Mục lục

1 Simple Max Matching

```
bool dfs(int u) {
    if (mx[u] == T) return false;
    mx[u] = T;
    for(int v : ke[u]) {
        if (!my[v] || dfs(my[v])) {
            my[v] = u;
            return true;
        }
    }
    return false;
}

int main() {
    For(i,1,n) {
        T++;
        res += dfs(i);
    }
    // choose my & i
}
```

2 Konig

```
void konig(){
   queue<int> qu;

   f1(i,m) if (!Assigned[i]) qu.push(i);
   f1(i,n) if (!Assigned[N-i]) qu.push(N-i);

   while (qu.size()){
      int u=qu.front(); qu.pop();
      for (int i=0; int v=a[u][i]; i++)
      if (!(Choosed[v]++)) qu.push(Assigned[v]);
   }

   f1(i,m) if (Assigned[i] && !Choosed[i] && !Choosed[Assigned[i]])
   Choosed[i]=true;
}
```

3 HopcroftKarp Max Matching algorithm

```
const int MAXN = 50005, MAXM = 50005;
vector<int> gph[MAXN];
int dis[MAXN], l[MAXN], r[MAXM], vis[MAXN];
void clear() {
    for (int i = 0; i < MAXN; i++)
        gph[i].clear();
}
void add_edge(int l, int r) {
    gph[l].push_back(r);
}
bool bfs(int n) {
    queue<int> que;
    bool ok = 0;
    memset(dis, 0, sizeof(dis));
    for (int i = 0; i < n; i++) {
        if (l[i] == -1 && !dis[i]) {
            que.push(i);
            dis[i] = 1;
        }
    }
    while (!que.empty()) {
        int x = que.front();
        que.pop();
        for (auto &i : gph[x]) {
            if (r[i] == -1)
                ok = 1;
            else if (!dis[r[i]]) {
                dis[r[i]] = dis[x] + 1;
                que.push(r[i]);
            }
        }
    }
    return ok;
}
bool dfs(int x) {
    for (auto &i : gph[x]) {
        if (r[i] == -1 || (!vis[r[i]] && dis[r[i]] == dis[x] + 1 &&
                           dfs(r[i]))) {
            vis[r[i]] = 1;
            l[x] = i;
            r[i] = x;
            return 1;
        }
    }
    return 0;
}
int match(int n) {
    memset(l, -1, sizeof(l));
    memset(r, -1, sizeof(r));
    int ret = 0;
    while (bfs(n)) {
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < n; i++)
            if (l[i] == -1 && dfs(i))
                ret++;
    }
    return ret;
}
bool chk[MAXN + MAXM];
void rdfs(int x, int n) {
    if (chk[x])
        return;
    chk[x] = 1;
    for (auto &i : gph[x]) {
        chk[i + n] = 1;
        rdfs(r[i], n);
    }
}
vector<int> getcover(int n, int m) {
    // solve min. vertex cover
    match(n);
    memset(chk, 0, sizeof(chk));
    for (int i = 0; i < n; i++)
        if (l[i] == -1)
            rdfs(i, n);
    vector<int> v;
    for (int i = 0; i < n; i++)
        if (!chk[i])
            v.push_back(i);
    for (int i = n; i < n + m; i++)
        if (chk[i])
            v.push_back(i);
    return v;
}
```

