Square Root Decomposition

Jeremy Chow
1-7-2019

Agenda

- Decomposition on array
- Decomposition on queries
- Mo’s algorithm on array
- Mo’s algorithm on tree

Why is sqrt special?

- You will always see $O(x + \frac{N}{x})$ when we deal with sqrt decomposition problem
- $x = N^{1/3}$
- $O(N^{1/3}+N^{2/3}) = O(N^{2/3})$

- $x = \sqrt{N}$ is the minimum point of $O(x + \frac{N}{x})$
- $O(\sqrt{N}+ \sqrt{N}) = O(\sqrt{N})$

Problem A (Decomposition on array)

- Given an array A with length N and Q queries
- Each query can be either
  - Find the sum of value of array A in [l, r]
  - Change a[x] to v

- Range sum problem

Problem A (Decomposition on array)

- Standard problem for segment tree
- Can be solved with segment tree in [O(log(n)), O(log(n))]
- Data Structure (III) https://assets.hkoi.org/training2019/ds-iii.pdf

- But let's solve with sqrt decomposition in [$O(\sqrt{N})$, O(1)]
- More intuitively

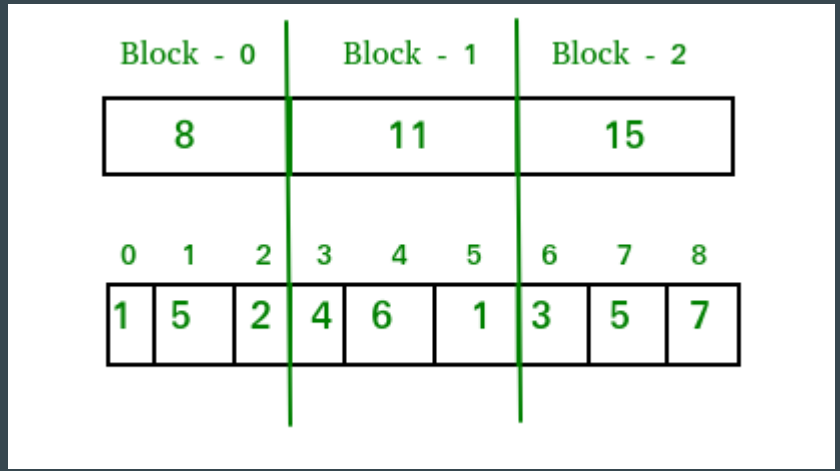
Problem A (Decomposition on array)

- Naive Solution
- For query, just go through [l, r] with for loop and get the sum
- For update, just set a[x] to v
- [O(N), O(1)]

- We can solve it with sqrt decomposition in a similar way

Problem A (Decomposition on array)

- A[ ] = {1, 5, 2 4, 6, 1, 3, 5, 7}
- Let split the array into $\sqrt{N}$ blocks
  - siz = $\sqrt{N}$
  - Each block consists of $\sqrt{N}$ elements
- Calculate the sum of each block

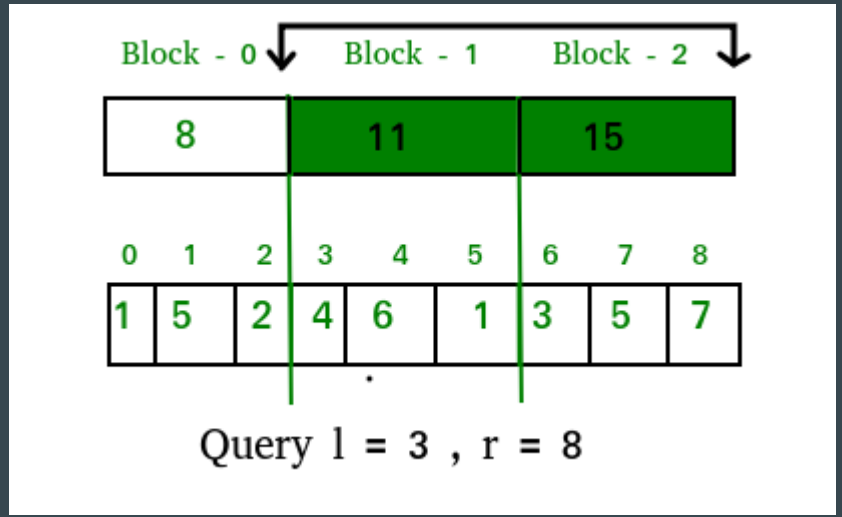


Picture from GeeksforGeeks

Problem A (Decomposition on array)

- Special Case

- l, r lay on block's boundaries
- Go through blocks between [l, r]
- Add up the sum of the blocks
- Time complexity = $O(\sqrt{N})$

