[UFMG] TRUPE DA BIOLOGIA (2017-18)

Contents

1 InContests

1.1 Makefile

```
CXX=g++
CXXFLAGS=-std=c++11 -Wall

SRC=$(*.cpp)
OBJ=$(SRC:%.cpp=%)
```

1.2 Vimrc

```
set ts=2 si ai sw=2 number mouse=a
syntax on
```

1.3 Template

```
#include <bits/stdc++.h>
using namespace std;
#define sc(a) scanf("%d", &a)
#define sc2(a, b) scanf("%d%d", &a, &b)
#define sc3(a, b, c) scanf("%d%d%d", &a, &b, &c)
#define pri(x) printf("%d\n", x)
#define prie(x) printf("%d ", x)
#define mp make_pair
#define pb push_back
#define BUFF ios::sync_with_stdio(false);
#define db(x) cerr << #x << " == " << x << endl
typedef long long int ll;
typedef long double ld;
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
const int INF = 0x3f3f3f3f;
const ld pi = acos(-1);
```

2 Graph Algorithms

2.1 2 SAT

```
/* Supondo que cada vertice u, o seu
 * positivo e 2*u,  e negativo e 2*i+1
 * resposta[i]=0, significa que o positivo de i e resposta
 * resposta[i]=1, significa que o negativo de i e resposta
 * chamar Sat(n) , n e o numero de vertices do grafo
 * contando com os negativos .. na maioria dos problemas
 * chamar 2*n;
 * testado em :http://codeforces.com/contest/781/problem/D
 * */
int resposta[N];
vi graph[N], rev[N];
int us[N];
stack<int> pilha;
void dfs1(int u)
{
    us[u] = 1;
    for (int v : graph[u])
        if (!us[v]) dfs1(v);
    pilha.push(u);
}
void dfs2(int u, int color)
{
    us[u] = color;
    for (int v : rev[u])
        if (!us[v]) dfs2(v, color);
}
int Sat(int n)
{
    for (int i = 0; i < n; i++)
        if (!us[i]) dfs1(i);
    int color = 1;
    memset(us, 0, sizeof(us));
    while (!pilha.empty()) {
        int topo = pilha.top();
        pilha.pop();
        if (!us[topo]) dfs2(topo, color++);
    }
    for (int i = 0; i < n; i += 2) {
        if (us[i] == us[i + 1]) return 0;
        resposta[i / 2] = (us[i] < us[i + 1]);
    }
    return 1;
}
inline void add(int u, int v)
{
    graph[u].pb(v);
    rev[v].pb(u);
}
```

2.2 Kosaraju

```
//Retorna os componentes fortemente conectados
//Se o usados[i]=usados[j], temos que i e j
//pertencem ao mesmo componente, col-1= numero
//de componentes fortemente conectados do grafo
class kosaraju {
 private:
  vi usados;
  vvi graph;
  vvi trans;
  vi pilha;

 public:
  kosaraju(int N)
  {
    graph.resize(N);
    trans.resize(N);
  }
  void AddEdge(int u, int v)
  {
    graph[u].pb(v);
    trans[v].pb(u);
  }
  void dfs(int u, int pass, int color)
  {
    usados[u] = color;
    vi vizinhos;
    if (pass == 1)
      vizinhos = graph[u];
    else
      vizinhos = trans[u];
    for (int j = 0; j < vizinhos.size(); j++) {
      int v = vizinhos[j];
      if (usados[v] == 0) {
        dfs(v, pass, color);
      }
    }
    pilha.pb(u);
  }
  int SSC(int n)
  {
    pilha.clear();
    usados.assign(n, 0);
    for (int i = 0; i < n; i++) {
      if (!usados[i]) dfs(i, 1, 1);
    }
    usados.assign(n, 0);
    int color = 1;
    for (int i = n - 1; i >= 0; i--) {
      if (usados[pilha[i]] == 0) {
        dfs(pilha[i], 2, color);
        color++;
      }
    }
    return color - 1;
  }
};
```

2.3 LCA

```
//antes de usar as queries de lca, e etc..
//certifique-se de chamar a dfs da arvore e
//process()
const int N = 100000;
const int M = 22;
int P[N][M];
int big[N][M], low[N][M], level[N];
vii graph[N];
int n;

void dfs(int u, int last, int l)
{
  level[u] = l;
  P[u][0] = last;
  for (ii v : graph[u])
    if (v.first != last) {
      big[v.first][0] = low[v.first][0] = v.second;
      dfs(v.first, u, l + 1);
    }
}

void process()
{
  for (int j = 1; j < M; j++)
    for (int i = 1; i <= n; i++) {
      P[i][j] = P[P[i][j - 1]][j - 1];
      big[i][j] = max(big[i][j - 1], big[P[i][j - 1]][j - 1]);
      low[i][j] = min(low[i][j - 1], low[P[i][j - 1]][j - 1]);
    }
}

int lca(int u, int v)
{
  if (level[u] < level[v]) swap(u, v);
  for (int i = M - 1; i >= 0; i--)
    if (level[u] - (1 << i) >= level[v]) u = P[u][i];
  if (u == v) return u;
  for (int i = M - 1; i >= 0; i--) {
    if (P[u][i] != P[v][i]) u = P[u][i], v = P[v][i];
  }
  return P[u][0];
}

int maximum(int u, int v, int x)
{
  int resp = 0;
  for (int i = M - 1; i >= 0; i--)
    if (level[u] - (1 << i) >= level[x]) {
      resp = max(resp, big[u][i]);
      u = P[u][i];
    }
  for (int i = M - 1; i >= 0; i--)
    if (level[v] - (1 << i) >= level[x]) {
      resp = max(resp, big[v][i]);
      v = P[v][i];
    }
  return resp;
}

int minimum(int u, int v, int x)
{
  int resp = INF;
  for (int i = M - 1; i >= 0; i--)
    if (level[u] - (1 << i) >= level[x]) {
      resp = min(resp, low[u][i]);
```

```
    u = P[u][i];
  }
  for (int i = M - 1; i >= 0; i--)
    if (level[v] - (1 << i) >= level[x]) {
      resp = min(resp, low[v][i]);
      v = P[v][i];
    }
  return resp;
}
```

2.4 Bridges and Articulation Points

```
class ponte {
 private:
  vvi graph;
  vi usados;
  vi e_articulacao;
  vi dfs_low;
  vi dfs_prof;
  vector<ii> pontes;
  int tempo;

 public:
  ponte(int N)
  {
    graph.clear();
    graph.resize(N);
    usados.assign(N, 0);
    dfs_low.assign(N, 0);
    dfs_prof.assign(N, 0);
    e_articulacao.assign(N, 0);
    tempo = 0;
  }
  void AddEdge(int u, int v)
  {
    graph[u].pb(v);
    graph[v].pb(u);
  }
  void dfs(int u, int pai)
  {
    usados[u] = 1;
    int nf = 0;
    dfs_low[u] = dfs_prof[u] = tempo++;
    for (int v : graph[u]) {
      if (!usados[v]) {
        dfs(v, u);
        nf++;
        if (dfs_low[v] >= dfs_prof[u] and pai != -1) e_articulacao[u] = true;
        if (pai == -1 and nf > 1) e_articulacao[u] = true;
        if (dfs_low[v] > dfs_prof[u]) pontes.pb(mp(u, v));
        dfs_low[u] = min(dfs_low[u], dfs_low[v]);
      }
      else if (v != pai)
        dfs_low[u] = min(dfs_low[u], dfs_prof[v]);
    }
  }
  void olha_as_pontes()
  {
    for (int i = 0; i < graph.size(); i++)
      if (!usados[i]) dfs(i, -1);
    if (pontes.size() == 0)
      cout << " Que merda! nao tem ponte!" << endl;
    else {
      for (ii i : pontes) cout << i.first << " " << i.second << endl;
    }
  }
  void olha_as_art()
  {
    for (int i = 0; i < graph.size(); i++)
      if (!usados[i]) dfs(i, -1);
    for (int i = 0; i < e_articulacao.size(); i++)
      if (e_articulacao[i]) cout << " OIAAA A PONTE " << i << endl;
  }
}
```

2.5 Eulerian Tour

```
multiset<int> graph[N];
stack<int> path;
// -> It suffices to call dfs1 just
// one time leaving from node 0.
// -> To calculate the path,
// call the dfs from the odd degree node.
// -> O(n * log(n))
void dfs1(int u)
{
  while(graph[u].size())
  {
    int v = *graph[u].begin();
    graph[u].erase(graph[u].begin());
    graph[v].erase(graph[v].find(u));
    dfs1(v);
  }
  path.push(u);
}
```

2.6 Floyd Warshall

```
//menor caminho para todos os vertices
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    if (graph[i][j] != INF) pai[i][j] = i;

for (int k = 0; k < n; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (graph[i][j] > graph[i][k] + graph[k][j]) {
        graph[i][j] = graph[i][k] + graph[k][j];
        pai[i][j] = pai[k][j];
      }
    }
  }
}
```

2.7 Closest Pair of Points

```
double INF = 1e100;
double EPS = 1e-12;

struct PT {
  ll x, y;
  PT() {}
  PT(ll x, ll y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
};

ll dist2(PT p, PT q)   { return (p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y); }

int n;
PT pts[100005];
int id[100005];

bool cmpx(const int &a, const int &b) {
  return pts[a].x < pts[b].x;
}

bool cmpy(const int &a, const int &b) {
  return pts[a].y < pts[b].y;
}

pair<ll, ii> getStrip(vi &strip, ll dmax) {
  sort(strip.begin(), strip.end(), cmpy);
  pair<ll, ii> ret = mp(LINF, mp(-1, -1));
  int id1, id2;
  ll delta;
  for(int i = 0; i < strip.size(); i++) {
    id1 = strip[i];
    for(int j = i + 1; j < strip.size(); j++) {
      id2 = strip[j];
      delta = pts[id1].y - pts[id2].y;
      if(delta * delta > dmax) break;
      ret = min(ret, mp(dist2(pts[id1], pts[id2]), mp(id1, id2)));
    }
  }
  return ret;
}

pair<ll, ii> solve(int b, int e) {
  if(b >= e) return mp(LINF, mp(-1, -1));
  int mid = (b + e) / 2;
  ll xsplit = pts[id[mid]].x;
  pair<ll, ii> p1 = solve(b, mid), p2 = solve(mid + 1, e);
  pair<ll, ii> ret = min(p1, p2);
  ll dmax = ret.first;
  vi strip;
  ll delta;
```

```
  for(int i = mid; i <= e; i++) {
    int idx = id[i];
    delta = pts[idx].x - xsplit;
    if(delta * delta > dmax) break;
    strip.pb(idx);
  }
  for(int i = mid - 1; i >= b; i--) {
    int idx = id[i];
    delta = xsplit - pts[idx].x;
    if(delta * delta > dmax) break;
    strip.pb(idx);
  }
  pair<ll, ii> aux = getStrip(strip, dmax);
  return min(aux, ret);
}

int main() {
  BUFF;
  cin >> n;
  for(int i = 0; i < n; i++) {
    cin >> pts[i].x >> pts[i].y;
    id[i] = i;
  }
  sort(id, id + n, cmpx);
  pair<ll, ii> ans = solve(0, n - 1);
  if(ans.second.first > ans.second.second) swap(ans.second.first, ans.second.second);
  cout << setprecision(6) << fixed;
  cout << ans.second.first << " " << ans.second.second << " " << sqrt(ans.first) << endl;
  return 0;
}
```

2.8 Centroid Decomposition Example

```
/*
        MUST CALL DECOMP(1,-1) FOR A 1-BASED GRAPH
*/

vi g[MAXN];
int forb[MAXN];
int sz[MAXN];
int pai[MAXN];
int n, m;
unordered_map<int, int> dist[MAXN];
void dfs(int u, int last) {
  sz[u] = 1;
  for(int v : g[u]) {
    if(v != last and !forb[v]) {
      dfs(v, u);
      sz[u] += sz[v];
    }
  }
}

int find_cen(int u, int last, int qt) {
  int ret = u;
  for(int v : g[u]) {
    if(v == last or forb[v]) continue;
    if(sz[v] > qt / 2) return find_cen(v, u, qt);
  }
  return ret;
}

void getdist(int u, int last, int cen) {
  for(int v : g[u]) {
    if(v != last and !forb[v]) {
      dist[cen][v] = dist[v][cen] = dist[cen][u] + 1;
      getdist(v, u, cen);
    }
  }
}


void decomp(int u, int last) {
  dfs(u, -1);
  int qt = sz[u];
  int cen = find_cen(u, -1, qt);
  forb[cen] = 1;
  pai[cen] = last;
  dist[cen][cen] = 0;
  getdist(cen, -1, cen);
  for(int v : g[cen]) {
    if(!forb[v]) {
      decomp(v, cen);
    }
  }
}
```

```
int main() {
  sc2(n, m);

  for(int i = 0; i < n - 1; i++) {
    int a, b;
    sc2(a, b);
    g[a].pb(b);
    g[b].pb(a);
  }

  decomp(1, -1);
  return 0;
}
```

