

Contents

1	Template	2	4.5	Diameter	15
2	Data structures	2	4.6	Heavy Light Decomposition	16
2.1	Segment Tree	2	5	Math	17
2.2	Merge Sort Tree	3	5.1	Identities	17
2.3	Fenwick Tree	3	5.1.1	Combi	17
2.4	Sparse Table	4	5.1.2	XOR	17
2.5	Disjoint Set Union	4	5.2	Theorems	17
2.6	Lazy SegTree	5	6	Strings	17
2.6.1	Min/Max Range / Single Acces Element	5	6.1	Suffix Array	17
3	Graphs	6	7	Others	18
3.1	DFS	6	7.1	Binpow	18
3.2	BFS	6	7.2	Matrix exponentiation	18
3.3	Topological Sort	6			
3.4	Flood Fill	6			
3.5	MST	6			
3.5.1	Kruskal	6			
3.5.2	Prims	7			
3.6	Dijkstra	7			
3.6.1	Implementation	7			
3.6.2	Space-State graph	7			
3.7	Bellman-Ford	7			
3.7.1	Path	7			
3.7.2	Cycle	8			
3.8	Floyd-Warshall	8			
3.9	SCC Kosaraju	8			
3.10	Flows	9			
3.10.1	Dinic	9			
3.11	2-SAT	10			
3.12	Euler Paths and cycles	11			
3.12.1	Directed Graphs	11			
3.12.2	Unirected Graphs	12			
4	Trees	13			
4.1	LCA - Binary Lifting	13			
4.1.1	Normal LCA	13			
4.1.2	Min/Max weight on the path	14			
4.2	Centroid Decomposition	14			
4.3	Euler Tour Technique	15			
4.4	Tree Matching	15			

1 Template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define all(v) v.begin(),v.end()
6 #define rall(v) v.rbegin(),v.rend()
7 #define pb push_back
8 #define sz(a) int(a.size())
9 #define F first
10 #define S second
11 typedef long long ll;
12 typedef vector<int> vi;
13 typedef vector<ll> vll;
14 typedef pair<int,int> pii;
15 typedef pair<ll,ll> pll;
16 const int MOD = 1e9+7;
17 const int INF = INT_MAX;
18 const ll INF64 = LLONG_MAX;
19 const long double EPS = 1e-9;
20 const long double PI = acosl(-1.0L);
21 void setIO(string p){
22     freopen((p + ".in").c_str(), "r", stdin);
23     freopen((p + ".out").c_str(), "w", stdout);
24 }
25
26 void solve(){
27
28 }
29
30
31 int main(){
32     ios_base::sync_with_stdio(0);
33     cin.tie(0);
34     cout.tie(0);
35     int tc = 1;
36     //cin >> tc;
37     for (int t = 1; t <= tc; t++)solve();
38 }

```

2 Data structures

2.1 Segment Tree

```

1 struct Node{
2     ll val = 0;
3 };
4 Node operator+(Node a,Node b){return {a.val+b.val};}
5 class Segment_Tree{
6 public:
7     int n;
8     vector<Node> tree;
9
10    Segment_Tree(int x = 1e5+10){
11        n = x;
12        tree.resize(2*n+1);
13    }
14
15    void build(vi &arr){
16        for(int i=0;i<n;i++)tree[n+i] = {arr[i]};
17        for(int i = n-1; i>0;i--)tree[i] = tree[i*2]+tree[i*2+1];
18    }
19
20    void update(int p, ll val){
21        for(tree[p+=n] = {val};p>1;p>>=1)tree[p>>1] = tree[p]+tree[p^1];
22    }
23
24    ll query(int l,int r){
25        Node s;
26        for(l+=n,r+=n ; l<r ; l>>=1 , r>>=1)
27            if(l&1)s = s+ tree[l++];
28            if(r&1)s = s+ tree[--r];
29        }
30        return s.val;
31    }
32 };

```

2.2 Merge Sort Tree

```

1 struct Node{
2     vi val;
3
4     int search(int x){
5         int l = 0, r = sz(val)-1;
6         while(l<=r){
7             int m = l+(r-l)/2;
8             if(val[m] < x) l = m+1;
9             else r = m-1;
10        }
11        return sz(val)-1;
12    }
13 };
14
15 Node operator+(Node a,Node b){
16     int i=0,j=0;
17     Node aux;
18     while (i< sz(a.val) && j<sz(b.val)){
19         if (a.val[i]< b.val[j] ) aux.val.pb(a.val[i++]);
20         else aux.val.pb(b.val[j++]);
21     }
22     while (i<sz(a.val)) aux.val.pb(a.val[i++]);
23     while (j<sz(b.val)) aux.val.pb(b.val[j++]);
24     return aux;
25 }
26 //Call segment Tree

```

2.3 Fenwick Tree

```

1 struct Node{
2     ll n = 0;
3     Node operator+(Node b){return {n+b.n};}
4 };
5
6 class Fenwick_Tree{
7 public:
8     vector<Node> tree;
9     int n;
10    Fenwick_Tree(int x){
11        n = x+1;tree.resize(n);
12    }
13
14    ll sum(int r){
15        r++;
16        Node ans = {0};
17        while(r){
18            ans = ans + tree[r];
19            r -= r&-r;
20        }
21        return ans.n;
22    }
23
24    void update(int id, Node val){
25        id++;
26        while(id < n){
27            tree[id] = tree[id]+val;
28            id += id&-id;
29        }
30    }
31 };

```

2.4 Sparse Table