4 Max matching min cost

```
// numbered from 0. i -> mx[i]
const int V = 1000, INF = 1e9;
int g[V][V], mx[V], my[V], fx[V], fy[V], d[V], ar[V], tr[V], p;
int slack(int u, int v) {
    return g[u][v] - fx[u] - fy[v];
}
int augment(int s) {
    queue<int> q;
    q.push(s);
    fill_n(tr, p, -1);
    for(int i = 0; i < p; ++i) d[i] = slack(s, i), ar[i] = s;
    while(true) {
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            for(int v = 0; v < p; ++v) if(tr[v] == -1) {
                    int w = slack(u, v);
                    if(w == 0) {
                        tr[v] = u;
                        if(my[v] == -1) return v;
                        q.push(my[v]);
                    }
                    if(d[v] > w) d[v] = w, ar[v] = u;
                }
        }
        int delta = INF;
        for(int v = 0; v < p; ++v) if(tr[v] == -1) delta =
                    min(delta, d[v]);
        fx[s] += delta;
        for(int v = 0; v < p; ++v)
            if(tr[v] == -1) d[v] -= delta;
            else fx[my[v]] += delta, fy[v] -= delta;
        for(int v = 0; v < p; ++v) if(tr[v] == -1 && d[v] == 0) {
                tr[v] = ar[v];
                if(my[v] == -1) return v;
                q.push(my[v]);
            }
    }
}
void maxMatchMinCost() {
    fill_n(mx, p, -1);
    fill_n(my, p, -1);
    for(int i = 0; i < p; ++i) fx[i] = *min_element(g[i], g[i]+p);
    for(int s = 0; s < p; ++s) {
        int f = augment(s);
        while(f != -1) {
            int x = tr[f], nx = mx[x];
            mx[x] = f;
            my[f] = x;
            f = nx;
        }
    }
}
```

5 Ganeral Matching

```
// General matching on graph

const int maxv = 1000;
const int maxe = 50000;

// Index from 1
// Directed
struct EdmondsLawler {
    int n, E, start, finish, newRoot, qsize, adj[maxe], next[maxe], last[
        maxv], mat[maxv], que[maxv], dad[maxv], root[maxv];
    bool inque[maxv], inpath[maxv], inblossom[maxv];

    void init(int _n) {
        n = _n; E = 0;
        for(int x=1; x<=n; ++x) { last[x] = -1; mat[x] = 0; }
    }
    void add(int u, int v) {
        adj[E] = v; next[E] = last[u]; last[u] = E++;
    }
    int lca(int u, int v) {
        for(int x=1; x<=n; ++x) inpath[x] = false;
        while (true) {
            u = root[u];
            inpath[u] = true;
            if (u == start) break;
            u = dad[mat[u]];
        }
        while (true) {
            v = root[v];
            if (inpath[v]) break;
            v = dad[mat[v]];
        }
        return v;
    }
    void trace(int u) {
        while (root[u] != newRoot) {
            int v = mat[u];

            inblossom[root[u]] = true;
            inblossom[root[v]] = true;

            u = dad[v];
            if (root[u] != newRoot) dad[u] = v;
        }
    }
    void blossom(int u, int v) {
        for(int x=1; x<=n; ++x) inblossom[x] = false;

        newRoot = lca(u, v);
        trace(u); trace(v);

        if (root[u] != newRoot) dad[u] = v;
        if (root[v] != newRoot) dad[v] = u;

        for(int x=1; x<=n; ++x) if (inblossom[root[x]]) {
            root[x] = newRoot;
            if (!inque[x]) {
                inque[x] = true;
                que[qsize++] = x;
            }
        }
    }
    bool bfs() {
        for(int x=1; x<=n; ++x){
            inque[x] = false;
            dad[x] = 0;
            root[x] = x;
        }
        qsize = 0;
        que[qsize++] = start;
        inque[start] = true;
        finish = 0;

        for(int i=0; i<qsize; ++i) {
            int u = que[i];
            for (int e = last[u]; e != -1; e = next[e]) {
                int v = adj[e];
                if (root[v] != root[u] && v != mat[u]) {
                    if (v == start || (mat[v] > 0 && dad[mat[v]] > 0))
                        blossom(u, v);
                    else if (dad[v] == 0) {
                        dad[v] = u;
                        if (mat[v] > 0) que[qsize++] = mat[v];
                        else {
                            finish = v;
                            return true;
                        }
                    }
                }
            }
        }
        return false;
    }
    void enlarge() {
        int u = finish;
        while (u > 0) {
            int v = dad[u], x = mat[v];
            mat[v] = u;
            mat[u] = v;
            u = x;
        }
    }
    int maxmat() {
        for(int x=1; x<=n; ++x) if (mat[x] == 0) {
            start = x;
            if (bfs()) enlarge();
        }

        int ret = 0;
        for(int x=1; x<=n; ++x) if (mat[x] > x) ++ret;
        return ret;
    }
} edmonds;
```

