Mục lục

1 Java Fast Input

```
class InputReader {
    private final BufferedReader reader;
    private StringTokenizer tokenizer;

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

    public String nextLine() {
        try {
            return reader.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            tokenizer = new StringTokenizer(nextLine());
        }
        return tokenizer.nextToken();
    }

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

2 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
}
```

3 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;
}
```

4 Hopcroft Karp Max Matching algorithm

```
// Worse Case: E * sqrt(v)
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;
}
```

5 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;
        }
    }
}
```

6 Ganeral Matching

```
class MatchingGraph {
public:
    vector <vector<int> > adj;
    vector <bool> blossom;
    vector <int> parent;
    vector <int> base;
    vector <int> match;
    int n;
    MatchingGraph() {
        n = 0;
    }
    void addEdge(int x, int y) {
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    void clearGraph() {
        int i;
        for (i=0; i<SZ(adj); ++i)
            adj[i].clear();
        fill(blossom.begin(),blossom.end(),false);
        fill(parent.begin(),parent.end(),-1);
        for (i=0; i<n; ++i)
            base[i] = i;
        for (i=0; i<n; ++i)
            match[i] = -1;
    }
    void setN(int newn) {
        n = newn;
        adj.resize(n);
        blossom.resize(n);
        base.resize(n);
        match.resize(n);
        parent.resize(n);
        clearGraph();
    }
    int lca(int x, int y) {
        vector <bool> fy;
        fy.resize(n);
        fill(fy.begin(),fy.end(),false);
        while (true) {
            x = base[x];
            fy[x] = true;
            if (match[x] == -1)
                break;
            x = parent[match[x]];
        }
        while (true) {
            y = base[y];
            if (fy[y])
                return y;
            y = parent[match[y]];
        }
        return -1;
    }
    void path(int now, int child, int curbase) {
        while (base[now] != curbase) {
            blossom[base[now]] = blossom[base[match[now]]] = true;
            parent[now] = child;
            child = match[now];
            now = parent[match[now]];
        }
    }
    int augmentPath(int x) {
        int i,j;
        for (i=0; i<n; ++i)
            base[i] = i;
        for (i=0; i<n; ++i)
            parent[i] = -1;
        queue <int> bfs;
        vector <bool> sudah;
        sudah.resize(n);
        fill(sudah.begin(),sudah.end(),false);
        sudah[x] = true;
        bfs.push(x);
        while (!bfs.empty()) {
            int now = bfs.front();
            bfs.pop();
            for (i=0; i<SZ(adj[now]); ++i) {
                int next = adj[now][i];
                if (base[next]==base[now] || match[next] == now);
                else if (next == x || (match[next]!=-1 &&
                                       parent[match[next]]!=-1)) {
                    int curbase = lca(now,next);
                    fill(blossom.begin(),blossom.end(),false);
                    path(now,next,curbase);
                    path(next,now,curbase);
                    for (j = 0; j < n; ++j)
                        if (blossom[j]) {
                            base[j] = curbase;
                            if (!sudah[j]) {
                                sudah[j] = true;
                                bfs.push(j);
                            }
                        }
                } else if (parent[next]==-1) {
                    parent[next] = now;
                    if (match[next] == -1)
                        return next;
                    sudah[match[next]] = true;
                    bfs.push(match[next]);
                }
            }
        }
        return -1;
    }
    int edmondsMatch() {
        int i;
        int res = 0;
        for (i=0; i<n; ++i) {
            if (match[i]==-1) {
                int x = augmentPath(i);
                while (x>=0) {
                    int p = parent[x];
                    int pp = match[p];
                    match[x] = p;
                    match[p] = x;
                    x = pp;
                }
            }
        }
        for (i=0; i<n; ++i)
            if (match[i]!=-1)
                ++res;
        return res >> 1;
    }
};
```

7 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;
    }
```

8 Mincost MaxFlow SPFA

