Team 37

Radboud University Nijmegen

Contents

1 Basic

Power (Fast Exponentiation)

```
int pwr(int a, int b){
        int result = 1;
    while (b){
        if (b % 2) result *= a;
        b /= 2;
        a *= a;
    }

    return result;
}
```

Binomial

```
const int64_t M = 1000000007;

vector<vector<int64_t> > DP(4001, vector<int64_t>(4001, -1));

int64_t binom(int64_t n, int64_t k) {
    if(k == 0 || k == n) return 1;
    if(DP[n][k] != -1) return DP[n][k];
    DP[n][k] = binom(n - 1, k - 1) + binom(n - 1, k);
    DP[n][k] %= M;
    return DP[n][k];
}
```

Prime Sieve

```
vector<int> prime_sieve(int n){

        if (n < 2) return vector<int>();

        vector<int> primes;
        vector<bool> l (n+1,true);
        primes.push_back(2);

    int i = 3;
        for(; i <= sqrt(n); i += 2){
                while(!l[i]) i += 2;

                primes.push_back(i);
                for(int j=i*i; j < n; j+=i) l[j] = false;
        }

        for(; i < n; i+=2){
        if(l[i]) primes.push_back(i);
        }

        return primes;
}
```

Extended Euclidean

```
//Input two numbers a and b
//Return gcd(a,b)
int64_t gcd(int64_t a, int64_t b){
    if(a < b) swap(a,b);
    while(b != 0){
        int64_t r = a % b;
        a = b;
        b = r;
    }
    return a;
}

//Input two numbers a and b;
//Return triple (x,y,c) satisfying:
//x * a + y * b = c, with c = gcd(a,b)
pair<pair<int64_t,int64_t>, int64_t> egcd(int64_t a, int64_t b){
    int64_t p_prev = 0, p_cur = 1;
    int64_t q_prev = 1, q_cur = 0;
    int m = 0;
    if(a < b) {
        m++;
        swap(a,b);
        swap(q_prev,p_prev);
        swap(q_cur,p_cur);
    }
    while(b != 0){
        m++;
        int64_t r = a % b;
        int64_t k = a / b;
        int64_t s_temp = k * q_cur + q_prev;
        q_prev = q_cur, q_cur = s_temp;
        int64_t t_temp = k * p_cur + p_prev;
        p_prev = p_cur, p_cur = t_temp;
        a = b;
        b = r;
    }
    if(m % 2 == 0) m = 1;
    else m = -1;
    return make_pair(make_pair(m*q_prev,-m*p_prev),a);
}

//solved: https://open.kattis.com/problems/modulararithmetic
//https://open.kattis.com/problems/wipeyourwhiteboards
```

Extended Euclidean2

```
int64_t gcd(int64_t a, int64_t b) { while(b) {a %= b; swap(a,b); } return a
    ; }
int64_t lcm(int64_t a, int64_t b) { return (a / gcd(a, b)) * b; }
int64_t mod(int64_t a, int64_t b) { return ((a % b) + b) % b; }

//find x, y, s.t. ax + by = d = gcd(a,b)
void extended_euclid(int64_t a, int64_t b, int64_t& x, int64_t& y, int64_t&
     d){
    int64_t xx = y = 0;
    int64_t yy = x = 1;
    while(b) {
```

```
        int64_t q = a / b;
        int64_t t = b; b = a % b; a = t;
        t = xx; xx = x - q * xx; x = t;
        t = yy; yy = y - q * yy; y = t;
    }
    d = a;
    return;
}

//Credits to RagnarGrootKoerkamp
```

