More about Query Problem

HKOI Training Camp 2018-7-1

Agenda

- Query Problem on Tree
- Centroid Decomposition (Divide & Conquer on Tree)
- CDQ Divide & Conquer for offline query/update
- CDQ Divide & Conquer for DP problem

Query Problem on Tree

Common Form for Tree Query Problem

- Update + Query Type
  - Rooted tree + update a node value + query sum of node value for a subtree
  - Rooted tree + update a node value + query sum of node value for a path
- Find Total Number Type
  - Un-rooted tree + find total number of path satisfying path length == a Constant K

Common Form for Tree Query Problem

- Update + Query Type + Query on subtree
  - Usually able to solve by Segment Tree built by Pre-order of tree
- Update + Query Type + Query on path
  - Usually able to solve by LCA + Heavy light decomposition
- Count Total Number + Count on Path
  - Usually able to solve by Centroid Decomposition
- Count Total Number + Count on subTree → Usually it is Easy ☺

Problem A

IOI 2011 Race

Basic D&C on Tree

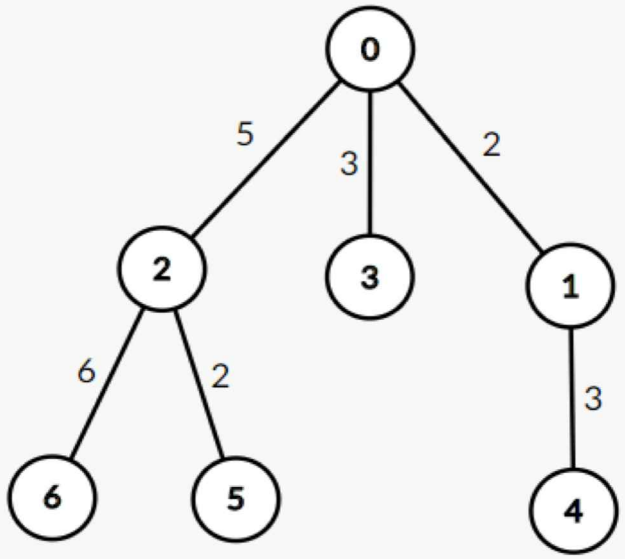
IOI 2011 Race

- Given a weighted unrooted tree
- Find number of pair(x, y)
  - satisfying distance between node x and node y = K where K is a constant
- Very typical Count Total Number + Count on Path

IOI 2011 Race

- Assume K = 8
- The answer = 3
- {(2, 3), (3, 4), (5, 6)}

- An O(N^2) solution can be achieved easily by N DFS



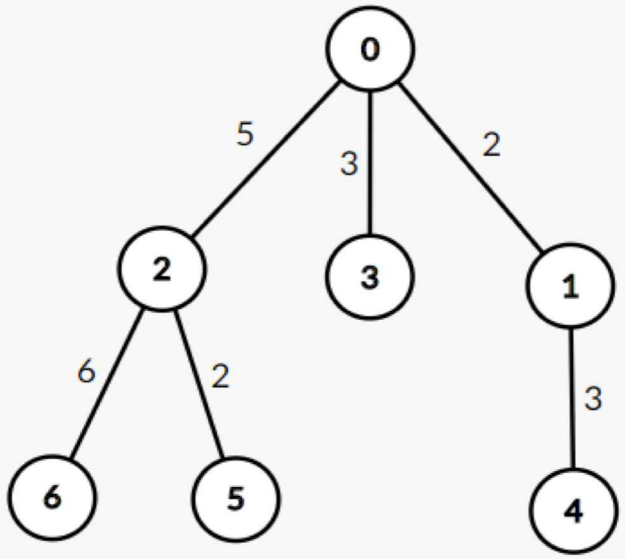
IOI 2011 Race

- To achieve a better solution, we can……

- Consider an easier version first
- find number of pair(x, y)
  - satisfying distance between node x and node y = K where K is a constant
  - and the path between x and y must pass through node 0

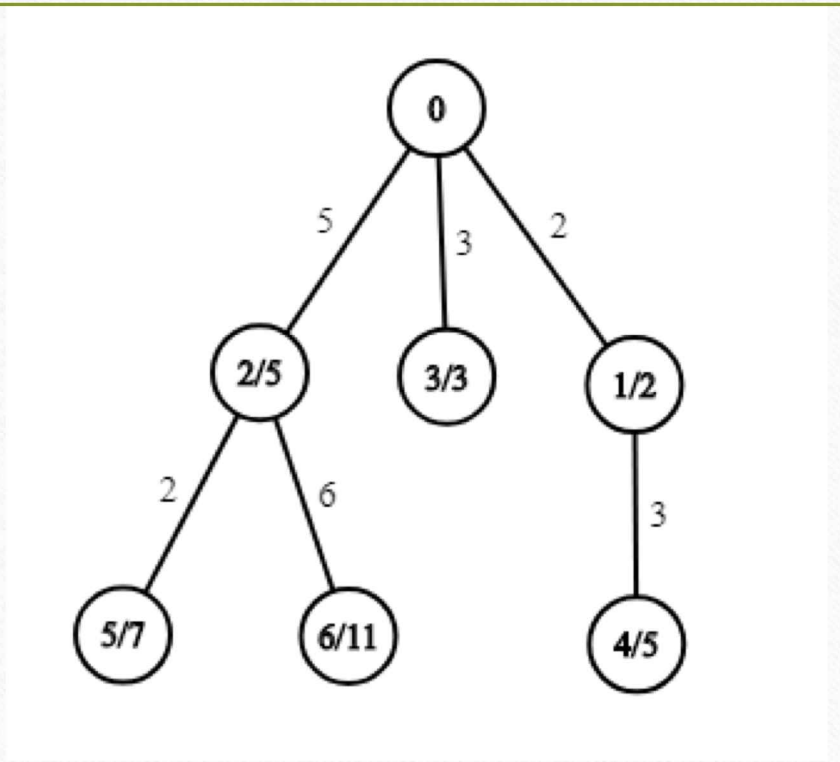
IOI 2011 Race (easier version)

- Assume K = 8
- The answer = 2
- {(2, 3), (3, 4)}
- (2 → 0 → 3), (3 → 0 → 1 → 4)



