Team - Silent 3lers

Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar

April 10, 2024

Contents

1 Bit-Manipulations **1**
1.1 Bit-Manipulations . . . 1
1.2 Iterate-Submasks . . . 1

2 DP **1**
2.1 Digit-DP . . . 1
2.2 Divide-Conquer-DP . . . 2
2.3 Knuth-Optimization . . . 2
2.4 SOS . . . 3

3 Graph **3**
3.1 Articulation-Point . . . 3
3.2 Bridge-In-Graph . . . 4
3.3 Check-Bipartite-Online-DSU . . . 4
3.4 DFS-Start-End-Time . . . 4
3.5 Dijkstra . . . 5
3.6 DSU . . . 5
3.7 Floyd-Warshall . . . 5
3.8 Maximum-Bipartite-Matching . . . 5
3.9 Reachability-Tree . . . 6

4 Kosaraju **7**
4.1 2-SAT . . . 7
4.2 Strongly-Connected-Components . . . 7

5 Main-Template **8**
5.1 Main . . . 8
5.2 Others . . . 9

6 Math **9**
6.1 Euclid . . . 9
6.2 Euler-Totient . . . 10
6.3 Nck-Npk . . . 11
6.4 Power-And-Modulo . . . 11
6.5 Primes . . . 11

7 Miscellaneous **11**
7.1 Catalan-Numbers . . . 11
7.2 Comments . . . 12
7.3 Convex-Hull . . . 12
7.4 Coordinate-Compression . . . 13
7.5 Matrix-Exponentian . . . 13
7.6 MO-Algorithm . . . 14

8 Segment-Tree **14**
8.1 Fenwick-Tree . . . 14
8.2 Modify-Addition . . . 15
8.3 Modify-Assignment . . . 16
8.4 Simple-Segment-Tree . . . 18

9 Strings **18**
9.1 Manachar . . . 18
9.2 Prefix-Function . . . 18
9.3 Z-Function . . . 18

10 Tree **19**
10.1 Centroid-Decomposition . . . 19
10.2 LCA . . . 19

11 Trie **19**
11.1 Binary-Trie . . . 19
11.2 String-Trie . . . 20

1 Bit-Manipulations

1.1 Bit-Manipulations

```
// 1) Flipping all the bits of a number :- input :- 9
    <-- 1001, output 6 <-- 110
ll invert_bits(ll invbit)
{
    if (invbit == 0)
        return 1;
    ll llog = log2l(invbit) + 1ll;
    ll all_set_bits = (1ll << llog) - 1ll;
    return invbit ^ all_set_bits;
}

// 3) Compute XOR from 1 to n :-
ll computeXOR_1ton(ll n)
{
    if (n % 4ll == 0ll)
        return n;
    if (n % 4ll == 1ll)
        return 1ll;
    if (n % 4ll == 2ll)
        return n + 1ll;
    else
        return 0ll;
}

// 4) If x is power of 2 :-
bool is_Power_Of_Two(ll x)
{
    // First x in the below expression is for the case
        when x is 0.
    return x && (!(x & (x - 1)));
}

// 6) Change all even bits to 0 :-
ll convert_Even_Bit_To_Zero(ll n)
{
    return (n & 0xaaaaaaaaaaaaaaaa);
}

// 8) Bitwise & from a to b :-
// If you want bitwise & of range [a..b] then it is
    just the common prefix of binary of a and b
// eg. a=1011011 and b=1010100 then bitwise & of range
    [a..b] will be 1010000.
ll ANDinRange(ll a, ll b)
{
        ll shiftcount = 0;
        while(a != b and a > 0) {
            shiftcount++;
            a = a >> 1ll;
            b = b >> 1ll;
    }
    return (a << shiftcount);
}

ll __log2l(long long x)
{
    return 64 - __builtin_clzll(x) - 1;
}

ll nextpowerof2(long long x)
{
    if(x==0ll) return 1;
    if(x==1ll) return 1;
    return (1ll<<(__log2l(x-1ll) + 1ll));
}

ll prevpowerof2(long long x)
{
    if(x==0ll) return 0;  //actually not applicable to
        0, so just returning 0
    if(x==1ll) return 1;

    return (1ll<<__log2l(x));
}
```

1.2 Iterate-Submasks

```
// Code :-
/////////////////////////////////////////////////////////////

ll nlen = (1<<n);
vll mask_value(nlen);

f(i,0,nlen)
{
    f(j,0,n) {
        if(i&(1ll<<j))
            mask_value[i] += inp[j]; ///inp is an array
                of n elements , ///operation can be
                different +,^,% etc...
    }
}
///Iterating through all masks with their submasks.
    Complexity O(3^n)
dp[0]=0;
f(i,1,nlen)
{
    ll j=i;
    while(j)
    {
        ///do something
        j = (j-1)&i;
    }
}
```

2 DP

2.1 Digit-DP

```
// Code :-
/////////////////////////////////////////////////////////////
vll dig;

ll dp[22][4][10];

void precomputation()
{
    ms(dp, 0);
    /// Initialize all dp values here
}

ll func(ll num)
{
    num++;
    dig.clear();
```

```
    while (num)
        dig.pb(num % 10), num /= 10;

    reverse(all(dig));
    ///--> If num is string then vector dig was not
        required.

    ll nn = dig.size();

    ll ans = 0;
    ll coun = 3;

    /// Do something here

    return ans;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    ll n, q, A, B;
    cin >> q;

    precomputation();
    while (q--)
    {
        cin >> A >> B;
        ll funB = func(B);
        ll funA = func(A - 1);
        ll ans = funB - funA;

        cout << ans << "\n";
    }
    return 0;
}
```

2.2 Divide-Conquer-DP

```
int k, n;
//vector<long long> dp_before, dp_cur;
ll dp_before[5005],dp_cur[5005];
//vll inp;
ll inp[5005];
//vvll Cc;
ll Cc[5005][5005];

void precomputation()
{
    //Cc.resize(n,vll(n));
    f(i,0,n)
    {
        Cc[i][i] = inp[i];
        f(j,i+1,n)
        {
            Cc[i][j] = Cc[i][j-1]|inp[j];
        }
    }
}

/*long long C(int i, int j)          //If needed then
    make the function
{
    return Cc[i][j];
}*/

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {-1ll, -1};        //If
        min is used then take Inf in first place, Also
        take care of the datatype !!!

    int laast = min(mid, optr);
    for (int kkk = optl; kkk <= laast; kkk++) {
        best = max(best, {(kkk ? dp_before[kkk - 1] :
            0) + Cc[kkk][mid], kkk});  //Take care
            about max or min
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
    return;
}

ll solve() {

    cin >> n >> k;
    //inp.resize(n);
    input(inp,0,n);

    precomputation();
    //dp_before.resize(n);
    //dp_cur.resize(n);

    for (int i = 0; i < n; i++)
        dp_before[i] = Cc[0][i];

    for (int i = 1; i < k; i++) {
        compute(0, n - 1, 0, n - 1);
        f(j,0,n) dp_before[j] = dp_cur[j];
    }

    //dp_before.clear();
    //dp_cur.clear();
    //inp.clear();
    //Cc.clear();

    return dp_before[n - 1];
}
// Note :- In the code the precomputation is used if
    you have to make the Cc 2d array. Otherwise you
    can make function of it also.
```

2.3 Knuth-Optimization

```
// (dp[i][j] = min{i < k < j}(dp[i][k] + dp[k][j]) +
    C[i][j]):-
const ll inf = 1e18;
#define SIZE 5005

ll dp[SIZE][SIZE];
int p[SIZE][SIZE];
```

```
int arr[SIZE];
ll prefix_arr[SIZE];
int m;

void precomputation()
{
    prefix_arr[0]=0ll;
    f(i,0,m)
    {
        prefix_arr[i+1] = prefix_arr[i] + (ll)(arr[i]);
    }
}

ll Cost(int i,int j)
{
    return prefix_arr[j+1] - prefix_arr[i];
}

ll solve()
{
    cin >> m;

    f(i,0,m)
    {
        cin >> arr[i];
    }

    precomputation();

    for(int i=0;i<m;i++)
    {
        dp[i][i]=0ll;
        p[i][i]=i;
    }

    for(int i=1;i<m;i++)
    {
        for(int j=0;j+i < m ;j++)
        {
            dp[j][j+i] = inf;
            for(int k=p[j][i+j-1];k<=p[j+1][i+j];k++)
            {
                ll temp = dp[j][k] + dp[k+1][i+j] +
                    Cost(j,i+j);
                if(temp < dp[j][j+i])
                {
                    dp[j][j+i] = temp;
                    p[j][j+i] = k;
                }
            }
            //cout << j << ' ' << i+j << ' ' <<
                dp[j][i+j] << "\n";
        }
    }
    return dp[0][m-1];
}
```