CRT

```
// Chinese remainder theorem: finds z s.t. z % xi = ai. z is
// unique modulo M = lcm(xi). Returns (z, M), m = -1 on failure.

int64_t mod(int64_t a, int64_t b) { return ((a % b) + b) % b; }

typedef pair<int64_t,int64_t> pii;

pii crm(int64_t x1, int64_t a1, int64_t x2, int64_t a2) {
    int64_t s, t, d;
    extended_euclid(x1, x2, s, t, d);
    if (a1 % d != a2 % d) return pii(0, -1);
    return pii(mod(s * a2 * x1 + t * a1 * x2, x1 * x2) / d, x1 * x2 / d);
}

pii crm(const vector<int64_t> &x, const vector<int64_t> &a){
    pii ret = pii(a[0], x[0]);
    for (size_t i = 1; i < x.size(); ++i) {
        ret = crm(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

//Credits to RagnarGrootKoerkamp
```

2 Graphs

Dijsktra:

```
#define INF (1LL<<60)
#define endl '\n'
#define mp make_pair

typedef pair<int64_t,int64_t> pii;

vector<int64_t> dist, pred;
vector<vector<pii> > g;

void dijkstra(int64_t u){
        int64_t n = g.size();
        dist.assign(n,INF);
        pred.assign(n, -1);
    dist[u] = 0;
    priority_queue<pii> Q;
```

```
    Q.push({-dist[u],u});
    vector<bool> seen (n);

    while(!Q.empty()){
        pii p = Q.top();
        int64_t w = p.second;
        Q.pop();
        if(seen[w]) continue;
            seen[w] = true;
        for(auto to : g[w]){

                if(seen[to.first] || dist[to.first] <= dist[w] + to.second)
                        continue;

            dist[to.first] = dist[w] + to.second;
            pred[to.first] = w;
            Q.push({-dist[to.first],to.first});
        }
    }
}

//Since edit 19/10/2016 not tested.
//Solved : https://open.kattis.com/problems/shortestpath1
//http://codeforces.com/problemset/problem/20/C
//http://www.spoj.com/problems/SHPATH/
```

Floyd Warshall

```
#define INF (1LL << 60)

vector<vector<int64_t> > dist;

void floyd_warshall (){
        int64_t n = dist.size();
    for(int64_t k = 0; k < n; k++){
        for(int64_t i = 0; i < n; i++){
            for(int64_t j = 0; j < n; j++){
                if(dist[i][k] != INF && dist[k][j] != INF){
                    if(dist[i][j] > dist[i][k] + dist[k][j]){
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
    }

    //Extra loop for Infinite loop checks.
    //Alternative check dist[u][u] < 0 || dist[v][v] < 0;
    for(int64_t i = 0; i < n; i++){
                for(int64_t j = 0; j < n; j++){
                    for(int64_t k = 0; k < n; k++){
                        if(dist[i][k] != INF && dist[k][j] != INF && dist[k
                            ][k] < 0){
                             dist[i][j] = -INF;
                        }
                    }
        }
        }
```

```
}

/** The distance options
if(dist[u][v] == INF) cout << "Impossible" << endl;
else if(dist[u][u] == -INF) cout << "-Infinity" << endl;
else cout << dist[u][v] << endl;
**/

//Solved : https://open.kattis.com/problems/allpairspath
```

Bellmanford

```
#define INF (1LL<<60)

struct Edge{
    int64_t u, v, w;
};

vector<int64_t> dist;
vector<Edge> edges;

//s is the start node, n is the amount of nodes.
bool bellmanford(int64_t s, int64_t n){
    dist.clear();
    dist.resize(n,INF);
    dist[s] = 0;

    for(int64_t i = 0; i < n-1; i++){
        for(const Edge& e: edges){
            dist[e.v] = min(dist[e.v],dist[e.u] + e.w);
        }
    }

    for(const Edge& e: edge){
        if(dist[e.v] > dist[e.u] + e.w){
            return false;
        }
    }
}
```