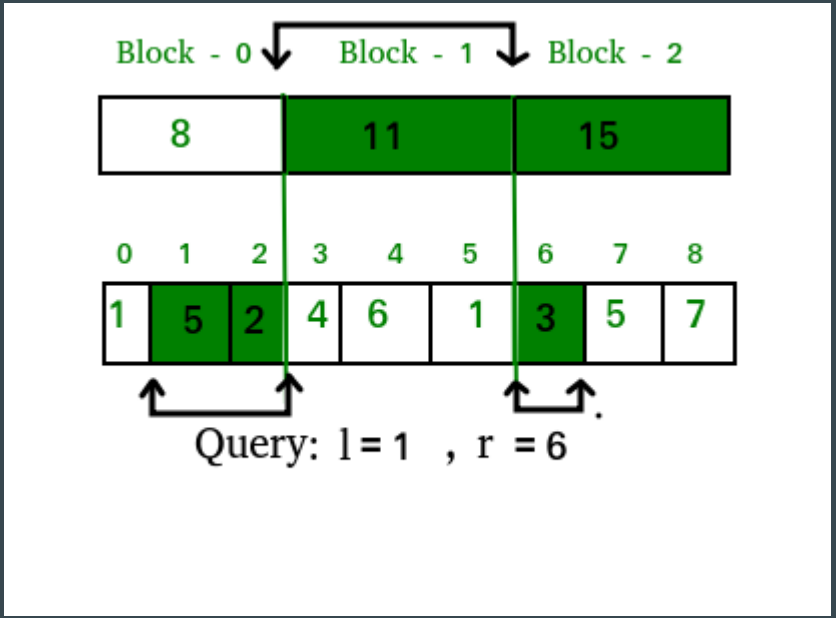


Picture from GeeksforGeeks

Problem A (Decomposition on array)

- General Case

- l, r do not lay on block's boundaries
- Split into Head, Body, Tail
- Body is sum of blocks
- For loop to go through Head and Tail
- $O(\sqrt{N})$



Picture from GeeksforGeeks

Problem A (Decomposition on array)

bl = l / siz , br = r / siz

if (bl == br)

add up array value in [l, r]

else

add up array value in [l, (bl + 1) * siz - 1]

add up block sum from $(bl + 1)^{th}$ to $(br - 1)^{th}$ block

add up array value in [br * siz, r]

Problem A (Decomposition on array)

- For update, update the block sum and a[x]
- sum -= a[x]
- a[x] = v;
- sum += a[x];
- O(1)

- $[O(\sqrt{N}), O(1)]$

Problem B (Decomposition on array)

- Too easy?
- CF551E GukiZ and GukiZiana
  - https://codeforces.com/problemset/problem/551/E

- Given an array A with length N and Q queries
- Each query can be either
  - Add x to [l, r]
  - Find the maximum value of j-i s.t. a[i] = a[j] = y

Problem B (Decomposition on array)

Sample 1

[1, 2, 3, 4] -> [3, 3, 3, 4]

1 1 2 1

1 1 1 1

2 3

Answer : 2

Problem B (Decomposition on array)

- Naive Solution
- For loop

- Lets apply the same trick again
- Spilt the array into $\sqrt{N}$ block
    - Each with $\sqrt{N}$ elements

Problem B (Decomposition on array)

{1, 1, 1, 2, 1, 3, 1, 1}

- Previously we store sum within a block
  - What information we need?

1, 1, 1	2, 1, 3	1, 1

Problem B (Decomposition on array)

- We want to find the minimum i and maximum j
  - s.t. a[i] = a[j] = y
  - Maximize j - i

- How to do better than linear search?
- Binary search

Problem B (Decomposition on array)

- Let's store an sorted array for each block
  - Each element is (a[i], i), sorted by a[i] then i ascendingly

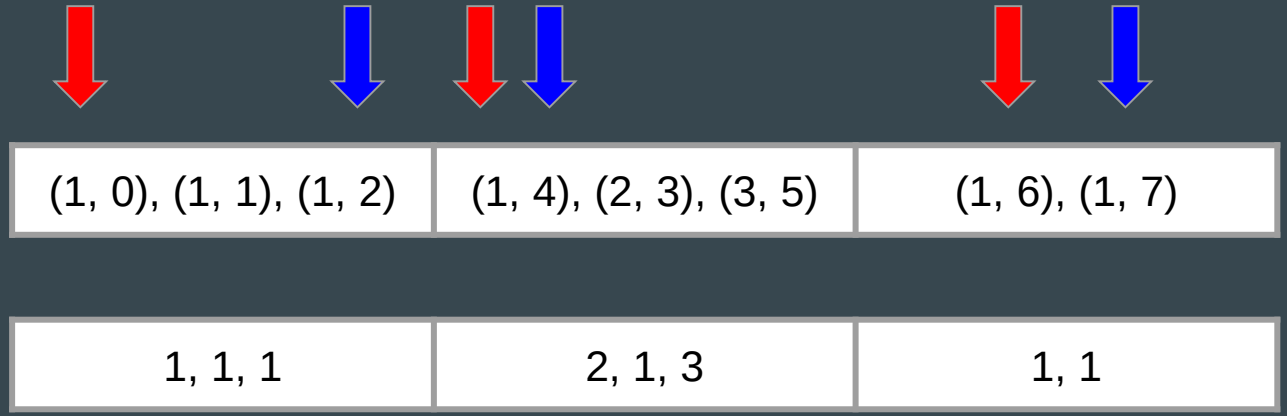
(1, 0), (1, 1), (1, 2)	(1, 4), (2, 3), (3, 5)	(1, 6), (1, 7)

1, 1, 1	2, 1, 3	1, 1

- $O(\sqrt{N} * (\sqrt{N} * \log(\sqrt{N}))) \rightarrow O(N\log(\sqrt{N}))$

Problem B (Decomposition on array)