6 Dinic MaxFlow

```
class DinicFlow {
private:
    vector<int> dist, head, work;
    vector<int> point, flow, capa, next;
    int n, m;

    bool bfs(int s, int t) {
        For(i, 1, n) dist[i] = -1;
        queue<int> q;
        dist[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = head[u]; i >= 0; i = next[i])
                if (flow[i] < capa[i] && dist[point[i]] < 0) {
                    dist[point[i]] = dist[u] + 1;
                    q.push(point[i]);
                }
        }
        return dist[t] >= 0;
    }

    int dfs(int s, int t, int f) {
        if (s == t) return f;
        for (int &i = work[s]; i >= 0; i = next[i])
            if (flow[i] < capa[i] && dist[point[i]] == dist[s] + 1) {
                int d = dfs(point[i], t, min(f, capa[i] - flow[i]));
                if (d > 0) {
                    flow[i] += d;
                    flow[i ^ 1] -= d;
                    return d;
                }
            }
        return 0;
    }

public:
    DinicFlow(int n = 0) {
        this->n = n;
        this->m = 0;
        dist.assign(n + 7, 0);
        head.assign(n + 7, -1);
        work.assign(n + 7, 0);
    }

    void addEdge(int u, int v, int c1, int c2 = 0) {
        point.push_back(v);
        capa.push_back(c1);
        flow.push_back(0);
        next.push_back(head[u]);
        head[u] = m++;
        point.push_back(u);
        capa.push_back(c2);
        flow.push_back(0);
        next.push_back(head[v]);
        head[v] = m++;
    }

    int maxFlow(int s, int t) {
        int totFlow = 0;
        while (bfs(s, t)) {
            For(i, 1, n) work[i] = head[i];
            while (true) {
                int d = dfs(s, t, cmax);
                if (d == 0) break;
                totFlow += d;
            }
        }
        return totFlow;
    }
```

7 Two sat

```
int n, m, g[maxn];
bool cx[maxn];
vector <int> listV, ke[maxn], K[maxn];

int cal(int x) {
    if (x%2 == 0) return x - 1;
    else return x + 1;
}

void add(int u, int v) {
    ke[u].pb(v);
    K[v].pb(u);
}

void dfs(int u) {
    cx[u] = true;
    for(int v : ke[u])
        if (!cx[v]) dfs(v);
    listV.pb(u);
}

void dfs(int u, int x) {
    g[u] = x;
    for(int v : K[u])
        if (g[v] == 0) dfs(v,x);
}

int main() {
    cin >> m >> n;
    n += n;
    For(i,1,m) {
        int u, v;
        cin >> u >> v;
        u *= 2;
        v *= 2;
        if (u < 0) u = cal(abs(u));
        if (v < 0) v = cal(abs(v));
        add(cal(u),v);
        add(cal(v),u);
    }
    listV.pb(0);
    For(i,1,n)
    if (!cx[i]) dfs(i);
    int ng = 0;
    Ford(i,n,1) {
        int u = listV[i];
        if (g[u] == 0) dfs(u,++ng);
    }
    for(int i = 2; i <= n; i += 2)
        if (g[i] == g[i-1]) NO;
    YES;
    vector <int> result;
    for(int i = 2; i <= n; i += 2)
        if (g[i] > g[i-1]) result.pb(i>>1);
}
```