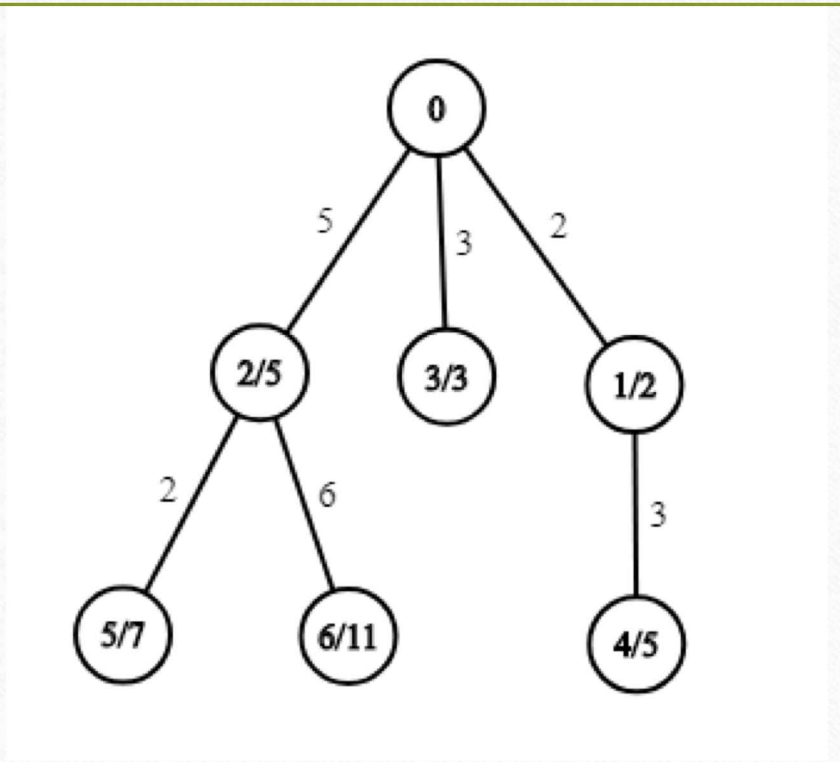
IOI 2011 Race (easier version)

- Let's fix node 0 as root
- Compute the distance from 0 to every node
  - Let's denote as dist[u]
- Then, for a pair of node (u, v), if
  - dist[u] + dist[v] == k
  - path(u, v) passing through 0
- Then path(u, v) satisfy the constraints



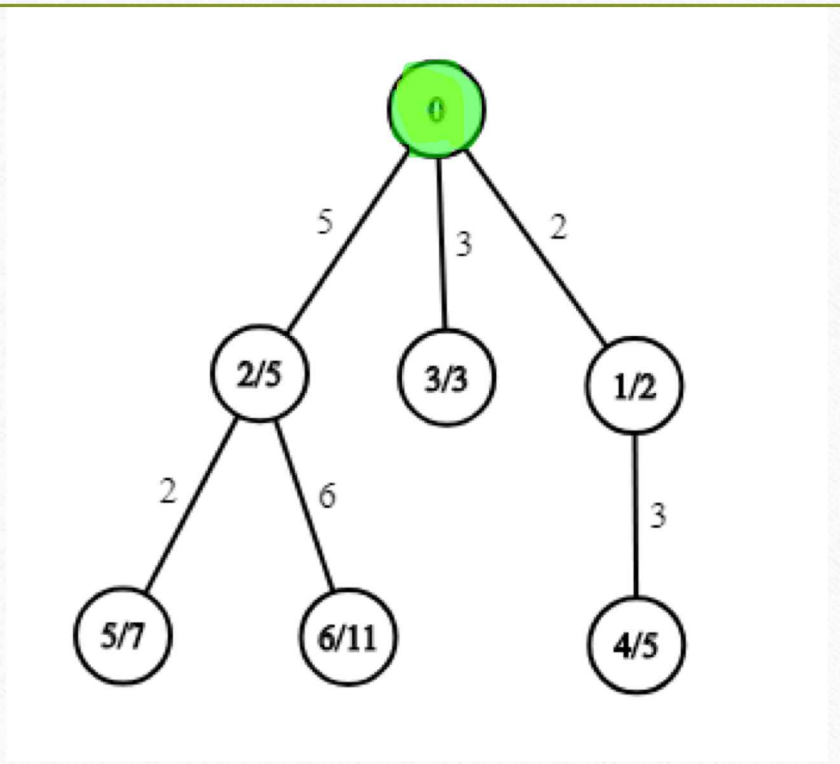
IOI 2011 Race (easier version)

- To find all pairs satisfying dist[u] + dist[v] = k:
  - When iterate each node u by DFS order from 0
  - ans += freq[k - dist[u]];
  - freq[dist[u]] += 1;
- To ensure it pass through node 0
  - When iterate each node u by DFS order from 0
  - ans += freq[k – dist[u]]
  - But only update freq[] when we finish iterating a whole subtree of 0



IOI 2011 Race (easier version)

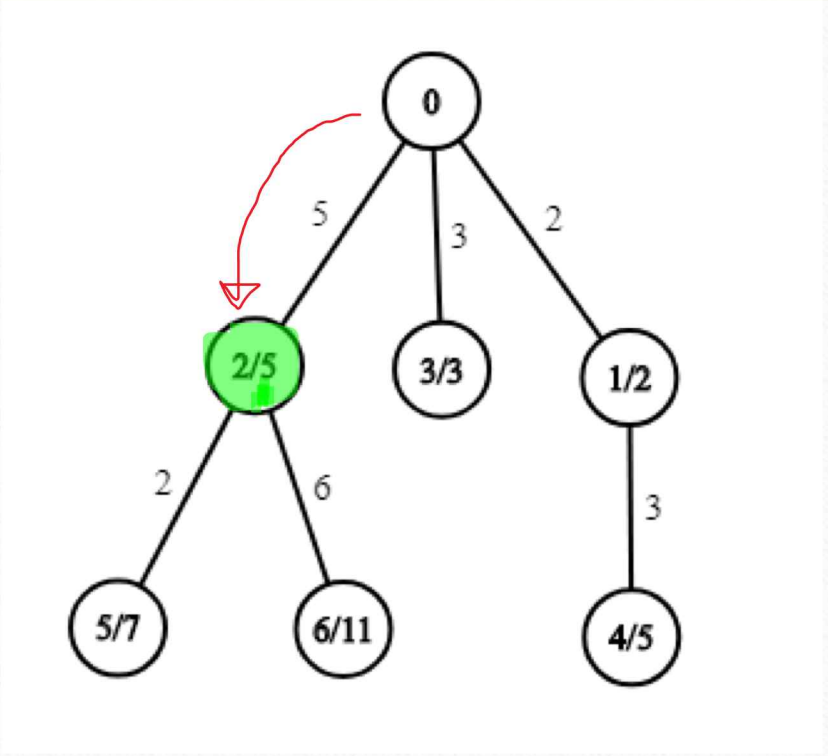
- We start iterating at node 0
- `Ans += freq[k - 0]`
- `Freq[0]++;`



IOI 2011 Race (easier version)