```
 Min Cost Max Flow - SPFA
 Index from 0
 edges cap changed during find flow
 Lots of double comparison --> likely to fail for double
 Example:
 MinCostFlow mcf(n);
 mcf.addEdge(u, v, cap, cost);
 cout << mcf.minCostFlow() << endl;


template<class Flow=int, class Cost=int>
struct MinCostFlow {
    const Flow INF_FLOW = 1000111000;
    const Cost INF_COST = 1000111000111000LL;

    int n, t, S, T;
    Flow totalFlow;
    Cost totalCost;
    vector<int> last, visited;
    vector<Cost> dis;
    struct Edge {
        int to;
        Flow cap;
        Cost cost;
        int next;
        Edge(int to, Flow cap, Cost cost, int next) :
                to(to), cap(cap), cost(cost), next(next) {}
    };
    vector<Edge> edges;

    MinCostFlow(int n) : n(n), t(0), totalFlow(0), totalCost(0), last(n,
        -1), visited(n, 0), dis(n, 0) {
        edges.clear();
    }

    int addEdge(int from, int to, Flow cap, Cost cost) {
        edges.push_back(Edge(to, cap, cost, last[from]));
        last[from] = t++;
        edges.push_back(Edge(from, 0, -cost, last[to]));
        last[to] = t++;
        return t - 2;
    }

    pair<Flow, Cost> minCostFlow(int _S, int _T) {
        S = _S; T = _T;
        SPFA();
        while (1) {
            while (1) {
                REP(i,n) visited[i] = 0;
                if (!findFlow(S, INF_FLOW)) break;
            }
            if (!modifyLabel()) break;
        }
        return make_pair(totalFlow, totalCost);
    }

private:
    void SPFA() {
        REP(i,n) dis[i] = INF_COST;
        priority_queue< pair<Cost,int> > Q;
        Q.push(make_pair(dis[S]=0, S));
        while (!Q.empty()) {
            int x = Q.top().second;
            Cost d = -Q.top().first;
            Q.pop();
            // For double: dis[x] > d + EPS
            if (dis[x] != d) continue;
            for(int it = last[x]; it >= 0; it = edges[it].next)
                if (edges[it].cap > 0 && dis[edges[it].to] > d + edges[it].cost)
                    Q.push(make_pair(-(dis[edges[it].to] = d + edges[it].cost), edges[it].to));
        }
        Cost disT = dis[T]; REP(i,n) dis[i] = disT - dis[i];
    }

    Flow findFlow(int x, Flow flow) {
        if (x == T) {
            totalCost += dis[S] * flow;
            totalFlow += flow;
            return flow;
        }
        visited[x] = 1;
        Flow now = flow;
        for(int it = last[x]; it >= 0; it = edges[it].next)
            // For double: fabs(dis[edges[it].to] + edges[it].cost - dis[x]) < EPS
            if (edges[it].cap && !visited[edges[it].to] && dis[edges[it].to] + edges[it].cost == dis[x]) {
                Flow tmp = findFlow(edges[it].to, min(now, edges[it].cap));
                edges[it].cap -= tmp;
                edges[it ^ 1].cap += tmp;
                now -= tmp;
                if (!now) break;
            }
        return flow - now;
    }

    bool modifyLabel() {
        Cost d = INF_COST;
        REP(i,n) if (visited[i])
            for(int it = last[i]; it >= 0; it = edges[it].next)
                if (edges[it].cap && !visited[edges[it].to])
                    d = min(d, dis[edges[it].to] + edges[it].cost - dis[i]);

        // For double: if (d > INF_COST / 10)     INF_COST = 1e20
        if (d == INF_COST) return false;
        REP(i,n) if (visited[i])
            dis[i] += d;
        return true;
    }
};
```

9 Upper Lower

```
- For each edge in original flow:
  - Add edge with cap = upper bound - lower bound.
- Add source s, sink t.
- Let M[v] = (sum of lowerbounds of ingoing edges to v) - (sum of lower bounds of outgoing edges from v).
- For all v, if M[v] > 0, add (s, v, M), else add (v, t, -M).
- If all outgoing edges from S are full --> feasible flow exists, it is flow + lower bounds.