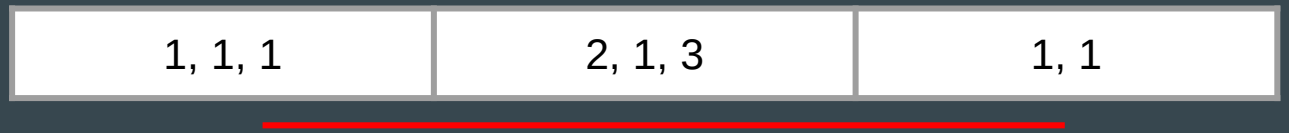
- Query 1

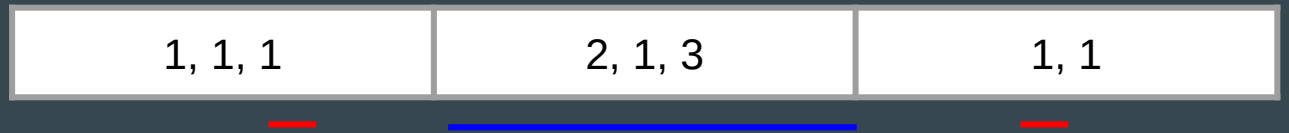


- Answer = 7 - 0 = 7
- $O(\sqrt{N}\log(\sqrt{N}))$

Problem B (Decomposition on array)

- How to maintain after update?

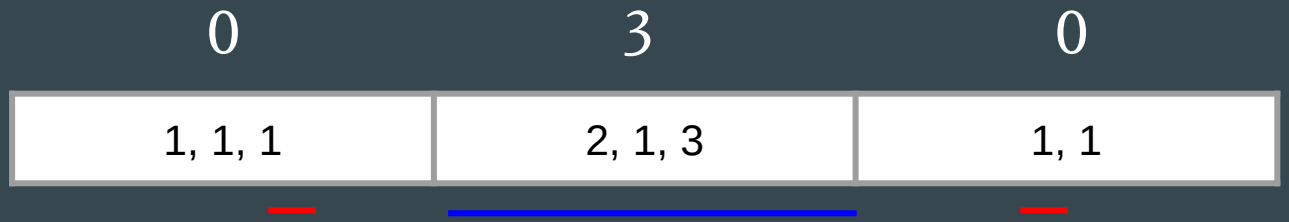




- Split into Head, Body, Tail

Problem B (Decomposition on array)

- Update for Body (whole blocks)
  - Add v to the lazy tag
  - Query(y - lazy)



Problem B (Decomposition on array)

- Update for Head and Tail
  - change a[i] += v
  - Reconstruct the sorted array located in Head's and Tail's block

(1, 0), (1, 1), (4, 2)	(1, 4), (2, 3), (3, 5)	(4, 6), (1, 7)

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

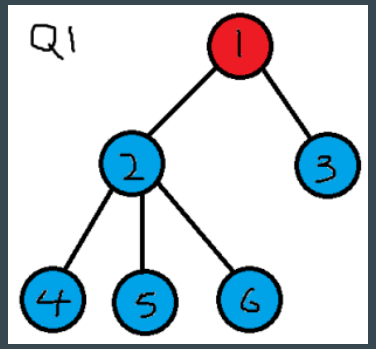
- $O(\sqrt{N}\log(\sqrt{N}))$

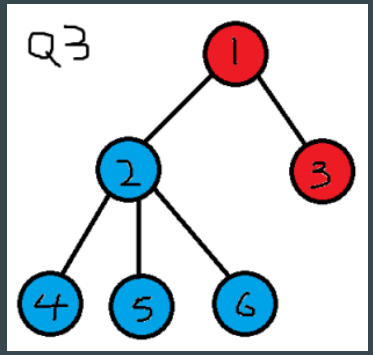
Problem C (Decomposition on queries)

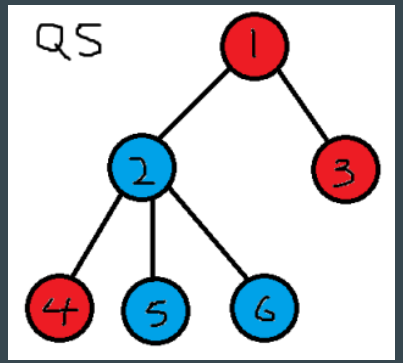
- CF342E Xenia and Tree
  - https://codeforces.com/problemset/problem/342/E
- You are given a tree with N nodes labelled from 1 to N
- Node 1 is initially red while others are blue
- You need to process M queries
- Each query will be one of the following types
- Type 1: Given u, paint u as red
- Type 2: Given v, find the distance of v from the nearest red node
- N, M <= 10^5

Problem C (Decomposition on queries)

- Query 1: ask 5
- Query 2: paint 3
- Query 3: ask 5
- Query 4: paint 4
- Query 5: ask 1







Problem C (Decomposition on queries)

- Naive Solution
- dist[i] = distance to closest red node
- For query, print dist[v], ~~do dfs~~
- For update, do multi-source bfs to calculate dist[i], ~~paint a[u]~~
- O(MN)