```

1 struct Node{
2     int val = 0;
3     Node operator+(Node b){return {__gcd(val,b.val)};}
4 };
5
6 class SparseTable{
7     public:
8     vector<vector<Node>> sparse;
9
10    SparseTable(int n=2e5+10){
11        sparse.assign(n , vector<Node> (log2(n)+1));
12    }
13
14    void build(const vi &a){
15        for(int i=0;i<sz(a);i++)
16            sparse[i][0] = {a[i]};
17
18        for(int j=1;(1<<j)<=sz(a);j++)
19            for(int i=0;i+(1<<j)-1 <sz(a);i++)
20                sparse[i][j]= sparse[i][j-1]+sparse[i+(1<<(j-1))][j-1];
21    }
22
23    int query(int l, int r){
24        int len = r-l+1;
25        int k = log2(len);
26        return (sparse[l][k]+sparse[r-(1<<k)+1][k]).val;
27    }
28
29 };

```

2.5 Disjoint Set Union

```

1 class DSU{
2     private:
3     vi parent,rank,size;
4     int c;
5     public:
6     int mx = 0;
7     DSU(int n):parent(n+1),rank(n+1,0),size(n+1,1),c(n){
8         iota(all(parent),0);
9     }
10
11    int find(int u){return parent[u] == u?u:parent[u] = find(parent[u]) ;}
12
13    bool same(int u,int v){return find(u)==find(v);}
14
15    int get_size(int u){return size[find(u)];}
16
17    int count(){return c;}
18
19    void merge(int u,int v){
20        u = find(u);
21        v = find(v);
22        if(u!=v){
23            c--;
24            if(rank[u] > rank[v])swap(u,v);
25            parent[u] = v;
26            size[v] += size[u];
27            if(rank[u] == rank[v])rank[v]++;
28            mx = max(mx,size[v]);
29        }
30    }
31 };

```

2.6 Lazy SegTree

2.6.1 Min/Max Range / Single Acces Element

```

1 struct Node{
2     ll val = 0;
3 };
4
5 struct LazyNode{
6     ll val = 0;
7     bool can(){return val!=0;}
8 };
9
10 Node operator+(Node a, LazyNode b){return {a.val+b.val};}
11 Node operator+(Node a, Node b){return {a.val+b.val};}
12 LazyNode operator+(LazyNode a, LazyNode b){return {a.val+b.val};}
13
14 class Segment_Tree{
15 private:
16     int n,h;
17
18     void apply(int p, ll val){
19         tree[p].val += val;
20         if(p < n)d[p].val += val;
21     }
22
23     void build_lazy(int p){
24         while(p>1){
25             p >>= 1;
26             tree[p] = (tree[p*2]+tree[p*2+1])+d[p];
27         }
28     }
29
30     void push(int p){
31         for(int s = h;s>0;--s){
32             int i = p>>s;
33             if(d[i].can()){
34                 apply(2*i,d[i].val);
35                 apply(2*i+1,d[i].val);
36                 d[i].val = 0;
37             }
38         }
39     }

```

```

40
41 public:
42     vector<Node> tree;
43     vector<LazyNode> d;
44     Segment_Tree(int x = 1e5+10){
45         n = x;
46         tree.resize(2*n+1);
47         d.resize(n+1);
48         h = sizeof(int)*8 - __builtin_clz(n);
49     }
50     void build(vi &arr){
51         for(int i=0;i<n;i++)tree[n+i] = {arr[i]};
52         for(int i = n-1; i>0;i--)tree[i] = tree[i*2]+tree[i*2+1];
53     }
54     ll query(int l, int r){
55         l += n;r+=n;
56         push(l);
57         push(r-1);
58         Node s;
59         for(;l<r; l>>=1,r>>=1){
60             if(l&1)s = s+ tree[l++];
61             if(r&1)s = s+ tree[--r];
62         }
63         return s.val;
64     }
65
66     void range_increment(int l,int r,ll val){
67         l += n;
68         r += n;
69         int l0=l,r0=r;
70
71         for(;l<r;l>>=1,r>>=1){
72             if(l&1)apply(l++,val);
73             if(r&1)apply(--r,val);
74         }
75         build_lazy(l0);
76         build_lazy(r0-1);
77     }
78
79 };

```

3 Graphs

3.1 DFS

```

1 void dfs(int u){
2     vis[u] = 1;
3     for(auto v:adj[u]){
4         if(!vis[v])dfs(v);
5     }
6 }

```

3.2 BFS

```

1 queue<int> q;
2 q.push(0);
3 while(!q.empty()){
4     int u = q.front();
5     q.pop();
6     for(int v:adj[u]){
7         if (!vis[v]) {
8             vis[v] = 1;
9             q.push(v);
10        }
11    }
12 }

```

3.3 Topological Sort

```

1 vi ts;
2 queue<int> q;
3 for (int i = 1; i < n+1; i++)
4     if(!in[i])q.push(i);
5
6 while (!q.empty()) {
7     int u=q.front();
8     q.pop();
9     ts.pb(u);
10    for (int v : adj[u]) {
11        in[v]--;
12        if(!in[v])q.push(i);
13    }
14 }
15
16 if(sz(ts)!=n)cout <<"IMPOSSIBLE";

```

3.4 Flood Fill

```

1 int dx[] = {1,1,0,-1,-1,-1,0,1};
2 int dy[] = {0,1,1,1,0,-1,-1,-1};
3
4 int floodfill(int x,int y,char color1,char color2){
5     if(grid[x][y] == "outside")return 0;
6     if(grid[x][y] != color1)return 0;
7     int ans = 1;
8     grid[x][y] = color2;
9     for(int d=0;d<8;d++)ans += floodfill(x+dx[d],y+dy[d],color1,colr2);
10    return ans;
11 }

```

3.5 MST

3.5.1 Kruskal