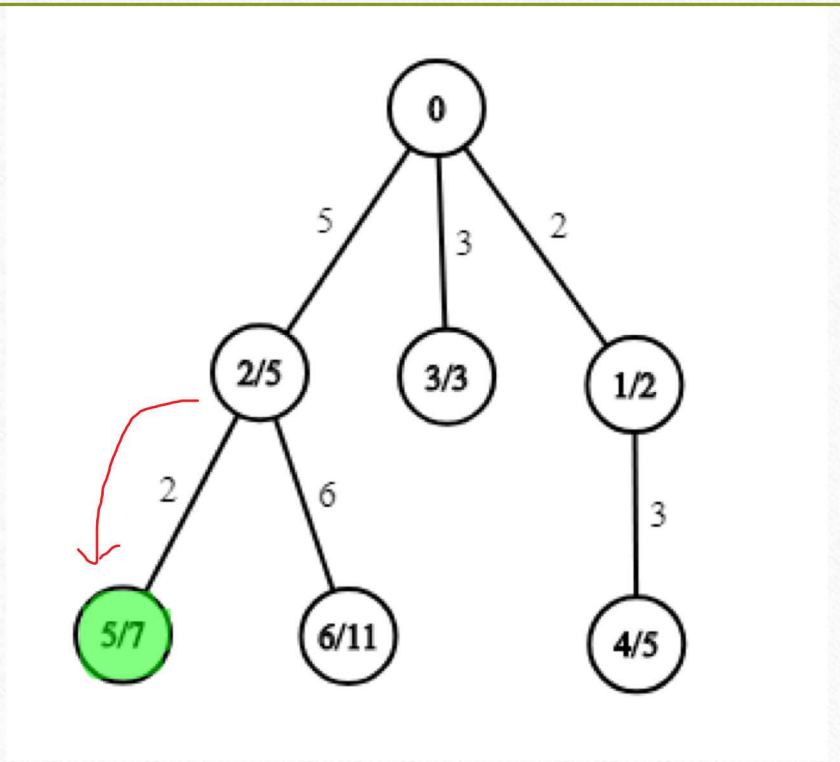
- `Ans += freq[k - 5]`

- Note that we ***won't*** perform freq[5]++;
- As we haven't iterate all the node in this subtree {2, 5, 6}
- To avoid counting path that not passing 0, we should not freq[5]++ currently



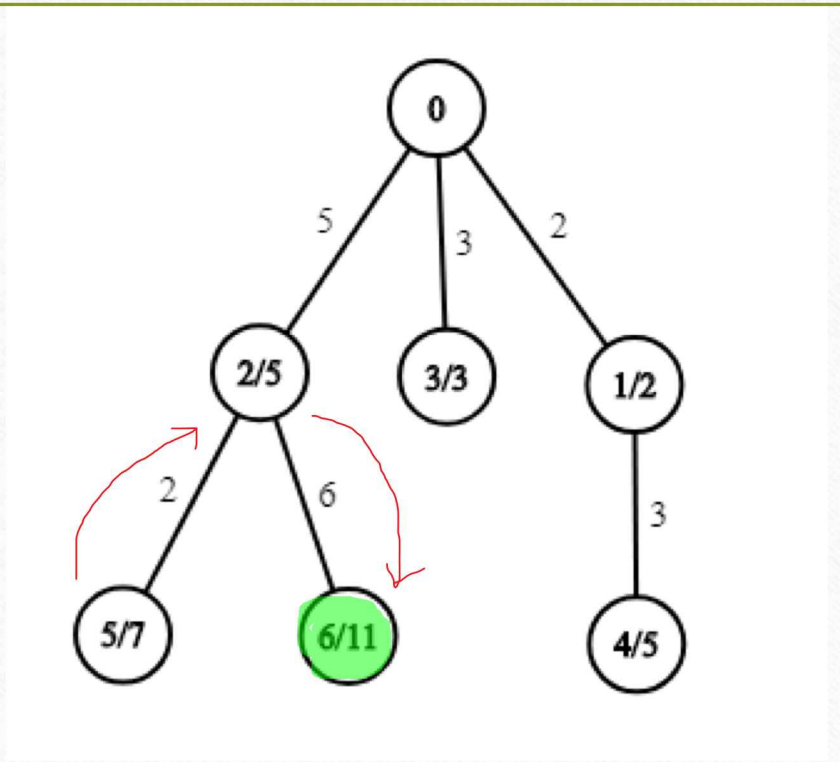
IOI 2011 Race (easier version)

- `Ans += freq[k - 7]`



IOI 2011 Race (easier version)

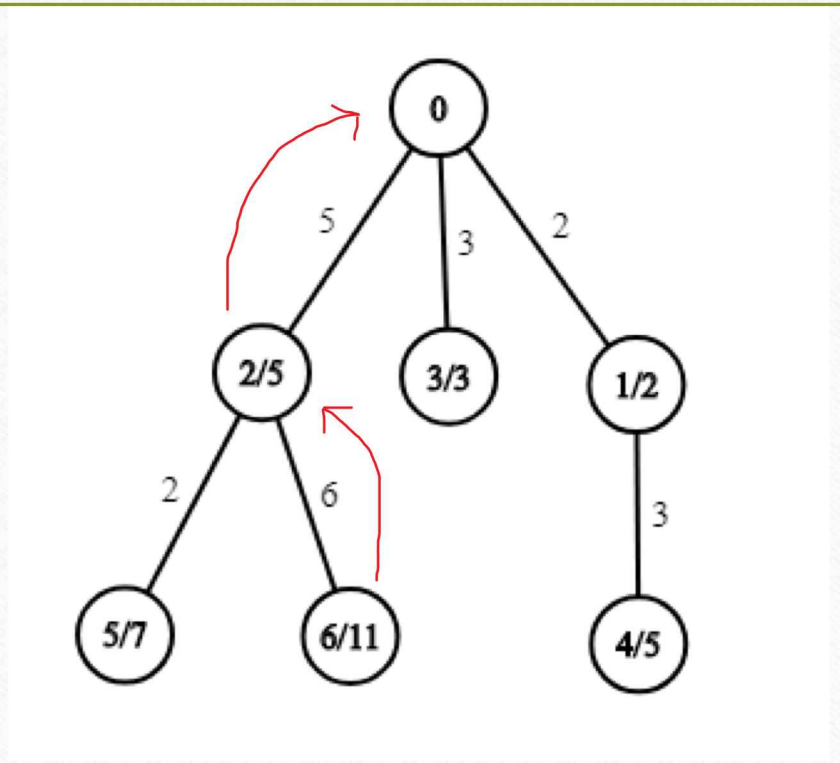
- `Ans += freq[k - 11]`



IOI 2011 Race (easier version)