3 Strings

3.1 Aho Corasick

```
//N= tamanho da trie, M tamanho do alfabeto
int to[N][M], Link[N], fim[N];
int idx = 1;
void add_str(string &s)
{
  int v = 0;
  for (int i = 0; i < s.size(); i++) {
    if (!to[v][s[i]]) to[v][s[i]] = idx++;
    v = to[v][s[i]];
  }
  fim[v] = 1;
}

void process()
{
  queue<int> fila;
  fila.push(0);
  while (!fila.empty()) {
    int cur = fila.front();
    fila.pop();
    int l = Link[cur];
    fim[cur] |= fim[l];
    for (int i = 0; i < 200; i++) {
      if (to[cur][i]) {
        if (cur != 0) {
          Link[to[cur][i]] = to[l][i];
        }
        else
          Link[to[cur][i]] = 0;
        fila.push(to[cur][i]);
      }
      else {
        to[cur][i] = to[l][i];
      }
    }
  }
}

int resolve(string &s)
{
  int v = 0, r = 0;
  for (int i = 0; i < s.size(); i++) {
    v = to[v][s[i]];
    if (fim[v]) r++, v = 0;
  }
  return r;
}
```

3.2 KMP

```
int p[N];
int n;
void process(vi &s)
{
    int i = 0, j = -1;
    p[0] = -1;
    while (i < s.size()) {
        while ( j >= 0 and s[i] != s[j] ) j = p[j];
        i++, j++;
        p[i] = j;
    }
}
```

```cpp
// s=texto , t= padrao
int match(string &s, string &t)
{
    int ret = 0;
    process(t);
    int i = 0, j = 0;
    while (i < s.size()) {
        while (j >= 0 and  (s[i] != t[j] ) ) j = p[j];
        i++, j++;
        if (j == t.size()) {
            j = p[j];
            ret++;
        }
    }
    return ret;
}
```

3.3 Hashing

```cpp
//Certificar que os valores da string correspondente se encontrem
//entre 1 - x, x e o valor maximo. B    um menor primo maior que x.
struct Hashing{
    vector<ull> h, eleva;
    ull B;
    const string &s;
    Hashing(const string &s, ull B) : s(s), h(s.size()), eleva(s.size()){
        eleva[0] =1;
        for(int i=1;i<s.size();i++) eleva[i] = eleva[i-1]*B;
        ull hp=0;
        for(int i=0;i<s.size();i++){
            hp = hp*B + s[i];
            h[i] = hp;
        }
    }
    ull getHash(int i, int j){
        if(i==0) return h[j];
        return h[j] - h[i-1]*eleva[j-i+1];
    }
};
```

3.4 Suffix Array

```cpp
/*
 * O(nlog^2(n)) para o sufix array
 * O(logn) para o LCP(i,j)
 * LCP de i para j;
 */
struct SA {
  const int L;
  string s;
  vvi P;
  vector<pair< ii,int> > M;

  SA(const string &s) : L(s.size()), s(s), P(1, vi(L, 0)), M(L) {
    for (int i = 0; i < L; i++) P[0][i] =s[i]-'a';
    for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
      P.pb(vi(L, 0));
      for (int i = 0; i < L; i++)
        M[i] = mp(mp(P[level-1][i], i + skip < L ? P[level-1][i + skip] : -1000), i);
      sort(M.begin(), M.end());
      for (int i = 0; i < L; i++)
        P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[level][M[i-1].second] : i;
    }
  }

  vi GetSA() {
    vi v=P.back();
    vi ret(v.size());
    for(int i=0;i<v.size();i++){
      ret[v[i]]=i;
    }
    return ret;
  }
  int LCP(int i, int j) {
    int len = 0;
    if (i == j) return L - i;
    for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
      if (P[k][i] == P[k][j]) {
        i += 1 << k;
        j += 1 << k;
        len += 1 << k;
      }
    }
    return len;
  }
  vi GetLCP(vi &sa)
  {
    vi lcp(sa.size()-1);
    for(int i=0;i<sa.size()-1;i++){
      lcp[i]=LCP(sa[i],sa[i+1]);
    }
    return lcp;
  }
};
```

3.5 Suffix Array 2

```cpp
/***********************************************************************
  Suffix Array. Builing works in O(NlogN).
  Also LCP array is calculated in O(NlogN).
  This code counts number of different substrings in the string.
  Based on problem I from here: http://codeforces.ru/gym/100133
***********************************************************************/

const int MAXN = 205000;
const int ALPH = 256;
const int MAXLOG = 20;

int n;
char s[MAXN];
int p[MAXN]; // suffix array itself
int pcur[MAXN];
int c[MAXN][MAXLOG];
int num[MAXN];
int classesNum;
int lcp[MAXN];

void buildSuffixArray() {
  n++;

  for (int i = 0; i < n; i++)
    num[s[i]]++;

  for (int i = 1; i < ALPH; i++)
    num[i] += num[i - 1];

  for (int i = 0; i < n; i++) {
    p[num[s[i]] - 1] = i;
    num[s[i]]--;
  }

  c[p[0]][0] = 1;
  classesNum = 1;
  for (int i = 1; i < n; i++) {
    if (s[p[i]] != s[p[i - 1]])
      classesNum++;
    c[p[i]][0] = classesNum;
  }

  for (int i = 1; ; i++) {

    int half = (1 << (i - 1));

    for (int j = 0; j < n; j++) {
      pcur[j] = p[j] - half;
      if (pcur[j] < 0)
        pcur[j] += n;
    }

    for (int j = 1; j <= classesNum; j++)
      num[j] = 0;

    for (int j = 0; j < n; j++)
      num[c[pcur[j]][i - 1]]++;
    for (int j = 2; j <= classesNum; j++)
      num[j] += num[j - 1];

    for (int j = n - 1; j >= 0; j--) {
      p[num[c[pcur[j]][i - 1]] - 1] = pcur[j];
      num[c[pcur[j]][i - 1]]--;
    }

    c[p[0]][i] = 1;
    classesNum = 1;

    for (int j = 1; j < n; j++) {
      int p1 = (p[j] + half) % n, p2 = (p[j - 1] + half) % n;
      if (c[p[j]][i - 1] != c[p[j - 1]][i - 1] || c[p1][i - 1] != c[p2][i - 1])
        classesNum++;
      c[p[j]][i] = classesNum;
```

```
    }

    if ((1 << i) >= n)
      break;
  }

  for (int i = 0; i < n; i++)
    p[i] = p[i + 1];
  n--;
}

int getLcp(int a, int b) {
  int res = 0;
  for (int i = MAXLOG - 1; i >= 0; i--) {
    int curlen = (1 << i);
    if (curlen > n)
      continue;
    if (c[a][i] == c[b][i]) {
      res += curlen;
      a += curlen;
      b += curlen;
    }
  }
  return res;
}

void calcLcpArray() {
  for (int i = 0; i < n - 1; i++)
    lcp[i] = getLcp(p[i], p[i + 1]);
}

int main() {
  assert(freopen("substr.in","r",stdin));
  assert(freopen("substr.out","w",stdout));

  gets(s);
  n = strlen(s);

  buildSuffixArray();

  // Now p from 0 to n - 1 contains suffix array of original string

  /*for (int i = 0; i < n; i++) {
    printf("%d ", p[i] + 1);
  }*/

    calcLcpArray();

  long long ans = 0;
  for (int i = 0; i < n; i++)
    ans += n - p[i];
  for (int i = 1; i < n; i++)
    ans -= lcp[i - 1];

  cout << ans << endl;

  return 0;
}
```

3.6 Suffix Array Dilson

```
struct SuffixArray{
    const string& s;
    int n;
    vector<int> order, rank, lcp;
    vector<int> count, x, y;
    vector<int> sparse[22];
    SuffixArray(const string& s) : s(s), n(s.size()), order(n), rank(n),
    count(n + 1), x(n), y(n), lcp(n){
        for(int i=0;i<22;i++) sparse[i].resize(n, 0);
        build();
        buildLCP();
    }

    void build(){
        //sort suffiixes by the first character
        for(int i = 0; i < n; i++) order[i] = i;
        sort(order.begin(), order.end(), [&](int a, int b){return s[a] < s[b];});
        rank[order[0]] = 0;
        for(int i = 1; i < n; i++){
            rank[order[i]] = rank[order[i - 1]];
            if(s[order[i]] != s[order[i - 1]]) rank[order[i]]++;
        }


        //sort suffixex by the the first 2*p characters, for p in 1, 2, 4, 8,...
        for(int p = 1; p < n, rank[order[n - 1]] < n - 1; p <<= 1){
            for(int i = 0; i < n; i++){
                x[i] = rank[i];
                y[i] = i + p < n ? rank[i + p] + 1 : 0;
            }

            radixPass(y);
            radixPass(x);

            rank[order[0]] = 0;
            for(int i = 1; i < n; i++){
                rank[order[i]] = rank[order[i - 1]];
                if(x[order[i]] != x[order[i - 1]] or y[order[i]] != y[order[i - 1]]) rank[order[i
                    ]]++;;
            }
        }
    }

    //Stable counting sort
    void radixPass(vector<int>& key){
        fill(count.begin(), count.end(), 0);
        for(auto index : order) count[key[index]]++;
        for(int i = 1; i <= n; i++) count[i] += count[i - 1];
        for(int i = n - 1; i >= 0; i--) lcp[--count[key[order[i]]]] = order[i];
        order.swap(lcp);
    }

    //Kasai's algorithm to build the LCP array from order, rank and s
    //For i >= 1, lcp[i] refers to the suffixes starting at order[i] and order[i - 1]
    void buildLCP(){
        lcp[0] = 0;
        int k = 0;
        for(int i = 0; i < n; i++){
            if(rank[i] == n - 1){
                k = 0;
            }else{
                int j = order[rank[i] + 1];
                while(i + k < n and j + k < n and s[i + k] == s[j + k]) k++;
                lcp[rank[j]] = k;
                if(k) k--;
            }
        }
        for(int i=0;i<n;i++) sparse[0][i] = lcp[i];
        for(int j=1;j<22;j++)
            for(int i=n-1;i - (1 << (j-1) ) >=0; i--)
                sparse[j][i] = min(sparse[j-1][i], sparse[j-1][i - (1<< (j-1))]);
    }
    //Calcula o LCP do intervalo i e j.
    int LCP(int i, int j){
        if(i>j) return 0;
        if(i==j) return n-order[j];
        int k = log2(j-i);
        while(j - (1<<k) > i) k++;
        while(j - (1<<k) < i) k--;
        return min(sparse[k][j], sparse[k][i+ (1<<k) ]);
    }
};


int main(){
  ios::sync_with_stdio(false);
  string s;
  cin >> s;
  SuffixArray sa(s);
  for(int i = 0; i < s.size(); i++) cout << sa.order[i] << '\n';
}
```

3.7 Manacher Algorithm

```
/*************************************************************************
  Manacher's algorithm for finding all subpalindromes in the string.
  Based on problem L from here: http://codeforces.ru/gym/100133
*************************************************************************/

const int MAXN = 105000;

string s;
int n;
int odd[MAXN], even[MAXN];
int l, r;
long long ans;

int main() {
  assert(freopen("palindrome.in","r",stdin));
  assert(freopen("palindrome.out","w",stdout));

  getline(cin, s);
  n = (int) s.length();
```

```
  // Odd case
  l = r = -1;
  for (int i = 0; i < n; i++) {
    int cur = 1;
    if (i < r)
      cur = min(r - i + 1, odd[l + r - i]);
    while (i + cur < n && i - cur >= 0 && s[i - cur] == s[i + cur])
      cur++;
    odd[i] = cur;
    if (i + cur - 1 > r) {
      l = i - cur + 1;
      r = i + cur - 1;
    }
  }

  // Even case
  l = r = -1;
  for (int i = 0; i < n; i++) {
    int cur = 0;
    if (i < r)
      cur = min(r - i + 1, even[l + r - i + 1]);
    while (i + cur < n && i - 1 - cur >= 0 && s[i - 1 - cur] == s[i + cur])
      cur++;
    even[i] = cur;
    if (i + cur - 1 > r) {
      l = i - cur;
      r = i + cur - 1;
    }
  }

  for (int i = 0; i < n; i++) {
    if (odd[i] > 1) {
      ans += odd[i] - 1;
    }
    if (even[i])
      ans += even[i];
  }

  cout << ans << endl;

  return 0;
}
```

