```
import java.awt.geom.Line2D;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Monster {

  ////////////// Geometry Library //////////////////////
  private static final double EPS = 1e-10;

  private static int cmp(double x, double y) {
    return (x <= y + EPS) ? (x + EPS < y) ? -1 : 0 : 1;
  }
  // Immutable Point Class.
  private static class Point implements Comparable<Point> {
    public double x;
    public double y;
    public Point(double x, double y) {
      this.x = x;
      this.y = y;
    }
    public Point() {
      this.x = 0.0;
      this.y = 0.0;
    }
    public double dotProduct(Point o) {
      return this.x * o.x + this.y * o.y;
    }
    public double crossProduct(Point o) {
      return this.x * o.y - this.y * o.x;
    }
    public Point add(Point o) {
      return new Point(this.x + o.x, this.y + o.y);
    }
    public Point substract(Point o) {
      return new Point(this.x - o.x, this.y - o.y);
    }
    public Point multiply(double m) {
      return new Point(this.x * m, this.y * m);
    }
    public Point divide(double m) {
      return new Point(this.x / m, this.y / m);
    }
    @Override
    public int compareTo(Point o) {
      if (this.x < o.x) return -1;
      if (this.x > o.x) return 1;
      if (this.y < o.y) return -1;
      if (this.y > o.y) return 1;
      return 0;
    }
    // Euclidean distance between two points;
    double distance(Point o) {
        double d1 = x - o.x, d2 = y - o.y;
        return Math.sqrt(d1 * d1 + d2 * d2);
    }
```

```
}

// Calculates the angle between the two vectors defined by p - r and q - r.
// The formula comes from the definition of the dot and the cross product:
//
// A . B = |A||B|cos(c)
// A x B = |A||B|sin(c)
//
// sin(c)   A x B
// ------ = ------ = tan(c)
// cos(c)   A . B
private static double angle(Point p, Point q, Point r) {
  Point u = p.substract(r), v = q.substract(r);
  return Math.atan2(u.crossProduct(v), u.dotProduct(v));
}

// Calculates sign of the turn between the two vectors defined by <p-r> and
// <q-r>.
//
// Just to remember, the cross product is defined by (x1 * y2) - (x2 * y1) and
// is negative if it is a right turn and positive if it is a left turn. e.g.
//
//             .p3
//             ^
//            /
//       .p2 /
//       ^  /
//       | /
//       .p1
// The cross product between the vectors <p2-p1> and <p3-p1> is negative, that
// means it is a right turn.
private static int turn(Point p, Point q, Point r) {
  return cmp((p.substract(r)).crossProduct(q.substract(r)), 0.0);
}

// Decides if the point r is inside the segment defined by the points p and q.
// To do this, we have to check two conditions:
// 1. That the turn between the two vectors formed by p - q and r - q is zero
// (that means they are parallel).
// 2. That the dot product between the vector formed by p - r and q - r (that
// means the testing point as the initial point for both vectors) is less than
// or equal to zero (that means that the two vectors have opposite direction).
private static boolean between(Point p, Point q, Point r) {
  return turn(p, r, q) == 0 && cmp((p.substract(r)).dotProduct(q.substract(r)), 0.0) <= 0;
}

// Returns 0, -1 or 1 depending if p is in the exterior, the frontier or the
// interior of the given polygon respectively, the polygon must be in clockwise
// or counterclockwise order [MANDATORY!!].
// The idea is to iterate over each of the points in the polygon and consider
// the segment formed by two adjacent points, if the test points is inside that
// segment, the point is in the frontier, if not, we add the angles inside the
// vectors formed by the two points of the polygon and the test point. For a
// point outside the polygon this sum is zero because the angles cancel
// themselves.
private static int inPolygon(Point p, Point[] polygon, int polygonSize) {
  double a = 0; int N = polygonSize;
  for (int i = 0; i < N; ++i) {
    if (between(polygon[i], polygon[(i + 1) % N], p)) return -1;
    a += angle(polygon[i], polygon[(i + 1) % N], p);
```

```
    }
    return (cmp(a, 0.0) == 0) ? 0 : 1;
  }

  private static Point GetIntersection(Line2D.Double l1, Line2D.Double l2) {
    double A1 = l1.y2 - l1.y1;
    double B1 = l1.x1 - l1.x2;
    double C1 = A1 * l1.x1 + B1 * l1.y1;
    double A2 = l2.y2 - l2.y1;
    double B2 = l2.x1 - l2.x2;
    double C2 = A2 * l2.x1 + B2 * l2.y1;
    double det = A1*B2 - A2*B1;
      if(det == 0){
          // Lines are parallel, check if they are on the same line.
