# Day 3: Problem Analysis

Niyaz Nigmatullin

September 10, 2015

# Problem A. Bipartite graph

## Statement

- Given the graph, find if it can be colored in two colors, and output coloring if it exists

## Solution

- Assume that your graph is connected
- Then if you color one vertex in one of the colors, all other vertices are colored in a unique way

# Problem A. Bipartite graph

## Solution

- So one can make DFS or BFS, when getting into new vertex, just color it in opposite color
- Check for every edge, that its ends are colored in distinct colors
- Solve for each connected component independently
- Time complexity: $O(E)$

# Problem A. Bipartite graph

```
for (int v = 0; v < n; v++) {
    if (!visited[v]) {
        bfs(v);
    }
}
```

- BFS can be called $n$ times
- Don't fill any arrays in BFS
- BFS should make "number of visited vertices" operations

# Problem B. Number of inversions

## Statement

- Given array of distinct elements, find number of pairs $(i, j)$ such that $a[i] > a[j]$

## Solution 1

- Use mergesort — divide-and-conquer
- Divide array in two parts
- Count inversions in parts recursively
- Count inversions: $i < m$ and $j \geq m$
- Merge two sorted parts: can count for elements on right, number of sought on left
- $O(n \log n)$ solution

# Problem B. Number of inversions

## Solution 2

- Use interval tree or binary indexed tree
- Keep track of $f[x]$ — how many indices $i$ such that $a[i] = x$, was there for $i < j$
- Now the number of elements, which form an inversion with $a[j]$ is $f[a[j]+1] + f[a[j]+2] + \ldots + f[M]$, where $M$ is maximal element in array
- So the solution is just to store $f$ in interval tree, to change the element and find the sum fastly
- $O(n \log n)$ solution

# Problem C. Three lines

## Statement

- Given points in 2D
- Is it possible to choose three lines parallel to axes, to cover all the points?

## Solution

- We have two cases here:
  - All the lines are parallel
  - Two of the three lines parallel

# Problem C. Three lines

## Case 1: all lines are parallel

- Count number of different coordinates
- If and only if it doesn't exceed three, then can do three parallel lines

## Case 2: two of three lines are parallel

- Suppose we want one vertical and two horizontal lines
- Try all $x$ coordinates for the vertical line
- The number of different $y$ coordinates for the lines, not covered by vertical line shouldn't exceed 2

# Problem C. Three lines

## Case 2: two of three lines are parallel

- Keep track of number of $y$ coordinates
- When trying vertical line, remove all points covered by it and count number of $y$ coordinates
- Since every point is removed at most once, the algorithm works in $O(n)$ removals and additions
- Use `std::map` or `java.util.HashMap`
- Two vertical and one horizontal are done in the same way

# Problem D. Islands

## Statement

- You are given discrete landscape by its heights
- The level of water may cover some of the parts of landscape
- Count the maximum number of islands, that could be formed

# Problem D. Islands

## Solution

- Every part can be covered or not
- Describe the state as binary string, 1 is covered, and 0 is ground
- 011001010 — has four islands
- Add two covered parts as the first and last parts: 10110010101
- Number of islands is: the number of equal consecutive parts divided by two
- Number of equal consecutive parts: the number of distinct neighboring elements

# Problem D. Islands

## Solution

- Start to simulate process
- Cover the parts starting with the lowest ones
- Keep track of distinct neighboring elements
- Find the number of islands after each change
- Cover the parts with the same length simultaneously
- Time complexity is $O(n \log n)$ to sort parts by their height

# Problem E. Baleshare

## Statement

- Given a multiset of numbers
- Split the multiset into three, such that the maximum sum of the elements of them is minimized

## Solution

- Use dynamic programming approach
- $F[i][s][t]$ — is there a way to split first $i$ elements in array into three sets, such that the sum of elements of the first set is $s$, and the second is $t$

# Problem E. Baleshare

## Solution

- To compute: either you take it to the first set, second set or the third set
- $F[i][s][t] = F[i-1][s][t]$ `or` $F[i-1][s-a[i]][t]$ `or` $F[i-1][s][t-a[i]]$
- $F[i]$ only depends of $F[i-1]$, so you can store only this two arrays
- The number of states is $2000 \cdot 2000 \cdot 40 = 160 \cdot 10^6$, and $2000 \cdot 2000 \cdot 2 = 8 \cdot 10^6$ elements of array needed

# Problem F. Mountain Climbing

## Statement

- Formally you have $n$ jobs
- And two machines
- Each job must be made on first machine and then, after its finished on the first, on second
- Each machine can do only one job at a time

# Problem F. Mountain Climbing

## Statement

- Jobs are done in the same order on both machines
- Sort the jobs by comparator: $\min(a_i, b_j) < \min(a_j, b_i)$
- This problem is also known as $F_2||C_{max}$ as one of the scheduling theory problems
- The proof of above statements can be found in the book of Peter Brucker "Scheduling Algorithms"

# Problem G. Grass Planting

## Statement

- Given a tree, support two types of queries
- First type: add one to value on each edge on some path
- Second type: output the value on the given edge

## Solution 1

- Heavy-light decomposition is a straightforward solution
- Time complexity: $O(n \log^2 n)$

# Problem G. Grass Planting

## Solution 2

- Let $r$ be the root of the rooted tree
- Let's reduce `add(v, u, x)` to three calls `add(v, r, x)`, `add(u, r, x)`, `add(lca(v, u), r, -2x)`
- Now for change query we add at some "prefix" of a tree: path from $v$ to the root, call it `addp(v, x)`
- What `addp(v, x)` calls change the value of vertex $u$?
- `addp(v, x)` changes $u$, if and only if $u$ is ancestor of $v$

# Problem G. Grass Planting

## Solution 2

- Let's keep track of value `g[v]` — which is the total value added to path from $v$ to $r$, considering only calls `addp(v, x)`
- The value in vertex $u$ is the sum of `g[v]` for all $v$ — descendants of vertex $u$
- So, problem is reduced to the following queries:
  - Make `g[v] += x`
  - Get sum of `g[v]` in subtree rooted at $u$

# Problem G. Grass Planting

```
void dfs(int v, int pv) {
    enter[v] = T++;
    for (Edge e : edges[v]) {
        if (e.to == pv)
            continue;
        dfs(e.to, v);
    }
    exit[v] = T;
}
```

- Above code calculates the time of entrance to each vertex, and the time of exit

# Problem G. Grass Planting

- It can be seen that, vertex $u$ is ancestor of vertex $v$ if and only if $[enter_v, exit_v) \subset [enter_u, exit_u)$
- Or even easier to check: $enter_v \in [enter_u, exit_u)$
- Every subtree is interval $[enter_u, exit_u)$
- So to get the sum of values in subtree one can do it by ordering vertices by their $enter_v$, and suming up the interval $[enter_u, exit_u)$
- Time complexity: LCA finding, interval tree or BIT for queries makes it up to $O(n \log n)$

# Problem H. Simplifying the Farm

## Statement

- Weighted undirected graph is given
- There are at most 3 edges of each weight
- Find weight of minimal spanning tree
- Plus find the number of spanning trees with minimum weight

# Problem H. Simplifying the Farm

## Statement

- Use Kruskal algorithm to find minimum spanning tree
- The only ambiguity in kruskal algorithm is when there are equal weight edges
- Since there are only three edges, try all possible permutations of them
- And count how many different sets of these three edges could be added
- Multiply the answer by this number
- Time complexity: $O(E \log V)$ as for Kruskal algorithm