LIBRARY
EQUIPE ITA-CARTEADO

Sumário

1. Codigos de início de prova

1.1 Makefile

```
debug:
	g++ -Wall -g -o $n $n.cpp
release:
	g++ -Wall -O2 -o $n $n.cpp
tests: debug
	./$n < in1 | diff -ysN - out1
```

1.2 modelo.cpp

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cassert>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <list>
#include <deque>
#include <queue>
#include <stack>
#include <functional>
#include <sstream>
#include <iostream>
#include <ctime>
#include <algorithm>
using namespace std;

#define DEBUG(x...) printf(x)
#define all(v) (v).begin(),(v).end()
#define rall(v) (v).rbegin(),(v).rend()
#define _foreach(it, b, e) for(__typeof__(b) it = (b); it != (e); ++it)
#define foreach(x...) _foreach(x)

typedef long long int huge;

const int inf = 0x3f3f3f3f;
const huge hugeinf = 0x3f3f3f3f3f3f3f3fll; // sao dois L's!!!
const double eps = 1e-9;

// em caso de emergencia
#define _inline(f...) inline f() __attribute__((always_inline)); f
```

1.3 struct.sh

```
#!/bin/bash
for str in "$@"
do
    mkdir -p $str
    echo n=$str > $str/Makefile
    cat Makefile >> $str/Makefile
    cp modelo.cpp $str/$str.cpp
done
```

2. Itens Importantes

2.1 Antes de começar

- Reler a estratégia de prova algumas vezes
- Reler a lista de algoritmos no bizuário

2.2 Resolvendo um problema

- LAPDED - Leia a PORRA do enunciado direito!
- Faça as contas! Limites de arrays e cuidado com overflows
- Peça as clarifications antes

2.3 Debugando um problema

- WA/RE, não sabe por quê e mais de 30min sem idéia? Implementa de novo
- Bugs e casos extremos: criar casos de teste, sempre
- Compiler Error em ambientes toscos: busca binária em casos extremos

2.3.1 Bugs do milênio

- Verificar overflows, ver se o ∞ é tão infinito quanto parece
- Doubles: igualdade com tolerância
- Igualdade dentro de `if`
- `C-w C-y`? Errou algo.
- Tamanho de vetores
- `long long a = 1 << 40;` $\Longrightarrow$ `long long a = 1ll << 40;`
- Variáveis com nome min,max
- Inicialização de variáveis

- Casos extremos, muito pequenos ou muito grandes
- Self-loops e multiarestas em grafos
- Otimização de casos específicos
- Imprecisão ao subtrair números quase iguais
- Resto de divisão com números negativos

3. Estratégia

3.1 300 minutos - Início

- Digitar `Makefile`, `modelo.cpp` e `struct.sh`
- Criar os fontes modelo usando `struct.sh problema1 ... problemaN`
- Enquanto isso (ou enquanto o primeiro problema estiver sendo resolvido), outras duas pessoas dividem os problemas e começam a ler, anotando no quadro ordem de resolução e idéias preliminares. Não ter medo de colocar infinito no problema.
- Ler o máximo de problemas possível.
- Na dúvida, olhar os balões e o scoreboard.

3.2 200 minutos - Meio

- Todos já devem ter lido todos os problemas não resolvidos da prova
- Limite de questões em paralelo: **2 ou 3**
- Logo que mandar o problema, mande imprimir código e saída (com debug)
- WA/RE, não sabe por quê, 30+ min sem idéia? Pense em reimplementar.
- Na dúvida, olhar os balões e o scoreboard.

3.3 100 minutos - Começo do fim

- Limite de questões em paralelo: **1 ou 2**
- Olhar os baloẽs e o scoreboard - escolha dos problemas difíceis
- Equipe mais unida

3.4 15 minutos - Juízes calados

- Mexer um pouco e mandar. Tirar debug. Pensar só depois em mais casos de teste.

4. Geometria (números inteiros)

```
struct point
{
  huge x, y;
  point(huge x=0, huge y=0) : x(x), y(y) {}
  inline point operator+(const point &p) const {return point(x+p.x, y+p.y);}
  inline point operator-(const point &p) const {return point(x-p.x, y-p.y);}
  inline bool operator==(const point &p) const {return x==p.x&&y==p.y;}
};

struct line
{
  point p1, p2;
  line(point p1, point p2) : p1(p1), p2(p2) {}
};

inline huge dot(const point &a, const point &b) {return a.x*b.x + a.y*b.y;}
inline huge cross(const point &a, const point &b) {return a.x*b.y - a.y*b.x;}

huge ccw(const point &a, const point &b, const point &c)
{
  huge k;
  if( (k=cross(b-a,c-a)) > 0 ) return 1;
  if( k < 0 ) return -1;
  if( dot(c-a,b-a) < 0 ) return -1;
  if( dot(a-b,c-b) < 0 ) return 1;
  return 0;
}

// Falha se line a ou line b for pontual, ver se pode ocorrer!
bool intersect(const line &a, const line &b)
{
  return ( ccw(a.p1, a.p2, b.p1) * ccw(a.p1, a.p2, b.p2) <= 0 ) &&
    ( ccw(b.p1, b.p2, a.p1) * ccw(b.p1, b.p2, a.p2) <= 0 );
}

inline bool between(point a, point b, point p)
{
  return cross(b-a,p-a)==0 && dot(p-a,p-b) <= 0;
}
```

5. Geometria (ponto flutuante)

```
inline int cmp(double x, double y = 0)
{ return (x < y + eps) ? (x + eps < y) ? -1 : 0 : 1; }

struct point
{
  double x, y;
  point(double x=0, double y=0) : x(x), y(y) {}
  inline point operator-(const point &p) const {return point(x-p.x, y-p.y);}
  inline point operator+(const point &p) const {return point(x+p.x, y+p.y);}
  inline point operator*(const double &c) const {return point(x*c, y*c);}
  inline point operator/(const double &c) const {return point(x/c, y/c);}

  inline int cmp(const point &p) const
  { if(int t = ::cmp(x, p.x)) return t; return ::cmp(y, p.y); }
  inline bool operator==(const point &p) const { return cmp(p) == 0; }
  inline bool operator!=(const point &p) const { return cmp(p) != 0; }
  inline bool operator<(const point &p) const { return cmp(p) < 0; }

  static point pivot; // para radial_lt
};

point point::pivot;

struct line
{
  point a, b;
  line(point a = point(0, 0), point b = point(0, 0)) : a(a), b(b) {}
```

```
};

inline double dot(const point &a, const point &b) {return a.x*b.x+a.y*b.y;}
inline double cross(const point &a, const point &b) {return a.x*b.y-a.y*b.x;}
inline double norm(const point &p) {return sqrt(dot(p,p));}
inline double arg(const point &p) {return atan2(p.y,p.x);}
inline double angle(const point &a, const point &b, const point &c)
{ point u=a-b,v=c-b; return atan2(cross(u,v),dot(u,v)); }

inline bool between(const point &a, const point &b, const point &p)
{ return cmp(cross(p-a,p-b))==0 && cmp(dot(a-p,b-p))<=0; }

int ccw(const point &a, const point &b, const point &c)
{
  double k = cross(b-a,c-a);
  if( k > eps ) return 1;
  if( k < -eps ) return -1;
  if( dot(c-a,b-a) < -eps ) return -1;
  if( dot(a-b,c-b) < -eps ) return 1;
  return 0;
}

// Falha se line a ou line b for pontual, ver se pode ocorrer!
bool intersect(const line &a, const line &b)
{
  return ( ccw(a.a, a.b, b.a) * ccw(a.a, a.b, b.b) <= 0 ) &&
    ( ccw(b.a, b.b, a.a) * ccw(b.a, b.b, a.b) <= 0 );
}