Topological Sort

```
vector<vector<int> > g;

vector<int> topological_sort() {
    int n = g.size();

    vector<int> in_degs(n);
    for(int u = 0; u < n; u++) for(const int &v : g[u]) {
        in_degs[v]++;
    }

    queue<int> s;
    for(int u = 0; u < n; u++) if(in_degs[u] == 0) s.push(u);

    vector<int> order;
    while(!s.empty()) {
```

```
        int u = s.front();
        s.pop();
        order.push_back(u);

        for(const int &v : g[u]) {
            in_degs[v]--;
            if(in_degs[v] == 0) s.push(v);
        }
    }

    return order;
}
```

Disjoint Union

```
// Union-Find Disjoint Sets Library written in OOP manner, using both path
   compression and union by rank heuristics
class UnionFind {                                                   // OOP style
    private:
        vector<int> p, rank, setSize;
        int numSets;
    public:
        UnionFind(int N) {
            setSize.assign(N, 1);
            numSets = N;
            rank.assign(N, 0);
            p.assign(N, 0);
            for (int i = 0; i < N; i++) p[i] = i;
        }
        int findSet(int i) {
            return (p[i] == i) ? i : (p[i] = findSet(p[i]));
        }
        bool isSameSet(int i, int j) {
            return findSet(i) == findSet(j);
        }
        void unionSet(int i, int j) {
            if (!isSameSet(i, j)) {
                numSets--;
                int x = findSet(i), y = findSet(j);
                // rank is used to keep the tree short
                if (rank[x] > rank[y]) {
                    p[y] = x; setSize[x] += setSize[y];
                }
                else{
                    p[x] = y; setSize[y] += setSize[x];
                    if (rank[x] == rank[y]) rank[y]++;
                }
            }
        }
        int numDisjointSets() {
            return numSets;
        }
        int sizeOfSet(int i) {
            return setSize[findSet(i)];
        }
};

//Solved : https://open.kattis.com/problems/minspantree
```

Bipartite Graphs

Bipartite check

```
vector<vector<int> > g;
vector<int> colors;

bool dfs(const int &i, const int &color) {
    if(colors[i] != 0 && colors[i] != color) {
        return false;
    }
    if(colors[i] == 0) {
        colors[i] = color;
        for(unsigned int j = 0; j < g[i].size(); j++) {
            if(!dfs(g[i][j], -color)) {
                return false;
            }
        }
    }
    return true;
}

bool bipartite(const int &p) {
        colors.assign(p,0);
    for(int i = 0; i < p; i++) {
        if(colors[i] == 0 && !dfs(i, 1)) {
            return false;
        }
    }
    return true;
}
```

Hopcroft Karp

```
#define INF (1<<30)

struct Node {
    int match, deg, id;
    int a, b;
    set<int> adj;

    bool operator<(const Node& rhs) const{
        if(deg == rhs.deg) return id > rhs.id;
        else return deg > rhs.deg;
    }
};

vector<Node> U, V;
vector<int> dist;

bool bfs() {
    queue<int> Q;
    for(int i = 1; i < U.size(); i++) {
        if(U[i].match == 0) {
            dist[i] = 0;
            Q.push(i);
```

```
        } else {
            dist[i] = INF;
        }
    }
    dist[0] = INF;
    while(!Q.empty()) {
        int i = Q.front();
        Q.pop();
        if(dist[i] < dist[0]) {
            for(const int &j : U[i].adj) {
                if(dist[V[j].match] == INF) {
                    dist[V[j].match] = dist[i] + 1;
                    Q.push(V[j].match);
                }
            }
        }
    }
    return dist[0] != INF;
}

bool dfs(int i) {
    if(i != 0) {
        for(const int &j : U[i].adj) {
            if(dist[V[j].match] == dist[i] + 1) {
                if(dfs(V[j].match)) {
                    V[j].match = i;
                    U[i].match = j;
                    return true;
                }
            }
        }
        dist[i] = INF;
        return false;
    }
    return true;
}

int hopcroft_karp() {
    int matching = 0;
    while(bfs()) {
        for(int i = 1; i < U.size(); i++) {
            if(U[i].match == 0) {
                if(dfs(i)) {
                    matching++;
                }
            }
        }
    }
    return matching;
}
```

