Problem Analysis

Malaysian Computing Olympiad 2016

A. Cable Car

Subtask 1: O(N^3) Brute force

Iterate through all pairs of mountains. For each candidate pair **(i, j)**, iterate through **i+1**, **i+2**, **…**, **j-2**, **j-1** to check if **(i, j)** is a valid station pair. Keep track of longest range so far.

A. Cable Car

Subtask 1.5: O(N^2) Brute Force

Observation 1: the rightmost valid station for ***i*** is also the first station to the right of ***i*** that higher or equal to ***i***

Consider each mountain as a candidate for the left station. For each left station, find the rightmost valid station.

E.g. {<u>3</u>, 1, 1, 1, <u>4</u>, 2, 1}

A. Cable Car

Subtask 1.5: O(N^2) Brute Force

The catch: consider the example {9, 3, 1, 1, 1, 1, 4}

Our algorithm: {9, 3, 1, 1, 1, 1, 4}

Correct answer: {9, 3, 1, 1, 1, 1, 4}

Observation 2: for **(l, r)**, need to consider cases where **h[l]>=h[r]**, and **h[l]<=[r]**

So, scan once, reverse the array, and scan again!

A. Cable Car

Subtask 2: O(N) Sliding Window

Observation 3: (l, r) is longest possible for **l**, then setting **l<i<r** as a left station will not yield any longer ropeway.

Window starts at **l**, extends to first **r** such that **h[l]<=h[r]**. Restart sliding window with new **l** = old **r**.

Reversing the array still applies.

B. Relay Race

Subtask 1: Unweighted graph

- One can notice that the Fluffy should always wait at the center of the shortest path from S to E.
- Hence, D will be equal to ceil(shortest path distant / 2).
- And the shortest path can be found using a standard breadth first search (BFS) algorithm
- Total complexity: O(V+E) (BFS)

B. Relay Race

Subtask 2: Weighted graph

- If we are going to use the previous algorithm, we would get a wrong answer in this subtask.
- This is because the optimal waiting point is not always in the shortest path between S to E.
- Imagine the graph below: The shortest path from S to E is S -> E while Fluffy should actually wait at X which will yield a shortest require D.

B. Relay Race

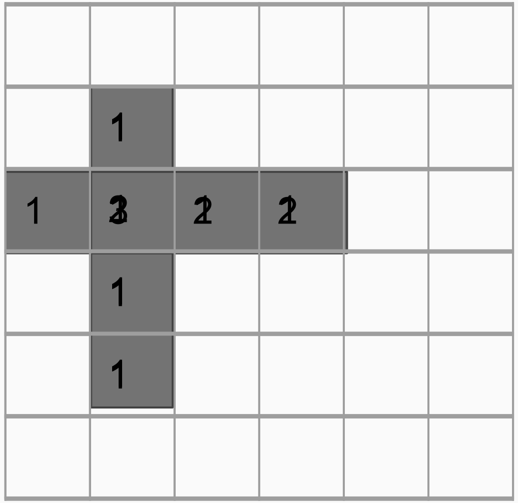
Subtask 2: Weighted graph (The Solution)

- To figure out D, first we need to figure out which is the optimal waiting point, when we know the optimal waiting point, D can be found easily
- The optimal waiting point, X should have the following properties
    - ➢ If X is the optimal point, then D = max(dist S to X, dist X to E)
    - ➢ D is minimal over all point.
- The dist can be found using Dijkstra. Since, the starting point S is fixed, and ending point E is fixed. We can run 2 times dijkstra, 1 from S, 1 from E and the dist we need are all precomputed.
- **Complexity:** O(E + Vlog V)

C. Painting

Subtask 1: O(N^2) Simulation

- Just do it
- 2D array (e.g. painting[5][5])



C. Painting

Subtask 2: O(N) Horizontal Solution

- Note that for N=1, we only need to worry about horizontal lines
- We consider only the edges of the interval as cells in the intervals remain the same
- Should lead to final solution once solved