4 Numerical Algorithms

4.1 Fast Fourier Transform

```
// FFT - The Iterative Version
//
// Running Time:
//    O(n*log n)
//
// How To Use:
//   fft(a,1)
//   fft(b,1)
//   mul(a,b)
//   fft(a,-1)
//
// INPUT:
//  - fft method:
//      * The vector representing the polynomial
//      * 1 to normal transform
//      * -1 to inverse transform
//  - mul method:
//      * The two polynomials to be multiplyed
//
// OUTPUT:
//  - fft method: Transforms the vector sent.
//  - mul method: The result is kept in the first vector.
//
// NOTES:
//  - You can either use the struct defined of define dificil as complex<double>
//
// SOLVED:
//  * Codeforces Round #296 (Div. 1) D. Fuzzy Search
//

struct dificil {
  double real;
  double im;
  dificil() {
    real=0.0;
    im=0.0;
  }

  dificil(double real, double im):real(real),im(im){}

  dificil operator+(const dificil &o)const {
    return dificil(o.real+real, im+o.im);
  }

  dificil operator/(double v) const {
    return dificil(real/v, im/v);
  }

  dificil operator*(const dificil &o)const {
    return dificil(real*o.real-im*o.im, real*o.im+im*o.real);
  }

  dificil operator-(const dificil &o) const {
    return dificil(real-o.real, im-o.im);
  }
};

dificil tmp[MAXN*2];
int coco,maiorpot2[MAXN];

void fft(vector<dificil> &A, int s)
{
  int n = A.size(), p = 0;

  while(n>1){
    p++;
    n >>= 1;
  }

  n = (1<<p);

  vector<dificil> a=A;

  for(int i = 0; i < n; ++i){
    int rev = 0;
    for(int j = 0; j < p; ++j){
      rev <<= 1;
      rev |= ( (i >> j) & 1 );
    }
    A[i] = a[rev];
  }

  dificil w, wn;

  for(int i = 1; i <= p; ++i){
    int M = 1 << i;
    int K = M >> 1;
    wn = dificil(cos(s*2.0*pi/(double)M), sin(s*2.0*pi/(double)M) );
    for(int j = 0; j < n; j += M){
      w = dificil(1.0, 0.0);
      for(int l = j; l < K + j; ++l){
        dificil t = w;
        t = t*A[l + K];
        dificil u = A[l];
        A[l] =A[l]+ t;
        u = u-t;
        A[l + K] = u;
        w = wn*w;
      }
    }
  }

  if(s==-1){
    for(int i = 0;i<n;++i)
      A[i] = A[i]/(double)n;
  }
}

void mul(vector<dificil> &a, vector<dificil> &b)
{
  for(int i=0;i<a.size();i++)
  {
    a[i]=a[i]*b[i];
  }
}
```

4.2 Fast Fourier Transform 2

```
// FFT - The Recursive Version
//
// Running Time:
//    O(n*log n)
//
// How To Use:
//   fft(&a[0], tam, 0)
```

```
//   fft(&b[0], tam, 0)
//   mul(a,b)
//   fft(&a[0], tam, 1)
//
// INPUT:
//  - fft method:
//      * The vector representing the polynomial
//      * 0 to normal transform
//      * 1 to inverse transform
//  - mul method:
//      * The two polynomials to be multiplyed
//
// OUTPUT:
//  - fft method: Transforms the vector sent.
//  - mul method: The result is kept in the first vector.
//
// NOTES:
//  - Tam has to be a power of 2.
//  - You can either use the struct defined of define dificil as complex<double>
//
// SOLVED:
//  * Codeforces Round #296 (Div. 1) D. Fuzzy Search
//

dificil tmp[MAXN*2];
int coco, maiorpot2[MAXN];

void fft(dificil *v, int N, bool inv)
{
  if(N<=1) return;
  dificil *vodd = v;
  dificil *veven = v+N/2;
  for(int i=0; i<N; i++) tmp[i] = v[i];
  coco = 0;
  for(int i=0; i<N; i+=2)
  {
    veven[coco] = tmp[i];
    vodd[coco] = tmp[i+1];
    coco++;
  }
  fft(&vodd[0], N/2, inv);
  fft(&veven[0], N/2, inv);

  dificil w(1);
  double angucomleite = 2.0*PI/(double)N;
  if(inv) angucomleite = -angucomleite;

  dificil wn(cos(angucomleite),sin(angucomleite));
  for(int i=0;i<N/2;i++)
  {
    tmp[i] = veven[i]+w*vodd[i];
    tmp[i+N/2] = veven[i]-w*vodd[i];
    w *= wn;
    if(inv)
    {
      tmp[i] /= 2;
      tmp[i+N/2] /= 2;
    }
  }
  for(int i=0; i<N; i++) v[i] = tmp[i];
}

void mul(vector<dificil> &a, vector<dificil> &b)
{
  for(int i=0; i<a.size(); i++)
  {
    a[i] = a[i]*b[i];
  }
}

void precomp()
{
  int pot=0;
  for(int i=1;i<MAXN;i++)
  {
    if((1<<pot)<i) pot++;
    maiorpot2[i] = (1<<pot);
  }
}
```

4.3 Fast Fourier XOR Transform

```
/*
  Walsh-Hadamard Matrix :
  1 1
  1 -1
  Inverse :
  1 1
  1 -1
  v.size() power of 2
  usage:
  fft_xor(a, false);
  fft_xor(b, false);
  mul(a, b);
  fft_xor(a, true);
*/

void fft_xor(vi &a, bool inv) {
  vi ret = a;
  ll u, v;
  int tam = a.size() / 2;
  for(int len = 1; 2 * len <= tam; len <<= 1) {
    for(int i = 0; i < tam; i += 2 * len) {
      for(int j = 0; j < len; j++) {
        u = ret[i + j];
        v = ret[i + len + j];
        ret[i + j] = u + v;
        ret[i + len + j] = u - v;
      }
    }
  }
  if(inv) {
    for(int i = 0; i < tam; i++) {
      ret[i] /= tam;
    }
  }
  a = ret;
}
```

4.4 Fast Fourier OR Transform

```
/*
  Matrix :
  1 1
  1 0
  Inverse :
  0 1
  1 -1
  v.size() power of 2
  usage:
  fft_or(a, false);
  fft_or(b, false);
  mul(a, b);
  fft_or(a, true);
*/

void fft_or(vi &a, bool inv) {
  vi ret = a;
  ll u, v;
  int tam = a.size() / 2;
  for(int len = 1; 2 * len <= tam; len <<= 1) {
    for(int i = 0; i < tam; i += 2 * len) {
      for(int j = 0; j < len; j++) {
        u = ret[i + j];
        v = ret[i + len + j];
        if(!inv) {
          ret[i + j] = u + v;
          ret[i + len + j] = u;
        }
        else {
          ret[i + j] = v;
          ret[i + len + j] = u - v;
        }
      }
    }
  }
  a = ret;
}

void mul(vi &a, vi &b) {
  for(int i = 0; i < a.size(); i++) {
    a[i] = a[i] * b[i];
  }
}
```

4.5 Fast Fourier AND Transform

```
/*
  Matrix :
  0 1
  1 1
```

```
  Inverse :
  -1 1
  1 0
  v.size() power of 2
  usage:
  fft_and(a, false);
  fft_and(b, false);
  mul(a, b);
  fft_and(a, true);
*/

void fft_and(vi &a, bool inv) {
  vi ret = a;
  ll u, v;
  int tam = a.size() / 2;
  for(int len = 1; 2 * len <= tam; len <<= 1) {
    for(int i = 0; i < tam; i += 2 * len) {
      for(int j = 0; j < len; j++) {
        u = ret[i + j];
        v = ret[i + len + j];
        if(!inv) {
          ret[i + j] = v;
          ret[i + len + j] = u + v;
        }
        else {
          ret[i + j] = -u + v;
          ret[i + len + j] = u;
        }
      }
    }
  }
  a = ret;
}

void mul(vi &a, vi &b) {
  for(int i = 0; i < a.size(); i++) {
    a[i] = a[i] * b[i];
  }
}
```

4.6 Simpson Algorithm

```
const int NPASSOS = 100000;
const int W=1000000;
//W= tamanho do intervalo que eu estou integrando
double integrall()
{
  double h = W / (NPASSOS);
  double a = 0;
  double b = W;
  double s = f(a) + f(b);
  for (double i = 1; i <= NPASSOS; i += 2) s += f(a + i * h) * 4.0;
  for (double i = 2; i <= (NPASSOS - 1); i += 2) s += f(a + i * h) * 2.0;
  return s * h / 3.0;
}
```

4.7 Matrix Exponentiation

```
//matmul multiplica m1 por m2
//matpow exponencia a matrix m1 por p
//mul vet multiplica a matrix m1 pelo vetor vet
vvi matmul(vvi &m1, vvi &m2)
{
  vvi ans;
  ans.resize(m1.size(), vi(m2.size(), 0));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++) {
        ans[i][j] += m1[i][k] * m2[k][j];
        ans[i][j] %= MOD;
      }
  return ans;
}
vvi matpow(vvi &m1, ll p)
{
  vvi ans;
  ans.resize(m1.size(), vi(m1.size(), 0));
  for (int i = 0; i < n; i++) ans[i][i] = 1;
  while (p) {
    if (p & 1) ans = matmul(ans, m1);
    m1 = matmul(m1, m1);
    p >>= 1;
  }
  return ans;
}
// VETOR TEM N LINHAS E A MATRIZ E QUADRADA
vi mulvet(vvi &m1, vi &vet)
{
  vi ans;
  ans.resize(vet.size(), 0);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) {
      ans[i] += (m1[i][j] * vet[j]);
      ans[i] %= MOD;
    }
  return ans;
}
```

5 Mathematics

5.1 Chinese Remainder

```
ll mulmod(ll a, ll b, ll m)
{
  ll ret = 0;
  while (b > 0) {
    if (b % 2 != 0) ret = (ret + a) % m;
    a = (a + a) % m;
    b >>= 1;
  }
  return ret;
}
ll expmod(ll a, ll e, ll m)
{
  ll ret = 1;
  while (e > 0) {
    if (e % 2 != 0) ret = mulmod(ret, a, m);
    a = mulmod(a, a, m);
    e >>= 1;
  }
  return ret;
}

ll invmul(ll a, ll m) { return expmod(a, m - 2, m); }
ll chinese(vector<ll> r, vector<ll> m)
{
  int sz = m.size();
  ll M = 1;
  for (int i = 0; i < sz; i++) {
    M *= m[i];
  }
  ll ret = 0;
  for (int i = 0; i < sz; i++) {
    ret += mulmod(mulmod(M / m[i], r[i], M), invmul(M / m[i], M), M);
    ret = ret % M;
  }
  return ret;
}
```

5.2 Chinese Remainder 2

```
// Chinese remainder theorem (special case): find z such that // z % m1 = r1, z
// % m2 = r2. Here, z is unique modulo M = lcm(m1, m2). // Return (z, M). On
// failure, M = -1;
ii chinese_remainder_theorem(int m1, int r1, int m2, int r2)
{
  int s, t;
  int g = extended_euclid(m1, m2, s, t);
  if (r1 % g != r2 % g) return mp(0, -1);
  return mp(mod(s * r2 * m1 + t * r1 * m2, m1 * m2) / g, m1 * m2 / g);
}
// Chinese remainder theorem: find z such that // z % m[i] =
// r[i] for all i
//    .Note that the solution is   unique modulo M = lcm_i (m[i]).
//    Return(z, M)
//    .On  // failure, M = -1. Note that we do not require the a[i]   s
// to be relatively prime.
ii chinese_remainder_theorem(const vi &m, const vi &r)
{
  ii ret = make_pair(r[0], m[0]);
  for (int i = 1; i < m.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i]);
```

```
      if (ret.second == -1) break;
    }
    return ret;
  }
```

5.3 Matrix Exponentiation

```
//matmul multiplica m1 por m2
//matpow exponencia a matrix m1 por p
//mul vet multiplica a matrix m1 pelo vetor vet
vvi matmul(vvi &m1, vvi &m2)
{
  vvi ans;
  ans.resize(m1.size(), vi(m2.size(), 0));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++) {
        ans[i][j] += m1[i][k] * m2[k][j];
        ans[i][j] %= MOD;
      }
  return ans;
}
vvi matpow(vvi &m1, ll p)
{
  vvi ans;
  ans.resize(m1.size(), vi(m1.size(), 0));
  for (int i = 0; i < n; i++) ans[i][i] = 1;
  while (p) {
    if (p & 1) ans = matmul(ans, m1);
    m1 = matmul(m1, m1);
    p >>= 1;
  }
  return ans;
}
// VETOR TEM N LINHAS E A MATRIZ E QUADRADA
vi mulvet(vvi &m1, vi &vet)
{
  vi ans;
  ans.resize(vet.size(), 0);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) {
      ans[i] += (m1[i][j] * vet[j]);
      ans[i] %= MOD;
    }
  return ans;
}
```