MST

Kruskall

```
struct Edge{
    int64_t first, second, weight;
};
```

```
bool edge_compare(Edge l, Edge r){
    return (l.weight < r.weight);
}

vector<Edge> kruskal(vector<Edge> e, int64_t n){
    UnionFind UF((int)n);
    vector<Edge> A;
    sort(e.begin(),e.end(),edge_compare);
    for(int i = 0; i < e.size(); i++){
        Edge edge = e[i];
        int u = edge.first, v = edge.second;
        if(!UF.isSameSet(u,v)){
            A.push_back(edge);
            UF.unionSet(u,v);
        }
    }
    return A;
}

//Solved : https://open.kattis.com/problems/minspantree
```

Prim

```
typedef pair<int64_t,int64_t> pii;

vector<vector<pii> > g;
vector<int64_t> dist, prev;

int64_t prim(int64_t start){
        int64_t n = g.size();
        dist.assign(n,INF);
    prev.assign(n,-1);
    int64_t length = 0;
    dist[start] = 0;

    priority_queue<pii> Q;
    Q.push({-dist[start],start});

    vector<bool> seen (n);

    while(!Q.empty()){
        pii p = Q.top();
        int64_t w = p.second;
        Q.pop();
        if(seen[w]) continue;
        seen[w] = true;
        length += dist[w];
        for(auto to : g[w]){
            if(!seen[to.first] && dist[to.first] > to.second){
                dist[to.first] = to.second;
                prev[to.first] = w;
                Q.push({-dist[to.first],to.first});
            }
        }
    }
    return length;
}
```

MaxFlow

Maxflow Julian

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
#include <set>
#include <climits>

using namespace std;

bool bfs(const vector<vector<int> > &graph, int s, int t, vector<int> &
    parents) {
    int n = graph.size();
    vector<bool> seen(n);

    queue<int> q;
    q.push(s);
    seen[s] = true;
    parents[s] = -1;

    while(!q.empty()) {
        int u = q.front();
        q.pop();

        for(int v = 0; v < n; v++) {
            if(seen[v] || graph[u][v] <= 0) continue;
            q.push(v);
            parents[v] = u;
            seen[v] = true;
        }
    }

    return (seen[t] == true);
}

int ford_fulkerson(vector<vector<int> > &resid, int s, int t) {
    int n = resid.size(), max_flow = 0;
    vector<int> parents(n);

    while(bfs(resid, s, t, parents)) {
        int path_flow = INT_MAX;
        for(int v = t; v != s; v = parents[v]) {
            path_flow = min(path_flow, resid[parents[v]][v]);
        }

        for(int v = t; v != s; v = parents[v]) {
            resid[parents[v]][v] -= path_flow;
            resid[v][parents[v]] += path_flow;
        }

        max_flow += path_flow;
    }
```

```
    return max_flow;
}


int main() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    vector<vector<int> > graph(n, vector<int>(n, 0));

    for(int i = 0; i < m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        graph[u][v] = c;
    }

    vector<vector<int> > resid(graph.begin(), graph.end());
    int max_flow = ford_fulkerson(resid, s, t), used = 0;

    for(int u = 0; u < n; u++) for(int v = 0; v < n; v++) {
        int f = graph[u][v] - resid[u][v];
        if(f > 0) used++;
    }

    cout << n << " " << max_flow << " " << used << endl;
    for(int u = 0; u < n; u++) for(int v = 0; v < n; v++) {
        int f = graph[u][v] - resid[u][v];
        if(f > 0) cout << u << " " << v << " " << f << endl;
    }

    return 0;
}
```

Maxflow David

```
#define INF (1LL<<60)
typedef pair<int64_t,int64_t> pii;

vector<pii> pred;
vector<vector<int64_t> > es, ec;
vector<int64_t> dist;

