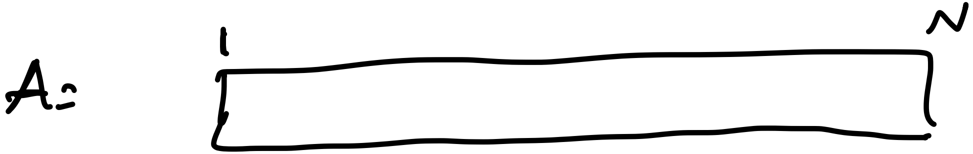
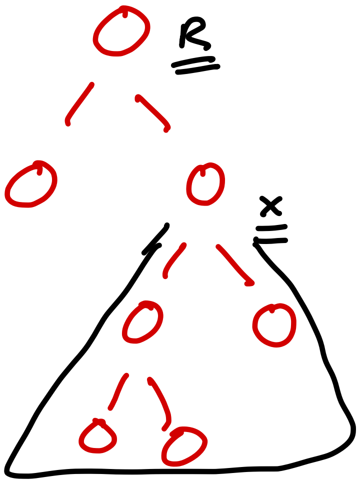
HLD & ETT (IIIT Lecture)

tanujkhattar@

Subtree Queries & Updates

- Given a rooted tree with N nodes and Q queries of the form:
  - Q X - Tell the subtree sum of node X.
  - U X Val - Add Val to all values in the subtree of node X.
- Practice Problem:
  - https://cses.fi/problemset/task/1137 (simpler version)





ETT

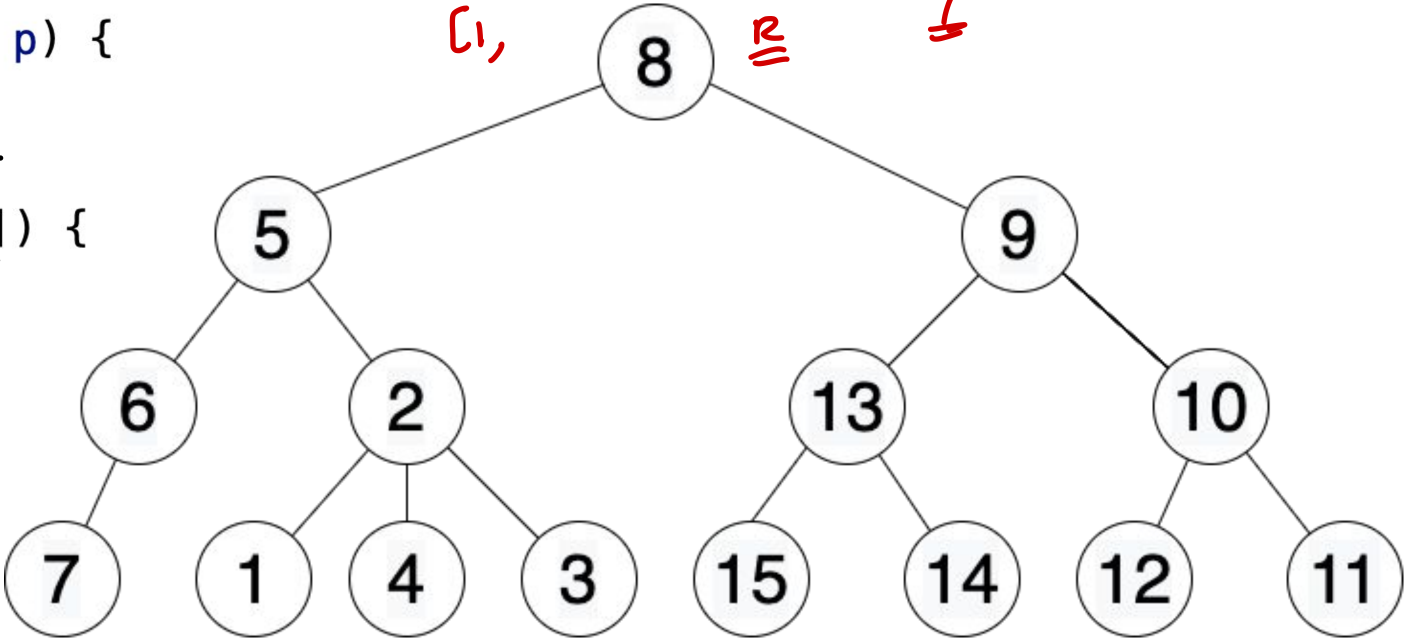
Euler Tour Trick: Way-1(Single Occurrence)

- Do a DFS and push every node in the Euler Tour Array when you first enter the node.

T= 0

```
void dfs(int x, int p) {
  st[x] = ++T;
  // E[T] = x;
  for (auto y : g[x]) {
    if (y != p) {
      dfs(y, x);
    }
  }
  en[x] = T;
}
```

[1,   R   I



Way-1(Single Occurrence): Properties

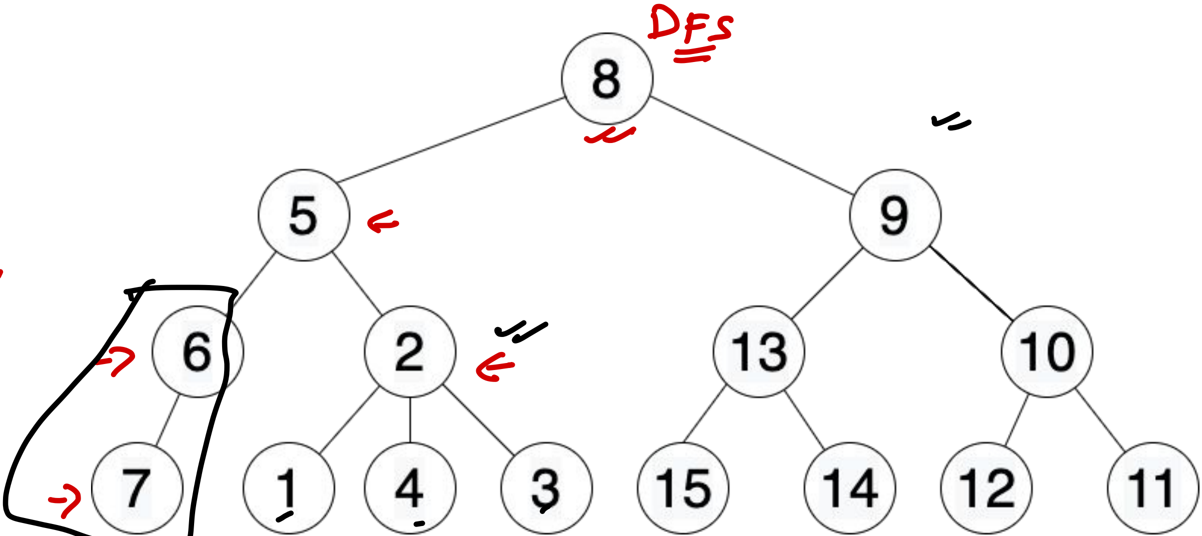
- Subtree of a node x corresponds to a subarray in the Euler Tour Array (E).
  - Subtree of node x == [st[x], en[x]]
- Every node in the tree occurs exactly once in the Euler Tour Array (E)
  - E Eg: [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]

x ⇒ [st[x], en[x]] ✓

E ⇒ [L , R] ✓

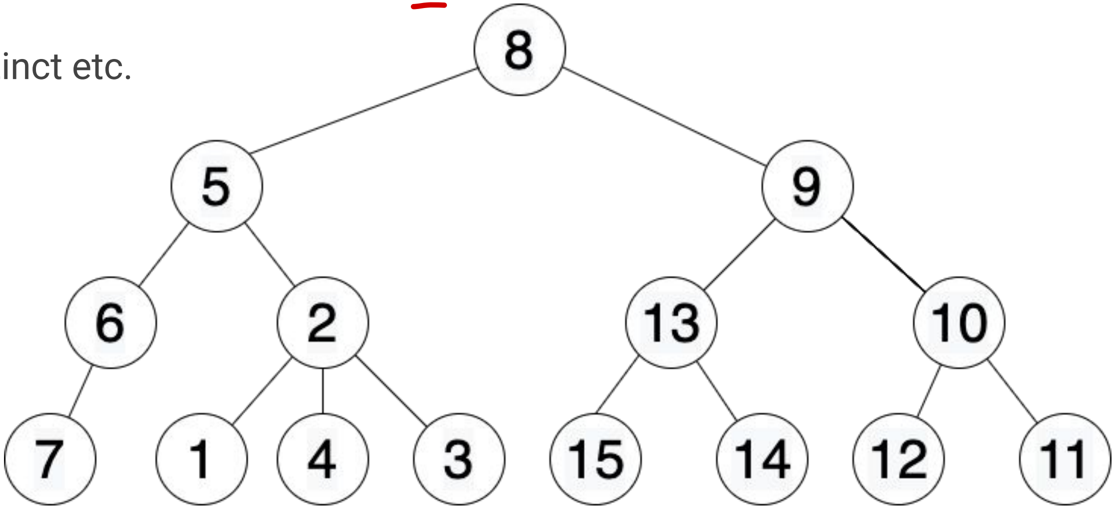
SU x ⇒ RU : [st[x], en[x]]

SQ x ⇒ RQ ||



Way-1(Single Occurrence): Properties

- **Any** subtree update & subtree query for a node x can be reduced to a range update & range query on the Euler Tour Array (E).
    - E : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
    - For any x; subtree(x) → [st[x], en[x]]
    - All sorts of tricks that can help us solve the problem on arrays can be used to solve the same problem for subtrees
        - Eg: Min / Max / Sum / Distinct etc.



Summary & Practice Problems

- Any subtree query/update problem can be reduced to an array query/update problem.
    - Subtree(x) → Range [st[x], en[x]] in the linear Euler Tour Array
- https://codeforces.com/problemset/problem/620/E
- https://codeforces.com/problemset/problem/893/F
- https://codeforces.com/problemset/problem/384/E

Point Update & Path Queries

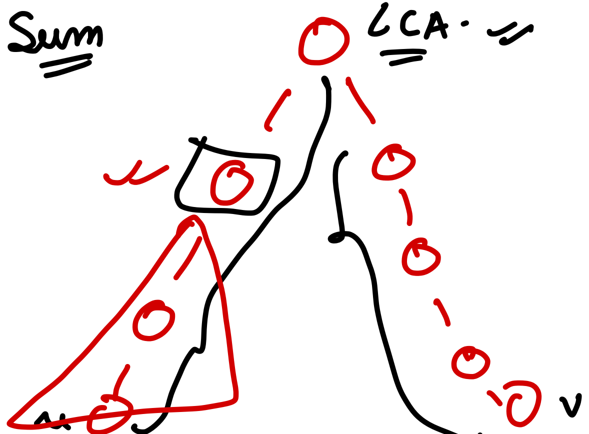
- Given a rooted tree with N nodes and Q queries of the form:
  - Q U V - Tell the sum of elements on the path from u to v.
  - U X V - Set A[x] = V.
- Practice Problem:
  - https://cses.fi/problemset/task/1138 (simpler version)

Path Query

Point Update.

* Min instead of Sum

D[x][y].

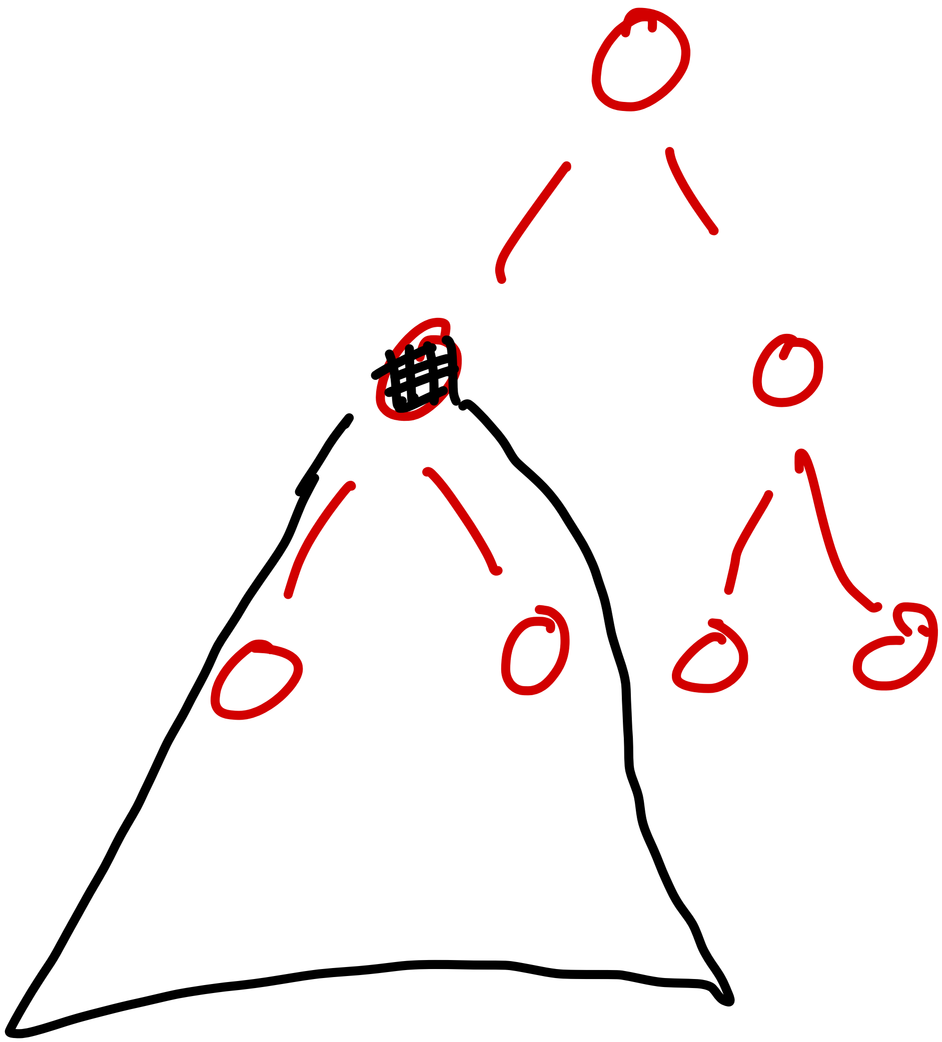


$l = LCA(u,v)$?.