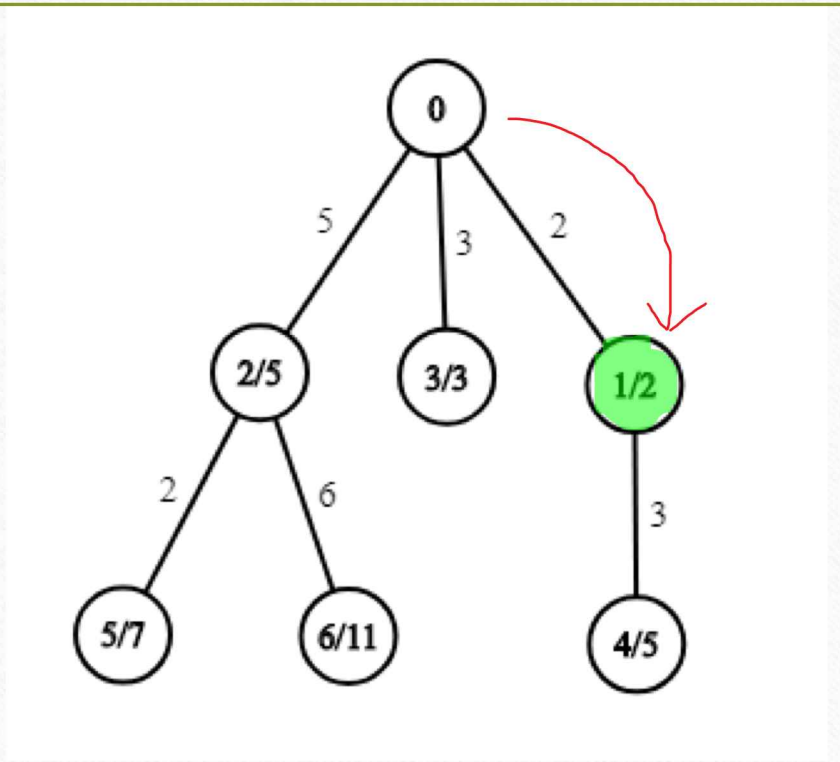
- Note that when our DFS go back to node 0
- This means we have iterated the whole subtree

- `Freq[5]++;`
- `Freq[7]++;`
- `Freq[11]++;`



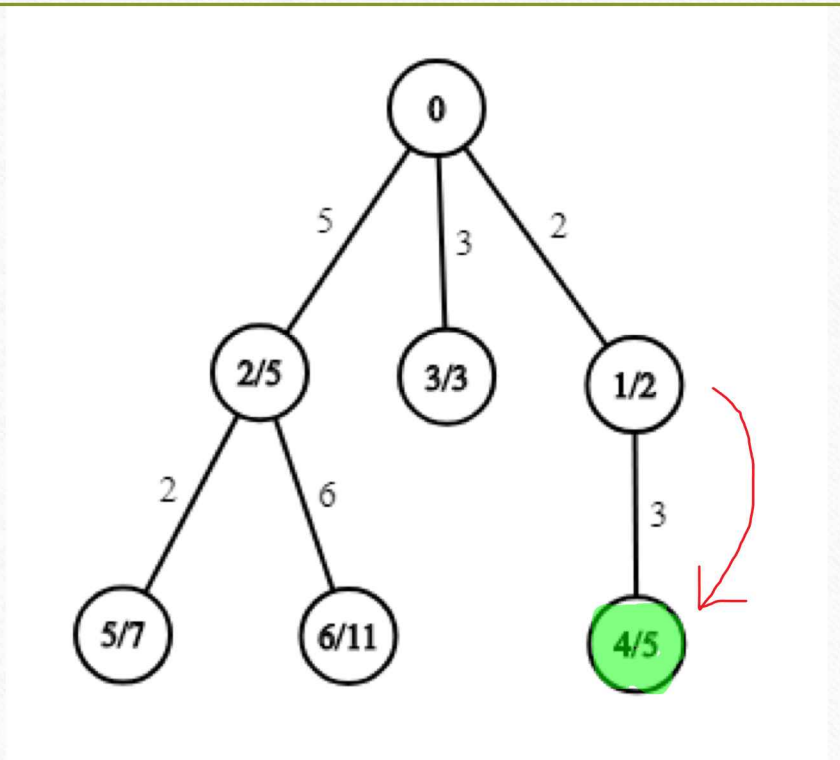
IOI 2011 Race (easier version)

- `Ans += freq[k - 2]`



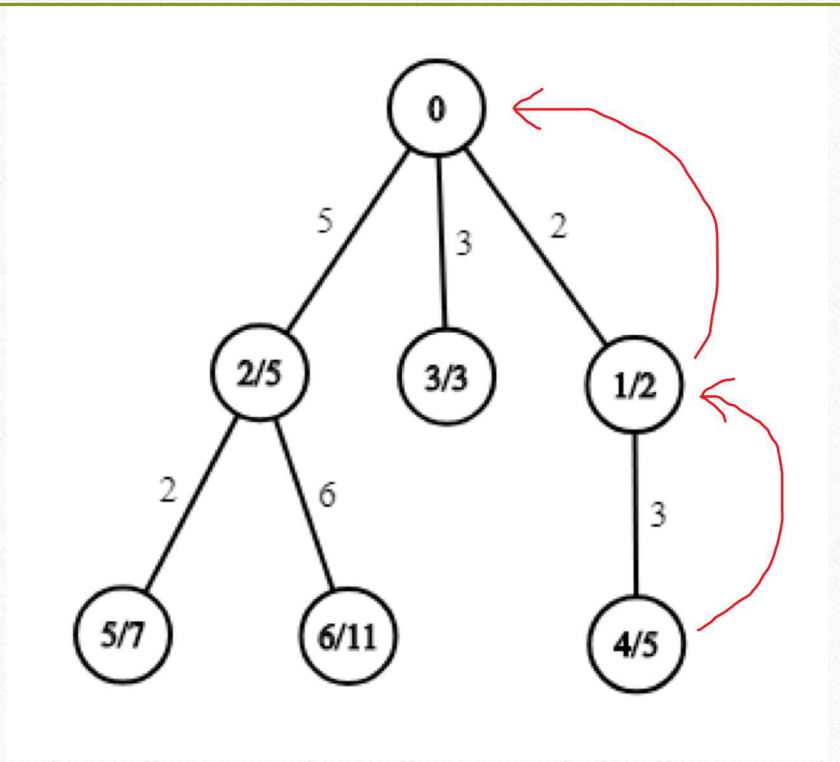
IOI 2011 Race (easier version)

- `Ans += freq[k - 5]`



IOI 2011 Race (easier version)

- `Freq[2]++;`
- `Freq[5]++;`
- …
- Do the rest yourself
- By this algorithm, we can solve this easier version in O(N)



