magUNbos' notebook (2017)

Contents

1 Graphs

1.1 Articulation points

```
/*Tarjan Algorithm to find articulation points
 * single dfs O(|v| + |e|)
 * visited =[false]
 * disc = [0]
 * low = [0]
 * parent = [-1]
 * ap = [false]
 * result is stored in ap boolean array

 * Tested in AIZU online Judge*/
#include<bist/stdc++.h>

using namespace std;

const int SIZE = 100013;

bool visited[SIZE], ap[SIZE];
int disc[SIZE], low[SIZE], parent[SIZE];

vector<int> G[SIZE];

void articulation(int u){
  static int time = 0;
  int children = 0;
  visited[u] = true;
  disc[u] = low[u] = ++time;
  for(int i = 0; i < G[u].size(); i++){
      int v = G[u][i];
      if(!visited[v]){
          children++;
          parent[v] = u;
          articulation(v);
          low[u] = min(low[u], low[v]);
          if(parent[u] == -1 && children > 1)
            ap[u] = true;
          if(parent[u] != -1 && low[v] >= disc[u])
            ap[u] = true;
      }
      else if(v != parent[u])
        low[u] = min(low[u], disc[v]);
  }
}
```

1.2 Biconected graph

```
/* Tarjan Algorithm to find Biconnected graph
 * single dfs O(|v| + |e|)
 * visited =[false]
 * disc = [0]
 * low = [0]
 * parent = [-1]  */

bool isBiconnected(vector<vector<int> > G, int u, bool visited[],
    int disc[], int low[], int parent[]){
    static int time = 0;

    int children = 0;

    visited[u] = true;

    disc[u] = low[u] = ++time;
```

```
    for(int i = 0; i<G[u].size(); i++){
        int v = G[u][i];

        if(!visited[v]){
            children++;
            parent[v] = u;

            if (isBiconnected(G, v, visited, disc, low, parent))
                return true;

            low[u] = min(low[u], low[v]);

            if(parent[u] == -1 && children > 1) return true;

            if(parent[u] != -1 && low[v] >= disc[u]) return true;
        }
        else if(v != parent[v]) low[u] = min(low[u], disc[v]);
    }
    return false;
}
```

1.3 Bridges

```
/* Tarjan Algorithm to find bridges
 * single dfs O(|v| + |E|)
 * visited =[false]
 * disc = [0]
 * low = [0]
 * parent = [-1]
 * the priority queue orders the bridges in ascending order,
 * use the function like: bridges(0, &queue)

 * tested in AIZU online Judge
 */
#include<bits/stdc++.h>

using namespace std;

const int SIZE = 100013;

typedef pair<int, int> pii;

bool visited[SIZE];
int disc[SIZE], low[SIZE], parent[SIZE];

vector<int> G[SIZE];


void bridges(int u, priority_queue<pii, vector<pii>, greater<pii>
    > *bridge){
  static int time = 0;
  int children = 0;
  visited[u] = true;
  disc[u] = low[u] = ++time;
  for(int i = 0; i < G[u].size(); i++){
    int v = G[u][i];
    if(!visited[v]){
      children++;
      parent[v] = u;
      bridges(v, bridge);
      low[u] = min(low[u], low[v]);
      if(low[v] > disc[u]) bridge->push({min(u,v),max(u,v)});
    }
    else if(v != parent[u])
      low[u] = min(low[u], disc[v]);
  }
}
```

1.4 Tarjan SCC

```
/* Tarjan Algorithm to find connected components
 * single dfs O(|v| + |e|)
 * visited =[false]
 * disc = [0]
 * low = [0]
 * parent = [-1]
 * tested on AIZU online Judge */

void dfsSCC(vector<vector<int> > G, int u, int disc[], int low[],
     stack<int> *st, bool stackMember[]){
    static int time = 0;

    disc[u] = low[u] = ++time;

    st->push(u);
    stackmember[u] = true;

    for(int i = 0; i < G[u].size(); i++){
        int v = G[u][i];

        if(disc[v] == -1){

            dfsSCC(G, v, disc, low, st, stackmember);

            low[u] = min(low[u], low[v]);
        }
        else if(stackmember[v] == true) low[u] = min(low[u], disc
            [v]);
    }

    int w = 0;

    if(low[u] == disc[u]){

        while(st->top() != u){
            w = st->top();
            cout<<w<<" ";
            stackmember[w] = false;
            st->pop();
        }

        w = st->top();
        cout<<w<<"\n";
        stackmember[w] = false;
        st->pop();
    }
}

void scc(G){
   int *disc = new int[V];
   int *low = new int[V];
   bool *stackMember = new bool[V];
   stack<int> *st = new stack<int>();

   memset(disc, -1, sizeof(disc));
   memset(low, 0, sizeof(low));
   memset(stackMember, false, sizeof(stackMember));

   for(int i = 0; i < G.size(); i++)
       if(disc[i] == -1) dfsScc(G, i, disc, low, st, stackMember)
           ;
}
```

1.5 Bellman Ford

```
/* Bellman Ford's algorithm to find SSSP with negative cycles O(V
   ^3)
 * returns true if it detects a negative cycle
 * dist [i] contains distance to i
 * parent[i] contains parent on path to i from source

 * tested on AIZU online judge
 */
#include<bits/stdc++.h>

const int SIZE = 1013;
const int oo = 999999;

typedef pair<int, int> pii;

int dist[SIZE];
int parent[SIZE];

bool bellmanFord(vector<vector<pii> > &G, int &source, int &N){
    for(int i = 0; i < N; i++)
      dist[i] = oo;
    dist[source] = 0;
    for(int k = 0; k < N; k++){
      for(int u = 0; u < N; u++){
        for(int i = 0; i < G[u].size(); i++){
          pii v = G[u][i];
          if(dist[v.first] > dist[u] + v.second && dist[u] != oo){
            if(k == N-1)
              return true;
            dist[v.first] = dist[u] + v.second;
            parent[v.first] = u;
          }
        }
      }
    }
    return false;
}
```

1.6 Eulerian Path

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
        int next_vertex;
        iter reverse_edge;

        Edge(int next_vertex)
                :next_vertex(next_vertex)
                { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];           // adjacency list

vector<int> path;

void find_path(int v)
{
        while(adj[v].size() > 0)
        {
                int vn = adj[v].front().next_vertex;
                adj[vn].erase(adj[v].front().reverse_edge);
                adj[v].pop_front();
                find_path(vn);
        }
        path.push_back(v);
}

void add_edge(int a, int b)
{
        adj[a].push_front(Edge(b));
        iter ita = adj[a].begin();
        adj[b].push_front(Edge(a));
        iter itb = adj[b].begin();
        ita->reverse_edge = itb;
        itb->reverse_edge = ita;
}
```

1.7 Topological sort

```
char c[TAM];
int l[TAM];
int r[TAM];
int in[TAM];

//can be priority queue
queue<int> Q;

void reset(){
    memset(l, 0, sizeof l);
    memset(r, 0, sizeof r);
    memset(in, 0, sizeof in);
    memset(balls, 0, sizeof balls);
    c[0] = 'L';
}

void topo(vector<vector<int> > G, int u){
    while(!Q.empty()){
        u = Q.front();Q.pop();
        update(u);
        for(int i = 0; i <G[u].size(); i++){
            int v = G[u][i];
            in[v]--;
            if(in[v] == 0) Q.push(v);
        }
    }
}

int main(){
    ll n;
    int m;
    while(cin>>n>>m){
        reset();
        vector< vector<int> > G(m + 1);
        for(int i = 1; i <=m; i++){
            int u,v;
            cin>>c[i]>>u>>v;
            G[i].push_back(u);
            G[i].push_back(v);
            in[u] ++; in[v]++;
            l[i] = u;r[i] = v;
        }
        for(int i = 1; i <=m; i++){
            if(in[i] == 0)Q.push(i);
        }
        topo(G, 1);
```

```
    }
}
```

1.8 Kruskal's minimum spanning tree

```
/*
Uses Kruskal's Algorithm to calculate the weight of the minimum
    spanning
forest (union of minimum spanning trees of each connected
    component) of
a possibly disjoint graph, given in the form of a matrix of edge
    weights
(-1 if no edge exists). Returns the weight of the minimum spanning
forest (also calculates the actual edges - stored in T). Note:
    uses a
disjoint-set data structure with amortized (effectively) constant
    time per
union/find. Runs in O(E*log(E)) time.

tested on AIZU online Judge
*/

#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

typedef int T;

struct edge
{
  int u, v;
  T d;
};

struct edgeCmp
{
  int operator()(const edge& a, const edge& b) { return a.d > b.d;
     }
};

int find(vector <int>& C, int x) { return (C[x] == x) ? x : C[x] =
     find(C, C[x]); }

T Kruskal(vector <vector <T> >& w)
{
  int n = w.size();
  T weight = 0;

  vector <int> C(n), R(n);
  for(int i=0; i<n; i++) { C[i] = i; R[i] = 0; }

  vector <edge> T;
  priority_queue <edge, vector <edge>, edgeCmp> E;

  for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++)
      if(w[i][j] >= 0)
      {
        edge e;
        e.u = i; e.v = j; e.d = w[i][j];
        E.push(e);
      }

  while(T.size() < n-1 && !E.empty())
  {
    edge cur = E.top(); E.pop();

    int uc = find(C, cur.u), vc = find(C, cur.v);
    if(uc != vc)
    {
      T.push_back(cur); weight += cur.d;

      if(R[uc] > R[vc]) C[vc] = uc;
      else if(R[vc] > R[uc]) C[uc] = vc;
      else { C[vc] = uc; R[uc]++; }
    }
  }

  return weight;
}

int main()
{
  int wa[6][6] = {
    { 0, -1, 2, -1, 7, -1 },
    { -1, 0, -1, 2, -1, -1 },
    { 2, -1, 0, -1, 8, 6 },
    { -1, 2, -1, 0, -1, -1 },
    { 7, -1, 8, -1, 0, 4 },
    { -1, -1, 6, -1, 4, 0 } };

  vector <vector <int> > w(6, vector <int>(6));

  for(int i=0; i<6; i++)
    for(int j=0; j<6; j++)
      w[i][j] = wa[i][j];

  cout << Kruskal(w) << endl;
  cin >> wa[0][0];
}
```

1.9 Lowest Common ancestor

```
const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes];         // children[i] contains
    the children of node i
int A[max_nodes][log_max_nodes+1];       // A[i][j] is the 2^j-th
    ancestor of node i, or -1 if that ancestor does not exist
int L[max_nodes];                        // L[i] is the distance
    between node i and the root