* $PS[u] + PS[v] - 2 * PS[LCA]$

* $DP[i][j]$: j ... 2^{j}th anc. of j

* Path Query
* Point Upd.



* Subtree Update
* Point Query : PS[u] + PS[v] - 2 * LCA.

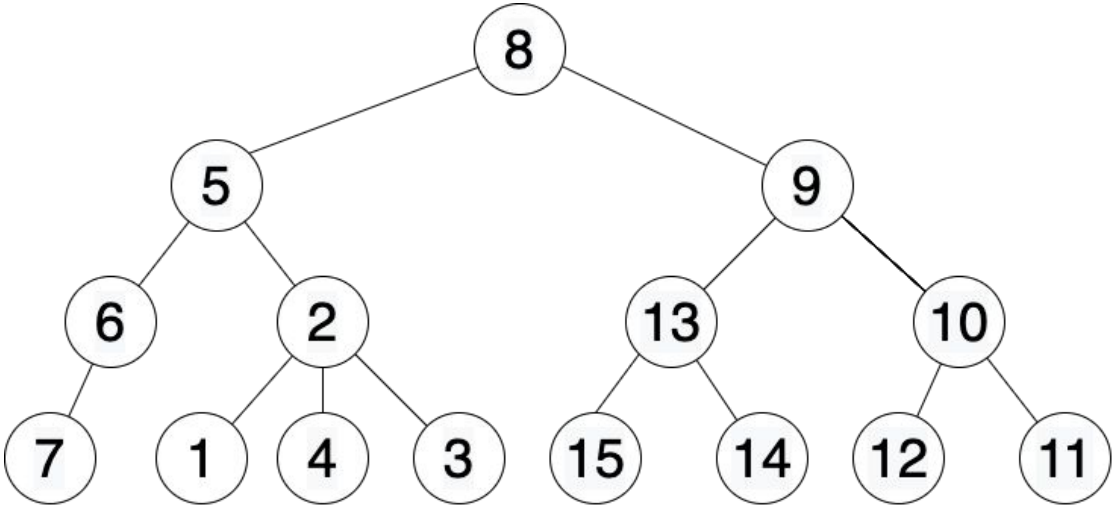
Can we do something with ETT Way-1?

Yes, Point Query

Subtree Update

- E : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
    - No real structure for handling path queries.

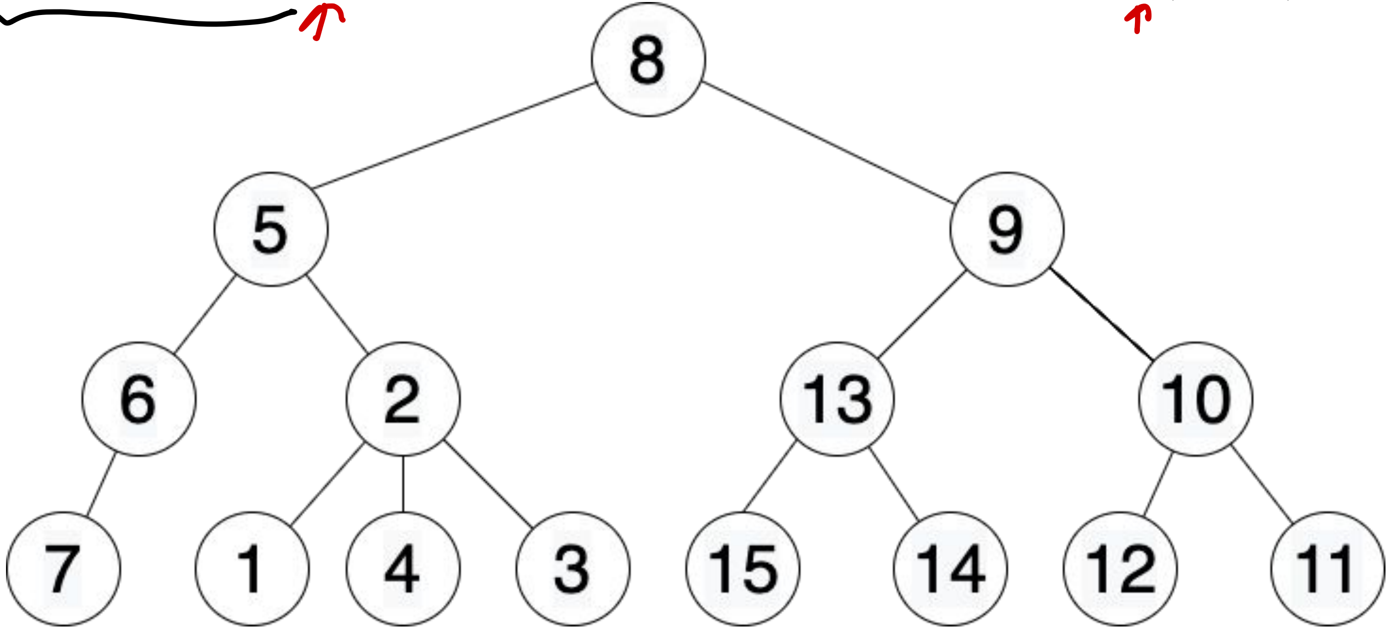
```
void dfs(int x, int p) {
  st[x] = ++T;
  // E[T] = x;
  for (auto y : g[x]) {
    if (y != p) {
      dfs(y, x);
    }
  }
  en[x] = T;
}
```



Euler Tour Trick: Way-2(Double Occurrence)

- Do a DFS and push every node in the Euler Tour Array when you enter and exit the node.
  - E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8](len=30)

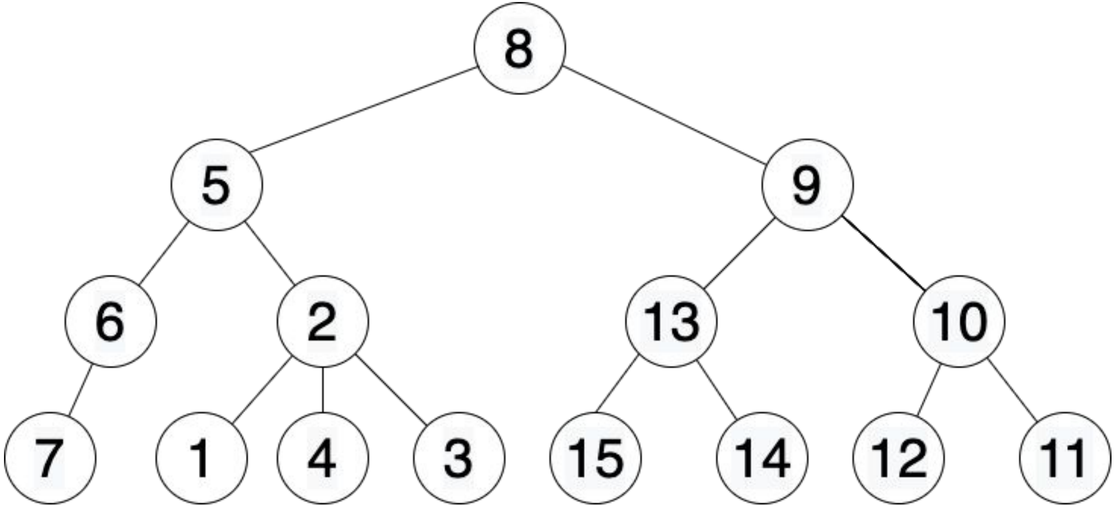
```cpp
void dfs(int x, int p) {
  st[x] = ++T;
  //   ET[x] = x;
  for (auto y : g[x]) {
    if (y != p) {
      dfs(y, x);
    }
  }
  en[x] = ++T;
  //   E[T] = x;
}
```



Way-2(Double Occurrence): Properties

1. All nodes in the subtree(x) now occur twice between [st[x], en[x]]
    - Therefore, most subtree query & update problems can also be solved by way-2
    - Eg: Min / Max (remains the same), Distinct Elements (remains the same), Sum (=query_ans/2)

E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]



Way-2(Double Occurrence): Properties

[L , R]

2. For any path from p → x, [st[p], st[x]] has single occurrence for the ancestors of x which lie on the path and every other node has double occurrence.

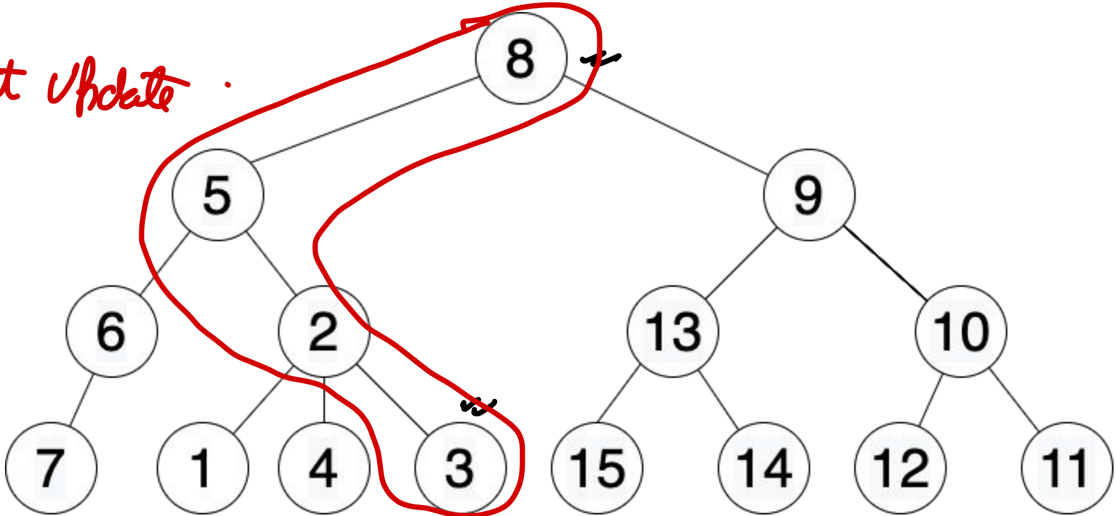
E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]

Path Query ; Point Update .

P → x

St[x] ⇒ + val[x]

en[x] ⇒ − val[x]



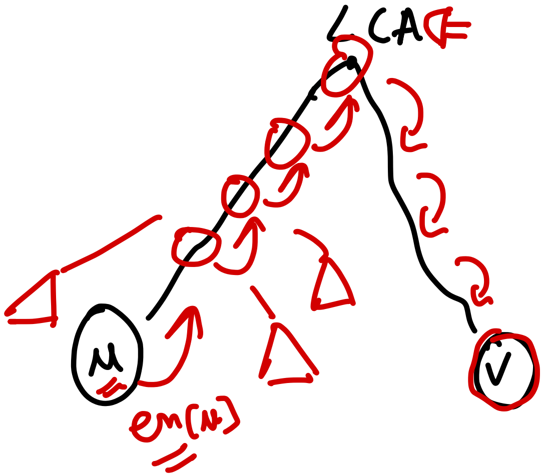
P → x

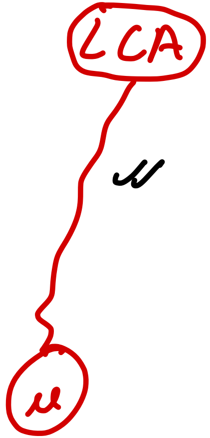
Way-2(Double Occurrence): Properties

[L, R] + LCA.

3. For any path u→ v [en[u], st[v]] has a single occurrence of all path nodes (except LCA) and double occurrence of all non-path nodes.

