2025 ICPC team note

Slower than Python3

Nov 2025

Contents

1 Data Structure

1.1 Fenwick Tree

```
#include <bits/stdc++.h>
using namespace std;
#define N 300030


struct FenwickTree {
    int bit[N];
    int n=N;

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    void update(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

1.2 Segment Tree

```
// Segtree
#include <bits/stdc++.h>
#define INF 1000000007
using namespace std;

int seg[1<<18];

void build_seg(int idx, int l, int r){
    if(l==r){
        // seg[idx]=
        return;
    }
    int mid=(l+r)>>1;
    build_seg(2*idx,l,mid);
    build_seg(2*idx+1,mid+1,r);
    //seg[idx]=min(seg[2*idx],seg[2*idx+1]);
    // Some other operation
}

// 원소를 찾으면 disable도 같이 해주면 될 거 같음
void update_seg(int idx, int l, int r, int t_idx){
    if(l==r){
        seg[idx]=INF;
        return;
    }
    int mid=(l+r)>>1;
    if(t_idx<=mid) update_seg(2*idx,l,mid,t_idx);
    else update_seg(2*idx+1,mid+1,r,t_idx);
    //seg[idx]=min(seg[2*idx],seg[2*idx+1]);
}

int find_seg(int idx, int l, int r, int t_l, int t_r){
    if(t_l<=l && r<=t_r) return seg[idx];
    int mid=(l+r)>>1, ans=INF;
    if(t_l<=mid) //ans=min(ans,find_seg(2*idx,l,mid,t_l,t_r));
    if(t_r>mid) //ans=min(ans,find_seg(2*idx+1,mid+1,r,t_l,t_r));
    return ans;
}

int main()
{
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    return 0;
}
```

1.3 Lazy Segment Tree

```
// Lazy seg
#include <bits/stdc++.h>
#define INF 1000000007
using namespace std;

class node{
public:
    // Do something!
    node(){}
    node(){

    }
    node operator+(node b){
        node result;
        // Do something!
        return result;
    }
    // int val, lazy_val;
};

class lazy_seg{
public:
    node seg[1<<18];

    lazy_seg(){}
    lazy_seg(int n){
        //fill(seg,seg+n,node());
    }

    void build_seg(int idx, int l, int r){
        if(l==r){
            // Do something!
        }
        else{
            int mid=(l+r)>>1;
            build_seg(2*idx,l,mid);
            build_seg(2*idx+1,mid+1,r);
            seg[idx]=seg[2*idx]+seg[2*idx+1];
        }
    }

    void update_down(int idx, int l, int r){
        if(l==r) return;
        int mid=(l+r)>>1;
        // seg[2*idx], seg[2*idx+1]

        // seg[idx]의 lazy_val초기화

    }

    void update_seg(int idx, int l, int r, int t_l, int t_r,
                    int val){
        update_down(idx, l, r);
        if(t_l<=l && r<=t_r){
            // seg[idx].val과 seg[idx].lazy_val업데이트
            return;
        }
        int mid=(l+r)>>1;
        if(t_l<=mid) update_seg(2*idx,l,mid,t_l,t_r,val);
        if(t_r>mid) update_seg(2*idx+1,mid+1,r,t_l,t_r,val);
        seg[idx]=seg[2*idx]+seg[2*idx+1];
    }

    node find_seg(int idx, int l, int r, int t_l, int t_r){
        update_down(idx, l, r);
        if(t_l<=l && r<=t_r){
            return seg[idx];
        }
        node result;
        int mid=(l+r)>>1;
        if(t_l<=mid)
                result=result+find_seg(2*idx,l,mid,t_l,t_r);
        if(t_r>mid)
                result=result+find_seg(2*idx+1,mid+1,r,t_l,t_r);
        return result;
    }
};

lazy_seg s;
```

1.4 Persistent Segment Tree

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

#define N_node 2000000 // V log|Y_RANGE| scale
#define NX 100001
#define Y_RANGE 100001

class node{
public:
    node(){}
    node operator+(node b){
        node res;
        // Implement. 예시는 단순히 개수 세는 것.
        // 왼쪽에 더해지는 node의 l_idx, r_idx를 보존한다.
        res.l_idx=l_idx, res.r_idx=r_idx;
        res.cnt=cnt+b.cnt;
        return res;
    }
    node operator-(node b){
        node res;
        // Implement
        res.l_idx=l_idx, res.r_idx=r_idx;
        res.cnt=cnt-b.cnt;
        return res;
    }
    int l_idx=-1, r_idx=-1; // 여기서 idx는 본인의 idx를 말한다.
    long long cnt=0;
};

class PST{
public:
    int seg_root[NX]; // X-wise root idx.
    node seg[N_node]; // All node datas
    int cur_x=0, s_cnt=0; // s_cnt는 현재까지 사용한 node의 개수
    PST(){}
    void push(long long y){
        // Pusing one element generate another root.
        // Regardless of the val x.
        seg_root[cur_x]=s_cnt++;
        update_seg(seg_root[cur_x],(cur_x==0?-1:seg_root[cur_x-1]),
                    0,Y_RANGE-1,y,1);
        cur_x++;
    }
    node query(long long x1, long long x2,
               long long y1, long long y2){
        if(x1==0) return find_seg(seg_root[x2],0,Y_RANGE-1,y1,y2);
        return find_seg(seg_root[x2],0,Y_RANGE-1,y1,y2)
                -find_seg(seg_root[x1-1],0,Y_RANGE-1,y1,y2);
    }
    void clear(){
        for(int i=0; i<s_cnt; i++) seg[i]=node();
        cur_x=s_cnt=0;
    }

private:
    void update_seg(int idx, int prev_idx, int l, int r, int t_idx,
                    long long val){
        // previous segtree의 idx도 같이 관리. (첫 번째 layer의 경우는 -1)
        if(l==r){
            // Implement. 과거의 데이터에 덮어씌우는 것은 이부분이 유일.
            seg[idx].cnt=val;
            if(prev_idx!=-1) seg[idx]=seg[idx]+seg[prev_idx];
            return;
        }
        int mid=(l+r)>>1;
        if(t_idx<=mid){
            seg[idx].l_idx=s_cnt++;
            seg[idx].r_idx=(prev_idx==-1?-1:seg[prev_idx].r_idx);
            update_seg(seg[idx].l_idx,(prev_idx==-1?-1
                :seg[prev_idx].l_idx),l,mid,t_idx,val);
        }
        else{
            seg[idx].r_idx=s_cnt++;
            seg[idx].l_idx=(prev_idx==-1?-1:seg[prev_idx].l_idx);
            update_seg(seg[idx].r_idx,(prev_idx==-1?-1
                :seg[prev_idx].r_idx),mid+1,r,t_idx,val);
        }
        // 아래의 덧셈연산은 l_idx와 r_idx를 보존하기 위한 방법
        if(seg[idx].l_idx!=-1) seg[idx]=seg[idx]+seg[seg[idx].l_idx];
        if(seg[idx].r_idx!=-1) seg[idx]=seg[idx]+seg[seg[idx].r_idx];
    }
    node find_seg(int idx, int l, int r, int t_l, int t_r){
        if(t_l<=l && r<=t_r) return seg[idx];
        int mid=(l+r)>>1;
        node res;
        if(t_l<=mid && seg[idx].l_idx!=-1)
            res=res+find_seg(seg[idx].l_idx,l,mid,t_l,t_r);
        if(t_r>mid && seg[idx].r_idx!=-1)
            res=res+find_seg(seg[idx].r_idx,mid+1,r,t_l,t_r);
        return res;
    }
};

PST p;
```

1.5 PBDS

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type, less<int>, \
rb_tree_tag,tree_order_statistics_node_update>

// (Set 이름).order_of_key(NUM) :
// ordered_set 에서 NUM 보다 작은(미만의) 원소의 개수를 반환한다.

