Codechef Learn, Episode 2
Lec-2 **SRD on Trees - Supernode Trees**

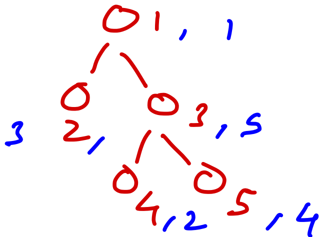
https://youtu.be/8VHWdNnP3h4

tanujkhattar@

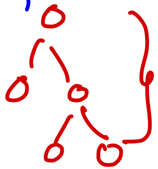
SRD on Trees, Part-1 Recap

| 1 | 2 | 3 | 4 | 5 |

| 1 | 3 | 5 | 2 | 4 |



- Perform an ETT/HLD of the given tree to ensure $\boxed{P[x] < x}$ in the linearized array.
- Do Array Square Decomposition on the linearized array s.t.
- Edges within the same block form a forest of Trees with at-most sqrt(N) nodes.
- Edges across blocks ($x \rightarrow y$) start at root node (x) of a decomposed tree and end at some node (y) lying in another block to the left (BLOCK[y] < BLOCK[x])
- This structure can be used to support
  - Subtree updates → Reduce to lazy range updates [L, R] on the linearized ETT array.
  - Path Queries → Answered by spending O(B) time in blocks of x, y and LCA and O(1) time per block for every block in between.

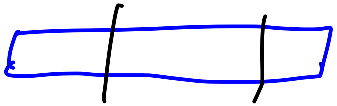


$O(1)$

$O(B + N/B)$

$\approx O(\sqrt{N})$

$O(\sqrt{N} \log N)$



Can we support Path Updates as well?

L R

Array.

X

Merge Sort Tree $O(\log^2 N)$

OR

Persistent Segment Tree $O(\log N)$

- Given a tree with N elements, support the following operations
  - Query U V X : Tell the minimum element >= X on the path from U to V
  - Update U V ~~V~~ A: Add ~~V~~ A to all elements on the path from U to V.
- https://www.codechef.com/problems/TRS

$\geq X$: Edge Decomposition

α α α Tree. α α α

$O(N)$ / Query : Brute Force.

* HLD.

* SRD on Trees P1 ⇒ Subtree Update : Lazy

\+

Path Queries : SRD on ETT

Can we support Path Updates as well?

- We can support subtree updates because they reduce to a single range update in the ETT array.
  - We spend O(sqrt(N) * f(B)) time to process 1 range update [L, R]
  - f(B) is the time taken by the DS maintained at each node + lazy propagation.
- Updates on paths would require us to maintain another structure because ETT doesn't store path information by default.
  - Any ideas ?

A.SRD + HLD ??

A: $\underset{L}{[}\ - - - \underset{R}{]}$: $\sqrt{N}\log N$

$\log N = HLD$

$\sqrt{N} = [L, R] : \sqrt{N} * f(B)$

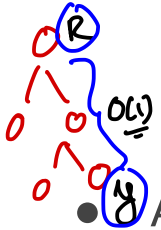
$f(B) = \log N$

Multiset

Can we support Path Updates as well?

$\boxed{\sqrt{N} * \log N} * \log N$

- We could do HLD and build a sqrt structure on top of it but then updates would take O(logN * sqrt(N) * f(B)) $\Rightarrow$ $O(\sqrt{N} * \log^2 N)$
  - This is because HLD divides a path query into O(logN) different [L, R] ranges. $O(\log N)$
  - Hence, we would need to update O(logN) different [L, R] ranges via sqrt decomp on array.
  - Even though union of the [L, R] ranges <= N, processing O(logN) individual blocks in which L, R lie would be expensive if done naively.



- Another problem is that we need to process the decomposed trees s.t. we can efficiently query the path x → root[x] for any given x in the decomposed tree.
  - Answering "Min element >= V on the path from x → root[x]" efficiently via precomputation would again require additional complex processing like persistent segment tree ending at each node built using it's parent node.

? X

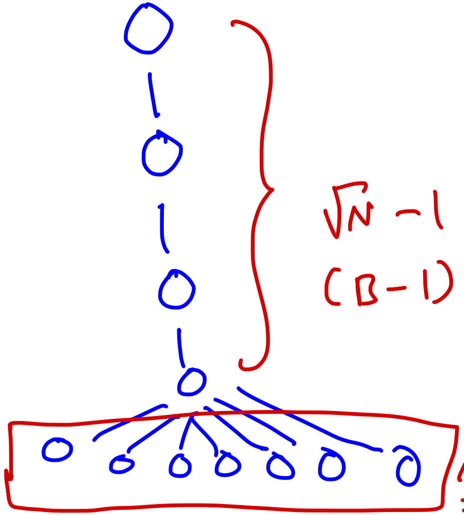
- We probably need something else for complex path queries & updates!

$B \approx \sqrt{N}$

Supernode Trees Introduction

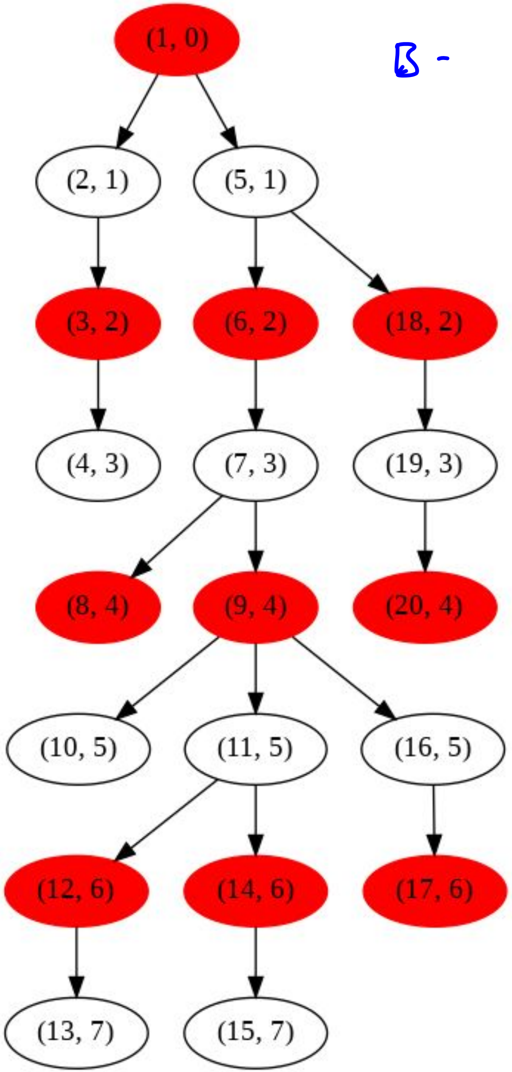
- Mark the nodes with level % B == 0 as special.
- How many special nodes can we have?
  - O(sqrt(N)) ?
  - O(N) ?

