Advanced Divide & Conquer

Brian Lee {lckcode}

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Sub-Topics

1. Basics of Divide & Conquer
2. Divide & Conquer on Range Query Problem
3. Divide & Conquer on Tree (Centroid Decomposition)
4. Divide & Conquer on Contribution Technique (CDQ Divide & Conquer)
   4.1. If have time left

Basics of D&C

1. We have a problem f with parameter n, f(n)

2. We can divide it to a SAME problem with SMALLER parameter f(m) (m < n)

3. By solving the f(m) first, we can solve the original problem much easier!
    3.1. f(m) may help us to compute f(n) easily
    3.2. Or by excluding f(m) from f(n), we can reduce f(n) to an easier problem g(n)

Basics of D&C – Sum of Geometric Sequence

Given a, k, m (m may not be a prime)

Find (a^0 + a^1 + a^2 + … a^k) % m

Solution 1:

General Formula: a^0 + a^1 + … + a^k = (a^k - 1) / (a - 1)

Basics of D&C – Sum of Geometric Sequence

Solution 1:

General Formula: a^0 + a^1 + ... + a^k = (a^k - 1) / (a - 1)

However, if gcd(a – 1, m) != 1, we may not able to find the modular inverse

Basics of D&C – Sum of Geometric Sequence

Solution 2:

Let f(n) = (a^0 + a^1 + … + a^n) % m

What if we know the answer of f(n / 2)?

f(n / 2) = (a^0 + a^1 + … a^(n/2)) % m

Does the answer of f(n / 2) able to help us find f(n) easily?

Basics of D&C – Sum of Geometric Sequence

When n is odd:

f(n / 2) + f(n / 2) * a^(n / 2 + 1) = a^0 + ... + a^n

When n is even:

f(n / 2) + f(n / 2) * a^(n / 2 + 1) - a^(n + 1) = a^0 + ... + a^n

Basics of D&C – Sum of Geometric Sequence

So, if we have known the value of f(n / 2)

We just need to know

- a^(n / 2 + 1) to find f(n) in odd case
- a^(n / 2 + 1) and a^(n + 1) to find f(n) in even case

Where a^k can be found by a BigMod algorithm

Basics of D&C – Sum of Geometric Sequence

Time complexity:

To calculate f(n), we need the value of f(n / 2)

→ we need to calculate log(n) value of f()

Calculating f(n) by f(n / 2) require us to find a^(n/2) e.t.c.

i.e. we need to do BigMod for log(n) times

Time complexity: O((log n)^2) (Actually O(log n) with careful analysis)

Basics of D&C – Sum of Geometric Sequence

We have a problem f with parameter n, f(n)

We can divide it to a SAME problem with SMALLER parameter f(m) (m < n)

By solving the f(m) first, we can solve the original problem much easier!

- f(m) may help us to compute f(n) easily → the above example
- Or by excluding f(m) from f(n), we can reduce f(n) to an easier problem g(n)
  - Repeated idea: Divide data into 2 parts, solve recursively, and combine cases that uses data in the both groups.

The following more advanced examples are about the 2nd type reduction

Example 1– Closest Pair

Given n points on a euclidean plane

Find the minimum distance between two points

Example 1– Closest Pair

Solution:

Divide:

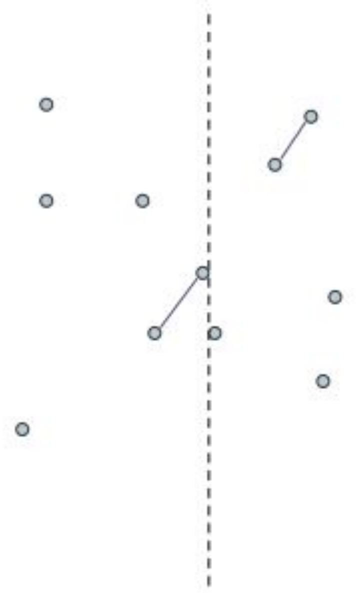
- Split the points by the median x-coordinate into P_l and P_r

Conquer:

- Solve recursively for both parts

Combine:

- Find min distance across boundary (focus)

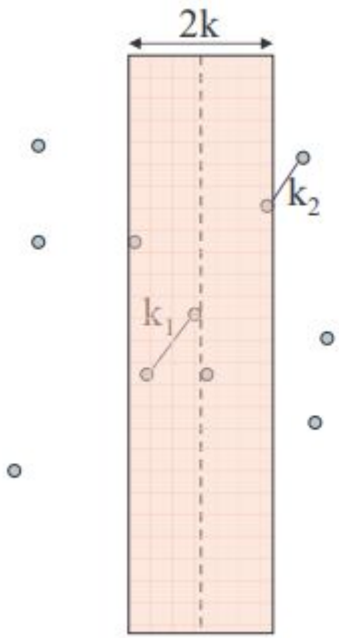


Example 1– Closest Pair

Let k_1 and k_2 be the min distance in P_l and P_r respectively.

and k = min(k_1, k_2)

Note that we are only interested in points with horizontal distance less than or equal to k to the median.

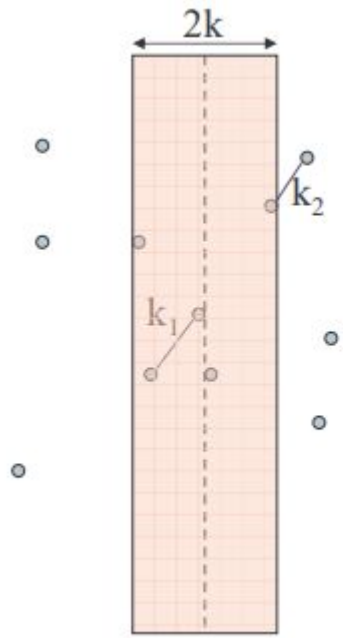


Example 1– Closest Pair

Let k_1 and k_2 be the min distance in P_l and P_r respectively.

and k = min(k_1, k_2)

Note that we are only interested in points with horizontal distance less than or equal to k to the median.

