Team Notebook

Salmon

October 13, 2023

Contents

1 DP

2 Flow

2.1 Dinic

```cpp
#include <bits/stdc++.h>
using namespace std;
bool vis[505];
int lev[505], n, m, ans;
typedef struct {
    int to, w, rev_ind;
} edge;
vector<edge> adj[505];
bool label_level(){ // Tag the depthif can't reach end => return false
    memset(lev, -1, sizeof(lev));
    lev[1] = 0;
    queue<int> q; q.push(1);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(auto i : adj[u]){
            if(i.w > 0 && lev[i.to] == -1){
                q.push(i.to);
                lev[i.to] = lev[u] + 1;
            }
        }
    }
    return (lev[n] == -1 ? false : true);
}
int dfs(int u, int flow){
    if(u == n) return flow;
    for(auto &i : adj[u]){
        if(lev[i.to] == lev[u] + 1 && !vis[i.to] && i.w > 0){
            vis[i.to] = true;
            int ret = dfs(i.to, min(flow, i.w));
            if(ret > 0) {
                i.w -= ret;
                adj[i.to][i.rev_ind].w += ret;
                return ret;
            }
        }
    }
    return 0;  // if can't reach end => return 0
}
void dinic(){
    while(label_level()){
        while(1){
            init(vis, 0);
            int tmp = dfs(1, inf);
            if(tmp == 0) break;
            ans += tmp;
        }
    }
}
void build(){
    rep(i, 1, m){
        int u, v, w; cin >> u >> v >> w;
        adj[u].push_back({v, w, (int)adj[v].sz}); // inverse flow's index
        adj[v].push_back({u, 0, (int)adj[u].sz-1}); // have pushed oneneed to -1
    }
}
// Police Chaseneed to open adj to Augment && ori to determine what pb give
//  Dinicdfs2then use reach as uif the edge pb has given && w == 0 && v is not in reachis the ans
void dfs2(int now, unordered_set<int> &reach){
    if(!vis[now]){
        vis[now] = 1;
        reach.insert(now);
        for(auto i : adj[now]){
            if(i.w > 0){
                dfs2(i.to, reach);
            }
        }
    }
}
// two two pair // School Dance
//  Dinicthen w == 0 edge, which pb has given is the ans

// Distinct Route
// edge set valid varif we need to argument pos roadthe reverse edge set true valid
// if we need argument the argumented edgeboth set falselast , from v dfs ans times
bool get_road(int now, vector<int> &ans, vector<bool> &vis){
    if(now == 1) return true;
    for(auto &v : adj[now]){
        if(v.arg_valid && !vis[v.to]){
            ans.push_back(v.to);
            vis[v.to] = true;
            bool flag = get_road(v.to, ans, vis);
            if(flag){
                v.arg_valid = false;
                return true;
            }
            ans.pop_back();
        }
    }
    return false;
}
```

2.2 MCMF

```cpp
#include <bits/stdc++.h>
using namespace std;
// Ceiled MCMFif not use return to determine
typedef struct {
    int from, to, w, cost;
} edge;
int n, m, parcel;
vector<edge> adj; // set num to each edge
vector<int> p[505]; // p[u] has edge's num
int now_edge = 0;
void add_edge(int u, int v, int w, int cost){
    adj.push_back({u, v, w, cost});
    p[u].push_back(now_edge);
    now_edge++;
    adj.push_back({v, u, 0, -cost}); // argumenting path use -
    p[v].push_back(now_edge);
    now_edge++;
}
ll Bellman_Ford(){
    vector<ll> dis(n+1, inf); dis[1] = 0;
    vector<int> par(m);
    vector<int> flow_rec(n+1, 0); flow_rec[1] = 1e9;
    lrep(i, 1, n){
        bool flag = 1;
        int size = adj.sz;
        lrep(i, 0, size){
            auto &[from, to, w, cost] = adj[i];
            if(w > 0 && dis[to] > dis[from] + cost){
                flag = 0;
                dis[to] = dis[from] + cost;
                par[to] = i;  // record num
                flow_rec[to] = min(flow_rec[from], w);
            }
        }
        if(flag) break;
    }
    if(dis[n] == 1e9) return 0;
    int mn_flow = flow_rec[n];
    int v = n;
    while(v != 1){
```

```
        int u = adj[par[v]].from;
        adj[par[v]].w -= mn_flow;
        adj[par[v] ^ 1].w += mn_flow;
        v = u;
    }
    mn_flow = min(mn_flow, parcel);
    parcel -= mn_flow;
    return mn_flow * dis[n];
}
void solve(){
    cin >> n >> m >> parcel;
    ll ans = 0;
    rep(i, 1, m){
        int u, v, w, cost; cin >> u >> v >> w >> cost;
        add_edge(u, v, w, cost);
    }
    while(parcel > 0){
        int tmp = Bellman_Ford();
        if(tmp == 0) break;
        ans += tmp;
    }
    cout << (parcel > 0 ? -1 : ans);
}
```