2.4 SOS

```
///Iterative Version :-
for(int mask = 0; mask < (1<<N); ++mask)
{
    dp[mask][0] = A[mask];    //handle base case
        separately (leaf states)
    for(int i = 1;i <= N; ++i){
        if(mask & (1<<(i-1)))
            dp[mask][i] = dp[mask][i-1] +
                dp[mask^(1<<(i-1))][i-1];
        else
            dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N];
}
// Note :- Used when two functions depends on each
    other.

// Code(Fully optimized way) :-
for(int i = 0; i<(1<<N); ++i)
    F[i] = A[i];

for(int i = 0;i < N; ++i)
{
    for(int mask = 0; mask < (1<<N); ++mask){
        if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
    }
}
```

3 Graph

3.1 Articulation-Point

```
#define pb push_back

const int N = 1e5 + 10;
vector<int> res;
vector<int> g[N];
int vis[N];
int tin[N];
int mx[N];
int tim = 0;

void dfs(int node, int par = -1)
{
    vis[node] = 1;
    tin[node] = mx[node] = tim++;
    int child = 0;
    int flag = 0;
    for (auto x : g[node])
    {
        if (x == par)
            continue;
        if (!vis[x])
        {
            dfs(x, node);
            if (mx[x] >= tin[node] && par != -1)
                flag = 1;
            child++;
        }
        mx[node] = min(mx[node], mx[x]);
    }
    if (par == -1 && child > 1)
        flag = 1;
    if (flag)
        res.pb(node);
}
```

3.2 Bridge-In-Graph

```
const int N = 1e5 + 10, M = 1e6;
int tim = 0;
int u[N], v[N], vis[N];
int tin[N], tout[N], isBridge[M], minAncestor[N];
vector<pair<int, int>> g[N]; // vertex, index of edge

void dfs(int k, int par)
{
    vis[k] = 1;
    tin[k] = ++tim;
    minAncestor[k] = tin[k];
    for (auto it : g[k])
    {
        if (it.first == par)
            continue;
        if (vis[it.first])
        {
            minAncestor[k] = min(minAncestor[k],
                tin[it.first]);
            continue;
        }
        dfs(it.first, k);
        minAncestor[k] = min(minAncestor[k],
            minAncestor[it.first]);
        if (minAncestor[it.first] > tin[k])
            isBridge[it.second] = 1;
    }
    tout[k] = tim;
}
```

3.3 Check-Bipartite-Online-DSU

```
// ***--> Support the parity of the path length /
    Checking bipartiteness Online
//     --> Means it can check that if you gonna add
    this edge then is the graph remains bipartite or
    not.
///0 indexed

class DSU{

public:
    ll n;
    vpll parent;
    vll at_height;
    vll bipartite;

    DSU(){};

    DSU(ll n):n(n)
    {
        at_height.resize(n,0);
        parent.resize(n);
        bipartite.resize(n,1);
        for(int i=0;i<n;i++)
            parent[i]=make_pair(i, 0);

    }

    void initialize(ll nn)
    {
        n=nn;
        at_height.resize(n,0);
        bipartite.resize(n,1);
        parent.resize(n);
        for(int i=0;i<n;i++)
            parent[i]=make_pair(i, 0);

    }

    pair<ll, ll> find_root(ll v) {
        if (v != parent[v].first) {
            ll parity = parent[v].second;
            parent[v] = find_root(parent[v].first);
            parent[v].second ^= parity;
        }
        return parent[v];
    }

    tell_color_for_bipartiteness(ll v) ///We can get
        the color(0 or 1) if the graph containing node
        'v' is bipartite.
    {
        ///Only works if graph is bipartite.
        return (find_root(v).se);
    }

    void merge_(ll a, ll b) {
        pair<ll, ll> pa = find_root(a);
        a = pa.first;
        ll x = pa.second;

        pair<ll, ll> pb = find_root(b);
        b = pb.first;
        ll y = pb.second;

        if (a == b) {
            if (x == y)
                bipartite[a] = false; //So by adding
                    edge between a and b graph becomes
                    non-bipartite.
        } else {
            if (at_height[a] < at_height[b])
                swap (a, b);
            parent[b] = make_pair(a, x^y^1);
            bipartite[a] &= bipartite[b];
            if (at_height[a] == at_height[b])
                ++at_height[a];
        }
    }

    bool is_bipartite(int v) {
        return bipartite[find_root(v).first];
    }
};
```

3.4 DFS-Start-End-Time

```
// 1) Starting time, finishing time
/////////////////////////////////////////////////////////////

ll timer = 0;
ll st[N]; //Starting time
ll ft[N]; //Finishing time

void dfs(ll v,vvll &adj,bool *visited)
{
    st[v] = timer++;
```

```
    visited[v] = true;
    //cout << v << ' ';
    for (ll u : adj[v])
    {
        if (!visited[u])
            dfs(u,adj,visited);
    }
    ft[v] = timer++;
}

// Note :- Lemma: If we run dfs(root) in a rooted tree,
    then v is an ancestor of u if and only if st[v] <=
    st[u] <= ft[u] <= ft[v].
/// So, given arrays st and ft we can rebuild the tree
    also.
```

3.5 Dijkstra

```
vector<int> dijkstra(vector<vector<pair<int,int>>>
    &adj, int src) {
        int A = adj.size();
        vector<int> dist(A, LONG_LONG_MAX);
        vector<bool> visited(A);
        priority_queue<pair<int,int>> dists;
        dists.push({0, src});
    dist[src] = 0;

        while (dists.size()) {
                pair<int,int> p = dists.top();
                dists.pop();
                if (visited[p.second]) continue;
                visited[p.second] = true;

                for (auto vd: adj[p.second]) if
                    (!visited[vd.first]) {
                        if (dist[vd.first] >
                            dist[p.second]+vd.second) {
                                dist[vd.first] =
                                    dist[p.second]+vd.second;
                                dists.push({-dist[vd.first],
                                    vd.first});
                        }
                }
        }
        return dist;
}
```

3.6 DSU

```
class DSU { public:
    ll n;
    vll parent,siz;

    DSU(){};
    DSU(ll n):n(n) {
        siz.resize(n,1);
        parent.resize(n);
        iota(parent.begin(),parent.end(),0);
    }

    void initialize(ll nn) {
        n=nn;
        siz.resize(n,1);
        parent.resize(n);
        iota(parent.begin(),parent.end(),0);
    }

    ll find_root(ll v) {
        if (v == parent[v])
            return v;
        return parent[v] = find_root(parent[v]);
    }

    ll give_size(ll v) {
        return siz[find_root(v)];
    }

    void merge_(ll a,ll b) {
        a = find_root(a);
        b = find_root(b);
        if (a != b) {
            if (siz[a] < siz[b])
                swap(a, b);
            parent[b] = a;
            siz[a]+=siz[b];
        }
    }
};
///merge_(u,v) ///to join node u and v.
///siz[i] gives us the no. of nodes in the subtree of
    node i(containing itself).
///find_root(i) gives you the top(root) node of the
    tree consisting i.

///So, siz[find_root(i)] will give you the no. of nodes
    in the tree containing node i,(or you can call the
    function give_size(i)).
```

3.7 Floyd-Warshall

```
// Floyd-Warshall Algorithm
f(k,0,n)
{
    f(i,0,n)
    {
        f(j,0,n)
        {
            dp[i][j] = min(dp[i][j],dp[i][k] + dp[k][j]);
        }
    }
}
```