N nodes



Level = √N / B

Rooted Tree



B -

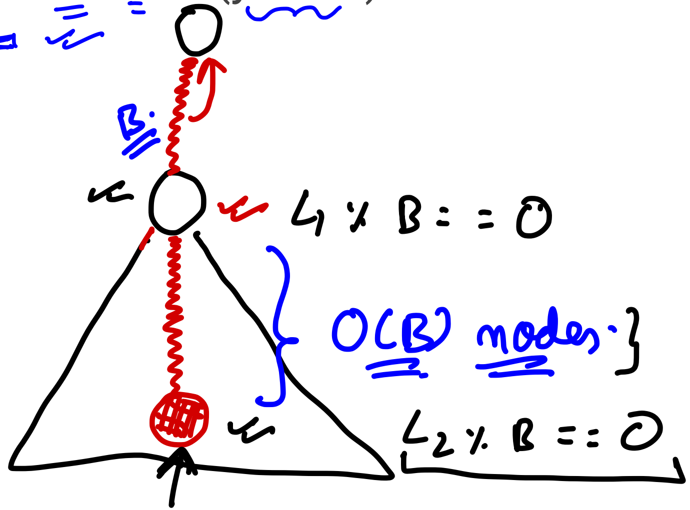
L[N] % B == 0 &&

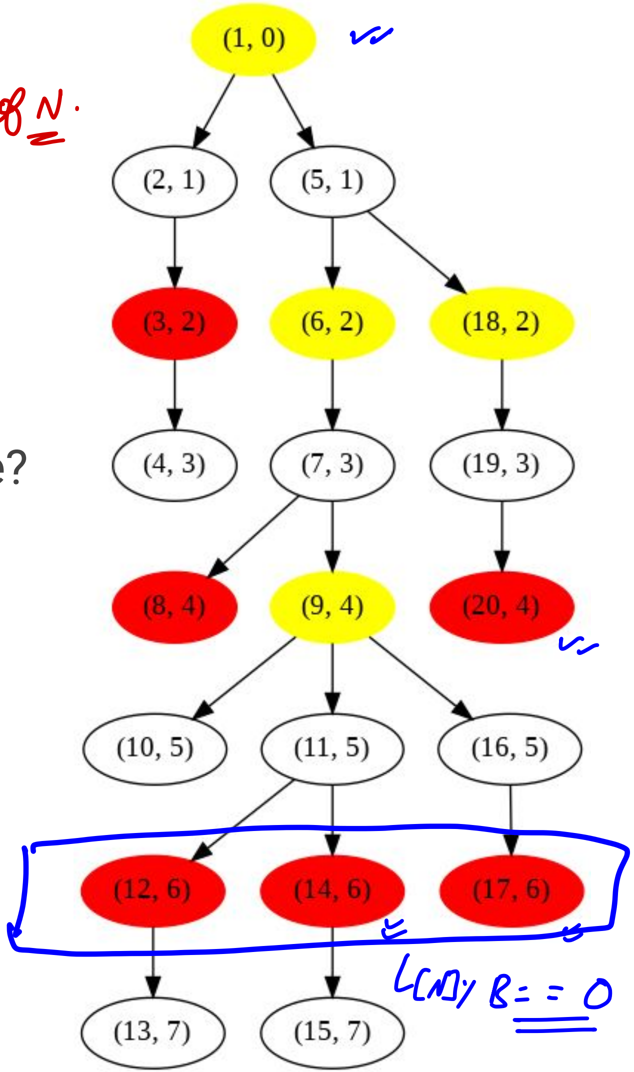
∃ M s.t. L[M] % B == 0 AND M lies in subtree of N.

Supernode Trees Introduction

- What if we mark the nodes special when
  - level[N] % B == 0 and N is an "internal" node i.e.
  - N is not the bottomost marked node in the tree ?
- How many special nodes (yellow) can we now have?
  - A) O(sqrt(N)) ?
  - B) O(N) ?

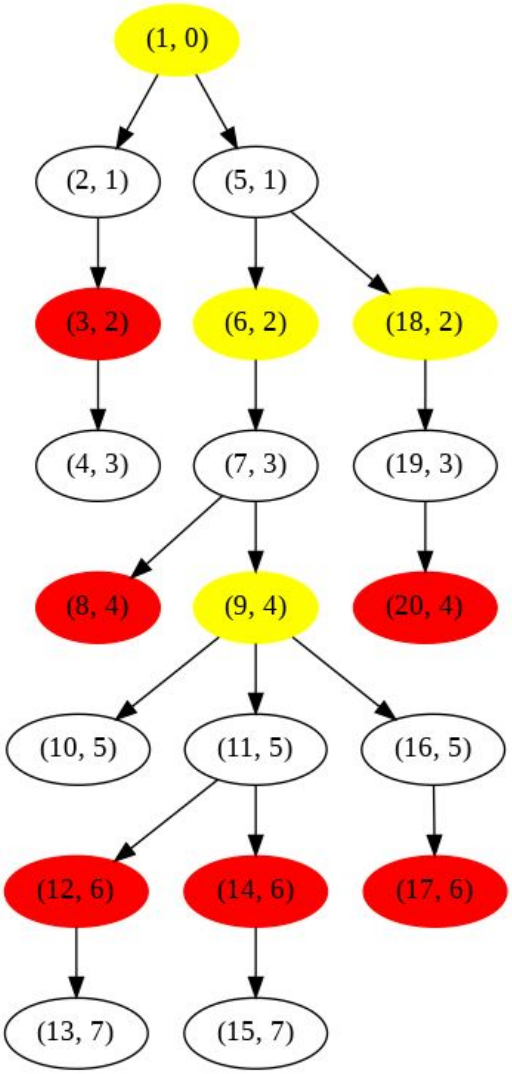
O(N/B)