8 Alternative Tree

```
int n, m, l, q, t, res, test,
    a[maxn], tin[maxn], tout[maxn], mark[maxn], terror[maxn], f[maxn][20];
vector<int> adj[maxn], _adj[maxn];
stack<int> stk;

void visit(const int &u) {
    tin[u] = ++t;
    for(int i = 1; i <= l; ++i) f[u][i] = f[f[u][i-1]][i-1];
    for(auto v : adj[u])
        if (v != f[u][0]) {
            f[v][0] = u;
            visit(v);
        }
    tout[u] = ++t;
}

bool anc(const int &u, const int &v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v) {
    if (anc(u,v)) return u;
    if (anc(v,u)) return v;
    for(int i = l; i >= 0; --i)
        if (!anc(f[u][i],v)) u = f[u][i];
    return f[u][0];
}

void query() {
    cin >> m;
    for(int i = 1; i <= m; ++i) {
        cin >> a[i];
        _adj[a[i]].clear();
        mark[a[i]] = test;
        terror[a[i]] = test;
    }
    sort(a+1,a+m+1,cmp);
    for(int i = 1; i < m; ++i) {
        int tmp = lca(a[i],a[i+1]);
        if (mark[tmp] < test) {
            mark[tmp] = test;
            a[++m] = tmp;
            _adj[tmp].clear();
        }
    }
    // sort theo tin
    sort(a+1,a+m+1,cmp);
    while (!stk.empty()) stk.pop();
    stk.push(a[1]);
    for(int i = 2; i <= m; ++i) {
        while (tout[stk.top()] < tout[a[i]]) stk.pop();
        _adj[stk.top()].push_back(a[i]);
        stk.push(a[i]);
    }
    res = 0;
    check(a[1]);
    cout << res << "\n";
}

int main() {
    l = log2(n);
    cin >> q;
    f[1][0] = 1;
    visit(1);
    for(test = 1; test <= q; ++test) query();
}
```

9 Aho Corasick

```
const int NODE = (int) 1e6 + 1;
const int NC = 26;

int nextNode[NODE][NC];
int chr[NODE];
int parent[NODE];
int prefix[NODE];
int numNodes;
set<int> match[NODE];

int getPrefix(int);

int go(int u, int c) {
    if (nextNode[u][c] != -1) return nextNode[u][c];
    if (u == 0) return 0;
    return nextNode[u][c] = go(getPrefix(u), c);
}

int getPrefix(int u) {
    if (prefix[u] != -1) return prefix[u];
    if (u == 0 || parent[u] == 0) return prefix[u] = 0;
    return prefix[u] = go(getPrefix(parent[u]), chr[u]);
}

void add(const string &s, int id) {
    int u = 0;
    for (int i = 0; i < (int) s.size(); ++i) {
        int c = s[i] - 'A';
        if (nextNode[u][c] == -1) {
            nextNode[u][c] = numNodes;
            fill(nextNode[numNodes], nextNode[numNodes] + NC, -1);
            chr[numNodes] = c;
            parent[numNodes] = u;
            prefix[numNodes] = -1;
            match[numNodes].clear();
            match[numNodes].insert(-1);
            ++numNodes;
        }
        u = nextNode[u][c];
    }
    match[u].insert(id);
}

set<int>& getMatch(int u) {
    if (match[u].count(-1) == 0) return match[u];
    const set<int> &foo = getMatch(getPrefix(u));
    match[u].insert(foo.begin(), foo.end());
    match[u].erase(-1);
    return match[u];
}

void init() {
    fill(nextNode[0], nextNode[0] + NC, -1);
    numNodes = 1;
}
```

10 Suffix Array

```
struct SuffixArray {
    const int L;
    string s;
    vector<vector<int> > P;
    vector<pair<pair<int,int>,int> > M;
    SuffixArray(const string &s) : L(s.length()), s(s), P(1,
                vector<int>(L, 0)), M(L) {
        for (int i = 0; i < L; i++) P[0][i] = int(s[i]);
        for (int skip = 1, lv = 1; skip < L; skip *= 2, lv++) {
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
```

```
                M[i] = make_pair(make_pair(P[lv-1][i], i + skip < L
                                           ? P[lv-1][i + skip] : -1000), i
                                               );
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[lv][M[i].se] = (i > 0 && M[i].fi == M[i-1].fi) ?
                                 P[lv][M[i-1].se] : i;
        }
    }
    vector<int> GetSuffixArray() {
        return P.back();
    }
// returns the length of the longest common prefix of s[i...L-1]
    and s[j...L-1]
    int LongestCommonPrefix(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
};
```