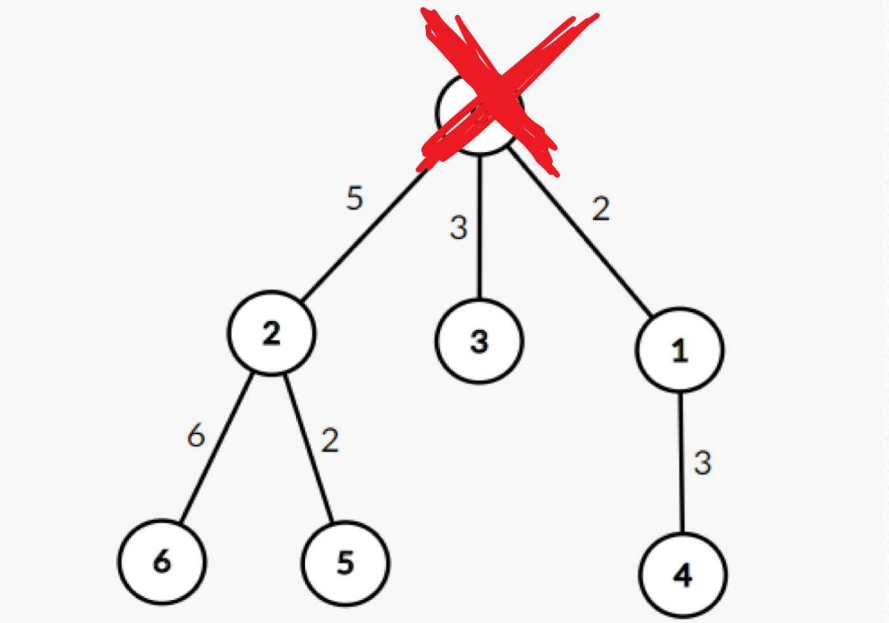
IOI 2011 Race (easier version)

```
void DFS(int x) {
      visit[x] = 1;
      ans += freq[k - dist[x]];
      if (x != 0) update_later.push_back(dist[x]);

      for (auto i: adj node of x) {
            if (visit[i]) continue;
            DFS(i);
            if (x == 0) {  // ----------------------→  This means we have iterated the whole subtree
                  for (auto j : update_later) freq[j]++;
                  update_later.clear();
            }
      }
}
```

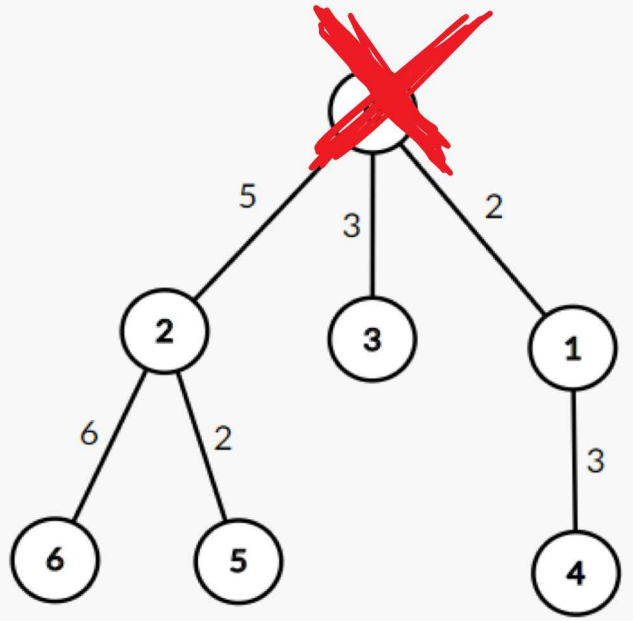
IOI 2011 Race

- Go back to our original problem
- We can iterate all the node, treat it as the root
- Note that when we choose u as the root, run the algorithm before
- Then we have considered ALL the path passing through u
- Which means we can delete node u for later iteration



IOI 2011 Race

- Note that for later iteration, we do not need to iterate all 7 nodes
- E.g. If we treat node as root in this order:
- {0, 2, 3, 1, 6, 5, 4}
- Number of nodes we will access = {7, 3, 1, 2, 1, 1, 1}
- For order {6, 5, 4, 3, 2, 1, 0}
- Number of nodes we will access = {7, 6, 5, 4, 3, 2, 1}



IOI 2011 Race

- In general, if we choose the node in the best order, the total number of node we visit in all DFS trials will be around N lg N
- But in the worst case, the total number of node we visit in all DFS trials will be N * (N - 1) / 2
- What is the best order?

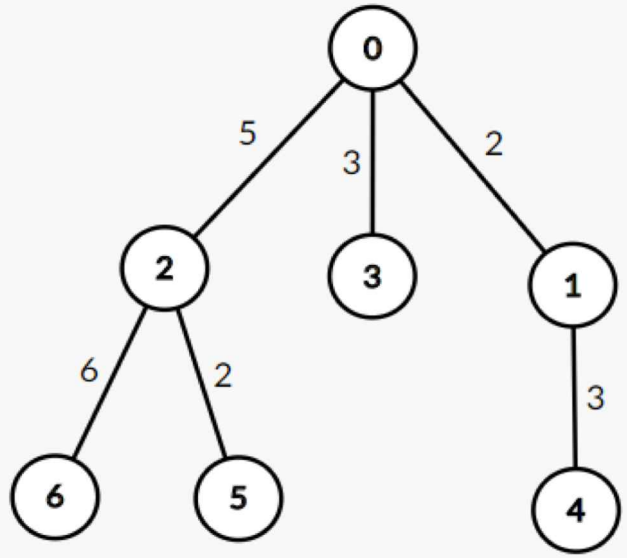
IOI 2011 Race

- The best order is, for each tree in the forest, we should select the Centroid of it as the root each time
- A centroid of a tree with N nodes is a node that after erasing it, all of the remaining component have a size <= N / 2
- Centroid(s) always exist(s) in a tree
- How to find a centroid? → Iterate all node and check the constraint directly

IOI 2011 Race

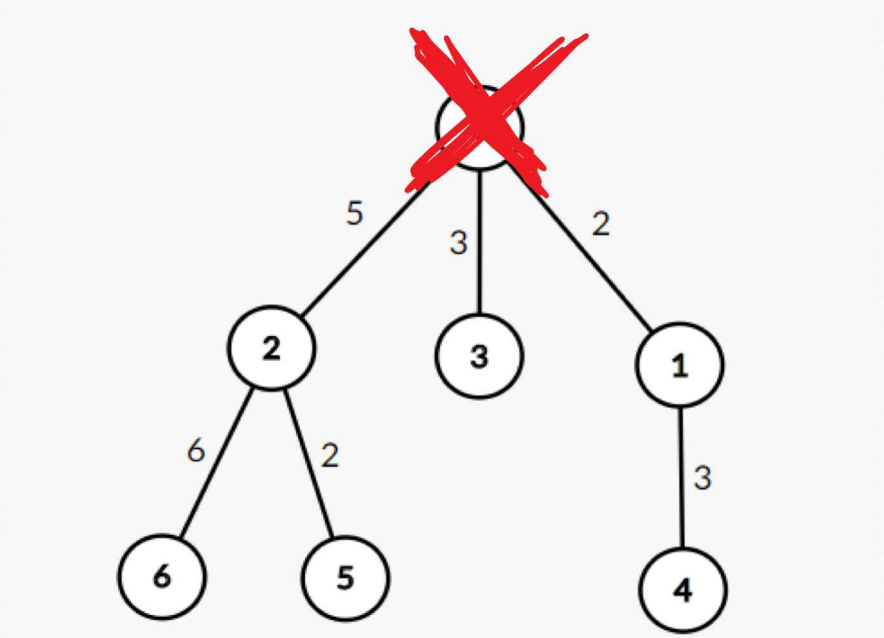
- Centroid = 0
- As the subtree after deleting node 0 is:
- {2, 5, 6}, {3}, {1, 4} → size = {3, 1, 2}