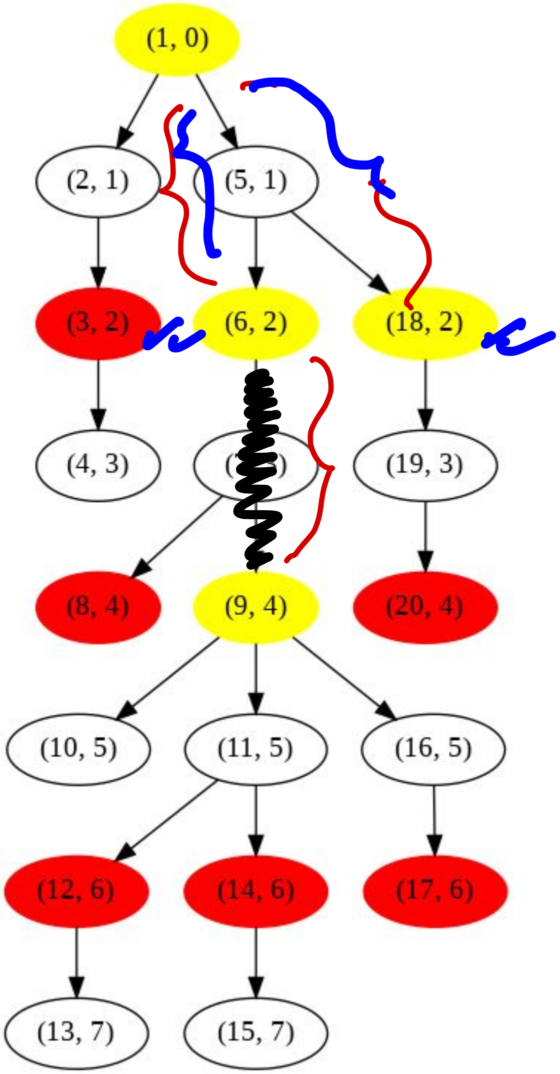
Supernode Trees Introduction

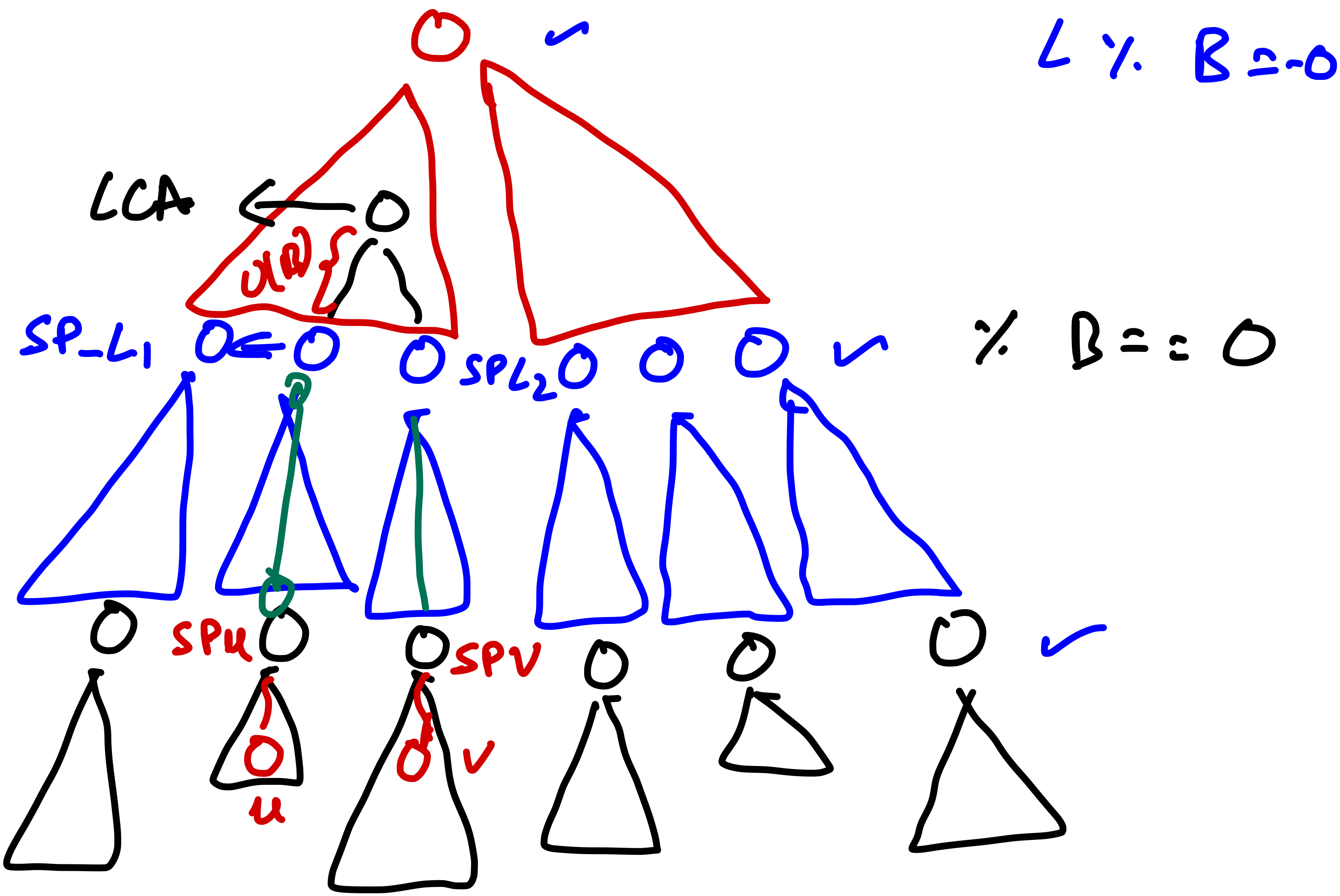
- What if we mark the nodes special when
  - level[N] % B == 0 and N is an “internal” node i.e.
  - N is not the bottomost marked node in the tree ?
- How many special nodes (yellow) can we now have?
  - Note that distance between two special nodes one below each other is O(B) .
  - Hence, each “internal” special node has at least O(B) non-special nodes below it.
  - Number of yellow “special” nodes would therefore be O(N / B).
  - If B ~ sqrt(N); we have ~O(sqrt(N)) marked nodes :)



Supernode Trees Introduction

- What can we do with these (yellow) special marked nodes?
- Can we compress straight chains similar to HLD ?
  - 6 → 9 can be compressed to a single edge and processed together.
  - But, 1 → 6 and 1 → 18 have overlapping node 5.
- Can we do something to have only
  - O(sqrt(N)) marked nodes
  - Compress all chains into single edges between marked nodes, except edges below the bottomost marked node