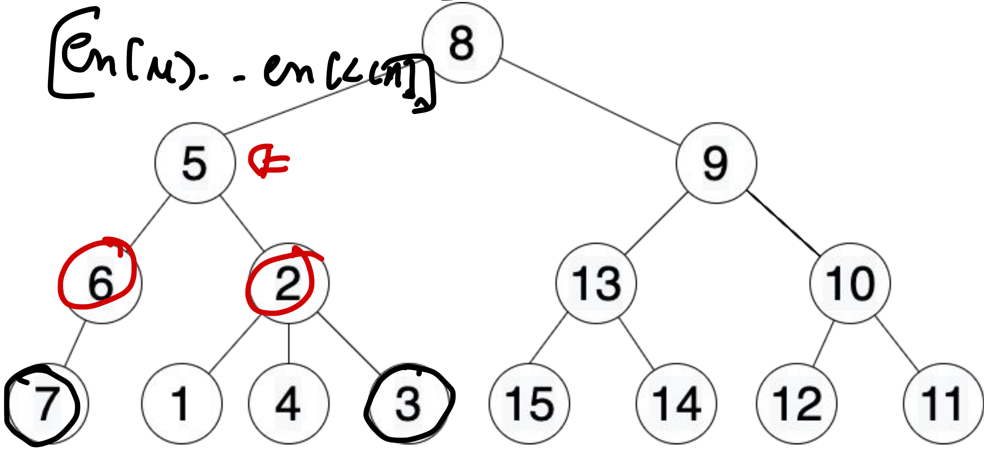
E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]





[St[L] .. St[v]]

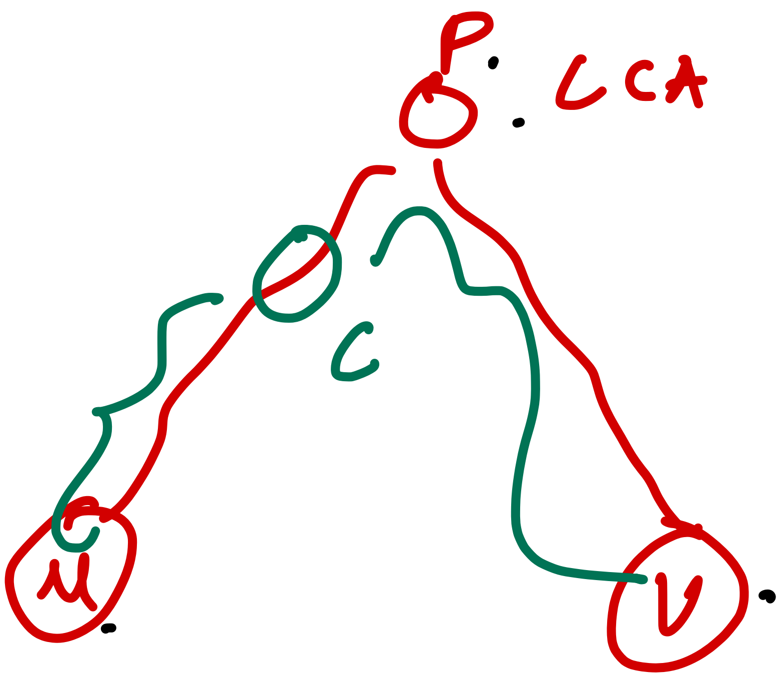
[en[u]. . en[LCA]]



* u → v   Distinct Elements ??

[L, R] ✓

* MO's Algorithm.



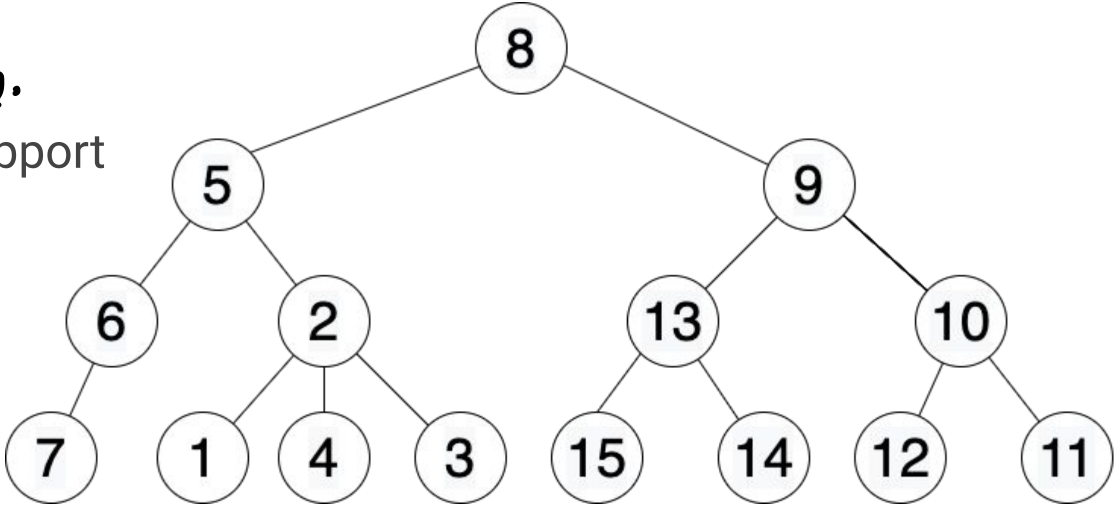
P.. — v

+

P ... v

Path Queries & Point Updates using Way - 2

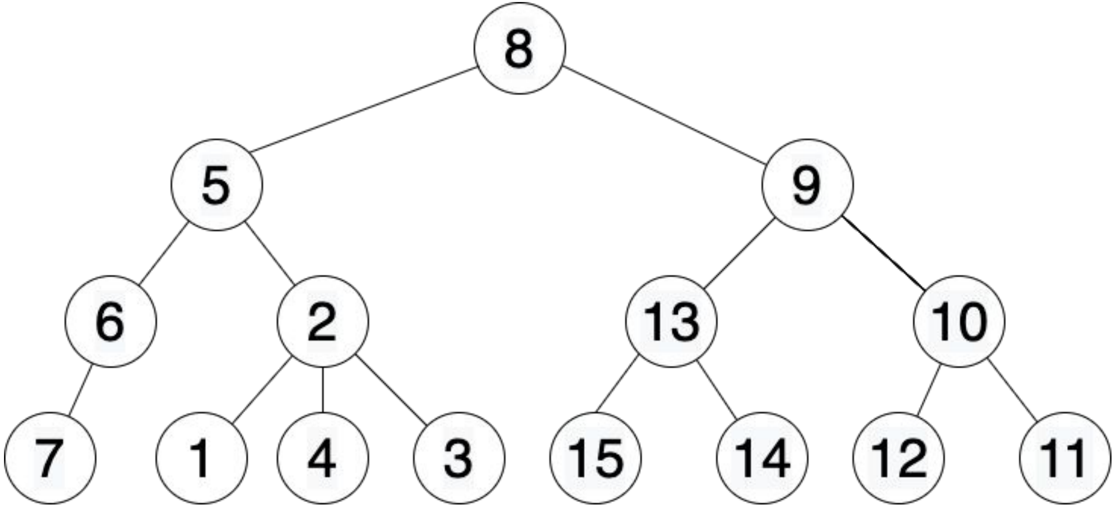
- For Path Query (u, v);
  - CombineAnswer(query(p, u), query(p, v)).
  - Query(p, u) = query(st[p], st[x])
- For Point Update (x, val):
  - update(st[x], val); +val
  - update(en[x], inverse(val)) -val.
- Only query/updates which support inverse are supported!



E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]

Problem - Path Sums & Subtree Updates (Invertible Function)

- Given a tree with N nodes, there are Q queries of the form:
  - U x val -- Add val to all nodes in the subtree of x.
  - Q u v -- Tell the sum of elements on the path from u to v.

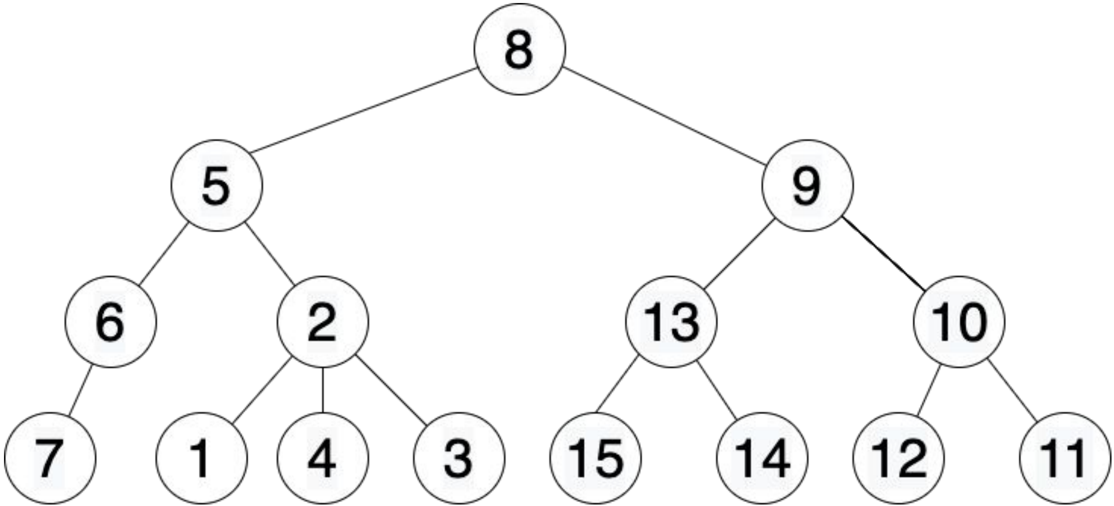


Can we use ETT Way-2 ?

- ETT Way-2 allowed us to answer Point Updates and Path Queries
  - Divide Path Query u → v to u → LCA and LCA → v
  - update(st[x], val[x]); update(en[x], -val[x]);
  - Ans = query(st[LCA], st[u]) + query(st[LCA], st[v]) - val[LCA];

Range Update

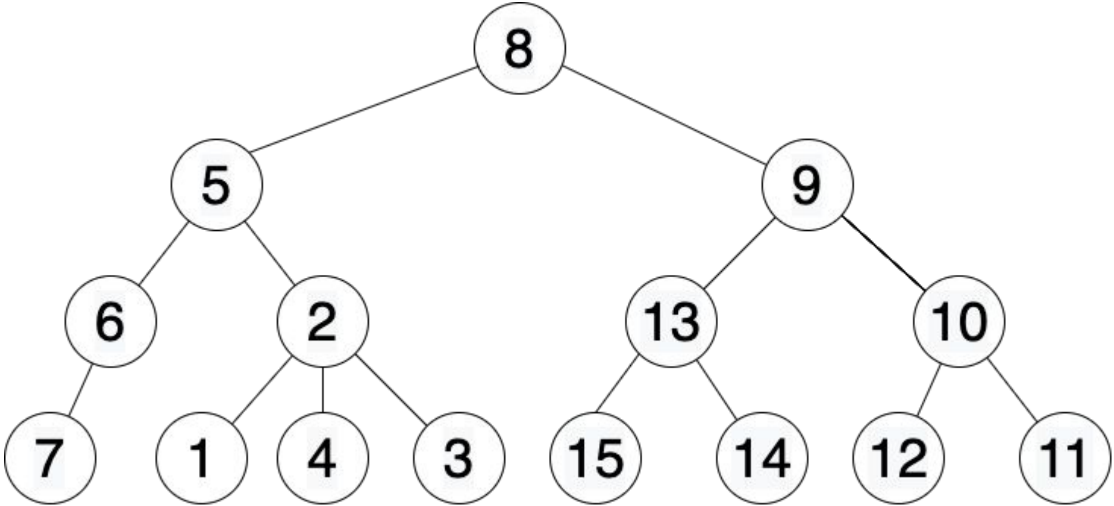
[st[x], en[x]] ⇒ ??



Can we use ETT Way-2 ?

- ETT Way-2 worked because +val exists on st[x] and -val exist on en[x].
- Doing a range update naively will break this.
- Can we do something to handle this?

[st[x], en[x]]



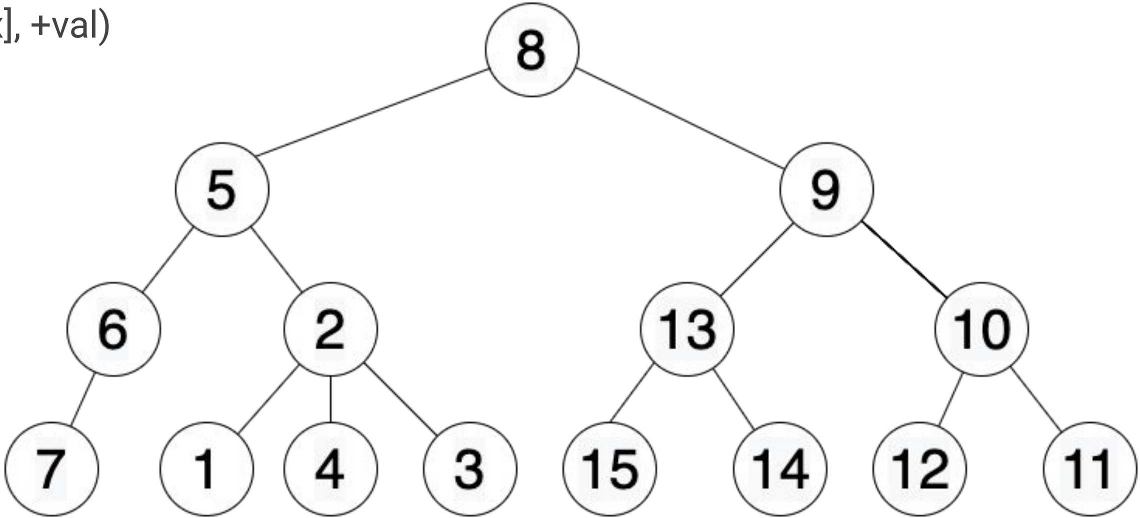
Modifying ETT Way-2 to support Range Updates

- Maintain two different arrays -- one for entry occurrences and other for exit occurrences

A_1 ← +

A_2 ← Inv.