C. Painting

O(N^2) Simulation

		1	1	1	1			O(N)
1	1	2	1	1	1			O(N)
1	1	3	2	2	1			O(N)
1	2	4	3	2	1			O(N)
1	2	4	3	2	1			

O(N) Trick

		1				-1		O(1)
1		1	-1			-1		O(1)
1		2	-1		-1	-1		O(1)
1	1	2	-1	-1	-2	-1		O(1)
1	2	4	3	2	1			O(N)

C. Painting

Subtask 3: O(N) Horizontal Solutions

- Using the O(N) trick, we apply it for all rows of the painting and we are done

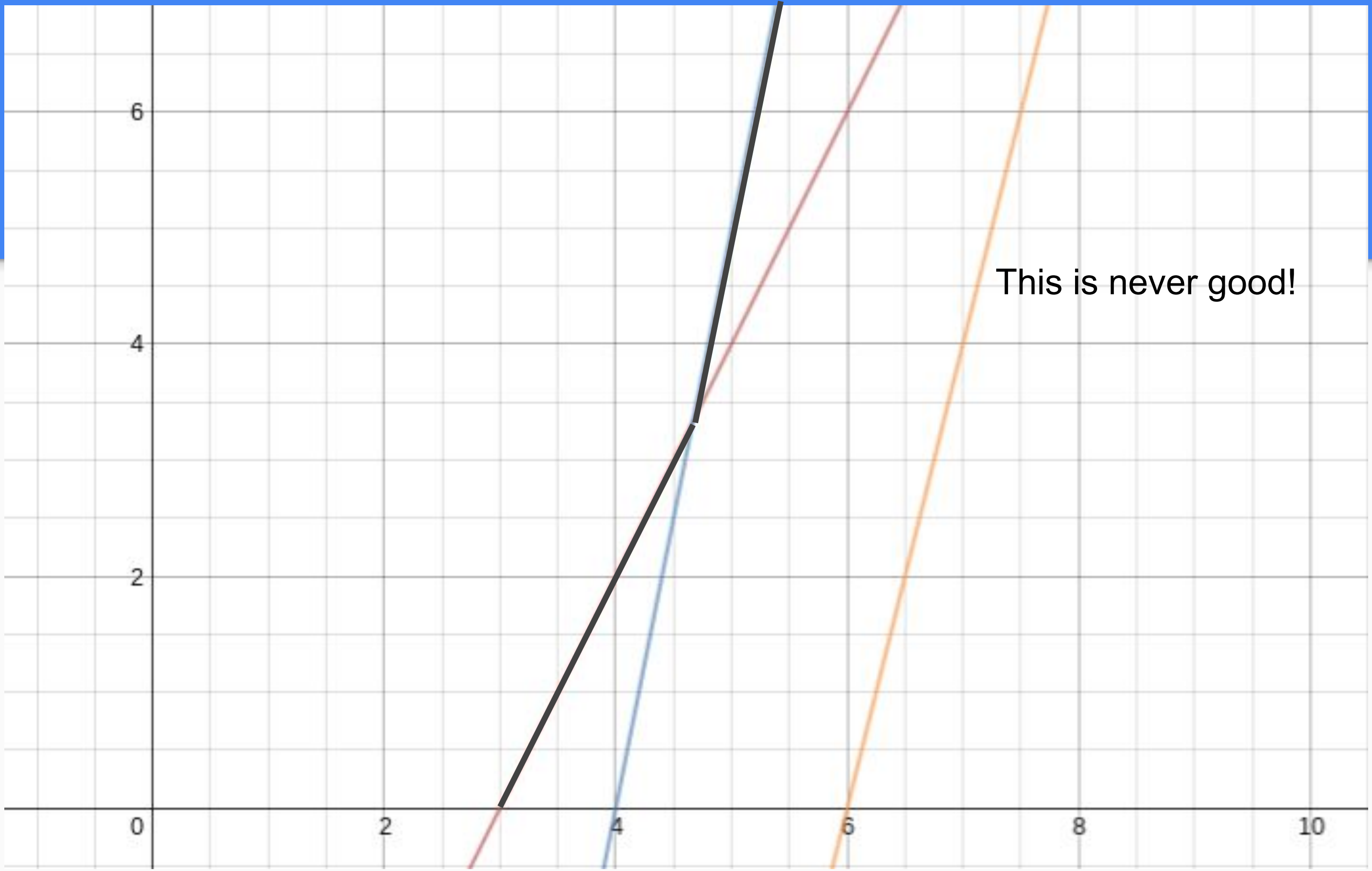
C. Painting

Subtask 4: Full Solution