// floor of the binary logarithm of n
int lb(unsigned int n)
{
    if(n==0)
        return -1;
    int p = 0;
    if (n >= 1<<16) { n >>= 16; p += 16; }
    if (n >= 1<< 8) { n >>=  8; p +=  8; }
    if (n >= 1<< 4) { n >>=  4; p +=  4; }
    if (n >= 1<< 2) { n >>=  2; p +=  2; }
    if (n >= 1<< 1) {           p +=  1; }
    return p;
}

void DFS(int i, int l)
{
    L[i] = l;
    for(int j = 0; j < children[i].size(); j++)
        DFS(children[i][j], l+1);
}
```

```
int LCA(int p, int q)
{
    // ensure node p is at least as deep as node q
    if(L[p] < L[q])
        swap(p, q);

    // "binary search" for the ancestor of node p situated on the
        same level as q
    for(int i = log_num_nodes; i >= 0; i--)
        if(L[p] - (1<<i) >= L[q])
            p = A[p][i];

    if(p == q)
        return p;

    // "binary search" for the LCA
    for(int i = log_num_nodes; i >= 0; i--)
        if(A[p][i] != -1 && A[p][i] != A[q][i])
        {
            p = A[p][i];
            q = A[q][i];
        }

    return A[p][0];
}

int main(int argc,char* argv[])
{
    // read num_nodes, the total number of nodes
    log_num_nodes=lb(num_nodes);

    for(int i = 0; i < num_nodes; i++)
    {
        int p;
        // read p, the parent of node i or -1 if node i is the
            root

        A[i][0] = p;
        if(p != -1)
            children[p].push_back(i);
        else
            root = i;
    }

    // precompute A using dynamic programming
    for(int j = 1; j <= log_num_nodes; j++)
        for(int i = 0; i < num_nodes; i++)
            if(A[i][j-1] != -1)
                A[i][j] = A[A[i][j-1]][j-1];
            else
                A[i][j] = -1;

    // precompute L
    DFS(root, 0);


    return 0;
}
```

2 Geometry

2.1 Basics

```
typedef complex<double> point;
typedef vector<point> polygon;

#define NEXT(i) (((i) + 1) % n)

struct circle { point p; double r; };
struct line { point p, q; };
using segment = line;

const double eps = 1e-9;

// fix comparations on doubles with this two functions
int sign(double x) { return x < -eps ? -1 : x > eps; }

int dblcmp(double x, double y) { return sign(x - y); }

double dot(point a, point b) { return real(conj(a) * b); }

double cross(point a, point b) { return imag(conj(a) * b); }

double area2(point a, point b, point c) { return cross(b - a, c -
    a); }

int ccw(point a, point b, point c)
{
        b -= a; c -= a;
        if (cross(b, c) > 0) return +1; // counter clockwise
        if (cross(b, c) < 0) return -1; // clockwise
        if (dot(b, c) < 0) return +2; // c--a--b on line
        if (dot(b, b) < dot(c, c)) return -2; // a--b--c on line
        return 0;
}

namespace std
{
        bool operator<(point a, point b)
        {
                if (a.real() != b.real())
                        return a.real() < b.real();
                return a.imag() < b.imag();
        }
}
```

2.2 Circle

```
/*
        Circles

        Tested: AIZU
*/

// circle-circle intersection
vector<point> intersect(circle C, circle D)
{
        double d = abs(C.p - D.p);
        if (sign(d - C.r - D.r) > 0) return {};      // too far
        if (sign(d - abs(C.r - D.r)) < 0) return {}; // too close
        double a = (C.r*C.r - D.r*D.r + d*d) / (2*d);
        double h = sqrt(C.r*C.r - a*a);
        point v = (D.p - C.p) / d;
        if (sign(h) == 0) return {C.p + v*a};         // touch
        return {C.p + v*a + point(0,1)*v*h,           // intersect
                        C.p + v*a - point(0,1)*v*h};
}

// circle-line intersection
vector<point> intersect(line L, circle C)
{
        point u = L.p - L.q, v = L.p - C.p;
```

```
        double a = dot(u, u), b = dot(u, v), c = dot(v, v) - C.r*C
            .r;
        double det = b*b - a*c;
        if (sign(det) < 0) return {};             // no solution
        if (sign(det) == 0) return {L.p - b/a*u}; // touch
        return {L.p + (-b + sqrt(det))/a*u,
                        L.p + (-b - sqrt(det))/a*u};
}

// circle tangents through point
vector<point> tangent(point p, circle C)
{
        // not tested enough

        double D = abs(p - C.p);

        if (D + eps < C.r) return {};
        point t = C.p - p;

        double theta = asin( C.r / D );
        double d = cos(theta) * D;

        t = t / abs(t) * d;
        if ( abs(D - C.r) < eps ) return {p + t};
        point rot( cos(theta), sin(theta) );
        return {p + t * rot, p + t * conj(rot)};
}

bool incircle(point a, point b, point c, point p)
{
        a -= p; b -= p; c -= p;
        return norm(a) * cross(b, c)
                        + norm(b) * cross(c, a)
                        + norm(c) * cross(a, b) >= 0;
                        // < : inside, = cocircular, > outside
}

point three_point_circle(point a, point b, point c)
{
        point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
        return (y - x) / (conj(x) * y - x * conj(y)) + a;
}

/*
    Get the center of the circles that pass through p0 and p1
    and has ratio r.

    Be careful with epsilon.
*/
vector<point> two_point_ratio_circle(point p0, point p1, double r)
    {
    if (abs(p1 - p0) > 2 * r + eps) // Points are too far.
        return {};

    point pm = (p1 + p0) / 2.0l;
    point pv = p1 - p0;

    pv = point(-pv.imag(), pv.real());

    double x1 = p1.real(), y1 = p1.imag();
    double xm = pm.real(), ym = pm.imag();
    double xv = pv.real(), yv = pv.imag();

    double A = (sqr(xv) + sqr(yv));
    double C = sqr(xm - x1) + sqr(ym - y1) - sqr(r);
    double D = sqrt( - 4 * A * C );
    double t = D / 2.0 / A;

    if (abs(t) <= eps)
        return {pm};
```

```
        return {c1, c2};
}

/*
        Area of the intersection of a circle with a polygon
        Circle's center lies in (0, 0)
        Polygon must be given counterclockwise

        Tested: LightOJ 1358
        Complexity: O(n)
*/

#define x(_t) (xa + (_t) * a)
#define y(_t) (ya + (_t) * b)

double radian(double xa, double ya, double xb, double yb)
{
        return atan2(xa * yb - xb * ya, xa * xb + ya * yb);
}

double part(double xa, double ya, double xb, double yb, double r)
{
        double l = sqrt((xa - xb) * (xa - xb) + (ya - yb) * (ya -
            yb));
        double a = (xb - xa) / l, b = (yb - ya) / l, c = a * xa +
            b * ya;
        double d = 4.0 * (c * c - xa * xa - ya * ya + r * r);
        if (d < eps)
                return radian(xa, ya, xb, yb) * r * r * 0.5;
        else
        {
                d = sqrt(d) * 0.5;
                double s = -c - d, t = -c + d;
                if (s < 0.0) s = 0.0;
                else if (s > l) s = l;
                if (t < 0.0) t = 0.0;
                else if (t > l) t = l;
                return (x(s) * y(t) - x(t) * y(s)
                                + (radian(xa, ya, x(s), y(s))
                                + radian(x(t), y(t), xb, yb)) * r
                                    * r) * 0.5;
        }
}

double intersection_circle_polygon(const polygon &P, double r)
{
        double s = 0.0;
        int n = P.size();;
        for (int i = 0; i < n; i++)
                s += part(P[i].real(), P[i].imag(),
                        P[NEXT(i)].real(), P[NEXT(i)].imag(), r);
        return fabs(s);
}
```

2.3 Centroid

```
/*
        Centroid of a (possibly nonconvex) polygon
        Coordinates must be listed in a cw or ccw.

        Tested: SPOJ STONE
        Complexity: O(n)
*/

point centroid(const polygon &P)
{
```

```
	point c(0, 0);
	double scale = 3.0 * area2(P); // area2 = 2 * polygon_area
	for (int i = 0, n = P.size(); i < n; ++i)
	{
		int j = NEXT(i);
		c = c + (P[i] + P[j]) * (cross(P[i], P[j]));
	}
	return c / scale;
}
```

2.4 Distance between closest points

```
/*
	Compute distance between closest points.

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.5A
	Complexity: O(n log n)
*/

double closest_pair_points(vector<point> &P)
{
	auto cmp = [](point a, point b)
	{
		return make_pair(a.imag(), a.real())
				< make_pair(b.imag(), b.real());
	};

	int n = P.size();
	sort(P.begin(), P.end());

	set<point, decltype(cmp)> S(cmp);
	const double oo = 1e9; // adjust
	double ans = oo;

	for (int i = 0, ptr = 0; i < n; ++i)
	{
		while (ptr < i && abs(P[i].real() - P[ptr].real())
		     >= ans)
			S.erase(P[ptr++]);

		auto lo = S.lower_bound(point(-oo, P[i].imag() -
		    ans - eps));
		auto hi = S.upper_bound(point(-oo, P[i].imag() +
		    ans + eps));

		for (decltype(lo) it = lo; it != hi; ++it)
			ans = min(ans, abs(P[i] - *it));

		S.insert(P[i]);
	}

	return ans;
}
```

2.5 Area Of Polygon

```
/*
	Tested: AIZU(judge.u-aizu.ac.jp) CGL.3A
	Complexity: O(n)
*/

double area2(const polygon &P)
{
	double A = 0;
	for (int i = 0, n = P.size(); i < n; ++i)
		A += cross(P[i], P[NEXT(i)]);
	return A;
}
```

2.6 Antipodal points

```
/*
	Antipodal points
	the antipodal point of a point on the surface of a sphere
	    is the point which is diametrically opposite to it

	Tested: AIZU(judge.u-aizu.ac.jp) CGL.4B
	Complexity: O(n)
*/

vector<pair<int, int>> antipodal(const polygon &P)
{
	vector<pair<int, int>> ans;
	int n = P.size();

	if (P.size() == 2)
		ans.push_back({ 0, 1 });

	if (P.size() < 3)
		return ans;

	int q0 = 0;

	while (abs(area2(P[n - 1], P[0], P[NEXT(q0)]))
			> abs(area2(P[n - 1], P[0], P[q0])))
		++q0;

	for (int q = q0, p = 0; q != 0 && p <= q0; ++p)
	{
		ans.push_back({ p, q });

		while (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
				> abs(area2(P[p], P[NEXT(p)], P[q
				    ])))
		{
			q = NEXT(q);
			if (p != q0 || q != 0)
				ans.push_back({ p, q });
			else
				return ans;
		}

		if (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
				== abs(area2(P[p], P[NEXT(p)], P[q
				    ])))
		{
			if (p != q0 || q != n - 1)
				ans.push_back({ p, NEXT(q) });
			else
				ans.push_back({ NEXT(p), q });
		}
	}

	return ans;
}
```

2.7 Semiplane Intersection

```
/*
	Check wether there is a point in the intersection of
	several semi-planes. if p lies in the border of some
```

```
        semiplane it is considered to belong to the semiplane.