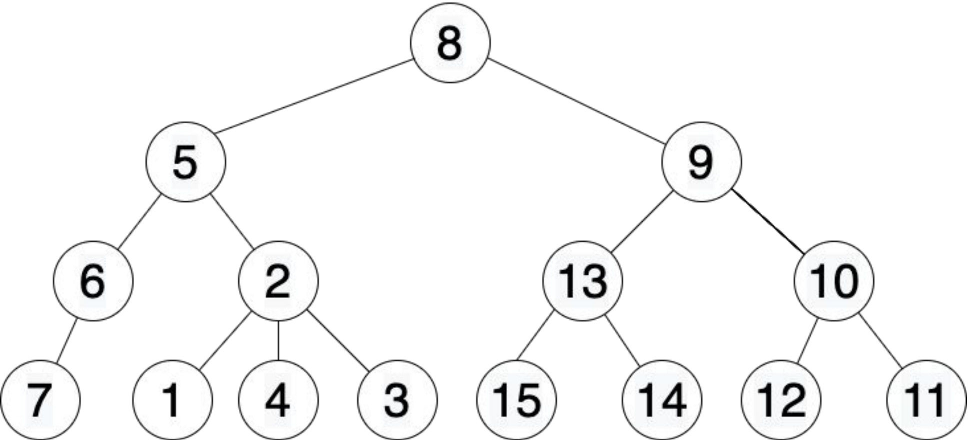
  - Original
    - E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]
    - Query(p, x) : SUM(st[p] … st[x]).
    - Update(x, v) : update(st[x], +val)



Modifying ETT Way-2 to support Range Updates

- Maintain two different arrays -- one for entry occurrences and other for exit occurrences
    - E[0] : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
    - E[1] : [7, 6, 1, 4, 3, 2, 5, 15, 14, 13, 12, 11, 10, 9, 8]

```
void dfs(int x, int p) {
  st[0][x] = ++T[0];
  st[1][x] = T[1];
  for (auto y : g[x]) {
    if (y != p) {
      dfs(y, x);
    }
  }
  en[0][x] = T[0];
  en[1][x] = ++T[1];
}
```



Modifying ETT Way-2 to support Range Updates

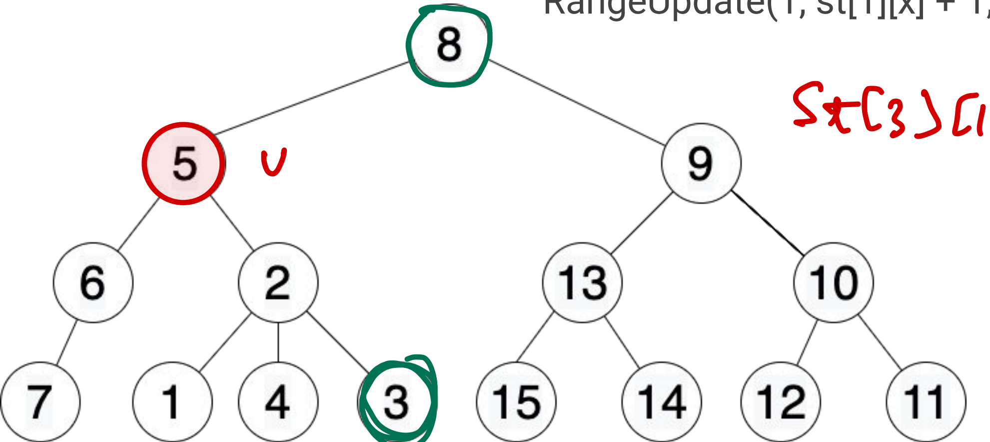
- Maintain two different arrays -- one for entry occurrences and other for exit occurrences

+val $\boxed{6, 7, 1, 4.}$ → Extra. P ⇒ 1

  - E[0] : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
  - E[1] : [$\boxed{7, 6, 1, 4}$ 3, 2, 5, 15, 14, 13, 12, 11, 10, 9, 8] -val.
  - Query(x) = RangeQuery(0, st[0][s]) + RangeQuery(1, st[1][s]);
  - Update(x, val) = RangeUpdate(0, st[0][x], en[0][x]) &

    RangeUpdate(1, st[1][x] + 1, en[1][x])

P → x



St[3][1] = 3

* St[P] → ST[x]

* ST[P] → ST[x] : E[0]

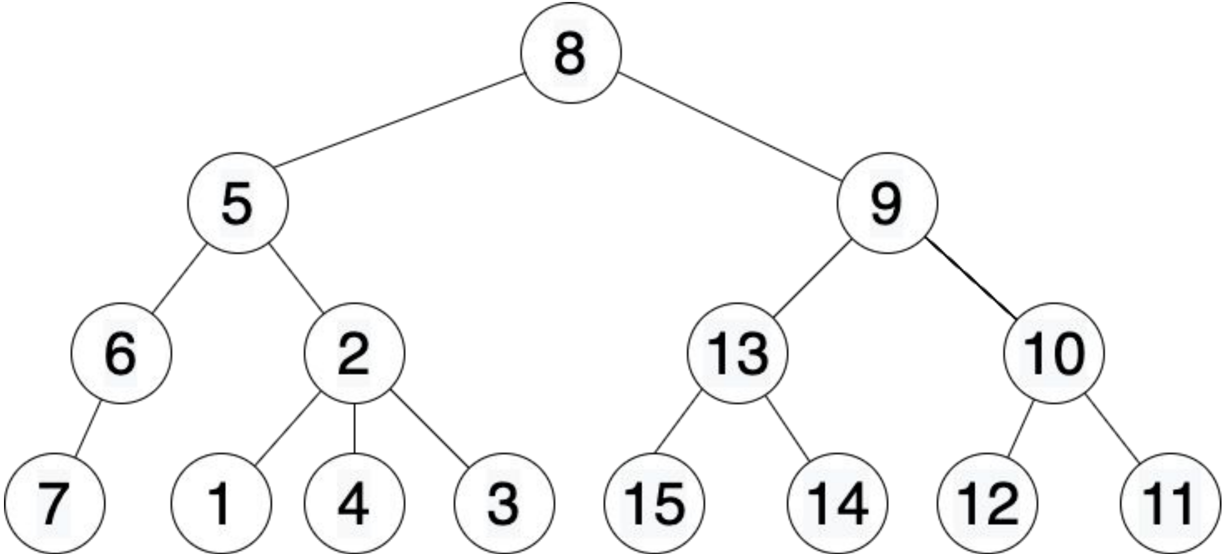
ST[P] + 1 → ST[x] : E[1]

Revisiting LCA Queries

- Given a tree with N nodes and Q queries of the form:
  - u, v → What is the LCA of nodes u & v in the tree?
- Can we do better than O(logN)?

Can we extract LCA info from Way-1 or Way-2?

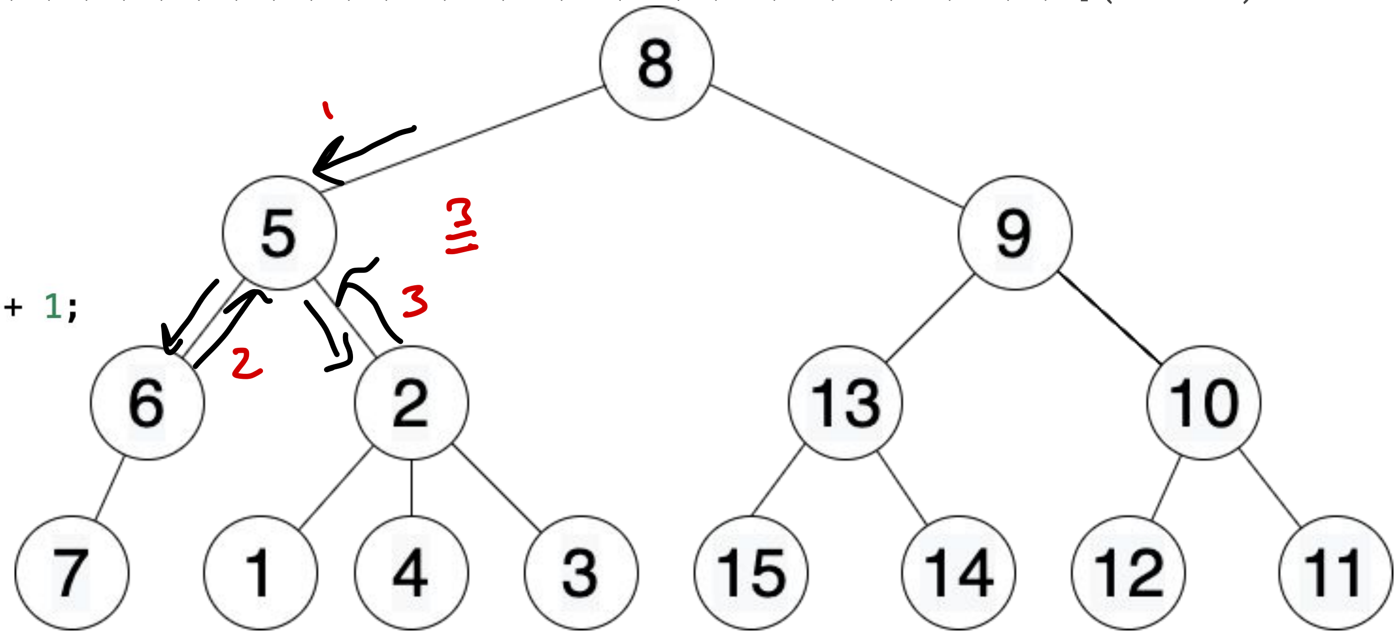
- Way-1: Single Occurrence
  - E : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
- Way-2: Double Occurrence
  - E: [8, 5, 6, 7, 7, 6, 2, 1, 1, 4, 4, 3, 3, 2, 5, 9, 13, 15, 15, 14, 14, 13, 10, 12, 12, 11, 11, 10, 9, 8]



Euler Tour Trick: Way-3 (All Edge Occurrences)

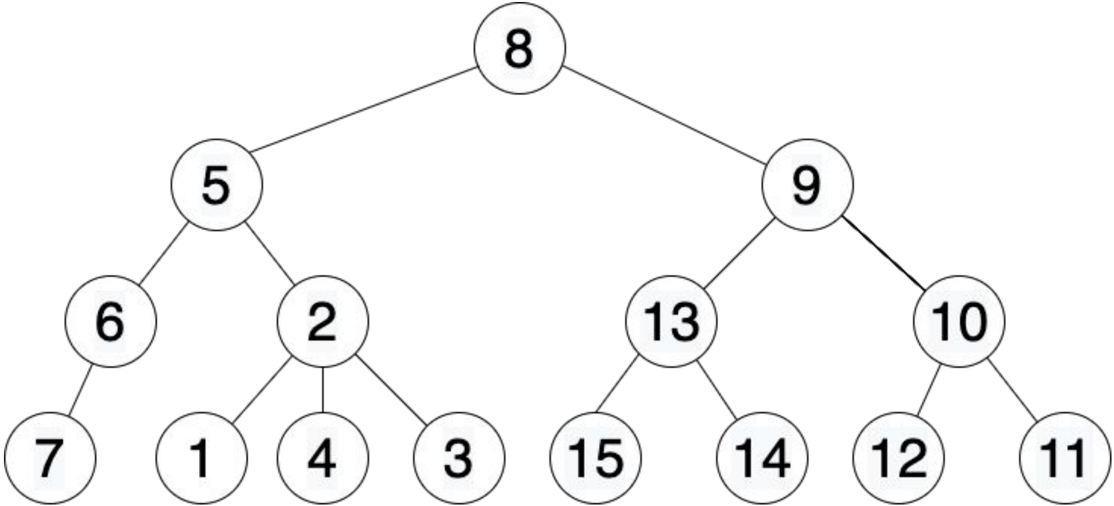
- Do a DFS and push every node in the Euler Tour Array whenever you traverse a directed edge and visit the node (every edge = 2 directed edges)
  - E: [8, 5, 6, 7, 6, 5, 2, 1, 2, 4, 2, 3, 2, 5, 8, 9, 13, 15, 13, 14, 13, 9, 10, 12, 10, 11, 10, 9, 8] (len = 29)

```
void dfs(int x, int p) {
  st[x] = ++T;
  E[T] = x;
  for (auto y : g[x]) {
    if (y != p) {
      level[y] = level[x] + 1;
      dfs(y, x);
      en[x] = ++T;
      E[T] = x;
    }
  }
}
```



