ICPC cheat sheet

PatrickBitman

```
// AUXILIARY
bool cmp(int u, int v) {
   return st[u] < st[v];
}
void auxiliary() {
   sort(all.begin(), all.end(), cmp);
   for (int i = 0; i < k - 1; i++)
       all.push_back(lca(all[i], all[i + 1]));
   sort(all.begin(), all.end(), cmp);
   all.resize(unique(all.begin(), all.end()) - all.begin());
   stack <int> s;
   for (int u: all) {
       while (s.size() && (st[s.top()] > st[u] || en[s.top()] <= st[u]))
           s.pop();
       if (s.size())
           tree[s.top()].push_back({u, h[u] - h[s.top()]});
       s.push(u);
   }
}


// 2-SAT ([a | b] & [a | !b] & ... & [c | !a])
void add_edge(int u, int v) {
   adj[u].push_back(v);
   rev_adj[v].push_back(u);
}
void add_clause(int u, int v) {
   add_edge(u ^ 1, v);
   add_edge(v ^ 1, u);
}
void dfs(int u, vector <int> *g, vector <int> &vec, int c = 1) {
   vis[u] = c;
   for (int v: g[u])
       if (vis[v] == 0)
           dfs(v, g, vec, c);
   vec.push_back(u);
}
void kosaraju() {
   for (int u = 0; u < n << 1; u++)
       if (vis[u] == 0)
           dfs(u, adj, ord);
   reverse(ord.begin(), ord.end());
   memset(vis, 0, sizeof vis);
   int cnt = 0;
   for (int u: ord)
       if (vis[u] == 0) {
           comp.clear();
           dfs(u, rev_adj, comp, ++cnt);
           for (int v: comp) {
               if (vis[v] == vis[v ^ 1]) {
                   cout << "Impossible" << endl;
                   exit(0);
               }
               if (v & 1 == 1 && vis[v ^ 1] == 0)
                   ans.push_back(v >> 1);
           }
       }
}


// EULER
int deg[N];
vector <int> adj[N], tour;
void add_edge(int u, int v) {
   adj[u].push_back(v);
   deg[u]++;
   deg[v]--;
}
void euler(int u) {
   while (adj[u].size()) {
       int v = adj[u].back();
       adj[u].pop_back();
       euler(v);
   }
   tour.push_back(u);
}
void make_graph() {
   cin >> n >> m;
   while (m--) {
       int u, v;
       cin >> u >> v;
       add_edge(--u, --v);
   }
}
```

```
void find_euler_tour() {
    int x1 = -1, x2 = -1;
    for (int u = 0; u < N; u++)
        if (x1 == -1 && deg[u] == 1)
            x1 = u;
        else if (x2 == -1 && deg[u] == -1)
            x2 = u;
        else if (deg[u] != 0)
            return cout << "NO" << endl, 0;
    for (int u = 0; u < N; u++)
        if (adj[u].size() && (u == x1 || x1 == -1)) {
            euler(u);
            break;
        }

    for (int u = 0; u < N; u++)
        if (adj[u].size())
            return cout << "NO" << endl, 0;
    reverse(tour.begin(), tour.end());
    for (int x: tour)
        cout << x + 1 << ' ';
}


// CUT VERTEX
void dfs(int v,int parent){
    dp[v]=h[v];
    mark[v]=true;
    int num=0;
    for(int u: adj[v]) {
        if(!mark[u]){
            h[u]=h[v]+1;
            dfs(u,v);
            if(v!=1 && dp[u]>=h[v])is[v]=true;
            dp[v]=min(dp[v],dp[u]);
            num++;
        }
        else{
            if(u!=parent){
                dp[v]=min(dp[v],h[u]);
            }
        }
    }
```

```
   if(v==1 && num>1)
       is[v]=true;
}


// CUT EDGE
void dfs(int v,int parent,int index){
   dp[v]=h[v];
   mark[v]=true;
   for(int i=0;i<adj[v].size();i++){
       int u=adj[v][i].first;
       int ind=adj[v][i].second;
       if(!mark[u]){
           h[u]=h[v]+1;
           dfs(u,v,ind);
           dp[v]=min(dp[v],dp[u]);
       }
       else{
           if(u!=parent){
               dp[v]=min(dp[v],h[u]);
           }
       }
   }
   if(v!=1){
       if(dp[v]==h[v])
           is[index]=true;
   }
}