void dijkstras(int64_t s){
    int64_t n = es.size();
    dist.assign(n,0);
    pred.assign(n,{-1,0});
    vector<bool> seen (n);
    priority_queue<pii> Q;
    dist[s] = INF;
    Q.push({0,s});

    while(!Q.empty()){
        int64_t cur = Q.top().second;
        Q.pop();
        if(seen[cur]) continue;
        seen[cur] = true;

        for(int i = 0; i < es[cur].size(); i++){
```

```
            int64_t t = es[cur][i],
                    c = ec[cur][i];

            if(seen[t] || min(dist[cur],c) <= dist[t]) continue;

            dist[t] = min(dist[cur],c);
            pred[t] = {cur,i};
            Q.push({dist[t],t});
        }
    }
}

int64_t maxflow(int64_t s, int64_t t){
    int64_t n = es.size();
    int64_t flow = 0;
    vector<vector<int64_t> > me(0); //To find and create backedges in
        residual graph
    for(int i = 0; i < n ; i++){
        me.push_back(vector<int64_t> (es[i].size(), -1));
    }

    while(true){
        dijkstras(s);

        if(dist[t] == 0) break;
        flow += dist[t];

        int64_t cur = t;
        while(cur != s){
            int64_t f = pred[cur].first;
            int64_t j = pred[cur].second;
            ec[f][j] -= dist[t];
            if(me[f][j] == -1){
                me[f][j] = es[cur].size();
                me[cur].push_back(j);
                es[cur].push_back(f);
                ec[cur].push_back(dist[t]);
            } else {
                ec[cur][me[f][j]] += dist[t];
            }
            cur = f;
        }
    }
    return flow;
}
```

Min Cost Max Flow

```
#define INF (1LL<<60)

typedef pair<int64_t,int64_t> pii;

vector<vector<int64_t> > es, ecap, ecost;
vector<int64_t> pot, dist;
vector<pii> pred;

void dijkstras(int64_t s){
    int64_t N = es.size();
```

```
    priority_queue<pii> Q;
    vector<bool> seen (N,false);

    dist.assign(N,INF);
    pred.assign(N,{-1,0});

    dist[s] = 0;
    Q.push({0,s});

    while(!Q.empty()){
        int64_t cur = Q.top().second;
        Q.pop();
        if(seen[cur]) continue;
        seen[cur] = true;

        for(int i = 0; i < es[cur].size(); i++){
            int64_t t = es[cur][i],
                    c = ecost[cur][i];

            if(seen[t] || dist[cur] + c >= dist[t]) continue;
                        //Add EXTRA CHECKS here!
            if(ecap[cur][i] == 0) continue;

            dist[t] = dist[cur] + c;
            pred[t] = {cur,i};
            Q.push({-dist[t],t});
        }
    }
}

pii maxflow(int64_t s, int64_t t){
    int64_t n = es.size();
    int64_t flow = 0, cost = 0;
    pot.assign(n,0);
    vector<vector<int64_t> > me(0);
    for(int i = 0; i < n; i++){
        me.push_back(vector<int64_t> (es[i].size(), -1));
    }
    while(true){
        dijkstras(s);
        if(dist[t] == INF) break;

        //find maxadd
        int64_t maxadd = INF;
        int64_t cur = t;
        while(cur != s){
            maxadd = min(maxadd, ecap[pred[cur].first][pred[cur].second]);
            cur = pred[cur].first;
        }

        cost += (pot[t] + dist[t]) * maxadd;
        flow += maxadd;

        //Potential adjust
        for(int i = 0; i < n; i++){
            for(int j = 0; j < es[i].size(); j++){
                ecost[i][j] += dist[i] - dist[es[i][j]];
            }
```