3.8 Maximum-Bipartite-Matching

```
// Code :-

/// 0 - indexed.

class matching
{
public:
    vector<vll> g;
    vector<ll> pa;
    vector<ll> pb;
    vector<ll> was;
    int n, m;
```

```
int res;
int iter;

matching(ll _n, ll _m) : n(_n), m(_m)
{
    assert(0 <= n && 0 <= m);
    pa = vector<ll>(n, -1);
    pb = vector<ll>(m, -1);
    was = vector<ll>(n, 0);
    g.resize(n);
    res = 0;
    iter = 0;
}

void add(ll from, ll to)
{
    assert(0 <= from && from < n && 0 <= to && to <
        m);
    g[from].push_back(to);
}

bool dfs(ll v)
{
    was[v] = iter;
    for (int u : g[v])
    {
        if (pb[u] == -1)
        {
            pa[v] = u;
            pb[u] = v;
            return true;
        }
    }
    for (ll u : g[v])
    {
        if (was[pb[u]] != iter && dfs(pb[u]))
        {
            pa[v] = u;
            pb[u] = v;
            return true;
        }
    }
    return false;
}

int solve()
{
    while (true)
    {
        iter++;
        int add = 0;
        for (ll i = 0; i < n; i++)
        {
            if (pa[i] == -1 && dfs(i))
            {
                add++;
            }
        }
        if (add == 0)
        {
            break;
        }
        res += add;
    }
    return res;
}

    int run_one(ll v)
    {
        if (pa[v] != -1)
        {
            return 0;
        }
        iter++;
        return (ll)dfs(v);
    }
};

// Note :- n corresponds to number of elements in 1st
    set.m corresponds to number of elements in 2nd set.
// After adding all edges using "edge()" function, call
    solve() function to get the maximum bipartite
    matching
```

3.9 Reachability-Tree

```
// ***---> Kruskal Reconstruction Tree(KRT)/
    Reachability Tree / DSU Tree.
//     To get read-only access to all the versions of
    DSU. (but not give permission to merge_ at some
    point of time in past structure ---> for that use
    Persistant DSU).

//     Idea :-
//     Each time when we merge 2 components in DSU,
    create a new node with directed edges with bith
    the nodes.

    ///Code :-

    const ll N=2e5+5;
    const ll M=3e5+5;
    // ll inp[N];
    vll adj[N+M];
    ll parent[N+M];

    void ok_boss()
    {
            ll n,m,qq;
            cin >> n>>m;

            f(i,0,n+m+1)
                    parent[i]=i;

            ll nn=n;
            ll fir,sec;

            function<ll(ll)> find_root = [&](ll
                node){
            if(parent[node]==node)
                    return node;

            return
                parent[node]=find_root(parent[node]);
            };

            function<ll(ll,ll)> merge_ = [&](ll
                node1,ll node2)
            {
                    node1=find_root(node1);
                    node2=find_root(node2);
```

```
                        parent[node1]=nn;
                        parent[node2]=nn;

                        adj[nn].pb(node1);
                        if(node1!=node2)
                                adj[nn].pb(node2);

                        nn++;

                        return 0;
                };

                f(i,0,m)
                {
                        cin>>fir>>sec;
                        fir--,sec--;

                        merge_(fir,sec);
                }

                return;
        }
```

4 Kosaraju

4.1 2-SAT

```
// 2) At Most One (Means at most 1 is true in a given
    set.):-

// Code :-
/////////////////////////////////////////////////////////////
struct TwoSat
{
    int NN;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

    TwoSat(int n = 0) : NN(n), gr(2 * n) {}

    int addVar()
    { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return NN++;
    }

    void add_clause_or(int f, int j)
    {
        f = max(2 * f, -1 - 2 * f);
        j = max(2 * j, -1 - 2 * j);
        gr[f].push_back(j ^ 1);
        gr[j].push_back(f ^ 1);
    }

    void add_clause_xor(int f, int j)
    {
        add_clause_or(f, j);
        add_clause_or(~f, ~j);
    }

    void setValue(int x) { add_clause_or(x, x); }

    void atMostOne(const vi &li)
    {
        if (sz(li) <= 1)
            return;
        int cur = ~li[0];
        f(i, 2, sz(li))
        {
            int next = addVar();
            add_clause_or(cur, ~li[i]);
            add_clause_or(cur, next);
            add_clause_or(~li[i], next);
            cur = ~next;
        }
        add_clause_or(cur, ~li[1]);
    }

    vi val, comp, z;
    int time = 0;
    int dfs(int i)
    {
        int low = val[i] = ++time, x;
        z.push_back(i);
        for (ll e : gr[i])
            if (!comp[e])
                low = min(low, val[e] ?: dfs(e));
        if (low == val[i])
            do
            {
                x = z.back();
                z.pop_back();
                comp[x] = low;
                if (values[x >> 1] == -1)
                    values[x >> 1] = x & 1;
            } while (x != i);
        return val[i] = low;
    }

    bool solve()
    {
        values.assign(NN, -1);
        val.assign(2 * NN, 0);
        comp = val;
        f(i, 0, 2 * NN) if (!comp[i]) dfs(i);
        f(i, 0, NN) if (comp[2 * i] == comp[2 * i + 1])
            return 0;
        return 1;
    }
};

/// The answer is stored in values vector.
/// Let say in the set {a,b,c,d,e,f,g} at most 1 is
    true , then make a vector consisting this and call
    function atmost one.
/// for negation of a simply write (neg a).

// Note :- For negation of a, use (neg a) in the code
```

4.2 Strongly-Connected-Components

```
// --> It is just an implementation of "Finding
    Strongly connected components ", with some changes.
// --> For more info, read it in the Antii-Lakinson
    Book OR from cp-algorithms.

// 1)

// Code :-
```

```cpp
/////////////////////////////////////////////////////////////
/// The code is 0 indexed.
struct two_sat
{
    int n;
    vector<vector<int>> g, gr;              // gr is the reversed graph
    vector<int> comp, topological_order, answer; // comp[v]: ID of the SCC containing node v
    vector<bool> vis;

    two_sat() {}

    two_sat(int _n) { init(_n); }

    void init(int _n)
    {
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        comp.resize(2 * n);
        vis.resize(2 * n);
        answer.resize(2 * n);
    }

    void add_edge(int u, int v)
    {
        g[u].push_back(v);
        gr[v].push_back(u);
    }

    // For the following three functions
    // int x, bool val: if 'val' is true, we take the variable to be x. Otherwise we take it to be x's complement.

    // At least one of them is true
    void add_clause_or(int i, bool f, int j, bool g)
    {
        add_edge(i + (f ? n : 0), j + (g ? 0 : n));
        add_edge(j + (g ? n : 0), i + (f ? 0 : n));
    }

    // Only one of them is true
    void add_clause_xor(int i, bool f, int j, bool g)
    {
        add_clause_or(i, f, j, g);
        add_clause_or(i, !f, j, !g);
    }

    // Both of them have the same value
    void add_clause_nxor(int i, bool f, int j, bool g)
    {
        add_clause_xor(i, !f, j, g);
    }

    // Topological sort
    void dfs(int u)
    {
        vis[u] = true;

        for (const auto &v : g[u])
            if (!vis[v])
                dfs(v);

        topological_order.push_back(u);
    }

    // Extracting strongly connected components
    void scc(int u, int id)
    {
        vis[u] = true;
        comp[u] = id;

        for (const auto &v : gr[u])
            if (!vis[v])
                scc(v, id);
    }

    // Returns true if the given proposition is satisfiable and constructs a valid assignment
    bool satisfiable()
    {
        fill(vis.begin(), vis.end(), false);

        for (int i = 0; i < 2 * n; i++)
            if (!vis[i])
                dfs(i);

        fill(vis.begin(), vis.end(), false);
        reverse(topological_order.begin(),
            topological_order.end());

        int id = 0;
        for (const auto &v : topological_order)
            if (!vis[v])
                scc(v, id++);

        // Constructing the answer
        for (int i = 0; i < n; i++)
        {
            if (comp[i] == comp[i + n])
                return false;
            answer[i] = (comp[i] > comp[i + n] ? 1 : 0);
        }

        return true;
    }
};
/////////////////////////////////////////////////////////////
// Note :- 2SAT problem for OR is given in the source code.

// Note :- For 2SAT problem for XOR see question "The_Door_Problem_doors_controlled_by_n_switches_2SAT_with in codeforces.
// Note :- ***For XOR instead of 2SAT, you can also use "Check if graph is bipartite or not" ***.

// ///One other thing ~(a^b) == (~a^b)
```