// ordered set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

typedef tree<pair <int, int>,
   null_type,
   less<pair <int, int>>,
   rb_tree_tag,
   tree_order_statistics_node_update> ordered_set;
```

```
struct multi_ordered_set {
    ordered_set os;
    unordered_map <int, int> cnt;
    void insert(int x) {
        os.insert({x, cnt[x]});
        cnt[x]++;
    }
    void erase(int x) {
        if (cnt[x]) {
            cnt[x]--;
            os.erase({x, cnt[x]});
        }
    }
    int find_by_order(int x) { // meqdare x om ro mi de (0 base)
        return os.find_by_order(x)->first;
    }
    int order_of_key(int x) { // mige adade x avalin ja koja ast (0 base)
        return os.order_of_key({x, 0});
    }
    int size() {
        return os.size();
    }
};


// SEGMENT 2D
const int N = 750 + 4;
struct node{
    int x;
};
struct seg2d{
    int n, m;
    node arr[N][N];
    node seg[4 * N][4 * N];
    node emp;
    node merge(node a, node b){
        node rt;
        rt.x = max(a.x, b.x);
        return rt;
    }
```

```
void buildy(int idx, int lx, int rx, int idy, int ly, int ry){
    if(ry - ly == 1){
        if (rx - lx == 1)
            seg[idx][idy] = arr[lx][ly];
        else
            seg[idx][idy] = merge(seg[2 * idx][idy], seg[2 * idx + 1][idy]);
    }else {
        int mid = (ly + ry) / 2;
        buildy(idx, lx, rx, 2 * idy, ly, mid);
        buildy(idx, lx, rx, 2 * idy + 1, mid, ry);
        seg[idx][idy] = merge(seg[idx][2 * idy], seg[idx][2 * idy + 1]);
    }
}
void buildx(int idx, int lx, int rx){
    if(rx - lx != 1){
        int mid = (lx + rx) / 2;
        buildx(2 * idx, lx, mid);
        buildx(2 * idx + 1, mid, rx);
    }
    buildy(idx, lx, rx, 1, 1, m + 1);
}

node gety(int idx, int idy, int ly, int ry, int st, int en) {
    if (st >= en)
        return emp;
    if (ly == st && ry == en)
        return seg[idx][idy];
    int mid = (ly + ry) / 2;
    return merge(gety(idx, 2 * idy, ly, mid, st, min(en, mid))
            , gety(idx, 2 * idy + 1, mid, ry, max(st, mid), en));
}

node getx(int idx, int lx, int rx, int stx, int enx, int sty, int eny) {
    if (stx >= enx)
        return emp;
    if (lx == stx && rx == enx)
        return gety(idx, 1, 1, m + 1, sty, eny);
    int mid = (lx + rx) / 2;
    return merge(getx(2 * idx, lx, mid, stx, min(enx, mid), sty, eny)
            , getx(2 * idx + 1, mid, rx, max(stx, mid), enx, sty, eny));
}
```

```
    void updatey(int idx, int lx, int rx, int idy, int ly, int ry, int x, int y,
int new_val) {
        if (ry - ly == 1) {
            if (rx - lx == 1){
                node X;
                X.x = new_val;
                seg[idx][idy] = X;
            }
            else
                seg[idx][idy] = merge(seg[2 * idx][idy], seg[2 * idx + 1][idy]);
        } else {
            int mid = (ly + ry) / 2;
            if (y < mid)
                updatey(idx, lx, rx, 2 * idy, ly, mid, x, y, new_val);
            else
                updatey(idx, lx, rx, 2 * idy + 1, mid, ry, x, y, new_val);
            seg[idx][idy] = merge(seg[idx][2 * idy], seg[idx][2 * idy + 1]);
        }
    }

    void updatex(int idx, int lx, int rx, int x, int y, int new_val) {
        if (rx - lx > 1) {
            int mid = (lx + rx) / 2;
            if (x < mid)
                updatex(2 * idx, lx, mid, x, y, new_val);
            else
                updatex(2 * idx + 1, mid, rx, x, y, new_val);
        }
        updatey(idx, lx, rx, 1, 1, m + 1, x, y, new_val);
    }
};


// 0 base suffix array O(nlg)
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