11 Suffix Array O(n)

```
#include <bits/stdc++.h>
#define FOR(i,a,b) for (int i=(a),_b=(b);i<=_b;i=i+1)
#define REP(i,n) for (int i=0,_n=(n);i<_n;i=i+1)
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x)>>(i))&1)
#define tget(i) BIT(t[(i) >> 3], (i) & 7)
#define tset(i, b) { if (b) t[(i) >> 3] |= MASK((i) & 7); else t[(i) >> 3]
     &= ~MASK((i) & 7); }
#define chr(i) (cs == sizeof(int) ? ((int *)s)[i] : ((unc *)s)[i])
#define isLMS(i) ((i) > 0 && tget(i) && !tget((i) - 1))

typedef unsigned char unc;
class SuffixArray {
public:
int *sa, *lcp, *rank, n;
unc *s;
void getbuckets(unc s[], vector<int> &bkt, int n, int k, int cs, bool
    end) {
    FOR(i, 0, k) bkt[i] = 0;
    REP(i, n) bkt[chr(i)]++;
    int sum = 0;
    FOR(i, 0, k) {
        sum += bkt[i];
        bkt[i] = end ? sum : sum - bkt[i];
    }
}
void inducesal(vector<unc> &t, int sa[], unc s[], vector<int> &bkt,
    int n, int k, int cs, bool end) {
    getbuckets(s, bkt, n, k, cs, end);
    REP(i, n) {
        int j = sa[i] - 1;
        if (j >= 0 && !tget(j)) sa[bkt[chr(j)]++] = j;
    }
}
void inducesas(vector<unc> &t, int sa[], unc s[], vector<int> &bkt,
    int n, int k, int cs, bool end) {
    getbuckets(s, bkt, n, k, cs, end);
    FORD(i, n - 1, 0) {
        int j = sa[i] - 1;
        if (j >= 0 && tget(j)) sa[--bkt[chr(j)]]=j;
    }
}
void build(unc s[], int sa[], int n, int k, int cs) {
    int j;
    vector<unc> t = vector<unc>(n / 8 + 1, 0);
    tset(n - 2, 0);
    tset(n - 1, 1);
    FORD(i, n - 3, 0) tset(i, chr(i) < chr(i+1) || (chr(i) == chr(i+1)
         && tget(i+1)));
    vector<int> bkt = vector<int> (k + 1, 0);
    getbuckets(s, bkt, n, k, cs, true);
    REP(i, n) sa[i] = -1;
    REP(i, n) if (isLMS(i)) sa[--bkt[chr(i)]] = i;
    inducesal(t, sa, s, bkt, n, k, cs, false);
    inducesas(t, sa, s, bkt, n, k, cs, true);
    bkt.clear();
    int n1 = 0;
```

```
        REP(i, n) if (isLMS(sa[i])) sa[n1++] = sa[i];
        FOR(i, n1, n - 1) sa[i] = -1;
        int name = 0;
        int prev = -1;
        REP(i, n1) {
            int pos = sa[i];
            bool diff = false;
            REP(d, n) {
                if (prev < 0 || chr(prev + d) != chr(pos + d) || tget(prev + d) != tget(pos + d)) {
                    diff = true;
                    break;
                }
                else if (d > 0 && (isLMS(prev + d) || isLMS(pos + d)))
                    break;
            }
            if (diff) {
                name++;
                prev = pos;
            }
            sa[n1 + pos / 2] = name - 1;
        }
        j = n - 1;
        FORD(i, n - 1, n1) if (sa[i] >= 0) sa[j--] = sa[i];
        int *sa1 = sa;
        int *s1 = sa + n - n1;
        if (name < n1) build((unc *)s1, sa1, n1, name-1, sizeof(int));
        else REP(i, n1) sa1[s1[i]] = i;
        bkt.assign(k + 1, 0);
        getbuckets(s, bkt, n, k, cs, true);
        j = 0;
        REP(i, n) if (isLMS(i)) s1[j++] = i;
        REP(i, n1) sa1[i] = s1[sa1[i]];
        FOR(i, n1, n - 1) sa[i] = -1;
        FORD(i, n1 - 1, 0) {
            j = sa[i];
            sa[i] = -1;
            sa[--bkt[chr(j)]] = j;
        }
        inducesal(t, sa, s, bkt, n, k, cs, false);
        inducesas(t, sa, s, bkt, n, k, cs, true);
        bkt.clear();
        t.clear();
    }
```

```
    void calc_lcp(void) {
        FOR(i,1,n) rank[sa[i]] = i;
        int h = 0;
        REP(i, n) if (rank[i] < n) {
            int j = sa[rank[i] + 1];
            while (s[i + h] == s[j + h]) h++;
            lcp[rank[i]] = h;
            if (h > 0) h--;
        }
    }
    SuffixArray() {
        n = 0;
        sa = lcp = rank = NULL;
        s=NULL;
    }
    SuffixArray(string ss) {
        n = ss.size();
        sa = new int[n + 7];
        lcp = new int [n + 7];
        rank = new int [n + 7];
        s = (unc *)ss.c_str();
        build(s, sa, n + 1, 256, sizeof(char));
        calc_lcp();
    }
};

//Sorted suffices are SA[1] to SA[N]. The values of SA[1], SA[2], ..., SA[N] are 0, 1, ..., N - 1
//The longest common prefix of SA[i] and SA[i + 1] is LCP[i]

int main(void) {
    string s = "mississippi";
    SuffixArray suffixArray(s);
    FOR(i, 1, 11) printf("%d %s %d\n", suffixArray.sa[i], s.substr(suffixArray.sa[i]).c_str(), suffixArray.lcp[i]);
}
```