// distancia de r a pq (segmento)
double linedist(const point &p, const point &q, const point &r)
{
  point A = r - q, B = r - p, C = q - p;
  double a = dot(A,A), b = dot(B,B), c = dot(C,C);
  if (cmp(b, a + c) >= 0) return sqrt(a);
  else if (cmp(a, b + c) >= 0) return sqrt(b);
  else return fabs(cross(A,B)) / sqrt(c);
}

// 0: ext; -1: front; 1: int
int inside(const point &p, const vector<point> &T)
{
  double a = 0; int n = T.size();
  for (int i = 0; i < n; i++)
    {
      if (between(T[i], T[(i+1)%n], p)) return -1;
      a += angle(T[i], p, T[(i+1) % n]);
    }
  return cmp(a) != 0;
}

bool radial_lt(const point &p, const point &q)
{
  point P = p - point::pivot, Q = q - point::pivot;
  double r = cross(P, Q);
  if(cmp(r)) return r>0;
  return cmp(dot(P, P), dot(Q, Q)) < 0;
}

vector<point> convex_hull(vector<point>& T)
{
  int j = 0, k, n = T.size(); vector<point> U(n);
  point::pivot = *min_element(all(T));
  sort(all(T), radial_lt);
  for (k = n-2; k >= 0 && cmp(cross(T[0] - T[k], T[n-1] - T[k])) == 0; k--);
  reverse((k+1) + all(T));
  for (int i = 0; i < n; i++) {
    // troque o <= por < para manter pontos colineares
    while (j > 1 && cmp(cross(U[j-1] - U[j-2], T[i] - U[j-2])) <= 0) j--;
    U[j++] = T[i];
  }
  U.erase(j + all(U));
  return U;
}

// area orientada
double poly_area(vector<point>& T)
{
  double s = 0; int n = T.size();
  for (int i = 0; i < n; i++)
    s += cross(T[i], T[(i+1) % n]);
  return s / 2;
}

// interseccao de retas
point line_intersect(const line &r, const line &s)
{
  point a = r.b - r.a, b = s.b - s.a, c = point(cross(r.a,r.b),cross(s.a,s.b));
  return point(cross(point(a.x, b.x),c), cross(point(a.y, b.y),c)) / cross(a,b);
}

// spanning circle
typedef pair<point, double> circle;
bool in_circle(circle C, point p){
  return cmp(norm(p - C.first), C.second) <= 0;
}

point circumcenter(point p, point q, point r) {
  point a = p - r, b = q - r, c = point(dot(a, p + r) / 2, dot(b, q + r) / 2);
  return point(cross(c, point(a.y, b.y)), cross(point(a.x, b.x), c)) / cross(a, b);
}

circle spanning_circle(vector<point>& T) {
  int n = T.size();
  random_shuffle(all(T));
  circle C(point(), -INFINITY);
```

```
  for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
      C = circle(T[i], 0);
      for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
          C = circle((T[i] + T[j]) / 2, norm(T[i] - T[j]) / 2);
          for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
              point o = circumcenter(T[i], T[j], T[k]);
              C = circle(o, norm(o - T[k]));
            }
        }
    }
  return C;
}
```

6. Teoria dos Números

```
int gcd(int x, int y) { return y?gcd(y,x%y):abs(x); }

int lcm(int x, int y)
{
  if(x&&y) return abs(x)/gcd(x,y)*abs(y);
  return abs(x|y);
}

bool is_prime(int n)
{
  if(n<0) return is_prime(-n);
  if(n<5||n%2==0||n%3==0) return (n==2||n==3);
  int maxn = sqrt(n)+2;
  for(int i=5; i<maxn; i+=6)
    if(n%i==0||n%(i+2)==0) return false;
  return true;
}

void squeeze(map<int,int> &f, int &n, int &p) {for(;n%p==0;n/=p) ++f[p];}
map<int,int> factor(int n)
{
  map<int,int> ans;
  if(n<0) return factor(-n);
  if(n<2) return ans;
  squeeze(ans,n,2); squeeze(ans,n,3);
  int maxn = sqrt(n)+2;
  for(int i=5; i<maxn; i+=6)
    squeeze(ans,n,i), squeeze(ans,n,i+2);
  if(n>1) ++ans[n];
  return ans;
}

typedef struct{int d, a, b;} bezout;

// retorna (d, a, b), onde d = ax + by.
bezout extgcd(int x, int y)
{
  if(y==0) return (bezout){x, 1, 0};
  bezout s = extgcd(y, x%y);
  return (bezout){s.d, s.b, s.a-x/y*s.b};
}
```

7. Garfos

```
////////////////////////////////////////////////////////////////////////////
// DIJKSTRA COM SET E ADJACENCY LIST (EH FACIL MODIFICAR PRA ADJ MTX)
////////////////////////////////////////////////////////////////////////////
vector<int> dist(tamanho do grafo);
void dijkstra(int ori)
{
  set<pair<int, int> > td;
  td.insert(make_pair(0, ori));
  dist[ori] = 0;
  while(!td.empty())
    {
      int v = td.begin()->second;
      td.erase(td.begin());
      foreach(it, all(graph[v])) // lista de adj
        if( dist[it->first] > dist[v] + it->second )
          {
            if( dist[it->first] != INF )
              td.erase(td.find(make_pair(dist[it->first], it->first)));
            dist[it->first] = dist[v] + it->second;
            td.insert(make_pair(dist[it->first], it->first));
          }
    }
}

////////////////////////////////////////////////////////////////////////////
// UNWEIGHTED BIPARTITE MATCHING
////////////////////////////////////////////////////////////////////////////
// Matching resultante: para matching[i]!=-1, (i, matching[i]).
// tab[i][j] -> de i para j. m linhas e n colunas. (mtx no sentido matematico)
////////////////////////////////////////////////////////////////////////////
// Ideias / extensoes:
// - Para determinar se a aresta encontra-se em algum perfect matching, force
//   ela estar no matching e de um augment() no vertice que sobra (nao esquecer
//   de limpar seen)
////////////////////////////////////////////////////////////////////////////
const int nmax = 200;

struct UBMatching
{
  int m, n, size, matching[nmax], seen[nmax];
  int (*tab)[nmax];
  bool augment(int v)
  {
    for(int i=0;i<m;i++)
      if(tab[i][v] && !seen[i])
        {
          seen[i]=1;
          if(matching[i]<0 || augment(matching[i]))
             { matching[i]=v; return 1; }
```

```
        }
    return 0;
  }
  void init(int nm, int nn, int mtx[nmax][nmax])
  {
    m = nm; n = nn; size = 0; tab = mtx;
    memset(matching, -1, sizeof(matching));
  }
  void match()
  {
    for(int i=0;i<n;i++)
      { memset(seen, 0, sizeof(seen)); size += augment(i); }
  }
} ubm;

////////////////////////////////////////////////////////////////////////////
// Dinic's blocking flow algorithm (Ahuja & Orlin) - O(V^2 E)
////////////////////////////////////////////////////////////////////////////

// graph structure
const int maxn = 5050;
vector<int> graph[maxn];
huge cap[maxn][maxn];
int V;