5.4 Pascal Triangle

```
//Fazer combinacao de N escolhe M
//por meio do triangulo de pascal
//Complexidade: O(m*n)
unsigned long long comb[61][61];
for (int i = 0; i < 61; i++) {
   comb[i][i] = 1;
   comb[i][0] = 1;
}
for (int i = 2; i < 61; i++)
   for (int j = 1; j < i; j++)
      comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
```

5.5 Eulerś Totient Function

```
//retorna quantos elementos coprimos
//a N e menores que n existem
int phi(int n)
{
  int result = n;
  for (int i = 2; i * i <= n; ++i)
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      result -= result / i;
    }
  if (n > 1) result -= result / n;
  return result;
}
```

5.6 Pollard Rho

```
ll u;
ll t;
const int tamteste=5;
ll abss(ll v){ return v>=0 ? v : -v;}
ll randerson()
{
  ld pseudo=(ld)rand()/(ld)RAND_MAX;
  return (ll)(round((ld)range*pseudo))+1LL;
}

ll mulmod(ll a, ll b, ll mod)
{
  ll ret=0;
  while(b>0)
  {
    if(b%2!=0) ret=(ret+a)%mod;
    a=(a+a)%mod;
    b=b/2LL;
  }
  return ret;
}

ll expmod(ll a, ll e, ll mod)
{
  ll ret=1;
  while(e>0)
  {
    if(e%2!=0) ret=mulmod(ret,a,mod);
    a=mulmod(a,a,mod);
    e=e/2LL;
  }
  return ret;
}
bool jeova(ll a, ll n)
{
  ll x = expmod(a,u,n);
  ll last=x;
  for(int i=0;i<t;i++)
  {
    x=mulmod(x,x,n);
    if(x==1 and last!=1 and last!=(n-1)) return true;
    last=x;
  }
  if(x==1) return false;
  return true;
}

bool isprime(ll n)
{

  u=n-1;
  t=0;
  while(u%2==0)
  {
    t++;
    u/=2LL;
  }
  if(n==2) return true;
  if(n==3) return true;
  if(n%2==0) return false;
  if(n<2) return false;
  for(int i=0;i<tamteste;i++)
  {
    ll v = randerson()%(n-2)+1;
    //cout<<"jeova "<<v<<" "<<n<<endl;
    if(jeova(v,n)) return false;
  }
  return true;
}

ll gcd(ll a, ll b){ return !b ? a : gcd(b,a%b);}

ll calc(ll x, ll n, ll c)
{
  return (mulmod(x,x,n)+c)%n;
}
ll pollard(ll n)
{
  ll d=1;
  ll i=1;
  ll k=1;
  ll x=2;
  ll y=x;
  ll c;
  do
  {
    c=randerson()%n;
```

```
    }while(c==0 or (c+2)%n==0);
    while(d!=n)
    {
      if(i==k)
      {
          k*=2LL;
          y=x;
          i=0;
      }
      x=calc(x,n,c);
      i++;
      d=gcd(abss(y-x),n);
      if(d!=1) return d;
    }
  }

  vector<ll> getdiv(ll n)
  {
    vector<ll> ret;
    if(n==1) return ret;
    if(isprime(n))
    {
      ret.pb(n);
      return ret;
    }
    ll d = pollard(n);
    ret=getdiv(d);
    vector<ll> ret2=getdiv(n/d);
    for(int i=0;i<ret2.size();i++) ret.pb(ret2[i]);
    return ret;
  }
```

5.7 Extended Euclidean Algorithm

```
/* parametros finais:
a -> gcd(a, b)
x -> "inverso aritmetico" de a mod b
y -> "inverso aritmetico" de b mod a
resolve d = ax + by
para outras solucoes:
x + t * b / d
y - t * a / d */

int extended_euclid(int a, int b, int &x, int &y)
{
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a / b;
    int t = b;
    b = a % b;
    a = t;
    t = xx;
    xx = x - q * xx;
    x = t;
    t = yy;
    yy = y - q * yy;
    y = t;
  }
  return a;
}
```

5.8 Multiplicative Inverse

```
//computes b such that ab = 1(mod n), returns - 1 on failure
int mod_inverse(int a, int n)
{
  int x, y;
  int g = extended_euclid(a, n, x, y);
  if (g > 1) return -1;
  return (x+n)%n;
}
```

5.9 Multiplicative Inverse 2

```
//inverso multiplicativo de A % MOD
//certifique de MOD estar definido antes bonito!
//complexidade: O(log(a))

ll mul_inv(ll a)
{
  ll pin0 = MOD, pin = MOD, t, q;
  ll x0 = 0, x1 = 1;
  if (pin == 1) return 1;
  while (a > 1) {
    q = a / pin;
    t = pin, pin = a % pin, a = t;
    t = x0, x0 = x1 - q * x0, x1 = t;
  }
  if (x1 < 0) x1 += pin0;
  return x1;
}
```

5.10 Gaussian Elimination

```
const int N=105;

//resolvendo o sisteminha Ax = B
//no final, B tem a solucao x
//det eh o determinante de A
// complexidade: O(n^3)

ld A[N][N], B[N];
int n;

void solve() {
        ld mult;
        ld det = 1;

        for(int i=0; i<n; i++) {
                int nx = i;
                while(nx < n and fabs(A[nx][i]) < 1e-9) nx++;
                if(nx == n) {
                        det = 0;
                        //NO SOLUTION or INFINITY SOLUTIONS
                }
                if(nx != i) {
                        swap(A[nx], A[i]);
                        swap(B[nx], B[i]);
                        det = -det;
                }

                det *= A[i][i];

                // normalizando
                mult = 1.00 / A[i][i];
                for(int j=0; j<n; j++) {
                        A[i][j] *= mult;
                }
                B[i] *= mult;

                for(int j=0; j<n; j++) {
                        if(j == i) continue;
                        if(fabs(A[j][i]) > 1e-9) {
                                mult = A[j][i];
                                for(int k=0; k<n; k++) {
                                        A[j][k] -= mult * A[i][k];
                                }
                                B[j] -= mult * B[i];
                        }
                }
        }
}
```

5.11 Gaussian Elimination with MOD

```
const int N=105;
const int MAXN = 1e6+10;

//resolvendo o sisteminha Ax = B
//fazendo operacoes de mod p
//no final, B tem a solucao x
//det eh o determinante de A
// complexidade: O(n^3)

ll A[N][N], B[N];
ll inv[MAXN];
int n, p;

ll extended_euclid(int i, int p) {

}
```

```
ll soma(ll a, ll b) {
	return ((a + b) % p + p) % p;
}
ll sub(ll a, ll b) {
	return ((a - b) % p + p) % p;
}
ll mul(ll a, ll b) {
	return ((a * b) % p + p) % p;
}

void solve() {
	for(int i=1; i<p; i++) {
		inv[i] = extended_euclid(i, p);
	}

	ll mult;
	ll det = 1;

	for(int i=0; i<n; i++) {
		int nx = i;
		while(nx < n and A[nx][i] == 0) nx++;
		if(nx == n) {
			det = 0;
			//NO SOLUTION or INFINITY SOLUTIONS
		}
		if(nx != i) {
			swap(A[nx], A[i]);
			swap(B[nx], B[i]);
			det = -det;
		}

		det = mul(det, A[i][i]);

		// normalizando
		mult = inv[A[i][i]];
		for(int j=0; j<n; j++) {
			A[i][j] = mul(A[i][j], mult);
		}
		B[i] = mul(B[i], mult);

		for(int j=0; j<n; j++) {
			if(j == i) continue;
			if(A[j][i] != 0) {
				mult = A[j][i];
				for(int k=0; k<n; k++) {
					A[j][k] = sub(A[j][k], mul(mult, A[i][k]));
				}
				B[j] = sub(B[j], mul(mult, B[i]));
			}
		}
	}
}
```

5.12 Gaussian Elimination with XOR

```
#include <bits/stdc++.h>
using namespace std;
#define sc(a) scanf("%d", &a)
#define sc2(a, b) scanf("%d%d", &a, &b)
#define sc3(a, b, c) scanf("%d%d%d", &a, &b, &c)
#define scs(a) scanf("%s", a)
#define pri(x) printf("%d\n", x)
#define prie(x) printf("%d ", x)
#define mp make_pair
#define pb push_back
#define BUFF ios::sync_with_stdio(false);
#define db(x) cerr << #x << " == " << x << endl
#define f first
#define s second
typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;
const ld pi = acos(-1);
const int MOD = 1e9 + 7;

const int N=105;

//esse eh BABA!
//ateh o mp aguenta
//sisteminha Ax = B de xor, B guarda solucao

int A[N][N], B[N];
int n;

void solve() {
	int det = 1;

	for(int i=0; i<n; i++) {
		int nx = i;
		while(nx < n and A[nx][i] == 0) nx++;
		if(nx == n) {
			//NO SOLUTION or MULTIPLE SOLUTIONS
		}
		if(nx != i) {
			swap(A[nx], A[i]);
			swap(B[nx], B[i]);
		}

		for(int j=0; j<n; j++) {
			if(j == i) continue;
			if(A[j][i] != 0) {
				for(int k=0; k<n; k++) {
					A[j][k] ^= A[i][k];
				}
				B[j] ^= B[i];
			}
		}
	}
}

int main() {
	return 0;
}
```

5.13 Determinant

```
const int N=105;

//calculo do determinante
//COM COEFICIENTES INTEIROS --> PICA!
//segue a ideia do calculo do GCD
//complexidade: O(n^3 lg MX)
//0 erro de precisao
//0-based porque sim!

ll mat[N][N];
int n;

void limpa(int a) {
	for(int i=0; i<n; i++) {
		mat[a][i] = -mat[a][i];
	}
}

void troca(int a, int b) {
	for(int i=0; i<n; i++) {
		swap(mat[a][i], mat[b][i]);
	}
}

ll det() {
	ll ans = 1;
	for(int i=0; i<n; i++) {
		for(int j=i+1; j<n; j++) {
			int a = i, b = j;

			if(mat[a][i] < 0)	limpa(a), ans = -ans;
			if(mat[b][i] < 0)	limpa(b), ans = -ans;

			while(mat[b][i] != 0) {
				ll q = mat[a][i] / mat[b][i];
				for(int k=0; k<n; k++) {
					mat[a][k] -= q * mat[b][k];
				}
				swap(a, b);
			}

			if(a != i) {
				troca(i, j);
				ans = -ans;
			}
		}
		ans *= mat[i][i];
	}

	return ans;
}
```

6 Combinatorial Optimization

6.1 Dinic

```
//grafo bipartido O(Esqrt(v))
//Para recuperar a resposta, e so colocar um bool
//de false na aresta de retorno e fazer uma bfs/dfs
//andando pelos vertices de capacidade =0 e arestas
//que nao sao de retorno
struct Edge {
  int v, rev;
  int cap;
  Edge(int v_, int cap_, int rev_) : v(v_), rev(rev_), cap(cap_) {}
};

struct MaxFlow {
  vector<vector<Edge> > g;
  vector<int> level;
  queue<int> q;
  int flow, n;

  MaxFlow(int n_) : g(n_), level(n_), n(n_) {}
  void addEdge(int u, int v, int cap)
  {
    if (u == v) return;
    Edge e(v, cap, int(g[v].size()));
    Edge r(u, 0, int(g[u].size()));
    g[u].push_back(e);
    g[v].push_back(r);
  }

  bool buildLevelGraph(int src, int sink)
  {
    fill(level.begin(), level.end(), -1);
    while (not q.empty()) q.pop();
    level[src] = 0;
    q.push(src);
    while (not q.empty()) {
      int u = q.front();
      q.pop();
      for (auto e = g[u].begin(); e != g[u].end(); ++e) {
        if (not e->cap or level[e->v] != -1) continue;
        level[e->v] = level[u] + 1;
        if (e->v == sink) return true;
        q.push(e->v);
      }
    }
    return false;
  }

  int blockingFlow(int u, int sink, int f)
  {
    if (u == sink or not f) return f;
    int fu = f;
    for (auto e = g[u].begin(); e != g[u].end(); ++e) {
      if (not e->cap or level[e->v] != level[u] + 1) continue;
      int mincap = blockingFlow(e->v, sink, min(fu, e->cap));
      if (mincap) {
        g[e->v][e->rev].cap += mincap;
        e->cap -= mincap;
        fu -= mincap;
      }
    }
    if (f == fu) level[u] = -1;
    return f - fu;
  }

  int maxFlow(int src, int sink)
  {
    flow = 0;
    while (buildLevelGraph(src, sink))
      flow += blockingFlow(src, sink, numeric_limits<int>::max());
    return flow;
  }
};
```