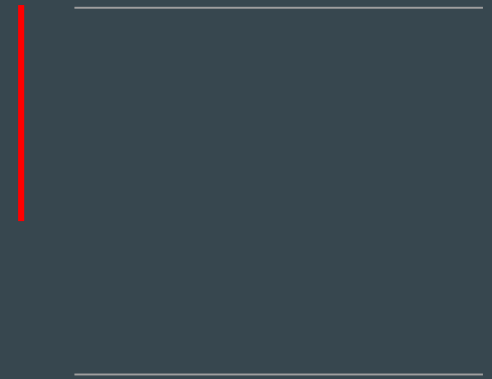
- Split updates into $\sqrt{M}$ block
  - Each Block consists of $\sqrt{M}$ updates

Problem C (Decomposition on queries)

- Split updates into $\sqrt{M}$ block
  - Each Block consist $\sqrt{M}$ updates
- Do muti-source bfs after processed a block of updates
  - $\sqrt{M}$ blocks
  - O(N$\sqrt{M}$)

Problem C (Decomposition on queries)

- Query in the middle of block?
- Iterate all the updates in the current block (before i)
  - res = min(dist[v], distance(v, u_i))
  - distance(v, u_i) can be calculated by using LCA
  - Graph (III) https://hkoi.org/en/training-materials/2019/



- Dist[v] = minimum answer for previous block
- Iterate all update in the current block = minimum answer in curent block
- Minimum answer = minimum of both

Problem C (Decomposition on queries)

- BFS per block
- $O(N\sqrt{M})$

- Iterate per update
- $O(M\sqrt{M})$

- $O(N\sqrt{M} + M\sqrt{M})$

Problem D (Mo's algorithm)

- CF 86D Powerful array
  - https://codeforces.com/problemset/problem/86/D

- Given an array consists of N integers and M queries
- Output the power of the subarray [l_i, r_i] for each query
- Power of the subarray
  - sum(cnt[s] * cnt[s] * s) for every unique integer s in [l_i, r_i]

Problem D (Mo’s algorithm)

- [1, 1, 2, 2, 1, 3, 1, 1]
- Query 2 7

- cnt[1] * cnt[1] * 1 + cnt[2] * cnt[2] * 2 + cnt[3] * cnt[3] * 3
- 9 + 8 + 3 = 20

Problem D (Mo’s algorithm)

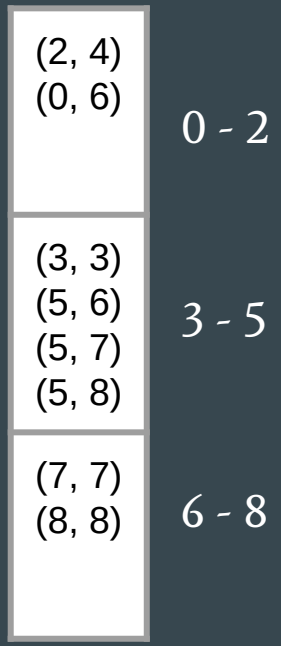
- Naive solution
- For each query, loop over [l_i, r_i] and calculate the answer

- Offline query
- Sorting to change the order of the query
- Make use of previous information

Problem D (Mo's algorithm)

- Sort all the queries by $\frac{l\_i}{\sqrt{N}}$, then by r_i
  - Queries with l_i in $[k\sqrt{N}, (k+1)\sqrt{N} - 1]$ is in the same block
  - $[0, \sqrt{N} - 1], [\sqrt{N}, 2\sqrt{N} - 1], \ldots$

- Process the queries one by one
  - Move the left and right pointer
  - Maintain the answer in a subarray



Problem D (Mo's algorithm)

- Assume you have the answer for [l, r]
- Able to compute answer to [l + 1, r], [l - 1, r], [l, r + 1], [l, r - 1] quickly
    - O(1) or O(logN)

- Maintain the answer between queries
- Avoid some calculations

Problem D (Mo's algorithm)

- Calculate the change of answer when adding or removing a number in index i

```
add(int x) {
        res -= cnt[a[x]] * cnt[a[x]] * a[x];
        cnt[a[x]]++;
        res += cnt[a[x]] * cnt[a[x]] * a[x]
}
```

```
remove(int x) {
        res -= cnt[a[x]] * cnt[a[x]] * a[x];
        cnt[a[x]]--;
        res += cnt[a[x]] * cnt[a[x]] * a[x]
}
```

- O(1)

Problem D (Mo's algorithm)

curL = 0; curR = -1;

For all query {

```
while (curL < L) remove(l++);

while (curL > L) add(--l);

while (curR < R) add(++r);

while (curR > R) remove(r--);

Store the answer
```

}

Problem D (Mo’s algorithm)

- Why is this faster than naive solution?
- Time complexity?

- How much curR is moved?
- How much curL is moved?

Problem D (Mo's algorithm)

- How much curR is moved?