        Expected Running time: linear

        Tested on Triathlon [Cuban Campament Contest]
*/
bool intersect( vector<line> semiplane ){

        function<bool(line&,point&)> side = [](line &l, point &p){
                // IMPORTANT: point p belongs to semiplane defined
                    by l
                // iff p it's clockwise respect to segment < l.p,
                    l.q >
                // i.e. (non negative cross product)

                return cross( l.q - l.p, p - l.p ) >= 0;
        };

        function<bool(line&, line&, point&)> crosspoint = [](const
             line &l, const line &m, point &x){
                double A = cross(l.q - l.p, m.q - m.p);
                double B = cross(l.q - l.p, l.q - m.p);
                if (abs(A) < eps) return false;
                x = m.p + B / A * (m.q - m.p);
                return true;
        };

        int n = (int)semiplane.size();

        random_shuffle( semiplane.begin(), semiplane.end() );

        point cent(0, 1e9);

        for (int i = 0; i < n; ++i){
                line &S = semiplane[ i ];

                if (side(S, cent)) continue;

                point d = S.q - S.p; d /= abs( d );

                point A = S.p - d * 1e8, B = S.p + d * 1e8;

                for (int j = 0; j < i; ++j){
                        point x;
                        line &T = semiplane[j];

                        if ( crosspoint(T, S, x) ){
                                int cnt = 0;


                                if (!side(T, A)){
                                        A = x;
                                        cnt++;
                                }

                                if (!side(T, B)){
                                        B = x;
                                        cnt++;
                                }

                                if (cnt == 2)
                                        return false;
                        }
                        else{
                                if (!side(T, A)) return false;
                        }
                }

                if (imag(B) > imag(A)) swap(A, B);
                cent = A;
        }
        return true;
}
```

2.8 Point inside a Polygon

```
/*
        Determine the position of a point relative
        to a polygon.

        Tested: AIZU(judge.u-aizu.ac.jp) CGL.3C
        Complexity: O(n)
*/
enum { OUT, ON, IN };
int contains(const polygon &P, const point &p)
{
        bool in = false;
        for (int i = 0, n = P.size(); i < n; ++i)
        {
                point a = P[i] - p, b = P[NEXT(i)] - p;
                if (imag(a) > imag(b)) swap(a, b);
                if (imag(a) <= 0 && 0 < imag(b))
                        if (cross(a, b) < 0) in = !in;
                if (cross(a, b) == 0 && dot(a, b) <= 0)
                        return ON;
        }
        return in ? IN : OUT;
}
```

2.9 Convex Cut

```
/*
        Cut a convex polygon by a line and
        return the part to the left of the line

        Tested: AIZU(judge.u-aizu.ac.jp) CGL.4C
        Complexity: O(n)
*/
polygon convex_cut(const polygon &P, const line &l)
{
        polygon Q;
        for (int i = 0, n = P.size(); i < n; ++i)
        {
                point A = P[i], B = P[(i + 1) % n];
                if (ccw(l.p, l.q, A) != -1) Q.push_back(A);
                if (ccw(l.p, l.q, A) * ccw(l.p, l.q, B) < 0)
                        Q.push_back(crosspoint((line){ A, B }, l))
                            ;
        }
        return Q;
}
```

2.10 Polygon width

```
/*
        Compute the width of a convex polygon

        Tested: LiveArchive 5138
        Complexity: O(n)
*/
const int oo = 1e9; // adjust
```

```
double check(int a, int b, int c, int d, const polygon &P)
{
	for (int i = 0; i < 4 && a != c; ++i)
	{
		if (i == 1) swap(a, b);
		else swap(c, d);
	}
	if (a == c) // a admits a support line parallel to bd
	{
		double A = abs(area2(P[a], P[b], P[d]));
		// double of the triangle area
		double base = abs(P[b] - P[d]);
		// base of the triangle abd
		return A / base;
	}
	return oo;
}

double polygon_width(const polygon &P)
{
	if (P.size() < 3)
		return 0;

	auto pairs = antipodal(P);
	double best = oo;
	int n = pairs.size();

	for (int i = 0; i < n; ++i)
	{
		double tmp = check(pairs[i].first, pairs[i].second
			,
				pairs[NEXT(i)].first, pairs[NEXT(i
				)].second, P);
		best = min(best, tmp);
	}

	return best;
}
```

2.11 Sweep Line (Rectangles)

```
/*
	Tested: MIT 2008 Team Contest 1 (Rectangles)
	Complexity: O(n log n)
*/

typedef long long ll;

struct rectangle
{
	ll xl, yl, xh, yh;
};

ll rectangle_area(vector<rectangle> &rs)
{
	vector<ll> ys; // coordinate compression
	for (auto r : rs)
	{
		ys.push_back(r.yl);
		ys.push_back(r.yh);
	}
	sort(ys.begin(), ys.end());
	ys.erase(unique(ys.begin(), ys.end()), ys.end());

	int n = ys.size(); // measure tree
	vector<ll> C(8 * n), A(8 * n);
	function<void(int, int, int, int, int, int)> aux =
		[&](int a, int b, int c, int l, int r, int
			k)
		{
			if ((a = max(a,l)) >= (b = min(b,r
				))) return;
			if (a == l && b == r) C[k] += c;
			else
			{
				aux(a, b, c, l, (l+r)/2,
					2*k+1);
				aux(a, b, c, (l+r)/2, r,
					2*k+2);
			}
			if (C[k]) A[k] = ys[r] - ys[l];
			else A[k] = A[2*k+1] + A[2*k+2];
		};

	struct event
	{
		ll x, l, h, c;
	};
	// plane sweep
	vector<event> es;
	for (auto r : rs)
	{
		int l = lower_bound(ys.begin(), ys.end(), r.yl) -
			ys.begin();
		int h = lower_bound(ys.begin(), ys.end(), r.yh) -
			ys.begin();
		es.push_back({ r.xl, l, h, +1 });
		es.push_back({ r.xh, l, h, -1 });
	}
	sort(es.begin(), es.end(), [](event a, event b)
		{return a.x != b.x ? a.x < b.x : a.c > b.c
			;});
	ll area = 0, prev = 0;
	for (auto &e : es)
	{
		area += (e.x - prev) * A[0];
		prev = e.x;
		aux(e.l, e.h, e.c, 0, n, 0);
	}
	return area;
}
```

2.12 Rectilinear minimum spanning tree

```
/*
	Tested: USACO OPEN08 (Cow Neighborhoods)
	Complexity: O(n log n)
  the rectilinear minimum spanning tree (RMST)
  of a set of n points in the plane is a minimum spanning tree of
	that set,
  where the weight of the edge between each pair of points is the
	rectilinear
  distance between those two points.
*/

typedef long long ll;
typedef complex<ll> point;

ll rectilinear_mst(vector<point> ps)
{
	vector<int> id(ps.size());
	iota(id.begin(), id.end(), 0);
```

```
	struct edge
	{
		int src, dst;
		ll weight;
	};

	vector<edge> edges;
	for (int s = 0; s < 2; ++s)
	{
		for (int t = 0; t < 2; ++t)
		{
			sort(id.begin(), id.end(), [&](int i, int j)
			{
				return real(ps[i] - ps[j]) < imag(ps[j] - ps[i]);
			});

			map<ll, int> sweep;

			for (int i : id)
			{
				for (auto it = sweep.lower_bound(-imag(ps[i]));
						it != sweep.end(); sweep.erase(it++))
				{
					int j = it->second;
					if (imag(ps[j] - ps[i]) < real(ps[j] - ps[i]))
						break;
					ll d = abs(real(ps[i] - ps[j]))
						+ abs(imag(ps[i] - ps[j]));
					edges.push_back({ i, j, d });
				}
				sweep[-imag(ps[i])] = i;
			}

			for (auto &p : ps)
				p = point(imag(p), real(p));
		}

		for (auto &p : ps)
			p = point(-real(p), imag(p));
	}

	ll cost = 0;
	sort(edges.begin(), edges.end(), [](edge a, edge b)
	{
		return a.weight < b.weight;
	});

	union_find uf(ps.size());
	for (edge e : edges)
		if (uf.join(e.src, e.dst))
			cost += e.weight;

	return cost;
}
```

2.13 Points 3d

```
const double pi = acos(-1.0);

// Construct a point on a sphere with center on the origin and radius R
// TESTED [COJ-1436]
struct vec{
	double x, y, z;
	vec (double x=0, double y=0, double z=0) : x(x), y(y), z(z) {}

	vec operator+(const vec a) const{
		return vec(x + a.x, y + a.y, z + a.z);
	}
	vec operator-(const vec a) const{
		return vec(x - a.x, y - a.y, z - a.z);
	}
	vec operator*(const double k) const{
		return vec(k * x, k * y, k * z);
	}
	vec operator/(const double k) const{
		return vec(x / k, y / k, z / k);
	}
	vec operator*(const vec a) const{
		return vec(y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x);
	}
	double dot(const vec a) const{
		return x * a.x + y * a.y + z * a.z;
	}
};

ostream &operator<<(ostream &os, const vec &p)
{
	cout << "(" << p.x << ";" << p.y << ";" << p.z << ")" << endl;
	return os;
}

double abs(vec p)
{
	return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}

vec from_polar(double lat, double lon, double R)
{
	lat = lat / 180.0 * pi;
	lon = lon / 180.0 * pi;
	return vec(R * cos(lat) * sin(lon),
			R * cos(lat) * cos(lon), R * sin(lat));
}

struct plane
{
	double A, B, C, D;
};

/*
 Geodisic distance between points in a sphere
 R is the radius of the sphere
 */
double geodesic_distance(vec p, vec q, double r)
{
	return r * acos(p * q / r / r);
}
```

```
// Find the rect of intersection of two planes on the space
// The rect is given parametrical
// TESTED [TIMUS 1239]
void planePlaneIntersection(plane p, plane q)
{
	if (abs(p.C * q.B - q.C * p.B) < eps)
		return; // Planes are parallel

	double mz = (q.A * p.B - p.A * q.B) / (p.C * q.B - q.C * p
	    .B);
	double nz = (q.D * p.B - p.D * q.B) / (p.C * q.B - q.C * p
	    .B);

	double my = (q.A * p.C - p.A * q.C) / (p.B * q.C - p.C * q
	    .B);
	double ny = (q.D * p.C - p.D * q.C) / (p.B * q.C - p.C * q
	    .B);

	// parametric rect: (x, my * x + ny, mz * x * nz)
}
```

2.14 Convex Hull 3d

```
/*
	Convex Hull 3d implementation.
	Coplanar points case not handled.