```
// FFT
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
```

```
   fft(fa, false);
   fft(fb, false);
   for (int i = 0; i < n; i++)
       fa[i] *= fb[i];
   fft(fa, true);
   vector<int> result(n);
   for (int i = 0; i < n; i++)
       result[i] = round(fa[i].real());
   return result;
}
// NTT mode = 988244353
const int md = 998244353;
int w[24], wi[24], inv2;

inline int mkey(int a,int b){
   int rt=a + b;
   if(rt >= md)
       rt-=md;
   return rt;
}

int pw(int a, int b){
   int rt = 1;
   while(b > 0){
       if(b & 1)
           rt = 1LL * rt * a % md;
       b /= 2;
       a = 1LL * a * a % md;
   }
   return rt;
}

void pre(){
   w[23] = 31;
   FORR(i, 22, 0)
       w[i] = 1LL * w[i + 1] * w[i + 1] % md;
   FOR(i, 0, 23)
       wi[i] = pw(w[i], md - 2);
   inv2 = pw(2, md - 2);
}
void fft(vector<int> &a, bool invert){
   int n = SZ(a);
```

```
    if(n == 1)
        return;
    vector<int> a0(n / 2), a1(n / 2);
    FOR(i, 0, n / 2 - 1){
        a0[i] = a[2 * i];
        a1[i] = a[2 * i + 1];
    }
    fft(a0, invert);
    fft(a1, invert);

    int W = 1;
    int wn = w[__builtin_ctz(n)];
    if(invert)
        wn = wi[__builtin_ctz(n)];
    FOR(i, 0, n / 2 - 1){
        a[i] = mkey(a0[i], (1LL * W * a1[i] % md));
        a[i + n / 2] = mkey(a0[i], (md - (1LL * W * a1[i] % md)));
        if(invert){
            a[i] = 1LL * a[i] * inv2 % md;
            a[i + n / 2] = 1LL * a[i + n / 2] * inv2 % md;
        }
        W = 1LL * wn * W % md;
    }
}

vector<int> mul(vector<int> a, vector<int> b){
    vector<int> fa(all(a)), fb(all(b));
    int n=1;
    while(n<SZ(a)+SZ(b)) n *= 2;
    fa.resize(n);
    fb.resize(n);
    fft(fa,0);
    fft(fb,0);
    FOR(i, 0, n - 1)
        fa[i] = 1LL * fa[i] * fb[i] % md;
    fft(fa, 1);
    vector<int> res(n);
    FOR(i, 0, n - 1){
        res[i] = fa[i];
    }
    return res;
}
```

```
// mcmf
ll n, m, N=0, M=0;
struct edge {
   ll u, v, cost, cap;
   edge () {}
   edge(int u1, int v1, int cost1, int cap1){
       u = u1;
       v = v1;
       cost = cost1;
       cap = cap1;
   }
};
edge ed[MXE];
vector<int> adj[MX];

// edges from left to right
void add_edge(int u, int v, int cap, int cost=0) {
   ed[M] = edge(u, v, cost, cap);
   adj[u].pb(M++);
   ed[M] = edge(v, u, -cost, 0);
   adj[v].pb(M++);
}

// used spfa
pii cheapest_flow(int source, int sink) {
   int d[MX] = {}, p[MX]={};
   bool inQ[MX] = {};
   queue<int> q;
   FOR(i, 1, N+2) d[i] = inf;
   d[source] = 0;
   int u = source;
   inQ[u] = true;
   q.push(u);
   while(!q.empty()) {
       u = q.front();
       q.pop();
       inQ[u] = false;
       for(int i:adj[u]) if(ed[i].cap) {
           int v = ed[i].v;
           if(d[v] > d[u] + ed[i].cost) {
               d[v] = d[u] + ed[i].cost;
               p[v] = i;
               if(!inQ[v]) q.push(v);
```

```
            }
        }
    }
    u = sink;
    pii ans = pii(0, 0);
    if(d[u] == inf) return ans;
    while(u != source) {
        int i=p[u];
        ed[i].cap --;
        ed[i^1].cap ++;
        ans.S += ed[i].cost;
        u = ed[i].u;
    }
    ans.F = 1;
    return ans;
}