$u \xrightarrow{\le O(B)} SP\_u$ ✓

$SP\_L_1 \xrightarrow{\le O(B)} LCA$

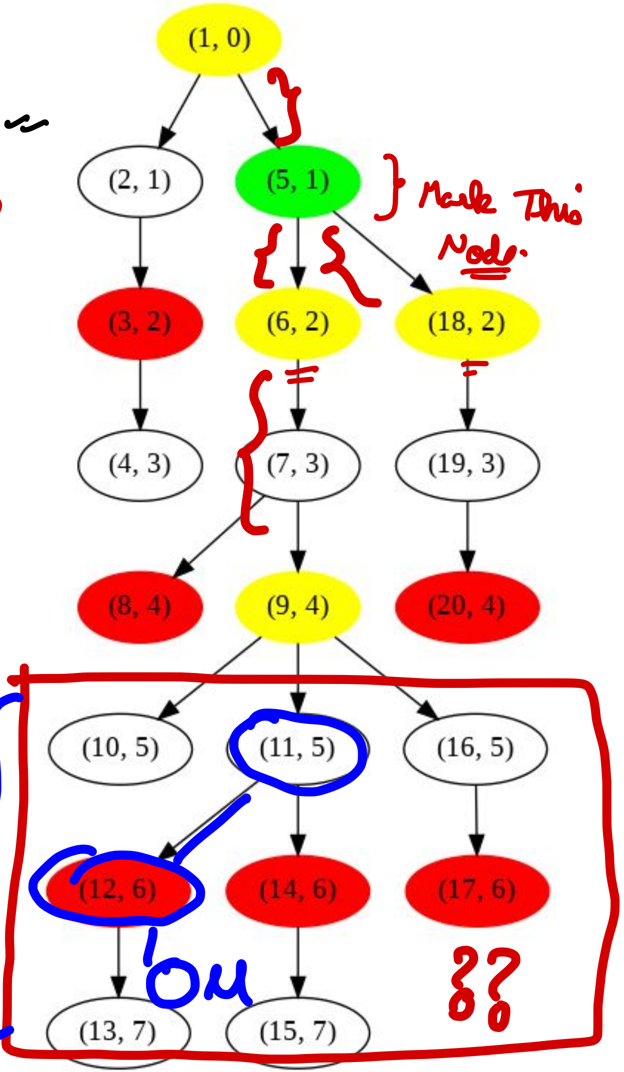
$SP\_u \rightsquigarrow SP\_L_1$

$O(N/B)$ edges

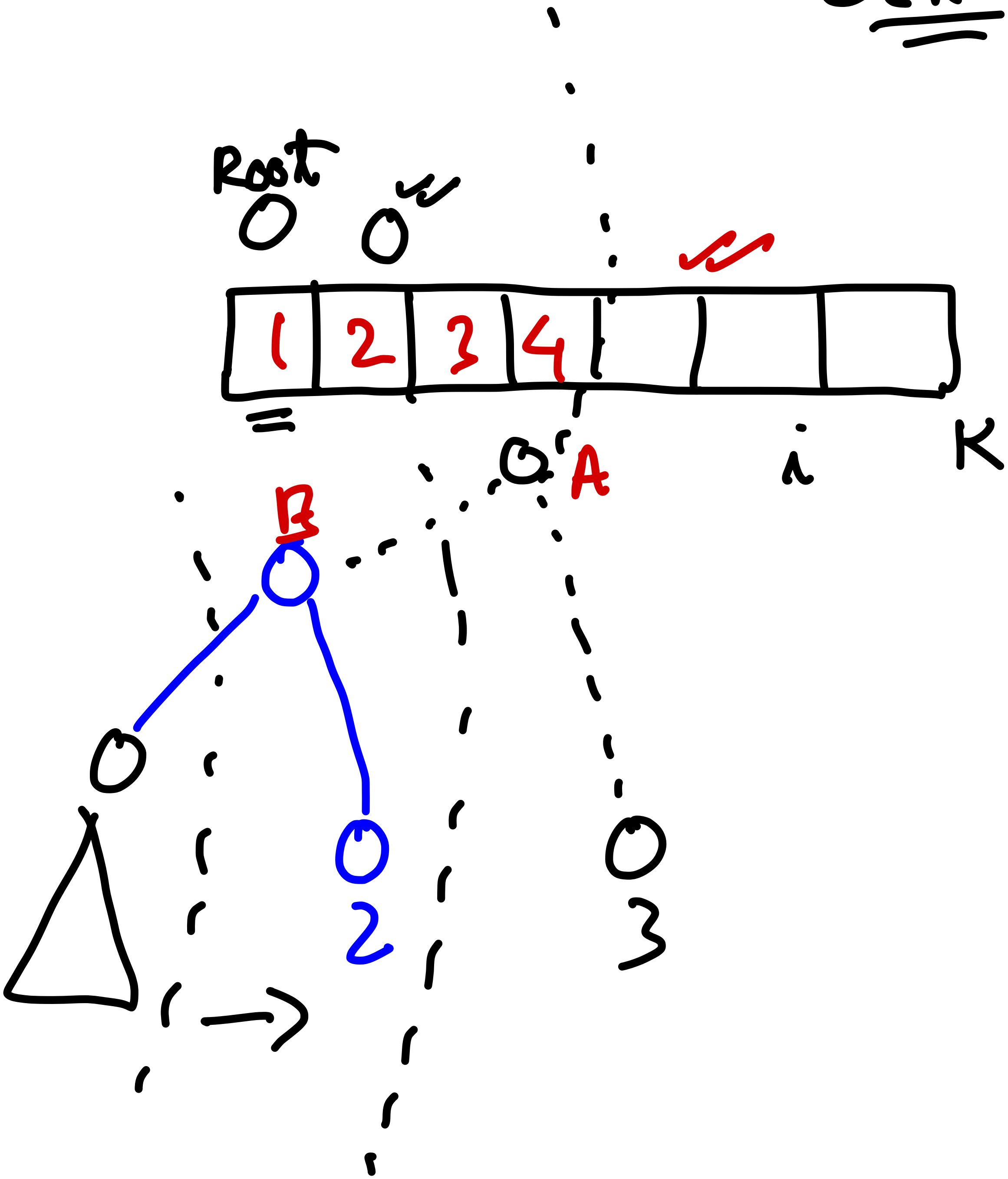
Supernode Trees Introduction

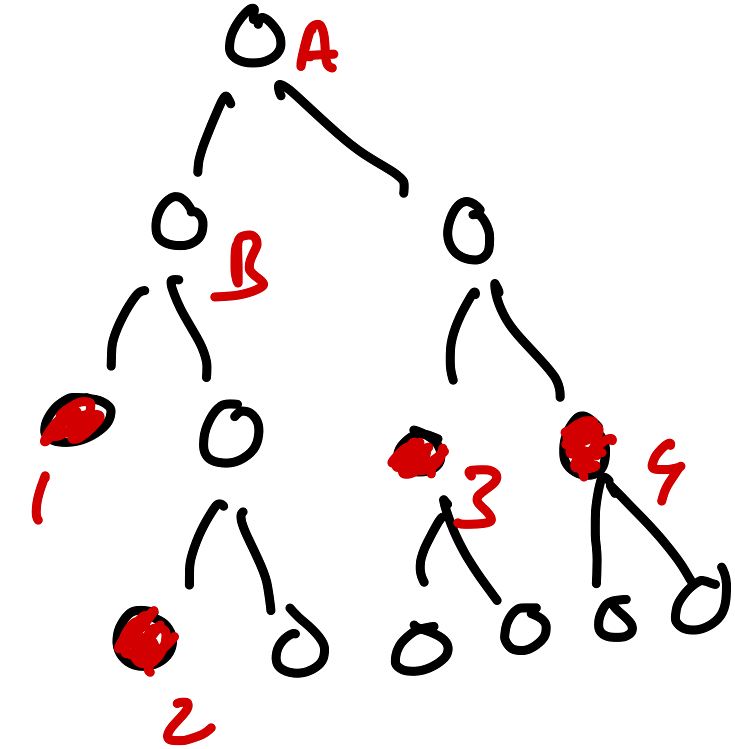
- What can we do with these (yellow) special marked nodes?
- Can we compress straight chains similar to HLD ?
  - 6 → 9 can be compressed to a single edge and processed together.
  - But, 1 → 6 and 1 → 18 have overlapping node 5.
- Can we do something to have only
  - O(sqrt(N)) marked nodes
  - Compress all chains into single edges between marked nodes, except edges below the bottommost marked node.
- Yes! Build an Auxiliary Tree of the K marked nodes
  - The idea is to add LCA of adjacent yellow nodes to the set of marked nodes.
  - It adds at-most K - 1 new nodes, thus total nodes ~O(sqrt(N))

K-1 2B



$O(k-1)$



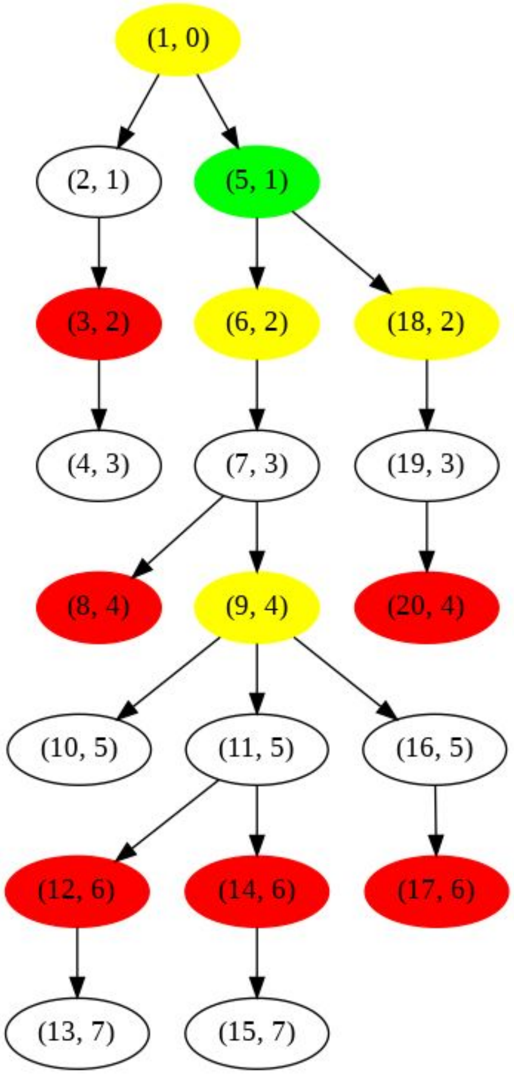


Supernode Trees Properties

- No. of "special" nodes in the supernode tree is O(sqrt(N))