        double m1 = A1 / B1;
        double m2 = A2 / B2;
        // Check whether their slopes are the same or not, or if they are vertical.
        if (cmp(m1, m2) == 0 || (B1 == 0 && B2 == 0)) {
          if ((l1.x1 == l2.x1 && l1.y1 == l2.y1) ||
              (l1.x1 == l2.x2 && l1.y1 == l2.y2)) return new Point(l1.x1, l1.y1);
          if ((l1.x2 == l2.x1 && l1.y2 == l2.y1) ||
              (l1.x2 == l2.x2 && l1.y2 == l2.y2)) return new Point(l1.x2, l1.y2);
        }
        return null;
      }
        double x = (B2*C1 - B1*C2) / det;
        double y = (A1*C2 - A2*C1) / det;
        return new Point(x, y);
  }
  ////////////////////////////////////////////////////////

  private static Line2D.Double[] lines;
  private static List<Integer>[] graph;
  private static int[] marked;
  private static int[] stack;
  private static int[] cycle;
  private static int stackLen;
  private static int cycleLen;
  private static boolean res;

  private static void FindCycle(int node) {
    cycleLen = 0;
    cycle[cycleLen++] = node;
    int k = stackLen - 1;
    while (stack[k] != node) {
      cycle[cycleLen++] = stack[k];
      --k;
    }
    cycle[cycleLen++] = stack[k];
    Point[] points = new Point[cycleLen];
    for (int i = 0; i < cycleLen - 1; ++i) {
      points[i] = GetIntersection(lines[cycle[i]], lines[cycle[i + 1]]);
    }
    if (inPolygon(new Point(), points, cycleLen - 1) != 0) res = true;
  }

  private static void DoIt(int act, int last) {
    marked[act] = 1;
    stack[stackLen++] = act;
    for (Integer i : graph[act]) {
```

```
      if (marked[i] == 1 && i != last) {
        FindCycle(i);
      } else if (marked[i] == 0) {
        DoIt(i, act);
      }
    }
    --stackLen;
    marked[act] = 2;
  }

  @SuppressWarnings("unchecked")
  public static void main(String[] args) throws IOException {
    System.setIn(new FileInputStream("monster.in"));
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    String[] parts;
    while (true) {
      int num = Integer.parseInt(reader.readLine());
      if (num == 0) break;
      lines = new Line2D.Double[num];
      for (int i = 0; i < num; ++i) {
        parts = reader.readLine().split("[ ]+");
        lines[i] = new Line2D.Double(Integer.parseInt(parts[0]),
            Integer.parseInt(parts[1]), Integer.parseInt(parts[2]),
            Integer.parseInt(parts[3]));
      }
      graph = (List<Integer>[]) new List[num];
      for (int i = 0; i < num; ++i) {
        graph[i] = new ArrayList<Integer>();
      }
      for (int i = 0; i < num; ++i) {
        for (int j = i + 1; j < num; ++j) {
          if (lines[i].intersectsLine(lines[j])) {
            graph[i].add(j);
            graph[j].add(i);
          }
        }
      }

      res = false;
      marked = new int[num];
      stack = new int[num];
      cycle = new int[num + 1];
      stackLen = 0;
      Arrays.fill(marked, 0);
      for (int i = 0; i < num; ++i) {
        if (marked[i] != 0) continue;
        DoIt(i, -1);
      }

      if (res) System.out.println("yes");
      else System.out.println("no");
    }
  }
}
```

```
#include ...