// returns (max flow, min cost);
pii mcmf(int source, int sink) {
    pii fc; // fc : (flow, cost)
    pii ans = pii(0, 0);
    do {
        fc = cheapest_flow(source, sink);
        ans.F += fc.F;
        ans.S += fc.S;
    } while(fc.F > 0);
    return ans;
}
// build graph in main using add_edge, don't forget to check N, MX, source,
sink
// min-cut & max-flow
// e[u][v] represents the remaining capacity of the uv edge in that direction
// look out for MX!
ll N, n, m, e[MX][MX], flow[MX][MX];
bool mark_f[MX];

void add_edge(int u, int v, ll cap) {
    e[u][v] += cap;
}

void push_flow(int u, int v, ll f) {
    flow[u][v] += f;
    flow[v][u] -= f;
    e[u][v]-=f;
    e[v][u]+=f;
}

ll dfs(int u, ll in_f, ll min_w, int sink){
    if(u==sink) return in_f;
    mark_f[u] = true;
    for(int v=1; v<=N; v++) if(!mark_f[v] and e[u][v]>=min_w){
        ll f = dfs(v, min(in_f, e[u][v]), min_w, sink);
        if(f) {
            push_flow(u, v, f);
            return f;
        }
    }
    return 0;
}

int main(){
    cin>>n;
    N = 2*n+2;
    FOR(i, 1, n){
        string s;
        cin>>s;
        FOR(j, 0, n-1) if(s[j] != '.'){
            add_edge(i, j+n+1, inf);
        }
    }
    int source = 2*n+1;
    int sink = 2*n+2;
    FOR(i, 1, n) {
        add_edge(source, i, 1);
    }
    FOR(i, n+1, 2*n) {
        add_edge(i, sink, 1);
    }
    ll max_flow = 0;
    // LOGW is the log of max edge capacity
    ll i = 1ll<<LOGW; while(i>0){
        FOR(u, 1, N) mark_f[u] = false;
        ll f = dfs(source, infl, i, sink);
        max_flow+= f;
        if(f==0) i>>=1;
    }
    cout<<max_flow<<'\n';
    //iterate all existing edges:
    FOR(u, 1, N) FOR(v, 1, N) if(flow[u][v])
        // condition for an edge to be in min cut
        if(mark_f[u] and !mark_f[v])
            cout<<u<<' '<<v<<'\n';
}
/*
  in bipartite graphs:
  min edge cover = n - max matching
  min vertex cover = n - max independent set
*/


// Convex Hull
struct Line {
    double a, b, x_left;
    Line(double a, double b): a(a), b(b), x_left(-1e18) {};
    double get_y(double x) {
        return a * x + b;
    }
    bool operator < (double x) const {
        return x_left < x;
    }
};
double inter_x(Line l1, Line l2) {
    return (l1.b - l2.b) / (l2.a - l1.a);
}
struct Convex_Hull {
    deque <Line> dq;
    enum Type {front, back} type;
    vector <pair <Type, vector <Line>>> rem;
    void pop_back() {
        dq.pop_back();
    }
    void push_back(Line l) {
        l.x_left = -1e18;
        if (dq.size())
            l.x_left = inter_x(l, dq.back());
        dq.push_back(l);
    }
    void add_back_line(Line l) {
        rem.push_back({back, {}});
        while (dq.size() >= 2 && dq.back().x_left >= inter_x(dq.back(), l)) {
            rem.back().second.push_back(dq.back());
            pop_back();
        }
        push_back(l);
    }
    void pop_front() {
        dq.pop_front();
        dq[0].x_left = -1e18;
    }
    void push_front(Line l) {
        if (dq.size())
            dq[0].x_left = inter_x(l, dq[0]);
        l.x_left = -1e18;
        dq.push_front(l);
    }
    void add_front_line(Line l) {
        rem.push_back({front, {}});
        while (dq.size() >= 2 && dq[1].x_left <= inter_x(dq[0], l)) {
            rem.back().second.push_back(dq[0]);
            pop_front();
        }
        push_front(l);
    }
    void undo() {
        if (rem.back().first == front)
            pop_front();
        else
            pop_back();
        reverse(rem.back().second.begin(), rem.back().second.end());
        for (auto l: rem.back().second)
            rem.back().first == front? push_front(l): push_back(l);
        rem.pop_back();
    }
    double get_max(double x) {
        auto best_line = lower_bound(dq.begin(), dq.end(), x) - 1;
        return best_line->get_y(x);
    }
};
```