```
                pot[i] += dist[i];
            }

            //adjust edges
            cur = t;
            while(cur != s){
                int64_t f = pred[cur].first,
                        j = pred[cur].second;
                ecap[f][j] -= maxadd;
                if(me[f][j] == -1){
                    me[f][j] = es[cur].size();
                    me[cur].push_back(j);
                    es[cur].push_back(f);
                    ecost[cur].push_back(0);
                    ecap[cur].push_back(maxadd);
                } else {
                    ecap[cur][me[f][j]] += maxadd;
                }
                cur = f;
            }
        }
        return {flow,cost};
}
```

3 Geometry

2D

```
// All functions should also work with other number types (doubles, floats)
struct Point {
        int x, y;
}

// Check orientation of point triplets
int orient(Point p, Point q, Point r) {
        int v = (q.y - p.y) * (r.x - q.x) -
                (q.x - p.x) * (r.y - q.y);
        if(v == 0) return 0;
        return v < 0 ? -1 : 1;
}

// Compute Euclidean distance between points
int dist(Point p, Point q) {
        return sqrt((p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.x - q.y));
}


//Point line distance
double point_line_dist(Point p, Point q, Point r) {
    double a = p.x - q.x,
        b = p.y - q.y,
        c = r.x - q.x,
        d = r.y - q.y;

    double dot = a * c + b * d,
        mag_sq = c * c + d * d;
```

```
    double v = -1;
    if(mag_sq != 0) v = dot / mag_sq;

    double dx = p.x - q.x - v * c,
        dy = p.y - q.y - v * d;

    if(v < 0) dx = p.x - q.x, dy = p.y - q.y;
    if(v > 1) dx = p.x - r.x, dy = p.y - r.y;

    return sqrt(dx * dx + dy * dy);
}

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool on_segment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
       return true;

    return false;
}

// The main function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool intersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orient(p1, q1, p2);
    int o2 = orient(p1, q1, q2);
    int o3 = orient(p2, q2, p1);
    int o4 = orient(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && on_segment(p1, p2, q1)) return true;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && on_segment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && on_segment(p2, p1, q2)) return true;

     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && on_segment(p2, q1, q2)) return true;

    return false; // Doesn't fall in any of the above cases
}

//Polygon Area
double polygon_area(const vector<Point> &poly) {
    double area = 0.0;
    int n = poly.size(),
```

```
        j = n - 1;

    for(int i = 0; i < n; i++) {
        area += (poly[j].x + poly[i].x) * (poly[j].y - poly[i].y);
        j = i;
    }

    return area / 2.0;
}

//Rotate Point
Point rotate_point(const Point &p, const Point &c, double v) {
    double x = p.x - c.x,
        y = p.y - c.y;

    double rot_x = x * cos(v) - y * sin(v),
        rot_y = x * sin(v) + y * cos(v);

    return { rot_x + c.x, rot_y + c.y };
}

//Convex Hull
vector<Point> convex_hull(vector<Point> ps) {
    int n = ps.size(), k = 0;
    vector<Point> hull(2 * n);

    sort(ps.begin(), ps.end());

    for(int i = 0; i < n; i++) {
        while(k >= 2 && orient(hull[k - 2], hull[k - 1], ps[i]) <= 0) k--;
        hull[k++] = ps[i];
    }

    for(int i = n - 2, t = k + 1; i >= 0; i--) {
        while(k >= t && orient(hull[k - 2], hull[k - 1], ps[i]) <= 0) k--;
        hull[k++] = ps[i];
    }

    hull.resize(k - 1);
    return hull;
}
```

3D

```
struct Point {
    double x, y, z;
};

Point cartesian(double lat, double lon) {
    lat *= M_PI / 180.0, lon *= M_PI / 180.0;
    return {cos(lat) * cos(lon),
        cos(lat) * sin(lon),
        sin(lat)};
}

double magnitude(const Point &p) {
    return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}
```

```cpp
Point normalize(const Point &p) {
    double length = magnitude(p);
    return {p.x / length,
        p.y / length,
        p.z / length};
}