- For each block of queries, curR moves atmost O(N) times
  - r_i is sorted ascendingly
- There are $\sqrt{N}$ block of queries
- Total movement of curR = $O(N\sqrt{N})$

Problem D (Mo's algorithm)

- How much curL is moved?
- Two case when we transit from one query to another

1. Within the same block, curL moves atmost $O(\sqrt{N})$ times
    a. O(M)
2. In the different block, curL moves atmost $O(\sqrt{N})$ block
    a. Each block has length of $\sqrt{N}$

- CurL moves atmost $O(M\sqrt{N}+ N)$

Problem D (Mo's algorithm)

- Time complexity = $O(N\sqrt{N} + M\sqrt{N})$

- Useful when the problem is offline
- Able to compute answer to [l + 1, r], [l - 1, r], [l, r + 1], [l, r - 1] quickly
- Often seen in CF

Mo’s algorithm with modification

- Actually we can still apply Mo’s algorithm when there is update
- Time of queries matters

- Normal mo’s algorithm has two dimension
  - L and R
- Mo’s algorithm with modification has three dimension
  - L, R and T
- Add one more dimension - Time

Mo's algorithm with modification

- Now a block is a 2D rectangle
- Let the size of block be S_1 and S_2
- It can be proved that $S_1 = S_2 = N^{2/3}$ is optimal

- Sort all the queries by $l_i / N^{2/3}$, then by $r_i / N^{2/3}$, then by t_i
- Able to compute answer to $[l \pm 1, r, t]$, $[l, r \pm 1, t]$, $[l, r, t \pm 1]$
    - Go through the update

Mo’s algorithm with modification

curL = 0; curR = -1; curT = -1;

For all query

```
while (curL < L) remove(curL++);
while (curR < R) add(++curR);
while (curT < T) add(++curT);
Store the answer
```

```
while (curL > L) add(--curL);
while (curR > R) remove(curR--);
while (curT > T) remove(curT--);
```

Mo’s algorithm with modification

- If N = M
- Time complexity = $O(N^{5/3})$

- Not so often seen
- Practise problem : bzoj 2120 數顏色
  - https://www.lydsy.com/JudgeOnline/problem.php?id=2120

Problem E (Mo's algorithm on tree)

- SPOJ Count on a Tree II
  - https://www.spoj.com/problems/COT2/

- Given a tree with N nodes, each nodes has an integer weight w_i
- Output the number of distinct weight on the path from u to v

Problem E (Mo's algorithm on tree)

- Let's think of a easier version first

- Given an array with N integers w_i
- Output the number of distinct w_i from l to v in the array

Problem E (Mo's algorithm on tree)

- Given an array with N integers w_i
- Output the number of distinct w_i from l to v in the array

- Can offline
- Able to compute answer to [l + 1, r], [l - 1, r], [l, r + 1], [l, r - 1] quickly
- Normal Mo's algorithm

Problem E (Mo's algorithm on tree)

```
add(int x) {
        if (cnt[w[x]] == 0) res++;
        cnt[w[x]]++;
}
```

```
remove(int x) {
        if (cnt[w[x]] == 1) res--;
        cnt[w[x]]--;
}
```

- Time complexity = $O(N\sqrt{N} + M\sqrt{N})$

Problem E (Mo's algorithm on tree)

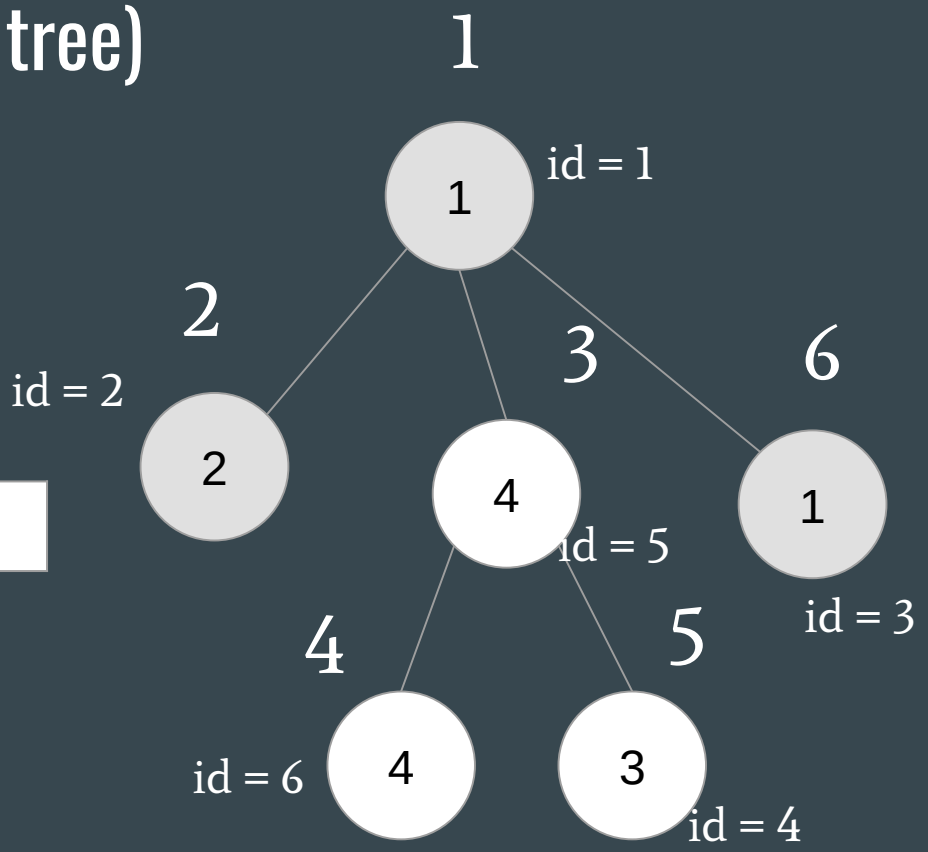
- A harder verison
- Given a rooted tree with N nodes, each nodes has an integer weight w_i
- Output the number of distinct weight in the subtree of u