```
//Convex Polygon
struct Point {
    long long x, y;
    bool type;
    Point() {}
    Point(long long x, long long y, bool type) : x(x), y(y), type(type) {}
    bool operator < (Point p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
};
long long cross_product(Point O, Point A, Point B) {
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}
double shib(Point a, Point b) {
    return 1. * (a.y - b.y) / (a.x - b.x);
}
vector <Point> convex_hull(vector <Point> points) {
    int n = points.size(), k = 0;
    sort(points.begin(), points.end());
    if (n <= 3)
        return points;
    vector <Point> ans(2 * n);
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && cross_product(ans[k - 2], ans[k - 1], points[i]) < 0)
            k--;
        ans[k++] = points[i];
    }
    for (int i = n - 1, t = k + 1; i > 0; --i) {
        while (k >= t && cross_product(ans[k - 2], ans[k - 1], points[i - 1]) <
0)
            k--;
        ans[k++] = points[i - 1];
    }
    ans.resize(k - 1);
    return ans;
}


//Dynamic convex hell
enum Type {line, query};
struct Line {
    double a, b, x_left;
    Type type;
    long long val;
    Line() : type(line) {}
    Line(long long a, long long b) : a(a), b(b), type(line), x_left(-1e18) {}
    Line(long long val) : val(val), type(query) {}
    long long get_y(long long x) const {
        return a * x + b;
    }
    bool operator < (Line l) const {
        if (l.type == line)
            return a < l.a;
        return x_left < l.val;
    }
};
bool are_parallel(Line l1, Line l2) {
    return l1.a == l2.a;
}
double inter_x(Line l1, Line l2) {
    return are_parallel(l1,l2)? 1e18: 1.0 * (l2.b - l1.b) / (l1.a - l2.a);
}
struct Convex_Hull_Dynamic {
    set <Line> s;
    bool has_prev(set <Line>::iterator it) {
        return it != s.begin();
    }
    bool has_next(set <Line>::iterator it) {
        return it != s.end() && next(it) != s.end();
    }
    bool irrelevant(Line l1, Line l2, Line l3) {
        return inter_x(l1, l3) <= inter_x(l1, l2);
    }
    bool irrelevant(set<Line>::iterator it) {
        return has_prev(it) && has_next(it) && irrelevant(*prev(it), *it,
*next(it));
    }
    set <Line>::iterator update_left_border(set <Line>::iterator it) {
        if (has_prev(it) == false)
            return it;
        Line tmp(*it);
        tmp.x_left = inter_x(*it, *prev(it));
        it = s.erase(it);
        it = s.insert(it, tmp);
        return it;
    }
    void add_line(Line l) {
        auto it = s.lower_bound(l);
        if (it != s.end() && are_parallel(*it, l)) {
            if (it->b < l.b)
                it = s.erase(it);
            else
                return;
        }
        it = s.insert(it, l);
        if (irrelevant(it)) {
            s.erase(it);
            return;
        }
        while (has_prev(it) && irrelevant(prev(it)))
            s.erase(prev(it));
        while (has_next(it) && irrelevant(next(it)))
            s.erase(next(it));
        it = update_left_border(it);
        if (has_prev(it))
            update_left_border(prev(it));
        if (has_next(it))
            update_left_border(next(it));
    }
    long long get_max(long long x) {
        auto best_line = --s.lower_bound(Line(x));
        return best_line->get_y(x);
    }
};


//Matrices
struct Matrix {
    int n, m;
    vector <vector <long long>> val;
    Matrix(int n, int m, int v = 0) : n(n), m(m) {
        for (int i = 0; i < n; i++) {
            val.push_back({});
            val.back().resize(m, 0);
            if (i < m)
                val[i][i] = v;
        }
    }
    Matrix operator * (Matrix a) {
        Matrix res(n, a.m);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < a.m; j++)
                for (int k = 0; k < m; k++)
                    res.val[i][j] += val[i][k] * a.val[k][j];


        return res;
    }
    Matrix operator ^ (int b) {
        if (b == 0)
            return Matrix(n, m, 1);
        auto tmp = *this ^ (b / 2);
        return tmp * tmp * (b % 2? *this: Matrix(n, m, 1));
    }
};
```