// algorithm data
int dist[maxn], father[maxn], ndst[maxn], curarc[maxn];
huge dinic(int s, int t)
{
  huge ans = 0;
  // reverse bfs
  fill(ndst, ndst+V+1, 0); ndst[V] = V;
  fill(dist, dist+V, V); --ndst[V]; dist[t] = 0; ++ndst[0];
  fill(curarc, curarc+V, 0);
  queue<int> q; q.push(t);
  while(!q.empty()) {
    int v = q.front(); q.pop();
    foreach(it, all(graph[v])) if(dist[*it] == V && cap[*it][v] > 0)
      { --ndst[V]; dist[*it] = dist[v]+1; ++ndst[dist[*it]]; q.push(*it); }
  }
  int i = s;
  father[i] = -1;
  while(dist[s] < V) {
    if(i == t) { // augment
      huge d = hugeinf;
      for(int j=t; j!=s; j=father[j]) d = min(d, cap[father[j]][j]);
      for(int j=t; j!=s; j=father[j]) cap[father[j]][j] -= d, cap[j][father[j]] += d;
      ans += d; i = s; continue;
    }
    bool found = false;
    foreach(it, curarc[i] + all(graph[i]))
      if(dist[i] == dist[*it]+1 && cap[i][*it] > 0)
        { found = true; father[*it] = i; curarc[i] = it-graph[i].begin(); i = *it; break; }
    if(!found) { // retreat
      curarc[i] = 0;
      int tmp = dist[i];
      --ndst[dist[i]]; dist[i] = V;
      foreach(it, all(graph[i])) if(cap[i][*it] > 0) dist[i] = min(dist[i], dist[*it]+1);
      ++ndst[dist[i]];
      if(ndst[tmp] == 0) break;
      if(i != s) i = father[i];
    }
  }
  return ans;
}

////////////////////////////////////////////////////////////////////////////
// EDMONDS-KARP MAXFLOW - O(max(MaxFlow*V, VE^2))
// Muda o m para ser grafo residual.
// Na lista de adj, tem de colocar aresta nos dois sentidos, mesmo em directed
////////////////////////////////////////////////////////////////////////////
const int nmax = 200;

struct EKMaxFlow
{
  int (*m)[nmax];
  vector<int> *l;
  int marked[nmax], prev[nmax];
  int n, s, t;
  void init(int mtx[nmax][nmax], vector<int> *list, int nn, int ns, int nt)
  { m = mtx; l = list; n = nn; s = ns; t = nt; }
  int augment()
  {
    int ans = 0;
    while(true)
      {
        memset(marked, 0, sizeof(marked));
        memset(prev, 0xff, sizeof(prev));
        queue<int> q;
        q.push(s);
        marked[s] = true;
        while(!q.empty())
          {
            int v = q.front(); q.pop();
            if(v == t) break;
            foreach(it, all(l[v])) // se for mudar pra mtx, eh aqui
              if(!marked[*it] && m[v][*it] > 0)
                {
                  marked[*it] = true;
                  q.push(*it);
                  prev[*it] = v;
                }
          }
        if(prev[t] == -1) break;
        int cap = inf;
        for(int i=t; i!=s; i=prev[i])
```

```
          cap = min(cap, m[prev[i]][i]);
        for(int i=t; i!=s; i=prev[i])
          {
            m[i][prev[i]] += cap;
            m[prev[i]][i] -= cap;
          }
        ans += cap;
      }
    return ans;
  }
} ekmf;

////////////////////////////////////////////////////////////////////
//Testa se existe um perfect-matching num grafo generico.
////////////////////////////////////////////////////////////////////
struct perfectmatching //depende de maxflow
{
  int floodfill(int v, int cor)
  {
    int r=1;
    comp[v]=cor;
    foreach(it, all(adj[v]))
      if (!comp[*it])
        r+=floodfill(*it,cor);
    return r;
  }
  bool perfectmatch()
  {
    int c=1, nk, fk;
    for(int i=1; i<=n; ++i)
      if (!comp[i])
        {
          nk=floodfill(i,c);
          if (nk&1)
            return false;
          for(int j=1; j<=n; ++j)
            if (comp[j]==c)
              {
                lista[0].push_back(j);
                lista[j].push_back(0);
                mtx[0][j]=1;
              }
          fk=ekmf.augment();
          ++c;
          if (fk!=nk)
            return false;
        }
    return true;
  }
} pfmt;

////////////////////////////////////////////////////////
// Stable Marriage Problem.
////////////////////////////////////////////////////////
int girl_order[MAXN][MAXN]; //ordem de preferencia das garotas
int boy_rate[MAXN][MAXN];   //nota dada pelo garoto para a garota
int girl_pos[MAXN];         //posicao atual da garota - inicializar com zero.
int boy_pair[MAXN];         //par do garoto - inicializar com infinito.
int n;                      //o algoritmo é arroz!

bool smp()
{
  for(int i=0, k=0, j=0; k<n; i=++k)
    while(1)
      {
        if (girl_pos[i]<n)
          j=girl_order[i][girl_pos[i]];
        else
          return 0;
        if (boy_pair[j]>n)
          {
            boy_pair[j]=i;
            break;
          }
        else if (boy_rate[j][i]<boy_rate[j][boy_pair[j]])
          {
            girl_pos[boy_pair[j]]++;
            swap(boy_pair[j], i);
          }
        else
          girl_pos[i]++;
      }
  return 1;
}

////////////////////////////////////////////////////////////////////////////
// HUNGARIAN ALGORITHM (MAXIMUM WEIGHTED BIPARTITE MATCHING)
////////////////////////////////////////////////////////////////////////////
#define N 55              //max number of vertices in one part
#define INF 100000000     //just infinity

int cost[N][N];           //cost matrix
int n, max_match;         //n workers and n jobs
int lx[N], ly[N];         //labels of X and Y parts
int xy[N];                //xy[x] - vertex that is matched with x,
int yx[N];                //yx[y] - vertex that is matched with y
bool S[N], T[N];          //sets S and T in algorithm
int slack[N];             //as in the algorithm description
int slackx[N];            //slackx[y] such a vertex, that
                          // l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
int prev[N];              //array for memorizing alternating paths

void init_labels()
{
  memset(lx, 0, sizeof(lx));
  memset(ly, 0, sizeof(ly));
```

```
  for (int x = 0; x < n; x++)
    for (int y = 0; y < n; y++)
      lx[x] = max(lx[x], cost[x][y]);
}

void update_labels()
{
  int x, y, delta = INF;             //init delta as infinity
  for (y = 0; y < n; y++)            //calculate delta using slack
    if (!T[y])
      delta = min(delta, slack[y]);
  for (x = 0; x < n; x++)            //update X labels
    if (S[x]) lx[x] -= delta;
  for (y = 0; y < n; y++)            //update Y labels
    if (T[y]) ly[y] += delta;
  for (y = 0; y < n; y++)            //update slack array
    if (!T[y])
      slack[y] -= delta;
}

void add_to_tree(int x, int prevx)
{
  S[x] = true;                    //add x to S
  prev[x] = prevx;                //we need this when augmenting
  for (int y = 0; y < n; y++)    //update slacks, because we add new vertex to S
    if (lx[x] + ly[y] - cost[x][y] < slack[y])
      {
        slack[y] = lx[x] + ly[y] - cost[x][y];
        slackx[y] = x;
      }
}

void augment()                         //main function of the algorithm
{
  if (max_match == n) return;        //check wether matching is already perfect
  int x, y, root;                    //just counters and root vertex
  int q[N], wr = 0, rd = 0;          //q - queue for bfs, wr,rd - write and read
  memset(S, false, sizeof(S));       //init set S
  memset(T, false, sizeof(T));       //init set T
  memset(prev, -1, sizeof(prev));    //init set prev - for the alternating tree
  for (x = 0; x < n; x++)            //finding root of the tree
    if (xy[x] == -1)
      {
        q[wr++] = root = x;
        prev[x] = -2;
        S[x] = true;
        break;
      }