  - Red Nodes: # of nodes O(N), not marked as "special"
    - level[x] % B == 0
  - Yellow Nodes: # of nodes O(N / B), marked as "special"
    - level[x] % B == 0 and
    - Has at least 1 red node in it's subtree.
  - Green Nodes: # of nodes O(N / B), marked as "special"
    - Additional nodes marked to ensure that every top-down path between consecutive special nodes is a chain.
    - Can be found using Auxiliary Tree type processing.
- The "special" nodes are called "supernodes"

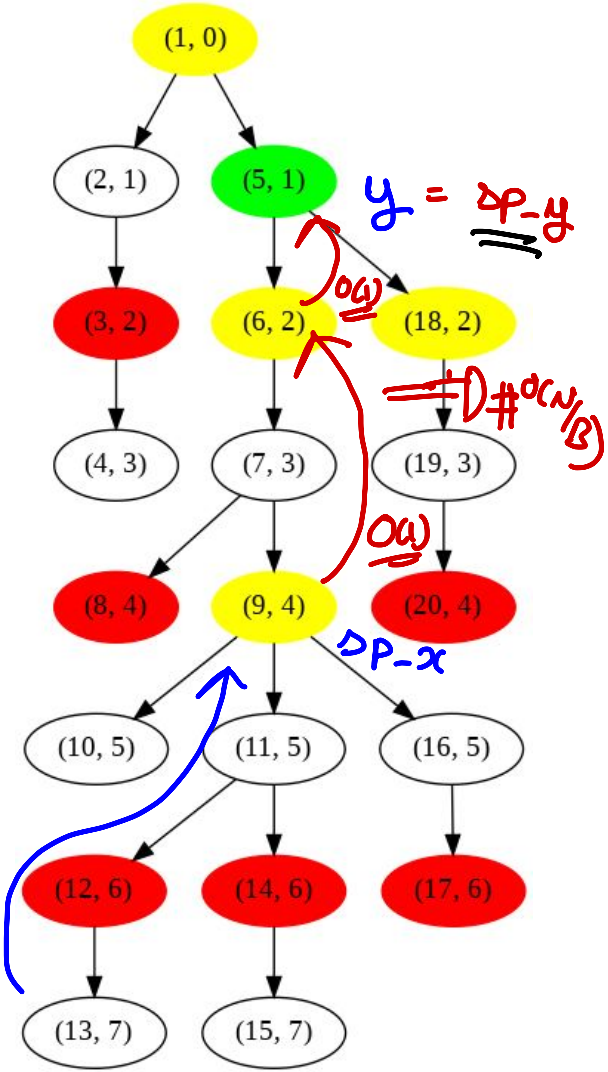
Green / Yellow

O(2N/B)



Supernode Trees Properties

- Any path x → y , where $\boxed{y}$ is an ancestor of $\boxed{x}$ can be written as:
    - x → sp_x: at-most O(B) non-special nodes
    - sp_x → sp_y : at-most O(N / B) special nodes by traversing compressed edges
    - sp_y → y: at-most O(B) non-special nodes.
- This is similar to how a query / update [L, R] gets processed in Array Square Root Decomposition!
    - Note that paths starting below the bottom-most special nodes can have O(2 * B) non-special nodes because of ignored bottom red layer.