```

1 struct Edge{
2     int u,v,w;
3     bool operator < (Edge b){return w<b.w;}
4 };
5
6 //inside main
7
8 vector<Edge> edges;
9 sort(all(edges));
10
11 DSU dsu(vertices);
12
13 for(int i=0;i<E;i++){
14     Edge front = edges[i];
15     if(!dsu.same(front.u,front.v)){
16         cost+= front.w;
17         dsu.merge(front.u,front.v);
18     }
19 }

```

3.5.2 Prims

```

1 vector<bool> vis(n);
2 priority_queue<pii, vector<pii>, greater<pii>> q;
3
4 auto process = [&](int u){
5     vis[u] = 1;
6     for(int i=0;i<sz(adj[u]);i++){
7         pii v = adj[u][i];
8         swap(v.S,v.F);
9         q.push(v);
10    }
11 }
12 process(0);
13
14 while(!q.empty()){
15     auto [w,u] = q.top();q.pop();
16     if(!vis[u]){cost += w;process(u);}
17 }

```

3.6 Dijkstra

3.6.1 Implementation

```

1 vll dist(n,INF64);
2 vector<bool> vis(n,0);
3 priority_queue<pll,vector<pll>,greater<pll>> pq;
4 pq.emplace(0,0);
5 dist[0] = 0;
6 vll cnt(n,0);//number of shortest paths
7 while(!pq.empty()){
8     auto [d1,u] = pq.top();
9     pq.pop();
10    if(vis[u])continue;
11    vis[u] = 1;
12    for(auto [v,c]:adj[u]){
13        if(dist[u]+c < dist[v]){
14            dist[v] = dist[u]+c;
15            pq.emplace(dist[v],v);
16            cnt[v] = cnt[u];
17        }else if(dist[u]+c == dist[v]){cnt[v] += cnt[u]}%MOD
18    }
19 }

```

3.6.2 Space-State graph

```

1 typedef array<ll,3> T;
2 priority_queue<T ,vector<T>,greater<T>> pq;
3 pq.push({0,0,state[0]});
4 dist[0][state[0]] = 0;
5 while(!pq.empty()){
6     auto [d,u,s] = pq.top();
7     pq.pop();
8     if(vis[u][s])continue;
9     vis[u][s] = 1;
10    for(auto [v,w]:adj[u]){
11        ll c = min(s,slow[v]);
12        //Nuevo estad cuando llegue a V
13        if(dist[u][s]+w*s<dist[v][c]){
14            dist[v][c] = dist[u][s]+w*s;
15            pq.push({dist[v][c],v,c});
16        }
17    }
18 }

```

3.7 Bellman-Ford

3.7.1 Path

```

1 vector<int> d(n, INF);
2 d[v] = 0;
3 vector<int> p(n, -1);
4
5 for (;) {
6     bool any = false;
7     for (Edge e : edges)
8         if (d[e.a] < INF)
9             if (d[e.b] > d[e.a] + e.cost) {
10                 d[e.b] = d[e.a] + e.cost;
11                 p[e.b] = e.a;
12                 any = true;
13             }
14     if (!any)
15         break;
16 }
17
18 if (d[t] == INF)
19     cout << "No path from " << v << " to " << t << ".";

```

```

20 else {
21     vector<int> path;
22     for (int cur = t; cur != -1; cur = p[cur])path.push_back(cur);
23     reverse(path.begin(), path.end());
24 }

```

3.7.2 Cycle

```

1 vector<int> d(n, INF);
2 d[v] = 0;
3 vector<int> p(n, -1);
4 int x;
5 for (int i = 0; i < n; ++i) {
6     x = -1;
7     for (Edge e : edges)
8         if (d[e.a] < INF)
9             if (d[e.b] > d[e.a] + e.cost) {
10                 d[e.b] = max(-INF, d[e.a] + e.cost);
11                 p[e.b] = e.a;
12                 x = e.b;
13             }
14 }
15 if (x == -1)cout << "No_negative_cycle_from_" << v;
16 else{
17     int y = x;
18     for (int i = 0; i < n; ++i)y = p[y];
19     vector<int> path;
20     for (int cur = y;; cur = p[cur]) {
21         path.push_back(cur);
22         if (cur == y && path.size() > 1)break;
23     }
24     reverse(path.begin(), path.end());
25 }

```

3.8 Floyd-Warshall

```

1 for (int i = 0; i < n; ++i)d[i][i] = 0;
2
3 for (int k = 0; k < n; ++k)
4     for (int i = 0; i < n; ++i)
5         for (int j = 0; j < n; ++j)
6             if (d[i][k] < INF && d[k][j] < INF)
7                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

```

3.9 SCC Kosaraju

```

1 class SCC{
2 private:
3     int n;
4     vector<vi> adj;
5     vector<vi> adj_t;
6     vector<vi> adj_cond;
7     vi order;
8     vector<bool> vis;
9     vi who;
10    int comp;
11 public:
12    SCC(int n,vector<vi> &a,vector<vi>&b):n(n),adj(a),adj_t(b){
13        comp = -1;
14        who.resize(n);
15        vis.resize(n);
16        for(int i=0;i<n;i++)
17            if(!vis[i])dfs1(i);
18
19        vis.assign(n,0);
20        for(int i=n-1;i>=0;i--){
21            if(!vis[order[i]]){
22                comp++;
23                dfs2(order[i]);
24            }
25            adj_cond.resize(comp+1);
26            for(int u=0;u<n;u++)
27                for(auto v:adj[u])
28                    if(who[v]!=who[u])
29                        adj_cond[who[u]].pb(who[v]);
30        }
31
32    void dfs1(int u) {
33        vis[u] = true;
34        for (int v:adj[u])
35            if (!vis[v])dfs1(v);
36        order.pb(u);
37    }
38
39    void dfs2(int u) {
40        vis[u] = 1;
41        for (int v : adj_t[u])

```