- It should be pretty obvious by now that we can apply the horizontal trick in the vertical direction (yes we can).
- By summing both horizontal grids and vertical grids, we then get the final painting.

D. Acorn

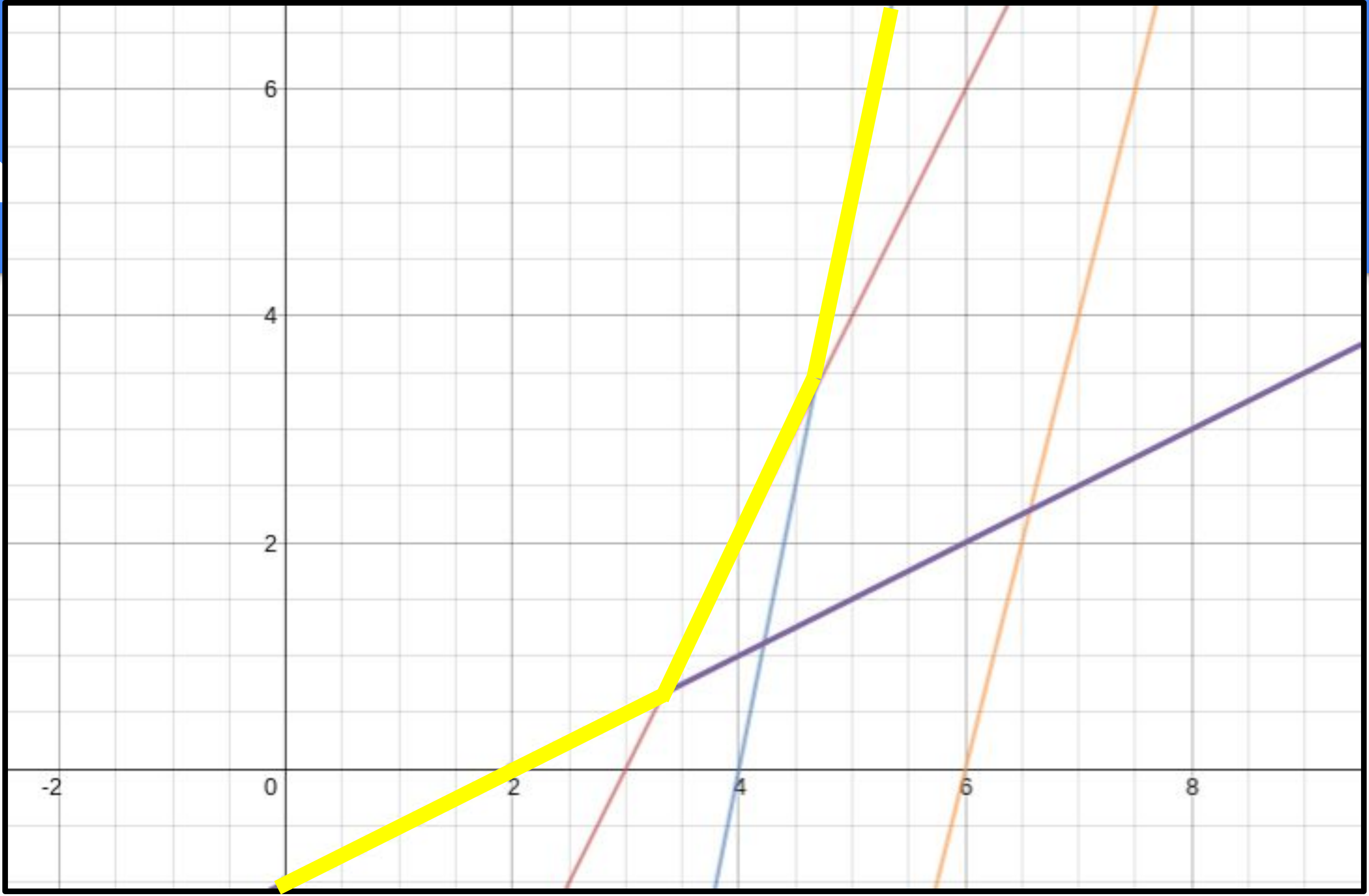
Given N lines of equation $y = b_i x - a_i b_i$, find the number of lines such that for all positive integer x, they will not yield the max value y among other lines.