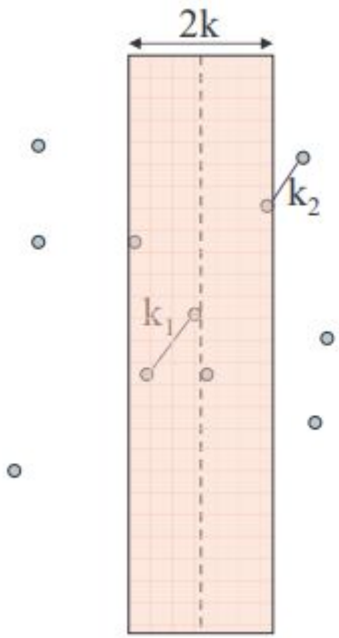


Example 1– Closest Pair

Combining:

Sort all points in the strip by y-coordinate

```
for i in range(len(strip)):
    for j in range(i, len(strip)):
        if strip[j].y - strip[i].y < k:
            update_min
        else:
            break
```



Example 1– Closest Pair

Claim: the inner loop iterates at most 7 times

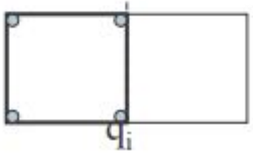
Key observations

- All points in the same strip must be at least k away from each other

Worst case:

Let q_i be the point we are considering

Total 8 points (corners of squares)



Example 1– Closest Pair

Complexity:

$T(n) = 2\,T(n/2) + O(\text{splitting cost aka sorting}) + O(\text{combining})$

$= 2\,T(n/2) + O(n \log n) + O(n)$

$= O(n \log^2 n)$

Improvement:

First sort by y and use quick select to find median

Splitting cost becomes linear

$T(n) = O(n \log n)$

D&C on Range Query

Given an Array A[1..n] and Q query (offline)

l, r is given in each query

For each query, find gcd(A[l], A[l + 1], ... A[r])

D&C on Range Query

Firstly, you may have seen a similar problem to find sum(A[l], A[l + 1] ... A[r]) instead of gcd(A[l], A[l + 1] ... A[r])

You can use partial sum to solve the sum version because:

- Sum(l, r) = Sum(1, r) - Sum(1, l - 1)

However, in gcd version, minus (-) operator is undefined

We can only define the add operator for gcd version

D&C on Range Query

Is it possible to extend the partial sum idea when minus operation is not defined?

YES!!! With the help of divide & conquer

D&C on Range Query

Consider a easier version of the original problem first:

- For each query (l, r), l <= n/2 <= r

In this case, we can compute two partial gcd array

- gcdA[i] = gcd(A[i], A[i + 1] ... A[n/2]) for all i <= n/2
- gcdB[i] = gcd(A[n/2], A[n/2 + 1] ... A[i]) for all i >= n / 2

To get the answer of query(l, r) where l <= n/2 <= r:

- Res = gcd(gcdA[l], gcdB[r])

D&C on Range Query

E.g. A = {2, 4, 6, 12, 3, 9, 6, 7} → n = 8, n / 2 = 4

gcdA = {2, 2, 6, 12} for 1 <= i <= 4

gcdB = {12, 3, 3, 3, 1} for 4 <= i <= 8

E.g. we want to find gcd(3, 6) → gcd(gcdA[3], gcdB[6]) = gcd(6, 3) = 3

Solve in O(log n)! (just find gcd of two number)

D&C on Range Query

To get the answer of query(l, r) where l <= n/2 <= r:

- Res = gcd(gcdA[l], gcdB[r])

We overcome the minus operator by building the partial gcd array from n/2

However what if the query(l, r) do not satisfy l <= mid <= r?

D&C on Range Query

However what if the query(l, r) do not satisfy l <= mid <= r?

Divide & Conquer help!

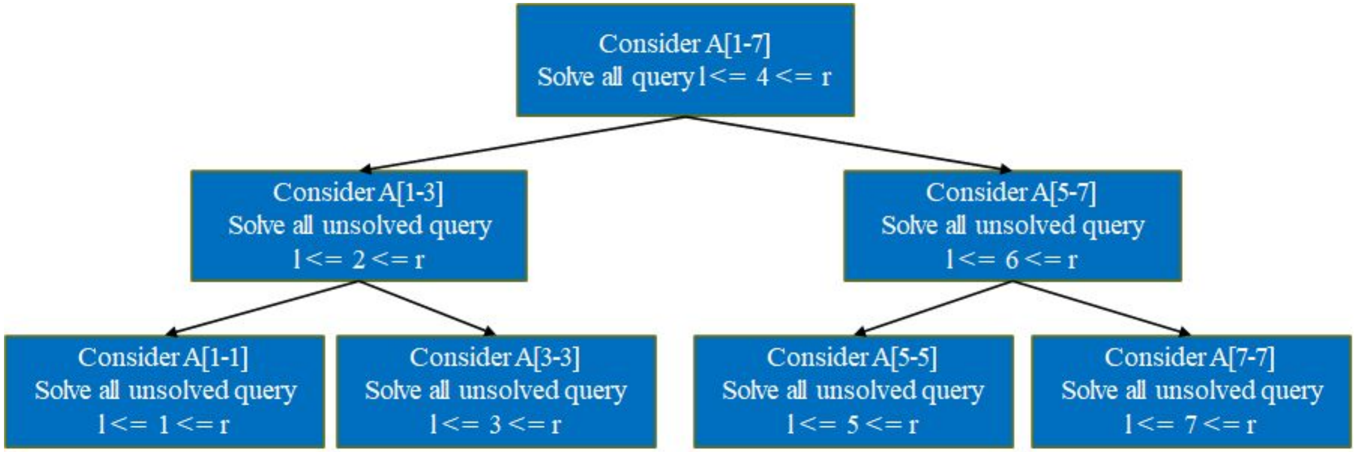
After solving all case with l <= mid <= r

We just care about the cases where:

- l <= r < mid → Consider the first half of array A only
- mid < l <= r → Consider the second half of array A only
- Which is the same problem with smaller scale

D&C on Range Query

When n = 7



D&C on Range Query

When n = 7,

query: [1, 4], [3, 5], [4, 6], [5, 6], [7, 7], [1, 3]

For the 1st instance, consider A[1-7] → solve all query l <= 4 <= r

- [1, 4], [3, 5], [4, 6]

For the 2nd instance, consider A[1-3] → solve all unsolved query l <= 2 <= r

- [1, 3]

...