```

42     if (!vis[v])dfs2(v);
43     who[u] = comp;
44 }
45 };

```

3.10 Flows

3.10.1 Dinic

```

1 struct flowEdge{
2     int u,v;
3     ll cap,flow = 0;
4     flowEdge(int u, int v, ll cap) : u(u), v(v), cap(cap) {};
5 };
6
7 struct Dinic{
8     vector<flowEdge> edges;
9     vector<vi> adj;
10    int n,s,t;
11    int id = 0;
12    vi level, next;
13
14    queue<int> q;
15
16    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
17        adj.resize(n);
18        level.resize(n);
19        next.resize(n);
20        fill(all(level),-1);
21        level[s] = 0;
22        q.push(s);
23    }
24
25    void add(int u, int v, ll cap){
26        edges.emplace_back(u,v,cap);
27        edges.emplace_back(v,u,0);
28        adj[u].pb(id++);
29        adj[v].pb(id++);
30    }
31
32    bool bfs(){
33        while (!q.empty()){
34            int curr = q.front();
35            q.pop();

```

```

36        for (auto e : adj[curr]){
37            if (edges[e].cap - edges[e].flow < 1) continue;
38            if (level[edges[e].v] != -1) continue;
39            level[edges[e].v] = level[edges[e].u] + 1;
40            q.push(edges[e].v);
41        }
42    }
43    return level[t] != -1;
44 }
45
46 ll dfs(int u, ll flow){
47     if (flow == 0) return 0;
48     if (u == t) return flow;
49     for (auto& cid = next[u]; cid < adj[u].size(); cid++){
50         int e = adj[u][cid];
51         int v = edges[e].v;
52
53         if (level[edges[e].u] + 1 != level[v] || edges[e].cap - edges[e].
54             flow < 1) continue;
55         ll f = dfs(v,min(flow,edges[e].cap - edges[e].flow));
56         if (f == 0) continue;
57         edges[e].flow += f;
58         edges[e ^ 1].flow -= f;
59         return f;
60     }
61     return 0;
62 }
63
64 ll maxFlow(){
65     ll flow = 0;
66     while (bfs()){
67         fill(all(next),0);
68         for (ll f = dfs(s,INF64); f != 0ll; f = dfs(s,INF64)) flow += f;
69         fill(all(level),-1);
70         level[s] = 0;
71         q.push(s);
72     }
73     return flow;
74 }
75
76 vector<pii> minCut(){
77     vector<pii> ans;

```

```

78 fill(all(level),-1);
79 level[s] = 0;
80 q.push(s);
81 while (!q.empty()){
82     int curr = q.front();
83     q.pop();
84     for (int id = 0; id < adj[curr].size(); id++){
85         int e = adj[curr][id];
86         if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow > 0){
87             q.push(edges[e].v);
88             level[edges[e].v] = level[edges[e].u] + 1;
89         }
90     }
91 }
92
93 for (int i = 0; i < level.size(); i++){
94     if (level[i] != -1){
95         for (int id = 0; id < adj[i].size(); id++){
96             int e = adj[i][id];
97             if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow == 0)
98                 ans.emplace_back(edges[e].u, edges[e].v);
99         }
100     }
101 }
102 return ans;
103 }
104
105 vector<pii> MaximumMatching(){
106     vector<pii> ans;
107     fill(all(level),-1);
108     level[s] = 0;
109     q.push(s);
110     while (!q.empty()){
111         int curr = q.front();
112         q.pop();
113         for (int id = 0; id < adj[curr].size(); id++){
114             int e = adj[curr][id];
115             if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow == 0 &&
116                 edges[e].flow != 0){
117                 q.push(edges[e].v);
118                 level[edges[e].v] = level[edges[e].u] + 1;
119             }
120         }
121     }
122     return ans;
123 }

```

```

120 }
121 for (int i = 0; i < level.size(); i++){
122     if (level[i] != -1)
123         for (int id = 0; id < adj[i].size(); id++){
124             int e = adj[i][id];
125             if (edges[e].u != s && edges[e].v != t && edges[e].cap - edges[e]
126                 ].flow == 0 && edges[e].flow != 0){
127                 ans.emplace_back(edges[e].u, edges[e].v);
128             }
129         }
130     return ans;
131 }
132 };

```

3.11 2-SAT

```

1 class SAT{
2 private:
3     int n;
4     vector<vi> adj;
5     vector<vi> adj_t;
6     vector<vi> adj_cond;
7     vi order;
8     vector<bool> vis;
9     vi who;
10    vector<bool> ans;
11    int comp;
12 public:
13
14    SAT(int m=0){
15        n = (m<<1);
16        adj.resize(n);
17        adj_t.resize(n);
18        vis.assign(n,0);
19        who.assign(n,-1);
20    }
21
22    void dfs1(int u) {
23        vis[u] = true;
24        for (int v:adj[u])
25            if (!vis[v])dfs1(v);
26        order.pb(u);

```

```

27 }
28
29 void dfs2(int u) {
30     vis[u] = 1;
31     for (int v : adj_t[u])
32         if (!vis[v]) dfs2(v);
33     who[u] = comp;
34 }
35
36 //~a->b ~b->a
37 void addOR(int a, bool af, int b, bool bf) {
38     a += a + (af ^ 1);
39     b += b + (bf ^ 1);
40     adj[a ^ 1].push_back(b);
41     adj[b ^ 1].push_back(a);
42     adj_t[b].push_back(a ^ 1);
43     adj_t[a].push_back(b ^ 1);
44 }
45
46 //( a OR b ) ^ (~a or ~b)
47 void addXOR(int a, bool af, int b, bool bf) {
48     addOR(a, af, b, bf);
49     addOR(a, !af, b, !bf);
50 }
51 //(a -> b)
52 void _add(int a, bool af, int b, bool bf) {
53     a += a + (af ^ 1);
54     b += b + (bf ^ 1);
55     adj[a].push_back(b);
56     adj_t[b].push_back(a);
57 }
58
59 //(a->b)^(b->a)
60 void add(int a, bool af, int b, bool bf) {
61     _add(a, af, b, bf);
62     _add(b, !bf, a, !af);
63 }
64
65 bool solve_SAT(){
66     for(int i=0;i<n;i++)
67         if(!vis[i]) dfs1(i);
68
69     comp = -1;

```