	Complexity: O(n^2)
    Tested: opencup 010376(Stars in a Can)
*/
struct face{
	int I[3];
	vec normal;
	int operator[](const int idx) const{
		return I[idx];
	}
};

vector<face> triangulation(vector<vec> &cloud){
	vector<face> F(4);
	for (int i = 0; i < 4; ++i){
		for (int j = 0; j < 3; ++j)
			F[i].I[j] = j + (j >= i);
		F[i].normal = ( cloud[ F[i][1] ] - cloud[ F[i][0]
		    ]) *
					     ( cloud[ F[i][2] ] -
					       cloud[ F[i][0] ]);

		if (F[i].normal.dot( cloud[i] - cloud[ F[i][0] ] )
		     > 0){
			F[i].normal = F[i].normal * -1.;
			swap( F[i].I[1], F[i].I[2] );
		}
	}

	int n = (int)cloud.size();
	vector<vi> cnt(n, vi(n));
	vector<vi> tmr(n, vi(n));

	auto mark = [&](int u, int v, int i){
		if (u > v) swap(u, v);
		if (tmr[u][v] != i){
			tmr[u][v] = i;
			cnt[u][v] = 0;
		}
		cnt[u][v]++;
	};

	auto get  = [&](int u, int v){
		if (u > v) swap(u, v);
		return cnt[u][v];
	};

	for (int i = 4; i < n; ++i){
		vector<face> QF;
		vec x = cloud[i];

		for (auto f : F){
			if (f.normal.dot( x - cloud[f[0]] ) > 0){
				mark(f[0], f[1], i);
				mark(f[1], f[2], i);
				mark(f[2], f[0], i);
			}
			else
				QF.push_back( f );
		}

		for (auto f : F){
			if (f.normal.dot( x - cloud[f[0]] ) > 0){
				for (int j = 0; j < 3; ++j){
					int a = f[j], b = f[j != 2
					    ? j + 1 : 0];
					if (get(a, b) != 1)
					    continue;
					vec u = cloud[a], v =
					    cloud[b];
					vec ref = cloud[ f[j ? j -
					    1 : 2] ];

					face qf = {i, a, b, vec()
					    };
					qf.normal = (u - x) * (v -
					    x);

					if (qf.normal.dot( ref - x
					    ) > 0){
						qf.normal = qf.
						    normal * -1.;
						swap(qf.I[1], qf.I
						    [2]);
					}

					QF.push_back(qf);
				}
			}
		}
		F.swap( QF );
	}

	return F;
}

/*
	Convex Hull 3d implementation.
	Complexity O(n^4)
	It works ok with coplanar points

	UNTESTED [With the new vec(point3) structure]
*/
struct face{
	int idx[3];

	face(){}
	face(int i, int j, int k){
		idx[0] = i, idx[1] = j, idx[2] = k;
	}
```

```
	int& operator[](int u) { return idx[u]; }
};

vector<face> convex_hull( vector<vec> &cloud ){

	int n = (int)cloud.size();
	vector<int> L(n);

	vector<face> faces;

	for (int i = 0; i < n; ++i)
		for (int j = i + 1; j < n; ++j)
			for (int k = j + 1; k < n; ++k){

				vec a = cloud[i], b = cloud[j], c
				    = cloud[k];
				vec nr = (b - a) * (c - a);

				int pnt = 0;
				L[ pnt++ ] = j;
				L[ pnt++ ] = k;

				bool proc = true;

				int v = 0, V = 0;

				for (int l = 0; l < n && proc; ++l
				    ){
					if (l == i || l == j || l
					    == k) continue;

					double t = nr.dot( cloud[l
					    ] - a );
					if ( abs(t) < eps){
						if (l < k) proc =
						    false;
						else L[ pnt++ ] =
						    l;
					}
					else{
						if (t < 0) v = -1;
						else       V = +1;
					}
				}

				if (!proc || v * V == -1) continue
				    ;

				function<bool(int,int)> compare =
				    [&](int u, int v){
					return nr.dot( (cloud[u] -
					    a) * (cloud[v] - a) )
					    > 0;
				};

				sort(L, L + pnt, compare);

				for (int l = 0; l + 1 < pnt; ++l)
					faces.push_back( face(i, L
					    [l], L[l + 1]) );
			}

	return faces;
}

// Find mass center of a polyhedron given the external faces
// O(n)
void mass_center( vector<vec> &cloud, vector<face> &faces ){
	vec pivot = cloud[0];
	double x = 0, y = 0, z = 0, v = 0;

	for (auto f : faces){
		auto value =    ( cloud[f[0]] - pivot ).dot(
		                     ( cloud[f[1]] -
		                       pivot) * (
		                       cloud[f[2]] -
		                       pivot)
		                     );

		vec sum = cloud[f[0]] + cloud[f[1]] + cloud[f[2]]
		    + pivot;
		double cvol = abs( 1. * value / 6 );

		v += cvol;

		cvol /= 4;

		x += cvol * sum.X[0];
		y += cvol * sum.X[1];
		z += cvol * sum.X[2];
	}

	x /= v, y /= v, z /= v;

	// Mass center of a polyhedron at (x, y, z)
}
```

2.15 intersections

```
/*
	Line and segments predicates

	Tested: AIZU(judge.u-aizu.ac.jp) CGL
*/

bool intersectLL(const line &l, const line &m)
{
	return abs(cross(l.q - l.p, m.q - m.p)) > eps || // non-
	    parallel
	           abs(cross(l.q - l.p, m.p - l.p)) < eps;
	               // same line
}

bool intersectLS(const line &l, const segment &s)
{
	return cross(l.q - l.p, s.p - l.p) *       // s[0] is left
	    of l
	           cross(l.q - l.p, s.q - l.p) < eps; // s[1]
	               is right of l
}

bool intersectLP(const line &l, const point &p)
{
	return abs(cross(l.q - p, l.p - p)) < eps;
}

bool intersectSS(const segment &s, const segment &t)
{
	return ccw(s.p, s.q, t.p) * ccw(s.p, s.q, t.q) <= 0
	           && ccw(t.p, t.q, s.p) * ccw(t.p, t.q, s.q)
	               <= 0;
}

bool intersectSP(const segment &s, const point &p)
{
	return abs(s.p - p) + abs(s.q - p) - abs(s.q - s.p) < eps;
	// triangle inequality
	return min(real(s.p), real(s.q)) <= real(p)
```

```
                        && real(p) <= max(real(s.p), real(s.q))
                        && min(imag(s.p), imag(s.q)) <= imag(p)
                        && imag(p) <= max(imag(s.p), imag(s.q))
                        && cross(s.p - p, s.q - p) == 0;
}

point projection(const line &l, const point &p)
{
        double t = dot(p - l.p, l.p - l.q) / norm(l.p - l.q);
        return l.p + t * (l.p - l.q);
}

point reflection(const line &l, const point &p)
{
        return p + 2.0 * (projection(l, p) - p);
}

double distanceLP(const line &l, const point &p)
{
        return abs(p - projection(l, p));
}

double distanceLL(const line &l, const line &m)
{
        return intersectLL(l, m) ? 0 : distanceLP(l, m.p);
}

double distanceLS(const line &l, const line &s)
{
        if (intersectLS(l, s)) return 0;
        return min(distanceLP(l, s.p), distanceLP(l, s.q));
}

double distanceSP(const segment &s, const point &p)
{
        const point r = projection(s, p);
        if (intersectSP(s, r)) return abs(r - p);
        return min(abs(s.p - p), abs(s.q - p));
}

double distanceSS(const segment &s, const segment &t)
{
        if (intersectSS(s, t)) return 0;
        return min(min(distanceSP(s, t.p), distanceSP(s, t.q)),
                        min(distanceSP(t, s.p), distanceSP(t, s.q)
                            ));
}

point crosspoint(const line &l, const line &m)
{
        double A = cross(l.q - l.p, m.q - m.p);
        double B = cross(l.q - l.p, l.q - m.p);
        if (abs(A) < eps && abs(B) < eps)
                return m.p; // same line
        if (abs(A) < eps)
                assert(false); // !!!PRECONDITION NOT SATISFIED!!!
        return m.p + B / A * (m.q - m.p);
}
```

2.16 minimal circle

```
/*
    Find circle with minimal ratio that contain
    a cloud of points in 2d.