- All subtree size <= 7 / 2 = 3



IOI 2011 Race

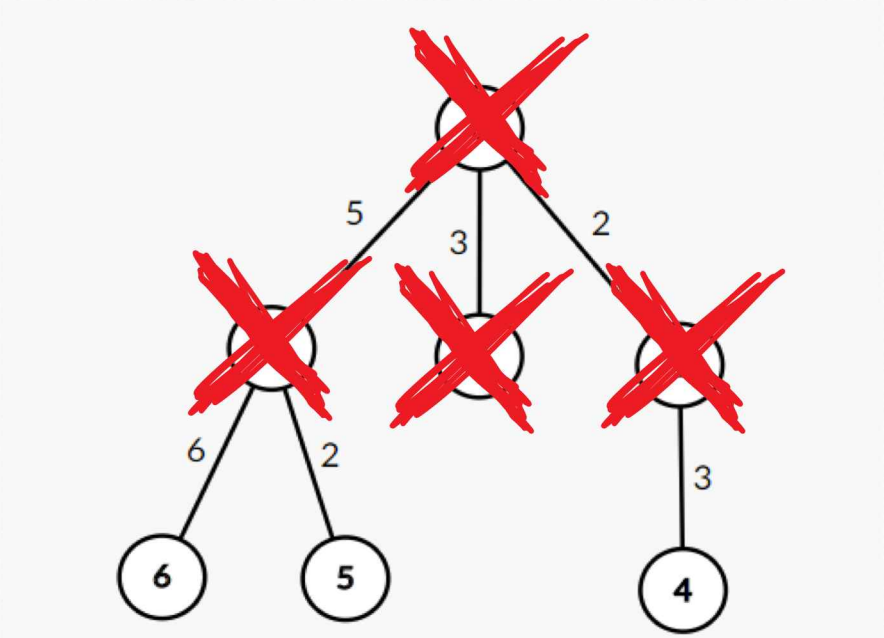
- For the subtree {2, 5, 6} → centroid = 2
- For the subtree {3} → centroid = 3
- For the subtree {1, 4} → centroid = 1 (or 4)



IOI 2011 Race

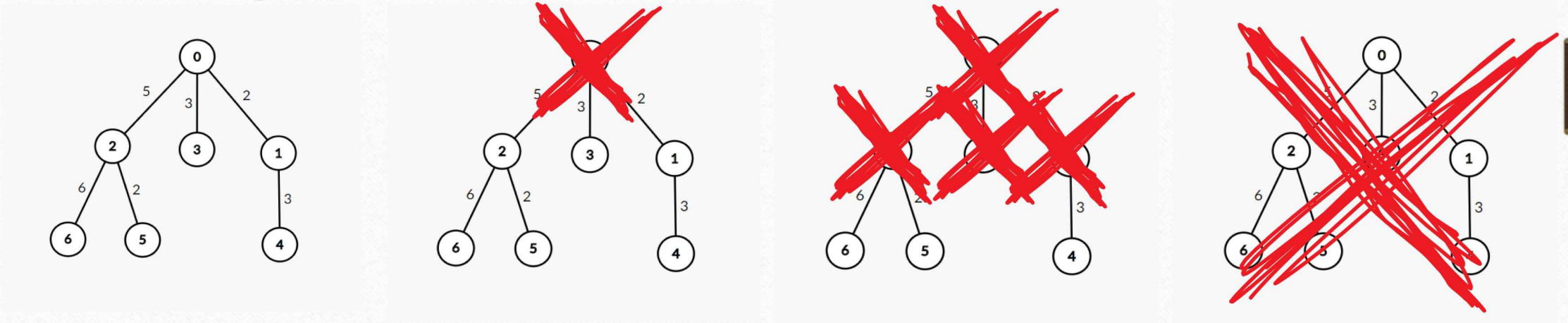
- For the subtree {4} → centroid = 4
- For the subtree {5} → centroid = 5
- For the subtree {6} → centroid = 6

- Done !



IOI 2011 Race

- Time complexity:



IOI 2011 Race

- Time complexity:
- We need 4 layers of deletion to delete all the node
- Note that for each layer, we use O(N) to iterate all the node
- Total number of layer = O(log(N)) as which time we perform a deletion on centroid, the remaining component size decreased by at least half
- Total Time Complexity O(N log N)

CDQ Divide and Conquer

Problem B

Alternative Query

Basic CDQ D&C

Alternative Query

- Problem Description:
  - Performing the following operation
  - 1. Insert(x) → Add x in S
  - 2. Query(x) → Count number of value v in S satisfying $v < x$
  - OFFLINE QUERY

Alternative Query

- 7
- Insert(3)
- Insert(5)
- Query(4)
- Query(3)
- Insert(6)
- Insert(2)
- Query(6)

- 1
- 0
- 3

Alternative Query

- You may find this task can be solved by Binary Tree, Segment tree, BIT……
- CDQ D&C is another way to solve
- To understand D&C, we may consider an easier version first
  - 1. Insert(x) → Add x in S
  - 2. Query(x) → Count number of value v in S satisfying $v < x$
  - OFFLINE QUERY
  - All Insert(x) operations are executed before all Query operations

Alternative Query (easy)

- 7
- Insert(3)
- Insert(5)
- Insert(6)
- Insert(2)
- Query(4)
- Query(3)
- Query(6)

Alternative Query (Easier version)

- If all Insert(x) go before Query(x)
- We can just simply sort all x in insert(x), binary search / two pointer to answer the query

Alternative Query

- If Insert(x) does not go before all Query(x)
- We can use Divide and Conquer to make Insert(x) go before Query(x)

Alternative Query

- `Insert(3)` → `Op(1)`
- `Insert(5)` → `Op(2)`
- `Query(4)` → …
- `Query(3)`
- `Insert(6)`
- `Insert(2)`
- `Query(6)`

- For each Query() operations, only some Insert() operations need to be considered (those go before that Query)
- Query(4) → Insert(3), Insert(5)
- Query(3) → Insert(3), Insert(5)
- Query(6) → Inst(3), Inst(5), Inst(6), Inst(2)
- To simplify, we may use ID to denote operation
- Op(3) → Op(1), Op(2)