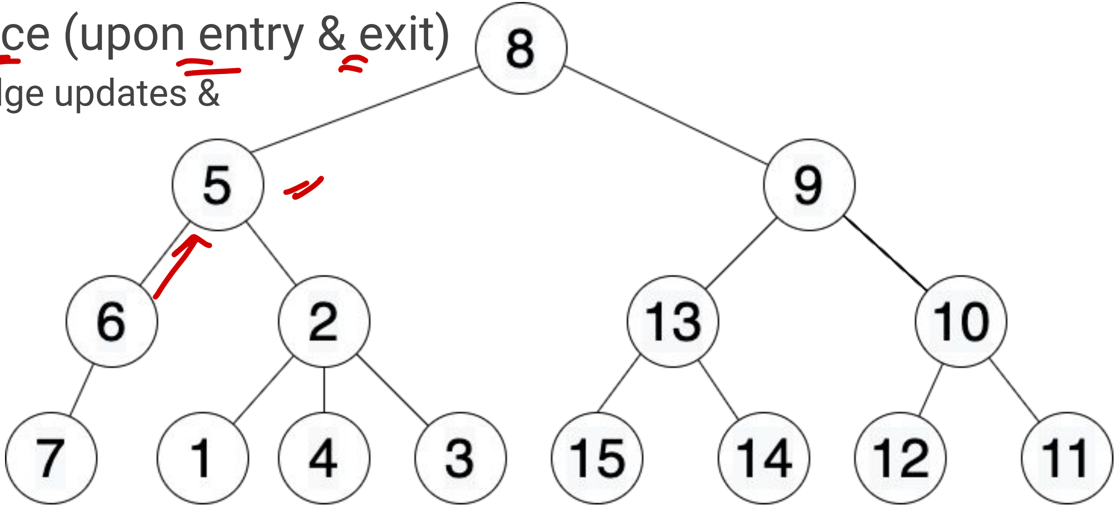
Way-3 (All Edge Occurrences): Properties

- Every node x (except root) will occur deg(x) (+1 for root) times in the ETT array
  - E: [8, 5, 6, 7, 6, 5, 2, 1, 2, 4, 2, 3, 2, 5, 8, 9, 13, 15, 13, 14, 13, 9, 10, 12, 10, 11, 10, 9, 8] (len = 29)
- Subtree(x) still corresponds to [st[x], en[x]], but with multiple occurrences of different nodes.
  - Not very suitable for subtree queries.



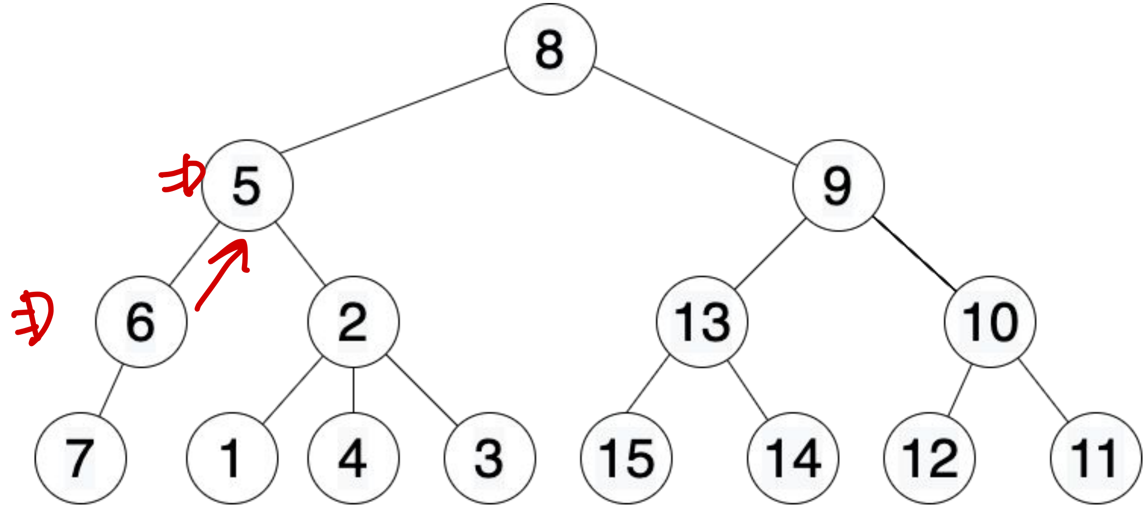
Way-3 (All Edge Occurrences): Properties

- Every node x (except root) will occur deg(x) (+1 for root) times in the ETT array
  - E: [8, 5, 6, 7, 6, 5, 2, 1, 2, 4, 2, 3, 2, 5, 8, 9, 13, 15, 13, 14, 13, 9, 10, 12, 10, 11, 10, 9, 8] (len = 29)
- Subtree(x) still corresponds to [st[x], en[x]], but with multiple occurrences of different nodes.
  - Not very suitable for subtree queries.
- Every edge occurs exactly twice (upon entry & exit)
  - Useful for problems involving edge updates & queries in subtrees!



LCA using ETT Way-3 (All Edge Occurrences)

- For any u, v, all the nodes lying on path from u → v in the tree occur between [st[u], st[v]] (+extra nodes)
  - E: [8, 5, 6, 7, 6, 5, 2, 1, 2, 4, 2, 3, 2, 5, 8, 9, 13, 15, 13, 14, 13, 9, 10, 12, 10, 11, 10, 9, 8]
- To find the LCA of u, v; Find argmin(level[E[x]]) where st[u] <= x <= st[v]
  - This reduces LCA to Range Min Query on E array.

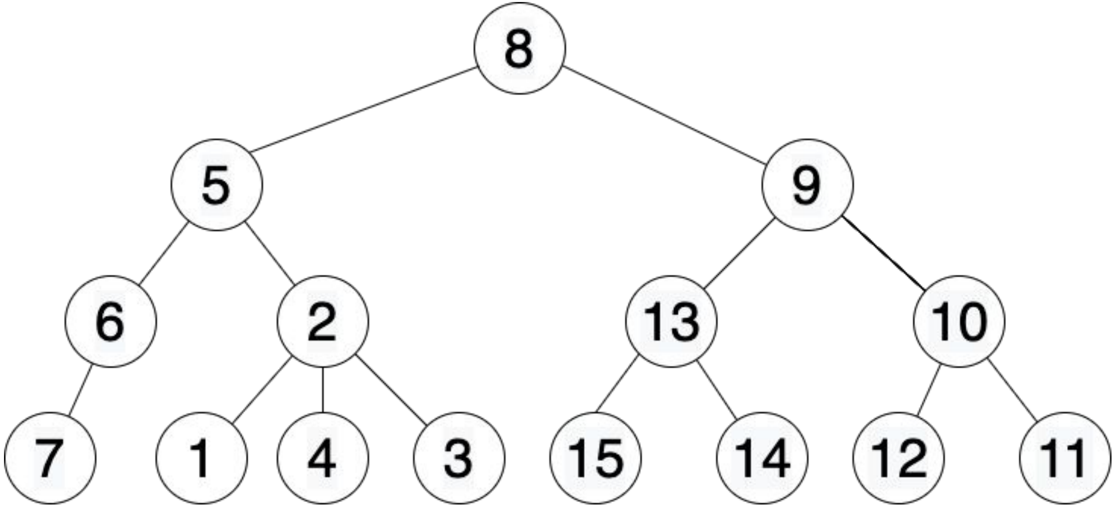


RMQ on Arrays

- RMQ on arrays can be answered in O(1) using O(NlogN) preprocessing using sparse tables.
  - Preprocessing is similar to Binary Lifting / Binary Search (Way - 2)
  - For Query, notice that min(A[L] … A[R]) == min(A[L .. L + 2 ^ i], A[R - 2 ^ i … R])

* $DP[i][j]$

min $i \ldots i+2^j$



ETT Summary

- There are 3 different ways to linearize the tree using Euler Tour Trick.
- Each approach has its own pros and cons and are useful in different scenarios.
  - Way-1 (Single Occurrence) : Subtree Queries & Updates
  - Way-2 (Double Occurrence) : Also useful for
    - Path queries & Point updates for invertible functions.
    - Path queries & Subtree updates for invertible functions.
  - Way-3 (All Edge Occurrences) : This is Way-2 for edge queries & updates. Also reduces LCA to RMQ in array which can be answered in O(1).

Point Update & Path Queries (non-invertible fns)

- Given a rooted tree with N nodes and Q queries of the form:
    - Q U V - Tell the maximum value on the path from U to V.
    - U X V - Set A[X] = V.
- Practice Problem:
    - https://cses.fi/problemset/task/2134

Heavy Light Decomposition

- Break the tree into vertex-disjoint “chains” going from “top” to “bottom”

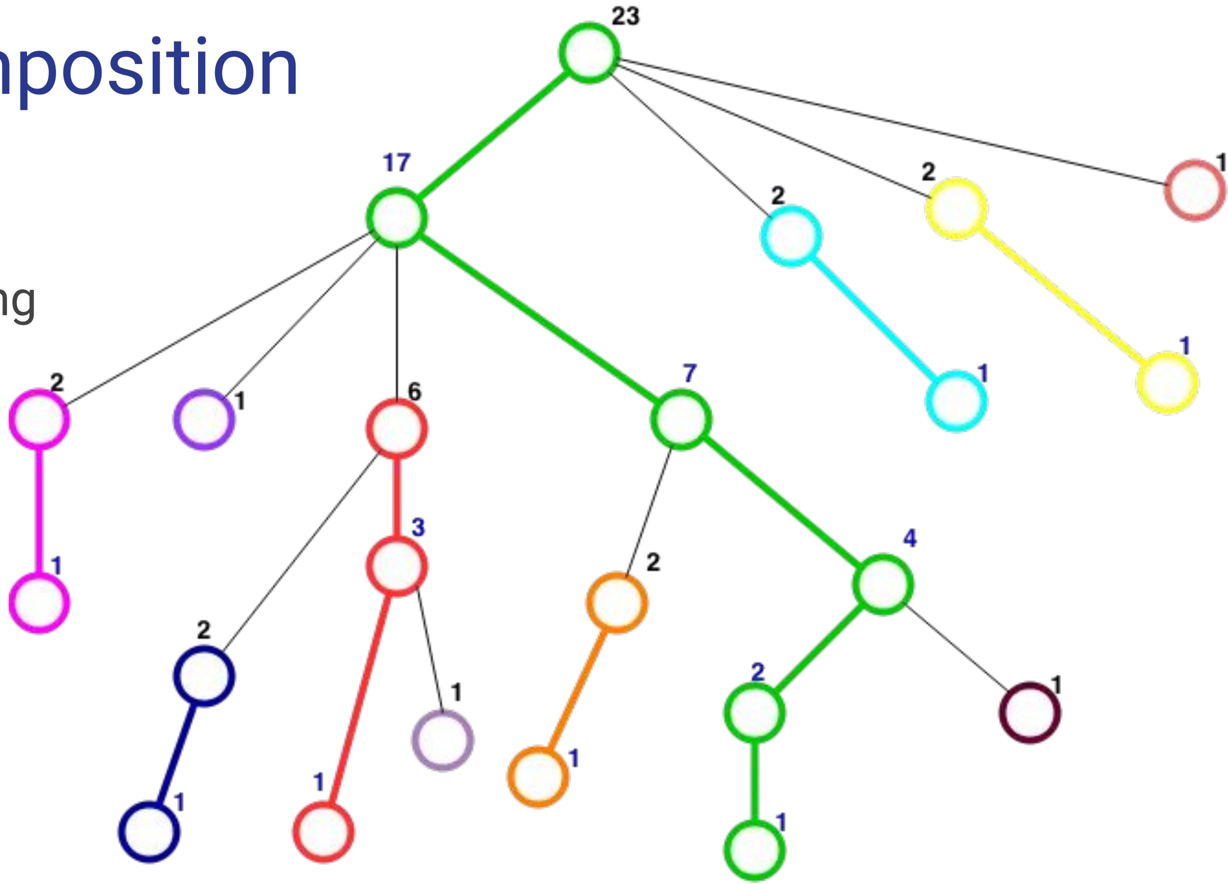


Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

Heavy Light Decomposition

- Break the tree into vertex-disjoint “chains” going from “top” to “bottom”
- For every node, the edge b/w max. size subtree child (**special child**) will be a “heavy” edge, rest will be “light” edges (**normal child**).

