USACO OPEN15 Problem 'palpath' Analysis

by Nick Wu

This is a DP problem where we iteratively count the number of palindromes that we can build from the middle.

Let $f(a,r_1,r_2)$ be the number of palindromic strings that we can build of length $2a+1$, where the start of the string is on row r_1, the end of the string is on row r_2, and the middle of the string is on the diagonal of the grid that goes from the top-right to the bottom-left of the grid. We initialize $f(0,i,i)=1$ for all possible rows. Because of the constraints of the DP state, the beginning and ending squares are uniquely determined by their row. Therefore, $f(a,r_1,r_2)$ affects at most four other quantities: $f(a+1,r_1,r_2)$, $f(a+1,r_1\ \ 1,r_2)$, $f(a+1,r_1,r_2+1)$, and $f(a+1,r_1\ \ 1,r_2+1)$. This gives an $O(N^3)$ algorithm which can be implemented in $O(N^2)$ memory because you only need to keep track of $f(a,r_1,r_2)$ and $f(a+1,r_1,r_2)$ concurrently over all possible pairs (r_1,r_2). Here is my code.

```
import java.io.*;
import java.util.*;
public class palpathG {
  static int n;
  public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader("palpath.in"));
    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter("palpath.out")));
    n = Integer.parseInt(br.readLine());
    char[][] grid = new char[n][n];
    for(int i = 0; i < n; i++) {
      String s = br.readLine();
      for(int j = 0; j < n; j++) {
        grid[i][j] = s.charAt(j);
      }
    }
    long[][] dp = new long[n][n];
    for(int i = 0; i < n; i++) {
      dp[i][i] = 1;
    }
    final long MOD = 1000000007;
    for(int num = n-1; num >= 1; num--) {
      long[][] next = new long[n][n];
      for(int a = 0; a < n; a++) {
        int rowA = a;
        int colA = (num-1-a);
        if(colA < 0) continue;
        for(int b = 0; b < n; b++) {
          int rowB = b;
          int colB = 2*n-num-rowB-1;
          if(colB >= n) continue;
          if(grid[rowA][colA] != grid[rowB][colB]) continue;
          next[rowA][rowB] += dp[rowA][rowB];
          if(rowA+1 < n) next[rowA][rowB] += dp[rowA+1][rowB];
          if(rowB-1 >= 0) next[rowA][rowB] += dp[rowA][rowB-1];
          if(rowA+1 < n && rowB-1 >= 0) next[rowA][rowB] += dp[rowA+1][rowB-1];
          next[rowA][rowB] %= MOD;
        }
      }
      dp = next;
    }
    pw.println(dp[0][n-1]);
    pw.close();
  }
}
```