

Contents

1 Data structures	1
1.1 Segment Tree	1
1.2 Merge Sort Tree	2
1.3 Fenwick Tree	2
1.4 Sparse Table	3
1.5 Disjoint Set Union	3
2 Graphs	4
2.1 DFS	4
2.2 BFS	4
2.3 Topological Sort	4
2.4 Dijkstra	4
2.5 Bellman-Ford	4
2.6 Floyd-Warshall	4
2.7 SCC Kosaraju	4
2.8 2-SAT	5
2.9 Euler Paths and cycles	6
2.9.1 Directed Graphs	6
2.9.2 Unirected Graphs	6
3 Trees	7
3.1 LCA - Binary Lifting	7
3.1.1 Normal LCA	7
3.1.2 Min/Max weight on the path	8
3.2 Centroid Decomposition	8
3.3 Euler Tour Technique	9
3.4 Tree Matching	9
3.5 Diameter	9
3.6 Heavy Light Decomposition	10
4 Math	11
4.1 Identities	11
4.2 Theorems	11
5 Strings	11
5.1 Suffix Array	11
6 Others	12

1 Data structures

1.1 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(t[p+=n] = {val};p>1;p>>=1)tree[p>>1] = tree[p]+t[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 };

```

1.2 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 class Segment_Tree{
27 public:
28     int n,k=0;
29     vector<Node> tree;
30     Segment_Tree(int x = 1e5+10){
31         n = x;
32         while((1<<k)<n)k++;
33         tree.resize(2*(1<<k)+1);
34     }
35     void build(vi &arr){
36         for(int i=0;i<n;i++)tree[(1<<k)+i].val.pb(arr[i]);
37         for(int i = (1<<k)-1; i>0;i--)tree[i] = tree[i*2]+tree[i*2+1];
38     }
39
40     void query(int a,int b,ll &ans,int i){
41         a += (1<<k);

```

```

42         b += (1<<k);
43         while(a<=b){
44             if(a%2 == 1)ans += tree[a++].search(i);
45             if(b%2 == 0)ans += tree[b--].search(i);
46             a/=2;
47             b/=2;
48         }
49     }
50 };

```

1.3 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 };

```

1.4 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 };

```

1.5 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 Graphs

2.1 DFS

```

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

```

2.2 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 }

```

2.3 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";

```

2.4 Dijkstra

2.5 Bellman-Ford

2.6 Floyd-Warshall

2.7 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 };

```

2.8 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 };

```

2.9 Euler Paths and cycles

2.9.1 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 };

```

2.9.2 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         if(odd>2)return 0;
45
46         if(root == -1)
47             for(int i=0;i<n;i++){
48                 if(deg[i]){root = i;break;}
49             }
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 };

```

3 Trees

3.1 LCA - Binary Lifting

3.1.1 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 }

```

3.1.2 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--) if(deep[par[u][k]] >= deep[v]) {
33         ans.F = min(ans.F, mn[u][k]);
34         ans.S = max(ans.S, mx[u][k]);
35         u = par[u][k];
36     }
37     if(u==v) return ans;
38     for(int k=LG; k>=0; k--) if(par[u][k] != par[v][k]) {
39         ans.F = min({ans.F, mn[u][k], mn[v][k]});

```



```

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 }

```

3.2 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     int dfs(int u,int p){
30         if(check[u])return 0;
31         sub[u] = 1;
32

```

```

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 };

```

3.3 Euler Tour Technique

3.4 Tree Matching

3.5 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 }

```

3.6 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 };

```

4 Math

4.1 Identities

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

$$\sigma(n) = O(\log(\log(n))) \text{ (number of divisors of } n)$$

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^n F_i = F_{n+2} - 1$$

$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_i F_j$$

$$\text{(Möbius Inv. Formula) Let } g(n) = \sum_{d|n} f(d), \text{ then } f(n) = \sum_{d|n} d \mu\left(\frac{n}{d}\right).$$

4.2 Theorems

5 Strings

5.1 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 };

```

6 Others