  for (y = 0; y < n; y++)            //initializing slack array
    {
      slack[y] = lx[root] + ly[y] - cost[root][y];
      slackx[y] = root;
    }
  while (true)
    {
      while (rd < wr)
        {
          x = q[rd++];
          for (y = 0; y < n; y++)
            if (cost[x][y] == lx[x] + ly[y] &&  !T[y])
              {
                if (yx[y] == -1) break;
                //augmenting path exists!
                T[y] = true;
                q[wr++] = yx[y];
                //with y, to the queue
                add_to_tree(yx[y], x);
              }
          if (y < n) break;
        }
      if (y < n) break;

      update_labels();
      wr = rd = 0;
      for (y = 0; y < n; y++)
        if (!T[y] &&  slack[y] == 0)
          {
            if (yx[y] == -1)
              {
                x = slackx[y];
                break;
              }
            else
              {
                T[y] = true;
                if (!S[yx[y]])
                  {
                    q[wr++] = yx[y];
                    add_to_tree(yx[y], slackx[y]);
                  }
              }
          }
      if (y < n) break;
    }

  if (y < n)
    {
      max_match++;
      for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty)
        {
          ty = xy[cx];
          yx[cy] = cx;
          xy[cx] = cy;
        }
      augment();
```

```
    }
}

int hungarian()
{
  int ret = 0;                      //weight of the optimal matching
  max_match = 0;                    //number of vertices in current matching
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  init_labels();                    //step 0
  augment();                        //steps 1-3
  for (int x = 0; x < n; x++)       //forming answer there
    ret += cost[x][xy[x]];
  return ret;
}

////////////////////////////////////////////////////////////////////////////
// MIN COST MAX FLOW
////////////////////////////////////////////////////////////////////////////
/*Fluxo de custo mínimo em O(n^2f) (successive shortest path, com potenciais*/
/*Se desejado o menor custo para um certo fluxo c, criar um novo nó com aresta
  para s com custo 0 e capacidade c*/
/*Observação: no final não há ciclo de custo negativo na matriz residual
  (condição de optimalidade)*/
/*vértices de 0 a n, 0 eh t, n eh s*/

#define MAXN 200
#define INF 10000000000000000LL

int n,m;
/*cap[0][1][0] = fluxo desejado, cst[0][1][0] = 0*/
long long cap[MAXN][MAXN][2];/*cap[i][j][0] = capacidades dadas, cap[i][j][1] = 0*/
long long cst[MAXN][MAXN][2];/*cst[i][j][0] = custos dados, cst[i][j][1] = -cst[j][i][0]*/
int par[MAXN],pr[MAXN];
long long p[MAXN];

void bellman(int s,int t)
{
  int sw;
  int i,j,k,h;

  for(i=0;i<=n;i++) {
    p[i] = INF;
    pr[i] = -1;
  }
  p[s] = 0;
  sw = 1;
  for(h=0;(h<=n)&&(sw);h++) {
    sw = 0;

    for(i=0;i<=n;i++) {
      for(j=0;j<=n;j++) {
        for(k=0;k<=1;k++) {
          if(cap[i][j][k] <= 0)
            continue;

          if(p[j] > p[i] + cst[i][j][k]) {
            p[j] = p[i] + cst[i][j][k];
            sw++;
          }
        }
      }
    }
  }
}

bool dijkstra(int s,int t)
{
  int v,i,k;
  bool intree[MAXN];
  long long dst[MAXN],men,c;

  for(i=0;i<=n;i++)
    {
      dst[i] = INF;
      par[i] = -1;
      pr[i] = -1;
      intree[i] = false;
    }
  dst[s] = 0;
  v = s;
  while(!intree[v])
    {
      intree[v] = true;

      for(i=0;i<=n;i++)
        {
          if(intree[i])
            continue;

          for(k=0;k<=1;k++)
            {
              if(cap[v][i][k] <= 0)
                continue;

              c = cst[v][i][k] + p[v] - p[i];
              if(dst[i] > dst[v] + c){
                dst[i] = dst[v] + c;
                par[i] = v;
                pr[i] = k;
              }
            }
        }

      men = INF;
      for(i=0;i<=n;i++)
```

```
        {
          if(intree[i])
            continue;

          if(dst[i] < men)
            {
              men = dst[i];
              v = i;
            }
        }
    }

  for(i=0;i<=n;i++)
    p[i] += dst[i];

  return intree[t];
}

long long mincost(int s,int t)
{
  int i,j,k,v;
  long long aug,ret;

  /*somente se houver aresta com custo negativo, senao comece com p[i] = 0*/
  bellman(s,t);

  while(1)
    {
      if(!dijkstra(s,t))
        break;

      v = t;
      aug = INF;
      while(v != s)
        {
          if(aug > cap[par[v]][v][pr[v]])
            {
              aug = cap[par[v]][v][pr[v]];
            }
          v = par[v];
        }

      v = t;
      while(v != s)
        {
          cap[par[v]][v][pr[v]] -= aug;
          cap[v][par[v]][1-pr[v]] += aug;
          v = par[v];
        }
    }

  ret = 0;
  for(i=0;i<=n;i++)
    {
      for(j=0;j<=n;j++)
        {
          ret += cap[j][i][1] * cst[i][j][0];
        }
    }
  return ret;
}

////////////////////////////////////////////////////////////////////////////
// STOER-WAGNER GLOBAL MIN-CUT O(n^3)
////////////////////////////////////////////////////////////////////////////
const int nmax = 200;

int graph[nmax][nmax];
bool valid[nmax];
bool marked[nmax];
int tightness[nmax];
int order[nmax];
int n, nvalid;

_inline(int minimumCutPhase)()
{
  memset(marked, 0, sizeof(marked));
  memset(tightness, 0, sizeof(tightness));
  int v;
  for(v=0; v<n; ++v)
    if(valid[v]) break;
  tightness[v] = 1;
  for(int tam=0; tam<nvalid; ++tam)
    {
      int c = max_element(tightness, tightness+n) - tightness;
      marked[c] = true;
      order[tam] = c;
      for(int i=0; i<n; ++i)
        if(!marked[i] && valid[i])
          tightness[i] += graph[c][i];
      tightness[c] = 0;
    }
  int ans = 0;
  for(int i=0; i<n; ++i)
    if(i!=order[nvalid-1] && valid[i])
      ans += graph[i][order[nvalid-1]];
  int &a = order[nvalid-2], &b = order[nvalid-1];
  for(int i=0; i<n; ++i)
    graph[a][i] = graph[i][a] = graph[a][i] + graph[b][i];
  valid[b] = false;
  --nvalid;
  return ans;
}

_inline(int minimumCut)()
{
```

```
  int curmin = inf;
  fill(valid, valid+n, true);
  nvalid = n;
  while(nvalid>1)
    curmin = min(curmin, minimumCutPhase());
  return curmin;
}

//////////////////////////////////////////////////////
// Eulerian Tour
//////////////////////////////////////////////////////
list<int> euleriantour(int start) //para multigrafos não direcionados.
{
  int v;
  bool viz;
  list<int> res;
  stack<int> dfs;
  dfs.push(start);
  while(!dfs.empty())
    {
      v=dfs.top();
      viz=0;
      for(int i=0; !viz&&i<maxn; ++i)
        if (mtx[v][i])
          {
            dfs.push(i);
            mtx[v][i]--;
            mtx[i][v]--;
            viz=1;
          }
      if(!viz)
        {
          dfs.pop();
          res.push_front(v);
        }
    }
  return res;
}
```

8. BigNum