// (Set 이름).find_by_order(K) :
// ordered_set 에서 (K+1)번째 원소가 있는 iterator 을 반환한다. (K가 0이면 1번째)
```

1.6 Mo's algorithm

```
int sqrtN=400;

class Query{
public:
    int idx, s, e;
    bool operator < (Query &x){
        if(s/sqrtN != x.s/sqrtN) return s/sqrtN < x.s/sqrtN;
        return (e < x.e) ^ ((s/sqrtN)%2);
    }
};

Query qs[N];
int ans[N];

void go(int idx){

}

void back(int idx){

}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int q; // number of q
    int s = qs[0].s, e = qs[0].e;
    for(int i=s; i<=e; i++){
        go(i);
    }
    //ans[qs[0].idx] = ;

    for(int i=1; i<q; i++){
        while(s > qs[i].s) go(--s);
        while(e < qs[i].e) go(++e);
        while(s < qs[i].s) back(s++);
        while(e > qs[i].e) back(e--);
        //ans[qs[i].idx] = ;
    }
    for(int i=0; i<q; i++) cout << ans[i] << endl;

    return 0;
}
```

1.7 HLD

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

#define N 100000

vector<int> g[N];
int parent[N], depth[N], sz[N];
int head[N], pos[N], timer;

void dfs_sz(int u, int p = -1) {
    sz[u] = 1;
    parent[u] = p;
    for(int &v : g[u]) {
        if(v == p) continue;
        depth[v] = depth[u] + 1;
        dfs_sz(v, u);
        sz[u] += sz[v];
        if(sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
}

void dfs_hld(int u, int p = -1) {
    pos[u] = ++timer;
    for(int v : g[u]) {
        if(v == p) continue;
        head[v] = (v == g[u][0] ? head[u] : v);
        dfs_hld(v, u);
    }
}

int lca(int u, int v) {
    while(head[u] != head[v]) {
        if(depth[head[u]] < depth[head[v]]) swap(u, v);
        u = parent[head[u]];
    }
    return depth[u] < depth[v] ? u : v;
}

// Query path from u to v (use segment tree on pos[])
void path_query(int u, int v) {
    while(head[u] != head[v]) {
        if(depth[head[u]] < depth[head[v]]) swap(u, v);
        // Query segment tree: [pos[head[u]], pos[u]]
        u = parent[head[u]];
    }
    if(depth[u] > depth[v]) swap(u, v);
    // Query segment tree: [pos[u], pos[v]]
}

int main() {
    int n;
    cin >> n;

    for(int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    depth[1] = 0;
    dfs_sz(1);
    head[1] = 1;
    timer = 0;
    dfs_hld(1);

    // pos[u] = position of node u in segment tree
    // Use segment tree for path/subtree queries
}
```

1.8 Knuth Optimization

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


// cost function
// for (a,b,c,d) such that a <= b <= c <= d
// following must be satisfied:
// C(b,c) <= C(a,d)
// C(a,c) + C(b,d) <= C(a,d) + C(b,c)

int C(int i, int j) {
}

int solve() {
    int N;
    // read N and input
    int dp[N][N], opt[N][N];

    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        // Initialize dp[i][i] according to the problem
    }

    // works in O(N^2)
    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j]); k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }

    return dp[0][N-1];
}
```

1.9 Linear Convex Hull Trick

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

using namespace std;

typedef long long ll;
#define int ll
#define pb push_back
#define all(x) (x).begin(), (x).end()

const int len = 1<<19;

// assuming decreasing M / Increasing X

struct CHT {
    struct Line {
        ll m, c;
        ll eval(ll x) { return m * x + c; }
    };

    vector<Line> hull;
    int ptr = 0;

    bool bad(Line l1, Line l2, Line l3) {
        return (__int128)(l2.c - l1.c) * (l3.m - l2.m) <= \
        (__int128)(l3.c - l2.c) * (l2.m - l1.m);
    }

    void addLine(ll m, ll c) {
        Line L = {m, c};
        if (!hull.empty() && hull.back().m == L.m) {
            if (hull.back().c >= L.c) return;
            hull.pop_back();
        }
        while (hull.size() >= 2 && bad(hull[hull.size()-2], hull.back(), L)) {
            hull.pop_back();
        }
        hull.pb(L);
    }

    ll query(ll x) {

        // can change to binary search here to avoid condition of increasing X.

        while(ptr+1<(int)hull.size()&&hull[ptr + 1].eval(x)>=hull[ptr].eval(x)){
            ptr++;
        }
        return hull[ptr].eval(x);
    }
};
```

2 Graph

2.1 Dijkstra

```
// Dijkstra algorithm
#include <bits/stdc++.h>
using namespace std;
#define N 200000

vector<pair<int,int>> edges[N]; // {idx, dist}
int min_dist[N];

void dijkstra(int s){
    priority_queue<pair<int,int>> pq;
    fill(min_dist,min_dist+N,-1);
    pq.push({0,s});
    while(!pq.empty()){
        int dist=-pq.top().first, idx=pq.top().second;
        pq.pop();
        if(min_dist[idx]!=-1) continue;
        min_dist[idx]=dist;
        for(int i=0; i<edges[idx].size(); i++){
            int next=edges[idx][i].first,
                n_dist=dist+edges[idx][i].second;
            if(min_dist[next]==-1) pq.push({-n_dist,next});
        }
    }
}
```

2.2 Least Common Ancestor

```
// LCA(sparse table)

#include <bits/stdc++.h>
#define endl '\n'

using namespace std;

#define N 100100
#define MAX 20 // Should satisfy (1<<(MAX-1)) >= N

vector<int> edges[N];
int sparse[N][MAX];
int level[N];

void DFS_init(int idx, int p){
    sparse[idx][0]=p;
    for(int i=0; i<edges[idx].size(); i++){
        int next=edges[idx][i];
        if(next!=p){
            level[next]=level[idx]+1;
            DFS_init(next,idx);
        }
    }
}

void sparse_init(int s){
    level[s]=0;
    DFS_init(s,-1);
    for(int r=1; r<MAX; r++){
        for(int i=0; i<N; i++){
            if(sparse[i][r-1]==-1) sparse[i][r]=-1;
            else sparse[i][r]=sparse[sparse[i][r-1]][r-1];
        }
    }
}

int get_LCA(int a, int b){
    if(level[a]>level[b]) swap(a,b);
    for(int r=MAX-1; r>=0; r--){
        if(sparse[b][r]!=-1 && level[a]<=level[sparse[b][r]])
            b=sparse[b][r];
    }
    if(a==b) return a;
    for(int r=MAX-1; r>=0; r--){
        if(sparse[a][r]!=-1 && sparse[b][r]!=-1
            && sparse[b][r]!=sparse[a][r]){
            a=sparse[a][r], b=sparse[b][r];
        }
    }
    return sparse[a][0];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    int n,m,a,b;
    cin>>n;
    for(int i=0; i<n-1; i++){
        cin>>a>>b;
        a--, b--;
        edges[a].push_back(b);
        edges[b].push_back(a);
    }
    sparse_init(0);
    cin>>m;
    for(int i=0; i<m; i++){
        cin>>a>>b;
        cout<<get_LCA(a-1,b-1)+1<<endl;
    }
}
```

2.3 2-SAT

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


struct TwoSatSolver {
    int n_vars;
    int n_vertices;
    vector<vector<int>> adj, adj_t;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;

    TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 * n_vars),\
    adj(n_vertices), adj_t(n_vertices), used(n_vertices), order(), \
    comp(n_vertices, -1), assignment(n_vars) {
        order.reserve(n_vertices);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u])
                dfs1(u);
        }
        order.push_back(v);
    }