$\underbrace{O(B)}_{\text{For Blocks of } x \,\&\, y}$ + $\underbrace{O(N/B)}_{\#\text{ Compressed Path b/w } x \,\&\, y.}$

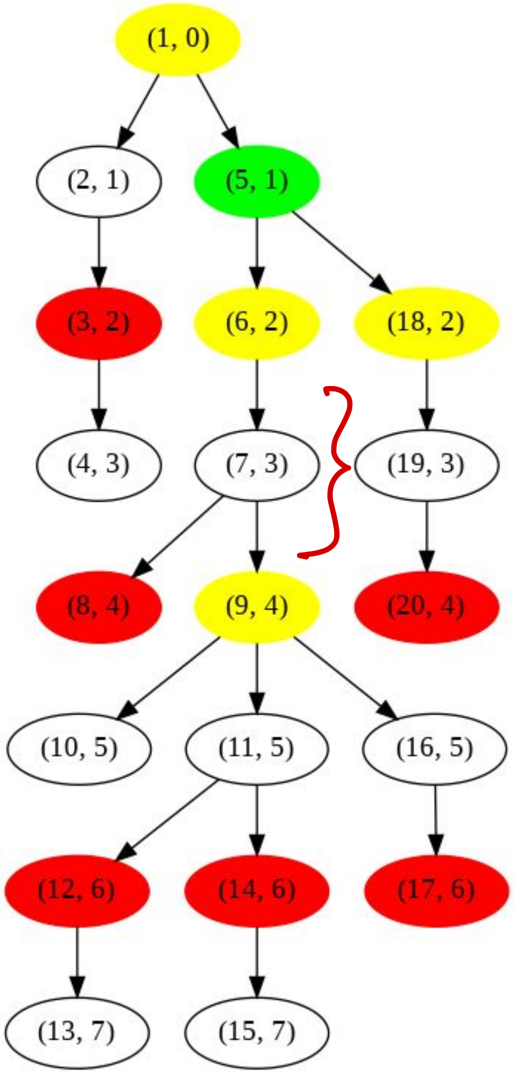
O(2B) Manually Brute Force

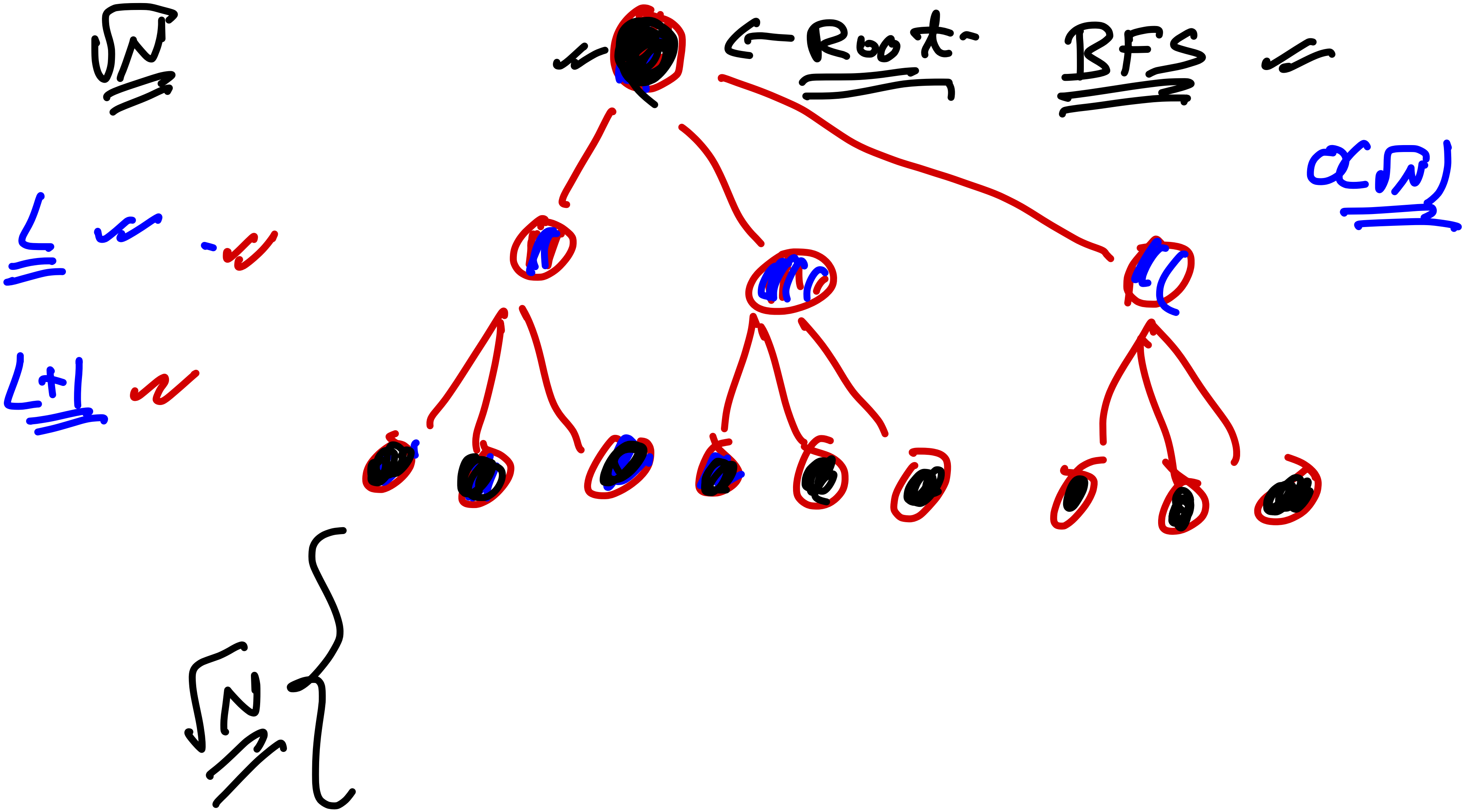
Supernode Trees Properties

- Thus, we can maintain a DS for each compressed chains to answer query / updates efficiently
  - This is similar maintaining a DS for each block in Array Sqrt Decomposition.