```

70     vis.assign(n,0);
71     for(int i=sz(order)-1;i>=0;i--)
72         if(!vis[order[i]]){
73             comp++;
74             dfs2(order[i]);
75         }
76     ans.assign(n/2,0);
77     for(int i=0;i<n;i+=2){
78         if(who[i] == who[i+1]) return 0;
79         ans[i/2] = (who[i] > who[i+1]);
80     }
81     return 1;
82 }
83
84 bool operator[] (int i){
85     return ans[i];
86 }
87
88 };

```

3.12 Euler Paths and cycles

3.12.1 Directed Graphs

```

1 struct EulerSolver{
2     private:
3         vector<vector<pii>> adj;
4         int n, edges;
5         vi in, out;
6         vi ans, edge_ans;
7         int root;
8         vi done;
9         void dfs(int u){
10             while(done[u] < sz(adj[u])){
11                 auto [v, id] = adj[u][done[u]++];
12                 dfs(v);
13                 edge_ans.pb(id);
14             }
15             ans.pb(u);
16         }
17     public:
18         EulerSolver(vector<vector<pii>> &a, int t): adj(a){
19             n = t;
20             edges = 0;

```

```

21     in.assign(n,0);
22     out.assign(n,0);
23     done.assign(n,0);
24     root = -1;
25     for(int u=0;u<n;u++){
26         for(auto v:adj[u]){
27             edges++;
28             in[v.F]++;
29             out[u]++;
30         }
31     }
32 }
33
34 bool possible(){
35     int cnt1=0,cnt2=0;
36     bool ok = 1;
37     for(int i=0;i<n;i++){
38         if (in[i] - out[i] == 1) cnt1++;
39         if (out[i] - in[i] == 1) cnt2++; root = i;
40         if (abs(in[i] - out[i]) > 1) ok = 0;
41     }
42     if(cnt1 > 1 || cnt2 > 1)ok = 0;
43     if(!ok)return 0;
44
45     if(root == -1)
46         for(int i=0;i<n;i++)
47             if(out[i])root = i;
48
49     if(root == -1)return 1;
50     dfs(root);
51
52     if(sz(ans) != edges+1)return 0;
53     reverse(all(ans));
54     reverse(all(edge_ans));
55     return 1;
56 }
57
58 int size(){
59     return ans.size();
60 }
61
62 int operator()(int i,bool x){
63     if(x)return ans[i];

```

```

64     else return edge_ans[i];
65 }
66 };

```

3.12.2 Unirected Graphs

```

1 struct EulerSolver{
2 private:
3     vector<vector<pii>> adj;
4     int n,edges;
5     vi ans,edge_ans;
6     vi deg;
7     int root;
8     vi done;
9     vector<bool> vis;
10
11 void dfs(int u){
12     while(done[u] < sz(adj[u])){
13         auto [v,id] = adj[u][done[u]++];
14         if(vis[id])continue;
15         vis[id]=1;
16         dfs(v);
17         edge_ans.pb(id);
18     }
19     ans.pb(u);
20 }
21 public:
22 EulerSolver(vector<vector<pii>> &a,int t):adj(a),n(t){
23     edges = 0;
24     done.assign(n,0);
25     deg.assign(n,0);
26     root = -1;
27     for(int u=0;u<n;u++){
28         for(auto v:adj[u]){
29             edges++;
30             deg[v.F]++;
31             deg[u]++;
32         }
33     vis.assign(edges/2,0);
34 }
35
36 bool possible(){
37     int odd = 0;

```

```

38 for(int i=0;i<n;i++)
39     if((deg[i]/2)&1){
40         odd++;
41         root = i;
42     }
43
44
45 if(odd>2)return 0;
46
47 if(root == -1)
48     for(int i=0;i<n;i++)
49         if(deg[i]){root = i;break;}
50
51 if(root == -1)return 1;
52 dfs(root);
53
54 if(sz(ans) != edges/2+1)return 0;
55 reverse(all(ans));
56 reverse(all(edge_ans));
57 return 1;
58 }
59
60 int size(){
61     return ans.size();
62 }
63
64 int operator()(int i,bool x){
65     if(x)return ans[i];
66     else return edge_ans[i];
67 }
68 };

```

4 Trees

4.1 LCA - Binary Lifting

4.1.1 Normal LCA

```

1  const int MAX = 1e5+5,LG=18;
2  vi deep(MAX); //Si tiene peso vi cost(MAX)
3  int par[MAX][LG+1];
4
5  void dfs(int u = 1,int p=0){ // U = raiz del arbol
6      par[u][0] = p;
7      deep[u] = deep[p]+1; //cost[u] += cost[p]
8      for(int i=1;i<=LG;i++)par[u][i]=par[par[u][i-1]][i-1];
9      for(int v:adj[u]){
10         if(v!=p){
11             dfs(v,u);
12         }
13     }
14 }
15
16 int lca(int u,int v){
17     if(deep[u] < deep[v])swap(u,v);
18     for(int k=LG;k>=0;k--){
19         if(deep[par[u][k]] >= deep[v])
20             u = par[u][k];
21     if(u==v)return u;
22     for(int k=LG;k>=0;k--){
23         if(par[u][k]!=par[v][k])
24             u=par[u][k],v=par[v][k];
25     return par[u][0];
26 }
27
28 int dist(int u,int v){
29     int lc = lca(u,v);
30     return deep[u]+deep[v]-(deep[lc]<<1);
31 }
32
33 int kth(int u,int k){
34     assert(k>=0);
35     for(int i=0;i<=LG;i++){
36         if(k&(1<<i))u = par[u][i];
37     return u;

```