```
////////////////////////////////////////////////////////////////////////////
// BigNum Library
////////////////////////////////////////////////////////////////////////////
// + e -: in-place; * e /: out-of-place
////////////////////////////////////////////////////////////////////////////
const int EXP = 4;      // base = 10^EXP
const int BASE = 10000; // base. Cuidado por overflows em multiplicacao.
                        // 4 eh um bom valor.
const int TAM = 2048;   // Maximo numero representavel = BASE^TAM = 10^(EXP*TAM)

struct bignum
{
int n;                        // numero de digitos
int d[TAM];
bignum(int x = 0) : n(1) { memset(d, 0, sizeof(d)); d[n++] = x; fix(); }
bignum(const char *s) : n(1) { // De um trim em s antes!
  memset(d, 0, sizeof(d));
  char sign = 1;
  if(s[0] == '-') { sign = -1; ++s; }
  char *b = strdup(s), *e = b + strlen(b);
  while(e > b)
    { *e = 0; e = max(b, e-EXP); sscanf(e, "%d", d+n); d[n++] *= sign; }
  free(b); fix();
}
bignum &fix(int m = 0) {
  n = max(m, n);
  char sign = 0;
  for(int i=1, car=0; i <= n || car && (n = i); ++i)
    { d[i]+=car; car=d[i]/BASE; d[i]%=BASE; if(d[i]) sign=(d[i]>0)?1:-1; }
  for(int i=n-1; i>0; --i)
    if(d[i] * sign < 0) { d[i] += BASE * sign; d[i+1] -= sign; }
  while(n && !d[n]) --n;
  return *this;
}
char compare(const bignum &x = 0) const {
  for(int i=max(n, x.n); i>0; --i) {
    if(d[i] < x.d[i]) return -1;
    else if(d[i] > x.d[i]) return 1;
  }
  return 0;
}
bool operator<(const bignum &x) const { return compare(x) < 0; }
bool operator==(const bignum &x) const { return compare(x) == 0; }
bool operator!=(const bignum &x) const { return compare(x) != 0; }
char *c_str() {               // dar free depois!
  char *s = (char*)malloc(EXP*n+10);
  for(int k=n-1, i=sprintf(s, "%d", d[n]); k>0; i+=sprintf(s+i, "%04d", abs(d[k--])));
  return s;
}
bignum &operator+=(const bignum &x)
{ for(int i=1; i<=x.n; ++i) d[i] += x.d[i]; return fix(x.n); }
bignum &operator-=(const bignum &x)
{ for(int i=1; i<=x.n; ++i) d[i] -= x.d[i]; return fix(x.n); }
bignum operator+(const bignum &x) { return bignum(*this) += x; }
bignum operator-(const bignum &x) { return bignum(*this) -= x; }
bignum operator-() { bignum b(0); return b -= *this; }
void ams(const bignum &x, const int &m, const int &b) {
  for(int i=1, car=0; (i <= x.n || car) && (n = i+b); ++i)
    { d[i+b] += x.d[i]*m + car; car = d[i+b]/BASE; d[i+b] %= BASE; }
}
bignum operator*(const bignum &x) const {
  bignum r(0);
  for(int i=1; i<=n; ++i) r.ams(x, d[i], i-1);
  return r;
}
```

```
  bignum &operator*=(const bignum &x) { return *this = *this * x; }
  bignum div(const bignum &x) {
    if(x == 0) throw "divisao por zero";
    bignum tr, q(0); q.n = max(n - x.n + 1, 0);
    for(int i=q.n; i>0; --i) {
      for(int l=0, u=BASE-1; u > l; ) {
        tr = 0;
        q.d[i] = (l+u+1)/2;
        tr.ams(x, q.d[i], i-1);
        if(*this < tr) u = q.d[i] - 1, q.d[i] = u;
        else l = q.d[i]; // tr <= *this
      }
      tr = 0; tr.ams(x, q.d[i], i-1); *this -= tr;
    }
    return q.fix();
  }
  bignum &operator/=(const bignum &x) { return *this = div(x); }
  bignum &operator%=(const bignum &x) { div(x); return *this; }
  bignum operator/(const bignum &x) { return bignum(*this).div(x); }
  bignum operator%(const bignum &x) { return bignum(*this) %= x; }
  bignum pow(int x) {        // Pode ser otimizado a tempo logaritmico,
                             // mas custaria mais espaco.
    if(x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
    bignum r = 1;
    for(int i=0; i<x; ++i) r *= *this;
    return r;
  }
  bignum root(int x) {
    if(compare() == 0 || compare() < 0 && x % 2 == 0) return 0;
    if(*this == 1 || x == 1) return *this;
    if(compare() < 0) return -(-*this).root(x);
    // melhorando o chute inicial
    bignum d(0);
    d.n = this->n/x + 2;
    for(int i=1; i<=d.n; ++i) d.d[i] = BASE-1;
    bignum a = 1; //, d = *this;
    while(d != 1) {
      bignum b = a + (d /= 2);
      if(compare(b.pow(x)) >= 0) { d += 1; a = b; }
    }
    return a;
  }
};
```

9. Álgebra Linear

```
////////////////////////////////////////////////////////////////////////////
// ELIMINACAO GAUSSIANA
////////////////////////////////////////////////////////////////////////////
// obtem echelon form (singular) ou row-reduced echelon form (nonsingular)
// retorna true se nonsingular, false caso contrario
struct submult_op
{
  double p;
  submult_op(const double &a) : p(a) {}
  double operator()(const double &a, const double &b) {return a-p*b;}
};
bool GaussElim(vector<vector<double> > &A)
{
  int m=A.size(),n=A[0].size();
  int i,j;
  for(i=0,j=0; i<m&&j<n;)
    {
      int maxi = i;
      for(int k=i+1; k<m; ++k) if( fabs(A[k][j]) > fabs(A[maxi][j]) ) maxi = k;
      swap(A[i], A[maxi]);
      if( fabs(A[i][j]) > eps )
        {
          transform(all(A[i]),A[i].begin(),bind2nd(divides<double>(),A[i][j]));
          for(int k=i+1; k<m; ++k)
            transform(all(A[k]),A[i].begin(),A[k].begin(),submult_op(A[k][j]));
          ++i;
        }
      ++j;
    }
  if( i == m && j == m )
    {
      for(int i=0; i<m; ++i) for(int j=0; j<i; ++j)
        transform(all(A[j]),A[i].begin(),A[j].begin(),submult_op(A[j][i]));
      return true;
    }
  return false;
}
```

10. Strings

```
////////////////////////////////////////////////////////////////////////////
// KMP O(t+p)
////////////////////////////////////////////////////////////////////////////
vector <const char *> v;
const char *kmp_search(const char *texto, const char *p)
{
  int T[2000];
  int i,j;
  const char  *result = NULL;
  if (p[0] == 0) return texto;
  T[0] = -1;
  for (i=0; p[i] != 0; i++)
    {
      T[i+1] = T[i] + 1;
      while (T[i+1] > 0 && p[i] != p[T[i+1]-1])
        T[i+1] = T[T[i+1]-1] + 1;
    }
  for (i=j=0; texto[i] != 0; )
    {
      if (j < 0 || texto[i] == p[j])
        {
          ++i, ++j;
```

```
          if (p[j] == 0)
            {
              result=texto+i-j;
              v.push_back(result);
            }
        }
      else j = T[j];
    }
  return result;
}