using namespace std;

const int MAXN = 105;
int g[2 * MAXN][2 * MAXN]; // 0 source, n+m+1 sink
int flow[2 * MAXN][2 * MAXN];
int prev[2 * MAXN];
pair <double, double> gophers [MAXN];
pair <double, double> holes [MAXN];


bool canReach(int i, int j, double max_dist){
    double x1 = gophers[i].first; double y1 = gophers[i].second;
    double x2 = holes[j].first; double y2 = holes[j].second;
    double d = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    if (d <= max_dist) return true;
    return false;
}

int max_flow(int s, int t){
    int max_flow = 0;

    for (int i = s; i <= t; i++)
        for (int j = s; j <= t; j++)
            flow[i][j] = 0;

    while (true){
        // Find path s to t
        for (int i = s; i <= t; i++)
            prev[i] = -1;

        queue <int> q;
        q.push(s);
        prev[s] = -2;
        while (q.size() > 0){
            int u = q.front(); q.pop();
            if (u == t) break;
            for (int v = s; v <= t; v++){
                if (prev[v] == -1 and g[u][v] - flow[u][v] > 0){
                    q.push(v);
                    prev[v] = u;
                }
            }
        }
        if (prev[t] == -1) break;

        // Find bottlneck
        int curr = t;
        int bottleneck = 1 << 30;
        while (curr != s){
            bottleneck = min(bottleneck, g[prev[curr]][curr] - flow[prev[curr]][curr]);
```

```
            curr = prev[curr];
        }

        // Pump
        curr = t;
        while (curr != s){
            flow[prev[curr]][curr] += bottleneck;
            flow[curr][prev[curr]] -= bottleneck;
            curr = prev[curr];
        }

        // Add flow to answer
        max_flow += bottleneck;
    }

    return max_flow;
}


int main(){
    int m, n, sec, vel;
    while (cin >> n >> m >> sec >> vel){
        int s = 0;
        int t = n + m + 1;

        for (int i = 0; i <= t; i++)
            for (int j = 0; j <= t; j++)
                g[i][j] = 0;

        for (int i = 0; i < n; i++){
            double x, y;
            cin >> x >> y;
            gophers[i] = make_pair(x, y);
            g[s][i + 1] = 1;
        }
        for (int i = 0; i < m; i++){
            double x, y;
            cin >> x >> y;
            holes[i] = make_pair(x, y);
            g[i + 1 + n][t] = 1;
        }
        double max_dist = sec * vel;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (canReach(i, j, max_dist)) {
                    g[i + 1][n + 1 + j] = 1;
                }
        cout << n - max_flow(s, t) << endl;
    }
}
```

```
// Encontrar puentes.
#include ...
using namespace std;

const int MAXN = 10005;

vector<int> g[MAXN];
int p[MAXN], d[MAXN], low[MAXN], tick;

int find(int x) {
    return p[x] == x ? x : p[x] = find(p[x]);
}

int link(int x, int y) {
    int a = find(x), b = find(y);
    if (a != b) {
        p[a] = b;
    }
}

// It's assumed that there is at most one edge
// between two nodes.
void dfs(int u, int parent = -1) {
    d[u] = low[u] = tick++;
    foreach(out, g[u]) {
        int v = *out;
        if (v == parent) continue;
        if (d[v] == -1) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], d[v]);
        }

        if (low[v] > d[u]) {
            //printf("edge from %d to %d is a bridge\n", u + 1, v + 1);
            link(u, v);
        }
    }
}

int main(){
    int n, e, q;
    while (scanf("%d %d %d", &n, &e, &q) == 3) {
        if (n == 0 and e == 0 and q == 0) break;
        for (int i = 0; i < n; ++i) {
            g[i].clear();
            p[i] = i;
            d[i] = -1;
        }
        // read edges
        for (int i = 0; i < e; ++i) {
            int u, v; scanf("%d %d", &u, &v);
            u--, v--;
            g[u].push_back(v);
            g[v].push_back(u);