    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : adj_t[v]) {
            if (comp[u] == -1)
                dfs2(u, cl);
        }
    }

    bool solve_2SAT() {
        order.clear();
        used.assign(n_vertices, false);
        for (int i = 0; i < n_vertices; ++i) {
            if (!used[i])
                dfs1(i);
        }

        comp.assign(n_vertices, -1);
        for (int i = 0, j = 0; i < n_vertices; ++i) {
            int v = order[n_vertices - i - 1];
            if (comp[v] == -1)
                dfs2(v, j++);
        }

        assignment.assign(n_vars, false);
        for (int i = 0; i < n_vertices; i += 2) {
            if (comp[i] == comp[i + 1])
                return false;
            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }

    void add_disjunction(int a, bool na, int b, bool nb) {
        // na and nb signify whether a and b are to be negated
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        adj[neg_a].push_back(b);
        adj[neg_b].push_back(a);
        adj_t[b].push_back(neg_a);
        adj_t[a].push_back(neg_b);
    }

    static void example_usage() {
        TwoSatSolver solver(3); // a, b, c
        solver.add_disjunction(0, false, 1, true);  //     a  v  not b
        solver.add_disjunction(0, true, 1, true);   // not a  v  not b
        solver.add_disjunction(1, false, 2, false); //     b  v      c
        solver.add_disjunction(0, false, 0, false); //     a  v      a
        assert(solver.solve_2SAT() == true);
        auto expected = vector<bool>({true, false, true});
        assert(solver.assignment == expected);
    }
} scc;
```

2.4 SCC

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

#define N 100000

vector<int> g[N], gr[N];
bool vis[N];
int comp[N], num_scc;
vector<int> order;

void dfs1(int u) {
    vis[u] = true;
    for(int v : g[u]) {
        if(!vis[v]) dfs1(v);
    }
    order.push_back(u);
}

void dfs2(int u, int c) {
    comp[u] = c;
    for(int v : gr[u]) {
        if(comp[v] == -1) dfs2(v, c);
    }
```

```cpp
}

// Kosaraju's algorithm
void find_scc(int n) {
    // First DFS to get finish times
    fill(vis, vis + n + 1, false);
    for(int i = 1; i <= n; i++) {
        if(!vis[i]) dfs1(i);
    }

    // Second DFS on reversed graph in reverse order
    fill(comp, comp + n + 1, -1);
    num_scc = 0;
    reverse(order.begin(), order.end());
    for(int u : order) {
        if(comp[u] == -1) {
            dfs2(u, num_scc++);
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    for(int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        gr[v].push_back(u); // reversed graph
    }

    find_scc(n);

    // comp[u] = SCC id of node u (0 to num_scc-1)
    // Build condensation graph if needed
}
```

2.5 Eulerian Tour

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

// Eulerian Paths -- a path that goes through every edge exactly once
// Eulerian Tours - eulerian path that starts and ends at the same node

// a graph can have an Eulerian cycle (not path!) if and only if every node
// has an even degree.

// The condition of existence for an eulerian path in a directed graph
// is: At most one node has out[i] - in[i]=1 and at most one node has
// in[i] - out[i] = 1. This property is because an Eulerian path or
// cycle leaves a node the same number of times it enters the node.
// In a directed graph the exception are the start node and the end node.

{
    // Eulerian Tour

    int n, m;
    vector<vector<pair<int, int>>> g;
    vector<int> path;
    vector<bool> seen;

    void dfs(int node) {
        while (!g[node].empty()) {
            auto [son, idx] = g[node].back();
            g[node].pop_back();
            if (seen[idx]) { continue; }
            seen[idx] = true;
            dfs(son);
        }
        path.push_back(node);
    }

    int main() {
        cin >> n >> m;

        vector<int> degree(n, 0);
        g.resize(n);
        degree.resize(n);
        seen.resize(m);

        for (int i = 0; i < m; i++) {
            int x, y;
            cin >> x >> y;
            x--, y--;
            g[x].emplace_back({y, i});
            g[y].emplace_back({x, i});
            degree[x]++;
            degree[y]++;
        }

        for (int node = 0; node < n; node++) {
            if (degree[node] % 2) {
                cout << "IMPOSSIBLE" << endl;
                return 0;
            }
        }

        dfs(0);

        if (path.size() != m + 1) {
            cout << "IMPOSSIBLE";
        } else {
            for (int node : path) { cout << node + 1 << ' '; }
        }
        cout << endl;
    }
}

{

    // Eulerian Path

    int n, m;
    vector<vector<int>> g;
    vector<int> in, out, path;

    void dfs(int node) {
        while (!g[node].empty()) {
            int son = g[node].back();
            g[node].pop_back();
            dfs(son);
        }
        path.push_back(node);
    }

    int main() {
        cin >> n >> m;

        g.resize(n + 1);
        in.resize(n + 1);
        out.resize(n + 1);

        for (int i = 0; i < m; i++) {
            int x, y;
            cin >> x >> y;
            g[x].push_back(y);
            out[x]++;
            in[y]++;
        }

        bool flag = true;
        for (int node = 2; node < n && flag; node++) {
            if (in[node] != out[node]) { flag = false; }
        }
        if (out[1] != in[1] + 1 || out[n] != in[n] - 1 || !flag) {
            cout << "IMPOSSIBLE";
            return 0;
        }

        dfs(1);

        reverse(path.begin(), path.end());
        if (path.size() != m + 1 || path.back() != n) {
            cout << "IMPOSSIBLE";
        } else {
            for (auto node : path) { cout << node << ' '; }
        }
    }
}
```

2.6 Euler Tour

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

#define N 100000

vector<int> g[N];
int tin[N], tout[N], timer;

void dfs(int u, int p = -1) {
    tin[u] = ++timer;
    for(int v : g[u]) {
        if(v != p) dfs(v, u);
    }
    tout[u] = timer;
}

// Check if u is ancestor of v
bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[v] <= tout[u];
}

int main() {
    int n;
    cin >> n;

    for(int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    timer = 0;
    dfs(1);

    // Subtree of node u is [tin[u], tout[u]]
    // Can use segment tree/BIT on this range
}
```