////////////////////////////////////////////////////////////////////////////
// Suffix Tree
////////////////////////////////////////////////////////////////////////////
const int maxn = 100100; //10^5
char str[maxn];
struct node
{
  int st, ed;
  map<char,int> pt;
  int link;
};
node vet[2*maxn];
int total;
_inline(void canonize)(int &s, int &k, int p)
{
  if (k<=p)
    {
      int ss=vet[s].pt.find(str[k])->second,
        kk=vet[ss].st, pp=vet[ss].ed;
      while(pp-kk<=p-k)
        {
          k+=pp-kk+1;
          s=ss;
          if (k<=p)
            ss=vet[s].pt.find(str[k])->second,
              kk=vet[ss].st, pp=vet[ss].ed;
        }
    }
}
_inline(void init)(node &x, int st, int ed) {x.st=st; x.ed=ed; x.pt.clear();}
void make_tree()
{
  int s=0, oldr, rr, r, ss, k=0, kk;
  map<char,int>::iterator mi;
  for(int i=0; (!i)||str[i-1]; ++i)
    {
      oldr=0;
      while(1)
        {
          if (k<i)
            {
              ss=(mi=vet[s].pt.find(str[k]))->second;
              kk=vet[ss].st;
              if (str[i]==str[kk+i-k])
                break;
              else
                {
                  init(vet[r=total++], kk, kk+i-k-1);
                  vet[r].pt.insert(make_pair(str[kk+i-k],ss));
                  mi->second=r;
                  vet[ss].st+=i-k;
                }
            }
          else if (vet[s].pt.find(str[i])==vet[s].pt.end())
            r=s;
          else
            break;
          init(vet[rr=total++], i,inf);
          vet[r].pt.insert(make_pair(str[i],rr));
          if (oldr)
            vet[oldr].link=r;
          oldr=r;
          if (!s)
            ++k;
          s=vet[s].link;
          canonize(s, k, i-1);
        }
      if (oldr)
        vet[oldr].link=s;
      canonize(s,k,i);
    }
}
```

11. Binary Indexed Tree

```
#define MAXN //N+1, pois os indices comecam em 1.
int bit1[MAXN]; //update em um elemento, query no intervalo.

void update_elem(int pos, int diff)
{
  if (pos)
    for(;pos<=N; pos+=pos& -pos)
      bit1[pos]+=diff;
}

int query_int(int pos) // [1,pos]
{
  int res=0;
  for(;pos>0; pos-=pos& -pos)
    res+=bit1[pos];
  return res;
}

int bit2[MAXN]; //update no intervalo, query no elemento

void update_int(int pos, int diff) // [1,pos]
```

```
{
  for(;pos>0;pos-=pos& -pos)
    bit2[pos]+=diff;
}

int query_pos(int pos)
{
  int res=0;
  if (pos)
    for(;pos<=N; pos+=pos& -pos)
      res+=bit2[pos];
  return res;
}

////////////////////////////////////////////////////////////////////////////////
// 2D
////////////////////////////////////////////////////////////////////////////////
#define MAXN 32
const int N = MAXN/2;
const int root = MAXN-1;
inline int pai(int v) {return N+(v>>1);}
int BIT[MAXN][MAXN]; //[y][x]
void update1(int p, int lx, int ux, int v)
{
  for(;lx<=ux;lx=pai(lx),ux=pai(ux))
    {
      if (lx&1)
        BIT[p][lx++]+=v;
      if (!(ux&1))
        BIT[p][ux--]+=v;
    }
}
void update2(int ly, int uy, int lx, int ux, int v)
{
  for(;ly<=uy;ly=pai(ly),uy=pai(uy))
    {
      if (ly&1)
        update1(ly++, lx, ux, v);
      if (!(uy&1))
        update1(uy--, lx, ux, v);
    }
}
int query(int y, int x)
{
  int r=0;
  for(;x<root;x=pai(x))
    for(int t=y; t<root; t=pai(t))
      r+=BIT[t][x];
  return r;
}
```

12. Interval Tree

```
template <int n>
class itree // (expt 2 14)16384 > 10^5
{
  int fim;
  int it[n<<1];
  int (*func)(int,int);
  int sentinel;
  inline int pai(int p) { return n|(p>>1); }
public:
  itree(int(*f)(int,int), int s)
  {
    func=f;
    sentinel=s;
    fim=(n<<1)-1;
  }
  void update(int pos, int val)
  {
    for(;pos<fim;pos=pai(pos))
      it[pos]=func(it[pos],val);
  }
  int query(int left, int right)
  {
    int res=sentinel;
    for(;left<=right;left=pai(left), right=pai(right))
      {
        if (left&1)
          res=func(res, it[left++]);
        if (!(right&1))
          res=func(res, it[right--]);
      }
    return res;
  }
  void clear()
  {
    for(int i=0; i<=fim; ++i)
      it[i]=sentinel;
  }
};
```

13. Problemas

```
////////////////////////////////////////////////////////////////////////////////
// GEOMETRIA 3D
////////////////////////////////////////////////////////////////////////////////

////
// PAPERWEIGHT: convex hull face, distances
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <set>
```

```
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

const long double eps = 1e-9;

struct point
{
  long double x, y, z;
};

point cross(point a, point b)
{
  return (point){
    a.y*b.z - a.z*b.y,
      a.z*b.x - a.x*b.z,
      a.x*b.y - a.y*b.x};
}

long double dot(point a, point b)
{
  return a.x*b.x + a.y*b.y + a.z*b.z;
}

long double mixedproduct(point a, point b, point c)
{
  return dot(a, cross(b, c));
}

point operator-(const point &a, const point &b)
{
  return (point){a.x - b.x, a.y - b.y, a.z - b.z};
}

point operator+(const point &a, const point &b)
{
  return (point){a.x + b.x, a.y + b.y, a.z + b.z};
}

point operator/(const point &a, long double k)
{
  return (point){a.x / k, a.y / k, a.z / k};
}

point operator*(const point &a, long double k)
{
  return (point){a.x * k, a.y * k, a.z * k};
}

long double norm(point p) { return sqrt(dot(p, p)); }

bool ishullface(const vector<point> &poly, int a, int b, int c)
{
  long double mp[6];
  int pos=0,neg=0;
  for (int i=0; i<5; ++i)
    mp[i]=mixedproduct(poly[b] - poly[a], poly[c] - poly[a], poly[i] - poly[a]);
  for (int i=0;i<5;i++)
    {
      if (mp[i]<-eps)
        neg++;
      else if (mp[i]>eps)
        pos++;
    }
  if (neg*pos == 0)
    {
      //printf("ishullface(%d %d %d) = true\n", a, b, c);
      return true;
    }
  else
    {
      //printf("ishullface(%d %d %d) = false\n", a, b, c);
      return false;
    }
}

long double volume(point a, point b, point c, point d)
{
  return fabs(mixedproduct(b - a, c - a, d - a));
}

point cmass(point a, point b, point c, point d)
{
  return (a + b + c + d) / 4;
}

point totalcmass(const vector<point> &poly)
{
  point CM1 = cmass(poly[0], poly[1], poly[2], poly[3]);
  long double VM1 = volume(poly[0], poly[1], poly[2], poly[3]);
  point CM2 = cmass(poly[0], poly[1], poly[2], poly[4]);
  long double VM2 = volume(poly[0], poly[1], poly[2], poly[4]);
  return (CM1 * VM1 + CM2 * VM2) / (VM1 + VM2);
}

bool isinside(point p, point a, point b, point c)
{
  if (fabs(norm(cross(p - a, b - a)) + norm(cross(p - b, c - b))
        + norm(cross(p - c, a - c)) - norm(cross(b - a, c - a))) < eps)
    return true;
  else
    return false;
}

point project(point p, point a, point b, point c)
```

```
{
  point v = cross(b - a, c - a);
  return p - v * dot(p-a, v) / dot(v, v);
}

long double pointlinedist(point p, point a, point b)
{
  return norm(cross(p - a, p - b)) / norm(b - a);
}

bool isreallyinside(point p, point a, point b, point c)
{
  if (pointlinedist(p, a, b) > .2 && pointlinedist(p, b, c) > .2
      && pointlinedist(p, a, c) > .2)
    return true;
  else
    return false;
}

bool isreallyinside(point p, point a, point b, point c, point d)
{
  if (pointlinedist(p, a, c) > .2 && pointlinedist(p, b, c) > .2
      && pointlinedist(p, a, d) > .2 && pointlinedist(p, b, d) > .2)
    return true;
  else
    return false;
}

bool isstable(const vector<point> &poly, int a, int b, int c)
{
  point tcmass = totalcmass(poly);
  point P = project(tcmass, poly[a], poly[b], poly[c]);
  if (isinside(P, poly[a], poly[b], poly[c])
      && isreallyinside(P, poly[a], poly[b], poly[c]))
    return true;
  else
    return false;
}