        }

        tick = 0;
        for (int i = 0; i < n; ++i) {
```

```
            if (d[i] == -1) dfs(i);
        }

        // read queries
        for (int i = 0; i < q; ++i) {
            int u, v; scanf("%d %d", &u, &v);
            u--, v--;

            if (find(u) == find(v)) {
                puts("Y");
            } else {
                puts("N");
            }
        }
        puts("-");
    }
    return 0;
}
```

```
// another fine solution by misof
#include <algorithm>
#include <numeric>

#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <stack>

#include <cstdio>
#include <cstdlib>
#include <cctype>
#include <cassert>

#include <cmath>
#include <complex>
using namespace std;

#define SIZE(t) ((int)((t).size()))

// interval tree class
class interval_tree_vertex {
  public:
  int color; // 0 empty, 1-7 rainbow
  int summary[8];
  void set_color(int c) { color=c; for (int i=0; i<8; ++i) summary[i]=0; }
  interval_tree_vertex() { set_color(0); }
};

class interval_tree {
  int L;
  void insert(int lo, int hi, int color, int kde, int left, int length);
  int count(int lo, int hi, int color, int kde, int left, int length);
  public:
  vector<interval_tree_vertex> data;
  interval_tree(int N);
  void insert(int lo, int hi, int color);
  int count(int lo, int hi, int color);
};

interval_tree::interval_tree(int N) {
  for (int i=1; ; i*=2) if (i>N+2) { L=i; break; }
  data.resize(2*L);
}

void interval_tree::insert(int lo, int hi, int color) { insert(lo,hi,color,1,0,L); }
int interval_tree::count(int lo, int hi, int color) { return count(lo,hi,color,1,0,L); }

void interval_tree::insert(int lo, int hi, int color, int kde, int left, int length) {
  if (hi <= left || lo >= left+length) return; // mimo
  if (lo <= left && left+length <= hi) { // cely dnu
    data[kde].set_color(color);
    data[kde].summary[color] = length;
    return;
  }
  if (data[kde].color != 0) {
```

```
    int cc = data[kde].color;
    data[2*kde].set_color(cc);           data[2*kde+1].set_color(cc);
    data[2*kde].summary[cc] = length/2;  data[2*kde+1].summary[cc] = length/2;
  }
  data[kde].set_color(0);
  insert(lo,hi,color,2*kde,left,length/2);
  insert(lo,hi,color,2*kde+1,left+length/2,length/2);
  for (int i=0; i<8; ++i) data[kde].summary[i] += data[2*kde].summary[i];
  for (int i=0; i<8; ++i) data[kde].summary[i] += data[2*kde+1].summary[i];
}

int interval_tree::count(int lo, int hi, int color, int kde, int left, int length) {
  if (hi <= left || lo >= left+length) return 0; // mimo
  if (lo <= left && left+length <= hi) return data[kde].summary[color]; // cely dnu
  if (data[kde].color != 0) {
    int cc = data[kde].color;
    data[2*kde].set_color(cc);           data[2*kde+1].set_color(cc);
    data[2*kde].summary[cc] = length/2;  data[2*kde+1].summary[cc] = length/2;
  }
  return count(lo,hi,color,2*kde,left,length/2) + count(lo,hi,color,2*kde+1,left+length/2,length/2);
}

// the original tree
int N;
vector<vector<int> > G;

// rooted tree as parent/child edges
vector<vector<int> > children;
vector<int> parent;

// vertex processing times for the DFS
vector<int> time_in, time_out;

// heavy-light decomposition of the tree into paths
vector< vector<int> > paths;
vector<int> path_id, path_offset;

// an interval tree for each path
vector<interval_tree> trees;

void load() {
  cin >> N;
  G.clear(); G.resize(N);
  for (int i=0; i<N-1; ++i) {
    int x,y;
    cin >> x >> y;
    G[x].push_back(y);
    G[y].push_back(x);
  }
}

void dfs() {
  parent.clear(); parent.resize(N);
  children.clear(); children.resize(N);
  time_in.clear(); time_in.resize(N);
  time_out.clear(); time_out.resize(N);
  paths.clear();
  vector<bool> visited(N,false);
  vector<int> walk;