5 Main-Template

5.1 Main

```cpp
#include<bits/stdc++.h>
using namespace std;
#define fastio() ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define f(i_itr, a, n) for (ll i_itr = a; i_itr < n; i_itr++)
```

```
#define ai(n, arr) f(i,0,n) cin>>arr[i]
#define matin(n, m, mat) f(i,0,n) f(j,0,m)
    cin>>mat[i][j]
#define all(x) x.begin(), x.end()
#define nl "\n"
#define pa(n, arr) f(i,0,n) cout<<arr[i]<<' ';
    cout<<"\n"
#define rev_f(i_itr, n, a) for (ll i_itr = n; i_itr >
    a; i_itr--)
#define pb push_back
#define fi first
#define se second
#define ms(a, val) memset(a, val, sizeof(a))
#define cty cout << "YES" << nl
#define ctn cout << "NO" << nl
#define lmax LLONG_MAX
#define lmin LLONG_MIN
#define sz(v) (v).size()
#define c(x) cout << (x)
#define csp(x) cout << (x) << " "
#define c1(x) cout << (x) << nl
#define c2(x, y) cout << (x) << " " << (y) << nl
#define c3(x, y, z) cout << (x) << " " << (y) << " " <<
    (z) << nl
#define c4(a, b, c, d) cout << (a) << " " << (b) << " "
    << (c) << " " << (d) << nl

typedef long long int ll;
#define int long long
typedef long double ld;
typedef std::vector<ll> vll;
typedef std::pair<ll, ll> pll;

void solve() {

}

signed main(){
   fastio();
   // cout << fixed << setprecision(15); // activate
       if the answers are in decimal
   int t;
   cin>>t;
   while (t--) solve();
}
```

5.2 Others

```
// ordered set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<ll, null_type, less<ll>,
    rb_tree_tag,tree_order_statistics_node_update>

// execution time
using namespace std::chrono;

auto start = high_resolution_clock::now();
// something
auto stop = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(stop-start);
cout << duration.count();
```

6 Math

6.1 Euclid

```
#include<bits/stdc++.h>
using namespace std;
#define mod(x, y) ((x)%(y))
typedef vector<int> VI;
typedef pair<int, int> PII;

// This is a collection of useful code for solving
    problems
// that
   // involve modular linear equations. Note that all
       of the
   // algorithms described here work on nonnegative
       integers.
   // returns g = gcd(a, b); finds x, y such that d =
       ax + by
int extended_euclid(int a, int b, int &x, int &y)
{
   int xx = y = 0;
   int yy = x = 1;
   while (b)
   {
      int q = a / b;
      int t = b;
      b = a % b;
      a = t;
      t = xx;
      xx = x - q * xx;
      x = t;
      t = yy;
      yy = y - q * yy;
      y = t;
   }
   return a;
}
// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n)
{
   int x, y;
   VI ret;
   int g = extended_euclid(a, n, x, y);
   if (!(b % g))
   {
      x = mod(x * (b / g), n);
      for (int i = 0; i < g; i++)
         ret.push_back(mod(x + i * (n / g), n));
   }
   return ret;
}
// computes b such that ab = 1 (mod n), returns-1 on
    failure
int mod_inverse(int a, int n)
{
   int x, y;
   int g = extended_euclid(a, n, x, y);
   if (g > 1)
      return -1;
   return mod(x, n);
}
// Chinese remainder theorem (special case): find z
    such that

   // z % m1 = r1, z % m2 = r2. Here, z is unique
       modulo M = lcm(m1, m2).
```

```
    // Return (z, M). On failure, M =-1.
    // Team
    // - Pay Attention - Dhirubhai ambani institute of
        information and communication technology,
    // Gandhinagar 2
PII chinese_remainder_theorem(int m1, int r1, int m2,
     int r2)
{
    int s, t;
    int g = extended_euclid(m1, m2, s, t);
    if (r1 % g != r2 % g)
        return make_pair(0, -1);
    return make_pair(mod(s * r2 * m1 + t * r1 * m2, m1
        * m2) /
                     g,
                 m1 * m2 / g);
}
// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i. Note that the solution is
// unique modulo M = lcm_i (m[i]). Return (z, M). On
// failure, M =-1. Note that we do not require the a[i]s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &m, const VI &r)
{
    PII ret = make_pair(r[0], m[0]);
    for (int i = 1; i < m.size(); i++)
    {
        ret = chinese_remainder_theorem(ret.second,
             ret.first, m[i], r[i]);
        if (ret.second == -1)
            break;
    }
    return ret;
}
// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x,
     int &y)
{
    if (!a && !b)
    {
        if (c)
            return false;
        x = 0;
        y = 0;
        return true;
    }
    if (!a)
    {
        if (c % b)
            return false;
        x = 0;
        y = c / b;
        return true;
    }
    if (!b)
    {
        if (c % a)
            return false;
        x = c / a;
        y = 0;
        return true;
    }
    int g = gcd(a, b);
    if (c % g)
        return false;
    x = c / g * mod_inverse(a / g, b / g);
    y = (c - a * x) / b;
    return true;
}

int main()
{
    // expected: 2
    cout << gcd(14, 30) << endl;
    // expected: 2-2 1
    int x, y;
    int g = extended_euclid(14, 30, x, y);
    cout << g << " " << x << " " << y << endl;
    // expected: 95 451
    VI sols = modular_linear_equation_solver(14, 30,
        100);
    for (int i = 0; i < sols.size(); i++)
        cout << sols[i] << " ";
    cout << endl;
    // expected: 8
    cout << mod_inverse(8, 9) << endl;
    // expected: 23 105
    //
    PII ret = chinese_remainder_theorem(VI({3, 5, 7}),
        VI({2, 3, 2}));
    cout << ret.first << " " << ret.second << endl;
    ret = chinese_remainder_theorem(VI({4, 6}), VI({3,
        5}));
    cout << ret.first << " " << ret.second << endl;
    // expected: 5-15
    if (!linear_diophantine(7, 2, 5, x, y))
        cout << "ERROR" << endl;
    cout << x << " " << y << endl;
    return 0;
}
```

6.2 Euler-Totient

```
// Total coprime numbers to n
int phi(int n) {
    int ans =n;
    for (int f: prime_factors(n)) {ans *= f-1; ans/=f;}
    return ans;
}

// Total coprime numbers to n below m
int phi(int n, int m) {
    if (m>n) return (m/n)*phi(n, n) + phi(n, m-n*(m/n));

    vector<int> primef = prime_factors(n);
    int ans = 0;
    f(i, 0, pow(2, primef.size())) {
        int j = i, num=m, idx=0, totones=0;
        while (j) {
            if (j&1) {
                num /= primef[idx];
                totones++;
            }
            j>>=1;
            idx++;
        }
        if (totones%2) ans -= num;
        else ans += num;
    }
    return ans;
}
```

6.3 Nck-Npk

```
const ll N=2e5+5;
const int fac_len=524288+5;
int fact[fac_len+1], invfact[fac_len+1];

int nck(int n, int k) {
    if (n < k) return 0;
    if (!k) return 1;
    return
        (((fact[n]*invfact[k])%mod)*invfact[n-k])%mod;
}

int npk(int n, int k) {
    return (nck(n, k)*fact[k])%mod;
}

void factorials_precompute() {
    fact[0] = 1;
    f(i, 0, fac_len) {
        fact[i+1] = (fact[i]*(i+1))%mod;
    }
    invfact[fac_len] = modInv(fact[fac_len]);
    for (int i=fac_len-1; i>=0; i--) {
        invfact[i] = (invfact[i+1]*(i+1))%mod;
    }
}
```