    Complexity O(n) (Expected running time)
    Tested: opencup 010376(Stars in a Can)

    Note: Something went wrong when random_shuffle was applied.
    If random_shuffle is removed complexity might increase
    to O(n^3)
*/
struct circle{
    point center;
    double r;

    bool contain(point &p){
        return abs(center - p) < r + eps;
    }
};

circle min_circle(vector<point> &cloud, int a, int b){
    point center = (cloud[a] + cloud[b]) / double(2.);
    double rat = abs(center - cloud[a]);
    circle C = {center, rat};

    for (int i = 0; i < b; ++i){
        point x = cloud[i];
        if (C.contain(x)) continue;
        center = three_point_circle( cloud[a], cloud[b], cloud[i]
            );
        rat = abs(center - cloud[a]);
        C = {center, rat};
    }
    return C;
}

circle min_circle(vector<point> &cloud, int a){
    point center = (cloud[a] + cloud[0]) / double(2.);
    double rat = abs(center - cloud[a]);
    circle C = {center, rat};

    for (int i = 0; i < a; ++i){
        point x = cloud[i];
        if (C.contain(x)) continue;
        C = min_circle(cloud, a, i);
    }

    return C;
}

circle min_circle(vector<point> cloud){
    // random_shuffle(cloud.begin(), cloud.end());

    int n = (int)cloud.size();

    for (int i = 1; i < n; ++i){
        int u = rand() % i;
        swap(cloud[u], cloud[i]);
    }

    point center = (cloud[0] + cloud[1]) / double(2.);
    double rat = abs(center - cloud[0]);
    circle C = {center, rat};
    for (int i = 2; i < n; ++i){
        point x = cloud[i];
        if (C.contain(x)) continue;
        C = min_circle(cloud, i);
    }
    return C;
}
```

2.17 minkowski Sum of Polygons

```
/*
```

```
    Minkowski sum of two convex polygons. O(n + m)

    Note: Polygons MUST be counterclockwise
*/

polygon minkowski(polygon &A, polygon &B){
        int na = (int)A.size(), nb = (int)B.size();

        if (A.empty() || B.empty()) return polygon();

        rotate(A.begin(), min_element(A.begin(), A.end()), A.end()
            );
        rotate(B.begin(), min_element(B.begin(), B.end()), B.end()
            );

        int pa = 0, pb = 0;

        polygon M;

        while (pa < na && pb < nb){
                M.push_back(A[pa] + B[pb]);
                double x = cross(A[(pa + 1) % na] - A[pa],
                                                  B[(pb + 1) % nb]
                                                      - B[pb]);

                if (x <= eps) pb++;
                if (-eps <= x) pa++;
        }

        while (pa < na) M.push_back(A[pa++] + B[0]);
        while (pb < nb) M.push_back(B[pb++] + A[0]);

        return M;
}
```

3 Data Structures

3.1 Mo's algorithm

```
#include<bits/stdc++.h>
#define TAM 30000 + 7
#define QTAM 200000 + 7
#define MTAM 1000000 + 7
#define whatis(x) cerr<<#x<<" is "<<x<<endl

using namespace std;

int a[TAM], r[QTAM], cnt[MTAM];
int ans, BLOCK, currL , currR ;

struct node{
    int L, R, idx;
}q[QTAM];

bool comp(node a, node b){
    if(a.L/BLOCK < b.L/BLOCK) return true;
    if(a.L/BLOCK > b.L/BLOCK) return false;
    return a.R < b.R;
}

void remove(int i){
    cnt[a[i]]--;
    if(cnt[a[i]] == 0)ans--;
}

void add(int i){
    cnt[a[i]]++;
    if(cnt[a[i]]==1)ans++;
}

int query(node i){
    while(currL< i.L){
        remove(currL);
        currL++;
    }
    while(currL > i.L){
        currL--;
        add(currL);
    }
    while(currR< i.R){
        currR++;
        add(currR);
    }
    while(currR > i.R){
        remove(currR);
        currR--;
    }
    return ans;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    #ifdef LOCAL
    freopen("in","r",stdin);
    #endif
    int n, que;
    cin>>n;
    BLOCK = sqrt(n);
    for(int i = 1; i <= n; i++)cin>>a[i];

    cin>>que;
    for(int i = 1; i <= que; i++){
        cin>>q[i].L>>q[i].R;
        q[i].idx = i;
    }
    sort(q +1, q + que+1, comp);

    for(int i = 1; i <=que; i++)
        r[q[i].idx] = query(q[i]);

    for(int i = 1; i <= que; i++)
        cout<<r[i]<<"\n";
}
```

3.2 Segment Trees with lazy propagation

```
//querys and build takes O(log n)
//example with segment sum
//tested in AIZU online Judge
#include<bits/stdc++.h>

using namespace std;

long long *p;

struct SegmentTree{
    SegmentTree *L, *R;
    long long sum = 0;
    long long lazy = 0;
    int l, r;

    void update(int a, int val){
        if(l == r){
            sum += val;
```

```cpp
            return;
        }
        int mid = (l + r)/2;
        if(l <= a && a<= mid)
            L->update(a, val);
        else
            R->update(a, val);
        sum = L->sum + R->sum;
    }

    void updateRange(int a, int b, long long val){
        if(lazy != 0){
            sum += (r-l+1)*lazy;
            if(l != r){
                R->lazy = lazy + R->lazy;
                L->lazy = lazy + L->lazy;
            }
            lazy = 0;
        }
        if(b < l or a > r)
            return;
        if(l >= a && r <= b){
            sum += (r-l+1)*val;
            if(l != r){
                R->lazy = val + R->lazy;
                L->lazy = val + L->lazy;
            }
            return;
        }
        L->updateRange(a, b, val);
        R->updateRange(a,b,val);
        sum = L->sum + R->sum;
    }

    long long query(int a, int b){
        if(b < l or a > r)
            return 0;
        if(lazy != 0){
            sum += (r-l+1)*lazy;
            if(l != r){
                R->lazy = lazy + R->lazy;
                L->lazy = lazy + L->lazy;
            }
            lazy = 0;
        }
        //this section can be used in non lazy segment tree
        if(a == l && b == r) return sum;
        if(b <= L->r) return L->query(a,b);
        if(a >= R->l) return R->query(a,b);
        return (L->query(a,L->r) + R->query(R->l, b));
    }

    SegmentTree(int a, int b): l(a), r(b){
        if(a == b){
            sum = p[a];
            L = R = nullptr;
        }
        else{
            L = new SegmentTree ( a, (a+b)/2 );
            R = new SegmentTree ( (a+b)/2 + 1, b );
            sum = L->sum + R->sum;
        }
    }
};


int main(){
    cin.tie(0);
    ios_base::sync_with_stdio(0);
    #ifdef LOCAL
        freopen("input.txt", "r", stdin);
    #endif // LOCAL
        long long T;
        cin >> T;
    while(T--){
        long long n, c;
        cin >> n >> c;
        long long l[n];
        memset(l,0,sizeof(l));
        p = l;
        SegmentTree *stree = new SegmentTree(0, n-1);
        while(c--){
            long long aux, p, q;
            cin >> aux >> p >> q;
            if(aux == 0){
                long long val;
                cin >> val;
                stree->updateRange(p-1, q-1, val);
            }
            else
                cout << stree->query(p-1, q-1) << endl;
        }
    }
}
```

3.3 Segment Trees with lazy propagation (Java)

```java
public class SegmentTreeRangeUpdate {
        public long[] leaf;
        public long[] update;
        public int origSize;
        public SegmentTreeRangeUpdate(int[] list)       {
                origSize = list.length;
                leaf = new long[4*list.length];
                update = new long[4*list.length];
                build(1,0,list.length-1,list);
        }
        public void build(int curr, int begin, int end, int[] list
            )     {
                if(begin == end)
                        leaf[curr] = list[begin];
                else    {
                        int mid = (begin+end)/2;
                        build(2 * curr, begin, mid, list);
                        build(2 * curr + 1, mid+1, end, list);
                        leaf[curr] = leaf[2*curr] + leaf[2*curr
                            +1];
                }
        }
        public void update(int begin, int end, int val) {
                update(1,0,origSize-1,begin,end,val);
        }
        public void update(int curr,  int tBegin, int tEnd, int
            begin, int end, int val)        {
                if(tBegin >= begin && tEnd <= end)
                        update[curr] += val;
                else    {
                        leaf[curr] += (Math.min(end,tEnd)-Math.max
                            (begin,tBegin)+1) * val;
                        int mid = (tBegin+tEnd)/2;
                        if(mid >= begin && tBegin <= end)
                                update(2*curr, tBegin, mid, begin,
```

```
                        end, val);
                if(tEnd >= begin && mid+1 <= end)
                        update(2*curr+1, mid+1, tEnd,
                            begin, end, val);
            }
        }
        public long query(int begin, int end)   {
                return query(1,0,origSize-1,begin,end);
        }
        public long query(int curr, int tBegin, int tEnd, int
            begin, int end)   {
                if(tBegin >= begin && tEnd <= end)      {
                        if(update[curr] != 0)   {
                                leaf[curr] += (tEnd-tBegin+1) *
                                    update[curr];
                                if(2*curr < update.length){
                                        update[2*curr] += update[
                                            curr];
                                        update[2*curr+1] += update
                                            [curr];
                                }
                                update[curr] = 0;
                        }
                        return leaf[curr];
                }
                else    {
                        leaf[curr] += (tEnd-tBegin+1) * update[
                            curr];
                        if(2*curr < update.length){
                                update[2*curr] += update[curr];
                                update[2*curr+1] += update[curr];
                        }
                        update[curr] = 0;
                        int mid = (tBegin+tEnd)/2;
                        long ret = 0;
                        if(mid >= begin && tBegin <= end)
                                ret += query(2*curr, tBegin, mid,
                                    begin, end);
                        if(tEnd >= begin && mid+1 <= end)
                                ret += query(2*curr+1, mid+1, tEnd
                                    , begin, end);
                        return ret;
                }
        }
}
```

3.4 Fenwick Tree

```
/* Fenwick tree or Binary Indexed Tree
 * query time O(logN)
 * update time O(logN)
 * LOGSZ must be higher than log(SIZE)
 * operations must have inverse

 * tested on AIZU online Judge
 */

#include<bits/stdc++.h>

using namespace std;

const int LOGSZ = 17;

int tree[(1<<LOGSZ) + 1];
int n = (1<<LOGSZ);

void add(int x, int v){
  while(x<=n){
    tree[x] += v;
    x+= (x & -x);
  }
}

int get(int x){
  int ans = 0;
  while(x){
    ans += tree[x];
    x-= (x & -x);
  }
  return ans;
}


int rsq(int x, int y){
    return get(y) - get(x - 1);
}
```

3.5 Union Find

```
/*
 * Disjoint set data structure, merge and find takes O(logN)
 * tested on AIZU online Judge
 */

#include <iostream>
#include <vector>
using namespace std;
int find(vector<int> &C, int x) { return (C[x] == x) ? x : C[x] =
    find(C, C[x]); }
bool same(vector<int> &C, int x, int y){ return find(C, x) == find
    (C, y);}
void merge(vector<int> &C, int x, int y) { C[find(C, x)] = find(C,
     y); }
int main()
{
        int n = 5;
        vector<int> C(n);
        for (int i = 0; i < n; i++) C[i] = i;
        merge(C, 0, 2);
        merge(C, 1, 0);
        merge(C, 3, 4);
        for (int i = 0; i < n; i++) cout << i << " " << find(C, i)
            << endl;
        return 0;
}
```

3.6 Persistent Treaps