- If you are familiar with subtree sum problem
- You will know a technique called "Flattening a tree"

Problem E (Mo's algorithm on tree)

- Flatten the tree by preorder

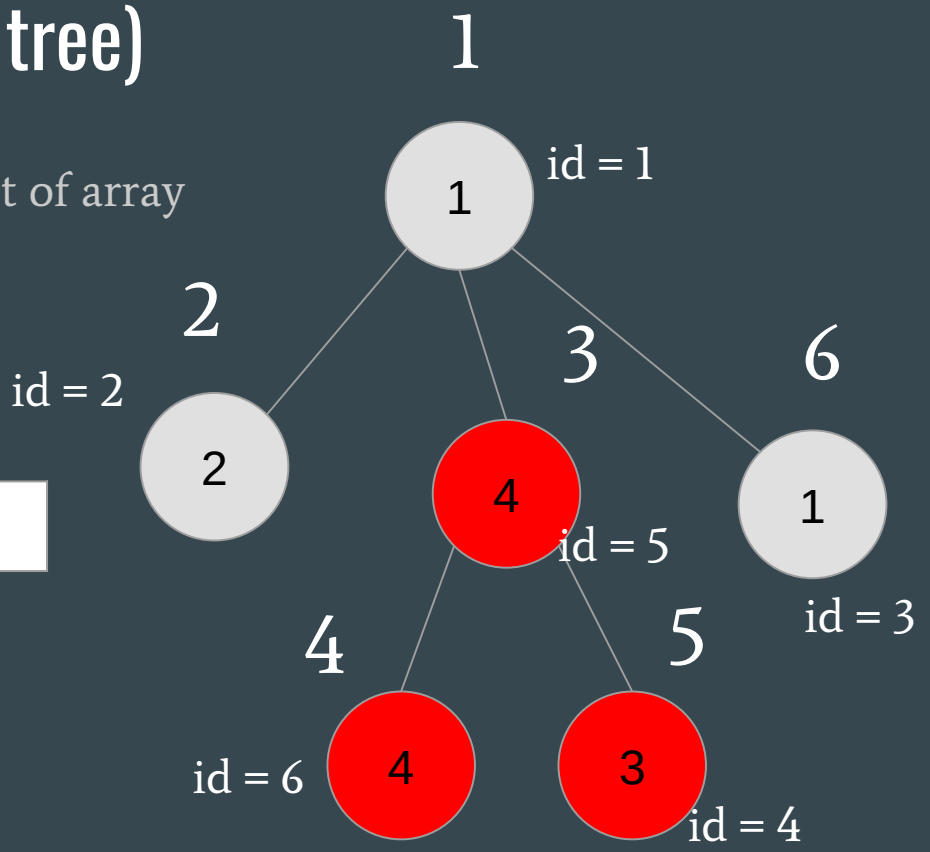
1	2	4	4	3	1



Problem E (Mo's algorithm on tree)

- Subtree become a consecutive segment of array
- Query 5's subtree
- = Query [3, 5]

1	2	4	4	3	1



Problem E (Mo’s algorithm on tree)

- Transform query of subtree to query of array
- Same with previous easier version
- Apply Mo’s algorithm on array

Problem E (Mo's algorithm on tree)

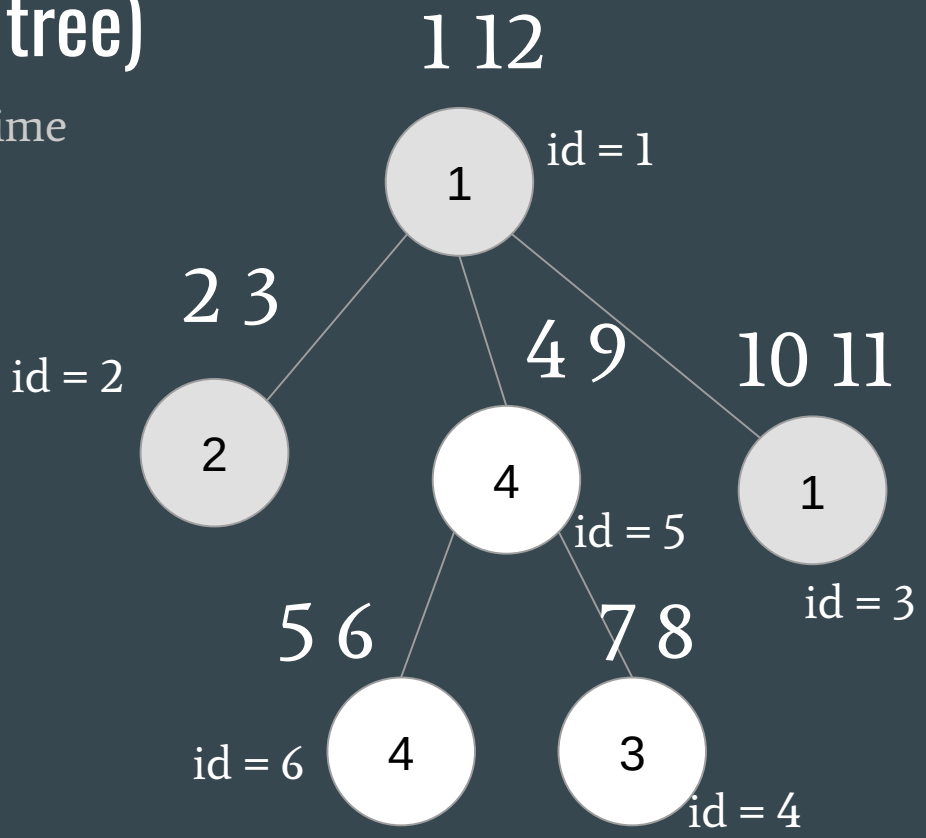
- Come back to the original problem
- Given a tree with N nodes, each nodes has an integer weight w_i
- Output the number of distinct weight on the path from u to v

- We can also transform it into query of array
- But how?

