USACO FEB15 Problem 'hopscotch' Analysis

by Nick Wu

The DP that would have worked for the Silver version of this problem is too slow here. You can compute prefix sums to remove a linear factor, but $O(RC^2)$ is also too slow.

If we look at the DP more carefully, we see that for a given square, we only care about which squares have the same number as the given square. Therefore, if we maintain binary indexed trees (BITs) over columns for every single color as well as the total number of paths possible across all colors, we can efficiently determine the number of paths that end in a given square in logarithmic time. This gives us the desired $O(RC\log C)$ bound.

There are two implementation notes that we need to be careful about. Firstly, although we can loop over rows from top to bottom, we actually need to loop over columns from right-to-left so that we don't double-count paths from the BIT that stores the count of all paths across all columns. Secondly, using $O(C^3)$ memory - $O(C)$ for each of $O(C^2)$ possible colors, will be too slow. However, we don't need to allocate $O(C)$ memory for each color. If a color doesn't appear at all, for example, we don't need to allocate memory for a BIT for that color. More specifically, for a given color, if it appears at k indices, then we only need to keep track of how the count changes at those k different values, as opposed to all C. One way of accomplishing this is by storing a parallel array of the columns where the color appears and doing a binary search at the beginning to figure out the indices at which to start updating and querying.

Here is my Java code implementing this algorithm.

```
import java.io.*;
import java.util.*;
public class barnGold {
  static char[][] grid;
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader("barnjump.in"));
    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter("barnjump.out")));
    StringTokenizer st = new StringTokenizer(br.readLine());
    int r = Integer.parseInt(st.nextToken());
    int c = Integer.parseInt(st.nextToken());
    int colors = Integer.parseInt(st.nextToken());
    int[][] grid = new int[r][c];
    ArrayList<Integer>[] columns = new ArrayList[colors+1];
    for(int i = 1; i < columns.length; i++)
      columns[i] = new ArrayList<Integer>();
    for(int i = 0; i < r; i++) {
      st = new StringTokenizer(br.readLine());
      for(int j = 0; j < c; j++)
        grid[i][j] = Integer.parseInt(st.nextToken());
    }
    for(int j = 0; j < c; j++)
      for(int i = 0; i < r; i++) {
        int color = grid[i][j];
        if(!columns[color].isEmpty() && columns[color].get(columns[color].size()-1) == j)
continue;
        columns[color].add(j);
      }
    BIT[] bits = new BIT[colors+1];
    for(int i = 1; i < bits.length; i++) {
      if(columns[i].size() > 0)
        bits[i] = new BIT(columns[i]);
    }
    ArrayList<Integer> gen = new ArrayList<Integer>();
```

```
    for(int i = 0; i < c; i++)
      gen.add(i);
    BIT full = new BIT(gen);
    full.update(0, 1);
    bits[grid[0][0]].update(0, 1);
    for(int i = 1; i < r-1; i++) {
      for(int j = c-2; j > 0; j--) {
        int val = full.query(j-1) - bits[grid[i][j]].query(j-1);
        if(val < 0) val += MOD;
        full.update(j, val);
        bits[grid[i][j]].update(j, val);
      }
    }
    int ret = full.query(c-2) - bits[grid[r-1][c-1]].query(c-2);
    if(ret < 0) ret += MOD;
    pw.println(ret);
    pw.close();
  }

  static final int MOD = 1000000007;

  static class BIT {
    public int[] tree;
    public int[] indices;
    public BIT(ArrayList<Integer> set) {
      indices = new int[set.size()+2];
      tree = new int[indices.length];
      indices[0] = -1;
      int index = 1;
      for(int out: set) {
        indices[index++] = out;
      }
      indices[indices.length-1] = Integer.MAX_VALUE;
    }
    public void update(int index, int val) {
      int actual = Arrays.binarySearch(indices, index);
      while(actual < indices.length) {
        tree[actual] += val;
        if(tree[actual] >= MOD) tree[actual] -= MOD;
        actual += actual & -actual;
      }
    }
    public int query(int index) {
      int left = 0;
      int right = indices.length-1;
      while(left != right) {
        int mid = (left+right+1)/2;
        if(indices[mid] > index)
          right = mid-1;
        else
          left = mid;
      }
      int ret = 0;
      while(left > 0) {
        ret += tree[left];
        if(ret >= MOD) ret -= MOD;
        left -= left & -left;
      }
      return ret;
    }
  }

}
```