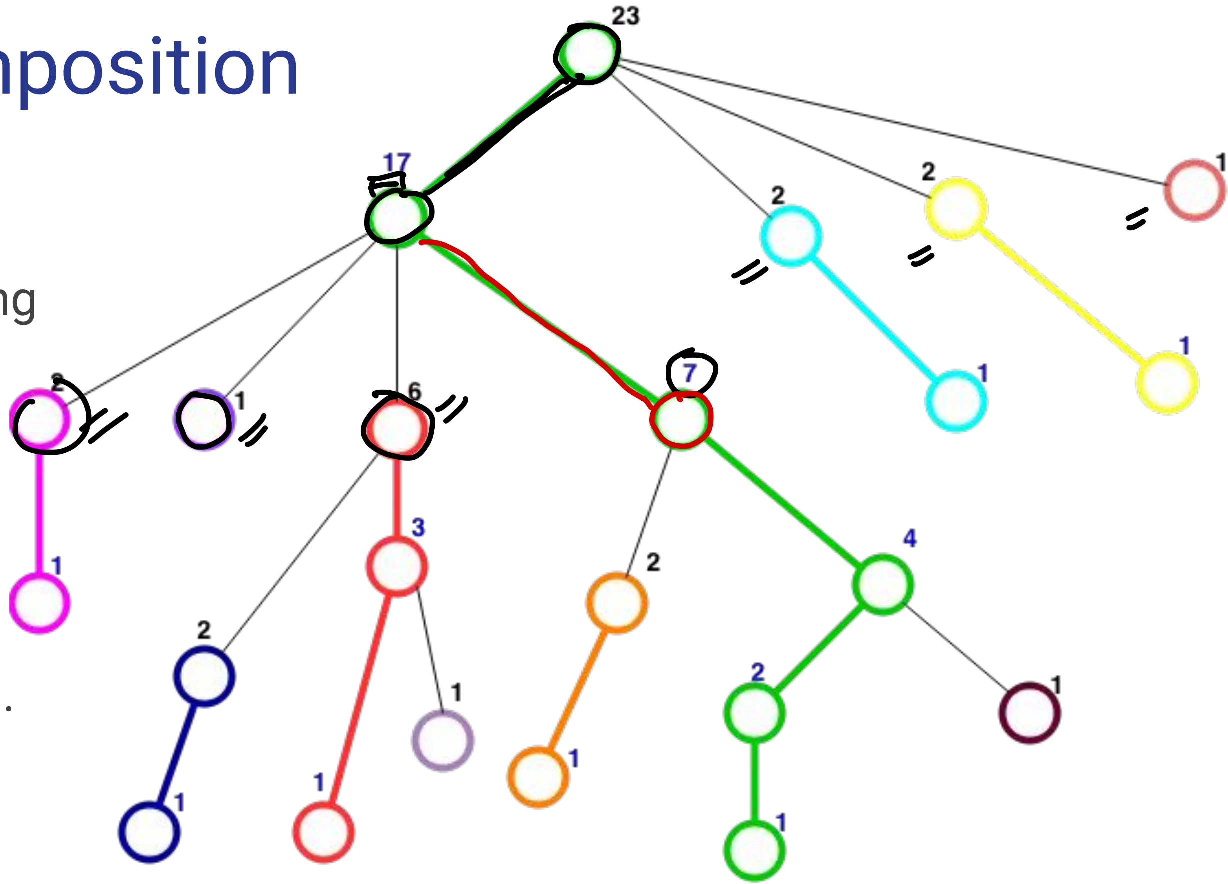


Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

Heavy Light Decomposition

- Break the tree into vertex-disjoint “chains” going from “top” to “bottom”
- For every node, the edge b/w max. size subtree child (**special child**) will be a “heavy” edge, rest will be “light” edges (**normal child**).
- Every light edge connects two different chains / a new chain starts after every light edge.

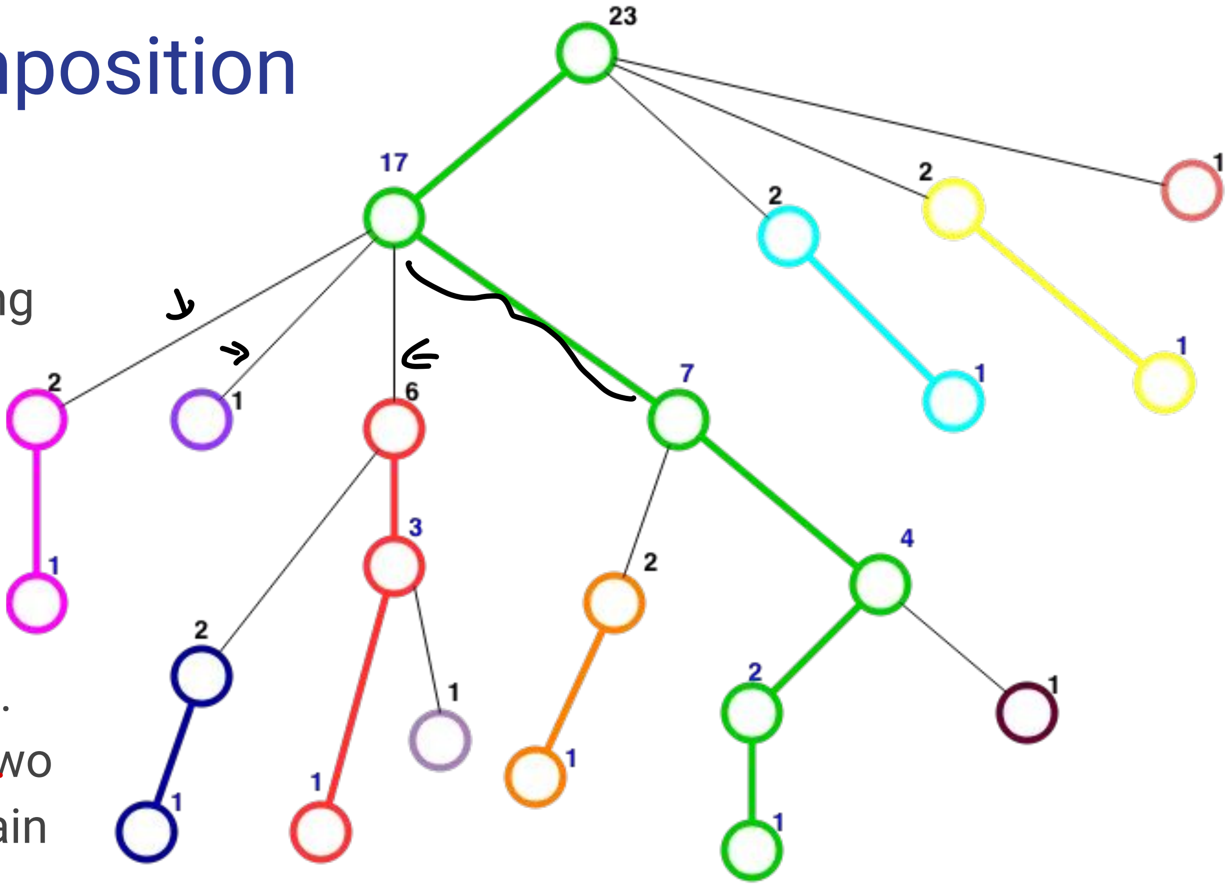


Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

HLD Visualization

Green chain

HLD: [1, 2, 9, 16, 23, 21] [8, 14, 18] [ ]

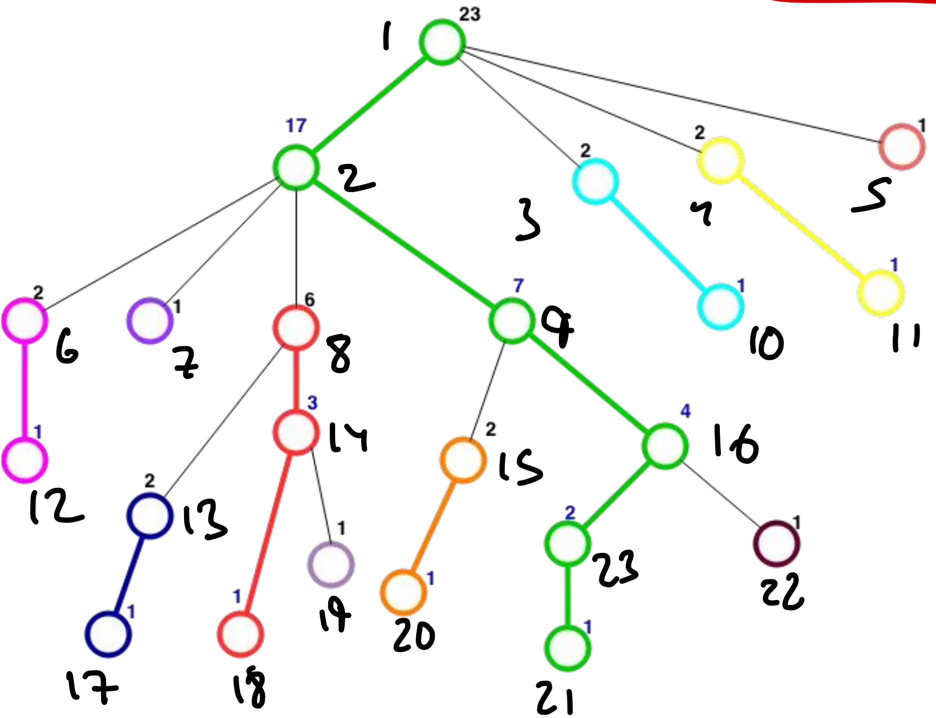


Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.

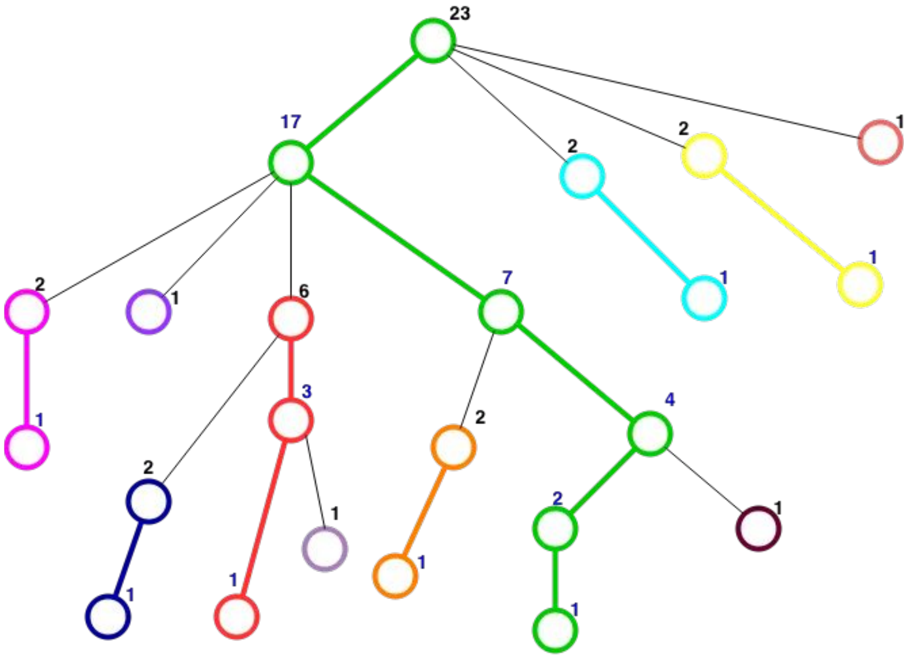
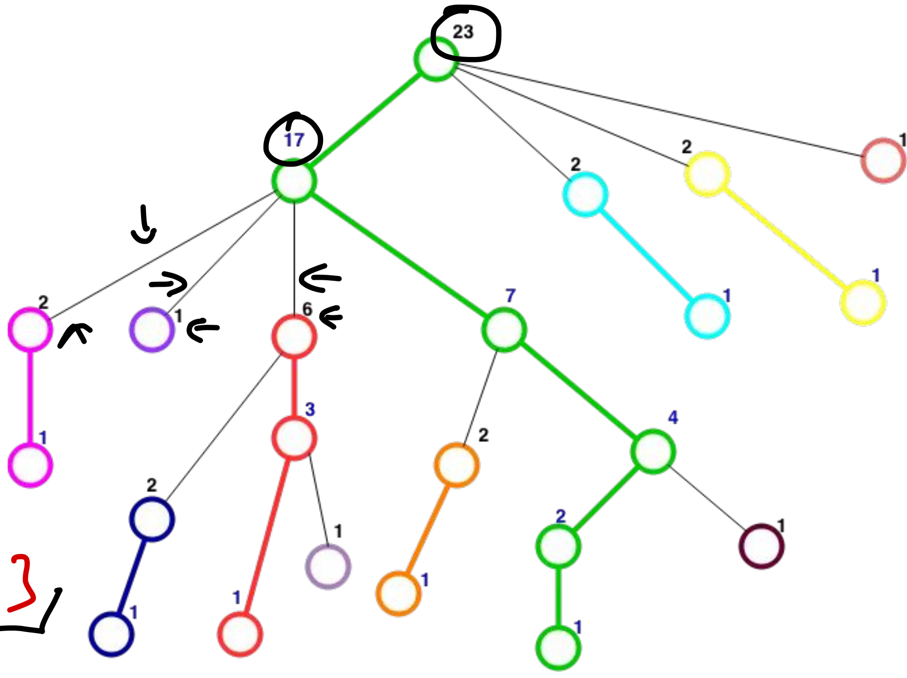


Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

$Sub[y] \geq Sub[z]$

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.



$Sub(x) = Sub[y] + \underbrace{Sub(z) + \dots [\ \ ]}_{\leq Sub[y]}$

$Sub(x) \geq 2 * Sub[z] \quad \forall z \neq y$

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.
- Therefore, we can go up from any node **x** to it's ancestor node **p** by changing at-most logN chains.