6.4 Power-And-Modulo

```
const ll mod = 1000000007; // 998244353
const long double pi = 3.141592653589793238;

ll pct(ll x) { return __builtin_popcount(x); } // #of
    set bits

int power(int x, int y) {
    int res = 1;
    x = x % mod;
    while (y > 0) {
        if (y&1)
            res = (res * x) % mod;
        y = y>>1;
        x = (x * x) % mod;
    }
    return res%mod;
}
ll poww(ll a, ll b)
{
    ll res = 1;
    while (b)
    {
        if (b & 1)
            res = (res * a);
        a = (a * a);
        b >>= 1;
    }
    return res;
}
int modInv(int n, int mod = mod) {return power(n,
    mod-2);}
```

6.5 Primes

```
const int maxn = 1e6;
int spf[maxn];
vector<int> primes;

// 1 - 1e7
void precompute() {
    f(i, 2, maxn) spf[i] = i;
    f(i, 2, maxn) if (spf[i]==i) for (int j=i*i;
        j<maxn; j+=i) if (spf[j]==j) spf[j] = i;
}
vector<int> prime_factors(int n) { // prime_factors(72)
    = {2, 3}
    vector<int> ans;
    while (n!=1) {
        if (!ans.size() || ans.back()!=spf[n])
            ans.push_back(spf[n]);
        n /= spf[n];
    }
    return ans;
}

// 1e7 - 1e9
void precompute() {
    int M = 1e6;
    bool siv[M];
    memset(siv, true, sizeof(siv));
    for (int i=2; i<M; i++) {
        if (siv[i]) {
            for (int j=i*i; j<M; j+=i) siv[j] = false;
        }
    }
    for (int i=2; i<M; i++) if (siv[i])
        primes.push_back(i);
}
vector<int> prime_factors(int n) { // prime_factors(72)
    = {2, 3}
    int n1 = n;
    vector<int> ans;
    for (int p: primes) {
        if (!(n%p)) {
            ans.push_back(p);
            while (!(n%p)) n /= p;
        }
        if (p*p > n1) break;
    }
    if (n!=1) ans.push_back(n);
    return ans;
}

// O(sqrt(n))
vector<int> getfactors(int n) {
    vector<int> ans;
    int i;
    for (i=1; i*i<n; i++) if (!(n%i)) {
        ans.push_back(i);
        ans.push_back(n/i);
    }
    if (i*i == n) ans.push_back(i);
    return ans;
}
```

7 Miscellaneous

7.1 Catalan-Numbers

```
long long catalan(ll n)
{
    return (calcnCr(2*n,n)*power(n+1,mod-2,mod))%mod;
}
```

7.2 Comments

```
// Eulerian tours
// It is quite like DFS, with a little change :

// vector E
// dfs (v):
//        color[v] = gray
//        for u in adj[v]:
//               erase the edge v-u and dfs(u)
//        color[v] = black
//        push v at the end of e
// e is the answer.

//Eulerian Tour is only possible when no. of nodes with
    odd degree is <= 2.(Start with node with odd
    degree if there is one.)


//Random Generator
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().
    count());
int getRand(int l, int r) {
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}


// Given a tree and a subset of vertices v1,v2,...,vm.
    Let's reorder them in the (preorder) DFS order.
    For any i and j with vi!=vj ,
    lca(vi,vj)=lca(vk,v[k+1]) for some k. In other
    words: the lowest common ancestor of any two
    vertices in the array can be represented as the
    lowest common ancestor of two adjacent vertices.

// Consider a permutation p with length n. At least one
    of the following is true:
// There exists an increasing subsequence with length
    root(n).
// There exists a decreasing subsequence with length
    root(n).


// Adding a new item is classical:
// # we go from large to small so that the already
    updated dp values won't affect any calculations
for (int i = dp.size() - 1; i >= weight; i--) {
    dp[i] += dp[i - weight];
}
// To undo what we just did, we can simply do
    everything backwards.

// # this moves the array back to the state as it was
    before the item was added
for (int i = weight; i < dp.size(); i++) {
    dp[i] -= dp[i - weight];
}
```

```
// Problem : Handle queries of type: answer with 2 most
    distant nodes that are inside segment [l,r] of
    array a, where a is some permutation of nodes from
    1 to n.
// Solution : At each node of segment tree store the
    merged diameter of nodes contained in that segment
    (only 2 most distant nodes a and b), merging two
    ranges [l,r) and [l2,r2) can be done in O(logn)
    (as explained in above problem, consider each pair
    of a b c d where a and b are most distant nodes in
    left node and c and d are most distant nodes in
    right node) and there is also O(logn) ranges when
    querying the segment tree.Complexity: O(qlog^2(n)).

// mobius function
// {
//     A Mobius function is number theory function
    which is defined as
//     mu(1) = 1
//     mu(n) = 0 If n has one or more than one
    repeated factors
//     mu(n) = (-1)^k , If n is product of k distinct
    prime numbers
// }

const ll N = 2e5 + 5;

ll mob[N];

void mobius()
{
    mob[1] = 1;
    for (ll i = 2; i < N; i++)
    {
        mob[i]--;
        for (ll j = i + i; j < N; j += i)
        {
            mob[j] -= mob[i];
        }
    }
}

// dirx-diry
ll dirx[8] = { -1, 0, 0, 1, -1, -1, 1, 1 };
ll diry[8] = { 0, 1, -1, 0, -1, 1, -1, 1 };

//If the player can move in all its eight directions
    then take a for loop for all these 8 values.
//If the player can move only in 4 directions(adjacent
    to present cell) then take a for loop for first 8
    values.

// Note for knight moving in a board then :-

ll directionx[8]={-2,-2,-1,-1, 1, 1, 2, 2};
ll directiony[8]={-1,1,-2,2,-2,2,-1,1};
```

7.3 Convex-Hull

```
bool compcross(pair<int,int> &v1, pair<int,int> &v2) {
    // true - change
    return v1.first*v2.second > v1.second*v2.first;
}
```

```
int isSafe(int n, vector<vector<int>> &points, int x,
    int y){
    set<pair<int,int>> avail;
    for (int i=0; i<n; i++) avail.insert({points[i][0],
        points[i][1]});
    pair<int,int> pt = *avail.begin();

    pair<int,int> pt1 = pt, pt2, minpt, minvec, vec;
    do {
        pt2 = *avail.rbegin();
        minvec = {pt2.first-pt1.first,
            pt2.second-pt1.second};
        minpt = pt2;
        for (auto pt2: avail) {
            vec = {pt2.first-pt1.first,
                pt2.second-pt1.second};
            if (compcross(vec, minvec)) {
                minvec = vec;
                minpt = pt2;
            }
        }
        vec = {x-pt1.first, y-pt1.second};
        if (compcross(vec, minvec)) return 0;
        pt1 = minpt;
        avail.erase(pt1);
    }
    while (pt!=pt1);
    return 1;
}
```

7.4 Coordinate-Compression

```
template<class TT>
void coordinate_compressor(vector<TT> &vec)
{
    int n = vec.size();
    vector<pair<TT,int>> ranking_elements(n);

    for(int jj=0;jj<n;jj++)
        ranking_elements[jj] = {vec[jj],jj};

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

    TT nxt = 0; ///If you want to start from 1 then put
        1 here.

    vec[ranking_elements[0].second] = nxt;

    for(int i = 1; i < n; ++i) {
        if(ranking_elements[i-1].first !=
            ranking_elements[i].first) nxt++;
            //nxt++       ///If you want all
                elements different then turn this on
                ans remove the upper line.

        vec[ranking_elements[i].second] = nxt;
    }
}
```