D&C on Range Query

Time complexity for one instance to compute the partial gcd array:

- $O(n + \log M)$ where M is the largest value

Time complexity for all instance to compute the partial gcd array:

- $O(n \log n + n \log M)$

Time complexity to answer all the query: $O(Q \log M)$

Total time complexity: $O((n + Q) * \log(n + M)) \rightarrow$ one log only

D&C on Range Query

Somebody may think of using segment tree to solve Range Query problem

It is usually Okay but sometimes D&C can give a faster time complexity!

D&C on Range Query

Problem:

Given array A[1..n] where A[i] < 20

Q query l, r

For each query, find number of subsequence in subarray A[l, r] such that sum of subsequence % 20 == 0

D&C on Range Query

Solution D&C + dp or segment tree + dp

Node.dp[i] = number of subsequence such that sum % 20 = i

In D&C, we use partial sum concept to store a partial dp value

- dpA[i][k] = number of way to use A[i] to A[mid] to make a subset sum = k (mod 20)
- dpB[i][k] = number of way to use A[mid+1] to A[i] to make a subset sum = k (mod 20)
- X, dpA[i][j] * dpB[i][X - j]

However, segment tree time complexity will be O(Q log n * 20^2)

D&C will be O(n log n * 20 + 20 * Q), faster !!!

D&C on Range Query

In short: Steps to use D&C to solve range query problem

- Think whether it can be solved easily for query l <= mid <= r
- Put the queries to the suitable instance to solve it
- Use recursion to code the D&C part!

Common Form for Tree Query Problem

If you encounter an tree problem asking:

- Count total number of path satisfying xxxxxx
- Consider all the path, find the optimal pathing satisfying xxxxxx

Then, the problem is usually able to be solved by 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

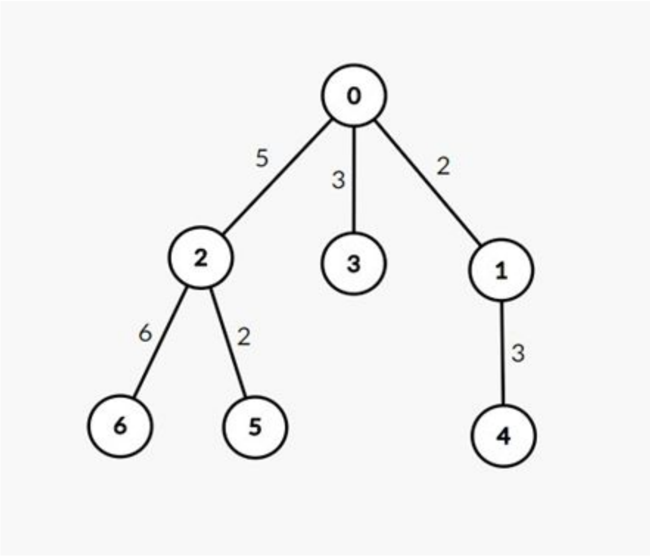
IOI 2011 Race

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

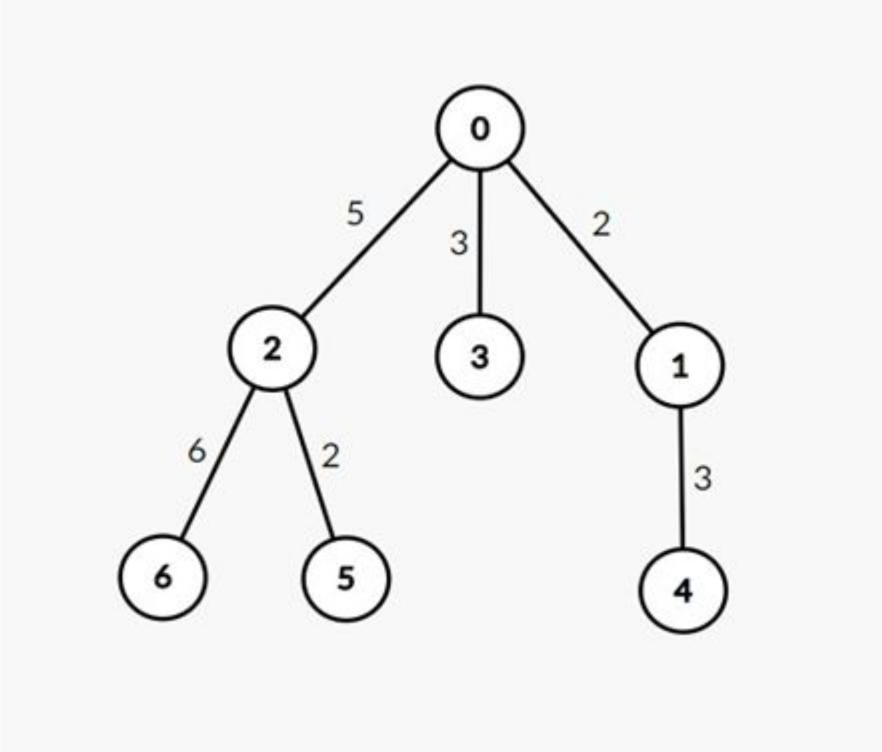
IOI 2011 Race

Assume K = 8

The answer = 2

{(2, 3), (3, 4)}

(2 → 0 → 3), (3 → 0 → 1 → 4)