```
38 | }
```

4.1.2 Min/Max weight on the path

```
1 | const int MAX = 1e5+5, LG=18;
2 | vector<vector<pii>> adj(MAX, vector<pii> ());
3 | vi deep(MAX);
4 | int par[MAX] [LG+1];
5 | int mn[MAX] [LG+1];
6 | int mx[MAX] [LG+1];
7 |
8 | void dfs(int u=1, int p=0, int costmn = INF, int costmx = -INF){
9 |     par[u] [0] = p;
10 |    deep[u] = deep[p]+1;
11 |
12 |    if(p!=0){
13 |        mn[u] [0] = costmn;
14 |        mx[u] [0] = costmx;
15 |    }
16 |    for(int i=1; i<=LG; i++){
17 |        par[u] [i] = par[par[u] [i-1]] [i-1];
18 |        mn[u] [i] = min(mn[u] [i-1], mn[par[u] [i-1]] [i-1]);
19 |        mx[u] [i] = max(mx[u] [i-1], mx[par[u] [i-1]] [i-1]);
20 |    }
21 |
22 |    for(auto [v, t] : adj[u]){
23 |        if(v!=p){
24 |            dfs(v, u, t, t);
25 |        }
26 |    }
27 | }
28 |
29 | pii minmax(int u, int v){
30 |     pii ans = {INF, -INF};
31 |     if(deep[u] < deep[v]) swap(u, v);
32 |     for(int k=LG; k>=0; k--){
33 |         if(deep[par[u] [k]] >= deep[v]){
34 |             ans.F = min(ans.F, mn[u] [k]);
35 |             ans.S = max(ans.S, mx[u] [k]);
36 |             u = par[u] [k];
37 |         }
38 |     }
39 |     if(u==v) return ans;
40 |     for(int k=LG; k>=0; k--){
41 |         if(par[u] [k] != par[v] [k]){
42 |             ans.F = min({ans.F, mn[u] [k], mn[v] [k]});
43 |             ans.S = max({ans.S, mx[u] [k], mx[v] [k]});
44 |         }
45 |     }
46 | }
```

```
40 |     ans.S = max({ans.S, mx[u] [k], mx[v] [k]});
41 |     u=par[u] [k], v=par[v] [k];
42 | }
43 | ans.F = min({ans.F, mn[u] [0], mn[v] [0]});
44 | ans.S = max({ans.S, mx[u] [0], mx[v] [0]});
45 |
46 | return ans;
47 | }
```

4.2 Centroid Decomposition

```
1 | class Centroid{
2 | private:
3 |     vector<vi> tree;
4 |     vi par;
5 |     vi sub;
6 |     vector<bool> check;
7 |     int n;
8 | public:
9 |     Centroid(vector<vi> &t):tree(t){
10 |         n = tree.size();
11 |         par.resize(n);
12 |         sub.resize(n);
13 |         check.resize(n);
14 |         build();
15 |     }
16 |
17 |     void build(int u = 0, int p = -1){
18 |         dfs(u, p);
19 |         int c = get_centroid(u, p, sub[u]);
20 |         check[c] = 1;
21 |         par[c] = p+1;
22 |
23 |         for (auto v : tree[c]){
24 |             if(!check[v])
25 |                 build(v, c);
26 |         }
27 |
28 |     }
29 |
30 |     int dfs(int u, int p){
31 |         if(check[u]) return 0;
32 |         sub[u] = 1;
```

```

33     for(int v:tree[u])
34         if(v!=p)sub[u]+=dfs(v,u);
35
36     return sub[u];
37 }
38
39 int get_centroid(int u,int p,int x){
40     for(auto v:tree[u])
41         if(v!=p && sub[v]*2>x && !check[v])return get_centroid(v,u,x);
42
43     return u;
44 }
45
46 int operator[](int i){
47     return par[i];
48 }
49 };

```

4.3 Euler Tour Technique

```

1 int timer = 0;
2 vi start,final;
3 vector<vi> adj;
4 void dfs(int u = 0,int prev = -1){
5     start[u]=timer++;
6     for(int v:adj[u]){
7         if(v!=prev)dfs(v,u);
8     }
9     final[u] = timer;
10 }
11 // [start,final)

```

4.4 Tree Matching

```

1 vector<vll> dp(2,vll(MAX));
2
3 void dfs(ll u = 0, ll p=-1){
4     dp[0][u] = 0;
5     dp[1][u] = -INF;
6
7     for(ll v:adj[u]){
8         if(v == p)continue;
9         dfs(v,u);
10

```

```

11     dp[0][u] += max(dp[1][v],dp[0][v]);
12
13     ll opt = min(dp[0][v]-dp[1][v],0ll);
14
15     dp[1][u] = max(dp[1][u],opt);
16 }
17
18 dp[1][u] += dp[0][u];
19 dp[1][u]++;
20 }

```

4.5 Diameter