7.5 Matrix-Exponentian

```
const int SZ = 2;
struct matrix
{
        ll arr[SZ][SZ];
        matrix(){
                reset();
        }

        void reset(){
                memset(arr, 0, sizeof(arr));
        }

        void makeiden(){
                reset();
                for(int i=0;i<SZ;i++){
                        arr[i][i] = 1;
                }
        }

        matrix operator + (const matrix &o) const
        {
                matrix res;
                for(int i=0;i<SZ;i++)
                {
                        for(int j=0;j<SZ;j++)
                        {
                                res.arr[i][j] = (arr[i][j]
                                    + o.arr[i][j])%mod;
                        }
                }
                return res;
        }

        matrix operator * (const matrix &o) const
        {
                matrix res;
                for(int i=0;i<SZ;i++)
                {
                        for(int j=0;j<SZ;j++)
                        {
                                res.arr[i][j] = 0;
                                for(int k=0;k<SZ;k++)
                                {
                                        res.arr[i][j] =
                                            (res.arr[i][j]
                                            + ((arr[i][k]
                                            *
                                            o.arr[k][j])%mod))%mod
                                }
                        }
                }
                return res;
        }
};

matrix mat_power(matrix a, ll b)
{
        matrix res;
        res.makeiden();
        while(b)
        {
                if(b & 1)
                {
                        res = res * a;
                }
                a = a * a;
                b >>= 1;
        }
```

```
        return res;
}

ll G(ll term,ll mod)
{

        ll G0=1,G1=3;  //0th element and 1st element

        if(term==0)
                return G0%mod;

        if(term==1)
                return G1%mod;

        matrix matt;  //matrix corresponding to the
            recurrance relation
        matt.arr[0][0] = 2,matt.arr[0][1] = 2;
        matt.arr[1][0] = 1,matt.arr[1][1] = 0;

        /*

                [a,b] [fib[n]] ***:- Be careful take the
                    matrix left side.
                [c,d] [fib[n-1]      ***

        */

        matrix final_mat = mat_power(matt,term-1);

        return (G1*final_mat.arr[0][0] +
            G0*final_mat.arr[0][1] + mod)%mod ;
}
```

7.6 MO-Algorithm

```
// --> We are just using the sqrt decomposition method
    here but first we are sorting the queries(thus it
    should be offline)
// accordingly as we need such that it optimizes the
    complexity to some (N+Q)*sqrt(N).

// Code :-
const ll block_size = 350; /// define block size
    earlier , (320 to 400) if n <= 1e5 , (420 to 480)
    if n <= 2e5 etc.

struct Query
{
    ll l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
              make_pair(other.l / block_size, other.r);
    }
};

class Mos
{
public:
    void remove_(ll idx)
    {
        // TODO: remove value at idx from data structure
    }

    void add(ll idx)
    {
        // TODO: add value at idx from data structure
    }

    ll get_answer()
    {
        // TODO: extract the current answer of the data
            structure
    }

    vector<ll> mo_s_algorithm(vector<Query> &queries)
    {
        vector<ll> answers(queries.size());
        sort(queries.begin(), queries.end());

        // TODO: initialize data structure

        ll cur_l = 0;
        ll cur_r = -1;
        // invariant: data structure will always
            reflect the range [cur_l, cur_r]
        for (Query q : queries)
        {
            while (cur_l > q.l)
            {
                cur_l--;
                add(cur_l);
            }
            while (cur_r < q.r)
            {
                cur_r++;
                add(cur_r);
            }
            while (cur_l < q.l)
            {
                remove_(cur_l);
                cur_l++;
            }
            while (cur_r > q.r)
            {
                remove_(cur_r);
                cur_r--;
            }
            answers[q.idx] = get_answer();
        }

        return answers;
    }
};
```

8 Segment-Tree

8.1 Fenwick-Tree

```
// point update, range query
struct fenwick_tree { // 0-indexed tree
    vector<int> tree; // tree[i] = sum(arr[g(i):i]),
        g(i) = i&(i+1)
    int n;
    void init(vector<int> arr) {
        n = arr.size();
        tree.resize(n);
        f(i,n) add(i, arr[i]);
    }
    int sum(int r) {
```

```
        int ans = 0;
        for (; r>=0; r=(r&(r+1))-1) ans += tree[r];
        return ans;
    }
    void add(int idx, int v) { // for all g(i)<=idx<=i:
        tree[i]+=v
        for (; idx<n; idx = idx|(idx+1)) tree[idx] += v;
    }
};

// range update, point query
struct fenwick_tree1 {
    vector<int> tree;
    int n;
    void init(vector<int>& arr) {
        n = arr.size();
        tree.resize(n+1);
        f(i,n) range_add(i, i, arr[i]);
    }
    int point_query(int r) {
        int ans = 0;
        for (; r>=0; r=(r&(r+1))-1) ans += tree[r];
        return ans;
    }
    void add(int idx, int v) {
        for (; idx<n; idx = idx|(idx+1)) tree[idx] += v;
    }
    void range_add(int l, int r, int v) {
        add(l, v), add(r+1, -v);
    }
};
```

8.2 Modify-Addition

```
// add value in interval by modify(l, r, val) function
    in [l, r] (note that both inclusive)
// get sum of elements in range by query(l, r)
class seg_tree {
    public:
    int n;
    vector<int> tree;
    vector<int> lazy;
    int operation(int a, int b) {return a+b;}
    seg_tree(vector<int> arr) {
        n = arr.size();
        tree.resize(4 * n);
        lazy.resize(4 * n);
        build(arr, 0, 0, n - 1);
    }
    void build(vector<int>& arr, int node, int start,
        int end) {
        if (start == end) tree[node] = arr[start];
        else {
            int mid = (start + end) / 2;
            build(arr, 2 * node + 1, start, mid);
            build(arr, 2 * node + 2, mid + 1, end);
            tree[node] = operation(tree[2 * node + 1],
                tree[2 * node + 2]);
        }
    }
    int query(int l, int r) {return query(0, 0, n - 1,
        l , r);}
    int query(int node ,int start, int end ,int l ,int
        r) {
        if (lazy[node] != 0) {
            tree[node] += (end - start + 1) * lazy[node];
            if (start != end) {
                lazy[2 * node + 1] += lazy[node];
                lazy[2 * node + 2] += lazy[node];
            }
            lazy[node] = 0;
        }
        if (start > end || start > r || end < l) return
            0;
        if(start >= l && end <= r){
            return tree[node]; // completely within range
        }

        int mid = (start+end)/2;
        int p1 = query(2*node+1,start,mid,l,r);
        int p2 = query(2*node+2,mid+1,end,l,r);
        return p1+p2; // partially within range
    }
    void modify(int l,int r,int val) {modify(0
        ,0,n-1,l,r,val);}
    void modify(int node,int start,int end,int l,int
        r,int val){
        if(lazy[node]!=0){
            tree[node]+=(end-start+1)*lazy[node];
            if(start!=end){
                lazy[2*node+1]+=lazy[node];
                lazy[2*node+2]+=lazy[node];
            }
            lazy[node]=0;
        }
        if(start>end || start>r || end<l) return;
        if(start>=l && end<=r){
            tree[node]+=(end-start+1)*val;
            if(start!=end){
                lazy[2*node+1]+=val;
                lazy[2*node+2]+=val;
            }
            return;
        }
        int mid=(start+end)/2;
        modify(2*node+1,start,mid,l,r,val);
        modify(2*node+2,mid+1,end,l,r,val);
        tree[node] = tree[2 * node + 1] + tree[2 * node
            + 2];
    }
};

// add value in interval by modify(l, r, val) function
    in [l, r] (note that both inclusive)
// get min or max of elements in range by query(l, r)
#define ll long long
#define vll vector<long long>
class segment_tree_minmax {
    public :

    vll seg_tree;
    ll n;
    ll h;
    ll base;
    vll d;

    ll __log2l(long long x)
    {
        return 64 - __builtin_clzll(x) - 1;
    }
    int operation(int a, int b) {
        base = 2e18;
```

```
        return min(a,b );
    }
    segment_tree_minmax(ll n) : n(n),seg_tree(n<<1,0){
        d.resize(n+1,0);
        h = __log2l(n) + 1;
    }
    segment_tree_minmax(vll &vec) :
         n(vec.size()),seg_tree(vec.size()<<1,0){
        d.resize(n+1,0);
        h = __log2l(n) + 1;

        f(i,0,n)
            seg_tree[n+i] = vec[i];

        rev_f(i,n-1,0)
            seg_tree[i] =
                min(seg_tree[i<<1],seg_tree[i<<1|1]);
                ///function max is used here
    }
    void apply(ll p, ll value) {
        seg_tree[p] += value;
        if (p < n) d[p] += value;
    }
    void build(ll p) {
        while (p > 1) p >>= 1, seg_tree[p] =
            operation(seg_tree[p<<1], seg_tree[p<<1|1])
            + d[p];
    }
    void push(ll p) {
        for (ll s = h; s > 0; --s) {
        ll i = p >> s;
        if (d[i] != 0) {
            apply(i<<1, d[i]);
            apply(i<<1|1, d[i]);
            d[i] = 0;
        }
        }
    }

    void modify(ll l, ll r, ll value) {
        r++;
        l += n, r += n;
        ll l0 = l, r0 = r;
        for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) apply(l++, value);
        if (r&1) apply(--r, value);
        }
        build(l0);
        build(r0 - 1);
    }
    ll query(ll l, ll r) {
        r++;
        l += n, r += n;
        push(l);
        push(r - 1);
        ll res = base;
        for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) res = operation(res, seg_tree[l++]);
        if (r&1) res = operation(seg_tree[--r], res);
        }
        return res;
    }
};

signed main() {
    vector<int> arr = {1, 3, 3, 2, 5, 7};
    segment_tree_minmax st(arr);
    st.modify(0, 3, 2);
    c2(st.query(0, -1), st.query(5, 4));
    c3(st.query(0, 4) , st.query(1, 4), st.query(4, 4));

    c1(st.query(2, 4));
    st.modify(0, 2, 2);
    c1(st.query(2, 4));

    seg_tree st1(arr);
    c1(st1.query(1, 4));
    c1(st1.query(0,0));
    c1(st1.query(5,5));
    st1.modify(0,1,2);
    c1(st1.query(0,1));
    c1(st1.query(1,3));
}
```