2.7 Graph Coloring

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

vector<int> edges[100];
#define INF 100

int chromatic_number(int n){ // n is # of vertices of the graph
    // Minimum number of colors require to satisty "Graph Coloring Problem"
    assert(n<20); // O(3^N) algorithm
    int DP[(1<<n)];
    DP[0]=1; // Convinient in DP process
    for(int k=0; k<n; k++){
        for(int j=0; j<(1<<k); j++){
            int num=(1<<k)+j, res=INF; // num indicates current subset
            bool indep=0;
            if(DP[j]==1){ // Check independence
                indep=1;
                for(auto next : edges[k]){
                    if(j&(1<<next)){
                        indep=0;
                        break;
                    }
                }
            }
            if(indep) res=1;
            else{
                for(int t=num; t>0; t=(t-1)&num){
                    // t swaps all the subsets of num
                    if(num==t) continue;
                    res=min(res, DP[t]+DP[num-t]);
                }
            }
            DP[num]=res;
        }
    }
    return DP[(1<<n)-1];
}
```

3 Flows / Matchings

3.1 Min Cost Max Flow

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

#define N 1000
#define INF 1000000007

using namespace std;

vector<pair<int,int>> edges[N]; // next, cost.
// Must initialize after each testcase
int capa[N][N];
int flow[N][N];
bool inQ[N];
// (inQ[i]) means dist[i] changed ans need to be visited.
int pre[N];
int dist[N];

void add_edge(int a, int b, int w, int f){
    // a->b cost:w capa:f (Reverse edge has negative weight)
    edges[a].push_back({b,w}); capa[a][b]=f; flow[a][b]=0;
    edges[b].push_back({a,-w}); capa[b][a]=0; flow[b][a]=0;
}

void floyd_warshall(int s){
    // Only update pre[], dist[]
    fill(inQ, inQ+N, false);
    fill(dist, dist+N, INF);
    int ff=INF;
    queue<int> q;
    q.push(s);
    inQ[s]=true; // To maintain light queue
    dist[s]=0;
    while(!q.empty()){
        int idx=q.front(); inQ[idx]=false;
        q.pop();
        for(int i=0; i<edges[idx].size(); i++){
            int next=edges[idx][i].first, w=edges[idx][i].second;
            if(capa[idx][next]-flow[idx][next]>0
                && dist[idx]+w<dist[next]){
                dist[next]=dist[idx]+w;
                pre[next]=idx;
                if(!inQ[next]) q.push(next);
                inQ[next]=true;
            }
        }

    }
}

int MCMF(int s, int t){
    // Return minimum cost for maximum flow
    int res=0;
    while(true){
        floyd_warshall(s);
        if(dist[t]==INF) break;
        // Get flow
        int cur=t, ff=INF;
        while(cur!=s){
            ff=min(ff,capa[pre[cur]][cur]-flow[pre[cur]][cur]);
            cur=pre[cur];
        }
        // Flow update
        cur=t;
        while(cur!=s){
            flow[pre[cur]][cur]+=ff;
            flow[cur][pre[cur]]-=ff;
            cur=pre[cur];
        }
        res+=dist[t]*ff;
    }
    return res;
}
```

3.2 Dinic

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

using namespace std;

#define INF 1000000007
#define N 10000

vector<int> edges[N];
int flow[N][N]; // Use map if N is larger than 20000
int capa[N][N];
int levels[N];
bool visited[N];
int work[N];

void add_edge(int a, int b, int c){ // edge a->b (c)
    // this is NOT bidirectional!
    edges[a].push_back(b); flow[a][b]=0; capa[a][b]=c;
    edges[b].push_back(a); flow[b][a]=0; capa[b][a]=0;
    // If capa accumulate,
    //edges[a].push_back(b); flow[a][b]=0; capa[a][b]=c;
    //edges[b].push_back(a); flow[b][a]=0; capa[b][a]=0;
}

void level_BFS(int s){
    fill(visited,visited+N,false);
    queue<pair<int,int>> q;
    q.push({s,0});
    while(!q.empty()){
        int idx=q.front().first;
        int level=q.front().second;
        q.pop();
        if(visited[idx]) continue;
        levels[idx]=level;
        visited[idx]=true;
        for(int i=0; i<edges[idx].size(); i++){
            int next=edges[idx][i];
            if(!visited[next] && capa[idx][next]>flow[idx][next]){
                q.push({next,level+1});
            }
        }
    }
}

int main_DFS(int idx, int f, int t){
    visited[idx]=true;
    if(idx==t || f==0) return f;
    for(int &i=work[idx]; i<edges[idx].size(); i++){
        int next=edges[idx][i];
        if(!visited[next] && levels[next]==levels[idx]+1
                && capa[idx][next]>flow[idx][next]){
            int f_temp=main_DFS(next,
                    min(f,capa[idx][next]-flow[idx][next]),t);
            if(f_temp==0) continue;
            flow[idx][next]+=f_temp;
            flow[next][idx]-=f_temp;
            return f_temp;
        }
    }
    return 0;
}

int DINIC(int s, int t){
    int result=0;
    while(true){
        levels[t]=-1;
        level_BFS(s);
        if(levels[t]==-1) break;

        fill(work,work+N,0);
        while(true){
            fill(visited,visited+N,false);
            int f_temp=main_DFS(s,INF,t);
            if(f_temp==0) break;
            result+=f_temp;
        }
    }
    return result;
}

// 유량이 남은 모든 정점을 탐색
void cut_DFS(int idx){
    if(visited[idx]) return;
    visited[idx]=true;
    for(int i=0; i<edges[idx].size(); i++){
        int next=edges[idx][i];
        if(!visited[next] && capa[idx][next]>flow[idx][next])
            cut_DFS(next);
    }
}
```

3.3 Bipartite matching

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

#define N 100000

vector<int> edges[N];
vector<int> rev_edges[N]; // To find maximum independent set.
int pre[N];
bool visited[N];

bool DFS(int idx){
    // Bipartite matching
    if(visited[idx]) return false;
    visited[idx]=true;
    for(int i=0; i<edges[idx].size(); i++){
        int next=edges[idx][i];
        if(pre[next]==-1
            || (!visited[pre[next]] && DFS(pre[next]))){
            pre[next]=idx;
            return true;
        }
    }
    return false;
}
// fill(visited, visited+n, 0); 을 매번 해줘야 한다.
// N^2이 나오는 게 정상
// if(DFS(i)) result++; 이렇게 유량을 구함.
```

4 Strings

4.1 Aho Corasic

```
// Aho-corasic
// Upper-case code임에 주의.

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

#define NUM_ALPHA 26
#define N 10000

class node{
public:
    node* next[NUM_ALPHA]={NULL,};
    node* pphi; // parent of phi ; i.e. 최대 접미사의 가장 마지막 노드.
    // 다음 원소는 직접 찾아야 함.
    int val; // 부모를 통해 찾아줘도 된다. // 0-based idx of character
    bool end_of_snippet=false;
    // End node of snippet. Propagated in BFS phase.
    int length_of_snippet;
    // Only valid of end of snippet is true. Propagated in BFS phase.
    int depth=0; // Actual length of the sequence
    bool end=true; // All next pointers are NULL
    node(){}
    node(int v){
        val=v;
    }
};

class aho_trie{
public:
    node* root= new node();
    aho_trie(){}
    bool is_s(string &s){
        return is_re(root, 0, s);
    }
    void insert_s(string &s){
        insert_re(root, 0, s);
    }
    void clear(){ // clear_all
```

```
        clear_re(root);
        for(int i=0; i<NUM_ALPHA; i++) root->next[i]=NULL;
    }
    void get_phi(){
        queue<pair<node*,node*>> phi_q; // cur_pos, pphi
        phi_q.push({root,root});
        while(!phi_q.empty()){
            pair<node*,node*> temp=phi_q.front();
            phi_q.pop();
            node* cur=temp.first;
            node* ptar=temp.second;
            if(cur->depth>1){ // To prevent phi[self]=self
                while(ptar!=root && (ptar->next[cur->val]==NULL))
                    ptar=ptar->pphi;
                if(ptar->next[cur->val]!=NULL)
                    ptar=ptar->next[cur->val];
            }
            cur->pphi=ptar;
            if(!cur->end_of_snippet && ptar->end_of_snippet){
                cur->end_of_snippet=true;
                cur->length_of_snippet=ptar->length_of_snippet;
            }

            for(int i=0; i<NUM_ALPHA; i++){
                if((cur->next[i])!=NULL){
                    phi_q.push({cur->next[i],ptar});
                }
            }
        }
    }
    void find_matching(string &s_ref, vector<int> &ans){
        int n=s_ref.size();
        ans.clear();
        node* ptar=root;
        for(int i=0; i<n; i++){
            int cur=s_ref[i]-'A'; // transition function
            while(ptar!=root && ptar->next[cur]==NULL)
                ptar=ptar->pphi;
            if(ptar->next[cur]!=NULL) ptar=ptar->next[cur];
            //cout<<(char)(ptar->val+'A')<<' '<<ptar->cnt<<'
            //'<<ptar->depth<<endl;
            if(ptar->end_of_snippet)
                ans.push_back(i+1-(ptar->length_of_snippet));
            if(ptar->end) ptar=ptar->pphi;
        }
    }

private:
    bool is_re(node *cur, int idx, string &s){
        if(idx==s.size()) return (cur->end_of_snippet);
        if((cur->next[s[idx]-'A'])==NULL) return false;
        return is_re(cur->next[s[idx]-'A'],idx+1,s);
    }
    void insert_re(node* cur, int idx, string &s){
        if(idx==s.size()){
            cur->end_of_snippet=true;
            cur->length_of_snippet=s.size();
            return;
        }
        cur->end=false;
        if((cur->next[s[idx]-'A'])==NULL){
            cur->next[s[idx]-'A']=new node(s[idx]-'A');
            cur->next[s[idx]-'A']->depth=idx+1;
        }
        insert_re(cur->next[s[idx]-'A'],idx+1,s);
    }
    void clear_re(node* cur){
        for(int i=0; i<NUM_ALPHA; i++){
            if((cur->next[i])!=NULL){
                clear_re(cur->next[i]);
                delete cur->next[i];
            }
        }
    }
};

aho_trie aho;
```