3 Geometry

3.1 ConvexHull

```
vector<pii> P, L, U;
ll cross(pii o, pii a, pii b){ // OA OB >0 counterclock
    return (a.first - o.first) * (b.second - o.second) - (a.
        second-o.second) * (b.first-o.first);
}
ll Andrew_monotone_chain(ll n){
    sort(P.begin(), P.end());
    ll l = 0, u = 0; // upper and lower hull
    for (ll i=0; i<n; ++i){
        while (l >= 2 && cross(L[l-2], L[l-1], P[i]) <= 0){
            l--;
            L.pop_back();
        }
        while (u >= 2 && cross(U[u-2], U[u-1], P[i]) >= 0){
            u--;
            U.pop_back();
        }
        l++;
        u++;
        L.push_back(P[i]);
        U.push_back(P[i]);
    }
    cout << l << ' ' << u << '\n';
    return l + u;
}
int main(){
    ll n,x,y;
    cin >> n;
    for(ll i = 0;i < n;i++){
        cin >> x >> y;
        P.push_back({x,y});
    }
    ll ans = Andrew_monotone_chain(n) - 2;
    cout << ans << "\n";
    return 0;
}
```

3.2 CrossProduct

```
const double EPS = 1e-9;
struct point{
    double x, y;
    point operator * (ll a){return {a * x, a * y};}
    point operator + (point b){return {x + b.x, y + b.y};}
    point operator - (point b){return {x - b.x, y - b.y};}
    double operator * (point b){return x * b.x + y * b.y;}
    double operator ^ (point b){return x * b.y - y * b.x;}
    bool operator < (point b){return x == b.x ? y < b.y : x <
        b.x;}
};
// len
double abs(point a){return sqrt(a.x * a.x + a.y * a.y);}
int sign(double a){
    if(abs(a) < EPS)
        return 0;
    else
        return (a > 0 ? 1 : -1);
}
//cross product
int ori(point a,point b,point c){
    return sign((b - a) ^ (c - a));
}
bool colinear(point a,point b,point c){
    return sign((b - a) ^ (c - a)) == 0;
}
bool between(point a,point b,point c){ // c between a and b
    if(!colinear(a,b,c))
        return false;
    return sign((a - c) * (b - c)) <= 0;
}
bool intersect(point a,point b,point c,point d){ // line(a,b
    ) line(c,d)
    int abc = ori(a,b,c);
    int abd = ori(a,b,d);
    int cda = ori(c,d,a);
    int cdb = ori(c,d,b);
    if(abc == 0 || abd == 0)
        return between(a,b,c) || between(a,b,d) || between(c,
            d,a) || between(c,d,b);
    return abc * abd <= 0 && cda * cdb <= 0;
}
int main(){
    int n;
    cin >> n;
    point p[1010];
    cin >> p[0].x >> p[0].y;
    ll ans = 0;
    for(int i = 1;i < n;i++){
        cin >> p[i].x >> p[i].y;
        ans += (p[i] ^ p[i - 1]);
    }
    ans += (p[0] ^ p[n - 1]);
    cout << abs(ans) << '\n';

    return 0;
}
```

4 Graph

4.1 AllRoadPath

```
//        adj       [i].sz == odd => IMPOSSIBLEroad.sz != m
    +1 => IMPOSSIBLE
//        1        ->2    in_degree     == out_degree
int n, m;
set<int> adj[maxn];// rev_adj[maxn];
int in[maxn];
void dfs(int now, vector<int> &road){
    while(!adj[now].empty()){
        int nxt = *adj[now].begin();
        adj[now].erase(nxt);
        dfs(nxt, road);
    }
    road.push_back(now);
}
void solve(){
    cin >> n >> m;
```

```
    init(in, 0);
    rep(i, 1, m){
        int u, v; cin >> u >> v;
        adj[u].insert(v);
        in[v]++;
    }
    in[1]++;
    in[n]--;
    rep(i, 1, n){
        if(adj[i].size() != in[i]){
            cout << "IMPOSSIBLE";
            return;
        }
    }
    vector<int> road;
    dfs(1, road);
    if(road.size() != m+1){
        cout << "IMPOSSIBLE";
        return;
    }
    reverse(all(road));
    for(auto i : road) cout << i << " ";
}
```