IOI 2011 Race

Let's fix node 0 as root

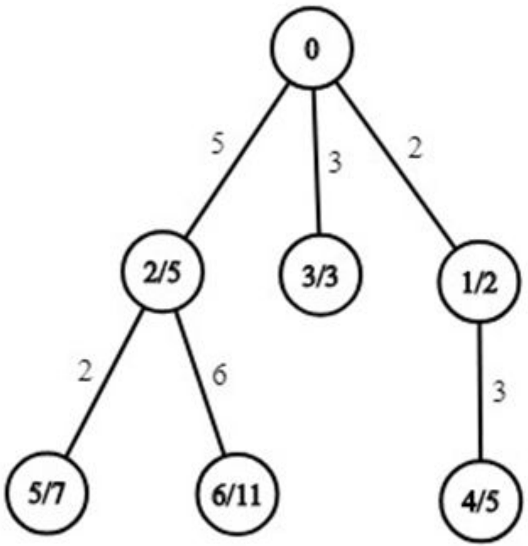
Compute the distance from 0 to every node

- 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



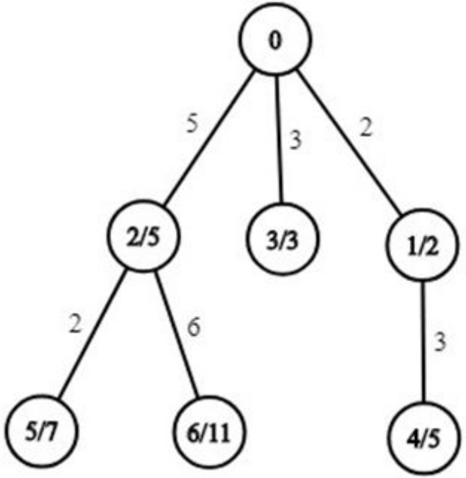
IOI 2011 Race

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

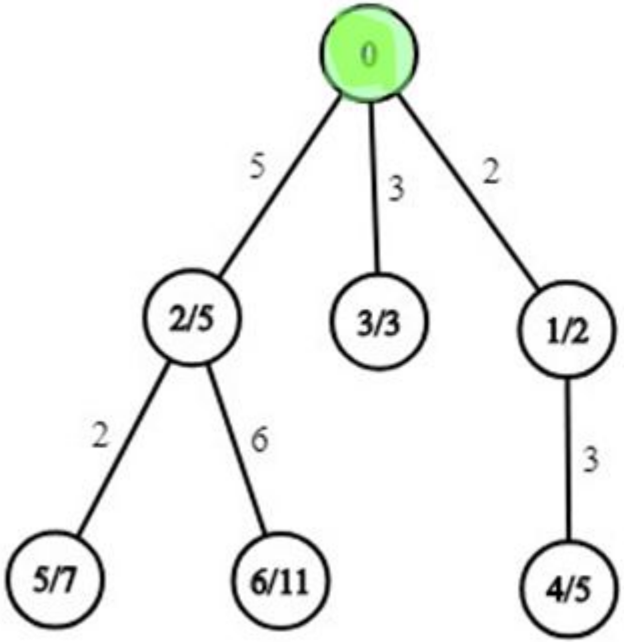


IOI 2011 Race

We start iterating at node 0

Ans += freq[k – 0]

Freq[0]++;



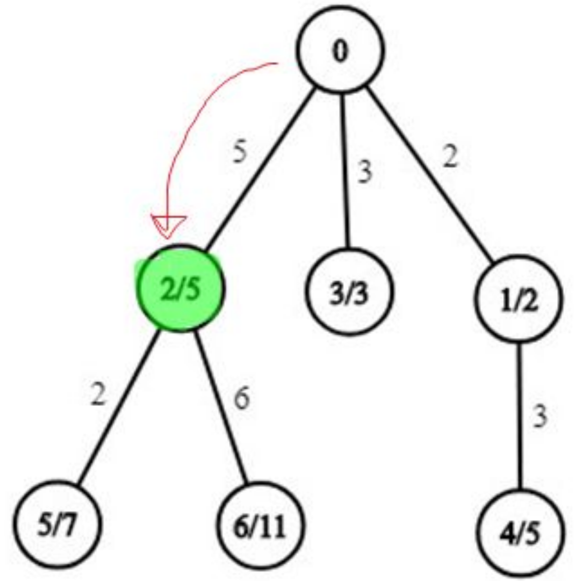
IOI 2011 Race

Ans += freq[k – 5]

Note that we won't perform freq[5]++;

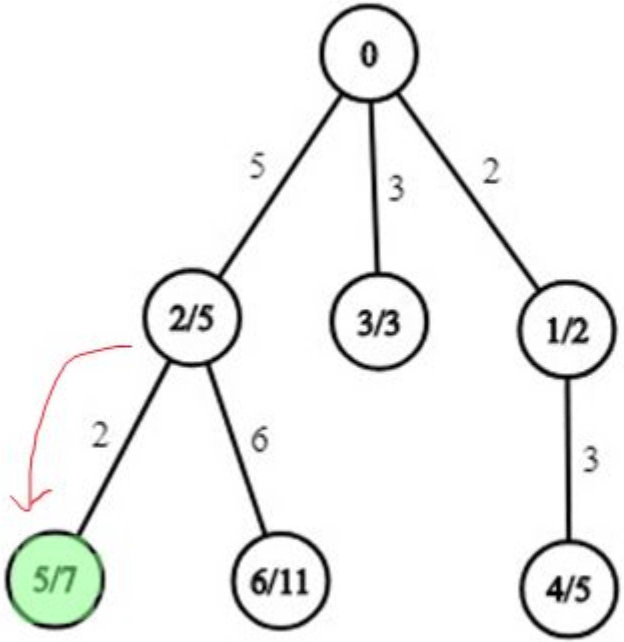
As we haven't iterate all the node in this subtree {2, 5, 6}