/// novo codigo
bool isstable(const vector<point> &poly, int a, int b, int c, int d)
{
  int other = 1 + 2 + 3 + 4 - a - b - c - d;
  point tcmass = totalcmass(poly);
  point P = project(tcmass, poly[a], poly[b], poly[c]);
  if ((isinside(P, poly[a], poly[c], poly[d])
       || isinside(P, poly[b], poly[c], poly[d]))
      && isreallyinside(P, poly[a], poly[b], poly[c], poly[d]))
    return true;
  else
    return false;
}
```

```
long double pointplanedist(point p, point a, point b, point c)
{
  return norm(p - project(p, a, b, c));
}

int main()
{
  vector<point> points;
  point F;

  for (int ncase = 1;;++ncase)
    {
      points.clear();
      for (int i=0; i<5; ++i)
        {
          double x, y, z;
          if (scanf(" %lf %lf %lf", &x, &y, &z) != 3)
            return 0;
          points.push_back((point){x, y, z});
        }

      scanf(" %Lf %Lf %Lf", &F.x, &F.y, &F.z);

      long double mindist = 1e100, maxdist = -1;

      for (int i=0; i<5; ++i)
        for (int j=0; j<i; ++j)
          for (int k=0; k<j; ++k)
            if (ishullface(points, i, j, k) && isstable(points, i, j, k))
              {
                //printf("(%d %d %d)\n", i, j, k);
                mindist = min(mindist,
                    pointplanedist(F, points[i], points[j], points[k]));
                maxdist = max(maxdist,
                    pointplanedist(F, points[i], points[j], points[k]));
              }

      for (int i=0; i<5; ++i)
        for (int j=0; j<i; ++j)
          for (int k=0; k<j; ++k)
            for (int l=0; l<k; ++l)
              if (volume(points[i], points[j], points[k], points[l]) < eps
                  && isstable(points, i, j, k, l))
              {
                //printf("(%d %d %d)\n", i, j, k);
                mindist = min(mindist,
                    pointplanedist(F, points[i], points[j], points[k]));
                maxdist = max(maxdist,
                    pointplanedist(F, points[i], points[j], points[k]));
              }

      printf("Case %d: %.5Lf %.5Lf\n", ncase, mindist, maxdist);
    }
```

```
  return 0;
}

////
// GUERRA
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cassert>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <list>
#include <deque>
#include <queue>
#include <stack>
#include <functional>
#include <sstream>
#include <iostream>
#include <ctime>
#include <algorithm>
using namespace std;

#define all(v) (v).begin(), (v).end()
#define _foreach(it, b, e) for (__typeof__(b) it = (b); it != (e); ++it)
#define foreach(x...) _foreach(x)

typedef long long huge;

const int inf = 0x3f3f3f3f;
const huge hugeinf = 0x3f3f3f3f3f3f3f3fll;
const double eps = 1e-9;
struct point
{
  long double x, y, z;
  point(long double x=0, long double y=0, long double z=0) :x(x), y(y), z(z) {}
  inline point operator+(const point &p) const {return point(x+p.x, y+p.y, z+p.z);}
  inline point operator-(const point &p) const {return point(x-p.x, y-p.y, z-p.z);}
  inline point operator*(long double p) const {return point(x*p, y*p, z*p); }
  inline bool operator==(const point &p) const
  {return fabsl(x - p.x) < eps && fabsl(y - p.y) < eps && fabsl(z - p.z) < eps; }
};
inline long double dot(const point &a, const point &b)
{
  return a.x*b.x+a.y*b.y+a.z*b.z;
}
inline point cross(const point &a, const point &b)
{
  return point(a.y*b.z-a.z*b.y,
               a.z*b.x-a.x*b.z,
               a.x*b.y-a.y*b.x);
}

point proj(point R, point P) // projeta P sobre R
{
  return R * (dot(P, R) / dot(R, R));
}

long double abs(point P)
{
  return sqrt(dot(P, P));
}

bool between(point A, point B, point P) // inclusive, assume colinear
{
  return dot(P - A, P - B) < eps;
}

long double linedist(point A, point B, point P)
{
  point Pproj = proj(B - A, P - A) + A;
  if (between(A, B, Pproj))
    return abs(P - Pproj);
  else
    return min(abs(P - A), abs(P - B));
}

long double pointtri(point A, point B, point C, point P)
{
  point X = B - A;
  point Y = C - A;
  point Pori = P;
  P = P - A;
  point PP = P - proj(cross(X, Y), P);
  point PPP = PP + A;
  point R1 = cross(B - A, PPP - A);
  point R2 = cross(C - B, PPP - B);
  point R3 = cross(A - C, PPP - C);
  if (dot(R1, R2) > -eps && dot(R2, R3) > -eps && dot(R1, R3) > -eps)
    return abs(Pori - PPP);
  else
    return min(linedist(A, B, Pori), min(linedist(B, C, Pori),
                                         linedist(C, A, Pori)));
}

int ccw(point A, point B, point C, point Ref)
{
  double k = dot(cross(B - A, C - A), Ref);
  if (k > eps) return 1;
  if (k < -eps) return -1;
  if (dot(C - A, B - A) < -eps) return -1;
  if (dot(A - B, C - B) < -eps) return 1;
  return 0;
}
```

```
bool intersect(point A, point B, point C, point D)
{
  point Ref;
  point X[4] = {A, B, C, D};
  for (int i=0; i<4; ++i)
    for (int j=0; j<4; ++j)
      for (int k=0; k<4; ++k)
        {
          point RC = cross(X[j] - X[i], X[k] - X[i]);
          if (abs(RC) > eps)
            Ref = RC;
        }
  assert(abs(Ref) > eps);
  return (ccw(A, B, C, Ref) * ccw(A, B, D, Ref) <= 0)
         && (ccw(C, D, A, Ref) * ccw(C, D, B, Ref) <= 0);
}

long double segseg(point A, point B, point C, point D)
{
  point E = proj(cross(D - C, B - A), A - D);
  point Cl = C + E;
  point Dl = D + E;
  if (abs(cross(D - C, B - A)) > eps && intersect(A, B, Cl, Dl))
    return abs(E);
  else
    return min(min(abs(A - C), abs(A - D)), min(abs(B - C), abs(B - D)));
}

void read(point &P)
{
  long double x, y, z;
  scanf(" %Lf %Lf %Lf", &x, &y, &z);
  P = point(x, y, z);
}

int main()
{
  int ntests;
  scanf(" %d", &ntests);
  while (ntests--)
    {
      point T[2][4];
      for (int i=0; i<2; ++i)
        for (int j=0; j<4; ++j)
          read(T[i][j]);
      long double bestdist = 0x3f3f3f3f3f3f3f3fll;
      for (int i=0; i<2; ++i)
        {
          for (int j=0; j<4; ++j)
            {
              point P = T[0][j];
              bestdist = min(bestdist,
                             pointtri(T[1][0], T[1][1], T[1][2], P));
              bestdist = min(bestdist,
                             pointtri(T[1][0], T[1][1], T[1][3], P));
              bestdist = min(bestdist,
                             pointtri(T[1][0], T[1][2], T[1][3], P));
              bestdist = min(bestdist,
                             pointtri(T[1][1], T[1][2], T[1][3], P));
            }
          for (int j=0; j<4; ++j)
            for (int l=j+1; l<4; ++l)
              for (int m=0; m<4; ++m)
                for (int n=m+1; n<4; ++n)
                  bestdist = min(bestdist,
                                 segseg(T[0][j], T[0][l], T[1][m], T[1][n]));
          for (int j=0; j<4; ++j)
            swap(T[0][j], T[1][j]);
        }
      printf("%.2Lf\n", bestdist);
    }
  return 0;
}