```

1 vector<vector<pii>> adj;
2 vector<bool> vis;
3
4 int bfs(int n,vll &d,int v = 0){
5     vis.assign(n,0);
6     d.assign(n,0);
7     d[v] = 0;
8     queue<int> q;
9     q.push(v);
10    pll last = {v,0};
11    vis[v] = 1;
12    while(!q.empty()){
13        int u = q.front();q.pop();
14        for(auto [w,c]:adj[u]){
15            if(vis[w])continue;
16            d[w] = d[u]+c;
17            q.push(w);
18            vis[w] = 1;
19            if(d[w]>last.S)last = {w,d[w]};
20        }
21    }
22    return int(last.F);
23 }
24
25 void solve(){
26     int n;cin >> n;
27     adj.assign(n,vector<pii> ());
28     for(int i=1;i<n;i++){
29         int a,b;ll c;
30         cin >> a >> b >> c;

```

```

31     adj[--a].pb({--b,c});
32     adj[b].pb({a,c});
33 }
34 vll dx(n),dy(n);
35 int a =bfs(n,dx);
36 int b =bfs(n,dy,a);
37 bfs(n,dx,b);
38
39 for(int i=0;i<n;i++){
40     cout << max(dx[i],dy[i]) << "␣\n"[i==(n-1)];
41 }
42
43 }

```

4.6 Heavy Light Decomposition

```

1  class HLD{
2  private:
3      int n;
4      vector<vi> adj;
5      vi parent;
6      vi heavy;
7      vi depth;
8      vi root;
9      vi treePos;
10     Segment_Tree tree;
11
12     int dfs(int u =0){
13         int size = 1, mx_sub = 0;
14         for(auto v:adj[u])
15             if(v!=parent[u]){
16                 parent[v] = u;
17                 depth[v] = depth[u]+1;
18                 int sub = dfs(v);
19                 if(sub > mx_sub){
20                     heavy[u] = v;
21                     mx_sub = sub;
22                 }
23                 size += sub;
24             }
25         return size;
26     }
27

```

```

28     template <class BinaryOperation>
29     void processPath(int u, int v, BinaryOperation op) {
30         for (; root[u] != root[v]; v = parent[root[v]]) {
31             if (depth[root[u]] > depth[root[v]]) swap(u, v);
32             op(treePos[root[v]], treePos[v] + 1);
33         }
34         if (depth[u] > depth[v]) swap(u, v);
35         op(treePos[u], treePos[v] + 1);
36     }
37
38     void init(vi &a){
39         for(int i=0,crt=0;i<n;++i){
40             if(parent[i] == -1 || heavy[parent[i]]!=i){
41                 for(int j = i;j!=-1;j = heavy[j]){
42                     root[j] = i;
43                     treePos[j] = crt++;
44                 }
45             }
46         }
47         tree = Segment_Tree(n);
48         for(int i=0;i<n;i++){
49             tree.update(treePos[i],a[i]);
50         }
51     }
52
53 public:
54
55     HLD(int n,vector<vi> &vale,vi &a):n(n),adj(vale){
56         parent.assign(n,0);
57         heavy.assign(n,-1);
58         depth.assign(n,0);
59         root.resize(n);
60         treePos.assign(n,0);
61         parent[0] = -1;
62         dfs();
63         init(a);
64     }
65
66     void update(int u,int &val){
67         tree.update(treePos[u],val);
68     }
69
70     void updatePath(int u, int v, int value) {

```



```

71 processPath(u, v, [this, &value](int l, int r) {
72     tree.update_range(l, r, value);
73 });
74 }
75
76 int query(int u, int v){
77     int ans = 0;
78     processPath(u, v, [this, &ans](int l, int r){
79         ans = max(ans, tree.query(l, r));
80     });
81     return ans;
82 }
83 };

```

5 Math

5.1 Identities

5.1.1 Combi

4. $k \binom{n}{k} = n \binom{n-1}{k-1}$
 5. $\binom{n}{k} = \binom{n}{n-k}$
 6. $\sum_{i=0}^n \binom{n}{i} = 2^n$
 7. $\sum_{i=0}^n \binom{n}{2i} = 2^{n-1}$
 8. $\sum_{i=0}^n \binom{n}{2i+1} = 2^{n-1}$
 9. $\sum_{i=0}^k (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$
 10. $\sum_{i=0}^k \binom{n+i}{i} = \sum_{i=0}^k \binom{n+i}{n} = \binom{n+k+1}{k}$
 11. $\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \dots + n\binom{n}{n} = n2^{n-1}$
 12. $1^2\binom{n}{1} + 2^2\binom{n}{2} + 3^2\binom{n}{3} + \dots + n^2\binom{n}{n} = (n+n^2)2^{n-2}$
 15. $\sum_{i=0}^k \binom{k}{i}^2 = \binom{2k}{k}$
 16. $\sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \binom{2n}{n}$
 17. $\sum_{k=q}^n \binom{n}{k} \binom{k}{q} = 2^{n-q} \binom{n}{q}$
 18. $\sum_{i=0}^n k^i \binom{n}{i} = (k+1)^n$
 19. $\sum_{i=0}^n \binom{2n}{i} = 2^{2n-1} + \frac{1}{2} \binom{2n}{n}$
 20. $\sum_{i=1}^n \binom{n}{i} \binom{n-1}{i-1} = \binom{2n-1}{n-1}$
 21. $\sum_{i=0}^n \binom{2n}{i}^2 = \frac{1}{2} \left(\binom{4n}{2n} + \binom{2n}{n}^2 \right)$

5.1.2 XOR

$$a + b = a \oplus b + 2(a \& b)$$

$$a + b = a|b + a \& b$$

$$a \oplus b = a|b - a \& b$$

$$1 \oplus 2 \oplus 3 \oplus \dots \oplus (4k-1) = 0 \quad k \geq 0$$

5.2 Theorems

6 Strings

6.1 Suffix Array