```
/*
 * implicit persistent treaps, solved Genetics problem
 */

#include <bits/stdc++.h>

using namespace std;

typedef long long int64;
typedef pair<int,int> pii;
typedef vector<int> vi;

const int oo = 0x3f3f3f3f;
const double eps = 1e-9;
```

```cpp
struct node{
    int64 count[4], size;
    int idx;
    node *l, *r;
};

string dna = "AGTC";
map<char,int> dnaid;

int num(char c){
    return dnaid[ c ];
}

node* clone(node *u){
    node *v = new node();
    v->size = u->size, v->idx = u->idx;
    v->l = u->l, v->r = u->r;
    for (int i = 0; i < 4; ++i)
        v->count[i] = u->count[i];
    return v;
}

int64 size(node *u){
    return u ? u->size : 0LL;
}

int get(node *u, int p){
    return u ? u->count[p] : 0;
}

node *update(node *u){
    u->size = size(u->l) + size(u->r) + 1;

    for (int i = 0; i < 4; ++i)
        u->count[ i ] = get(u->l, i) + get(u->r, i);
    u->count[ u->idx ]++;
    return u;
}

node* merge( node *u, node *v ){
    if (!u || !v) return u ? u : v;

    int l = size(u), r = size(v);

    if (rand() % (l + r) < l){
        u = clone( u );
        u->r = merge( u->r, v );
        return update( u );
    }
    else{
        v = clone( v );
        v->l = merge(u, v->l);
        return update( v );
    }
}

pair<node*,node*> split(node *u, int k){
    if (!u) return make_pair(nullptr, nullptr);

    u = clone(u);

    if (k <= size( u->l )){
        auto cur = split( u->l, k );
        u->l = cur.second;
        return {cur.first, update(u) };
    }
    else{
        auto cur = split( u->r, k - size(u->l) - 1);
        u->r = cur.first;
        return {update(u), cur.second};
    }
}

node* build(int b, int e, string &s){
    if (b > e) return nullptr;

    int m = (b + e) >> 1;
    node *u = new node();

    u->idx = num( s[m] );
    u->l = build(b, m - 1, s);
    u->r = build(m + 1, e, s);

    return update(u);

}

node *mutate(node *u, int k, int v){
    u = clone(u);

    if (k <= size(u->l))
        u->l = mutate(u->l, k, v);
    else if (size(u->l) + 1 == k)
        u->idx = v;
    else
        u->r = mutate(u->r, k - size(u->l) - 1, v);

    return update(u);
}

void dfs( node* u){
    if (!u) return;

    dfs(u->l);

    cout << dna[ u->idx ] << endl;
    for (int i = 0; i < 4; ++i)
        cout << u->count[i] << " ";
    cout << endl;

    dfs(u->r);
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    for (int i = 0; i < 4; ++i)
        dnaid[ dna[i] ] = i;

    int n; cin >> n;

    vector<node*> version(1);

    for (int i = 1; i <= n; ++i){
        string s; cin >> s;
        version.push_back( build(0, s.length() - 1, s ) );
    }

    int q; cin >> q;

    while (q--){
        string comm;
        cin >> comm;

        if (comm == "MUTATE"){
            int id, k; char m;
            cin >> id >> k >> m;

            version[id] = mutate( version[id], k, num(m) );
        }
        else if (comm == "CROSS"){
```

```
            int id1, id2, k1, k2;
            cin >> id1 >> id2 >> k1 >> k2;

            auto dna1 = split( version[id1], k1 );
            auto dna2 = split( version[id2], k2 );

            node* dna3 = merge( dna1.first, dna2.second );
            node* dna4 = merge( dna2.first, dna1.second );

            version.push_back( dna3 );
            version.push_back( dna4 );
        }
        else{
            int id, k1, k2;
            cin >> id >> k1 >> k2;

            node *u = split(version[id], k2).first;
                  u = split(u, k1 - 1).second;

            for (int i = 0; i < 4; ++i)
                cout << u->count[i] << " \n"[i + 1 == 4];
        }
    }

    return 0;
}
```

4 Strings

4.1 Prefix Function

```
vector<int> prefix_function (string s) {
        int n = (int) s.length();
        vector<int> pi (n);
        for (int i=1; i<n; ++i) {
                int j = pi[i-1];
                while (j > 0 && s[i] != s[j])
                        j = pi[j-1];
                if (s[i] == s[j])  ++j;
                pi[i] = j;
        }
        return pi;
}
```

4.2 KMP Automata

```
string s; //
const int alphabet = 256; //
                      '

s += '#';
int n = (int) s.length();
vector<int> pi = prefix_function (s);
vector < vector<int> > aut (n, vector<int> (alphabet));
for (int i=0; i<n; ++i)
        for (char c=0; c<alphabet; ++c)
                if (i > 0 && c != s[i])
                        aut[i][c] = aut[pi[i-1]][c];
                else
                        aut[i][c] = i + (c == s[i]);
```

4.3 Suffix Array

```
#include <bits/stdc++.h>
using namespace std;

vector<int> suffix_array(const string &s) {
    int n = s.size();
    vector<int> sa(n), rank(n);
    vector<long long> rank2(n);

    for (int i = 0; i < n; i++) {
        sa[i] = i;
        rank[i] = s[i];
    }
    for (int len = 1; len < n; len *= 2) {
        for (int i = 0; i < n; i++) rank2[i] = ( (long long) rank[
            i] << 32) + (i+len < n ? rank[i+len] : -1);
        sort(sa.begin(), sa.end(), [&](int i, int j){
            return rank2[i] < rank2[j];
        });
        for (int i = 0; i < n; i++) {
            if (i > 0 && rank2[sa[i]] == rank2[sa[i-1]]) rank[sa[i
                ]] = rank[sa[i-1]];
            else rank[sa[i]] = i;
        }
    }
    return sa;
}

vector<int> lcp_array(const vector<int> &sa, const string &s) {
    int n = s.size();
    vector<int> rank(n);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;

    vector<int> ans(n);
    for (int i = 0, l = 0; i < n; i++) if (rank[i] > 0) {
        int j = sa[rank[i]-1];
        while (s[i+l] == s[j+l]) l++;
        ans[rank[i]] = l > 0 ? l-- : l;
    }
    return ans;
}

int main() {
    string s = "banana";
    s += char(0);

    auto sa = suffix_array(s);
    for (auto x : sa) cerr << x << ' '; cerr << endl;

    auto lcp = lcp_array(sa,s);
    for (auto x : lcp) cerr << x << ' '; cerr << endl;

    return 0;
}
```

4.4 Trie

```
// K = tama o del alfabeto
// NMAX = m ximo n mero de nodos que tendr  el trie

struct vertex {
        int next[K];
        bool leaf;
};
```

```
int sz;
vertex t[NMAX+1];

void add_string (const string & s) {
        int v = 0;
        for (size_t i=0; i<s.length(); ++i) {
                char c = s[i]-'a'; //

                if (t[v].next[c] == -1) {
                        memset (t[sz].next, 255, sizeof t[sz].next
                            );
                        t[v].next[c] = sz++;
                }
                v = t[v].next[c];
        }
        t[v].leaf = true;
}
```

4.5 Hash

```
#include <bits/stdc++.h>
using namespace std;

#define forn(i,n) for(int i=0;i<(int)(n);i++)
#define si(c) ((int)(c).size())
#define forsn(i,s,n) for(int i = (int)(s); i<((int)n); i++)
#define dforsn(i,s,n) for(int i = (int)(n)-1; i>=((int)s); i--)
#define all(c) (c).begin(), (c).end()
#define D(a) cerr << #a << "=" << a << endl;
#define pb push_back
#define eb emplace_back
#define mp make_pair

typedef long long int ll;
typedef vector<int> vi;
typedef pair<int,int> pii;

mt19937 rng;
#define hash __nico_hash
struct hashing {
    int mod, mul;

    bool prime(int n) {
        for (int d = 2; d*d <= n; d++) if (n%d == 0) return false;
        return true;
    }

    void setValues(int mod, int mul) {
        this->mod = mod;
        this->mul = mul;
    }

    void randomize() {
        rng.seed(time(0));
        mod = uniform_int_distribution<>(0, (int) 5e8)(rng) + 1e9;
        while (!prime(mod)) mod++;
        mul = uniform_int_distribution<>(2,mod-2)(rng);
    }

    vi h, pot;
    void process(const string &s) {
        h.resize(si(s)+1);
        pot.resize(si(s)+1);
        h[0] = 0; forn(i,si(s)) h[i+1] = (((ll)h[i] * mul) + s[i])
             % mod;
        pot[0] = 1; forn(i,si(s)) pot[i+1] = (ll) pot[i] * mul %
            mod;
    }

    int hash(int i, int j) {
        int res = h[j] - (ll) h[i] * pot[j-i] % mod;
        if (res < 0) res += mod;
        return res;
    }

    int hash(const string &s) {
        int res = 0;
        for (char c : s) res = (res * (ll) mul + c) % mod;
        return res;
    }
};

hashing h1,h2;

int main() { }
```

5 Flows

5.1 Ford Fulkerson

```
#include<bits/stdc++.h>

using namespace std;
///---------------------- Ford-Fulkerson O(MaxFlow * |E|)
    ----------------------

//tested on AIZU online Judge

struct OutEdge {
        int to, cap, rIdx;
        OutEdge ( ) { }
        OutEdge(int to, int cap, int rIdx) :
                to(to), cap(cap), rIdx(rIdx) { }
};

struct Network{
        vector<vector<OutEdge> > out;
        vector<bool> seen;

        int sink;
        int augment ( int i, const int cur ) {
                if ( i == sink ) return cur;
                if ( seen[i] ) return false;
                seen[i] = true;
                int ans;
                for ( OutEdge& e : out[i] )
                        if ( e.cap > 0 && ( ans = augment(e.to,min
                            (cur,e.cap)) ) ) {
                                e.cap -= ans;
                                out[e.to][e.rIdx].cap += ans;
                                return ans;
                        }
                return 0;
        }

        int maxflow ( int source, int _sink ) {
                sink = _sink;
                int curflow = 0, aug;
                while ( true ) {
                        fill ( seen.begin(), seen.end(), false );
                        aug = augment(source,INT_MAX);
```

```
                        if ( aug == 0 ) break;
                        curflow += aug;
                }
                return curflow;
        }

        void addEdge ( int fr, int to, int c ) {
                assert ( fr != to );
                out[fr].push_back(OutEdge(to, c, out[to].size()));
                out[to].push_back(OutEdge(fr, 0, out[fr].size() -
                    1));
        }