4.2 KMP

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

#define N 2020202
string s_ref;
string s_sni;
int phi[N];
vector<int> match_res;

void get_phi(){
    int m=s_sni.size();
    phi[0]=0;
    for(int i=1, j=0; i<m; i++){
        while(j>0 && s_sni[i]!=s_sni[j]) j=phi[j-1];
        if(s_sni[i]==s_sni[j]) j++;
        phi[i]=j;
    }
}
```

```
void KMP(){
    int n=s_ref.size(), m=s_sni.size();
    match_res.clear();
    for(int i=0, j=0; i<n; i++){
        while(j!=0 && s_ref[i]!=s_sni[j]) j=phi[j-1];
        if(s_ref[i]==s_sni[j]) j++;
        if(j==m) match_res.push_back(i+1-m), j=phi[j-1];
    }
}
```

4.3 Trie

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

using namespace std;

#define NUM_ALPHA 26

class node{
public:
    node* next[NUM_ALPHA]={NULL,};
    char val; // 대부분 필요없다.
    int cnt=0; // 여기서 끝나는 단어들의 개수
    node(){}
    node(char c){
        val=c;
    }
};

class trie{
public:
    node* root= new node();
    trie(){}
    bool is_s(string &s){
        return is_re(root, 0, s);
    }
    void insert_s(string &s){
        insert_re(root, 0, s);
    }
    void clear(){ // clear_all
        clear_re(root);
        for(int i=0; i<NUM_ALPHA; i++) root->next[i]=NULL;
    }

private:
    bool is_re(node *cur, int idx, string &s){
        if(idx==s.size()) return ((cur->cnt)>=1);
        // 여기까지 왔으면 그냥 true해도 되지 않나?
        // 처음에 들어올 때 때문인가
        if((cur->next[s[idx]-'A'])==NULL) return false;
        return is_re(cur->next[s[idx]-'A'],idx+1,s);
    }
    void insert_re(node* cur, int idx, string &s){
        if(idx==s.size()){
            cur->cnt++;
            return;
        }
        if((cur->next[s[idx]-'A'])==NULL)
            cur->next[s[idx]-'A']=new node(s[idx]);
        insert_re(cur->next[s[idx]-'A'],idx+1,s);
    }
    void clear_re(node* cur){
        for(int i=0; i<NUM_ALPHA; i++){
            if((cur->next[i])!=NULL){
                clear_re(cur->next[i]);
                delete cur->next[i];
            }
        }
    }
};

trie tr;
```

4.4 Suffix LCP

```
// LCP[1] + ... + LCP[N-1] 은 모든 중복되는 substring의 개수

// result <- result + max(0,prev-LCP[i])
// prev <- LCP[i]
// 이러면 중복되는 substring의 종류를 얻는다.

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

#define N 200000
string s;
int g[N+1], temp_g[N]; // g[i]는 정렬 전 i번째 접미사가 접근하는 group ordering
int t; // 전역변수
int sa[N], order[N]; // order[i]: position of suffix [i, ..., N-1] in the sorted array

int LCP[N]; // length of the prefix overlap between i th and i-1 th suffix
// -> You should use LCP[order[i]] to access LCP value of suffix starts at index i.

bool compare(int i, int j){
    if(g[i]==g[j]) return g[i+t]<g[j+t];
    else return g[i]<g[j];
}
```

```
// SA[i]는 정렬된 suffix중 i번째의 것이 원래의 suffix중 몇 번째 인지
// order[i]는 SA[i]의 역함수인데, i번째의 원래 suffix가 정렬되면 몇 번째로 들어가냐.

void suffix_sort(const string &s){
    int n=s.size();
    for(int i=0; i<n; i++) sa[i]=i, g[i]=s[i];
    g[n]=-1, temp_g[0]=0;
    for(t=1; t<n; t<<=1){
        sort(sa,sa+n,compare);
        for(int i=1; i<n; i++){
            temp_g[sa[i]]=temp_g[sa[i-1]]+compare(sa[i-1],sa[i]);
        }
        for(int i=0; i<n; i++) g[i]=temp_g[i];
    }
    for(int i=0; i<n; i++) order[sa[i]]=i;
}

void get_lcp(const string &s){
    int n=s.size();
    for(int i=0, j=0; i<n; i++){
        if(order[i]!=0){
            int pre=sa[order[i]-1];
            while(max(i+j,pre+j)<n && s[i+j]==s[pre+j]) j++;
            LCP[order[i]]=j;
            j=max(j-1,0);
        }
    }
}
```

4.5 Manacher

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Manacher {
public:

    // p[i] = radius of longest palindrome centered at i
    // in transformed string
    vector<int> p;

    // transformed string with # and sentinels
    string ms;

    // preprocess the string and run the algorithm
    Manacher(string &s) {
        // left sentinel
        ms = "@";
        for (char c : s) {
            ms += "#" + string(1, c);
        }

        // right sentinel
        ms += "#$";

        // run Manacher's algorithm
        runManacher();
    }

    void runManacher() {
        int n = ms.size();
        p.assign(n, 0);
        int l = 0, r = 0;

        for (int i = 1; i < n - 1; ++i) {

            // mirror of i around center (l + r)/2
            int mirror = l + r - i;

            // initialize p[i] based on its mirror
            // if within bounds
            if (i < r)
                p[i] = min(r - i, p[mirror]);

            // expand palindrome centered at i
            while (ms[i + 1 + p[i]] == ms[i - 1 - p[i]]){
                ++p[i];
            }

            // update [l, r] if the palindrome expands
            // beyond current r
            if (i + p[i] > r) {
                l = i - p[i];
                r = i + p[i];
            }
        }
    }

    // returns length of longest palindrome centered
    // at 'cen' in original string
    // 'odd' = 1 → check for odd-length, 'odd' = 0 → even-length
    int getLongest(int cen, int odd) {

        // map original index to transformed string index
        int pos = 2 * cen + 2 + !odd;
        return p[pos];
    }

    // checks if s[l..r] is a palindrome in O(1)
    bool check(int l, int r) {
        int len = r - l + 1;
        int cen = (l + r) / 2;
        return len <= getLongest(cen, len % 2);
    }
};
```

5 Mathematics

5.1 Arithmetic Inverse

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

#define MOD 998244353
#define MAX_DIGIT 62

// (ak)*(bk)^(-1) === a*b^(-1) === (ak%P)*(bk%P)^(-1) (mod P)
long long ari_inv(long long num){
    // Calculate num^(MOD-2)
    long long res=1, mult=num;
    for(int i=0; i<MAX_DIGIT; i++){
        if((MOD-2)&(1LL<<i)){
            res=(res*mult)%MOD;
        }
        mult=(mult*mult)%MOD;
    }
    return res;
}
```