$x \to LCA$

- To process a path query / update from $x \to y$ $y \to LCA$.
  - $x \to sp\_x$: Brute force by traversing O(B) non-special nodes
  - $sp\_x \to sp\_y$: Traverse O(N / B) special nodes by jumping over the compressed chains in O(f(B)) using the maintained DS.
  - $sp\_y \to y$: Brute force by traversing O(B) non-special nodes
- Total Complexity: O((B + N / B) * TimeTakenByDS)



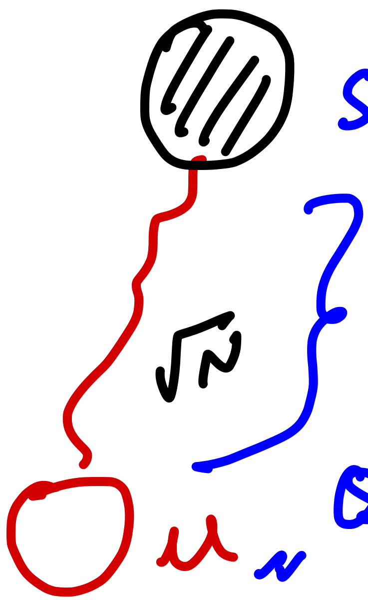


$\sqrt{N}$
Root
BFS
$\alpha\sqrt{N}$
L
L+1
$\sqrt{N}$

C) Compressed Paths are straight lines

+ Auxc. Tree Modification.

A)



sub[x]

Queries

```
def dfs(x):
    sub[x] = 1
    for(y in g[x])
        dfs(y)
        sub[x] += sub[y]
        if sub[x] >= √N:
            spcl[x] = 1
    if spcl[x]:
        sub[x] = 0
```

+ Auxilliary Tree Modif-

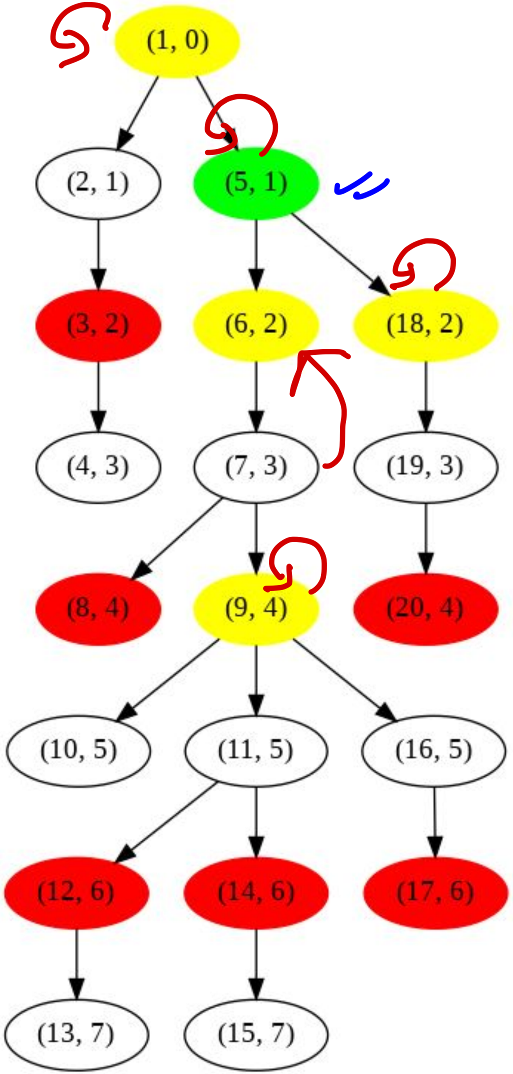
B) # of SPCL Nodes ~ $O(N/B)$??

Yes!!

Supernode Trees Implementation

O(N) DFS

```
int dfs(int x, int p = 0) {
  par[x] = head[x] = p;
  level[x] = level[p] + 1;
  bool seen_spcl = false, is_sqrt_node = !(level[x] % SQRT);
  is_spcl[x] = !p; // root is always special.
  int ret = 0;
  for (auto y : g[x]) {
    if (y == p) continue;
    int w = dfs(y, x);
    if (!w) continue;
    is_spcl[x] |= is_sqrt_node || (seen_spcl && is_spcl[w]);
    seen_spcl |= is_spcl[w];
    ret = w;
  }
  if (is_spcl[x]) head[x] = x;
  return (is_spcl[x] || is_sqrt_node) ? x : ret;
}
```



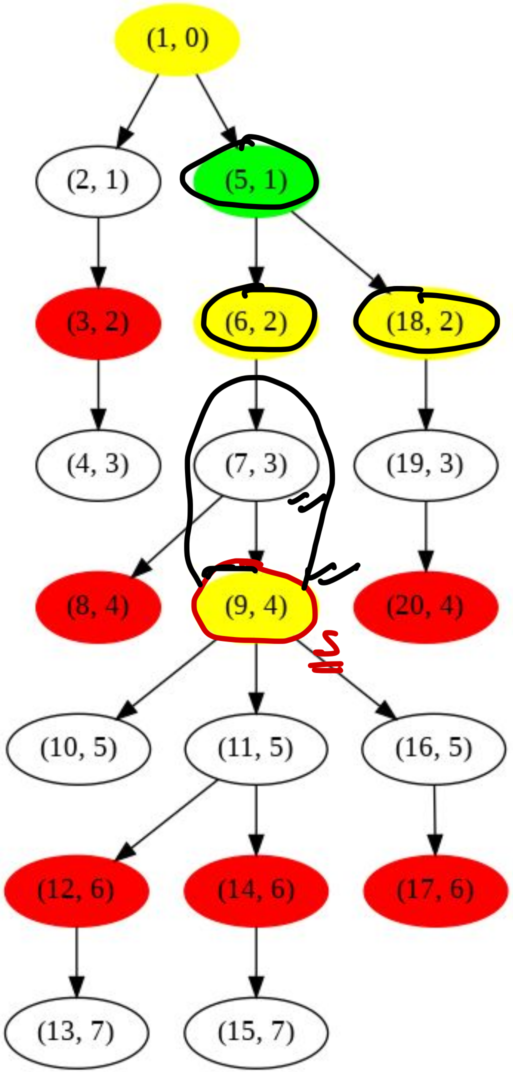
TRS using Supernode Trees

- Given a tree with N elements, support the following operations
  - Query U V X : Tell the minimum element >= X on the path from U to V
  - Update U V V: Add V to all elements on the path from U to V.
- https://www.codechef.com/problems/TRS