```
int r(int n) {
    return rand() % n + 1;
}

int main() {
    int c;
    cin >> c;
    srand(c);
    int n = r(10);
}
```

```
g++ myCode.cpp -std=c++17 -o myCode.out
g++ correctCode.cpp -std=c++17 -o correctCode.out
g++ generator.cpp -std=c++17 -o generator.out

ok=0;

for i in `seq 1 100`
do
```

```
    echo $i > tmp
    ./generator.out < tmp > test
    ./correctCode.out < test > correctRet
    ./myCode.out < test > myRet
    diff myRet correctRet > /dev/null
    if [ $? == 0 ]
    then
        ((ok++))
    else
        echo "Wrong answer on test $i"
        echo "Test"
        ./generator.out < tmp
        echo "Output"
        ./myCode.out < test
        echo "Answer"
        ./correctCode.out < test
        break
    fi
done
if [ $ok == 100 ]
then
    echo ACCEPTED
fi
```

```
// returns d = 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;
}
```

تقاطع دو خط: $aX + b = Y,\ a'X + b' = Y \ \ -> \ \ x_0 = \frac{b-b'}{a'-a}$

فاصله نقطه از خط: $(x_0,\ y_0),\ aX + bY + c = 0 \ \ -> \ \ d = \frac{|ax_0+by_0+c|}{\sqrt{a^2+b^2}}$

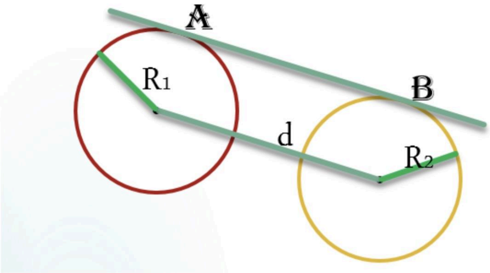
فاصله دو خط موازی: $aX + bY + c = 0,\ a'X + b'Y + c' = 0 \ \ -> \ \ d = \frac{|c-c'|}{\sqrt{a^2+b^2}}$

زاویه بین دو خط با شیب‌های a و a' برحسب رادیان: $tan(\alpha) = \left|\frac{a-a'}{1+aa'}\right|$

مساحت مثلث برحسب ۲ ضلع و زاویه بین: $\frac{1}{2}.a.b.sin(\alpha)$

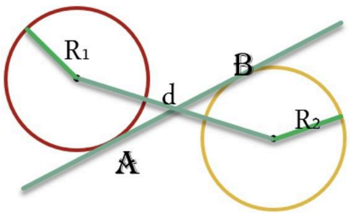
مساحت مثلث برحسب ۳ راس آن: $\frac{1}{2}\left[x_A(y_B - y_C) + x_B(y_C - y_A) + x_C(y_A - y_B)\right]$

مماس مشترک خارجی

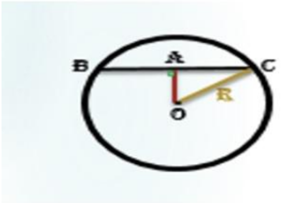


AB^۲=d^۲-(R۱-R۲)^۲

مماس مشترک داخلی

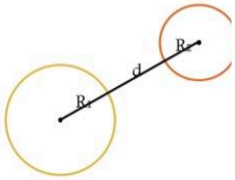


AB^۲=d^۲-(R۱+R۲)^۲



BC = 2√(R²−(OA)²)

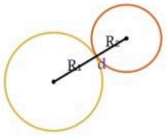
متخارج



d>R۱+R۲

d همان خط المرکزین است.

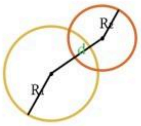
مماس خارج



d=R۱+R۲

d همان خط المرکزین است.

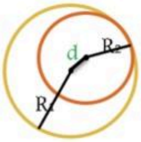
متقاطع



R۱-R۲<d<R۱+R۲

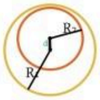
d همان خط المرکزین است.

مماس داخل



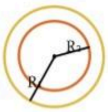
d= اختلاف شعاع ها

متداخل



d< اختلاف شعاع ها

هم مرکز



d=۰

```
struct query {
    int l, r, id;
} q[N];

void add(int l, int r) {
    while (l < r) {
        //..
        l++;
    }
}
void remove(int l, int r) {
    while (l < r) {
        //..
        l++;
    }
}

int main() {
    ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
    sort(q, q + m, [] (query x, query y) {
        return x.l / Sq != y.l / Sq? x.l < y.l: x.r < y.r;
    });
    int st = 0, en = 0;
    for (int i = 0; i < query; i++) {
        auto [l, r, id] = q[i];
        l < st? add(l, st): remove(st, l);
        r < en? remove(r, en): add(en, r);
        st = l, en = r;
        res[id] = ans;
    }
}
```