Feasible flow in network with upper + lower constraint, with source & sink:
- Add edge (t, s) with capacity [0, INF].
- Check feasible in network without source & sink.
```

```
Max flow with both upper + lower constraints, source s, sink t: add edge (
    t, s, +INF).
- Binary search lower bound, check whether feasible flow exists WITHOUT
    source / sink
```

10 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);
}
```

11 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();
}
```

12 Max Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg ++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};
```

13 Euler Path

```
 NOTES:
 - When choosing starting vertex (for calling find_path), make sure deg[
     start] > 0.
 - If find Euler path, starting vertex must have odd degree.
 - Check no solution: SZ(path) == nEdge + 1.

 If directed:
 - Edge --> int
 - add_edge(int a, int b) { adj[a].push_back(b); }
 - Check for no solution:
 - - for all u, |in_deg[u] - out_deg[u]| <= 1
 - - At most 1 vertex with in_deg[u] - out_deg[u] = 1
 - - At most 1 vertex with out_deg[u] - in_deg[u] = 1 (start vertex)
 - - BFS from start vertex, all vertices u with out_deg[u] > 0 must be
     visited
struct Edge {
    int to;
    list<Edge>::iterator rev;

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

const int MN = 100111;
list<Edge> adj[MN];
vector<int> path; // our result

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

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

14 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;
}
```

15 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;
    }
};
```

16 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]);
}
```

17 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;
        }
    }
}
```

18 DP knuth

```
 http://codeforces.com/blog/entry/8219
 Original Recurrence:
   dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]   for k = i+1..j-1
 Necessary & Sufficient Conditions:
   A[i][j-1] <= A[i][j] <= A[i+1][j]
   with A[i][j] = smallest k that gives optimal answer
 Also applicable if the following conditions are met:
   1. C[a][c] + C[b][d] <= C[a][d] + C[b][c] (quadrangle inequality)
   2. C[b][c] <= C[a][d]                     (monotonicity)
   for all a <= b <= c <= d
 To use:
   Calculate dp[i][i] and A[i][i]
   FOR(len = 1..n-1)
     FOR(i = 1..n-len) {
       j = i + len
       FOR(k = A[i][j-1]..A[i+1][j])
         update(dp[i][j])
     }

// OPTCUT
#include "../template.h"

const int MN = 2011;
int a[MN], dp[MN][MN], C[MN][MN], A[MN][MN];
int n;

void solve() {
    cin >> n; FOR(i,1,n) { cin >> a[i]; a[i] += a[i-1]; }
    FOR(i,1,n) FOR(j,i,n) C[i][j] = a[j] - a[i-1];

    FOR(i,1,n) dp[i][i] = 0, A[i][i] = i;

    FOR(len,1,n-1)
        FOR(i,1,n-len) {
            int j = i + len;
            dp[i][j] = 2000111000;
            FOR(k,A[i][j-1],A[i+1][j]) {
                int cur = dp[i][k-1] + dp[k][j] + C[i][j];
                if (cur < dp[i][j]) {
                    dp[i][j] = cur;
                    A[i][j] = k;
                }
            }
        }
    cout << dp[1][n] << endl;
}
```

19 DP divide conquer

```
 http://codeforces.com/blog/entry/8219
 Divide and conquer optimization:
 Original Recurrence
   dp[i][j] = min(dp[i-1][k] + C[k][j]) for k < j
 Sufficient condition:
   A[i][j] <= A[i][j+1]
   where A[i][j] = smallest k that gives optimal answer
 How to use:
   // compute i-th row of dp from L to R. optL <= A[i][L] <= A[i][R] <= optR
   compute(i, L, R, optL, optR)
       1. special case L == R
       2. let M = (L + R) / 2. Calculate dp[i][M] and opt[i][M] using O(optR - optL + 1)
       3. compute(i, L, M-1, optL, opt[i][M])
       4. compute(i, M+1, R, opt[i][M], optR)