8.3 Modify-Assignment

```
// assign value in interval by modify(l, r, val)
    function in [l, r] (note that both inclusive)
// get sum of elements in range by query(l, r)
#define ll long long
#define vll vector<long long>
#define vpll vector<pair<long long, long long>>
#define fi first
#define se second
class segment_tree {
    public :

    ll n,h;
    vll t;
    vll d;
    ll never = -1e18;

    ll __log2l(long long x)
    {
        return 64 - __builtin_clzll(x) - 1;
    }
    segment_tree(ll n) : n(n){
        t.assign(n<<1,0);
        d.assign(n+1,never);
        h = __log2l(n) + 1;
    };
    segment_tree(vll arr): n(arr.size()) {
        t.assign(n<<1,0);
        d.assign(n+1,never);
        h = __log2l(n) + 1;
        f(i, 0, n) modify(i, i, arr[i]);
    }
    void calc(ll p, ll k) {
        if (d[p] == never) t[p] = t[p<<1] + t[p<<1|1];
        else t[p] = d[p] * k;
    }
    void apply(ll p, ll value, ll k) {
        t[p] = value * k;
        if (p < n) d[p] = value;
    }
    void build(ll l, ll r) {   ///O(log(n) + |r-l|)
        r++;
        ll k = 2;
        for (l += n, r += n-1; l > 1; k <<= 1) {
        l >>= 1, r >>= 1;
        for (ll i = r; i >= l; --i) calc(i, k);
        }
```

```
    }
    void push(ll l, ll r) {    ///O(log(n) + |r-l|)
        r++;

        ll s = h, k = 1 << (h-1);
        for (l += n, r += n-1; s > 0; --s, k >>= 1)
        for (ll i = l >> s; i <= r >> s; ++i) if (d[i]
             != never) {
            apply(i<<1, d[i], k);
            apply(i<<1|1, d[i], k);
            d[i] = never;
        }
    }
    void modify(ll l, ll r, ll value) { ///O(log(n))
        r++;
        if (value == never) return;
        push(l, l);
        push(r - 1, r - 1);
        bool cl = false, cr = false;
        ll k = 1;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1, k
             <<= 1) {
        if (cl) calc(l - 1, k);
        if (cr) calc(r, k);
        if (l&1) apply(l++, value, k), cl = true;
        if (r&1) apply(--r, value, k), cr = true;
        }
        for (--l; r > 0; l >>= 1, r >>= 1, k <<= 1) {
        if (cl) calc(l, k);
        if (cr && (!cl || l != r)) calc(r, k);
        }
    }
    ll query(ll l, ll r) {    ///O(log(n))
        r++;
        push(l, l);
        push(r - 1, r-1);
        ll res = 0;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
        if (l&1) res += t[l++];
        if (r&1) res += t[--r];
        }
        return res;
    }
};

// add value in interval by modify(l, r, val) function
    in [l, r] (note that both inclusive)
// get min or max of elements in range by query(l, r)
#define ll long long
#define vll vector<long long>
#define vpll vector<pair<long long, long long>>
#define fi first
#define se second
class segment_tree_minmax {
    public :

    ll n,h;
    vll t;
    vll d;
    ll never;

    int operation(int a, int b) {
        never = 1e18;
        return min(a,b);
    }
    int __log2(int x) {
        return 32 - __builtin_clz(x) - 1;
    }

    ll __log2l(long long x)
    {
        return 64 - __builtin_clzll(x) - 1;
    }

    segment_tree_minmax(ll n) : n(n){
        t.assign(n<<1,0);
        d.assign(n+1,never);
        h = __log2l(n) + 1;
    };
    segment_tree_minmax(vll arr): n(arr.size()) {
        t.assign(n<<1,0);
        d.assign(n+1,never);
        h = __log2l(n) + 1;
        f(i, 0, n) modify(i, i, arr[i]);
    }
    void calc(ll p, ll k) {
        if (d[p] == never) t[p] = operation(t[p<<1] ,
             t[p<<1|1]);
        else t[p] = d[p];
    }
    void apply(ll p, ll value, ll k) {
        t[p] = value;
        if (p < n) d[p] = value;
    }
    void build(ll l, ll r) {    ///O(log(n) + |r-l|)
        r++;
        ll k = 2;
        for (l += n, r += n-1; l > 1; k <<= 1) {
        l >>= 1, r >>= 1;
        for (ll i = r; i >= l; --i) calc(i, k);
        }
    }
    void push(ll l, ll r) {    ///O(log(n) + |r-l|)
        r++;

        ll s = h, k = 1 << (h-1);
        for (l += n, r += n-1; s > 0; --s, k >>= 1)
        for (ll i = l >> s; i <= r >> s; ++i) if (d[i]
             != never) {
            apply(i<<1, d[i], k);
            apply(i<<1|1, d[i], k);
            d[i] = never;
        }
    }
    void modify(ll l, ll r, ll value) { ///O(log(n))
        r++;
        if (value == never) return;
        push(l, l);
        push(r - 1, r - 1);
        bool cl = false, cr = false;
        ll k = 1;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1, k
             <<= 1) {
        if (cl) calc(l - 1, k);
        if (cr) calc(r, k);
        if (l&1) apply(l++, value, k), cl = true;
        if (r&1) apply(--r, value, k), cr = true;
        }
        for (--l; r > 0; l >>= 1, r >>= 1, k <<= 1) {
        if (cl) calc(l, k);
        if (cr && (!cl || l != r)) calc(r, k);
        }
    }
    ll query(ll l, ll r) {    ///O(log(n))
```

```
        r++;
        push(l, l);
        push(r - 1, r-1);
        ll res = never;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
        if (l&1) res = operation(res,t[l++]);
        if (r&1) res = operation(res,t[--r]);
        }
        return res;
    }
};

signed main(){
    fastio();
    vector<int> arr = { 410, 52, 51, 180, 222, 33, 33 };

    segment_tree st(arr);
    c1(st.query(2, 3));
    st.modify(0, 2, 5);
    st.modify(2, 4, 7);
    c3(st.query(0, 1), st.query(2, 2), st.query(3, 4));

    segment_tree_minmax st1(arr);
    c1(st1.query(2, 3));
    st1.modify(0, 2, 5);
    st1.modify(2, 4, 7);
    c3(st1.query(0, 1), st1.query(2, 2), st1.query(3,
        4));
}
```