Problem E (Mo's algorithm on tree)

- Flatten the tree by starting and ending time

```
void dfs(int id) {
        ST[id] = ++t;
        for each child u
                dfs(u)
        EN[id] = ++t;
}
```

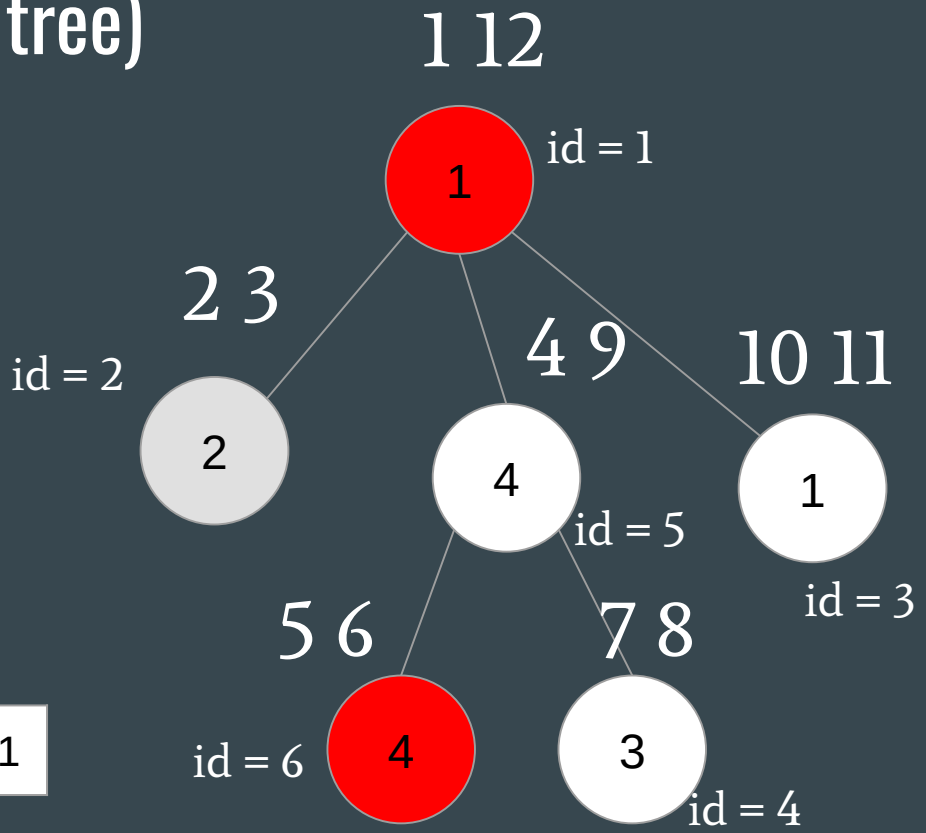


Problem E (Mo's algorithm on tree)

- Query u v
  - ST[u] < ST[v]
- E.g. Query 1 6
  - Let p = lca(u, v) = = 1
- lca = u = 1
- Consider [ST[1], ST[6]]
  - [1, 5]