  vector<int> subtree_size(N,0);
```

```
  int time = 0;

  // run the DFS to compute lots of information
  stack<int> vertex, edge;
  visited[0]=true; time_in[0]=time; parent[0]=0;
  vertex.push(0); edge.push(0);
  while (!vertex.empty()) {
    ++time;
    int kde = vertex.top(); vertex.pop();
    int e = edge.top(); edge.pop();
    if (e == SIZE(G[kde])) {
      walk.push_back(kde);
      time_out[kde] = time;
      subtree_size[kde] = 1;
      for (int i=0; i<SIZE(children[kde]); ++i) subtree_size[kde] += subtree_size[children[kde][i]];
    } else {
      vertex.push(kde); edge.push(e+1);
      int kam = G[kde][e];
      if (!visited[kam]) {
        visited[kam]=true; time_in[kam]=time; parent[kam]=kde; children[kde].push_back(kam);
        vertex.push(kam); edge.push(0);
      }
    }
  }

  // compute the heavy-light decomposition
  vector<bool> parent_edge_processed(N,false);
  parent_edge_processed[0] = true;
  for (int i=0; i<SIZE(walk); ++i) {
    int w = walk[i];
    if (parent_edge_processed[w]) continue;
    vector<int> this_path;
    this_path.push_back(w);
    while (1) {
      bool is_parent_edge_heavy = (2*subtree_size[w] >= subtree_size[parent[w]]);
      parent_edge_processed[w] = true;
      w = parent[w];
      this_path.push_back(w);
      if (!is_parent_edge_heavy) break;
      if (parent_edge_processed[w]) break;
    }
    paths.push_back(this_path);
  }

  path_id.clear(); path_id.resize(N); path_id[0]=-1;
  path_offset.clear(); path_offset.resize(N);

  for (int i=0; i<SIZE(paths); ++i)
    for (int j=0; j<SIZE(paths[i])-1; ++j) {
      path_id[ paths[i][j] ] = i;
      path_offset[ paths[i][j] ] = j;
    }

  trees.clear();
  for (int i=0; i<SIZE(paths); ++i) trees.push_back( interval_tree( SIZE(paths[i])-1 ) );
}

// return whether x is an ancestor of y
inline bool is_ancestor(int x, int y) {
  return (time_in[y] >= time_in[x] && time_out[y] <= time_out[x]);
```

```
}

// return the number of edges on the x-y path that do NOT have color c
// afterwards, color all edges on the x-y path using the color c
int query(int x, int y, int c) {
  if (x==y) return 0;
  if (is_ancestor(x,y)) return query(y,x,c);
  int p = path_id[x];
  int lo = path_offset[x], hi = SIZE(paths[p])-1;
  if (is_ancestor(paths[p][hi], y)) {
    while (hi-lo > 1) {
      int med = (hi+lo)/2;
      if (is_ancestor(paths[p][med], y)) hi=med; else lo=med;
    }
    lo = path_offset[x]; // keep hi at found value, restore lo
  }
  int result = hi-lo - trees[p].count(lo,hi,c);
  trees[p].insert(lo,hi,c);
  return result + query(paths[p][hi],y,c);
}

string color[7] = {"red","orange","yellow","green","blue","indigo","violet"};
map<string,int> C;

int main() {
  for (int i=0; i<7; ++i) C[color[i]]=i+1;
  int TC; cin >> TC;
  while (TC--) {
    load();
    dfs();
    int Q; cin >> Q;
    vector<long long> totals(8,0);
    while (Q--) {
      int x, y; string c; cin >> x >> y >> c;
      totals[C[c]] += query(x,y,C[c]);
    }
    for (int i=1; i<8; ++i) cout << color[i-1] << " " << totals[i] << endl;
  }
  return 0;
}
// vim: fdm=marker:commentstring=\ \"\ %s:nowrap:autoread
```

```
// Dijkstra con potenciales - 0.04s
int cost[50][50], cap[50][50], flow[50][50], s, t, V, inf = 1 << 29, par[50], dist[50], p[50];

inline int ecap(int a, int b){
    if(flow[b][a]) return flow[b][a];
    else return cap[a][b] - flow[a][b];
}

inline int ecost(int a, int b){
    if(flow[b][a]) return -cost[b][a] + p[a] - p[b];
    else return cost[a][b] + p[a] - p[b];
}

bool seen[50];
bool augment(){
    for(int i = 0; i < V; i++) par[i] = -1, dist[i] = inf;
    dist[s] = 0; par[s] = -2;

    memset(seen, 0, sizeof seen);
    int u = s;
    while(u != -1){
        seen[u] = true;
        for(int v = 0; v < V; v++)