8.4 Simple-Segment-Tree

```
class seg_tree
{
public:
    vector<int> tree;
    int n;
    int operation(int a, int b) { return min(a, b); }
    void construct(vector<int> &arr)
    {
        n = arr.size();
        tree.resize(2 * n);
        f(i, 0, n) tree[i + n] = arr[i];
        for (int i = n - 1; i > 0; i--)
            tree[i] = operation(tree[i << 1], tree[i <<
                1 | 1]);
    }
    void update(int pos, int value)
    {
        pos += n;
        tree[pos] = value;
        for (int i = pos; i > 1; i >>= 1)
            tree[i >> 1] = operation(tree[i], tree[i ^
                1]);
    }
    int query(int l, int r)
    {
        int res = LLONG_MAX;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1)
        {
            if (l & 1)
                res = operation(res, tree[l++]);
            if (r & 1)
                res = operation(tree[--r], res);
        }
        return res;
    }
};
```

9 Strings

9.1 Manachar

9.2 Prefix-Function

```
// Prefix-function :-
// e.g "abcabcd" - [0,0,0,1,2,3,0]
// "aabaaab" - [0,1,0,1,2,2,3]      ,here for the 6th
    element the substring will be [aabaaa] thus "aa"
    is prefix and suffix here.
// "abcabca" - [0,0,0,1,2,3,4]      ,here in the last
    element we can see that s[0...3] and s[3...6] thus
    index 3 is overlapping.
// "aaaaaaa" - [0,1,2,3,4,5,6]      , it is an another
    example of overlapping.
// Code to get the Prefix Function O(n):-
/////////////////////////////////////////////////////////
vector<int> prefix_function(const string &s)
{
    int n = s.size();
    vector<int> pi(n);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

9.3 Z-Function

```
// e.g "aaabaab" - [0,2,1,0,2,1,0], "abacaba" -
    [0,0,1,0,3,0,1], here the fifth element is 3 as
    [5-6-7] indexed elements of S are same as [0-1-2]
    indexed elements.
// Code to get the Z-Function O(n):-
/////////////////////////////////////////////////////////
vector<int> z_function(const string &s)
{
    int n = s.size();
    vector<int> z(n,0);

    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
```

```
}
```

10 Tree

10.1 Centroid-Decomposition

```
class centroid_decomposition{

        public :

        ll nodes = 0;
        vector<ll> subtree,parentcentroid;
        vector<set<ll>> adj;
        centroid_decomposition(ll n){
                subtree.resize(n);
                parentcentroid.resize(n);
                adj.resize(n);
                nodes=0;
        };

        void cal_sub_len(ll k, ll par)
        {
                nodes++;
                subtree[k]=1;
                for(auto it:adj[k])
                {
                        if(it==par)
                                continue;
                        cal_sub_len(it, k);
                        subtree[k]+=subtree[it];
                }
        }

        ll centroid(ll k, ll par)
        {
                for(auto it:adj[k])
                {
                        if(it==par)
                                continue;
                        if(subtree[it]>(nodes>>1))
                                return centroid(it, k);
                }
                return k;
        }

        void decompose(ll k, ll par)
        {
                nodes=0;
                cal_sub_len(k, k);
                ll node=centroid(k, k);
                parentcentroid[node]=par;
                for(auto it:adj[node])
                {
                        adj[it].erase(node);
                        decompose(it, node);
                }
        }
};

///Note :- Here the edges are stored in set.
///Take one element and then call function
    "decompose(0,-1)".
```

10.2 LCA

```
const int maxn=2e5, lg=20;
vector<int> adj[maxn];
int par[maxn][lg], level[maxn];

int goup(int u,int l) {
    for (int i=0; i<20; l>>=1, i++) {
        if (l&1) u = par[u][i];
    }
    return u;
}

int lca(int u, int v) {
    if (level[u]>level[v]) swap(u,v);
    v = goup(v, level[v]-level[u]);
    for (int i=19; i>=0; i--) {
        if  (par[u][i]!=par[v][i]) {
            u = par[u][i];
            v = par[v][i];
        }
    }
    return u;
}

void getparlev(int u, int p, int lev) {
    par[u][0] = p;
    for (int i=1; i<20; i++) {
        if (par[u][i-1]==-1) par[u][i] = -1;
        else par[u][i] = par[par[u][i-1]][i-1];
    }
    level[u]= lev;
    for (int v: adj[u]) if (v!=p) {
        getparlev(v,u,lev+1);
    }
}

signed main(){
    int n, q;
    cin>>n>>q;
    for (int i=0; i<n; i++) adj[i].clear();
    for(int u=1; u<n; u++) {
        int v;
        cin>>v;
        v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    getparlev(0, -1, 0);
}
```

11 Trie

11.1 Binary-Trie

```
struct Binary_trie {
        vector<array<long long, 2>> next;
        vector<long long> cnt;
        static const long long B = 16; //change msb
            accordingly
        long long root, cur;

        Binary_trie() {
                next.push_back({-1, -1});
```

```
        cnt.push_back(0);
        root = 0;
        cur = 1;
}
void insert(long long x) {   ///insert element
    in the trie
        long long tmp = root;
        ++cnt[tmp];
        for(long long i=B;i>=0;--i) {
                long long t = (x >> i) & 1;
                if(next[tmp][t] == -1) {
                        next[tmp][t] = cur++;
                        next.push_back({-1, -1});
                        cnt.push_back(0);
                }
                tmp = next[tmp][t];
                ++cnt[tmp];
        }
}

void erase(long long x) {   ///erase element in
    the trie
        long long tmp = root;
        --cnt[tmp];
        for(long long i=B;i>=0;--i) {
                tmp = next[tmp][(x >> i) & 1];
                --cnt[tmp];
        }
}
long long xor_max(long long x) {   ///maximum
    value among all (x^input[i]).
        long long tmp = root;
        long long ans = 0;
        if(cnt[tmp] == 0) {
                return -1;
        }
        for(long long i=B;i>=0;--i) {
                long long t = (x >> i) & 1;
                if(next[tmp][1 - t] != -1 &&
                    cnt[next[tmp][1 - t]] > 0) {
                        tmp = next[tmp][1 - t];
                        ans += 1LL << i;
                }
                else {
                        tmp = next[tmp][t];
                }
        }
        return ans;
}

long long xor_min(long long x) {    ///minimum
    value among all (x^input[i]).
        long long tmp = root;
        long long ans = 0;
        if(cnt[tmp] == 0) {
                return -1;
        }
        for(long long i=B;i>=0;--i) {
                long long t = (x >> i) & 1;
                if(next[tmp][t] != -1 &&
                    cnt[next[tmp][t]] > 0) {
                        tmp = next[tmp][t];
                }
                else {
                        tmp = next[tmp][1 - t];
                        ans += 1LL << i;
                }
                }
                return ans;
        }

        void clear() {
                next.clear();
                cnt.clear();
                root = 0;
                next.push_back({-1, -1});
                cnt.push_back(0);
                cur = 1;
        }
};
```

11.2 String-Trie

```
struct Trie {
    bool isEndofWord;
    Trie *children[26];
    Trie() {
        isEndofWord = false;
        for (int i=0; i<26; i++) children[i] = NULL;
    }
    bool search(string &s) {
        Trie *root = this;
        int n = s.size(), idx;
        for (int i=0; i<n; i++) {
            idx = s[i]-'a';
            if (!root->children[idx]) return false;
            root = root->children[idx];
        }
        return root->isEndofWord;
    }
    void insert(string s) {
        Trie *root = this;
        int n = s.size(), idx;
        for (int i=0; i<n; i++) {
            idx = s[i]-'a';
            if (!root->children[idx])
                root->children[idx] = new Trie();
            root = root->children[idx];
        }
        root->isEndofWord = true;
    }
};
```