University of Virginia ICPC Notebook

Contents

1 World Final 2004

1.1 C Image is Everything

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

/*    to fold
5|          front, left, back, right, top, bottom
2|1|4|3|      1     2     3     4      5     6
     |6|
*/

const int MaxN=11; // at most size of 10, but use 11 to avoid seeing the additional side
char f[6][MaxN][MaxN]; // color view read from data: [6]each view [][] square matrix
int p[6][MaxN][MaxN];  // number of cubic(s) removed: [6]each view [][] square matrix
int g[MaxN][MaxN][MaxN]; // 3D matrix: 0 as not exist, -1 no info yet, char A-Z (1-27) as color
int n; //cubic size

bool update(int m) {
   bool upd=false;
   for(int i=0; i<n; ++i) for(int j=0; j<n; ++j){ //analyze each pixel in current view
      int &t=p[m][i][j]; //the number of cubic removed at this pixel
      char ch=f[m][i][j]; //the color at this pixel in the original view
      if(t==n) continue; //already removed all cubics at this pixel
      int *val; //value of actual cubic at this pixel to analyze
      switch (m) {
          case 0: // front
            val = &(g[n-t-1][j][i]);
            break;
          case 1: // left
            val = &(g[j][t][i]);
            break;
          case 2: // back
            val = &(g[t][n-j-1][i]);
            break;
          case 3: // right
            val = &(g[n-j-1][n-t-1][i]);
            break;
          case 4: // top
            val = &(g[i][j][t]);
            break;
          case 5: // bottom
            val = &(g[n-i-1][j][n-t-1]);
            break;
          default:
            break;
      }
      if (*val==0){ ++t; upd=true;}
        //the cubic here does not exist
      else if(ch=='.'){ *val=0; ++t; upd=true; }
        //update the cubic here as not exist
      else if(*val==-1){ *val=ch-'A'+1; upd=true;}
        //if the cubic color is not analzed, use the color on the other side
      else if(*val!=ch-'A'+1){ *val=0; ++t; upd=true; }
        //if different from the cubic color from the other side, update the cubic here as not exist,
   }
   return upd;
}

int main() {
  while(true) {
    scanf("%d", &n); // read size of cubic
    if (n==0) break;
    for (int i=0; i<n; ++i) scanf(" %s %s %s %s %s %s", f[0][i], f[1][i], f[2][i], f[3][i], f[4][i], f
          [5][i]); // read view
    for (int i=0; i<6; ++i) fill(p[i][0], p[i][n], 0); //initize all view as non removed
    for (int i=0; i<n; ++i) fill(g[i][0], g[i][n], -1); //initize all cubics not exist
    // fill(g[0][0], g[n][0], -1);
    while (true) {
       bool quit=true;
       for (int i=0; i<6; i++) if(update(i)) quit=false;
       if(quit) break;
    }
    int ans=n*n*n; // init weight for perfect cubic
    for(int i=0; i<n; ++i) for(int j=0; j<n; ++j) for(int k=0; k<n; ++k)
      if(g[i][j][k]==0) --ans; // if the cubic at position[i][j][k] does not exit, remove this weight
    printf("Maximum weight: %d gram(s)\n", ans);
  }
  return 0;
}
```

1.2 E Intersecting Dates

```
#include <stdio.h>
#include <vector>
#include <stdlib.h>
#include <unordered_map>
using namespace std;

// there are at most 147000 different days between 1700 and 2100
#define MAX_DATE 147000

// keep track of the case number (needed for output)
static int caseNum = 1;

// records which dates are valid dates for our output
bool dates[MAX_DATE];

// "itod" (Index to Date): given a specific index for "dates",
// returns the associated calendar date for this index
int itod[MAX_DATE];

// "dtoi" (Date to Index): given a date in calendar,
// returns the index it is represented in our "dates" array
// **This is exactly the reverse of "itod";
// it has to be stored as unordered_map,
// since 21001231 exceeds the max memory size for an array
unordered_map<int, int> dtoi;

// given an integer between 17000101 and 21001231,
// returns whether this integer is a valid date in calendar
bool isCorrectDate(int date) {
    int y = date / 10000, m = (date%10000) / 100, d = date % 100;
    if (m == 0 || m > 12) return false;
    if (d == 0 || d > 31) return false;
    if (d==31 && (m==4 || m==6 || m==9 || m==11)) return false;
    if (m == 2) {
        if (y%4 == 0 && (y%100 != 0 || y%400 == 0)) return d <= 29;
        else return d <= 28;
    }
    return true;
}

// pre-generate "itod" and "dtoi" for all possible dates
void genMaps() {
    int ind = 0;
    for (int i = 17000101; i <= 21001231; i++) {
        if (!isCorrectDate(i)) continue;
        itod[ind++] = i;
        dtoi[i] = ind-1;
    }
}

// given two dates "start" and "end" (possibly the same),
// prints the output in the correct format to stdout
void printOutput(int start, int end) {
    int y1 = start / 10000, m1 = (start%10000) / 100, d1 = start%100;
    int y2 = end / 10000, m2 = (end%10000) / 100, d2 = end%100;
    if (start == end) // required to print only one date
        printf("    %d/%d/%d\n", m1, d1, y1);
    else
        printf("    %d/%d/%d to %d/%d/%d\n", m1, d1, y1, m2, d2, y2);
}

// main part of the solution
void solve(int b, int a) {
    printf("Case %d:\n", caseNum);