TRS using Supernode Trees

- Given a tree with N elements, support
  - Query   U V X : Tell minimum element >= X on the path from U to V
  - Update U V V: Add V to all elements on the path from U to V.
- Solution:
  - Construct the Supernode Tree of the given tree and maintain a multiset of all compressed edge values at each special node.
  - For update_up(U, P):
    - Brute force from U → sp_U in O(B * logB)
    - Lazy Update sp_U → sp_P in O(N / B)
    - Brute force from sp_P → P in O(B * logB)
  - For query_up(U, P)
    - Brute force from U → sp_U in O(B)
    - Lower Bound Query from sp_U → sp_P in O(N / B * logB)
    - Brute force from sp_P → P in O(B)

Query in Multiset

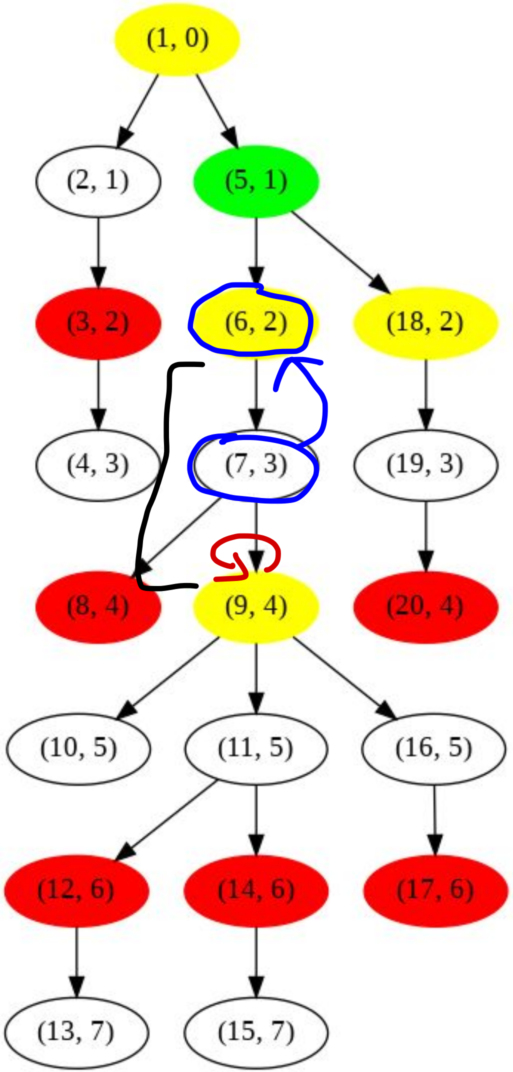


TRS using Supernode Trees

(x, p)

```
void update_up(int x, int p, int64_t add) {
  while (x != p) {
    int b = block[x];
    if (is_spcl[x] && level[head[par[x]]] >= level[p]) {
      block_add[b] += add;
      x = head[par[x]];
    } else {
      if (b) block_vals[b].erase(block_vals[b].find(val[x]));
      val[x] += add;
      if (b) block_vals[b].insert(val[x]);
      x = par[x];
    }
  }
}
```

Special Node above x:



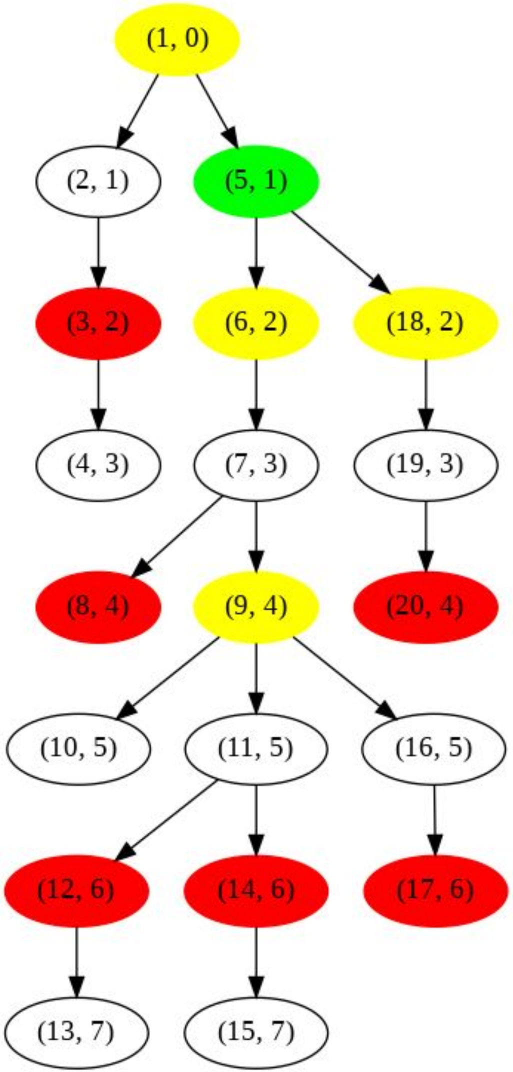
TRS using Supernode Trees

```cpp
int64_t query_up(int x, int p, int64_t min_val) {
  auto ans = INF;
  while (x != p) {
    int b = block[x];
    if (is_spcl[x] && level[head[par[x]]] >= level[p]) {
      auto it = block_vals[b].lower_bound(min_val - block_add[b]);
      if (it != block_vals[b].end()){
        ans = min(ans, *it + block_add[b]);
      }
      x = head[par[x]];
    } else {
      ans = min_op(ans, val[x] + block_add[b], min_val);
      x = par[x];
    }
  }
  return ans;
}
```

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

B.F.

$O(B)$



Implementation

- https://github.com/tanujkhattar/cp-teaching/blob/master/CWC/Ep02%20SRD%20on%20Trees/TRS.cpp

Supernode Tree vs HLD ?

$O(\log N)$ Jumps ; Size of chain ~ $O(N)$

- We do a similar thing in HLD, i.e. divide the tree into disjoint chains and process the chains so that we can "jump" over processed chains while traversing paths from x → y.
- However, one key difference here is that the size of each processed chain <= $\underbrace{\text{sqrt(N)}}$, hence we can store more complex data structures for each chain.
  - Eg: The multiset of all nodes in the chain!
  - This special property gives us the ability to extend Array Square Decomposition Ideas on Path Queries and hence solve some interesting hard problems!
  - Note that size of chains need to be bounded in order to support updates, as we end up iterating on elements partially inside the end chains.

$O(\sqrt{N})$ Jumps.

Further Reading and Practice Problems

- https://arxiv.org/ftp/arxiv/papers/1303/1303.5481.pdf
- https://codeforces.com/blog/entry/46843
- https://codeforces.com/blog/entry/68231 , Problem - F editorial
- https://atcoder.jp/contests/abc133/tasks/abc133_f