6.2 Hopcroft-Karp Bipartite Matching

```
/* O(v^3)
* Matching maximo de grafo bipartido de peso 1 nas arestas
* supondo que o grafo bipartido seja enumerado de 0-n-1
* chamamos maxMatch(n)
*/
class MaxMatch {
  vi graph[N];
  int match[N], us[N];

 public:
  MaxMatch(){};
  void addEdge(int u, int v) { graph[u].pb(v); }
  int dfs(int u)
  {
    if (us[u]) return 0;
    us[u] = 1;
    for (int v : graph[u]) {
      if (match[v] == -1 or (dfs(match[v]))) {
        match[v] = u;
        return 1;
      }
    }
    return 0;
  }
  int maxMatch(int n)
  {
    memset(match, -1, sizeof(match));
    int ret = 0;
    for (int i = 0; i < n; i++) {
      memset(us, 0, sizeof(us));
      ret += dfs(i);
    }
    return ret;
  }
};
```

6.3 Max Bipartite Matching 2

```
// This code performs maximum bipartite matching.
//
// Running time: O(|E| |V|) -- often much faster in practice
//
//   INPUT: w[i][j] = edge between row node i and column node j
//   OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//           mc[j] = assignment for column node j, -1 if unassigned
//           function returns number of matches made
#include <vector>
using namespace std;
typedef vector<int> VI;
typedef vector<VI> VVI;
bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}
int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
  mr = VI(w.size(), -1);
  mc = VI(w[0].size(), -1);
  int ct = 0;
  for (int i = 0; i < w.size(); i++) {
    VI seen(w[0].size());
    if (FindMatch(i, w, mr, mc, seen)) ct++;
  }
  return ct;
}
```

6.4 Maximum Matching in General Graphs (Blossom)

```
/*
GETS:
V->number of vertices
E->number of edges
pair of vertices as edges (vertices are 1..V)

GIVES:
output of edmonds() is the maximum matching
match[i] is matched pair of i (-1 if there isn't a matched pair)
```

```
Code for the SEAGRP problem at CodeChef.
SEAGRP's limits are: 1 <= V, E <= 100.
The problem asked if there is a perfect matching.
*/

#include <bits/stdc++.h>
using namespace std;
const int M=500;
struct struct_edge { int v; struct_edge* n; };
typedef struct_edge* edge;
struct_edge pool[M*M*2];
int topindex;
edge adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];

void clean()
{
  memset(ed, false, sizeof(ed));
  topindex=0;

  for(int i = 0; i < M; i++)
    adj[i] = NULL;
}

void add_edge(int u,int v)
{
  edge top = &pool[topindex++];
  top->v=v,top->n=adj[u],adj[u]=top;
  top = &pool[topindex++];
  top->v=u,top->n=adj[v],adj[v]=top;
}
int LCA(int root,int u,int v)
{
  static bool inp[M];
  memset(inp,0,sizeof(inp));
  while(1)
  {
    inp[u=base[u]]=true;
    if (u==root) break;
    u=father[match[u]];
  }
  while(1)
  {
    if (inp[v=base[v]]) return v;
    else v=father[match[v]];
  }
}
void mark_blossom(int lca,int u)
{
  while (base[u]!=lca)
  {
    int v=match[u];
    inb[base[u]]=inb[base[v]]=true;
    u=father[v];
    if (base[u]!=lca) father[u]=v;
  }
}
void blossom_contraction(int s,int u,int v)
{
  int lca=LCA(s,u,v);
  memset(inb,0,sizeof(inb));
  mark_blossom(lca,u);
  mark_blossom(lca,v);
  if (base[u]!=lca)
    father[u]=v;
  if (base[v]!=lca)
    father[v]=u;
  for (int u=0;u<V;u++)
    if (inb[base[u]])
    {
      base[u]=lca;
      if (!inq[u])
        inq[q[++qt]=u]=true;
    }
}
int find_augmenting_path(int s)
{
  memset(inq,0,sizeof(inq));
  memset(father,-1,sizeof(father));
  for (int i=0;i<V;i++) base[i]=i;
  inq[q[qh=qt=0]=s]=true;
  while (qh<=qt)
  {
    int u=q[qh++];
    for (edge e=adj[u];e!=NULL;e=e->n)
    {
      int v=e->v;
      if (base[u]!=base[v]&&match[u]!=v)
      {
        if ((v==s)||(match[v]!=-1 && father[match[v]]!=-1))
          blossom_contraction(s,u,v);
        else if (father[v]==-1)
        {
          father[v]=u;
          if (match[v]==-1)
            return v;
          else if (!inq[match[v]])
            inq[q[++qt]=match[v]]=true;
        }
      }
    }
  }
  return -1;
}
int augment_path(int s,int t)
{
  int u=t,v,w;
  while (u!=-1)
  {
    v=father[u];
    w=match[v];
    match[v]=u;
    match[u]=v;
    u=w;
  }
  return t!=-1;
}
int edmonds()
{
  int matchc=0;
  memset(match,-1,sizeof(match));
  for (int u=0;u<V;u++)
    if (match[u]==-1)
      matchc+=augment_path(u,find_augmenting_path(u));
  return matchc;
}
int main()
{
  int u, v, t;

  cin >> t;
  while(t--)
  {
    cin >> V >> E;
    clean();
    while(E--)
    {
      cin >> u >> v;
      if (!ed[u-1][v-1])
      {
        add_edge(u-1,v-1);
        ed[u-1][v-1]=ed[v-1][u-1]=true;
      }
    }

    //cout << "UE\n";
    //cout << V << " " << edmonds() << endl;
    //for (int i=0;i<V;i++)
    //  if (i<match[i])
    //    cout<<i+1<<" "<<match[i]+1<<endl;
    //cout << endl;
    if(2*edmonds() == V) cout << "YES\n";
    else cout << "NO\n";
  }
  return 0;
}
```

6.5 Min Cost Matching

```
///////////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs.  In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative.  To perform
// maximization, simply negate the cost[][] matrix.
///////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cmath>
#include <cstdio>
```

```
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate)
{
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
        break;
      }
    }
  }

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
  while (mated < n) {
    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++) dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {
      // find closest
      j = -1;
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        if (j == -1 || dist[k] < dist[j]) j = k;
      }
      seen[j] = 1;

      // termination condition
      if (Rmate[j] == -1) break;

      // relax neighbors
      const int i = Rmate[j];
      for (int k = 0; k < n; k++) {
        if (seen[k]) continue;
        const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
        if (dist[k] > new_dist) {
          dist[k] = new_dist;
          dad[k] = j;
        }
      }
    }

    // update dual variables
    for (int k = 0; k < n; k++) {
      if (k == j || !seen[k]) continue;
      const int i = Rmate[k];
      v[k] += dist[k] - dist[j];
      u[i] -= dist[k] - dist[j];
    }
    u[s] += dist[j];

    // augment along path
    while (dad[j] >= 0) {
      const int d = dad[j];
      Rmate[j] = Rmate[d];
      Lmate[Rmate[j]] = j;
      j = d;
    }
    Rmate[j] = s;
    Lmate[s] = j;

    mated++;
  }

  double value = 0;
  for (int i = 0; i < n; i++) value += cost[i][Lmate[i]];

  return value;
}
```

6.6 Min Cost Max Flow

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972).  This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]).  For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow:  O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <iostream>
#include <vector>

using namespace std;

typedef vector<VI> VVI;
typedef long long LL;
typedef vector<LL> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const LL INF = numeric_limits<LL>::max() / 4;

struct MinCostMaxFlow {
  int N;
  vector< vector<ll> >  cap, flow, cost;
  vector<int> found;
  vector<ll> dist, pi, width;
  vector< pair<int, int> > dad;

  MinCostMaxFlow(int N): N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
      found(N), dist(N), pi(N), width(N), dad(N){}

  void AddEdge(int from, int to, LL cap, LL cost)
  {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, ll cap, ll cost, int dir)
  {
    ll val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  LL Dijkstra(int s, int t)
  {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
```

```
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++) pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<LL, LL> GetMaxFlow(int s, int t)
  {
    LL totflow = 0, totcost = 0;
    while (LL amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        }
        else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

6.7 Min Cost Max Flow Dilson

```
#define INF 0x3f3f3f3f

struct Edge{
	int v, rev, cap, cost, orig_cost;
	bool orig;
	Edge(int v_, int cap_, int cost_, int rev_, bool orig_) : v(v_),
	rev(rev_), cap(cap_), cost(cost_), orig_cost(cost_), orig(orig_) {}
};

struct MinCostMaxFlow{
	vector<vector<Edge> > g;
	vector<int> p, pe, dist;
	int flow, cost, n;

	MinCostMaxFlow(int n_) : g(n_), p(n_), pe(n_), dist(n_), n(n_) {}

	void addEdge(int u, int v, int cap, int cost){
		if(u == v) return;
		Edge e(v, cap, cost, int(g[v].size()), true);
		Edge r(u, 0, 0, int(g[u].size()), false);
		g[u].push_back(e);
		g[v].push_back(r);
	}

	bool findPath(int src, int sink){
		set<pair<int, int> > q;
		fill(ALL(dist), INF);
		dist[src] = 0;
		p[src] = src;
		q.insert(make_pair(dist[src], src));
		while(not q.empty()){
			int u = q.begin()->second;
			q.erase(q.begin());

			FOREACH(e, g[u]){
				if(not e->cap) continue;
				int newdist = dist[u] + e->cost;
				if(newdist < dist[e->v]){
					if(dist[e->v] == INF) q.erase(make_pair(dist[e->v], e->v));
					dist[e->v] = newdist;
					q.insert(make_pair(newdist, e->v));
					p[e->v] = u;
					pe[e->v] = int(distance(g[u].begin(), e));
				}
			}
		}
		return dist[sink] < INF;
	}

	void fixCosts(){
		FORN(u, 0, n)
			FOREACH(e, g[u]){
				if(e->cap){
					if(e->cap) e->cost = min(INF, e->cost + dist[u] - dist[e->v]);
				}else{
					e->cost = 0;
				}
			}
	}

	void augmentFlow(int sink){
		int mincap = numeric_limits<int>::max();
		for(int v = sink; p[v] != v; v = p[v])
			mincap = min(mincap, g[p[v]][pe[v]].cap);
		for(int v = sink; p[v] != v; v = p[v]){
			Edge& e = g[p[v]][pe[v]];
			Edge& r = g[v][g[p[v]][pe[v]].rev];
			e.cap -= mincap;
			r.cap += mincap;
			cost += (e.orig ? e.orig_cost : -r.orig_cost) * mincap;

		}
		flow += mincap;
	}

	void fixInitialCosts(int src)
	{
		fill(ALL(dist), INF);
		dist[src] = 0;
		FORN(i, 0, n){
			FORN(u, 0, n){
				FOREACH(e, g[u]){
					if(e->orig) dist[e->v] = min(dist[e->v], dist[u] + e->cost);
				}
			}
		}
		fixCosts();
	}

	pair<int, int> maxFlow(int src, int sink){
		flow = 0;
		cost = 0;
		fixInitialCosts(src);
		while(findPath(src, sink)){
			fixCosts();
			augmentFlow(sink);
		}
		return make_pair(flow, cost);
	}

};
```