↳ Traversing a "light" edge.

O(log N) ~ O(N) ⇏ root.

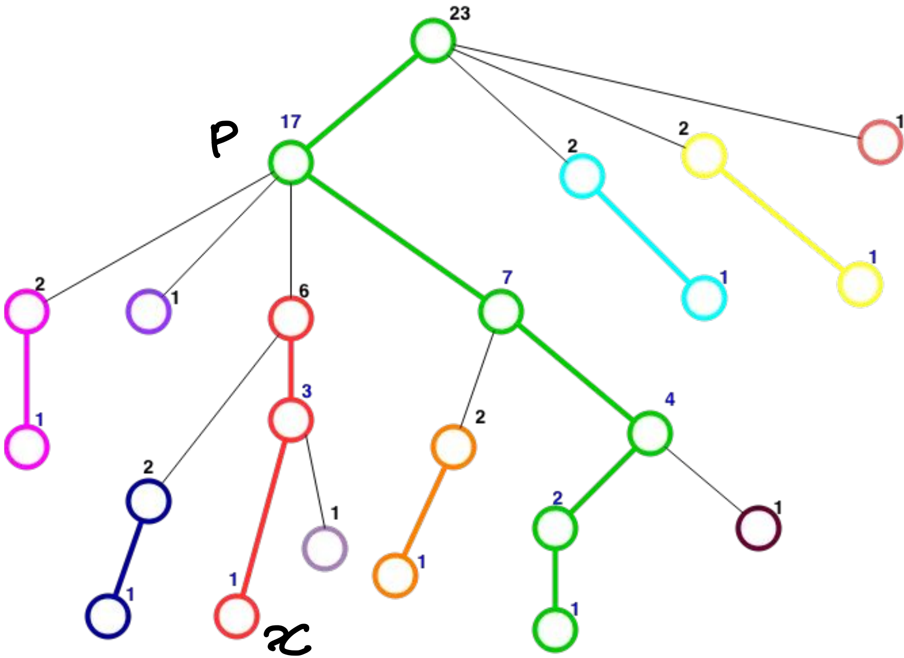


Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.
- Therefore, we can go up from any node **x** to it's ancestor node **p** by changing at-most logN chains.
- Any path A -- B can be written as A -- LCA + LCA -- B; and hence can be traversed by changing at-most 2 * logN chains.

$O(\log N) * O(\log N)$

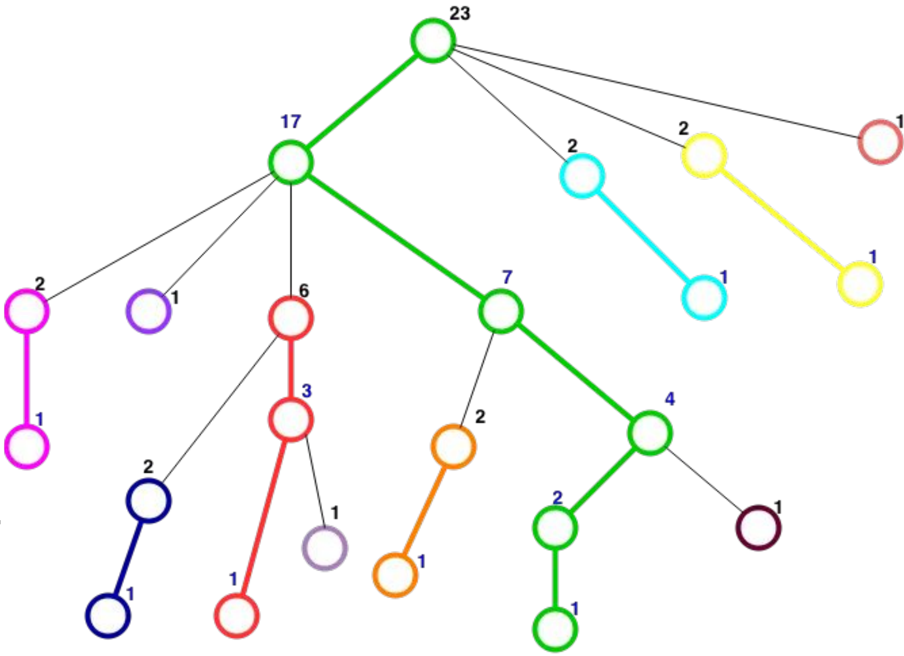


Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Steps to support path updates / queries

- Decompose the tree into chains via HLD.
- Linearise the chains into an array and build a Data Structure on the array that supports range queries / updates. ⇒ ST / SD / ...

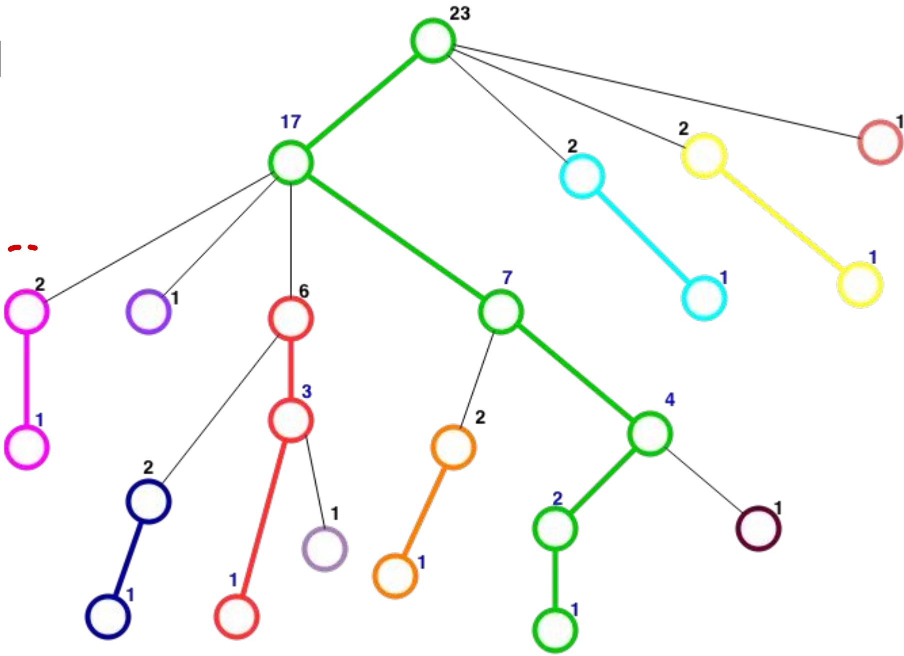


Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Steps to support path updates / queries

- Decompose the tree into chains via HLD.
- Linearise the chains into an array and build a Data Structure on the array that supports range queries / updates.
- For any path query/update b/w nodes A & B; process it as a query/update on O(logN) different ranges in the linearised array -- corresponding to O(logN) chains that we need to traverse while going from A -- LCA -- B in the original tree.
- Therefore, total time taken will be O(logN * TimeTakenByLinearDS)

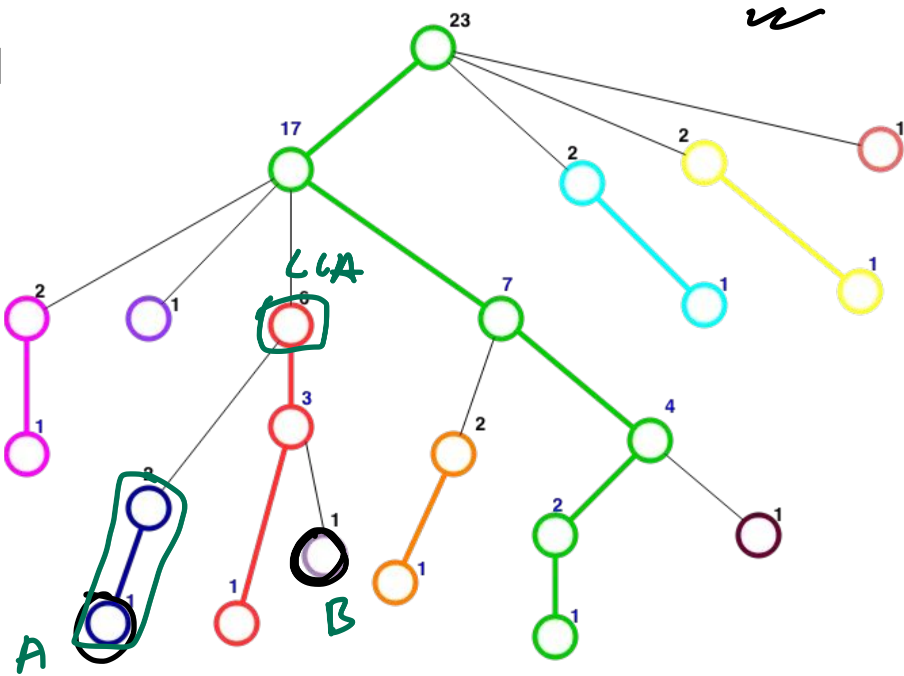


Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

Heavy Light Decomposition - Implementation

```
void dfs_sz(int x, int p) {
  sz[x] = 1;
  par[x] = p;
  head[x] = x;
  for (auto y : g[x]) {
    if (y != p) {
      dfs_sz(y, x);
      sz[x] += sz[y];
      if (sz[y] > sz[sc[x]]) sc[x] = y;
    }
  }
}
```

* sc[x] = 0

* sz[sc[x]] = 0

dfs_sz(1,1);

dfs_hld(1,1);

```
void dfs_hld(int x, int p) {
  st[x] = ++T;  -x;
  ST::A[T] = val[x];
  // dfs on heavy edge.
  if (sc[x]) {
    head[sc[x]] = head[x];
    dfs_hld(sc[x], x);
  }
  // dfs on light edges.
  for (auto y : g[x])
    if (y != p && y != sc[x])
      dfs_hld(y, x);
  en[x] = T;
}
```

LCA Using HLD

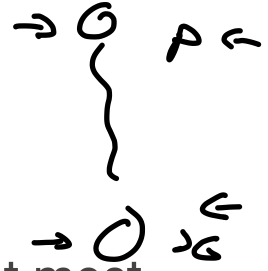
- Given a rooted tree with N nodes and Q queries of the form:
  - Q U V - Tell the LCA of U & V.
- Practice Problem:
  - https://cses.fi/problemset/task/1688

bool anc(p, x) : $O(1)$

$st[p] \le st[x]$

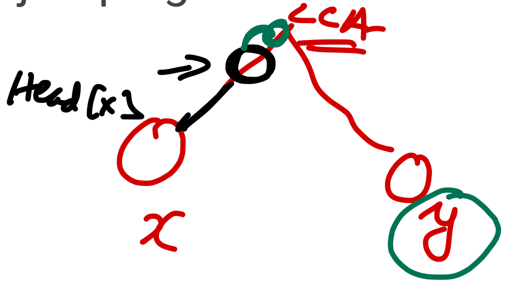
$en[p] \ge en[x]$

DP[ ][ ]



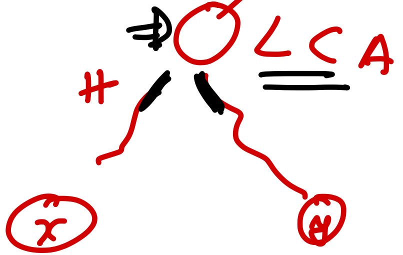
LCA Using HLD

- HLD has enough information for us to compute LCA by jumping at-most O(logN) chains.




```
int lca(int x, int y) {
    if (anc(x, y)) return x;
    if (anc(y, x)) return y;
    while (!anc(par[head[x]], y)) x = par[head[x]];
    while (!anc(par[head[y]], x)) y = par[head[y]];
    x = par[head[x]];
    y = par[head[y]];
    return anc(x, y) ? y : x;
}
```

$O(1)$



Point Update & Path Queries (non-invertible fns)

HLD.

- Given a rooted tree with N nodes and Q queries of the form:
  - Q U V - Tell the maximum value on the path from U to V.
  - U X V - Set A[X] = V.
- Practice Problem:
  - https://cses.fi/problemset/task/2134 (simpler version)

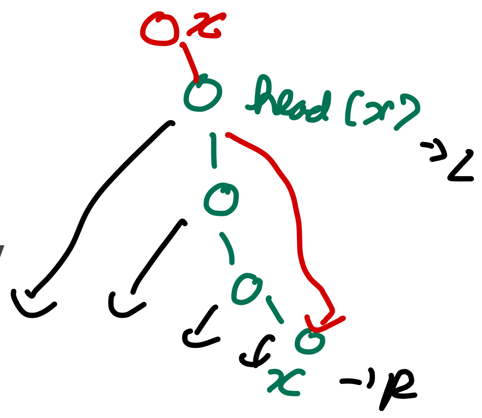
Path Queries & Updates (non-invertible fns)

- Linearize the tree using HLD and build a Segment Tree over the linearized values.
- Every path query / update will reduce to range query/updates for O(logN) different ranges in the linearized array + segment tree.