12 Manacher

```
void manacher() {
    memset(p,0,sizeof p);
    int center = 0, right = 0,mi;
    for (int i = 1; i < n; i++) {
        mi = 2 * center - i;
        if (right > i) p[i] = min(right - i, p[mi]);
        while (a[i+(1+p[i])] == a[i-(1+p[i])]) p[i]++;
        //printf("%d:%d\n",i,p[i]);
        if (i + p[i] > right) {
            right = i+p[i];
            center = i;
        }
    }
}
```

13 Convex Hull

```
struct Point {
    long long x, y;
    bool operator < (const Point &v) const {
        return x == v.x ? y < v.y : x < v.x;
    }
    long long cross(const Point &p, const Point &q) const {
        return (p.x - x) * (q.y - y) - (p.y - y) * (q.x - x);
    }
};
vector<Point> convexHull(vector<Point> p) {
    sort(p.begin(), p.end());
    int k = 0, n = p.size();
    vector<Point> poly (2 * n);
    for(int i = 0; i < n; ++i) {
        while(k >= 2 && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    for(int i = n-2, t = k+1; i >= 0; --i) {
        while(k >= t && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    poly.resize(min(n, max(0, k - 1)));
    return poly;
}
```

14 Geometry's tricks

```
const double eps = 1e-9;
bool equal(const double &x, const double &y) {
    return fabs(x - y) <
           eps;
}
struct Point {
    double x, y;
    Point(double x = 0, double y = 0): x(x), y(y) {}
    Point operator + (const Point &p) const {
        return {x + p.x, y +
                 p.y
                };
    }
    Point operator - (const Point &p) const {
        return {x - p.x, y -
                 p.y
                };
    }
    Point operator * (double t) const {
        return {x * t, y * t};
    }
    double operator * (const Point &p) const {
        return x * p.x + y *
                p.y;
    }
    double operator % (const Point &p) const {
        return x * p.y - y *
                p.x;
    }
    bool operator == (const Point &p) const {
        return equal(x, p.x)
                && equal(y, p.y);
    }
    double operator ~ () const {
        return sqrt(*this **this);
    }
};
struct Comparator {
    Point a, b;
    Comparator(Point a, Point b): a(a), b(b) {}
    bool operator () (const Point &p, const Point &q) {
        return (p-a) * (b-a) < (q-a) * (b-a);
    }
};
bool between(double x, double l, double r) {
    if (l > r) swap(l, r);
    return x + eps > l && x - eps < r;
}
bool inside(Point q, const vector<Point> &p) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = i + 1 < n ? i + 1 : 0;
        if (fabs((q - p[i]) % (p[j] - p[i])) > eps) continue;
        if ((q - p[i]) * (p[j] - p[i]) < -eps) continue;
        if ((q - p[j]) * (p[i] - p[j]) < -eps) continue;
        return true;
    }
    int fl = 0;
    for (int i = 0; i < n; i++) {
        int j = i + 1 < n ? i + 1 : 0;
        Point a = p[i], b = p[j];
