USACO January Bronze/Silver Post-Contest Explanations

Srinidhi Krishnamurthy

February 2018

1 Bronze: Out of Place

1.1 Full Problem Statement

Feeling ambitious, Farmer John plans to attempt something that never seems to go quite right: he wants to take a photograph of his entire herd of cows. To make the photograph look nice, he wants the cows to line up in a single row from shortest to tallest. Unfortunately, right after he has the cows line up this way, Bessie the cow, always the troublemaker, steps out of line and re-inserts herself at some other location in the lineup!

Farmer John would like to swap pairs of cows so the entire herd is again lined up properly. Please help him determine the minimum number of swaps he needs to make between pairs of cows in order to achieve this goal.

1.2 Input Statement

The first line of input contains N ($2 \leq N \leq 100$). The next N lines describe the heights of the cows as they are lined up after Bessie makes her move. Each cow height is an integer in the range $1 \ldots 1,000,000$. Cows may have the same height. Please output the minimum number of times Farmer John needs to swap pairs of cows in order to achieve a proper ordering. Swaps do not necessarily need to involve adjacent cows in the ordering.

Please output the minimum number of times Farmer John needs to swap pairs of cows in order to achieve a proper ordering. Swaps do not necessarily need to involve adjacent cows in the ordering.

1.3 Solution Approach

So usually in Bronze problems, we don't have to worry about the time complexity. In this case, all we have to do is work through the sample cases given to understand how to arrive at the algorithm. In this case, we just setup two arrays, sort one of them, and compare them.

1.4 USACO's Solution

```
import java.io.*;
import java.util.*;
public class outofplace {
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader("outofplace.in"));
    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter("outofplace.out")));

    int n = Integer.parseInt(br.readLine());
    int[] height = new int[n];
    int[] sorted = new int[n];
    for(int i = 0; i < n; i++) {
      height[i] = Integer.parseInt(br.readLine());
      sorted[i] = height[i];
    }
    Arrays.sort(sorted);
```

```
    int swaps = -1;
    for(int a = 0; a < n; a++) {
      if(sorted[a] != height[a]) {
        swaps++;
      }
    }
    swaps = Math.max(0, swaps);
    pw.println(swaps);
    pw.close();
  }

}
```

Listing 1: "Java solution by USACO"

2 Silver: MooTube

2.1 Full Problem Statement

In his spare time, Farmer John has created a new video-sharing service, which he names MooTube. On MooTube, Farmer John's cows can record, share, and discover many amusing videos. His cows already have posted N videos ($1 \leq N \leq 5000$), conveniently numbered $1 \ldots N$. However, FJ can't quite figure out how to help his cows find new videos they might like. FJ wants to create a list of "suggested videos" for every MooTube video. This way, cows will be recommended the videos most relevant to the ones they already watch.

FJ devises a metric of "relevance," which determines, as the name suggests, how relevant two videos are to each other. He picks $N-1$ pairs of videos and manually computes their pairwise relevance. Then, FJ visualizes his videos as a network, where each video is a node and the $N-1$ pairs of videos he manually considered are connected. Conveniently, FJ has picked his $N-1$ pairs so that any video can be reached from any other video along a path of connections in exactly one way. FJ decides that the relevance of any pair of videos should be defined as the minimum relevance of any connection along this path.

Farmer John wants to pick a value K so that next to any given MooTube video, all other videos with relevance at least K to that video will be suggested. However, FJ is worried that too many videos will be suggested to his cows, which could distract them from milk production! Therefore, he wants to carefully set an appropriate value of K. Farmer John would like your help answering a number of questions about the suggested videos for certain values of K.

2.2 Input Statement

The first line of input contains N and Q ($1 \leq Q \leq 5000$). The next $N-1$ lines each describe a pair of videos FJ manually compares. Each line includes three integers p_i, q_i, and r_i ($1 \leq p_i, q_i \leq N, 1 \leq r_i \leq 1,000,000,000$), indicating that videos p_i and q_i are connected with relevance r_i.

The next Q lines describe Farmer John's Q questions. Each line contains two integers, k_i and v_i ($1 \leq k_i \leq 1,000,000,000, 1 \leq v_i \leq N$), indicating that FJ's ith question asks how many videos will be suggested to viewers of video v_i if $K = k_i$.

Output Q lines. On line i, output the answer to FJ's ith question.

2.3 Solution Approach

So the best way to approach this question is to draw out the sample and formulate the problem more simply. When we do this, we find that we have an weighted graph. We want to find the minimum weight over all the edges on the path between two nodes. We also want to answer more than one query, so we have to keep complexity (and algorithms) in mind before we write the solution.

To solve this, we can start with a BFS from the source vertex. Along the way, we ignore any edges that have weight less than the destination weight. While doing this, we can count how many other vertices we have visited.

2.4 USACO's Solution

```java
import java.io.*;
import java.util.*;
public class mootube {
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader("mootube.in"));
    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter("mootube.out")));
    StringTokenizer st = new StringTokenizer(br.readLine());
    int n = Integer.parseInt(st.nextToken());
    int q = Integer.parseInt(st.nextToken());
    LinkedList<Edge>[] edges = new LinkedList[n];
    for(int i = 0; i < n; i++) {
      edges[i] = new LinkedList<Edge>();
    }
    for(int a = 1; a < n; a++) {
      st = new StringTokenizer(br.readLine());
      int x = Integer.parseInt(st.nextToken())-1;
      int y = Integer.parseInt(st.nextToken())-1;
      int w = Integer.parseInt(st.nextToken());
      edges[x].add(new Edge(y, w));
      edges[y].add(new Edge(x, w));
    }
    for(int query = 0; query < q; query++) {
      st = new StringTokenizer(br.readLine());
      int threshold = Integer.parseInt(st.nextToken());
      int start = Integer.parseInt(st.nextToken())-1;
      int ret = 0;
      LinkedList<Integer> queue = new LinkedList<Integer>();
      queue.add(start);
      boolean[] seen = new boolean[n];
      seen[start] = true;
      while(!queue.isEmpty()) {
        int curr = queue.removeFirst();
        for(Edge out: edges[curr]) {
          if(!seen[out.d] && out.w >= threshold) {
            seen[out.d] = true;
            queue.add(out.d);
            ret++;
          }
        }
      }
      pw.println(ret);
    }
    pw.close();
  }

  static class Edge {
    public int d, w;
    public Edge(int a, int b) {
      d=a;
      w=b;
    }
  }

}
```

Listing 2: "Java solution by USACO"