Alternative Query

- `Insert(3)`
- `Insert(5)`
- `Query(4)`
- `---------`
- `Query(3)`
- `Insert(6)`
- `Insert(2)`
- `Query(6)`

- Operation-pairs to consider
- `Op3 → Op{1,2}`
- `Op4 → Op{1,2}`
- `Op7 → Op{1,2,5,6}`

- 1. Divide the operations sequence to half

Alternative Query

- Insert(3)
- Insert(5)
- Query(4)
- ---------
- Query(3)
- Insert(6)
- Insert(2)
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- 1. Divide the operations sequence to half
- 2. Consider Insert() in first part and Query() in second part only

Alternative Query

- `Insert(3)`
- `Insert(5)`
- `---------`
- `Query(3)`
- `Query(6)`

- Operation-pairs to consider
- `Op3 → Op{1,2}`
- `Op4 → Op{1,2}`
- `Op7 → Op{1,2,5,6}`

- 1. Divide the operations sequence to half
- 2. Consider Insert() in first part and Query() in second part only

Note that now, the operations sequence become a Insert-first-sequence

Alternative Query

- Insert(3)
- Insert(5)
- ---------
- Query(3)
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 0
- Ans(Op4) → 0
- Ans(Op7) → 2

- 1. Divide the operations sequence to half
- 2. Consider Insert() in first part and Query() in second part only

  Note that now, the operations sequence become a Insert-first-sequence
- 3. Use the solution of easy version to solve this scenario

Alternative Query

- Insert(3)
- Insert(5)
- ---------
- Query(3)
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 0
- Ans(Op4) → 0
- Ans(Op7) → 2

- 1. Divide the operations sequence to half
- 2. Consider Insert() in first part and Query() in second part only

Note that now, the operations sequence become a Insert-first-sequence

- 3. Use the solution of easy version to solve this scenario
- 4. Note that the operation-pair highlighted in blue is what we have calculated

Alternative Query

- Insert(3)
- Insert(5)
- Query(4)
- ---------
- Query(3)
- Insert(6)
- Insert(2)
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 0
- Ans(Op4) → 0
- Ans(Op7) → 2

- Note that the operation-pair highlighted in blue is what we have calculated
- What we have NOT calculated is the operation-pairs that Both operations in the pair belongs to a single part only

- So, what we should do is to apply the above algorithm to the first half, second half respectively

Alternative Query

- 1st part only
- Insert(3)
- Insert(5)
- --------
- Query(4)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 1
- Ans(Op4) → 0
- Ans(Op7) → 2

- Recursively do the first part
- Again, divide into 2 half and consider Insert in first, query in second only
- The red part are the pairs we calculated in this scenario
- Again, then do it recursively…

Alternative Query

- Insert(3)
- ---------
- Insert(5)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 1
- Ans(Op4) → 0
- Ans(Op7) → 2

- No update in this scenario

Alternative Query

- Query(4)
  - Operation-pairs to consider
  - Op3 → Op{1,2}
  - Op4 → Op{1,2}
  - Op7 → Op{1,2,5,6}
  - Ans(Op3) → 1
  - Ans(Op4) → 0
  - Ans(Op7) → 2
  - No update in this scenario

Alternative Query

- 2nd part
- Query(3)
- Insert(6)
- ---------
- Insert(2)
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 1
- Ans(Op4) → 0
- Ans(Op7) → 2

- Again, divide into 2 half and consider Insert in first, query in second only
- The red part are the pairs we calculated in this scenario
- Again, then do it recursively…

Alternative Query

- Query(3)
- ---------
- Insert(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 1
- Ans(Op4) → 0
- Ans(Op7) → 2

- No update in this scenario

Alternative Query

- Insert(2)
- ---------
- Query(6)

- Operation-pairs to consider
- Op3 → Op{1,2}
- Op4 → Op{1,2}
- Op7 → Op{1,2,5,6}

- Ans(Op3) → 1
- Ans(Op4) → 0
- Ans(Op7) → 3

- Done !!!

Framework

- Let solve(1, n) be the procedure to solve the Online query problem

```
Void solve(int l, int r) {
    Int mid = (l + r) / 2;
    Insert_list = Extract_Insert(l, mid);
    Query_list = Extract_Query(mid + 1, r);
    Solve_Insert_First_Query_easier_version(Insert_list, Query_list);
    If (mid – l > 1) Solve(l, mid);
    If (r – (mid + 1) > 1) Solve(mid + 1, r);
}
```

Alternative Query

- Time Complexity:
- Let T(n) = Time complexity of

  ```
  Solve_Insert_First_Query_easier_version(Insert_list, Query_list);
  ```

  Where n = sum of size of the two list
- Note that we will call
- `solve(1, 8)` → `solve(1, 4) + solve(4, 8)` → `solve(1, 2) + solve(3, 4)` ….
- Like a merge sort, this recursive calling give a lg(n) factor
- i.e. Time complexity = T(n) lg(n)
- For the algorithm above, T(n) = O(n lg n) → Time complexity = n lg(n) lg(n)

Problem B

UVaLive 5871 - Arnook's Defensive Line

CDQ D&C on update/query problem

UVaLive 5871 - Arnook's Defensive Line

- Problem Description:
  - Performing the following operation
  - 1. Insert(l, r) → Add a segment [l, r] in S
  - 2. Query(l, r) → Count number of segment [a, b] in S satisfying `a <= l && r <= b`
  - OFFLINE QUERY

UVaLive 5871 - Arnook's Defensive Line

- Solution 1:
  - 2D segment tree → O(n lg n lg n)
  - Drawbacks: Hard to implement, Large constants
- Solution 2:
  - The question is similar to the last question, except what is inserting & what is querying
  - If we can solve the "insert-first-query-then" version, we can use CDQ D&C to solve it

UVaLive 5871 - Arnook's Defensive Line

- Consider a simpler problem
  - All query command appear after all insert command
- Solution:
  - Sweeping line + 1d segment tree (dynamic / discretize)
  - Sort the query and segment according to the right bound
  - Insert the left bound of the segment when we sweep to the right bound of it
  - Query sum in [1, x] when we sweep to the right bound of a query

Framework

- You can use the same framework, just change the Solve_Insert_First…… part

```
Void solve(int l, int r) {
    Int mid = (l + r) / 2;
    Insert_list = Extract_Insert(l, mid);
    Query_list = Extract_Query(mid + 1, r);
    Solve_Insert_First_Query_easier_version(Insert_list, Query_list);
    If (mid – l > 1) Solve(l, mid);
    If (r – (mid + 1) > 1) Solve(mid + 1, r);
}
```

UVaLive 5871 - Arnook's Defensive Line

- Time Complexity:
- Let T(n) = Time complexity of

```
Solve_Insert_First_Query_easier_version(Insert_list, Query_list);
```

- For the algorithm above, T(n) = O(n lg n) → Time complexity = n lg(n) lg(n)
- Same as 2d segment tree but smaller constant and way easier to implement