////////////////////////////////////////////////////////////////////////////
// AHO-CORASICK
////////////////////////////////////////////////////////////////////////////
#define maxc 1000000
#define maxn 10000
struct link {
  int v;
  link *n;
} out[maxn], *l;
struct node {
  node *s, *b, *f;
  link *o;
  char i;
} rt[maxc], *p, *q, *r, *s;
char buf[maxc+maxn], mtx[maxn], *str;
int word[maxn], nn, nm, n;
void push_str(int i)
{
  for(str=buf+word[i], p=rt; *str; ++str) {
    for(q=p->s; q; q=q->b)
      if (q->i==*str)
        break;
    if (q) p=q;
    else {
      q=p->s;
      p=(p->s=rt+nn++);
      p->i=*str; p->b=q; p->s=0; p->o=0; p->f=rt;
    }
  }
  (p->o=out+nm++)->v=i;
}
```

```
void fix_tree()
{
  queue<node*> td;
  for(r=rt->s; r; r=r->b)
    td.push(r);
  while(!td.empty()) {
    p=td.front();td.pop();
    for(r=p->s; r; r=r->b)
      for(q=p, td.push(r); q!=rt;)
        for(q=q->f, s=q->s; s; s=s->b)
          if (s->i==r->i) { r->f=s; q=rt; break; }
    if (p->o) p->o->n=p->f->o;
    else p->o=p->f->o;
  }
}
void match_str(int i)
{
  for(str=buf+word[i], p=rt; *str; ++str) {
    for(q=p->s; p!=rt||q; q=q->b) {
      for(; !q; p=p->f, q=p->s);
      if (q->i==*str) {        p=q; break; }
    }
    for(l=p->o; l&&!mtx[l->v]; l=l->n)
      mtx[l->v]=1;
  }
}

////////////////////////////////////////////////
// Colorfull Spanning Tree
////////////////////////////////////////////////
const int maxn = 210;
list<int> mtx[maxn][maxn];
list<pair<int,int> > adj[maxn];
bool expande[maxn];
pair<int,int> aresta[maxn];
bool e_marcada[maxn];
bool v_marcado[maxn];
bool temp[maxn];
bool pode_adicionar;
int n, k;
void adiciona_aresta(int x)
{
  v_marcado[aresta[x].second]=true;
  adj[aresta[x].first].push_back(make_pair(aresta[x].second,x));
  adj[aresta[x].second].push_back(make_pair(aresta[x].first,x));
}
void remove_aresta(int a, int b)
{
  foreach(it, all(adj[a]))
    if (it->first==b)
      {
        adj[a].erase(it);
        break;
      }
  foreach(it, all(adj[b]))
    if (it->first==a)
      {
        adj[b].erase(it);
        break;
      }
}
bool dfs(int v, int root)
{
  temp[v]=true;
  if (pode_adicionar)
    foreach(it, all(mtx[v][root]))
      if (!e_marcada[*it])
        {
          e_marcada[*it]=true;
          adj[root].push_back(make_pair(v,*it));
          adj[v].push_back(make_pair(root,*it));
          temp[v]=false;
          return true;
        }
  foreach (it, all(adj[v]))
    if (!temp[it->first])
      {
        if (pode_adicionar||!expande[it->second])
          {
            if (dfs(it->first,root))
              {
                temp[v]=false;
                return true;
              }
          }
        else
          {
            pode_adicionar=true;
            if (dfs(it->first,root))
              {
                adiciona_aresta(it->second);
                remove_aresta(v,it->first);
                temp[v]=false;
                pode_adicionar=false;
                return true;
              }
            pode_adicionar=false;
          }
      }
  temp[v]=false;
  return false;
}
bool augment()
{
  for(int i=0; i<k; ++i)
    expande[i]=false;
```

```
  for(int i=0; i<n; ++i)
    if (v_marcado[i])
      for(int j=0; j<n; ++j)
        if (!v_marcado[j])
          foreach (it, all(mtx[i][j]))
            {
              if (!e_marcada[*it])
                {
                  e_marcada[*it]=true;
                  adj[i].push_back(make_pair(j,*it));
                  adj[j].push_back(make_pair(i,*it));
                  v_marcado[j]=true;
                  return true;
                }
              if (!expande[*it])
                {
                  expande[*it]=true;
                  aresta[*it]=make_pair(i,j);
                }
            }
  for(int i=0; i<n; ++i)
    if (v_marcado[i])
      if (dfs(i,i))
        return true;
  return false;
}
void print_selected()
{
  for(int i=0; i<n; ++i)
    {
      printf("%d:", i+1);
      foreach(it, all(adj[i]))
        printf(" (%d,%d)", it->first+1, it->second+1);
      printf("\n");
    }
  printf("-\n");
}
int main()
{
  int a, b, c, m;
  bool ok;
  srand(time(NULL));
  for(int i=0; i<maxn; ++i)
    temp[i]=false;
  pode_adicionar=false;
  for(int teste=1;;++teste)
    {
      if (scanf(" %d %d %d", &n, &m, &k)!=3)
        return 0;
      for(int i=0; i<n; ++i)
        {
          for(int j=0; j<n; ++j)
            mtx[i][j].clear();
          v_marcado[i]=false;
          adj[i].clear();
        }
      for(int i=0; i<k; ++i)
        e_marcada[i]=false;
      for(int i=0; i<m; ++i)
        {
          scanf(" %d %d %d", &a, &b, &c);
          --a;--b;--c;
          if (a!=b)
            {
              mtx[a][b].push_back(c);
              mtx[b][a].push_back(c);
            }
        }
      printf("Instancia %d\n", teste);
      v_marcado[0]=true;
      ok=1;
      for(int i=1; ok&&i<n; ++i)
        {
          if (!augment())
            ok=0;
          //print_selected();
        }
      if (ok)
        printf("sim\n\n");
      else
        printf("nao\n\n");
    }
}

////////////////////////////////////////////
// Multiplicação de Nímeros O(n^2)
////////////////////////////////////////////
const int maxn=200;
const int lim=16;
int tab[maxn][maxn];
bool used[maxn];
void multiplica(int a, int b)
{
  for(int i=0; i<maxn; ++i)
    used[i]=0;
  for(int i=0; i<a; ++i)
    for(int j=0; j<b; ++j)
      used[tab[i][b]^tab[a][j]^tab[i][j]]=true;
  for(int i=0; ;++i)
    if (!used[i]) {
      tab[a][b]=i;
      return;
    }
}
```

A		**G**	
B		**H**	
C		**I**	
D		**J**	
E		**K**	
F		**L**	