6
4
2
0
2
4
6
8
10
This is never good!

Convex Hull Trick

1. Build Upper Envelope
2. Remove lines that yield the max only when x is some negative number
3. Remove lines that yield a negative max value
4. Remove lines that doesn’t yield the max in any integer

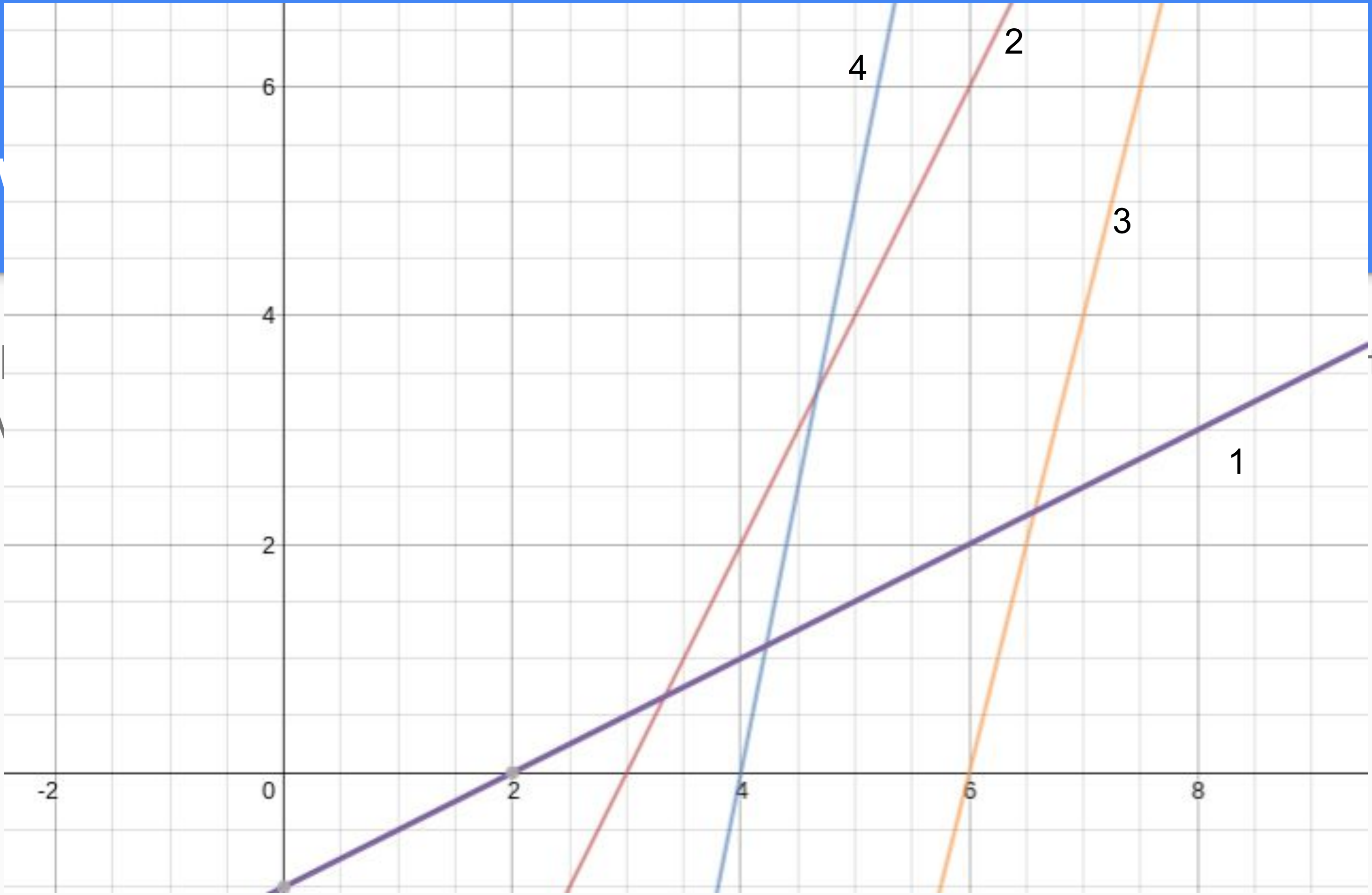
Wh



Ho

Sort t... t,

remo

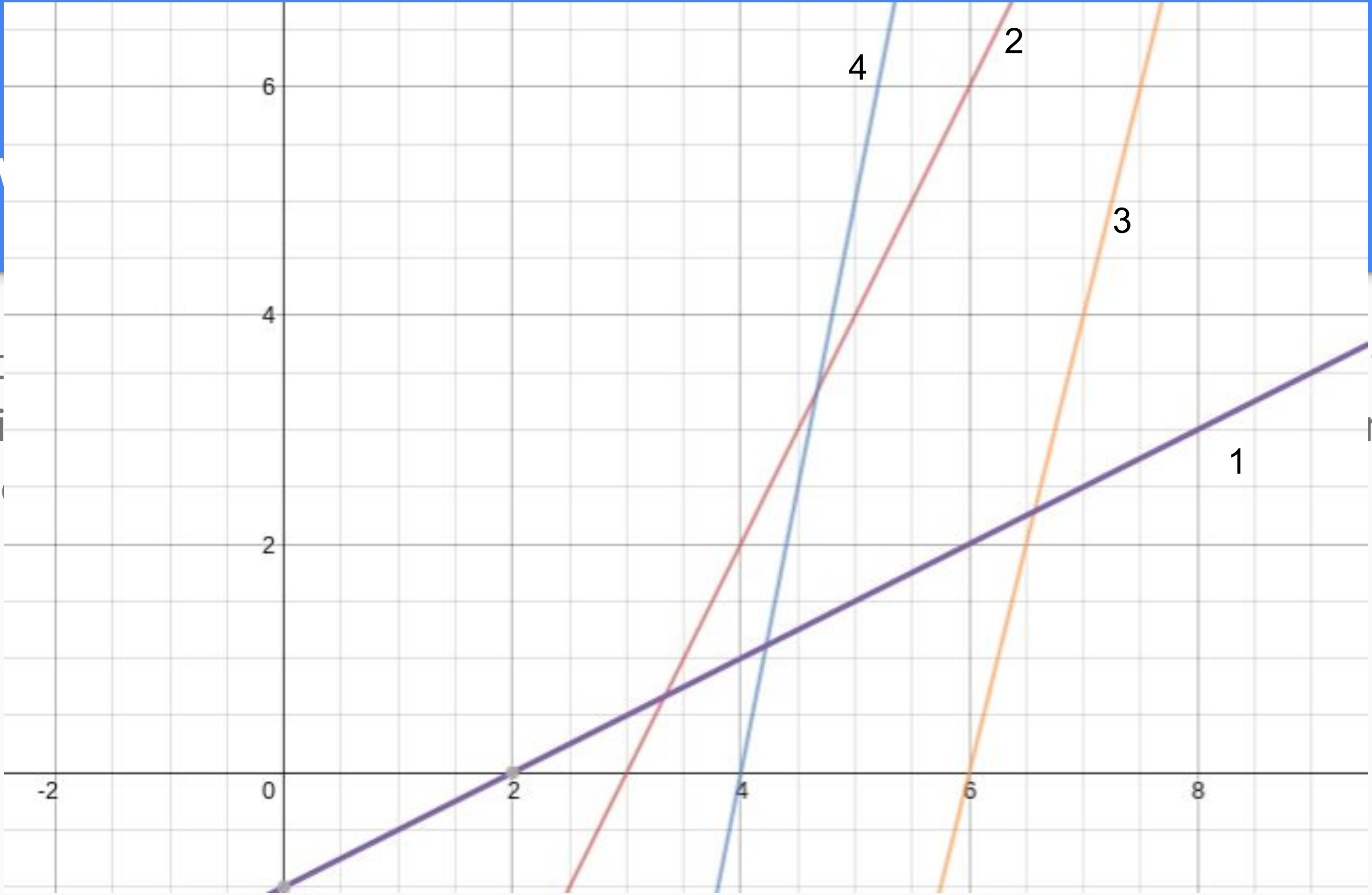


Ho

Insert the

new li ne

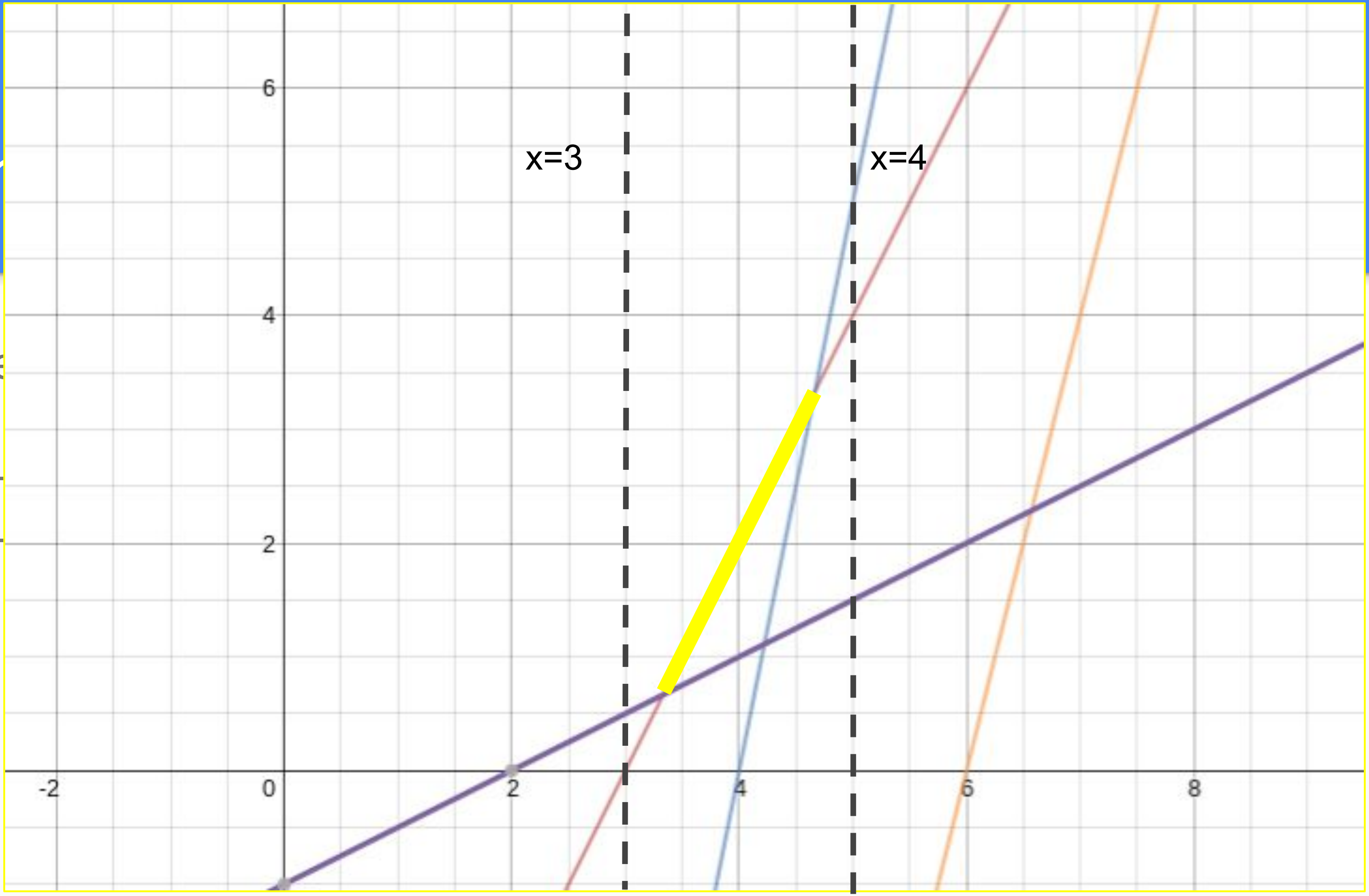
and s



Sor

These

1. L
2. L



Analysis

O(NlgN) - sorting lines

E. Penghulu

Abridged problem statement: The problem is a tree, and each query is querying, when p is the root of the tree, what is the weight of the subtree rooted at q.

Subtask 1: Brute force O(N) per query