        if(ecap(u, v) && dist[v] > ecost(u, v) + dist[u])
            dist[v] = dist[u] + ecost(u, v), par[v] = u;
        u = -1;
        for(int i = 0; i < V; i++) if(!seen[i] && dist[i] != inf && (u == -1 || dist[u] > dist[i])) u = i;
    }
    for(int v = 0; v < V; v++) if(dist[v] != inf) p[v] += dist[v];
    return dist[t] != inf;
}

int mcmf(){
    int res = 0;
    memset(p, 0, sizeof p);
    while(augment()){
        for(int v = t, u = par[v]; u != -2; u = par[v = u])
            if(flow[v][u]) flow[v][u]--, res -= cost[v][u];
            else flow[u][v]++, res += cost[u][v];
    }
    return res;
}

int main(){
    int y[16][2];
    int m;
    while(scanf("%d", &m) && m){
        for(int i = 0; i < m; i++) scanf("%d", &y[i][0]);
        for(int i = 0; i < m; i++) scanf("%d", &y[i][1]);
        s = m * 2; t = s + 1; V = t + 1;
        memset(cap, 0, sizeof cap);
        memset(cost, 0, sizeof cost);
        memset(flow, 0, sizeof flow);
        for(int i = 0; i < m; i++) for(int j = 0; j < m; j++)
            cap[i][j + m] = 1, cost[i][j + m] = abs(i - j) + abs(y[i][0] - y[j][1]);
        for(int i = 0; i < m; i++) cap[s][i] = cap[i + m][t] = 1;
        cout << mcmf() << endl;
    }
}
```

```
// Dijkstra sin potenciales - 0.03s
int cost[50][50], cap[50][50], flow[50][50], s, t, V, inf = 1 << 29, par[50], dist[50];

inline int ecap(int a, int b){
    if(flow[b][a]) return flow[b][a];
    else return cap[a][b] - flow[a][b];
}

inline int ecost(int a, int b){
    if(flow[b][a]) return -cost[b][a];
    else return cost[a][b];
}

bool seen[50];
bool augment(){
    for(int i = 0; i < V; i++) par[i] = -1, dist[i] = inf;
    dist[s] = 0; par[s] = -2;
    memset(seen, 0, sizeof seen);
    int u = s;
    while(u != -1){
        seen[u] = true;
        for(int v = 0; v < V; v++)
        if(ecap(u, v) && dist[v] > dist[u] + ecost(u, v)){
            dist[v] = dist[u] + ecost(u, v);
            par[v] = u;
            seen[v] = false;
        }
        u = -1;
        for(int i = 0; i < V; i++) if(!seen[i] && dist[i] != inf && (u == -1 || dist[u] > dist[i])) u = i;
    }
    return dist[t] != inf;
}

int mcmf(){
    int res = 0;
    while(augment()){
        for(int v = t, u = par[v]; u != -2; u = par[v = u])
            if(flow[v][u]) flow[v][u]--, res -= cost[v][u];
            else flow[u][v]++, res += cost[u][v];
    }
    return res;
}

int main(){
    int y[16][2];
    int m;
    while(scanf("%d", &m) && m){
        for(int i = 0; i < m; i++) scanf("%d", &y[i][0]);
        for(int i = 0; i < m; i++) scanf("%d", &y[i][1]);
        s = m * 2; t = s + 1; V = t + 1;
        memset(cap, 0, sizeof cap);
        memset(cost, 0, sizeof cost);
        memset(flow, 0, sizeof flow);
        for(int i = 0; i < m; i++) for(int j = 0; j < m; j++)
            cap[i][j + m] = 1, cost[i][j + m] = abs(i - j) + abs(y[i][0] - y[j][1]);
        for(int i = 0; i < m; i++) cap[s][i] = cap[i + m][t] = 1;
        cout << mcmf() << endl;
    }
}
```

```
// Bellman-Ford - 0.04s
int cost[50][50], cap[50][50], flow[50][50], s, t, V, inf = 1 << 29, par[50], dist[50];

inline int ecap(int a, int b){
    if(flow[b][a]) return flow[b][a];
    else return cap[a][b] - flow[a][b];
}

inline int ecost(int a, int b){
    if(flow[b][a]) return -cost[b][a];
    else return cost[a][b];
}

bool augment(){
    for(int i = 0; i < V; i++) par[i] = -1, dist[i] = inf;
    dist[s] = 0; par[s] = -2;
    bool changed = true;
    while(changed){
        changed = false;
        for(int u = 0; u < V; u++) if(dist[u] != inf) for(int v = 0; v < V; v++)
            if(ecap(u, v) && dist[v] > dist[u] + ecost(u, v)){
                dist[v] = dist[u] + ecost(u, v);
                par[v] = u;
                changed = true;
            }