6.8 Find Maximum Clique in Graphs

```
int n,k;
ll g[41];
ll dp[(1<<20)];
ll dp2[(1<<20)];
int t1,t2;
//graph is a bitmask
//meet in the middle technique
// complexity : O(sqrt(2)^n)
ll Adam_Sendler()
{
	t1=n/2;
	t2=n-t1;
	ll r=0;
	for(ll mask=1;mask<(1ll<<t1);mask++){
		for(ll j=0;j<t1;j++)
			if(mask&(1ll<<j)) {
				ll outra= mask-(1ll<<j);
				ll r1= __builtin_popcountll(dp[mask]);
				ll r2= __builtin_popcountll(dp[outra]);
				if(r2>r1) dp[mask] = dp[outra];
			}
		bool click=true;
		for(ll j=0;j<t1;j++)
			if( (1ll<<j)&mask)
				if( ((g[j]^mask)&mask)) click=false;
		if(click) dp[mask]=mask;
		ll r1= __builtin_popcountll(dp[mask]);
		r=max(r,r1);
	}

	for(ll mask=1;mask<(1ll<<t2);mask++){
		for(ll j=0;j<t2;j++)
			if(mask&(1ll<<j)) {
				ll outra= mask-(1ll<<j);
				ll r1= __builtin_popcountll(dp2[mask]);
				ll r2= __builtin_popcountll(dp2[outra]);
```

```
                        if(r2>r1) dp2[mask] = dp2[outra];
                }
        bool click=true;
        for(ll j=0;j<t2;j++){
                if( (1ll<<j)&mask){
                        ll m1= g[j+t1];
                        ll cara= mask<<t1;
                        if((m1^cara)&cara){
                                click=false;
                        }
                }
        }
        if(click) {
                dp2[mask]=mask;
        }
        ll r1= __builtin_popcountll(dp2[mask]);
        if(r1==0) db(mask);
        r=max(r,r1);
    }

    for(ll mask=0;mask<(1ll<<t1);mask++){
        ll tudo= (1ll<<n) -1;
        for(ll j=0;j<t1;j++)
                if( (1ll<<j)&mask) tudo&=g[j];

        tudo>>=t1;
        ll x=__builtin_popcountll(dp[mask]);
        ll y=__builtin_popcountll(dp2[tudo]);
        r=max(r, x+y);
    }
    return r;
}

int main()
{
    sc2(n,k);
    for(int i=0;i<n;i++){
        g[i]|=(1ll<<i);
        for(int j=0;j<n;j++){
                int x;
                sc(x);
                if(x){
                        g[i]|=(1ll<<j);
                }
        }
    }
    int m=Adam_Sendler();
    //db(m);
    cout<<fixed<<setprecision(10);
    cout<<(k*k*(m-1))/(2.0*m)<<endl;
    return 0;
}
```

7 Dynamic Programming

7.1 Convex Hull Trick

```
/* Esse convex hull trick e para achar a reta minima!
 * Para maximizar a reta dada , basta trocar o '>' para
 * para '<' na funcao query;
 * Nao chamar query com B ou A vazios! Atualizar dp para
 * depois fazer a query =)
 * ATENCAO COM O DOUBLE!! ESTA EM LONG LONG :)
*/
vi A[N], B[N];
int pont[N];
bool odomeioehlixo(int r1, int r2, int r3, int j)
{
  return (B[j][r1] - B[j][r3]) * (A[j][r2] - A[j][r1]) <
         (B[j][r1] - B[j][r2]) * (A[j][r3] - A[j][r1]);
}

void add(ll a, ll b, int j)
{
  B[j].pb(b);
  A[j].pb(a);
  while (B[j].size() >= 3 and
        odomeioehlixo(B[j].size() - 3, B[j].size() - 2, B[j].size() - 1, j)) {
    B[j].erase(B[j].end() - 2);
    A[j].erase(A[j].end() - 2);
  }
}

ll query(ll x, int j)
{
  if (pont[j] >= B[j].size()) pont[j] = B[j].size() - 1;
  while (pont[j] < B[j].size() - 1 and
        (A[j][pont[j] + 1] * x + B[j][pont[j] + 1] >
         A[j][pont[j]] * x + B[j][pont[j]]))
    pont[j]++;
  return A[j][pont[j]] * x + B[j][pont[j]];
}
/* Testado em :
 * http://www.spoj.com/problems/APIO10A/
 * http://www.spoj.com/problems/ACQUIRE/
*/
```

7.2 Dinamic Convex Hull Trick

```
/*
 * Given a set of pairs (m, b) specifying lines of the form y = m*x + b, process
 * a
 * set of x-coordinate queries each asking to find the minimum y-value when any
 * of
 * the given lines are evaluated at the specified x. To instead have the queries
 * optimize for maximum y-value, set the QUERY_MAX flag to true.
 * The following implementation is a fully dynamic variant of the convex hull
 * optimization technique, using a self-balancing binary search tree (std::set)
 * to
 * support the ability to call add_line() and get_best() in any desired order.
 * Explanation: http://wcipeg.com/wiki/Convex_hull_trick#Fully_dynamic_variant
 * Time Complexity: O(n log n) on the total number of calls made to add_line(),
 * for
 * any length n sequence of arbitrarily interlaced add_line() and get_min()
 * calls.
 * Each individual call to add_line() is O(log n) amortized and each individual
 * call to get_best() is O(log n), where n is the number of lines added so far.
 * Space Complexity: O(n) auxiliary on the number of calls made to add_line().
 * */

#include <limits>  // std::numeric_limits
#include <set>

class hull_optimizer {
  struct line {
    long long m, b, val;
    double xlo;
    bool is_query;
    bool query_max;

    line(long long m, long long b, long long val, bool is_query, bool query_max)
    {
      this->m = m;
      this->b = b;
      this->val = val;
      this->xlo = -std::numeric_limits<double>::max();
      this->is_query = is_query;
      this->query_max = query_max;
    }

    bool parallel(const line &l) const { return m == l.m; }
    double intersect(const line &l) const
    {
      if (parallel(l)) return std::numeric_limits<double>::max();
      return (double)(l.b - b) / (m - l.m);
    }

    bool operator<(const line &l) const
    {
      if (l.is_query) return query_max ? (xlo < l.val) : (l.val < xlo);
      return m < l.m;
    }
  };

  std::set<line> hull;
  bool _query_max;

  typedef std::set<line>::iterator hulliter;

  bool has_prev(hulliter it) const { return it != hull.begin(); }
  bool has_next(hulliter it) const
  {
    return (it != hull.end()) && (++it != hull.end());
  }

  bool irrelevant(hulliter it) const
  {
    if (!has_prev(it) || !has_next(it)) return false;
    hulliter prev = it, next = it;
    --prev;
    ++next;
    return _query_max ? prev->intersect(*next) <= prev->intersect(*it)
                      : next->intersect(*prev) <= next->intersect(*it);
```

```
  }

  hulliter update_left_border(hulliter it)
  {
    if ((_query_max && !has_prev(it)) || (!_query_max && !has_next(it)))
      return it;
    hulliter it2 = it;
    double val = it->intersect(_query_max ? *--it2 : *++it2);
    line l(*it);
    l.xlo = val;
    hull.erase(it++);
    return hull.insert(it, l);
  }

 public:
  hull_optimizer(bool query_max = false) { this->_query_max = query_max; }
  void add_line(long long m, long long b)
  {
    line l(m, b, 0, false, _query_max);
    hulliter it = hull.lower_bound(l);
    if (it != hull.end() && it->parallel(l)) {
      if ((_query_max && it->b < b) || (!_query_max && b < it->b))
        hull.erase(it++);
      else
        return;
    }
    it = hull.insert(it, l);
    if (irrelevant(it)) {
      hull.erase(it);
      return;
    }
    while (has_prev(it) && irrelevant(--it)) hull.erase(it++);
    while (has_next(it) && irrelevant(++it)) hull.erase(it--);
    it = update_left_border(it);
    if (has_prev(it)) update_left_border(--it);
    if (has_next(++it)) update_left_border(++it);
  }

  long long get_best(long long x) const
  {
    line q(0, 0, x, true, _query_max);
    hulliter it = hull.lower_bound(q);
    if (_query_max) --it;
    return it->m * x + it->b;
  }
};

/*** Example Usage ***/

#include <cassert>

int main()
{
  hull_optimizer h;
  h.add_line(3, 0);
  h.add_line(0, 6);
  h.add_line(1, 2);
  h.add_line(2, 1);
  assert(h.get_best(0) == 0);
  assert(h.get_best(2) == 4);
  assert(h.get_best(1) == 3);
  assert(h.get_best(3) == 5);
  return 0;
}
```

7.3 Divide and Conquer Example

```
//Um exemplo de Divide and conquer:
int MOD = 1e9 + 7;
const int N = 1010;
int dp[N][N], cost[N][N], v[N], pref[N], n, m;
void compDP(int j, int L, int R, int b, int e)
{
  if (L > R) return;
  int mid = (L + R) / 2;
  int idx = -1;
  for (int i = b; i <= min(mid, e); i++)
    if (dp[mid][j] > dp[i][j - 1] + cost[i + 1][mid]) {
      idx = i;
      dp[mid][j] = dp[i][j - 1] + cost[i + 1][mid];
    }
  compDP(j, L, mid - 1, b, idx);
  compDP(j, mid + 1, R, idx, e);
}
//chamada!
for(int i=1;i<=n;i++) dp[i][0]=cost[1][i];
for(int i=1;i<=m;i++) compDP(i,1,n,1,n);
```

8 Geometry

8.1 Convex Hull Monotone Chain

```
typedef struct sPoint {
        int x, y;
        sPoint(int _x, int _y)
        {
                x = _x;
                y = _y;
        }
} point;
bool comp(point a, point b)
{
        if (a.x == b.x) return a.y < b.y;
        return a.x < b.x;
}
int cross(point a, point b, point c)  // AB x BC
{
        a.x -= b.x;
        a.y -= b.y;
        b.x -= c.x;
        b.y -= c.y;
        return a.x * b.y - a.y * b.x;
}

bool isCw(point a, point b, point c)  // Clockwise
{
        return cross(a, b, c) < 0;
}

// >= if you want to put collinear points on the convex hull
bool isCcw(point a, point b, point c)  // Counter Clockwise
{
        return cross(a, b, c) > 0;
}

vector<point> convexHull(vector<point> p)
{
        vector<point> u, l;  // Upper and Lower hulls

        sort(p.begin(), p.end(), comp);
        for (unsigned int i = 0; i < p.size(); i++) {
                while (l.size() > 1 && !isCcw(l[l.size() - 1], l[l.size() - 2], p[i]))
                        l.pop_back();
                l.push_back(p[i]);
        }

        for (int i = p.size() - 1; i >= 0; i--) {
                while (u.size() > 1 && !isCcw(u[u.size() - 1], u[u.size() - 2], p[i]))
                        u.pop_back();
                u.push_back(p[i]);
        }
        u.pop_back();
        l.pop_back();
        l.insert(l.end(), u.begin(), u.end());
        return l;
}
```

8.2 Fast Geometry in Cpp

```
// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
```

```cpp
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
```

```
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
       << PointInPolygon(v, PT(2,0)) << " "
       << PointInPolygon(v, PT(0,2)) << " "
       << PointInPolygon(v, PT(5,2)) << " "
       << PointInPolygon(v, PT(2,5)) << endl;

  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
       << PointOnPolygon(v, PT(2,0)) << " "
       << PointOnPolygon(v, PT(0,2)) << " "
       << PointOnPolygon(v, PT(5,2)) << " "
       << PointOnPolygon(v, PT(2,5)) << endl;

  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

8.3 Point Inside Polygon O(lg N)

```
/*
 * Solution for UVa 11072 - Points
 *
 * On this problem you must calculate the convex hull on the
 * first set of points.
 *
 * And for each point of the second set, answer if the point
 * is inside or outside the convex hull.
 */
typedef struct sPoint {
  ll x, y;

  sPoint() {}
  sPoint (ll _x, ll _y) : x(_x), y(_y) {}
  bool operator<(const sPoint& other) const
  {
    if(x == other.x) return y < other.y;
    return x < other.x;
  }
} point;

vector<point> vp, ch;

ll cross(point a, point b, point c) // AB x BC
{
  a.x -= b.x;  a.y -= b.y;
  b.x -= c.x;  b.y -= c.y;
  return a.x*b.y - a.y*b.x;
}

vector<point> convexhull()
{
  sort(vp.begin(), vp.end());

  vector<point> l, u;
  for(int i = 0; i < vp.size(); i++)
  {
    while(l.size() > 1 && cross(l[l.size()-2], l[l.size()-1], vp[i]) <= 0)
      l.pop_back();
    l.pb(vp[i]);
  }

  for(int i = vp.size()-1; i >= 0; i--)
  {
    while(u.size() > 1 && cross(u[u.size()-2], u[u.size()-1], vp[i]) <= 0)
      u.pop_back();
    u.pb(vp[i]);
  }
  l.pop_back();  u.pop_back();
  l.insert(l.end(), u.begin(), u.end());
  return l;
}

ll area(point a, point b, point c)
{ return llabs(cross(a, b, c)); }

bool insideTriangle(point a, point b, point c, point p)
{
  return area(a, b, c) == (area(a, b, p) +
      area(a, c, p) +
      area(b, c, p));
}

bool isInside(point p)
{
  if(ch.size() < 3) return false;

  int i = 2, j = ch.size()-1;

  while(i < j)
  {
    int mid = (i+j)/2;
    ll c = cross(ch[0], ch[mid], p);
    if(c > 0) i = mid+1;
    else j = mid;
  }
  return insideTriangle(ch[0], ch[i], ch[i-1], p);
}

int main()
```

```
{
  int n;

  while(true)
  {
    ch.clear();
    vp.clear();
    cin >> n;
    if(not cin) break;

    while(n--)
    {
      point p;
      cin >> p.x >> p.y;
      vp.pb(p);
    }

    ch = convexhull();

    cin >> n;
    while(n--)
    {
      point p;
      cin >> p.x >> p.y;
      if(isInside(p)) cout << "inside\n";
      else cout << "outside\n";
    }
  }

  return 0;
}
```

8.4 Minimum Enclosing Circle O(N)