        Network(int n) {
                out.assign(n, vector<OutEdge>());
                seen.resize(n);
        }
};
```

5.2 Edmons Karp Min cut

```
///------------------- O(|V|*|E|^2) -------------------///
struct edge {
        int v, cap, inv;
        edge() {}
        edge(int v, int cap, int inv) : v(v), cap(cap), inv(inv)
            {}
};
struct edmons_karp {
        vector< vector<edge> > g;
        vector<int> from;
        vector<bool> color;
        int n;
        edmons_karp(int n) : n(n), g(n), color(n), from(n) {}
        // Call flood (source) to color one node
        // component of min cut.
        void flood(int u) {
                if (color[u]) return;
                color[u] = true;
                for(int i = 0; i < g[u].size(); i++) {
            edge &e = g[u][i];
                        if (e.cap > 0)
                flood(e.v);
                }
        }
        int max_flow(int s, int t) {
                int res = 0;
                while(1) {
                        fill(from.begin(), from.end(), -1);
                        queue<int> q;
                        q.push(s);
                        from[s] = -2;
                        for(int u; q.size(); q.pop()) {
                                u = q.front();
                                for(int i = 0; i < g[u].size(); i
                                    ++) {
                    edge &e = g[u][i];
                                        if(from[e.v] == -1 && e.
                                            cap) {
                                                from[e.v] = e.inv;
                                                q.push(e.v);
                                        }
                                }
                        }
                        if (from[t] == -1) break;
                        int aug = INT_MAX;
                        for (int i = t, j; i != s; i = j) {
                                j = g[i][ from[i] ].v;
                                aug = min(aug, g[j][ g[i][ from[i]
                                    ].inv ].cap);
                        }
                        for (int i = t, j; i != s; i = j) {
                                j = g[i][ from[i] ].v;
                                g[j][ g[i][from[i] ].inv ].cap -=
                                    aug;
                                g[i][ from[i] ].cap += aug;
                        }
                        res += aug;
                }
                return res;
        }
        void add_non_dir_edge(int a, int b, int c) {
                g[a].push_back(edge(b, c, g[b].size()));
                g[b].push_back(edge(a, c, g[a].size() - 1));
        }
        void add_edge(int a, int b, int c) {
                g[a].push_back(edge(b, c, g[b].size()));
                g[b].push_back(edge(a, 0, g[a].size() - 1));
        }
};
```

5.3 Dinic's Blocking Flow

```
// Adjacency list implementation of Dinic's blocking flow
    algorithm.
// This is very fast in practice, and only loses to push-relabel
    flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

//tested on AIZU online Judge

#include <bits/stdc++.h>

using namespace std;

const int INF = 2000000000;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge *> dad;
  vector<int> Q;

  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}
```

```
  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;

    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
        Edge &e = G[x][i];
        if (!dad[e.to] && e.cap - e.flow > 0) {
          dad[e.to] = &G[x][i];
          Q[tail++] = e.to;
        }
      }
    }
    if (!dad[t]) return 0;

    long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++) {
      Edge *start = &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from])
          {
        if (!e) { amt = 0; break; }
        amt = min(amt, e->cap - e->flow);
      }
      if (amt == 0) continue;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from])
          {
        e->flow += amt;
        G[e->to][e->index].flow -= amt;
      }
      totflow += amt;
    }
    return totflow;
  }

  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
```

5.4 Push Relabel

```
// Adjacency list implementation of FIFO push relabel maximum flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It solves
// random problems with 10000 vertices and 1000000 edges in a few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

//tested on AIZU online Judge
#include<bits/stdc++.h>

using namespace std;


typedef long long LL;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N),
      count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v)
        ; }
  }

  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }

  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }

  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
```

```
        dist[v] = min(dist[v], dist[G[v][i].to] + 1);
      count[dist[v]]++;
      Enqueue(v);
    }

    void Discharge(int v) {
      for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[
          v][i]);
      if (excess[v] > 0) {
        if (count[dist[v]] == 1)
          Gap(dist[v]);
        else
          Relabel(v);
      }
    }

    LL GetMaxFlow(int s, int t) {
      count[0] = N-1;
      count[N] = 1;
      dist[s] = N;
      active[s] = active[t] = true;
      for (int i = 0; i < G[s].size(); i++) {
        excess[s] += G[s][i].cap;
        Push(G[s][i]);
      }

      while (!Q.empty()) {
        int v = Q.front();
        Q.pop();
        active[v] = false;
        Discharge(v);
      }

      LL totflow = 0;
      for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
      return totflow;
    }
  };

  int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    #ifdef Larra
    freopen("in","r",stdin);
    freopen("out","w",stdout);
    #endif
    int n, m;
    cin>>n>>m;
    PushRelabel netw(n);
    for(int i = 0; i < m; i++){
      int a, b, c;
      cin>>a>>b>>c;
      netw.AddEdge(a, b, c);
    }
    cout<<netw.GetMaxFlow(0, n-1)<<"\n";
  }
```

5.5 Hopcroft karp's maximum bipartite matching

```
#include<bits/stdc++.h>

using namespace std;

///------------------- O(|E|*sqrt(|V|)) -------------------///
//tested in AIZU online Judge
struct hopcroft_karp {
    int l, r;
    vector<int> last, prev, head, matching, d;
    vector<bool> used, vis;
    hopcroft_karp(int l, int r) : l(l), r(r), last(l, -1),
        matching (r, -1), d(l), used(l), vis(l) {}
    /// u -> 0 to l, v -> 0 to r
    void add_edge(int u, int v) {
        head.push_back(v);
        prev.push_back(last[u]);
        last[u] = prev.size()-1;
    }
    void bfs() {
        fill(d.begin(), d.end(), -1);
        queue<int> q;
        for(int u = 0; u < l; u++) {
            if(!used[u]) {
                q.push(u);
                d[u] = 0;
            }
        }
        while(q.size()) {
            int u = q.front(); q.pop();
            for(int e = last[u]; e >= 0; e = prev[e]) {
                int v = matching[ head[e] ];
                if(v >= 0 && d[v] < 0) {
                    d[v] = d[u]+1;
                    q.push(v);
                }
            }
        }
    }
    bool dfs(int u) {
        vis[u] = true;
        for(int e = last[u]; e >= 0; e = prev[e]) {
            int v = head[e];
            int k = matching[v];
            if(k < 0 || ( !vis[k] && d[k] == d[u]+1 && dfs(k) )) {
                matching[v] = u;
                used[u] = true;
                return true;
            }
        }
        return false;
    }
    int max_matching() {
        int ans = 0;
        while(true) {
            bfs();
            fill(vis.begin(), vis.end(), false);
            int f = 0;
            for(int u = 0; u < l; u++)
                if(!used[u] && dfs(u)) f++;
            if(f == 0) return ans;
            ans += f;
        }
        return 0;
    }
};


int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  #ifdef Larra
  freopen("in","r",stdin);
```

```
  freopen("out","w",stdout);
  #endif
  int n1, n2, m;
  cin>>n1>>n2>>m;
  hopcroft_karp hp(n1, n2);
  for(int i = 0; i < m; i++){
      int u,v;
      cin>>u>>v;
      hp.add_edge(u, v);
  }
  cout<<hp.max_matching()<<"\n";
}
```

5.6 Hungarian's maximum bipartite matching

```
#include<bits/stdc++.h>

using namespace std;

///------------------- O(|V|*|E|) -------------------///
//tested in AIZU online Judge
struct hungarian_bipartite {
    int l, r;
    vector< vector<int> > g;
    vector<bool> seen;
    vector<int> match; /// match[i] is the left matching of right
        node i
    hungarian_bipartite(int l, int r) : l(l), r(r), seen(l), match
        (l+r, -1), g(l+r) {}
    /// [0-l) left, [l, r) right
    /// a and b are 0-indexed
    void add_edge(int a, int b) {
        g[a].push_back(l+b);
        g[l+b].push_back(a);
    }
    bool go(int u) {
        if(seen[u]) return false;
        seen[u] = true;
        for(int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            if(match[v] == -1 || go(match[v])) {
                match[v] = u;
                return true;
            }
        }
        return false;
    }
    int max_matching() {
        int ans = 0;
        for(int i = 0; i < l; i++) {
            fill(seen.begin(), seen.end(), false);
            ans += go(i);
        }
        return ans;
    }
};

int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  #ifdef Larra
  freopen("in","r",stdin);
  freopen("out","w",stdout);
  #endif
  int n1, n2, m;
  cin>>n1>>n2>>m;
  hungarian_bipartite hp(n1, n2);
  for(int i = 0; i < m; i++){
      int u,v;
      cin>>u>>v;
      hp.add_edge(u, v);
  }
  cout<<hp.max_matching()<<"\n";
}
```

6 Math

6.1 general math tricks

```
 long square(long n){ return n*n;}

 int fastPow(long x, long n){
     if(n == 0)
        return 1;

     if(n % 2 == 0)
        return square(fastPow(x, n/2));

     return x * (fastPow(x, n - 1));
 }

/* LCM */
int LCM(int m, n){return (m/__gcd(m, n)) * n; }


int main(){

    /* n es impar?*/
    odd = ((n & 1)? true : false);

    /*como saber si un numero es una potencia de 2*/
    power_of_2 = ((v & (v-1)) == 0);

    /*contar trailing 0's de una mascara */
    __builtin_ctz(n);

    /*contar 1's de una mascara*/
    __builtin_popcount(n);

    /*quitar el elemento j de la m scara*/

    mask &= ~(1<<j);

    /*revisar si el elemento j del arreglo esta en la m scara (
        si es 0 el resultado es porque no est )*/
    int t = mask & (1<<j);

    /*Obtener el bit menos significativo*/
     t = mask & -mask

    /*encender todos los n primeros bits de la m scara*/

    mask = (1<<n) - 1;

    /*iterar sobre cada uno de los subsets de un subset y*/
    for(int x = y; x>0; x = (y & (x-1)) )
}
```

6.2 Miller Rabin's primality test

```
// Randomized Primality Test (Miller-Rabin):
```

```
//   Error rate: 2^(-TRIAL)
//   Almost constant time. srand is needed

#include <stdlib.h>
#define EPS 1e-7

typedef long long LL;