```

1 struct suffix{
2     int index;
3     pii rank;
4     bool operator<(suffix b){return rank<b.rank;}
5     bool operator>(suffix b){return rank>b.rank;}
6 };
7
8 class Suffix_Array{
9     public:
10    string s;int n;
11    vector<suffix> suffixes;
12    vi lcp;
13    Suffix_Array(string x){
14        s = x+"$";
15        n = sz(s);
16        suffixes.resize(n);
17    }
18
19
20 void radix_sort(vector<suffix> &suff){
21     for(int i: vi{2,1}){
22         auto key = [&](const suffix &x){
23             return i == 1?x.rank.F:x.rank.S;
24         };
25
26         int mx = 0;
27         for (const auto &i:suff) { mx = max(mx, key(i)); }
28         vector<int> occs(mx + 1);
29         for (const auto &i:suff)occs[key(i)]++;
30         vector<int> start(mx + 1);
31         for (int i=1;i<=mx; i++) start[i] = start[i-1]+occs[i-1];
32
33         vector<suffix> new_arr(suff.size());
34         for (const auto &i : suff) {
35             new_arr[start[key(i)]] = i;
36             start[key(i)]++;
37         }

```

```

38     suff = new_arr;
39 }
40 }
41
42 void build(){
43     for(int i=0;i<n;i++)suffixes[i].index = i , suffixes[i].rank = {s[i]
44         ],(i+1<n?s[i+1]:-1)};
45     sort(all(suffixes));
46     vi equiv(n);
47
48     for(int i=1;i<n;i++){
49         auto c_val = suffixes[i].rank;
50         int cs = suffixes[i].index;
51         auto p_val = suffixes[i - 1].rank;
52         int ps = suffixes[i-1].index;
53         equiv[cs] = equiv[ps] + (c_val > p_val);
54     }
55
56     for(int cmp_ant = 1;cmp_ant <n; cmp_ant<=<1){
57         for(auto &x:suffixes)
58             x.rank = {equiv[x.index],equiv[(x.index+cmp_ant)%n]};
59
60         //Std sort O(nlognlogn) Radix Sort O(nlogn)
61         radix_sort(suffixes);
62
63         for(int i=1;i<n;i++){
64             auto c_val = suffixes[i].rank;
65             int cs = suffixes[i].index;
66             auto p_val = suffixes[i - 1].rank;
67             int ps = suffixes[i-1].index;
68             equiv[cs] = equiv[ps] + (c_val > p_val);
69         }
70     }
71
72     void build_lcp(){
73         vector<int> suff_ind(n);
74         for (int i = 0; i < n; i++)suff_ind[suffixes[i].index] = i;
75         lcp.resize(n-1);
76         int start_at = 0;
77         for (int i = 0; i < n - 1; i++) {
78             int prev = suffixes[suff_ind[i] - 1].index;
79             int curr_cmp = start_at;

```

```

80         while (s[i + curr_cmp] == s[prev + curr_cmp])curr_cmp++;
81         lcp[suff_ind[i] - 1] = curr_cmp;
82         start_at = max(curr_cmp - 1, 0);
83     }
84 }
85
86 void show(){
87     for(int i=0;i<n;i++)cout << suffixes[i].index << "\n"[(i+1)==n];
88 }
89 };

```

7 Others

7.1 Binpow

```

1 ll binpow(ll a,ll b){
2     a%=MOD;
3     ll ans = 1;
4     while(b){
5         if(b&1)(ans*=a)%=MOD;
6         (a*=a)%=MOD;
7         b>>=1;
8     }
9     return ans;
10 }

```

7.2 Matrix exponentiation

```

1 struct Mat {
2     int n, m;
3     vector<vector<int>>> a;
4     Mat() { }
5     Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(m, 0)); }
6     Mat(vector< vector<int>> > v) { n = v.size(); m = n ? v[0].size() : 0;
7         a = v; }
8     inline void make_unit() {
9         assert(n == m);
10        for (int i = 0; i < n; i++) {
11            for (int j = 0; j < n; j++) a[i][j] = i == j;
12        }
13    }
14    inline Mat operator + (const Mat &b) {
15        assert(n == b.n && m == b.m);

```

```

15   Mat ans = Mat(n, m);
16   for(int i = 0; i < n; i++) {
17       for(int j = 0; j < m; j++) {
18           ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
19       }
20   }
21   return ans;
22 }
23 inline Mat operator - (const Mat &b) {
24     assert(n == b.n && m == b.m);
25     Mat ans = Mat(n, m);
26     for(int i = 0; i < n; i++) {
27         for(int j = 0; j < m; j++) {
28             ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
29         }
30     }
31     return ans;
32 }
33 inline Mat operator * (const Mat &b) {
34     assert(m == b.n);
35     Mat ans = Mat(n, b.m);
36     for(int i = 0; i < n; i++) {
37         for(int j = 0; j < b.m; j++) {
38             for(int k = 0; k < m; k++) {
39                 ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j] % mod)
40                     % mod;
41             }
42         }
43     }
44     return ans;
45 }
46 inline Mat pow(long long k) {
47     assert(n == m);
48     Mat ans(n, n), t = a; ans.make_unit();
49     while (k) {
50         if (k & 1) ans = ans * t;
51         t = t * t;
52         k >>= 1;
53     }
54     return ans;
55 }
56 inline Mat& operator += (const Mat& b) { return *this = (*this) + b; }
57 inline Mat& operator -= (const Mat& b) { return *this = (*this) - b; }

```

```

57   inline Mat& operator *= (const Mat& b) { return *this = (*this) * b; }
58   inline bool operator == (const Mat& b) { return a == b.a; }
59   inline bool operator != (const Mat& b) { return a != b.a; }
60 };
61
62 // Usage
63 // Mat a(n, n);
64 // Mat b(n, n);
65 // Mat c = a * b;
66 // Mat d = a + b;
67 // Mat e = a - b;
68 // Mat f = a.pow(k);
69 // a.a[i][j] = x;

```