const int MN = 4011;
const int inf = 1000111000;
int n, k, cost[MN][MN], dp[811][MN];

inline int getCost(int i, int j) {
    return cost[j][j] - cost[j][i-1] - cost[i-1][j] + cost[i-1][i-1];
}

void compute(int i, int L, int R, int optL, int optR) {
    if (L > R) return ;

    int mid = (L + R) >> 1, savek = optL;
    dp[i][mid] = inf;
    FOR(k,optL,min(mid-1, optR)) {
        int cur = dp[i-1][k] + getCost(k+1, mid);
        if (cur < dp[i][mid]) {
            dp[i][mid] = cur;
            savek = k;
        }
    }
    compute(i, L, mid-1, optL, savek);
    compute(i, mid+1, R, savek, optR);
}

void solve() {
    cin >> n >> k;
    FOR(i,1,n) FOR(j,1,n) {
        cin >> cost[i][j];
        cost[i][j] = cost[i-1][j] + cost[i][j-1] - cost[i-1][j-1] + cost[i][j];
    }
    dp[0][0] = 0;
    FOR(i,1,n) dp[0][i] = inf;

    FOR(i,1,k) {
        compute(i, 1, n, 0, n);
    }
    cout << dp[k][n] / 2 << endl;
}
```

20 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;
}
```

21 Geometry's tricks

```
// hash func overload
struct point{
int x, y;
bool operator==(const point &p)const{ return x == p.x && y == p.y; }
};
struct hasher {
size_t operator()(const point &p)const{ return p.x * 2 + p.y * 3; }
};
unordered_map<point, int, hasher> hsh;
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';
    }
}
```

22 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;
}
```

23 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
```

24 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;
}
```

25 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]];
}
```

26 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);
}
```

27 Extended Euclid

```
 Gia su ket qua la (x0. y0), ho nghiem la  (x_0 + k * b / d, y_0 - k * a/d
    )
 Phuong trinh ax + by = d co nghiem khi va chi khi d chia het cho gcd(a, b
    )
 a x + b y = gcd(a, b)
int extgcd(int a, int b, int &x, int &y) {
    int g = a; x = 1; y = 0;
    if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
    return g;
}
```

28 Factorial Mod

```
int factmod (int n, int p) { // n!, excluding p^k of course
    int res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
        for (int i=2; i<=n%p; ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}
```

29 Sqrt Mod

```
// Jacobi Symbol (m/n), m,n >= 0  and n is odd
```

```
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
    if (a == 0) return m == 1 ? 1 : 0;
    if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
    else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
int invMod(int a, int m) {
    int x, y;
    if (extgcd(a, m, x, y) == 1) return (x + m) % m;
    else                                 return 0; // unsolvable
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
    int S, Q, W, i, m = invMod(n, p);
    for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
    do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
    for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
        int z = R * R * m % p;
        for (i = 0; i < S && z % p != 1; z *= z, ++i);
        if (i == 0) return R;
        R = (R * powMod(V, 1 << (S-i-1), p)) % p;
    }
}
int powMod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}
```

30 Interval line

```
// template Interval line Min
#define mid ((lo + hi)>>1)

class Line {
public:
    ll a, b;

    Line (ll x = cmax, ll y = cmax) {
        a = x, b = y;
    }

    ll get(int x) {
        return 1LL * val[x] * a + b;
    }
};

const Line oo = Line(cmax, cmax);


class ILTree {
    int m;
public:
    Line t[maxn*4];