To avoid counting path that not passing 0, we should not freq[5]++ currently



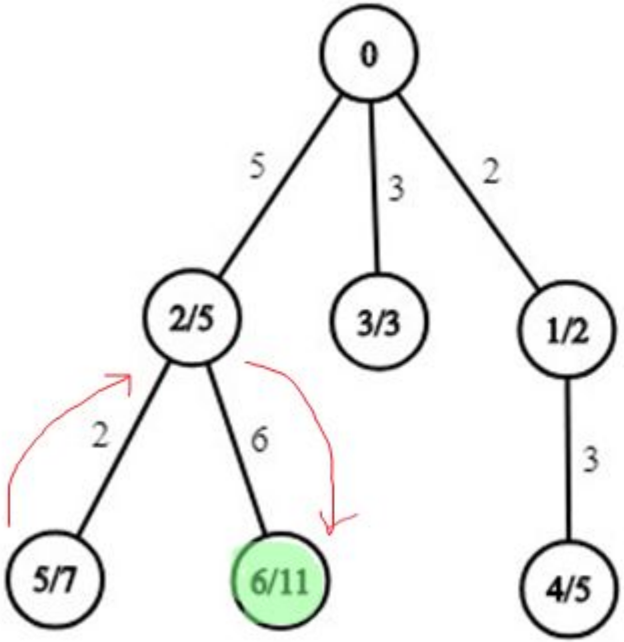
IOI 2011 Race

Ans += freq[k – 7]



IOI 2011 Race

Ans += freq[k – 11]



IOI 2011 Race

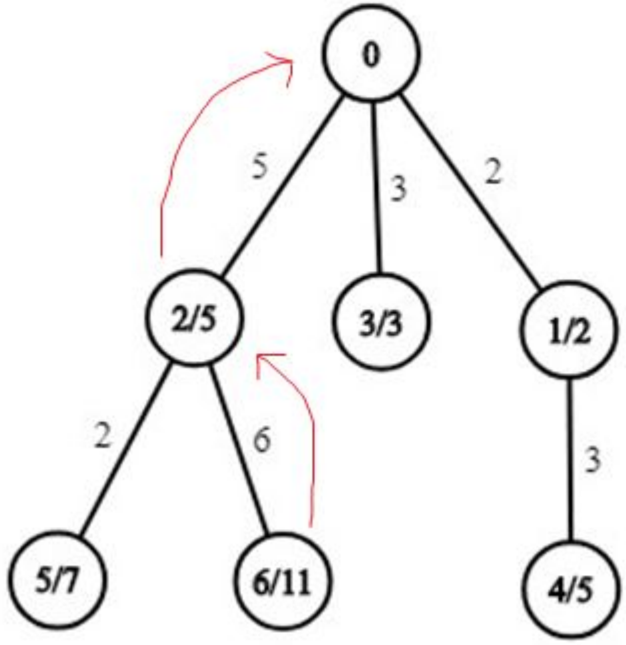
Note that when our DFS go back to node 0

This means we have iterated the whole subtree

Freq[5]++;

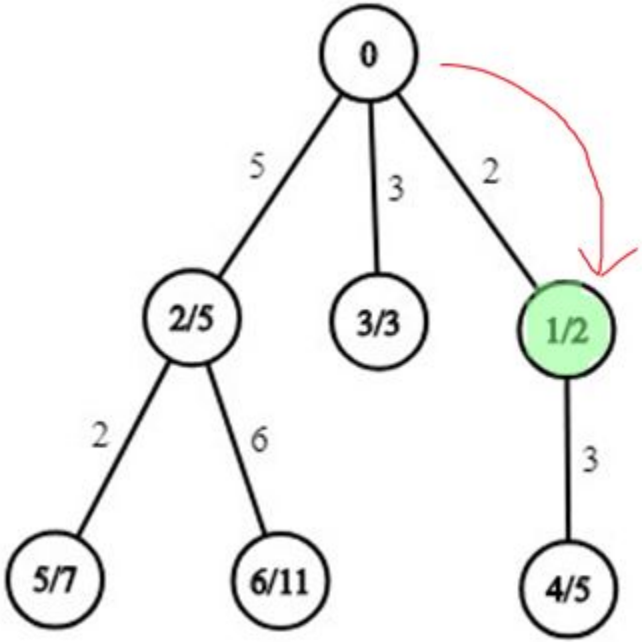
Freq[7]++;

Freq[11]++;



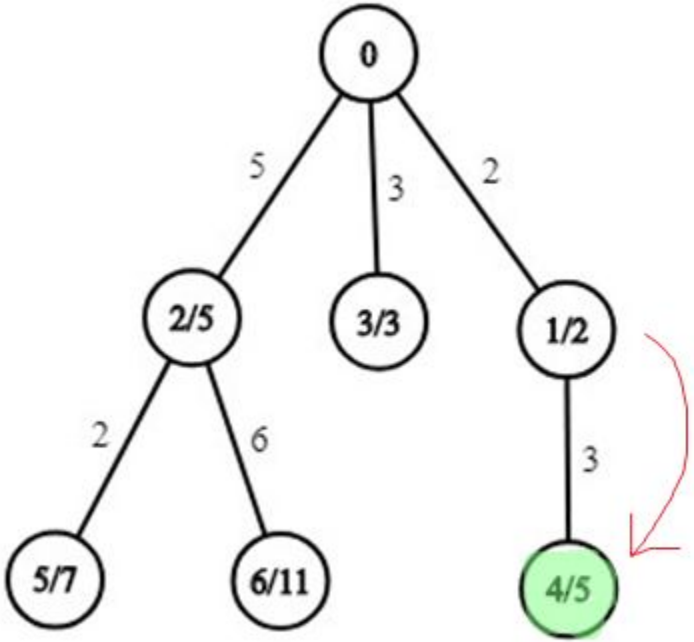
IOI 2011 Race

Ans += freq[k – 2]



IOI 2011 Race

Ans += freq[k – 5]