    }
    return dist[t] != inf;
}

int mcmf(){
    int res = 0;
    while(augment()){
        for(int v = t, u = par[v]; u != -2; u = par[v = u])
            if(flow[v][u]) flow[v][u]--, res -= cost[v][u];
            else flow[u][v]++, res += cost[u][v];
    }
    return res;
}

int main(){
    int y[16][2];
    int m;
    while(scanf("%d", &m) && m){
        for(int i = 0; i < m; i++) scanf("%d", &y[i][0]);
        for(int i = 0; i < m; i++) scanf("%d", &y[i][1]);
        s = m * 2; t = s + 1; V = t + 1;
        memset(cap, 0, sizeof cap);
        memset(cost, 0, sizeof cost);
        memset(flow, 0, sizeof flow);
        for(int i = 0; i < m; i++) for(int j = 0; j < m; j++)
            cap[i][j + m] = 1, cost[i][j + m] = abs(i - j) + abs(y[i][0] - y[j][1]);
        for(int i = 0; i < m; i++) cap[s][i] = cap[i + m][t] = 1;
        cout << mcmf() << endl;
    }
}
```

```
// Star War de Filipe Martins
#include <bits/stdc++.h>
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

#define fr(a,b,c) for( int a = b ; a < c ; ++a )
#define rep(a,b) fr(a,0,b)
#define db(x) cout << #x " == " << x << endl
#define dbg db
#define _ << ", " <<

#define EPS 1e-7
int comp(double x, double y) {
    if( fabs(x-y) < EPS ) return 0;
    return x < y ? -1 : 1;
}

struct P{
    double x,y,z;
    P() {}
    P(double x, double y, double z): x(x), y(y), z(z) {}

    P operator+(P b) { return P(x+b.x, y+b.y, z+b.z); }
    P operator-() { return P(-x,-y,-z); }
    P operator-(P b) { return *this+-b; }
    double operator*(P b){ return x*b.x + y*b.y + z*b.z; }
    P operator*(double k){ return P(x*k, y*k, z*k); }
    P operator%(P b){ return P(y*b.z - z*b.y, z*b.x - x*b.z, x*b.y - y*b.x); } // cross product
    P operator/(P b){ return b*(*this*b/(b*b)); } // projection of this onto b
    double operator!() { return sqrt(*this**this); } // length
} p[4], q[4];

// Distance from point c to segment [a, b]
double distSP(P a, P b, P c) {
    P pp = a + (c-a)/(b-a);
    if( !comp(!(a-pp) + !(pp-b), !(a-b)) ) return !( c-pp );
    return min(!(a-c), !(b-c));
}

// Distance from segment [a, b] to segment [c, d]
double distSS(P a, P b, P c, P d) {
    P ba = b-a;
    P cd = c-d;
    P ca = c-a;
    P w = ba%cd;
    double dd = w*w;
    if( !comp(dd,0) ) { // both segments are parallel
        return min(min(distSP(a,b,c), distSP(a,b,d)), min(distSP(c,d,a), distSP(c,d,b)));
    }
    double x = ((ca%cd)*w)/dd;
    double y = ((ba%ca)*w)/dd;
    double z = ((ba%cd)*ca)/dd;
    if( x >= 0 && x <= 1 && y >= 0 && y <= 1 ) return !(w*z);
    return min(min(distSP(a,b,c), distSP(a,b,d)), min(distSP(c,d,a), distSP(c,d,b)));
}

// Distance from point d to triangle [a, b, c]
double distPP(P a, P b, P c, P d) {
```

```
    P ba = b-a;
    P ca = c-a;
    P da = d-a;
    P w = ba%ca;
    P q = d-da/w;
    double x = (b-a)%(q-a) * w, y = (c-b)%(q-b) * w, z = (a-c)%(q-c) * w;
    if( x <= 0 && y <= 0 && z <= 0 || x >= 0 && y >= 0 && z >= 0 ) return !(da/w);
    return min( min(distSP(a,b,d), distSP(b,c,d)), distSP(c,a,d));
}

int read() {
    fr(i,0,4) scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
    fr(i,0,4) scanf("%lf%lf%lf", &q[i].x, &q[i].y, &q[i].z);

    double dist = 1./0. ;
    fr(i,0,4) fr(j,i+1,4) fr(k,j+1,4) fr(l,0,4) {
        double d = distPP(p[i], p[j], p[k], q[l]);
        if( d < dist ) dist = d;
        d = distPP(q[i], q[j], q[k], p[l]);
        if( d < dist ) dist = d;
    }
    fr(i,0,4) fr(j,i+1,4) fr(k,0,4) fr(l,k+1,4) {
        double d = distSS(p[i], p[j], q[k], q[l]);
        if( d < dist ) dist = d;
    }

    printf("%.2lf\n", dist);

    return 1;
}

void process() {

}

int main() {
    int t = 1;
    scanf("%d", &t);
    while( t-- && read() ) process();
    return 0;
}
```