For each query, root the tree and run a dfs with the following condition if node x is a leave, subtree weight x = weight x; else subtree weight x = sum of subtree weight of all child of x.
Complexity: O(N) (dfs) per query, total O(NQ)

E. Penghulu

Subtask 2: Precompute the answer. O(N) preprocessing, O(1) per query

Since now, the root would not change, we can actually root the tree at p and run a dfs as in subtask 1. The only changes is we store the value during the dfs so that all the weight of the subtree can be looked up later.
Complexity: O(N + Q)

Subtask 3: This subtask actually act as a hint to solve subtask 4 and we will proceed the explanation at subtask 4.

E. Penghulu

Subtask 4: O(N log N) solution

Assuming that we root the tree at X. For query (p, q), consider 2 case:

- Case 1: p is not in subtree of q.

  We can easily see that weight of q when root = p is equal to weight of q when root = X.

- Case 2: p is in subtree of q.

  This is a little more tricky. Let Q be a child of q such that p is in subtree of Q. Then it is not hard to see that weight of q when root = p is equal to total tree weight - weight of Q when root = X.

- Case 3: p = q. When p = q, just output the total weight.

E. Penghulu

Subtask 4: O(N log N) solution

Now the problem reduce to how to determine if p is in subtree of q and if it is, how to find Q. To solve this efficiently, we need to have a data structure such that, for every node X, we can access X k-th parent efficiently. If we can do that, simply get the (depth p - depth q -1)-th parent of q let it be Y and if the parent of Y = q then it is in the subtree of q. And the answer for query (p, q) is total weight - weight of Y if p is in subtree of q, else the answer is weight of q.

To query efficiently, for each node, we keep track of its 2^n -th parent for n = 0, 1, 2, 3.... Hence, the total memory is O(N log N) and the time to query its k-th parent is O(log k)

F. Town Planning

Problem statement: count number of paths such that when its vertices are removed, exactly ***K*** connected components remain.

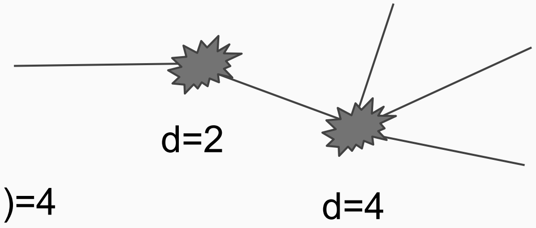
Subtask 1. Naive brute force - $O(N^3)$

For every possible path, remove the vertices and run a DFS to count the number of connected components. There are $O(N^2)$ paths to check, so overall complexity is $O(N^3)$.