IOI 2011 Race

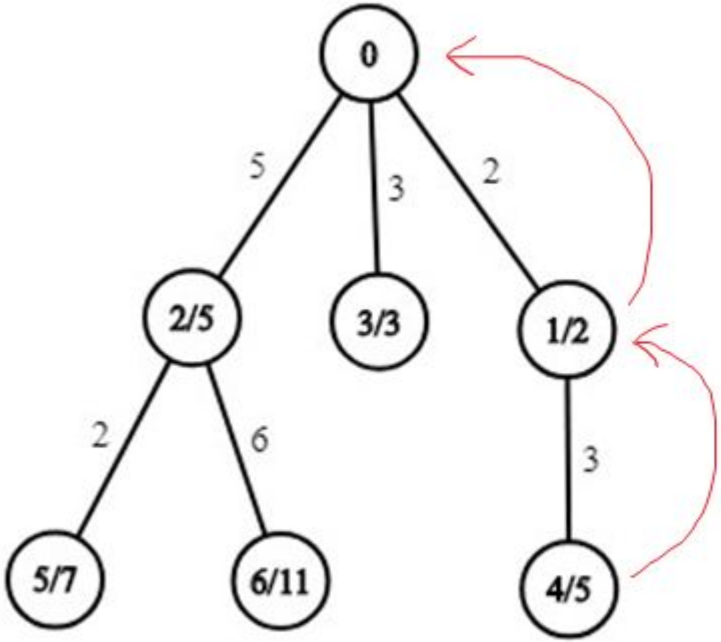
Freq[2]++;

Freq[5]++;

…

Do the rest yourself

By this algorithm, we can solve this easier version in O(N)



IOI 2011 Race

```
void DFS(int x) {
        visit[x] = 1;
        ans += freq[k – dist[x]];
        if (x != 0) update_later.push_back(dist[x]);
        else  freq[0]++;
        for (auto i: adj node of x) {
                if (!visit[i])
                 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

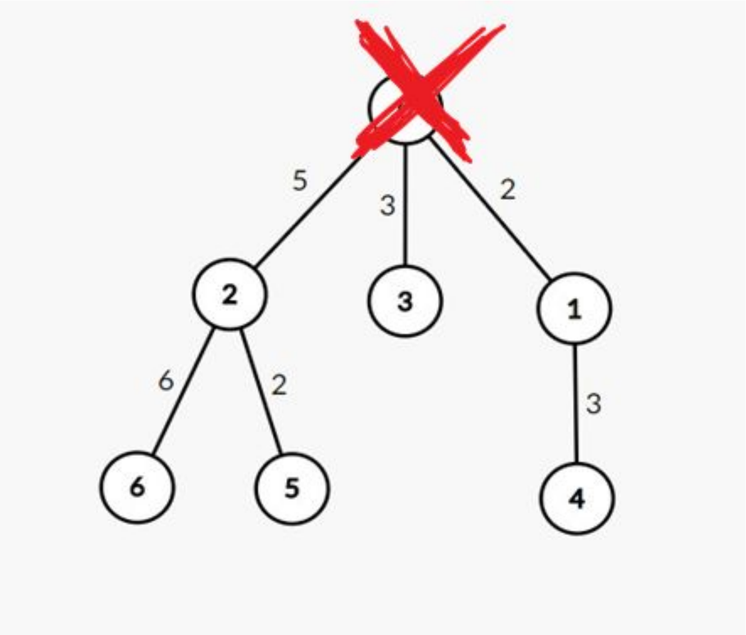
Go back to our original problem

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

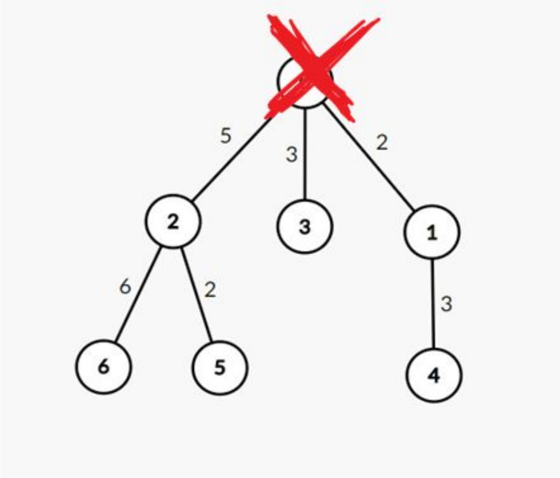
E.g. If we treat node as root in this order:

{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