        if (equal(a.x, b.x)) continue;
        if (a.x > b.x) swap(a, b);
        if (q.x < a.x - eps) continue;
        if (q.x > b.x - eps) continue;
        if ((q - a) % (b - a) > 0) fl ^= 1;
    }
    return fl;
}
void intersect(Point p, Point q, Point a, Point b, vector<Point>
               &ints) {
    double na = (a - p) % (q - p), nb = (b - p) % (q - p);
    if (na * nb > eps) return;
    if (equal(na, nb)) return;
    ints.push_back(a + (b - a) * (na / (na - nb)));
}
void intersectCircleLine() {
    double r, a, b, c;
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) puts ("no points");
    else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
        puts ("1 point");
        cout << x0 << ' ' << y0 << '\n';
    } else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        puts ("2 points");
        cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
    }
}
```

15 FFT

```
const double PI = acos(-1.0);
typedef complex<double> Complex;
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x) >> (i)) & 1)
#define LOG 17
Complex fftRoot[MASK(LOG)], invRoot[MASK(LOG)];
#define REP(i, n) for (int i = 0, _n = (n); i < _n; i = i + 1)
void initFFT(void) {
    REP(i, MASK(LOG)) {
        double alpha = 2 * PI / MASK(LOG) * i;
        fftRoot[i] = Complex(cos(alpha), sin(alpha));
        invRoot[i] = Complex(cos(-alpha), sin(-alpha));
    }
}
unsigned roundUp(unsigned v) {
    --v;
    REP(i, 5) v |= v >> MASK(i);
    return v + 1;
}
int reverse(int num, int lg) {
    int res = 0;
    REP(i, lg) if (BIT(num, i)) res |= MASK(lg - i - 1);
    return res;
}
vector<Complex> fft(vector<Complex> a, bool invert) {
    int n = a.size(), lg = 0;
    while (MASK(lg) < n) lg++;
    vector<Complex> roots(n);
    REP(i, n) roots[i] = invert ? invRoot[MASK(LOG) / n * i] :
                        fftRoot[MASK(LOG) / n * i];
    REP(i, n) {
        int rev = reverse(i, lg);
        if (i < rev) swap(a[i], a[rev]);
    }
    for (int len = 2; len <= n; len <<= 1)
```

```
        for (int i = 0; i < n; i += len)
            for (int j = 0; j < (len >> 1); j++) {
                Complex u = a[ i + j], v = a[i + j + (len >> 1)] *
                                           roots[n / len * j];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    if (invert) REP(i, n) a[i] /= n;
    return a;
}
vector<long long> multiply(const vector<int> &a, const vector<int>
                           &b) {
    int n = roundUp(size(a) + size(b) - 1);
    vector<Complex> pa (n), pb (n);
    for(int i = 0; i < size(a); ++i) pa[i] = a[i];
    for(int i = 0; i < size(b); ++i) pb[i] = b[i];
    pa = fft(pa, false);
    pb = fft(pb, false);
    for(int i = 0; i < n; ++i) pa[i] *= pb[i];
    pa = fft(pa, true);
    vector<long long> res (n);
    for(int i = 0; i < n; ++i) res[i] = round(real(pa[i]));
    return res;
}
```

