees (A brief look)

```
Def dfs(u):
    vis[u] = true
    for each u->v:
        if vis[v]: pass //backedge
        v.par = u
        dfs(v)
```

Properties of dfs tree (undirected graph)

- Every edge is either an backedge (connects ancestor and descendent) or an edge in the tree (proof: consider visiting order)
- Applications: There are very many, a notable one being bridge finding
- Definition: An edge in a graph is a bridge iff removing it increases the number of connected components





How to find bridges

- An edge is a bridge iff no back-edge goes past it
- The algorithm's correctness follows from backedges

```
//Bridge
int depth[N], lo[N]
bool vis[N]

function dfs(u, prev):
    vis[u] = true
    lo[u] = depth[u]
    for each node u->v DO
        if not vis[v]:
            depth[v] = depth[u]+1
            dfs(v, u)
            lo[u] = min(lo[u], lo[v])
        else
            lo[u] = min(lo[u], depth[v])

    if lo[u] == depth[u]
        (u,prev) is bridge
```

Applications:

- Given graph direct its edges so that you can reach any node from any other node or report impossible
- Find maximum independent set on graph of N nodes and $N + 8$ edges where $N \leq 10^5$
- This is just one of many possible applications of dfs trees (such as SCC, articulation points)

Problems

Core

- COI2018 Paprike (oj.uz)
- CEOI 2019 Magic Tree (CF)
- Max Flow
- Amazon II
- Organisational Enlightenment
- ACIO Tokens

Optional

- Courier (very very very very very very hard)
- IOI2019 Practice contest Job (very hard)
- ACIO Vibe Check
- FARIO 2011 Virus (hard)

Bridge finding problems

- Network (2nd priority to core problems)

Oh … one more thing

- Decomposing Tree into paths is often useful for all of things

- HLD for data structures
- Proofy things
- Modules

- Pattern: assign edge to path based on some subtree condition

HLD

- Consider the tree model
- Let $sz(x)$ be number of nodes in subtree of x
- Call an edge $(x, par(x))$ **heavy** iff $sz(x) > \frac{sz(par(x))}{2}$ and **light** otherwise
- In any path from x to root of tree there are at most $logN$ light edges as subtree size doubles each light edge you take

- Decompose tree into **heavy paths**

Proof of DSU with path compression

- Time for a find query is proportional to $\#Heavy\ edges + logN$
- Each Union operation creates 1 heavy edge at most
- The heavy edges removed by path compression ***amortises*** out its cost (at most 1 of resultant edges is heavy)

- Amortisation means paying for some operations in advance by "charging" an "potential function" (you can see it as a debit card)