5.2 Sum Over Subsets

```
// works in 3^n
void naive_iteration(int n) {
    for (int x = 0; x < (1 << n); x++) {
        // iterate over all subsets of x directly
        for (int i = x; i > 0; i = (i - 1) & x) {

        }
    }
}
```

5.3 Floor Sum

```
// calculates for(i = 1; i <= n; i ++) sum += n/i
long long floor_sum(long long n) {
    long long c = 1;
    long long sum = 0;
    while(c <= n) {
        long long v = n/c;
        long long cp = n / v;
        sum += (cp - c + 1) * v;
        c = cp+1;
    }
    return sum;
}
```

5.4 FFT

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

typedef complex<double> cpx;
const int SZ = 1048576;

void FFT(cpx g[], bool inv = false){
    int n = SZ;
    for(int i = 1, j = 0; i < n; ++i){
        int b = n/2;
        while(!((j ^= b) & b)) b /= 2;
        if(i < j) swap(g[i], g[j]);
    }
    for(int k = 1; k < n; k *= 2){
        double a = (inv ? M_PI/k : -M_PI/k);
        cpx w(cos(a), sin(a));
        for(int i = 0; i < n; i += k*2){
            cpx wp(1, 0);
            for(int j = 0; j < k; ++j){
                cpx x = g[i+j], y = g[i+j+k] * wp;
                g[i+j] = x + y;
                g[i+j+k] = x - y;
                wp *= w;
            }
        }
    }
    if(inv){
        for(int i = 0; i < n; ++i)
            g[i] /= n;
    }
}
```

5.5 NTT

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

typedef long long ll;
#define SZ 1<<18
#define MOD 998244353
#define AA 119
#define BB 23
#define PR 3
#define MAX_DIGIT 62

long long ari_inv(long long num){
    long long res=1, mult=num;
    for(ll i=0; i<MAX_DIGIT; i++){
        if((MOD-2)&(1LL<<i)){
            res=(res*mult)%MOD;
        }
        mult=(mult*mult)%MOD;
    }
    return res;
}

// The output can be negative
void NTT(ll g[], bool inv = false){
    ll n = SZ;
    ll w_pow[n+1];
    ll bit_cnt=0;
    for(; bit_cnt<=BB; bit_cnt++){
        if((n>>bit_cnt) == 1) break;
    }
    assert ((n>>bit_cnt)==1);

    ll w = 1;
    ll pow = AA<<(BB-bit_cnt);
    ll mul = PR;
    for(int i=0; i<MAX_DIGIT; i++){
        if(pow&(1LL<<i)) w=(w*mul)%MOD;
        mul=(mul*mul)%MOD;
    }
    w = (inv ? w : ari_inv(w));

    for(ll i = 1, j = 0; i < n; ++i){
        ll b = n/2;
        while(!((j ^= b) & b)) b /= 2;
        if(i < j) swap(g[i], g[j]);
    }

    ll cnt=0;
    for(ll k = 1; k < n; k *= 2){
        cnt++;
        ll ww = w;
        for(ll i=0; i<bit_cnt-cnt; i++) ww=(ww*ww)%MOD;
        for(ll i=0; i<k; i++) w_pow[i]=(i==0?1:(w_pow[i-1]*ww)%MOD);
        for(ll i = 0; i < n; i += k*2){
            ll wp = 1;
            for(ll j = 0; j < k; ++j){
                ll x = g[i+j], y = g[i+j+k]*w_pow[j];
                g[i+j] = (x + y);
                g[i+j+k] = (x - y);
            }
        }
        for(int i=0; i<n; i++) g[i]%=MOD;
    }
    if(inv){
        ll ari_n = ari_inv(n);
        for(ll i = 0; i < n; ++i)
            g[i] = (g[i]*ari_n)%MOD;
    }
}


/*
// convolution (size(datas) = SZ >= 2*n 인 2의 거듭제곱)
NTT(datas1);
NTT(datas2);
for(ll i=0; i<SZ; i++) datas1[i]=(datas1[i]*datas2[i])%MOD;
NTT(datas1,true);
*/
```

5.6 NTT + Chinese Remainder Thm

```
// NTT + CRT
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

typedef long long ll;

/* MOD = a 2^b + 1 일 때, [a, b, Primitive root] 순서쌍
N으로 사용하는 값이 2^b보다 크면 사용할 수 없다.
998244353 [119, 23, 3]
```

```
2281701377 [17, 27, 3]
2483027969 [37, 26, 3]
2113929217 [63, 25, 5]
104857601 [25, 22, 3]
1092616193 [521, 21, 3]
*/

ll SZ=1<<18;
#define MOD1 998244353
#define AA1 119
#define BB1 23
#define PR1 3
#define MOD2 2483027969
#define AA2 37
#define BB2 26
#define PR2 3
#define MAX_DIGIT 62

long long ari_inv(long long num, bool ver){
    long long res=1, mult=num;
    ll MOD = (ver?MOD1:MOD2);
    for(ll i=0; i<MAX_DIGIT; i++){
        if((MOD-2)&(1LL<<i)){
            res=(res*mult)%MOD;
        }
        mult=(mult*mult)%MOD;
    }
    return res;
}


void NTT(ll g[], bool ver, bool inv=false){
    ll MOD=(ver?MOD1:MOD2), AA=(ver?AA1:AA2), BB=(ver?BB1:BB2),
        PR=(ver?PR1:PR2);
    ll n = SZ;
    ll w_pow[n+1];
    ll bit_cnt=0;
    for(; bit_cnt<=BB; bit_cnt++){
        if((n>>bit_cnt) == 1) break;
    }
    assert ((n>>bit_cnt)==1);

    // w^n % MOD == 1 이어야 함.
    // w = PR^((p-1) / n) = PR ^ (a*2^(b-bit_cnt))
    ll w = 1;
    ll pow = AA<<(BB-bit_cnt);
    ll mul = PR;
    for(int i=0; i<MAX_DIGIT; i++){
        if(pow&(1LL<<i)) w=(w*mul)%MOD;
        mul=(mul*mul)%MOD;
    }
    w = (inv ? w : ari_inv(w,ver));

    for(ll i = 1, j = 0; i < n; ++i){
        ll b = n/2;
        while(!((j ^= b) & b)) b /= 2;
        if(i < j) swap(g[i], g[j]);
    }

    ll cnt=0;
    for(ll k = 1; k < n; k *= 2){
        cnt++;
        ll ww = w; // ww^(2*k) % MOD == 1 이어야 함.
        // ww=w^(n/(2*k))=w^(2^(bit_cnt-cnt))
        for(ll i=0; i<bit_cnt-cnt; i++) ww=(ww*ww)%MOD;
        for(ll i=0; i<k; i++) w_pow[i]=(i==0?1:(w_pow[i-1]*ww)%MOD);
        for(ll i = 0; i < n; i += k*2){
            ll wp = 1;
            for(ll j = 0; j < k; ++j){
                ll x = g[i+j], y = g[i+j+k]*w_pow[j];
                g[i+j] = (x + y);
                // MOD*MOD+MOD가 ll로 표현된다면 중간과정에서 %MOD 필요없음.
                g[i+j+k] = (x - y);
            }
        }
        for(int i=0; i<n; i++) g[i]%=MOD;
        // 위의 for문 동안 한 번만 방문
    }
    if(inv){
        ll ari_n = ari_inv(n,ver);
        for(ll i = 0; i < n; ++i)
            g[i] = (g[i]*ari_n)%MOD;
    }
}

inline ll large_mul(ll a, ll b){
    // 이것 때문에 TLE가 뜨진 않는다.
    ll MOD=(MOD1*MOD2);
    a=(a%MOD+MOD)%MOD;
    b=(b%MOD+MOD)%MOD;
    ll mul=a;
    ll res=0;
    for(int i=0; i<MAX_DIGIT; i++){
        if(b & (1LL<<i)) res=(res+mul)%MOD;
        mul=(mul+mul)%MOD;
    }
    return res;
}
```

```
/*
// for MOD1
NTT(datas1_M1, true, false);
NTT(datas2_M1, true, false);
for(ll i=0; i<SZ; i++) datas1_M1[i]=(datas1_M1[i]*datas2_M1[i])%MOD1;
NTT(datas1_M1, true, true);

// for MOD2
NTT(datas1_M2, false, false);
NTT(datas2_M2, false, false);
for(ll i=0; i<SZ; i++) datas1_M2[i]=(datas1_M2[i]*datas2_M2[i])%MOD2;
NTT(datas1_M2, false, true);

// Chinese Theorem
ll n1 = MOD2, n2=MOD1, MOD=(MOD1*MOD2);
ll s1 = ari_inv(MOD2, true), s2=ari_inv(MOD1, false);
for(int i=0; i<2*l; i++){
    datas1_M1[i] = (large_mul(datas1_M1[i]*n1,s1)
    + large_mul(datas1_M2[i]*n2,s2)) % MOD;
}
*/
```