Point cross(const Point &p, const Point &q) {
    return {p.y * q.z - p.z * q.y,
        p.z * q.x - p.x * q.z,
        p.x * q.y - p.y * q.x};
}

double dot(const Point &p, const Point &q) {
    return p.x * q.x + p.y * q.y + p.z * q.z;
}

double dist(const Point &p, const Point &q) {
    return atan2(magnitude(cross(p, q)), dot(p, q));
}

Point negation(const Point &p) {
    return {-p.x, -p.y, -p.z};
}

pair<Point, Point> compute_intersections(const Point &p, const Point &q,
   const Point &s, const Point &t) {
    Point v1 = cross(p, q), v2 = cross(s, t),
        d = cross(v1, v2),
        first = normalize(d), second = negation(first);
    return make_pair(first, second);
}

bool on_arc(const Point &p, const Point &q, const Point &s) {
    return abs(dist(p, q) - dist(p, s) - dist(q, s)) < epsilon;
}
```

4 Segment Trees

Basic

```cpp
vector<int> f,st;
//f for original array values
//st for range queries values

int left (int i){
    return (i << 1);
}

int right(int i){
    return (i << 1) + 1;
}

//rangemax query, change returns to answer different questions.
int rmq(int i, int L, int R, int l, int r){
    if(l > R || r < L) return -1;
```

```
    if(L >= l && R <= r) return st[i];
    return max( rmq(left(i), L,(L+R)/2, l,r, st), rmq(right(i),((L+R)/2)+1,
        R,l,r));
}

void build(int i, int L, int R){
    if(L == R){
        st[i] = f[L];
    } else {
        build(left(i),L,(L+R)/2);
        build(right(i),((L+R)/2)+1, R);
        st[i] = max(st[left(i)],st[right(i)]);
    }
}

int main()
{
    f.assign(n+1);
    st.assign(4*(n+1));
    build(1,1,n);
}
```

Lazy update

```
int query(int node, int L, int R, int l, int r){
    if(l > R || r < L) return 0;


    if(lazy[node].size() != 0){
        for(int i = 0; i < lazy[node].size(); i++){
            st[node] = magic(L,R,st[node],lazy[node][i]);
        }
        if(L != R){
            vector<char> a = lazy[left(node)], b = lazy[node];
            a.insert(a.end(),b.begin(),b.end());
            lazy[left(node)] = a;
            a = lazy[right(node)];
            a.insert(a.end(),b.begin(),b.end());
            lazy[right(node)] = a;
        }
        lazy[node].clear();
    }

    if(L >= l && R <= r) return st[node];

    return query(left(node),L,(L+R)/2,l,r) + query(right(node),(L+R)/2 + 1,
       R,l,r);
}

void update(int node, int L, int R, int l, int r, char c){

    if(lazy[node].size() != 0){
        for(int i = 0; i < lazy[node].size(); i++){
            st[node] = magic(L,R,st[node],lazy[node][i]);
        }
        if(L != R){
            vector<char> a = lazy[left(node)], b = lazy[node];
            a.insert(a.end(),b.begin(),b.end());
```

```
            lazy[left(node)] = a;
            a = lazy[right(node)];
            a.insert(a.end(),b.begin(),b.end());
            lazy[right(node)] = a;
        }
        lazy[node].clear();
    }

    if(l > R || r < L) return;
    if(L >= l && R <= r) {
        st[node] = magic(L,R,st[node],c);
        if(L != R){
            lazy[left(node)].push_back(c);
            lazy[right(node)].push_back(c);
        }
        return;
    }

    update(left(node),L,(L+R)/2,l,r,c);
    update(right(node),(L+R)/2 + 1, R, l,r,c);

    st[node] = st[left(node)] + st[right(node)];

}
```

5 Python

Read in

```
for line in sys.stdin:
    ab = line.split()
    a = int(ab[0])
    b = int(ab[1])
# Solve the test case and output the answer
```

[language=Python]