1	2	2	4	4	4	3	3	4	1	1	1

idx: 1 2 2 5 6 6 4 4 5 3 3 1

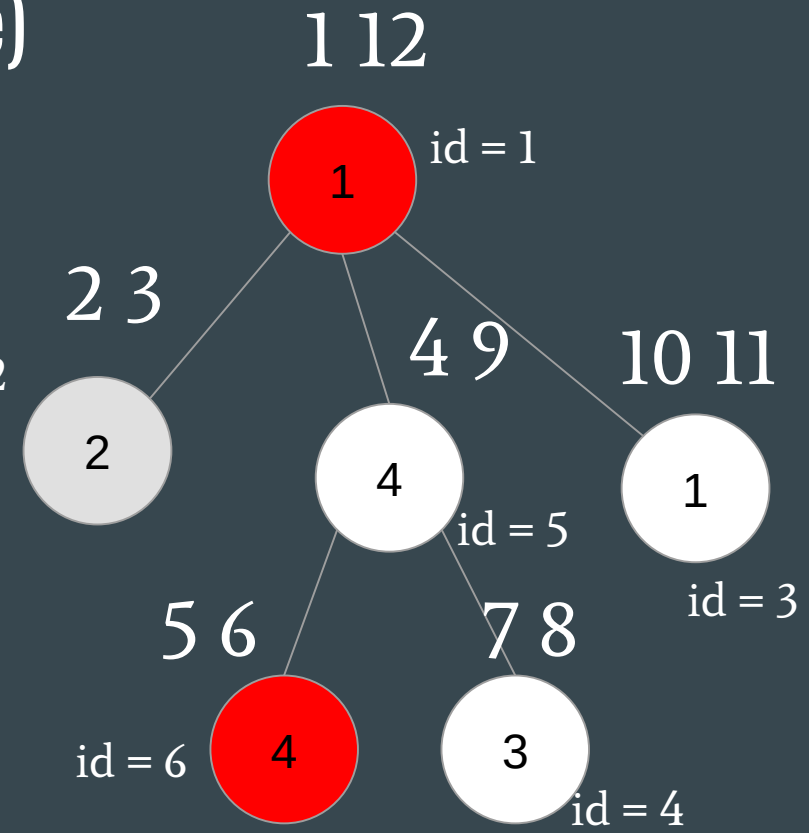


Problem E (Mo's algorithm on tree)

- Consider [ST[1], ST[6]]
  - [1, 5]
- Index of node from u to v only appear once
- 1 5 6
- If an index appear twice (e.g. idx = 2)
- Ignore it

1	2	2	4	4	4	3	3	4	1	1	1

idx: 1 2 2 5 6 6 4 4 5 3 3 1

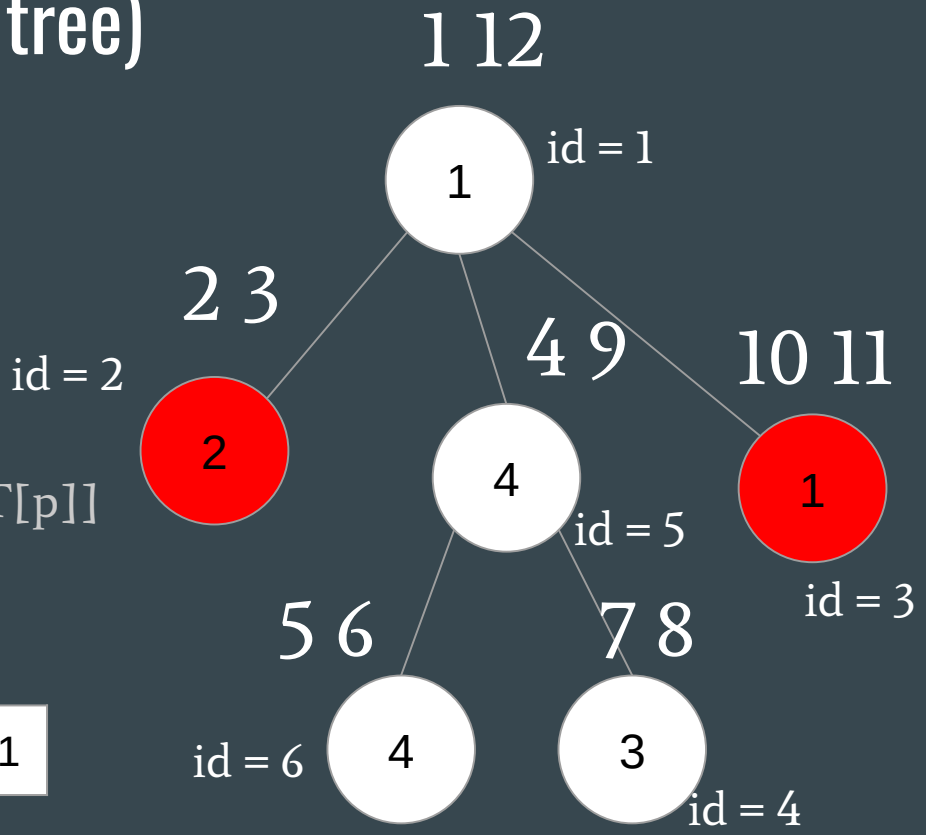


Problem E (Mo's algorithm on tree)

- What if lca != u?
- E.g. Query 2 3
  - p = 1
- p != u
- Consider [EN[2], ST[3]] + [ST[p], ST[p]]
  - [3, 10] + [1, 1]
  - index appear once -> 2, 3, 1

1	2	2	4	4	4	3	3	4	1	1	1

idx: 1 2 2 5 6 6 4 4 5 3 3 1



Problem E (Mo’s algorithm on tree)

- Ignore elements which index appear twice in the given range

- Transform query of path to query of array
- Case 1 : p = u -> [ST[u], ST[v]]
- Case 2 : p != u -> [EN[u], ST[v]] + [ST[p], ST[p]]
    - Don’t need to push [ST[p], ST[p]]

Problem E (Mo’s algorithm on tree)

For all query

```
while (curL < L) solve(l++);        while (curL > L) solve (--l);
while (curR < R) solve(++r);        while (curR > R) solve(r--);
if type 2 query
        add(ST[p]) ;
Store the answer
if type 2 query
        remove(ST[p]);
```

Problem E (Mo’s algorithm on tree)

```
solve(int x)
        if (vis[idx[x]] == 0)
                if (cnt[w[x]] == 0) res++;
                cnt[w[x]]++;
        else
                if (cnt[w[x]] == 1) res--;
                cnt[w[x]]--;
        vis[idx[x]] ^= 1
```

Problem E (Mo's algorithm on tree)

- Use sparse table to precompute
- Find lca in O(1)
- Time complexity = $O(Q\sqrt{N} + N\sqrt{N})$