5.7 Ternary Search

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

#define INF 1e18

long long f(long long x){
    long long res;
    // calculate f(x)
    return res;
}

long long ternary(long long s, long long e){
    // Return smallest x if there are multiple minimum values.
    while(3<=e-s){
        long long l=(s+s+e)/3, r=(s+e+e)/3;
        if(f(l)>f(r)) s=l;
        else e=r;
    }
    long long mx=INF, res;
    for(long long i=s; i<=e; i++){
        long long temp=f(i);
        if(mx>temp) mx=temp, res=i;
    }
    return res;
}
```

5.8 Gale-Shapley Algorithm

```
#include <bits/stdc++.h>
using namespace std;
#define rng(i,a,b) for(int i=a; i<=b; i++)
#define gnr(i,a,b) for(int i=a; i>=b; i--)
template<class t> using vc=vector<t>;
typedef long long ll;
using pii=pair<int,int>;
using pll=pair<ll,ll>;
#define N 1010

// O(N^3) algorithm
int A_pref[N][N], B_pref[N][N];
// Suppose u\in A and v\in B.
// A[u][v]: u's preference score for v (the higher the better).
// B[v][u]: v's preference score for u.

int A_order[N][N]; // Initialize before call Gale_Shapley()
// A_order[u][i]: i-th prefered element from B for u. (So prefered element idx first)
bool attempted[N][N];
// attempted[u][v]: Indicates whether the u attempted v previously
int A_match[N], B_match[N]; // current match. Initialize to -1

void Gale_Shapley(int n){
    // return: match idx in A_match and B_match
    fill(A_match,A_match+n,-1);
    fill(B_match,B_match+n,-1);
    rng(i,0,n-1) fill(attempted[i],attempted[i]+n,0);
    queue<int> unmatched_A;

    rng(i,0,n-1) unmatched_A.push(i);
    while(!unmatched_A.empty()){
        int u=unmatched_A.front();
        unmatched_A.pop();
        rng(i,0,n-1){
            int v=A_order[u][i];
            if(attempted[u][v]) continue;
            if(B_match[v]==-1){
                A_match[u]=v;
                B_match[v]=u;
                break;
            }
            else if(B_pref[v][B_match[v]] < B_pref[v][u]){
                // u is prefered over previous match
                unmatched_A.push(B_match[v]);
                A_match[u]=v;
                B_match[v]=u;
                break;
            }
        }
    }
}
```

6 Geometry

6.1 Line Intersection

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


class Point {
public:
    long long x, y;
    Point(long long x, long long y) {
        this->x = x;
        this->y = y;
    }
};

int ccw(Point A, Point B, Point C) {
    long long ccw = (B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x);

    if (ccw > 0) return 1;
    else if(ccw < 0) return -1;
    else if(ccw == 0) return 0;
}

bool isIntersect(Point A, Point B, Point C, Point D) {
    int ccw1 = ccw(A,B,C)*ccw(A,B,D);
    int ccw2 = ccw(C,D,A)*ccw(C,D,B);

    if (ccw1 <= 0 && ccw2 <= 0) {
        if (ccw1 == 0 && ccw2 ==0) {
            // All 4 points are collinear.
            // Check if they overlap
            return min(A.x, B.x) <= max(C.x, D.x) &&
                 min(C.x, D.x) <= max(A.x, B.x) &&
                 min(A.y, B.y) <= max(C.y, D.y) &&
                 min(C.y, D.y) <= max(A.y, B.y);
        }
        return true;
    }

    return false;
}
```

6.2 Basic Operations

```
// Credit : PLEASE OPEN TESTDATA teamnote
inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}

inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point {
    double x, y;
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    Point operator+(const Point& rhs) const {
        return Point{ x + rhs.x, y + rhs.y };
    }
    Point operator-(const Point& rhs) const {
        return Point{ x - rhs.x, y - rhs.y };
    }
    Point operator*(double t) const {
        return Point{ x * t, y * t };
    }
};

struct Circle {
    Point center;
    double r;
};

struct Line {
    Point pos, dir;
};

inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}
inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}
inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}
inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}
inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}

inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}

bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}

bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
    ret = b.pos + b.dir * t2;
    return true;
}

Point inner_center(const Point &a, const Point &b, const Point &c) {
    double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    double w = wa + wb + wc;
    return Point{ (wa * a.x + wb * b.x + wc * c.x) / w, \
    (wa * a.y + wb * b.y + wc * c.y) / w };
}

Point outer_center(const Point &a, const Point &b, const Point &c) {
    Point d1 = b - a, d2 = c - a;
    double area = outer(d1, d2);
    double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
        + d1.y * d2.y * (d1.y - d2.y);
    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
        + d1.x * d2.x * (d1.x - d2.y);
    return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0 };
}

vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    double a = 2 * inner(line.dir, line.dir);
    double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
        + line.dir.y * (line.pos.y - circle.center.y));
    double c = inner(line.pos - circle.center, line.pos - circle.center)
        - circle.r * circle.r;
    double det = b * b - 2 * a * c;
    int pred = diff(det, 0);
    if (pred == 0)
        result.push_back(line.pos + line.dir * (-b / a));
    else if (pred > 0) {
        det = sqrt(det);
        result.push_back(line.pos + line.dir * ((-b + det) / a));
        result.push_back(line.pos + line.dir * ((-b - det) / a));
    }
    return result;
}

vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    int pred = diff(dist(a.center, b.center), a.r + b.r);
    if (pred > 0) return result;
    if (pred == 0) {
        result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
        return result;
    }
    double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    double tmp = (bb - aa) / 2.0;
    Point cdiff = b.center - a.center;
    if (diff(cdiff.x, 0) == 0) {
        if (diff(cdiff.y, 0) == 0)
            return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line{ Point{ 0, tmp / cdiff.y }, Point{ 1, 0 } });
    }
    return circle_line(a,
        Line{ Point{ tmp / cdiff.x, 0 }, Point{ -cdiff.y, cdiff.x } });
}
```

```
Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b - a, cb = c - b;
    Line p{ (a + b) * 0.5, Point{ ba.y, -ba.x } };
    Line q{ (b + c) * 0.5, Point{ cb.y, -cb.x } };
    Circle circle;
    if (!get_cross(p, q, circle.center))
        circle.r = -1;
    else
        circle.r = dist(circle.center, a);
    return circle;
}

Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
    double det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        double h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x - a.x } * h;
        circle.r = r;
    }
    return circle;
}
```