$O(\log^2 N)$

st[x] = ++T.
A[T] = val[x]

Eg: Path Min Queries



- Query from u → v can be divided into u → LCA & LCA → v
- ans = min(query_up(u, LCA), query_up(v, LCA));

```
int query_up(int x, int p) {
  int ans = 0;
  while (head[x] != head[p]) {
    ans = max(ans, ST::query(st[head[x]], st[x] + 1));
    x = par[head[x]];
  }
  ans = max(ans, ST::query(st[p], st[x] + 1));
  return ans;
}
```

[L, R)

[L, R].

Practice Problems

- (Div1 E) https://codeforces.com/contest/226/problem/E DIV1 E
  - HLD + Persistent Segtree
- (Div1 E) https://codeforces.com/contest/487/problem/E DIV1 E
  - Build Block Cut Tree of the given Graph to reduce it to a path query & update problem.
- https://www.hackerearth.com/challenges/competitive/july-clash-15/algorithm/upgrade/
  - Treap over HLD

What functions can be computed using HLD?

- We need a way to combine answer of O(logN) different ranges “quickly”.
- So, any function for which you can combine answer of two ranges to get the answer of combined range “quickly” are supported.
    - i.e. Most functions for which you can build a SegTree are supported.
- Functions like distinct elements on the path are not supported because you cannot easily combine answer of two different ranges.

Subtree Update & Path Queries (non invertible fn)

- Given a rooted tree with N nodes and Q queries of the form:
  - A X V - Add V to all nodes in the subtree of X.
  - Q X Y - Report maximum value on the path from A to B.
- Practice Problem:
  - https://www.hackerrank.com/challenges/subtrees-and-paths/problem

* Subtree U & Q ⇒ Any Fn. : ETT

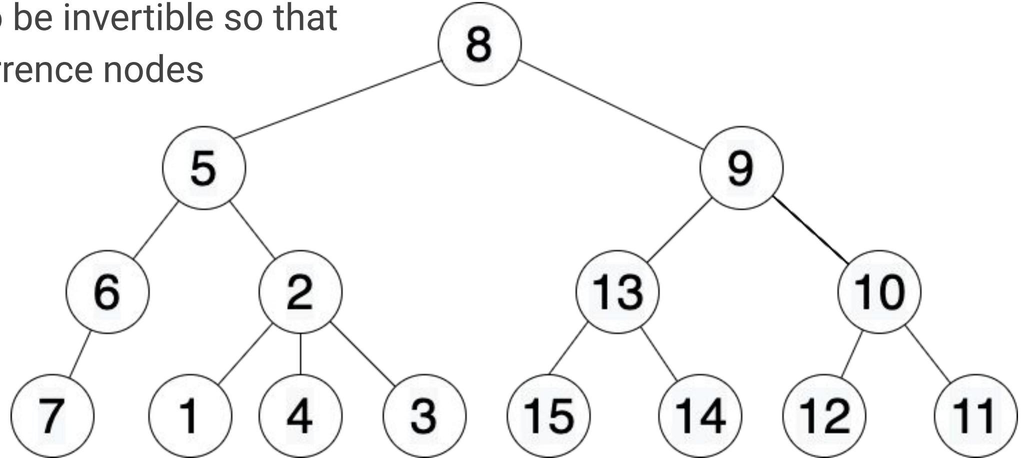
* Path U & Q ⇒ Any Fn : HLD

* Subtree U & Path Q ⇒ Invertible : ETT.

*

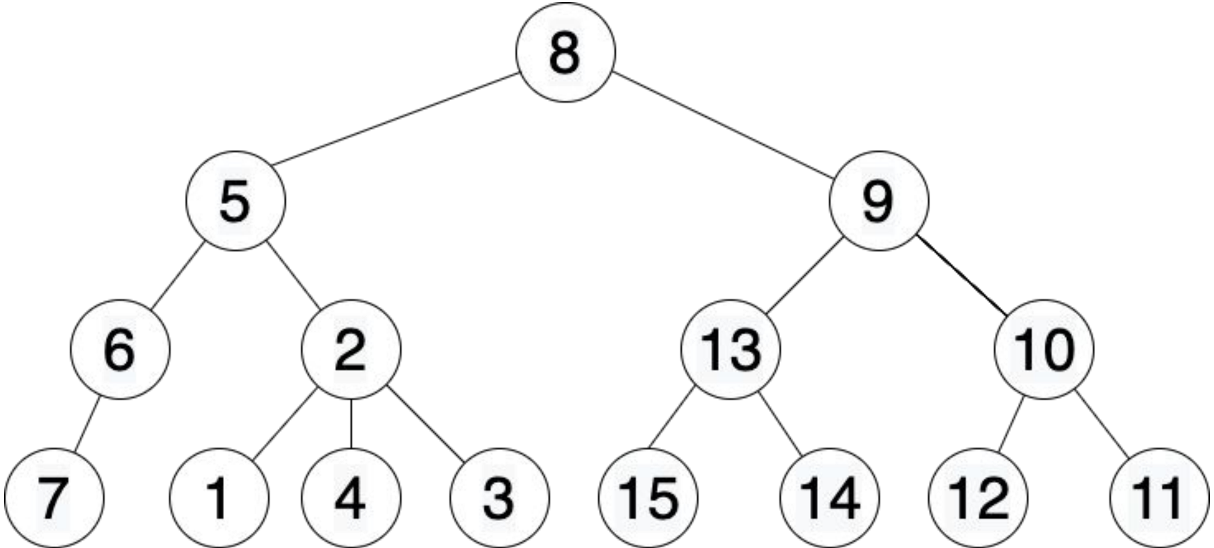
Can we use ETT Way-2?

- ETT Way-2 could support Subtree Updates & Path Sum Queries
  - E[0] : [8, 5, 6, 7, 2, 1, 4, 3, 9, 13, 15, 14, 10, 12, 11]
  - E[1] : [7, 6, 1, 4, 3, 2, 5, 15, 14, 13, 12, 11, 10, 9, 8]
  - Query(x) = RangeQuery(0, st[0][s]) + RangeQuery(1, st[1][s]);
  - Update(x, val) = RangeUpdate(0, st[0][x], en[0][x]) & RangeUpdate(1, st[1][x] + 1, en[1][x])
- Can we modify the idea to support Subtree Updates & Path Max Queries?
  - No! The approach requires function to be invertible so that contribution of non-path double occurrence nodes can be removed quickly.



Can we use HLD?

- HLD supports path updates and path queries, but we need subtree updates!
- Subtree updates are usually supported via ETT?

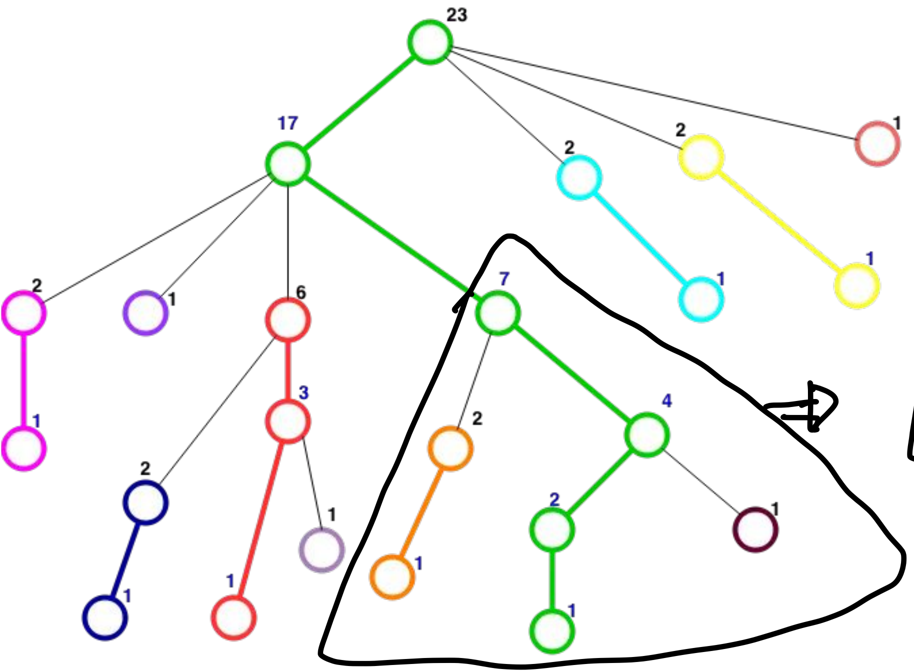


Combining HLD & ETT

- HLD is also a Way-1 ETT!
  - We do a DFS ordering and push nodes in a linearized array when we enter the node for the first time! → This was Way-1 ETT.
  - Only difference in HLD is that we traverse the nodes in a specific order (heaviest child first).

```
void dfs_hld(int x, int p) {
  st[x] = ++T;
  ST::A[T] = val[x];
  // dfs on heavy edge.
  if (sc[x]) {
    head[sc[x]] = head[x];
    dfs_hld(sc[x], x);
  }
  // dfs on light edges.
  for (auto y : g[x])
    if (y != p && y != sc[x])
      dfs_hld(y, x);
  en[x] = T;
}
```

Combining HLD & ETT - Visualization



A.

[St[r], en[r]]

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

Subtree Update & Path Queries Solution

- Linearize the tree using HLD & store st[x] and en[x] for every node (ETT).
- Build a lazy segment tree on the linearized array. ⇐
- For update(x, val):
  - ST::update(st[x], en[x] + 1, val);

[L, R) ⇐ +val.

- For Query(x, y):
  - ans = max(query_up(x, LCA), query_up(y, LCA))
- See code for more details!

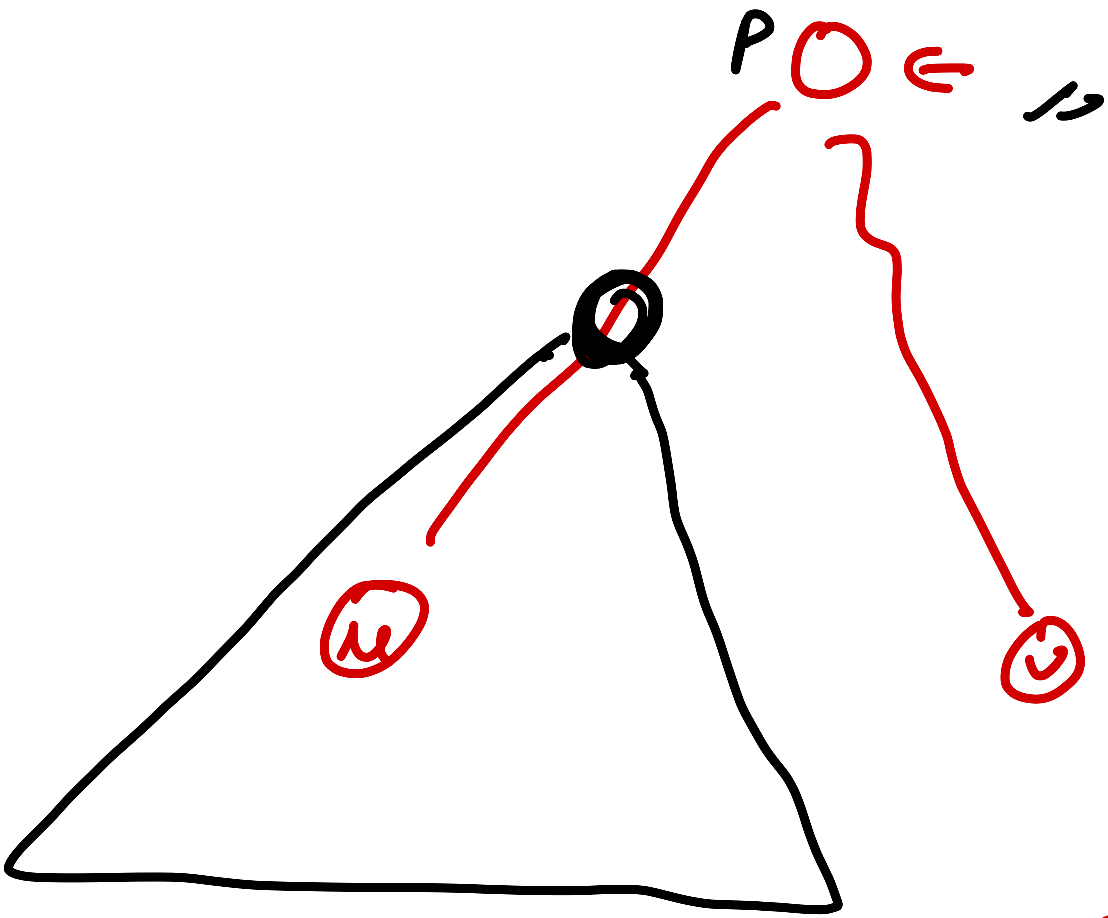
Conclusion

- HLD has ETT (Way-1, Single Occurrence) information disguised within its structure!
- It can hence be used to support all four
  - Path Updates
  - Path Queries
  - Subtree Updates
  - Subtree Queries
- The HLD idea of choosing a "special" child can be extended to other problems.

(v, d)

$\sum i * A[x]$

Updates.



u, v

$\min(PS[u], PS[v])$

QueryUP