```
dfs(v)
    Bool can_be_centroid = true
    For u in v.children
        dfs(u)
        If subsize(u) > N / 2
            can_be_centroid = false
    If N - subsize(v) > N / 2
        can_be_centroid = false
    If can_be_centroid then pick v
```

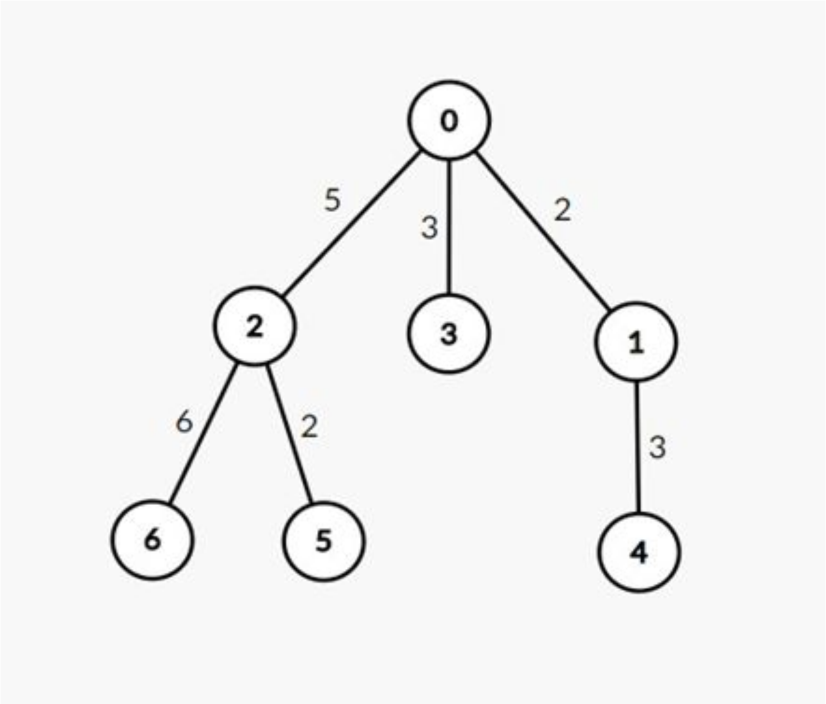
IOI 2011 Race

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

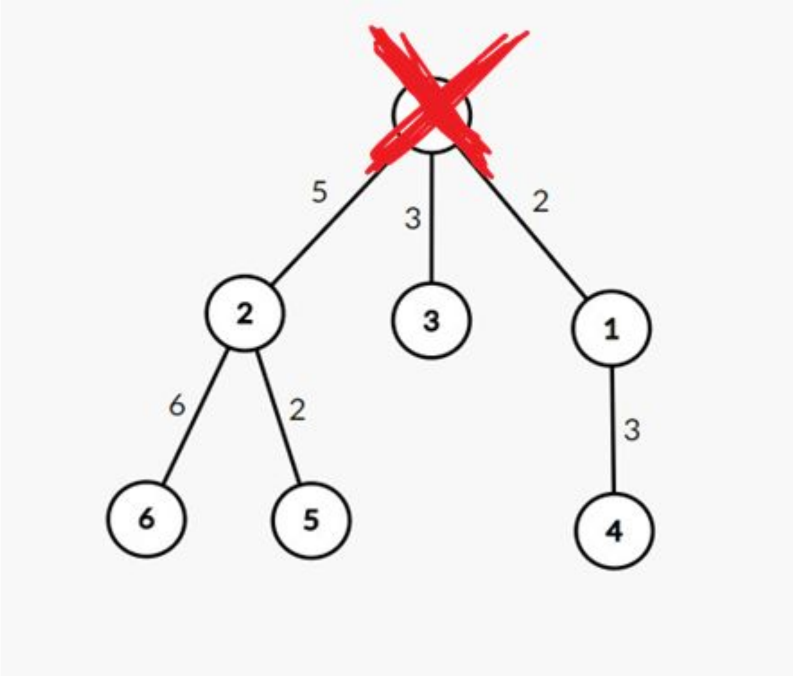


IOI 2011 Race

For the subtree {2, 5, 6} → centroid = 2

For the subtree {3} → centroid = 3

For the subtree {1, 4} → centroid = 1 (or 4)



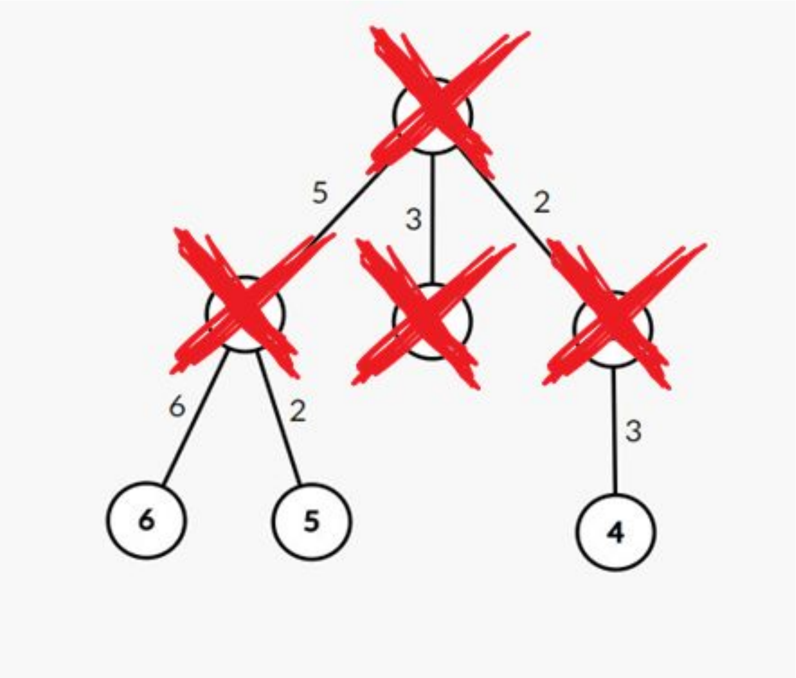
IOI 2011 Race

For the subtree {4} $\rightarrow$ centroid = 4

For the subtree {5} $\rightarrow$ centroid = 5

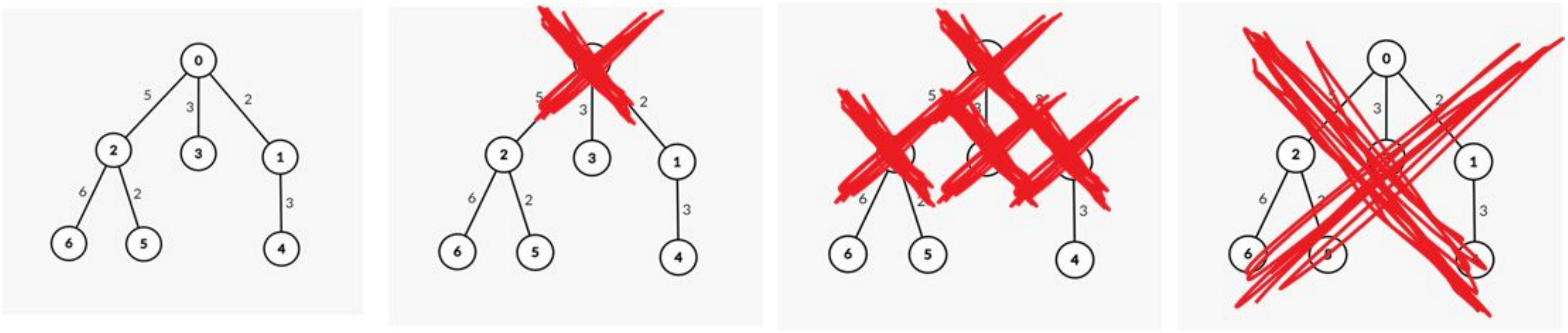
For the subtree {6} $\rightarrow$ centroid = 6

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)

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

When we encounter an insert-query problem

One of the idea is to maintain the inserted element with some data structure, such that we can perform the query efficiently

Another idea maybe calculate the contribution of each INSERT to each QUERY

CDQ Divide and Conquer

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

CDQ Divide and Conquer

7

Insert(3)

Insert(5)

Query(4)

Query(3)

Insert(6)

Insert(2)

Query(6)

Ans = {1, 0, 3}

CDQ Divide and Conquer

You may find this task can be solved by Binary Tree, Segment tree, BIT……

D&C & contribution technique 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

CDQ Divide and Conquer

7

Insert(3)

Insert(5)

Insert(6)

Insert(2)

Query(4)

Query(3)

Query(6)

CDQ Divide and Conquer

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

CDQ Divide and Conquer

If Insert(x) does not go before all Query(x)

We can use Divide and Conquer to make Insert(x) go before Query(x)

CDQ Divide and Conquer

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)

CDQ Divide and Conquer

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

CDQ Divide and Conquer

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

CDQ Divide and Conquer

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

CDQ Divide and Conquer

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
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

CDQ Divide and Conquer

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

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

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

CDQ Divide and Conquer

Let solve(1, n) be the procedure to solve the offline 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);
}
```