6.3 Convex Hull

```
// Convex hull
#include <bits/stdc++.h>
#define fi first
#define se second
using namespace std;
typedef long long ll;
#define N 100000

pair<ll,ll> datas[N]; // Vertices
// Convex hull algorithm never permute initial vertices.
int sorted_idx[N]; // Idx of 'sorted' datas.
int ans[N]; // Elements of convex hull

inline bool CCW(const pair<ll,ll>& a,
    const pair<ll,ll>& b, const pair<ll,ll>& c){
    return (b.fi-a.fi)*(c.se-a.se)-(b.se-a.se)*(c.fi-a.fi)>0;
    // 변 위의 점도 필요하다면 >=
}
inline long long dist(const pair<ll,ll>& a,const pair<ll,ll>& b){
    return (a.fi-b.fi)*(a.fi-b.fi)+(a.se-b.se)*(a.se-b.se);
}

class comp_c{
public:
    pair<ll,ll> mp;
    comp_c(pair<ll,ll> minpos){
        mp=minpos;
    }
    bool operator()(int a_idx, int b_idx){
        pair<ll,ll> a=datas[a_idx], b=datas[b_idx];
        ll com=(a.fi-mp.fi)*(b.se-mp.se)-(a.se-mp.se)*(b.fi-mp.fi);
        if(com==0) return dist(a,mp)<dist(b,mp);
        else return com>0;
    }
};

int convex_hull(int n){ // # number of vertices
    // Phase 1: Sort
    int m_idx=0;
    for(int i=1; i<n; i++) if(datas[m_idx]>datas[i]) m_idx=i;
    for(int i=0; i<n; i++) sorted_idx[i]=i;
    swap(sorted_idx[0], sorted_idx[m_idx]);
    sort(sorted_idx,sorted_idx+n,comp_c(datas[m_idx]));

    // Phase 2: Get Convexhull
    int st[N]; // Elements of st are the idx of datas
    int cur=0, st_cnt=0;
    // cur : # of visited elements of s,
    // st_cnt : # of elements in stack
    while(cur<n){
        while(st_cnt>=2 && !CCW(datas[st[st_cnt-2]],
        datas[st[st_cnt-1]],datas[sorted_idx[cur]])) st_cnt--;
        st[st_cnt++]=sorted_idx[cur++];
    }
    // Counter clockwise, starts with leftmost vertices.
    for(int i=0; i<st_cnt; i++) ans[i]=st[i];
    return st_cnt;
}
```

6.4 Rotating Calipers

```
// Rotating calipers
#include <bits/stdc++.h>
#define fi first
#define se second

using namespace std;
typedef long long ll;

#define N 100000
```

```
pair<ll,ll> datas[N]; // Vertices
// Convex hull algorithm never permute initial vertices.
int sorted_idx[N]; // Idx of 'sorted' datas.
int ans[N]; // Elements of convex hull

inline bool CCW(const pair<ll,ll>& a, const pair<ll,ll>& b,
    const pair<ll,ll>& c){
    return (b.fi-a.fi)*(c.se-a.se)-(b.se-a.se)*(c.fi-a.fi)>0;
    // 변 위의 점도 필요하다면 >=
}
inline bool CCW_4(const pair<ll,ll>& a,
    const pair<ll,ll>& na, const pair<ll,ll>& b, pair<ll,ll> &nb){
    return (na.fi-a.fi)*(nb.se-b.se)-(na.se-a.se)*(nb.fi-b.fi)>=0;
}
inline long long dist(const pair<ll,ll>& a, const pair<ll,ll>& b){
    return (a.fi-b.fi)*(a.fi-b.fi)+(a.se-b.se)*(a.se-b.se);
}

class comp_c{
public:
    pair<ll,ll> mp;
    comp_c(pair<ll,ll> minpos){
        mp=minpos;
    }
    bool operator()(int a_idx, int b_idx){
        pair<ll,ll> a=datas[a_idx], b=datas[b_idx];
        ll com=(a.fi-mp.fi)*(b.se-mp.se)-(a.se-mp.se)*(b.fi-mp.fi);
        if(com==0) return dist(a,mp)<dist(b,mp);
        else return com>0;
    }
};

int convex_hull(int n){ // # number of vertices
    // Phase 1: Sort
    int m_idx=0;
    for(int i=1; i<n; i++) if(datas[m_idx]>datas[i]) m_idx=i;
    for(int i=0; i<n; i++) sorted_idx[i]=i;
    swap(sorted_idx[0], sorted_idx[m_idx]);
    sort(sorted_idx,sorted_idx+n,comp_c(datas[m_idx]));

    // Phase 2: Get Convexhull
    int st[N]; // Elements of st are the idx of datas
    int cur=0, st_cnt=0;
    // cur : # of visited elements of s,
    // st_cnt : # of elements in stack
    while(cur<n){
        while(st_cnt>=2 && !CCW(datas[st[st_cnt-2]],
            datas[st[st_cnt-1]],datas[sorted_idx[cur]])) st_cnt--;
        st[st_cnt++]=sorted_idx[cur++];
    }
    for(int i=0; i<st_cnt; i++) ans[i]=st[i];
    // Counter clockwise, starts with leftmost vertices.
    return st_cnt;
}

pair<int,int> rotating_calipers(int sz_hull){
    // # of vertices in convex hull.
    // ans[] must be filled before call this function.
    int mx_a, mx_b; // mx_a and mx_b are the idx of two vertices
                    // making the maximum distance.
    long long mx=0;
    for(int a=0, p=0; a<sz_hull; a++){
        while(p+1<sz_hull && CCW_4(datas[ans[a]], datas[ans[a+1]],
                datas[ans[p]], datas[ans[p+1]])){
            ll temp=dist(datas[ans[a]], datas[ans[p]]);
            if(mx<temp) mx=temp, mx_a=ans[a], mx_b=ans[p];
            p++;
        }
        ll temp=dist(datas[ans[a]], datas[ans[p]]);
        if(mx<temp) mx=temp, mx_a=ans[a], mx_b=ans[p];
    }
    return {mx_a, mx_b}; // Idx of initial vertices. i.e. datas[]
}
```

7 Miscellaneous

7.1 Optimization Command

```
// optimization "O1"
#pragma GCC optimize("O1")

// ?
__attribute__((optimize("Ofast,unroll-loops"),target("avx,avx2,fma")))
```

7.2 Mathematics

Green's Theorem Let C be a positively oriented, piecewise smooth, simple closed curve in a plane. Then

$$\oint_C (Ldx + Mdy) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) dxdy,$$

where the path integral is taken counterclockwise.

Shoelace Lemma

```
for (int i=0; i<n; i++){
    area += X[i]*Y[(i+1)%len] - X[i]*Y[(i+len-1)%len];
}
return abs(area/2);
```

Burnside's Lemma

$$|X/G| = \sum_{g \in G} |X^g|$$

Pisano Period Pisano Period refers to the period with which Fibonacci numbers mod some P repeats. In other words, for some modulus P, its Pisano Period is equal to some M such that

$$F_a \equiv F_{(a\%M)} \mod P \ \ \forall\, a.$$

for moduli of form 10^k, its $15 * 10^k$.

Euler Characteristics For a planar graph, the following identity holds:

$$x = V - E + F$$

where V = vertices, E = edges, F = faces

Kőnig's Theorem For all Graphs, Minimum Cut = Max flow
Only for **Bipartite Graphs**: Minimum Vertex Cover = Minimum Cut = Max flow (Kőnig's Theorem)