    ILTree(int last = 200000) {
        m = last;
        init(1,1,m);
    }

    void init(int i, int lo, int hi) {
        t[i] = oo;
        if (lo == hi) return;
        init(i * 2,lo,mid);
        init(i * 2 + 1, mid + 1, hi);
    }

    void update(int i, int lo, int hi, int l, int r, Line d) {
        if (l > hi || r < lo) return;
        if (lo >= l && hi <= r) {
            // t[i] hoan toan nam duoi d
            if (t[i].get(lo) <= d.get(lo) && t[i].get(hi) <= d.get(hi))
                return;
            //t[i] hoan toan nam tren d thi cap nhap t[i] = d
            if (t[i].get(lo) >= d.get(lo) && t[i].get(hi) >= d.get(hi)) {
                t[i] = d;
                return;
            }
            //nua dau cua d tot hon
            if (t[i].get(lo) >= d.get(lo) && t[i].get(mid) >= d.get(mid))
                {
                update(i * 2 + 1, mid + 1, hi, l, r, t[i]);
                t[i] = d;
                return;
            }
            // nua dau cua t[i] tot hon
            if (t[i].get(lo) <= d.get(lo) && t[i].get(mid) <= d.get(mid)) {
                update(i * 2 + 1, mid + 1, hi, l, r, d);
                return;
            }
            // nua sau cua d tot hon
            if (t[i].get(mid + 1) >= d.get(mid + 1) && t[i].get(hi) >= d.get(hi)) {
                update(i * 2, lo, mid, l, r, t[i]);
                t[i] = d;
                return;
            }
            // nua sau cua t[i] tot hon
            if (t[i].get(mid + 1) <= d.get(mid + 1) && t[i].get(hi) <= d.get(hi)) {
                update(i * 2, lo, mid, l, r, d);
                return;
            }
        }
        update(i * 2, lo, mid, l, r, d);
        update(i * 2 + 1, mid + 1, hi, l, r, d);
    }

    ll get(int i, int lo, int hi, int pos) {
        if (lo > pos || hi < pos) return llmax;
        ll res = t[i].get(pos);
        if (lo == hi) return res;
        res = min(res, get(i * 2, lo, mid, pos));
        res = min(res, get(i * 2 + 1, mid + 1, hi, pos));
        return res;
    }
};

#undef mid
#undef oo
```

31 BIT 2D

```
class query{
public:
        int x, y, id, type;
        void next_x(){
                if(type==1) x+=x&-x;
                else x-=x&-x;
        }
        void next_y(){
                if(type==1) y+=y&-y;
                else y-=y&-y;
        }
};
const int maxqsize=100000;
vector <query> q;
deque <int> start[maxqsize+1];
vector <query> temp[n+1];
class node{
public:
        int value, y;
        int next, prev;
        node(int y){
                this->y=y;
                value=0;
                next=prev=-1;
        }
};
vector <node> ft[n+1];
int done[n+1];
int pos[n+1];
int main() {
        // your code goes here
        for(query a: q){
                while((a.x)&&(a.x<=n)){
                        temp[a.x].pb(a);
                        a.next_x();
                }
        }
        FOR(i, 1, n){
                for(query a: temp[i]){
                        while((a.y)&&(a.y<=n)){
                                if(done[a.y]!=i){
                                        done[a.y]=i;
                                        pos[a.y]=ft[i].size();
                                        ft[i].pb(node(a.y));
                                }
                                a.next_y();
                        }
                }
                for(node &a: ft[i]){
                        int yy=a.y+(a.y&(-a.y));
                        if(yy<=n){
                                if(done[yy]==i) a.next=pos[yy];
                        }
                        yy=a.y-(a.y&(-a.y));
                        if(done[yy]==i) a.prev=pos[yy];
                }
                for(query a: temp[i]){
                        if(a.type==1)///up
                                start[a.id].pb(pos[a.y]);
                        else start[a.id].push_front(pos[a.y]);
                }
        }
        return 0;

}
```

32 Heavy-Light Decomposition

```
chainHead [c] dinh dau cua chuoi c
chainInd [u] chuoi ma dinh u nam trong

void hld(int u) {

    //Neu chuoi hien tai chua co dinh dau dinh gan goc nhat thi dat u lam dinh dau cua no
        if (chainHead[nChain] == 0) chainHead[nChain] = u;

    //Gan chuoi hien tai cho u
        chainInd[u] = nChain;