    // reads all dates whose data have been previously retrieved
    // and store them in a vector (we need these data later)
    vector<int> dates1, dates2;
    for (int i = 0; i < b; i++) {
        int d1, d2; scanf("%d %d\n", &d1, &d2);
        dates1.push_back(d1);
```

```
            dates2.push_back(d2);
        }

        // reads all remaining dates whose data are to be retrieved
        for (int i = 0; i < a; i++) {
            int d1, d2; scanf("%d %d\n", &d1, &d2);
            // these dates are required for output; set them to true
            fill(dates+dtoi[d1], dates+dtoi[d2]+1, true);
        }

        for (int i = 0; i < b; i++) {
            // these dates have been previously retrieved (no longer needed)
            fill(dates+dtoi[dates1[i]], dates+dtoi[dates2[i]]+1, false);
        }

        // iterate over all possible dates to get desired time intervals
        int i = 0;
        bool need = false;
        while (i <= dtoi[21001231]) {
            if (dates[i]) {
                int start = itod[i];
                while (dates[++i]); // scan for the end of this interval
                int end = itod[i-1];
                printOutput(start, end);

                // the "no quotes" message should no longer be displayed
                need = true;
            } else
                i++;
        }

        // no dates are required, print the corresponding message
        if (!need) printf("    No additional quotes are required.\n");
    }

    int main() {
        genMaps();

        bool need = false;
        for (;;) {
            // the array "dates" need to be initialized as false everywhere
            fill(dates, dates+MAX_DATE, false);

            // read input
            int b, a; scanf("%d %d\n", &b, &a);

            // check if we reach the end of test file
            if (b == 0 && a == 0) break;

            // keep track of line feeds between cases
            if (!need) need = true;
            else printf("\n");

            solve(b, a);
            caseNum++;
        }

        return 0;
    }
```

1.3 H Tree-lined Street

```
    #include <stdio.h>
    #include <math.h>
    #include <iostream>
    #include <algorithm>
    using namespace std;
    struct line {                //line structure with two points (x1,y1) (x2,y2)
      int x1,y1,x2,y2;
    };
```

```
int cases,n,ans,s,i,j,k;    //global variables
// cases : case number    n : number of lines   ans : output
// s : number of intersections on one certain line    i, j, k: temporary variables

line a[110];                //record lines
double b[200];              //record each intersection on one line by its distance to the origin
double len;
bool check(int x1,int y1,int x2,int y2,int x3,int y3) {  //check if p(x1,y1)(x2,y2) is at the clock-
      wise side of p(x1,y1)(x3,y3)                                                //used for
      determining if two lines intersect
   long long t1,t2;
   t1=(long long)x1*(long long)y2-(long long)x2*(long long)y1;
   t2=(long long)x1*(long long)y3-(long long)x3*(long long)y1;
   return((t1>=0&&t2<=0)||(t1<=0&&t2>=0));
}

bool intersect(line A, line B) {                        //check if two points are on the different
      side of one line and vice versa, if both so they intersect
  return (check(A.x2-A.x1,A.y2-A.y1,B.x1-A.x1,B.y1-A.y1, B.x2-A.x1,B.y2-A.y1)
  && check(B.x2-B.x1,B.y2-B.y1,A.x1-B.x1,A.y1-B.y1, A.x2-B.x1,A.y2-B.y1));
}

double dis(double x,double y) { //distance between one node to the origin
   return sqrt(x*x+y*y);
}

void cross(line A,line B) { //for every two lines determine if they intersect, if so update b and s
   double A1,B1,C1,A2,B2,C2,x,y;
   A1=A.y2-A.y1;B1=A.x1-A.x2;C1 = -(A.x1*A1 + A.y1*B1);
   A2=B.y2-B.y1; B2=B.x1-B.x2; C2 =-(B.x1*A2+B.y1*B2);
   x=-(C1*B2-C2*B1) / (A1*B2-A2*B1);
   y=-(A1*C2-A2*C1) / (A1*B2-A2*B1);
   double l = dis(x-A.x1,y-A.y1);
   if (isfinite(l) && l < dis(A.x2-A.x1,A.y2-A.y1) ) {b[s] = l;}
}

bool init() { //initiate and input data to the global variable, if get 0 end
   int i;
   scanf("%d",&n);
   if(n==0)
   return false;
   for(i=1;i<=n;i++)
    scanf("%d %d %d %d", &a[i].x1,&a[i].y1,&a[i].x2,&a[i].y2);
   return true;
}

int main() {
   cases=0;
   while(init()) {
    ans=0;
    for(i=1;i<=n;i++){
      s = 2;
      b[1]=-25; b[2]=dis(a[i].x2-a[i].x1,a[i].y2-a[i].y1)+25;   //mark two end point 25m further so
            they can be viewed as intersections
      for(j=1;j<=n;j++)
        if(i!=j) //if two lines are the same skip
        if(intersect(a[i], a[j])) { //if two lines intersect
          s++;cross(a[i],a[j]); //increase the number of intersections (s), and call cross to update b
        }
      sort(b+1, b+s+1); //sort b so that all nodes are arranged in order
      for(j = 1; j < s ; j++) { // for each segments the max number of trees to be planted was [l/50]
            (lower bound)
        len = b[j+1] - b[j];
        k = (int)(len/50);
        ans+=k; //add to the output
      }
    }
    printf("Map %d\nTrees = %d\n",++cases,ans);
  }
}
```