16 NTT

```
const int MODULO = 998244353;
const int ROOT = 3; // Primitive root
void fft(vector<int> &a, bool invert) {
    int n = a.size();
    assert((n & (n - 1)) == 0);
    int lg = __builtin_ctz(n);
    for (int i = 0; i < n; ++i) {
        int j = 0;
        for (int k = 0; k < lg; ++k) if ((i&1<<k)!=0) j |= 1 <<
                        (lg-k-1);
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len *= 2) {
        int wlen = power(ROOT, (MODULO - 1) / len);
        if (invert) wlen = inverse(wlen);
        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; ++j) {
                int u = a[i + j];
                int v = 1LL * a[i + j + len / 2] * w % MODULO;
                a[i + j] = (u + v) % MODULO;
                a[i + j + len / 2] = (u - v + MODULO) % MODULO;
                w = 1LL * w * wlen % MODULO;
            }
        }
    }
    if (invert) {
        int mul = inverse(n);
        for (auto &x : a) x = 1LL * x * mul % MODULO;
    }
}
    998244353 = 119 * 2^23 + 1. Primitive root: 3.
    985661441 = 235 * 2^22 + 1. Primitive root: 3.
    1012924417 = 483 * 2^21 + 1. Primitive root: 5
```

17 Primitive Root

```
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i=2; i*i<=n; ++i) if (n % i == 0) {
        fact.push_back(i);
        while (n % i == 0) n /= i;
    }
    if (n > 1) fact.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

18 Range Prime Counting

```
// Primes up to 10^12 can be counted in ~1 second.
const int MAXN = 1000005; // MAXN is the maximum value of sqrt(N) +
2
bool prime[MAXN];
int prec[MAXN];
vector<int> P;
void init() {
    prime[2] = true;
    for (int i = 3; i < MAXN; i += 2) prime[i] = true;
    for (int i = 3; i*i < MAXN; i += 2) {
        if (prime[i]) {
            for (int j = i*i; j < MAXN; j += i+i) prime[j] = false;
        }
    }
    for(int i=1; i<MAXN; i++) {
        if (prime[i]) P.push_back(i);
        prec[i] = prec[i-1] + prime[i];
    }
}
lint rec(lint N, int K) {
    if (N <= 1 || K < 0) return 0;
    if (N <= P[K]) return N-1;
    if (N < MAXN && 1ll * P[K]*P[K] > N) return N-1 - prec[N] +
                prec[P[K]];
    const int LIM = 250;
    static int memo[LIM*LIM][LIM];
    bool ok = N < LIM*LIM;
    if (ok && memo[N][K]) return memo[N][K];
    lint ret = N/P[K] - rec(N/P[K], K-1) + rec(N, K-1);
    if (ok) memo[N][K] = ret;
    return ret;
}
lint count_primes(lint N) { //less than or equal to
    if (N < MAXN) return prec[N];
    int K = prec[(int)sqrt(N) + 1];
    return N-1 - rec(N, K) + prec[P[K]];
}
```

19 Knight's shortest path

```
int KSP(int x,int y) {
    if (x < y) swap(x, y);
    if (x == 1 && y == 0) return 3;
    if (x == 2 && y == 2) return 4;
    int d = x - y;
    if (y > d) return 2*((y-d+2)/3)+d;
    return d-2*((d-y)/4);
}
```