```
const int MOD=1e9+7;
const ll LINF=0x3f3f3f3f3f3f3f3f;
double INF = 1e100;
double EPS = 1e-12;

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

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }

PT RotateCW90(PT p)    { return PT(p.y,-p.x); }

PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

struct circle {
  PT cen;
  double r;
  circle() {}
  circle(PT cen, double r) : cen(cen), r(r) {}
};

bool inside(circle &c, PT &p) {
  return (c.r * c.r + 1e-6 > dist2(p, c.cen));
}

PT bestOf3(PT a, PT b, PT c) {
  if(dot(b - a, c - a) < 1e-9) return (b + c) / 2.0;
  if(dot(a - b, c - b) < 1e-9) return (a + c) / 2.0;
  if(dot(a - c, b - c) < 1e-9) return (a + b) / 2.0;
  return ComputeCircleCenter(a, b, c);
}

circle minCirc(vector<PT> v) {
  int n = v.size();
  random_shuffle(v.begin(), v.end());
  PT p = PT(0.0, 0.0);
  circle ret = circle(p, 0.0);
  for(int i = 0; i < n; i++) {
    if(!inside(ret, v[i])) {
      ret = circle(v[i], 0);
      for(int j = 0; j < i; j++) {
        if(!inside(ret, v[j])) {
          ret = circle((v[i] + v[j]) / 2.0, sqrt(dist2(v[i], v[j])) / 2.0);
          for(int k = 0; k < j; k++) {
            if(!inside(ret, v[k])) {
              p = bestOf3(v[i], v[j], v[k]);
              ret = circle(p, sqrt(dist2(p, v[i])));
            }
          }
        }
      }
    }
  }
  return ret;
}

int main() {
  int n;
  srand(time(NULL));
  BUFF;
  vector<PT> v;
  cin>>n;
  for(int i = 0; i < n; i++) {
    PT p;
    cin>>p.x>>p.y;
    v.pb(p);
  }
  circle c = minCirc(v);
  cout<<setprecision(6)<<fixed;
  cout<<c.cen.x<<" "<<c.cen.y<<" "<<c.r<<endl;
  return 0;
}
```

9 Data Structures

9.1 Disjoint Set Union

```
const int N=500010;
int p[N],Rank[N];
void init()
{
        for(int i=0;i<N;i++) p[i]=i, Rank[i]=1;
}
int findSet(int i)
{
        if(p[i]==i) return i;
        return p[i]=findSet(p[i]);
}
bool sameSet(int i, int j)
{
        return (findSet(i) == findSet(j));
}
void unionSet(int i, int j)
{
        if (!sameSet(i, j)) {
                int x = findSet(i), y=findSet(j);
                if (Rank[x] > Rank[y]){
                        p[y] = x;
                        Rank[x] += Rank[y];
                }
                else {
                        p[x] = y;
                        Rank[y] += Rank[x];
                }
        }
}
```

9.2 Persistent Segment Tree

```
//PRINTAR O NUMERO DE ELEMENTOS DISTINTOS
//EM UM INTERVALO DO ARRAY
const int N = 30010;
int tr[100 * N], L[100 * N], R[100 * N], root[100 * N];
```

```
int v[N], mapa[100 * N];
int cont = 1;
void build(int node, int b, int e)
{
  if (b == e) {
    tr[node] = 0;
  }
  else {
    L[node] = cont++;
    R[node] = cont++;
    build(L[node], b, (b + e) / 2);
    build(R[node], (b + e) / 2 + 1, e);
    tr[node] = tr[L[node]] + tr[R[node]];
  }
}
int update(int node, int b, int e, int i, int val)
{
  int idx = cont++;
  tr[idx] = tr[node] + val;
  L[idx] = L[node];
  R[idx] = R[node];
  if (b == e) return idx;
  int mid = (b + e) / 2;
  if (i <= mid)
    L[idx] = update(L[node], b, mid, i, val);
  else
    R[idx] = update(R[node], mid + 1, e, i, val);
  return idx;
}
int query(int nodeL, int nodeR, int b, int e, int i, int j)
{
  if (b > j or i > e) return 0;
  if (i <= b and j >= e) {
    int p1 = tr[nodeR];
    int p2 = tr[nodeL];
    return p1 - p2;
  }
  int mid = (b + e) / 2;
  return query(L[nodeL], L[nodeR], b, mid, i, j) +
        query(R[nodeL], R[nodeR], mid + 1, e, i, j);
}
int main()
{
  int n;
  sc(n);
  memset(mapa, -1, sizeof(mapa));
  for (int i = 0; i < n; i++) sc(v[i]);
  build(1, 0, n - 1);
  for (int i = 0; i < n; i++) {
    if (mapa[v[i]] == -1) {
      root[i + 1] = update(root[i], 0, n - 1, i, 1);
      mapa[v[i]] = i;
    }
    else {
      root[i + 1] = update(root[i], 0, n - 1, mapa[v[i]], -1);
      mapa[v[i]] = i;
      root[i + 1] = update(root[i + 1], 0, n - 1, i, 1);
    }
  }
  int q;
  sc(q);
  for (int i = 0; i < q; i++) {
    int l, r;
    sc2(l, r);
    int resp = query(root[l - 1], root[r], 0, n - 1, l - 1, r - 1);
    pri(resp);
  }
  return 0;
}
```

9.3 Sparse Table

```
//comutar RMQ , favor inicializar: dp[i][0]=v[0]
//sendo v[0] o vetor do rmq
//chamar o build!
int dp[200100][22];
int n;
int d[200100];
void build()
{
  d[0] = d[1] = 0;
  for (int i = 2; i < n; i++) d[i] = d[i >> 1] + 1;
  for (int j = 1; j < 22; j++) {
    for (int i = 0; i + (1 << (j - 1)) < n; i++) {
      dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
    }
  }
}
int query(int i, int j)
{
  int k = d[j - i];
  int x = min(dp[i][k], dp[j - (1 << k) + 1][k]);
  return x;
}
```

9.4 Cartesian Tree

```
int bigrand() { return (rand()<<16)^rand();}
struct Node{
        int  prior, val, sum, subtr, pref, suf, maximo;
        Node *l, *r;
        Node () {}
        Node (int x) : maximo(x) , val(x), prior(bigrand()), sum(x), subtr(1) , l(NULL), r(NULL), pref
              (x), suf(x){}
};
struct Treap{
        Node *root;
        Treap() : root(NULL) {};

        int cnt(Node *t) {
                if(t) return t->subtr;
                return 0;
        }
        int key(Node *t){
                if(t) return t->val;
                return 0;
        }
        int sum(Node *t) {
                if(t) return t->sum;
                return 0;
        }
        int pref(Node *t){
                if(t) return t->pref;
                return -INF;
        }
        int suf(Node *t){
                if(t) return t->suf;
                return -INF;
        }
        int maximo(Node *t){
                if(t) return t->maximo;
                return -INF;
        }

        void upd(Node* &t){
                if(t){
                        if(!(t->l)){
                                t->pref= max(t->val, t->val + pref(t->r));
                        }
                        else{
                                t->pref= max( pref(t->l), max( sum(t->l) + t->val, sum(t->l) + t->val
                                      + pref(t->r)));
                        }
                        if(!(t->r)){
                                t->suf= max(t->val, t->val + suf(t->l));
                        }
                        else{
                                t->suf= max( suf(t->r), max( sum(t->r) + t->val, sum(t->r) + t->val +
                                      suf(t->l)));
                        }
                        t->maximo= max( suf(t->l) + t->val, suf(t->l) +t->val + pref(t->r));
                        t->maximo = max(t->maximo , pref(t->r) + t->val);
                        t->maximo = max(t->maximo, max( maximo(t->l), maximo(t->r)));
                        t->maximo= max(t->maximo, t->val);
                        t->sum= sum(t->r)  + sum(t->l) + t->val;
                        t->subtr=cnt(t->l) + cnt(t->r) +1 ;
                }
        }
// junta todos menores que val e todos maiores ou iguais a val
        Node* merge(Node* L, Node *R){
                if(!L) return R;
                if(!R) return L;
                if(L->prior > R->prior){
                        L->r = merge(L->r, R);
                        upd(L);
                        return L;
                }
                R->l = merge(L, R->l);
                upd(R);
                return R;
        }
// separa t em todos menores que val , todos maiores ou igual a val
        pair<Node*, Node*> split(Node* t, int val, int add){
                if(!t){
                        return mp(nullptr, nullptr);
```

```
            }
            int cur_key= add+ cnt(t->l);
            if(cur_key < val){
                auto ret= split(t->r, val, cur_key+1);
                t->r= ret.first;
                upd(t);
                return mp(t, ret.second);
            }
            auto ret= split(t->l, val , add);
            t->l = ret.second;
            upd(t);
            return mp(ret.first, t);
        }

        int querymax(Node *&t, int i, int j){
            auto tr1= split(t, j+1, 0);
            auto tr2= split(tr1.first, i, 0);


            int prefi= pref(tr2.second->r);
            int sufi= suf(tr2.second->l);
            int val= key(tr2.second);

            int r=maximo(tr2.second);
            auto x= merge(tr2.first, tr2.second);
            t= merge(x, tr1.second);
            return r;
        }

        void insert(Node* &t, int x, int y){
            Node *aux= new Node(y);
            auto tr= split(t, x,0);
            auto traux=merge(tr.first,aux);
            t=merge(traux,tr.second);
        }

        void replace(Node *&t, int x, int y){
            Node *aux= new Node(y);
            erase(t, x);
            auto tr=split(t, x, 0);
            t=merge(tr.first,aux);
            //db(pref(t));
            //db(suf(t));
            t=merge(t, tr.second);
            //      db(pref(t));
            //      db(suf(t));

        }

        void erase(Node * &t, int x){
            auto tr=split(t,x+1,0);
            auto tr2=split(tr.first, x,0);
            t= merge(tr2.first, tr.second);
        }
};
int main()
{
        int n;
        sc(n);
        Treap T;
        for(int i=0;i<n;i++){
            int x;
            sc(x);
            T.insert(T.root, i, x);
        }
        int q;
        sc(q);
        while(q--){
            //db(T.cnt(T.root));
            char op;
            cin>>op;
            if(op=='I'){
                int x,y;
                sc2(x,y);
                x--;
                T.insert(T.root, x, y);
            }
            else if(op=='Q'){
                int l,r;
                sc2(l,r);
                l--,r--;
                pri(T.querymax(T.root, l,r));
            }
            else if(op=='R'){
                int x,y;
                sc2(x,y);
                x--;
                T.replace(T.root, x, y);
            }
            else{
                int x;
                sc(x);
                x--;
                T.erase(T.root, x);
            }
        }
        return 0;
}
```

9.5 Cartesian Tree 2

```
int bigrand() { return (rand()<<16)^rand();}
char r[500001];
struct Node{
        int prior , subtr, sujo;
        int val,add;
        Node *l, *r;
        Node () {}
        Node (int c) : add(0), val(c), prior(bigrand()), l(NULL), r(NULL), subtr(1) {}
};
struct Treap{
        Node *root;
        Treap() : root(NULL) {};
        int cnt(Node *t){
            if(t) return t->subtr;
            return 0;
        }
        void upd(Node* &t){
            if(t){
                if(t->sujo){
                    swap(t->l, t->r);
                    t->sujo=0;
                    if(t->l){
                        t->l->sujo^=1;
                    }
                    if(t->r){
                        t->r->sujo^=1;
                    }
                }
                t->val+=t->add;
                if(t->l){
                    t->l->add+=t->add;
                }
                if(t->r){
                    t->r->add+=t->add;
                }
                t->add=0;
                t->subtr= cnt(t->l) + cnt(t->r) + 1;
            }
        }
        Node* merge(Node *L, Node *R){
            upd(R);
            upd(L);
            if(!L) return R;
            if(!R) return L;
            if(L-> prior > R->prior){
                L->r = merge(L->r, R);
                upd(L);
                upd(R);
                return L;
            }
            R->l = merge(L,R->l);
            upd(R);
            upd(L);
            return R;
        }
        //<, >= val
        pair<Node*, Node*> split(Node *t, int val, int add){
            if(!t) {
                return mp(nullptr, nullptr);
            }
            upd(t);
            int cur_key= add + cnt(t->l);
            if(cur_key < val){
                auto ret= split(t->r, val , cur_key+1);
                t->r= ret.first;
                upd(t);
                return mp(t, ret.second);
            }
            auto ret= split( t->l, val , add);
            t->l = ret.second;
            upd(t);
            return mp(ret.first, t);
        }