CDQ Divide and Conquer

- This algorithm can widely apply on Insert/Query type problem if
  - It is easier to solve the "insert-first-query-then" version
  - The query is Offline (We know all the query before answering)
    If the input is encoded (e.g. real_x = (input_x + pre_ans) mod p,) CDQ D&C won't work
  - The thing we are querying is cumulative
    Cumulative example Sum, Count, and, or, xor, gcd, max, min
    Not cumulative example: median → median[full_set] != median of (median[1st half], median[2nd half])

Problem D

CDQ D&C for [Convex Hull Tricks] DP

CDQ Divide and Conquer

- CDQ D&C can not only apply to Update/Query Type Problem
- Also for DP (one of the example is CHT DP)

- What is CHT dp (I think you have heard in IOI 2016 - Alien) ?
- Simply speaking : `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
- If `g[], f[]` is monotonic, we can use CHT trick to optimize the dp
- If only `g[]` is monotonic, we can also use CHT trick + binary search on convex hull to optimize the dp
- What if both `g[] and f[]` is not monotonic ?

CDQ Divide and Conquer

- Simply speaking : `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
- If `g[], f[]` is monotonic, we can use CHT trick to optimize the dp
- If only `g[]` is monotonic, we can also use CHT trick + binary search on convex hull to optimize the dp
- What if both `g[] and f[]` is not monotonic …….

- Yes, you can still use CHT, with the help of CDQ D&C

CDQ Divide and Conquer

- What is DP Problem?
- You need to use the value of `dp[j]` to help you to find `dp[i] (j < i)`
- `E.g. A LIS Problem can be solved by dp`
- ```
  For(i = 1 to n)
    For(j = 1 to i - 1)
        if (val[i] > val[j])
            dp[i] = max(dp[i], dp[j] + 1)
  ```
- We can actually model it as a update/query problem
  ```

CDQ Divide and Conquer

- ```
  For(i = 1 to n)
    For(j = 1 to i - 1)
        if (val[i] > val[j])
            dp[i] = max(dp[i], dp[j] + 1)
  ```
- We can actually model it as a update/query problem
- ```
  For(i = 1 to n) {
    dp[i] = query(i);
   insert(dp[i]);
  }
  ```

CDQ Divide and Conquer

- Get some sense on how CDQ D&C and DP related ?
- Actually, for `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
  (no if-constraint in the dp transition is easier to understand)
- Query(dp[1]) → Insert(dp[1]) → Query(dp[2]) → Insert(dp[2]) → Query(dp[3]) → Insert(dp[3])
- We can treat Insert(dp[i]) as inserting dp[i] into S
- For Query(dp[i]), we can treat it as using previous inserted value in S, find the MAXIMUM value which is calculated by some function relating to
  1. A single optimal value in S
  2. Some parameter related to i

CDQ Divide and Conquer

- Query(dp[1]) → Insert(dp[1]) → Query(dp[2]) → Insert(dp[2]) → Query(dp[3]) → Insert(dp[3])
- We can treat Insert(dp[i]) as inserting dp[i] into S
- For Query(dp[i]), we can treat it as using previous inserted value in S, find the MAXIMUM value which is calculated by some function relating to
  1. A single optimal value in S
  2. Some parameter related to i
- Obviously, it is a offline query (we know what operation we are going to do at the beginning)
- Max() is also a cumulative function

CDQ Divide and Conquer

- Let's think about CDQ D&C on solving `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
- First, let's try to make a Insert_first_Query scenario
- i.e. insert(dp[1], dp[2] …… dp[mid]) → use them to solve query(dp[mid+1]… dp[n])

- Now we encounter two problem
- 1. Is it easier for us to solve the Insert_first_query scenario?
  Is it easy to find which is the optimal `j (1 <= j <= mid) for each i (mid + 1 <= i <= n)`?
- 2. Well, how we know the value of dp[1], dp[2] … dp[mid] at the same time? As dp[mid] may also calculated by dp[1] !!!

CDQ Divide and Conquer

- Let's think about CDQ D&C on solving `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
- Is it easier for us to solve the Insert_first_query scenario? → Yes, definitely yes!

- If g[j] is not monotonic, and if we need to do insert & query alternatively, we cannot maintain a convex hull
- But if all insert go before query, we can pre-build the convex hull according to g[j] first
- Also, we may not calculate in the order of dp[mid+1] → dp[mid+2] … dp[n]
- We can sort [mid + 1, n] according to f[i] first, and use this order to calculate
- As a result, f[i] and g[j] is both monotonic !!

CDQ Divide and Conquer

- Let's think about CDQ D&C on solving `dp[i] = max(dp[i], dp[j] + f[i] * g[j])`
- 2. How we know the value of dp[1], dp[2] … dp[mid] at the same time? As dp[mid] may also calculated by dp[1]

- Change the framework to
- `Solve(1, mid);`
- `Solve_insert_first_query();`
- `Solve(mid + 1, r);`

Summary

D&C on Tree / Centroid Decomposition

- Able to solve most problems in form of: Count number of path satisfying [_____] on a tree
  - Count number of path satisfying path length == K on a tree (basic form)
  - Count number of path satisfying path length <= K on a tree (use segment tree to maintain)
  - Count number of path satisfying path value sequence form a LIS
  - Count number of path satisfying path ……

CDQ D&C

- This algorithm can widely apply on Insert/Query type problem if
  - It is easier to solve the "insert-first-query-then" version
  - The query is Offline (We know all the query before answering)
    If the input is encoded (e.g. real_x = (input_x + pre_ans) mod p,) CDQ D&C won't work
  - The thing we are querying is cumulative
    Cumulative example Sum, Count, and, or, xor, gcd, max, min
    Not cumulative example: median → median[full_set] != median of (median[1st half], median[2nd half])
  - Sometimes on DP problems (which can be modeled as Insert/Query as well)
  - Usually able to reduce one dimension as it let us to rearrange the computation order
  - (e.g. Use 2d segment tree to maintain → 1d segment tree)

Finally

- CDQ = Acronym of 陳丹琦 (Female) which is a IOI contestant in 200x, representing the China
- She introduced CDQ D&C in her paper in 200x

- NOI 2014 Day 2 Q3 [Buying tickets] / ACM-ICPC ShenYang 2016 Problem I (Live Archive 7620)
- It is a question about applying an CHT DP on a tree
- The solution of this question is exactly Centroid decomposition + CDQ D&C lol

- But I recommend you to find some easier problems to make yourself familiar with them first

Practice Problem

- CDQ D&C:
- UVaLive 5871
- UVaLive 6374
- CEOI 2017 day-2 Building Bridges (can be found in CSAcademy)

- Centroid Decomposition
- IOI 2011 Race
- UVaLive 7148
- CSAcademy Round 58 – Path-Investions