CDQ Divide and Conquer

Problem Description:

- Given an array A[1..n], find the number of inversion of the sequence
- Inversion: a pair (x, y) (1 <= x, y <= n) where x < y and A[x] > A[y]

E.g. [1, 3, 2, 5, 4]

- Inversion: (3, 2), (5, 4) → 2

CDQ Divide and Conquer

Solution:

You can treat it to a insert-query problem

Iterate the array from the beginning to the end

- query the number of elements in S greater than A[i]
- Inserting A[i] to S

CDQ Divide and Conquer

E.g. A[] = [3, 2, 5, 4]

Query(3)

Insert(3)

Query(2)

Insert(2)

Query(5)

Insert(5)

Query(4)

Insert(4)

Sum of all query answer is the number of inversion

CDQ Divide and Conquer

So, we can transform it to insert-query and use CDQ D&C to solve it

CDQ Divide and Conquer

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)

CDQ Divide and Conquer

CDQ + sorting + DS 3D query

N, (x, y, z) -> (p, q, w) x > p and y > q and z > w

A = [(x1, y1, z1), (x2, y2, z2) ...] -> x_i < x_{i + 1} ... A[i].y < A[y + 1].i

```
Void solve(l, r) { // calculate all contribution
    sort(l, r, by y) // contribution of [l, mid] to [mid + 1, r]
    For each query
        -> two types (in first half, in second half)
            First half -> put it in BIT
            Second half -> calculate_contribute_of_first_half_to_itself(z)
    undo_sort(l, r)
    solve(l, mid) // contribution of [l, mid] to itself
    solve(mid + 1, r) // contribution of [mid + 1, r] to itself
}
```

Another Example - different perspective

Given n points on a Euclidean plane such that their x and y coordinates are distinct

Find sum of manhattan distance between all points (x1, y1) and (x2, y2) such that x1 < x2 and y1 < y2.

I.e. $$\sum_{i} \sum_{x_j < x_i \wedge y_j < y_i} (x_i - x_j) + (y_i - y_j)$$

Source: British Olympiad in Informatics

Another Example

Idea:

First sort by x

In each recursion call

1. Divide by median of x
2. Calculate contribution of left and right segment
3. Calculate contribution across boundaries
4. Merge segments by y-coordinate

Note that if $x1 < x2$ and $y1 < y2$, manhattan distance = $(x2 + y2) - (x1 + y1)$

Another Example

```cpp
void solve(int l, int r, vector<Point> &coords, vector<int> &ans) {
    if (l + 1 >= r) return;
    int m = (l + r + 1) / 2;
    /*
        guarantees all x in [l, m) < all x in [m, r)
        x-coordinates are sorted
    */
    solve(l, m, coords, ans);
    solve(m, r, coords, ans);
    /*
        After calling solve, the order of x may not be increasing in individual segments, but y are.
        The property that all x in [l, m) < all x in [m, r) still holds
    */
    multiset<int> s;
    for (int i = m; i < r; ++i)
        s.insert(coords[i].x + coords[i].y);

```

Another Example

```
// Updates answer for points on the left segment with 2 pointer.
int j = m;
for (int i = l; i < m; ++i) {
    while (j < r && coords[j].y <= coords[i].y) {
        s.erase(s.find(coords[j].x + coords[j].y));
        j++;
    }
    if (!s.empty()) {
        if (ans[coords[i].t] == 0) {
            ans[coords[i].t] = *s.begin() - coords[i].x - coords[i].y;
        } else {
            ans[coords[i].t] = min(ans[coords[i].t], *s.begin() - coords[i].x - coords[i].y);
        }
    }
}
```

Another Example

```cpp
    vector<Point> buffer;
    // O(n) merging of two sorted sequence
    merge(coords.begin() + l, coords.begin() + m,
          coords.begin() + m, coords.begin() + r,
          back_inserter(buffer), [] (auto lhs, auto rhs) {
              return lhs.y < rhs.y;
    });
    copy(buffer.begin(), buffer.end(), coords.begin() + l);
}
```

Another Example

Handy when the problem involves some kind of tuple ordering

Relation with range query

- Treat query/insertion time as a variable (t) in the tuple
- Queries to be processed in ascending order of t

CDQ Divide and Conquer

D&C often help you solve Data Structure problem (e.g. insert-query / range query problem)

With the help of D&C, we usually able to figure out a algorithm to get rid of using advanced data structure (2D segment → segment tree) or (Segment tree → array / 2 pointer)

D&C usually run in good constant time!

CDQ Divide and Conquer

Strongly recommended:

https://oi-wiki.org/misc/cdq-divide/

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