    //Giai thich ben duoi
        posInBase[u] = ++nBase;

    // Bien luu dinh con dac biet
        int mxVtx = -1;

    // Tim dinh con dac biet trong so nhung dinh con cua u
        for (int i = 0; i < adj[u].size(); i++) {
                int v = adj[u][i];
                if (v != parent[u]) {
                        if (mxVtx == -1 || nChild[v] > nChild[mxVtx]) {
                                mxVtx = v;
                        }
                }
        }

    //Neu tim ra dinh con dac biet (u khong phai la dinh la) thi di chuyen den dinh do
        if (mxVtx > -1)
                hld(mxVtx);

    // Sau khi di het mot chuoi thi tang nChain len va bat dau mot chuoi moi
        for (int i = 0; i < adj[u].size(); i++) {
                int v = adj[u][i];
                if (v != parent[u] && v != mxVtx) {
                        nChain++;
                        hld(v);
                }
        }
}


void update(int u, int a) {
    // uchain chuoi hien tai cua u
    // achain chuoi hien tai cua a
     int uchain = chainInd[u], achain = chainInd[a];

     while (1) {
        // Neu u va a cung nam tren mot chuoi thi update doan tu u den a va ket thuc
          if (uchain == achain) {
               updateIntervalTree(..., posInBase[a], posInBase[u], ...);
               break;
          }
        // Neu u va a khong nam tren cung mot chuoi thi update doan tu u den dinh dau cua chuoi hien tai
          updateIntervalTree(..., posInBase[chainHead[uchain]], posInBase[u], ...);

        // Nhay len dinh cha cua dinh dau hien tai
          u = parent[chainHead[uchain]];
          uchain = chainInd[u];
    }
}
```

33 Splay Tree

```
/**
 * Problem' s query: insert a[i] after a[j]
 */
struct node {
    node *par, *left, *right;
    int value, cnt;
};
int n, Q, tree[N];
node *root, *nilT;
void calc(node *x) {
    if (x != nilT)
        x->cnt = x->left->cnt + x->right->cnt + 1;
}

node *FindPosition(node *cur, int pos) {
    node *x = cur;
    while (x != nilT) {
        int ord = x->left->cnt + 1;
        if (ord == pos)
            return x;
        if (ord > pos)
            x = x->left;
        else
            x = x->right, pos -= ord;
    }
    return nilT;
}
void SetL(node *parent, node *child) {
    if (child != nilT)
        child->par = parent;
    if (parent != nilT)
        parent->left = child;
}
void SetR(node *parent, node *child) {
    if (child != nilT)
        child->par = parent;
    if (parent != nilT)
        parent->right = child;
}
void Uptree(node *x) {
    if (x == root)
        return;
    node *y = x->par, *z = y->par;
    if (y->left == x) {
        SetL(y, x->right);
        SetR(x, y);
    } else {
        SetR(y, x->left);
        SetL(x, y);
    }
    if (z == nilT)
        root = x, x->par = nilT;
    else if (z->left == y)
        SetL(z, x);
    else
        SetR(z, x);
    calc(y);
    calc(x);
}
void Splay(node *x) {
    while (1) {
        node *y = x->par;
        if (y == nilT)
            break;
        node *z = y->par;
        if (z != nilT) {
            if ((y == z->left) == (y->left == x))
                Uptree(y);
            else
                Uptree(x);
        }
        Uptree(x);
    }
    root = x;
}
void Split(node *r, int i, node *&r1, node *&r2) {
    if (i == 0) {
        r1 = nilT, r2 = r;
        return;
    }
    node *x = FindPosition(r, i);
    Splay(x);
    r2 = x->right;
    r1 = x;
    r1->right = nilT;
    calc(r1), calc(r2);
}
node *Join(node *r1, node *r2) {
    if (r1 == nilT)
        return r2;
    while (r1->right != nilT)
        r1 = r1->right;
    Splay(r1);
    SetR(r1, r2);
    calc(r1);
    return r1;
}
void Insert(int i, int val) {
    node *x = new node;
    x->value = val;
    x->par = nilT;
    node *r1, *r2;
    Split(root, i - 1, r1, r2);
    SetL(x, r1), SetR(x, r2);
    calc(x);
    root = x;
}