LL ModularMultiplication(LL a, LL b, LL m)
{
        LL ret=0, c=a;
        while(b)
        {
                if(b&1) ret=(ret+c)%m;
                b>>=1; c=(c+c)%m;
        }
        return ret;
}
LL ModularExponentiation(LL a, LL n, LL m)
{
        LL ret=1, c=a;
        while(n)
        {
                if(n&1) ret=ModularMultiplication(ret, c, m);
                n>>=1; c=ModularMultiplication(c, c, m);
        }
        return ret;
}
bool Witness(LL a, LL n)
{
        LL u=n-1;
  int t=0;
        while(!(u&1)){u>>=1; t++;}
        LL x0=ModularExponentiation(a, u, n), x1;
        for(int i=1;i<=t;i++)
        {
                x1=ModularMultiplication(x0, x0, n);
                if(x1==1 && x0!=1 && x0!=n-1) return true;
                x0=x1;
        }
        if(x0!=1) return true;
        return false;
}
LL Random(LL n)
{
  LL ret=rand(); ret*=32768;
        ret+=rand(); ret*=32768;
        ret+=rand(); ret*=32768;
        ret+=rand();
  return ret%n;
}
bool IsPrimeFast(LL n, int TRIAL)
{
  while(TRIAL--)
  {
    LL a=Random(n-2)+1;
    if(Witness(a, n)) return false;
  }
  return true;
}
```

6.3 Pollard rho

```
#include<bits/stdc++.h>
#include<time.h>

#define show(x) cout << #x << " = " << x << endl;

using namespace std;

typedef long long ll;
typedef pair<ll, ll> ii;
typedef pair<double, ii> iii;

const int MAX = 200005;
const double EPS = 1e-5;
const int INF = INT_MAX;

//modular multiplication for really big numbers
ll mul(ll a, ll b, ll mod) {
    ll ret = 0;
    for(a %= mod, b %= mod; b != 0;
        b >>= 1, a <<= 1, a = a >= mod ? a - mod : a) {
        if (b&1) {
            ret += a;
            if (ret >= mod) ret -= mod;
        }
    }
    return ret;
}

ll fpow(ll a, ll b, ll MOD) {
    ll ans = 1LL;
    while(b > 0) {
        if(b&1) ans = mul(ans, a, MOD);
        a = mul(a, a, MOD);
        b >>= 1LL;
    }
    return ans;
}

const int rounds = 6;
// Checks if a number is prime with prob 1 - 1 / (2 ^ it)
bool miller_rabin(ll n) {
    if(n == 2 || n == 3) return true;
    if(n < 2 || (n&1) == 0) return false;
    for(int i = 0; i < rounds; i++) {
        int a = rand()%(n-4)+2;
        if(fpow(a, n-1, n) != 1)
            return false;
    }
    return true;
}

// if n is prime , check with miller rabin (n^(1/4)) and check
    return != n and  != 1
ll pollard_rho(ll n, ll c) {
    ll x = 2, y = 2, i = 1, k = 2, d;
    while (true) {
        x = (mul(x, x, n) + c);
        if (x >= n)     x -= n;
        d = __gcd(x - y, n);
        if (d > 1) return d;
        if (++i == k) y = x, k <<= 1;
    }
    return n;
}

//return factorization of a big number

void factorize(ll n, vector<ll> &f) {
    if(n == 1) return;
    if (miller_rabin(n)) {
        f.push_back(n);
        return;
    }
```

```
    ll d = n;
    for (int i = 2; d == n; i++)
        d = pollard_rho(n, i);
    factorize(d, f);
    factorize(n/d, f);
}
```

6.4 number theory general

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int, int> PII;

// return a % b (positive value)
int mod(int a, int b) {
        return ((a%b) + b) % b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
        while (b) { int t = a%b; a = b; b = t; }
        return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
        return a / gcd(a, b)*b;
}

// (a^b) mod m via successive squaring
int powermod(int a, int b, int m)
{
        int ret = 1;
        while (b)
        {
                if (b & 1) ret = mod(ret*a, m);
                a = mod(a*a, m);
                b >>= 1;
        }
        return ret;
}

// returns g = gcd(a, b); finds x, y such that d = ax + by
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;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
        int x, y;
        VI ret;
        int g = extended_euclid(a, n, x, y);
        if (!(b%g)) {
                x = mod(x*(b / g), n);
                for (int i = 0; i < g; i++)
                        ret.push_back(mod(x + i*(n / g), n));
        }
        return ret;
}

// 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 mod(x, n);
}

// 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.
PII 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 make_pair(0, -1);
        return make_pair(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.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
        PII 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;
}

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y) {
        if (!a && !b)
        {
                if (c) return false;
                x = 0; y = 0;
                return true;
        }
        if (!a)
        {
                if (c % b) return false;
                x = 0; y = c / b;
                return true;
        }
        if (!b)
        {
                if (c % a) return false;
                x = c / a; y = 0;
                return true;
        }
        int g = gcd(a, b);
        if (c % g) return false;
```

```
        x = c / g * mod_inverse(a / g, b / g);
        y = (c - a*x) / b;
        return true;
}

int main() {
        // expected: 2
        cout << gcd(14, 30) << endl;

        // expected: 2 -2 1
        int x, y;
        int g = extended_euclid(14, 30, x, y);
        cout << g << " " << x << " " << y << endl;

        // expected: 95 451
        VI sols = modular_linear_equation_solver(14, 30, 100);
        for (int i = 0; i < sols.size(); i++) cout << sols[i] << "
            ";
        cout << endl;

        // expected: 8
        cout << mod_inverse(8, 9) << endl;

        // expected: 23 105
        //           11 12
        PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }), VI({
            2, 3, 2 }));
        cout << ret.first << " " << ret.second << endl;
        ret = chinese_remainder_theorem(VI({ 4, 6 }), VI({ 3, 5 })
            );
        cout << ret.first << " " << ret.second << endl;

        // expected: 5 -15
        if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" <<
            endl;
        cout << x << " " << y << endl;
        return 0;
}
```

7 Others

7.1 Fast Fourier Transform (convolution)

```
// Convolution using the fast Fourier transform (FFT).
//
// INPUT:
//     a[1...n]
//     b[1...m]
//
// OUTPUT:
//     c[1...n+m-1] such that c[k] = sum_{i=0}^k a[i] b[k-i]
//
// Alternatively, you can use the DFT() routine directly, which
    will
// zero-pad your input to the next largest power of 2 and compute
    the
// DFT or inverse DFT.

#include <iostream>
#include <vector>
#include <complex>

using namespace std;

typedef long double DOUBLE;
typedef complex<DOUBLE> COMPLEX;
typedef vector<DOUBLE> VD;
typedef vector<COMPLEX> VC;

struct FFT {
  VC A;
  int n, L;

  int ReverseBits(int k) {
    int ret = 0;
    for (int i = 0; i < L; i++) {
      ret = (ret << 1) | (k & 1);
      k >>= 1;
    }
    return ret;
  }

  void BitReverseCopy(VC a) {
    for (n = 1, L = 0; n < a.size(); n <<= 1, L++) ;
    A.resize(n);
    for (int k = 0; k < n; k++)
      A[ReverseBits(k)] = a[k];
  }

  VC DFT(VC a, bool inverse) {
    BitReverseCopy(a);
    for (int s = 1; s <= L; s++) {
      int m = 1 << s;
      COMPLEX wm = exp(COMPLEX(0, 2.0 * M_PI / m));
      if (inverse) wm = COMPLEX(1, 0) / wm;
      for (int k = 0; k < n; k += m) {
        COMPLEX w = 1;
        for (int j = 0; j < m/2; j++) {
          COMPLEX t = w * A[k + j + m/2];
          COMPLEX u = A[k + j];
          A[k + j] = u + t;
          A[k + j + m/2] = u - t;
          w = w * wm;
        }
      }
    }
    if (inverse) for (int i = 0; i < n; i++) A[i] /= n;
    return A;
  }

  // c[k] = sum_{i=0}^k a[i] b[k-i]
  VD Convolution(VD a, VD b) {
    int L = 1;
    while ((1 << L) < a.size()) L++;
    while ((1 << L) < b.size()) L++;
    int n = 1 << (L+1);

    VC aa, bb;
    for (size_t i = 0; i < n; i++) aa.push_back(i < a.size() ?
        COMPLEX(a[i], 0) : 0);
    for (size_t i = 0; i < n; i++) bb.push_back(i < b.size() ?
        COMPLEX(b[i], 0) : 0);

    VC AA = DFT(aa, false);
    VC BB = DFT(bb, false);
    VC CC;
    for (size_t i = 0; i < AA.size(); i++) CC.push_back(AA[i] * BB
        [i]);
    VC cc = DFT(CC, true);

    VD c;
    for (int i = 0; i < a.size() + b.size() - 1; i++) c.push_back(
        cc[i].real());
    return c;
  }
```

```
};

int main() {
  double a[] = {1, 3, 4, 5, 7};
  double b[] = {2, 4, 6};

  FFT fft;
  VD c = fft.Convolution(VD(a, a + 5), VD(b, b + 3));

  // expected output: 2 10 26 44 58 58 42
  for (int i = 0; i < c.size(); i++) cerr << c[i] << " ";
  cerr << endl;

  return 0;
}
```

7.2 c++ ios tricks

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    // Ouput a specific number of digits past the decimal point,
    // in this case 5
    cout.setf(ios::fixed); cout << setprecision(5);
    cout << 100.0/7.0 << endl;
    cout.unsetf(ios::fixed);

    // Output the decimal point and trailing zeros
    cout.setf(ios::showpoint);
    cout << 100.0 << endl;
    cout.unsetf(ios::showpoint);

    // Output a '+' before positive values
    cout.setf(ios::showpos);
    cout << 100 << " " << -100 << endl;
    cout.unsetf(ios::showpos);

    // Output numerical values in hexadecimal
    cout << hex << 100 << " " << 1000 << " " << 10000 << dec <<
        endl;
}
```

7.3 java IO template and iterative binary search

```
import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.InputStream;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        TaskC solver = new TaskC();
        solver.solve(1, in, out);
        out.close();
    }

    static class TaskC {
        private static final int ITERATIONS = 500;
        public void solve(int testNumber, InputReader in,
            PrintWriter out) {
            int n = in.nextInt();
            //Iterative binary search
            double l = 0.0, h = 1e17;
            for (int i = 0; i < ITERATIONS; i++) {
                double mid = (l + h) / 2.0;
                if (can(mid, a, b, p))
                    l = mid;
                else
                    h = mid;
            }
            double ans = l;
        }
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(
                stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()
                ) {
                try {
                    tokenizer = new StringTokenizer(reader.
                        readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }

        public double nextDouble() {
            return Double.parseDouble(next());
        }

    }
}
```