        Node* inverte(Node* &t, int i, int j, int val){
            if(i>j) return t;
            auto tr1= split(t, j+1, 0);
            auto tr2= split(tr1.first, i, 0);
```

```
            if(tr2.second){
                tr2.second->sujo^=1;
                tr2.second->add+=val;
            }
            auto x=merge(tr2.first,tr2.second);
            x=merge(x,tr1.second);
            return x;
        }

        void att(Node* &t, int l , int r, int i, int j){
            t= inverte(t,r+1,i-1,-1);
            t=inverte(t,l,j,1);
        }

        void imprime(Node* &t, int add){
            if(t){
                upd(t);
                int cur_key= add + cnt(t->l);
                imprime(t->l, add);
                imprime(t->r, cur_key+1);
                int aux=t->val+t->add;
                aux%=26;
                aux+=26;
                aux%=26;
                r[cur_key]=aux+'a';
            }
        }

        void poe(Node* &t, string &s){
            for(int i=0;i<s.size();i++){
                Node *aux = new Node(s[i]-'a');
                auto tr= split(t, i, 0);
                auto traux= merge(tr.first, aux);
                t= merge(traux, tr.second);
            }
        }

};
int main()
{
    BUFF;
    int X;
    cin>>X;
    while(X--){
        Treap T;
        string s;
        int op;
        cin>>s>>op;
        T.poe(T.root, s);
        //T.imprime(T.root,0);
        //for(int i=0;i<s.size();i++) {
        //      cout<<r[i];
        //      }
        //cout<<endl;
        //assert(T.root!=NULL);
        while(op--){
            int l,r,i,j;
            cin>>l>>r>>i>>j;
            l--,r--,i--,j--;
            T.att(T.root,l,r,i,j);
        }
        T.imprime(T.root,0);
        for(int i=0;i<s.size();i++) cout<<r[i];
        cout<<endl;
    }
    return 0;
}
```

9.6 Dynamic MST

```
/*
* Code for URI 1887
* It gives an tree and a bunch of queries to add
* edges from a to b with cost c.
*/
const int MOD = 1e9 + 9;
struct ed{
    int u,v,w,t;
    ed(int _u, int _v, int _w, int _t){ u=_u,v=_v,w=_w,t=_t;}
    ed(){};
    bool operator < ( const ed &a) const {
        return w<a.w;
    }
};
const int N=50010;
int p[N],id[N];
void init(int n)
{
    for(int i=1;i<=n;i++) p[i]=i;
}
int findSet(int i)
{
    if(p[i]==i) return i;
    return p[i]=findSet(p[i]);
}
bool unionSet(int i, int j)
{
    int x=findSet(i),y=findSet(j);
    if(x==y) return false;
    p[x]=y;
    return true;
}
void reduction(int l,int r,int &n,vector<ed> &graph,int &res)
{
    vector<ed> g;
    init(n);
    sort(graph.begin(),graph.end());
    for(int i=0;i<graph.size();i++)
        if(graph[i].t<=r and (graph[i].t>=l or unionSet(graph[i].u,graph[i].v))){
            g.pb(graph[i]);
        }
    graph=g;
}
void contraction(int l,int r,int &n,vector<ed> &graph,int &res)
{
    vector<ed> g;
    init(n);
    sort(graph.begin(),graph.end());

    for(int i=0;i<(int)graph.size();i++)
        if(graph[i].t>=l) unionSet(graph[i].u,graph[i].v);

    for(int i=0;i<(int)graph.size();i++){
        if(graph[i].t<l and unionSet(graph[i].u,graph[i].v)){
            g.pb(graph[i]);
            res+=graph[i].w;
        }
    }

    init(n);
    for(int i=0;i<g.size();i++){
        unionSet(g[i].u,g[i].v);
    }

    int tot=0;
    for(int i=1;i<=n;i++) id[i]=0;

    for(int i=1;i<=n;i++){
        int f=findSet(i);
        if(!id[f]) id[f]=++tot;
        id[i]=id[f];
    }
    for(int i=0;i<graph.size();i++){
        graph[i].u=id[graph[i].u],graph[i].v=id[graph[i].v];
    }
    n=tot;
}
void solve(int l,int r,int n,vector<ed> graph,int res)
{
    reduction(l,r,n,graph,res);
    contraction(l,r,n,graph,res);
    if(l==r)
    {
        init(n);
        sort(graph.begin(),graph.end());
        for(int i=0;i<(int)graph.size();i++)
            if(unionSet(graph[i].u,graph[i].v)){
                res+=graph[i].w;
            }
            pri(res);
        return;
    }
    int mid=(l+r)/2;
    solve(l,mid,n,graph,res);
    solve(mid+1,r,n,graph,res);
}
int main()
{
    int T;
    sc(T);
    while(T--)
    {
        int n,m,q;
        sc3(n,m,q);
        vector<ed> graph;
        for(int i=1;i<=m;i++)
        {
            int u,v,w;
            sc3(u,v,w);
```

```
                int t=0;
                graph.pb(ed(u,v,w,t));
            }
            for(int i=1;i<=q;i++)
            {
                int u,v,w;
                sc3(u,v,w);
                int t=i;
                graph.pb(ed(u,v,w,t));
            }
            solve(1,q,n,graph,0);
        }
        return 0;
}
```

10 Miscellaneous

10.1 Invertion Count

```
//conta o numero de inversoes de um array
//x e o tamanho do array, v e o array que quero contar
ll inversoes = 0;
void merge_sort(vi &v, int x)
{
  if (x == 1) return;
  int tam_esq = (x + 1) / 2, tam_dir = x / 2;
  int esq[tam_esq], dir[tam_dir];
  for (int i = 0; i < tam_esq; i++) esq[i] = v[i];
  for (int i = 0; i < tam_dir; i++) dir[i] = v[i + tam_esq];
  merge_sort(esq, tam_esq);
  merge_sort(dir, tam_dir);
  int i_esq = 0, i_dir = 0, i = 0;
  while (i_esq < tam_esq or i_dir < tam_dir) {
    if (i_esq == tam_esq) {
      while (i_dir != tam_dir) {
        v[i] = dir[i_dir];
        i_dir++, i++;
      }
    }
    else if (i_dir == tam_dir) {
      while (i_esq != tam_esq) {
        v[i] = esq[i_esq];
        i_esq++, i++;
        inversoes += i_dir;
      }
    }
    else {
      if (esq[i_esq] <= dir[i_dir]) {
        v[i] = esq[i_esq];
        i++, i_esq++;
        inversoes += i_dir;
      }
      else {
        v[i] = dir[i_dir];
        i++, i_dir++;
      }
    }
  }
}
```

10.2 Distinct Elements in ranges

```
const int MOD = 1e9 + 7;
const int N = 1e6 + 10;
int bit[N], v[N], id[N], r[N];
ii query[N];
int mapa[N];
bool compare(int x, int y) { return query[x] < query[y]; }
void add(int idx, int val)
{
  while (idx < N) {
    bit[idx] += val;
    idx += idx & -idx;
  }
}
int sum(int idx)
{
  int ret = 0;
  while (idx > 0) {
    ret += bit[idx];
    idx -= idx & -idx;
  }
  return ret;
}
int main()
{
  memset(bit, 0, sizeof(bit));
  memset(mapa, 0, sizeof(mapa));
  int n;
  sc(n);
  for (int i = 1; i <= n; i++) sc(v[i]);
  int q;
  sc(q);
  for (int i = 0; i < q; i++) {
    sc2(query[i].second, query[i].first);
    id[i] = i;
  }
  sort(id, id + q, compare);
  sort(query, query + q);
  int j = 1;
  for (int i = 0; i < q; i++) {
    int L = query[i].second;
    int R = query[i].first;
    while (j <= R) {
      if (mapa[v[j]] > 0) {
        add(mapa[v[j]], -1);
        mapa[v[j]] = j;
        add(mapa[v[j]], 1);
      }
      else {
        mapa[v[j]] = j;
        add(mapa[v[j]], 1);
      }
      j++;
    }
    r[id[i]] = sum(R);
    if (L > 1) r[id[i]] -= sum(L - 1);
  }
  for (int i = 0; i < q; i++) pri(r[i]);
  return 0;
}
```

10.3 Maximum Rectangular Area in Histogram

```
/*
 * Complexidade : O(N)
 */

ll solve(vi &h)
{
  int n = h.size();
  ll resp = 0;
  stack<int> pilha;
  ll i = 0;
  while (i < n) {
    if (pilha.empty() or h[pilha.top()] <= h[i]) {
      pilha.push(i++);
    }
    else {
      int aux = pilha.top();
      pilha.pop();
      resp =
          max(resp, (ll)h[aux] * ((pilha.empty()) ? i : i - pilha.top()-1));
    }
  }
  while (!pilha.empty()) {
    int aux = pilha.top();
    pilha.pop();
    resp = max(resp, (ll)h[aux] * ((pilha.empty()) ? n : n - pilha.top()-1));
  }
  return resp;
}
```

10.4 Multiplying Two LL mod n

```
/* Metodo para calcular (a*b) mod m onde a e b s o inteiros com 64-bits cada.
Fonte: http://bit.ly/1pp7xEZ */
ll mulmod(ll a, ll b, ll m) {
        ll y = (ll)( (ld)a*(ld)b/m + (ld)1/2 );
        y = y * m;
        ll x = a * b, r = x - y;
        if ((ll) r < 0) { r = r + m; y = y - 1; }
        return r;
}
```

10.5 Josephus Problem

```
/* Josephus Problem - It returns the position to be, in order to not die. O(n)*/
/* With k=2, for instance, the game begins with 2 being killed and then n+2, n+4, ... */
ll josephus(ll n, ll k) {
  if(n==1) return 1;
  else return (josephus(n-1, k)+k-1)%n+1;
}
```

10.6 Josephus Problem 2

```
/* Another Way to compute the last position to be killed  O ( d * log n ) */
ll josephus(ll n, ll d) {
  ll K = 1;
  while (K <= (d - 1)*n) K = (d * K + d - 2) / (d - 1);
  return d * n + 1 - K;
}
```

10.7 Ordered Static Set (Examples)

```
///USANDO ORDERED STATIC SET PRA ESTRUTURA
//aqui vai o template
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update

using namespace __gnu_pbds;
typedef struct cu {
        int a;
        int b;
        bool operator < (const struct cu &other) const {
                if(a != other.a) return a < other.a;
                return b < other.b;
        }

        bool operator == (const struct cu &other) const {
                return(a == other.a and b == other.b);
        }

}cuzao;

bool cmp(const cuzao &a, const cuzao &b) {
        return true;
}

typedef tree<
cuzao,
        null_type,
        less<cuzao>,
        rb_tree_tag,
        tree_order_statistics_node_update>
        ordered_set;


int main()
{
        ordered_set os;
        cuzao asd;
        asd.a = 1;
        asd.b = 2;
        os.insert(asd);
        asd.a = 4;
        os.insert(asd);
        cout<<(os.find(asd) == end(os))<<endl;//0
        cout<<os.order_of_key(asd)<<endl;//1
        asd.a = 1;
        cout<<os.order_of_key(asd)<<endl;//0
        cout<<os.find_by_order(0)->a<<" "<<os.find_by_order(0)->b<<endl;//1 2
        cout<<os.find_by_order(1)->a<<" "<<os.find_by_order(1)->b<<endl;//4 2
        return 0;
}



//aqui vai o template
//USANDO ORDERED STATIC SET PRA CONTAINER DO STL MESMO
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update

using namespace __gnu_pbds;

typedef tree<
int,
        null_type,
        less<int>,
        rb_tree_tag,
        tree_order_statistics_node_update>
        ordered_set;//n    multi


int main()
{
        ordered_set os;
        os.insert(1);
        os.insert(10);
        os.insert(1);
        os.insert(15);
        cout<<(os.find(10) == end(os))<<endl;//0 mesma coisa q !count
        cout<<os.order_of_key(10)<<endl;//1 qual o indice do valor 10, se n tem o indice, pega o
             proximo
        cout<<os.order_of_key(2)<<endl;//1
        cout<<*os.upper_bound(2)<<endl;//10
        cout<<*os.find_by_order(0)<<endl;//1
        cout<<*os.find_by_order(2)<<endl;//15
        return 0;
}
```