void Delete(int i) {
    node *x = FindPosition(root, i);
    Splay(x);
    node *r1 = x->left, *r2 = x->right;
    r1->par = nilT; r2->par = nilT;
    delete x;
    root = Join(r1, r2);
}
vector<int> arr;
void GetArray(node *x) {
    if (x == nilT)
        return;
    GetArray(x->left);
    arr.push_back(x->value);
    GetArray(x->right);
    delete x;
}
```

```
int main() {
    nilT = new node;
    nilT->value = nilT->cnt = 0;
    nilT->left = nilT->right = nilT->par = nilT;
    root = new node;
    root->value = root->cnt = 1;
    root->left = root->right = root->par = nilT;
    cin >> n >> Q;
    for (int i = 2; i <= n; i++)
        Insert(i, i);
    for (int i = 1; i <= Q; i++) {
        cin >> u >> v;
        int val = FindPosition(root, u)->value;
        Delete(u);
        Insert(v, val);
    }
    GetArray(root);
    int ans = 0;
    for (int i = 0; i < n; i++) {
        int F = 1;
        for (int x = arr[i]; x; x -= x & (-x))
            F = max(F, tree[x] + 1);
        ans = max(ans, F);
        for (int x = arr[i]; x <= n; x += x & (-x))
            tree[x] = max(tree[x], F);
    }
    cout << n - ans << endl;
}
```

/**

Geometry - Tungluu18

*/

Duong tron di qua 3 diem cho truoc Let A = (0, 0) centers are $Cy(Bx^2+By^2)-By(Cx^2+Cy^2)/D$ and $(Bx(Cx^2+Cy^2)-Cx(Bx^2+By^2))/D$ where $D=2(BxCy-ByCx)$.

```
Diem trong tam giac
bool isInside(const Vector &P) {
        Vector a = C - A, b = B - A, c = P - A;
        T under = a.x*b.y - b.x*a.y;
        T u = (c.x*b.y-c.y*b.x);
        T v = (a.x*c.y-a.y*c.x);
        return u >= 0 && v >= 0 && u+v <= under || u<=0
        && v<=0 && u+v >= under;
} //remove equalities if not want the boundary
```

```
Pick's theorem (So diem trong da giac co dinh nguyen)
        I = A - B/2 + 1
        where
                A is the area of a lattice polygon,
                I is number of lattice points inside it,
                B is number of lattice points on the boundary.
                Number of lattice points minus one on a line segment
                from (0, 0) and (x, y) is gcd(x, y).
```

Tich chap 3d: a x b = (aybz - azby, azbx - axbx, axby - aybx)

Distance from line AB to P (for any dimension) : $|(A-P)x(B-P)|/(A-B)$

Khoảng cách từ điểm đến đoạn thẳng if (dot(B-A, P-A) < 0) return dist(A,P); if (dot(A-B, P-B) < 0) return dist(B,P); return fabs(cross(P,A,B) / dist(A,B)); dot - tích vô hướng, cross - tích chập

Hình chiếu: Hình chiếu của C trên đường thẳng AB [dot(AB, AC) / dot(AB, AB)] AB

Đường tròn cắt đường tròn

```
// (x1,y1,r1) and (x2,y2,r2)
        a = x2 - x1;
        b = y2 - y1;
        $c = [(r1^2 - x1^2 - y1^2) - (r2^2 - x2^2 - y2^2)] / 2;
        $d = sqrt(a^2 + b^2);$

        if not |r1 - r2| <= d <= |r1 + r2|, return "no solution"
        if d == 0, circles are concentric, a special case
```

Đường tròn cắt đường thẳng