Subtask 2. No need for DFS - $O(N^2)$

- Let d_i be the degree of vertex i, that is, the number of edges adjacent to vertex i. For such a path, let the length of path be m, and sum of d_i over all vertices in the path be D.
- Notice that the number of connected components is $D-2*(m-1)$, the number of edges adjacent to the path but not in the path itself.
- Hence just checking each of the N^2 paths and summing the d_i yields an $O(N^2)$ solution.

length = m = 2
number of components = 6-2(1)=4



Subtask 3. Intended solution - $O(N\log^2 N)$

- Suppose we have a subtree and a root ***u***. We count the number of such paths in the subtree passing through ***u***.
- With one DFS, for all ***i*** in the subtree find
  - D_i - total degree of a path from ***i*** to ***u***,
  - $dist_i$ - distance from ***i*** to ***u***.
- Let $a_i = D_i - 2*dist_i$ and sort them. This takes $O(N\log N)$.
- For such a path through ***u*** with endpoints ***i*** and ***j***, since number of connected components is $D-2*(m-1)$, we must have $(D_i+D_j-d_u)-2(dist_i+dist_j)=K$, which simplifies to $a_i+a_j=K+d_u$.

Subtask 3. Intended solution - $O(N\log^2 N)$

- For each ***i*** we can figure out how many possible ***j*** there are by a binary search on the $\boldsymbol{a_i}$. We might also choose ***j*** in the same subtree as ***i***. To deal with this, store arrays of $\boldsymbol{a_i}$ in each subtree and subtract the overlaps.

E.g. ***a[]*** = {1, 4, 5, 5, 9, 9}, and ***K*** = 4, $\boldsymbol{d_u}$=6. Then we need $\boldsymbol{a_i+a_j=10}$.

From ***a[i]=1***, there are two ***9***.

Subtask 3. Intended solution - $O(N\log^2 N)$

- Now remove ***u*** - this splits the graph into more subtrees, so we can recursively use the above solution. To minimize the number of recursion levels, use **centroid decomposition**.
- The **centroid** of a tree with ***sz*** vertices is the vertex such that if the tree is rooted at that vertex, every subtree has size at most ***(½)*sz***.
- Run DFS to store the sizes of the subtrees at each vertex. Then run another DFS, going into the child node with size **> *(½)*sz***. If no such child exists, we are at the centroid. This takes O(size of tree).

Subtask 3. Intended solution - $O(N\log^2 N)$

- At each iteration, looking for centroid takes $O(N)$, and accounting for paths passing through the centroid takes $O(N\log N)$.
- In each iteration, size of each subtree is halved. Hence the number of iterations is $O(\log N)$, so overall complexity is **$O(N\log^2 N)$**.

Problem Credits

A. Cable Car - Christopher Boo
B. Relay Race - Christopher Boo
C. Painting - Christopher Boo
D. Acorn - Christopher Boo
E. Penghulu - Lim Yun Kai
F. Town Planning - Justin Lim

Contest Statistics

Malaysian Computing Olympiad 2016

Contest Statistics

Problem breakdown:

Problem	A	B	C	D	E	F
Mean	16.25	23.75	21.87	3.75	5.94	0
Number of ACs	4	6	2	0	0	0
Max Attained	80/80	100/100	120/120	40/200	150/300	0/300

Contest Statistics

- Number of submissions: 816
- Most solved problem: Relay Race (6 A.C.s)
- First AC: Ang Yee Chin, Cable Car (20 min)
- Last AC: Yeoh Zi Song, Cable Car (2 hr 59 min)
- Most number of submissions: Christopher Boo (103)
- Most number of submissions (contestant): Yeoh Zi Song (60)

Score Distribution