```
// (x1,y1,r1) and ax+by=c
Normalize line:
        d = sqrt(a^2+b^2)
        a /= d;
        b /= d;
        c /= d;
        e = c - a*x1 - b*y1;
        h = sqrt(r1^2 - e^2); // check if r1<e for circle-line test
        return (x1, y1) + (a*e, b*e) + h*(-b, a),
                (x1, y1) + (a*e, b*e) - h*(-b, a);
```

Template

```
double eps = 1e-8;

struct point2d {
        double x,y;
        point2d operator-(point2d v) { return (point2d){x-v.x, y-v.y}; }
        point2d operator+(point2d v) { return (point2d){x+v.x, y+v.y}; }
};

struct point3d {
        double x,y,z;
        point3d operator-(point3d v) { return (point3d){x-v.x, y-v.y, z-v.
            z}; }
};

double trip(point3d a, point3d b, point3d c) {
        return a.x * ( b.y * c.z - b.z * c.y )
                - a.y * ( b.x * c.z - b.z * c.x )
                + a.z * ( b.x * c.y - b.y * c.x );
}

double len(point2d a) { return sqrt(a.x*a.x+a.y*a.y); }

double len(point3d a) { return sqrt(a.x*a.x+a.y*a.y+a.z*a.z); }

point3d cross(point3d a, point3d b) {
        return (point3d){
                a.y*b.z - a.z*b.y,
                a.z*b.x - a.x*b.z,
                a.x*b.y - a.y*b.x
        };
}

double dot(point3d a, point3d b) { return a.x*b.x + a.y*b.y + a.z*b.z; }
double dot(point2d a, point2d b) { return a.x*b.x + a.y*b.y; }
double cross(point2d a, point2d b) {
        return a.x*b.y - a.y*b.x;
}
point2d rotate(point2d c, double angle) {
        return (point2d) {
                c.x*cos(angle)-c.y*sin(angle),
                c.x*sin(angle)+c.y*cos(angle)
        };
}

// distance AB to C
double linePointDist(point2d A, point2d B, point2d C, bool isSegment) {
        double dist = cross(B-A,C-A) / len(B-A);
        if(isSegment){
                if( dot(C-B,B-A) > eps) return len(B-C);
                if( dot(C-A,A-B) > eps) return len(A-C);
        }
        return fabs(dist);
}

struct Line { double A,B,C; };//Ax + By = C
Line makeline(point2d a1, point2d a2)
{
        Line ret = (Line){ a2.y-a1.y, a1.x-a2.x };
        ret.C = ret.A * a1.x + ret.B * a1.y;
        return ret;
}

double dist(Line l, point2d p)
{
        return fabs(p.x * l.A + p.y * l.B - l.C)/sqrt(l.A*l.A+l.B*l.B);;
}
Line rot90(Line l, point2d p)
{
        Line ret = (Line){ -l.B, l.A };
        ret.C = ret.A * p.x + ret.B * p.y;
        return ret;
}
point2d intersect(Line l1, Line l2)
{
        double det = l1.A*l2.B - l2.A*l1.B;
        if(fabs(det) < eps) det=0;  //zero means parallel
        return (point2d) { (l2.B*l1.C - l1.B*l2.C)/det,
        (l1.A*l2.C - l2.A*l1.C)/det};
}

//for segment segment intersection, check additionally
//min(x1,x2) <= x <= max(x1,x2)
//min(y1,y2) <= x <= max(y1,y2)
bool segmentsIntersect( point2d A, point2d B, point2d C, point2d E )
{
        point2d in = intersect( makeline(A,B), makeline(C,E) );
        return linePointDist(A,B,in,true) < eps
        && linePointDist(C,E,in,true) < eps;
}

// get a line passing between two points
Line getmidline(point2d a, point2d b)
{
        point2d mid(a+b); mid.x/=2; mid.y/=2;
        return rot90( makeline(a,b), mid );
}

//reflect a point into it's "mirror" with repect to a line
point2d reflectPoint(Line l, point2d p)
{
        Line r = rot90(l, p);
        point2d Y=intersect(l,r);
        return Y-(p-Y);
}
```