4.2 BellmanFord

```
typedef struct{
    int from; int to;
    ll weight;
} edge;
const ll inf = 1LL << 62;
// NegCyc_Finding_Road
vector<edge> graph;
int main(){
    int src = 0;
    int n, m;   cin >> n >> m;
    vector<int> par(n+1), dis(n+1);
    rep(i, 0, m){
        int a, b, w; cin >> a >> b >> w;
        graph.push_back({a, b, w});
    }
    rep(i, 0, n+1) {
        dis[i] = 1e9 + 5;
    }
    dis[1] = 0;
    rep(i, 0, n + 1){
        src = 0;
        for(auto [a, b, w] : graph){
            if(dis[b] > dis[a] + w){
                dis[b] = dis[a] + w;
                par[b] = a;
                src = b;
            }
        }
    }
    if(src){
        vector<int> ans;
        cout << "YES" << endl;
        rep(i, 0, n + 1) src = par[src];
        ans.push_back(src);
        for(int i = par[src]; i != src; i = par[i]){
            ans.push_back(i);
        }
        ans.push_back(src);
        reverse(all(ans));
        for (auto i : ans){
            cout << i << " ";
        }
    }
    else {
        cout << "NO" << endl;
    }
}
```

4.3 DFSandBFS

```
ll N = 1e6;
vi adj[N];
bool vis[N];
void DFS(ll s){
    if(vis[s])return;
    vis[s] = true;
    for(auto u: adj[s]){
        DFS(u);
    }
}
ll timer;
void dfs(ll now, ll pa) {
    pos[now] = ++timer;
    add(timer, v[now]);
    sz[now] = 1;
    for (ll v : g[now]) {
        if (v == pa) continue;
        dfs(v, now);
        sz[now] += sz[v];
    }
}
queue<ll> q;
ll dis[N];
void BFS(ll x){
    vis[x] = true;
    dis[x] = 0;
    q.push(x);
    while(!q.empty()){
        ll s = q.front();q.pop();
        for(auto u: adj[s]){
            if(vis[u]) continue;
            vis[u] = true;
            dis[u] = dis[s] + 1;
            q.push(u);
        }
    }
}
```

4.4 Dijkstra

```
void solve(){
    int n, m, noon, night;
    cin >> n >> m >> noon >> night;
    ll dis[n+1];  //              kpqtop
    vll graph[n+1];
    bool vis[n+1]; //           kpq          .top(). firstdis
         [pq.top().second]     vis
    rep(i, 1, m){
        int u, v, w; cin >> u >> v >> w;
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }
    priority_queue<vector<ll>, vector<vector<ll>>, greater<
         vector<ll>>> pq;
    // noon   -
    rep(i, 1, n){
        dis[i] = inf;  vis[i] = 0;
    }
    pq.push({0, -noon, 1});
    dis[1] = 0;
    while(!pq.empty()){
        vl now = pq.top(); pq.pop();
        ll now_noon = -now[1], u = now[2];
        if(vis[u]) continue;
        for(auto [nxt, w] : graph[u]){
            if(noon < w) continue; //
            ll tmp = dis[u] + (now_noon >= w ? w : now_noon +
                  night + w);
            if(tmp < dis[nxt]){
                dis[nxt] = tmp;
```

```
            pq.push({dis[nxt], -(now_noon >= w ? now_noon
                - w : noon - w), nxt});
        }
    }
    vis[u] = true;
}
if(dis[n] == inf) cout << -1 << endl;
else cout << dis[n] << endl;
}
```

4.5 FloydWarshall

```
const int maxn = 505;
ll graph[maxn][maxn];
ll dis[maxn][maxn];
ll n, m, q; ll a, b, c;
const ll INF = 1e18;
int main(){
    cin >> n >> m >> q;
    for(int i = 0; i <= n; i++) {
        for(int j = 0; j <= n; j++) {
            graph[i][j] = INF;
        }
    }
    for(int i = 0; i < m; i++){
        cin >> a >> b >> c;
        graph[a][b] = min(graph[a][b], c);
        graph[b][a] = min(graph[b][a], c);
    }
    for(int i = 0; i <= n; i++) {
        for(int j = 0; j <= n; j++) {
            dis[i][j] = graph[i][j];
        }
    }
    for(int i = 0; i <= n; i++) //       0
        dis[i][i] = 0;

    for(int k = 1; k <= n; k++){ //
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= n; j++){
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][
                    j]);
            }
        }
    }
    for(int i = 0; i < q; i++){
        cin >> a >> b;
        cout << (dis[a][b] >= INF ? -1 : dis[a][b]) << "\n";
    }
}
```

4.6 MaxDistance

```
//
                                queue

//                  1ndis                   [n] < 0
//
void print_ans(int n, vector<int> &par){
    deque<int> ans;
    int now = n;
    while(now != 1){
        ans.push_front(now);
        now = par[now];
    }
    ans.push_front(1);
    cout << ans.size() << endl;
    for(auto i : ans){
        cout << i << " ";
    }
}
void solve(){
    int n, m;
    cin >> n >> m;
    // int dis[maxn];
    vector<int> dis(n+1, -inf); dis[1] = 0;
    vi graph[n+1];
    vector<bool> vis(n+1, 0);
    vector<int> par(n+1);
    vector<int> in(n+1, 0);
    // priority_queue<pii, vii, greater<pii>> pq;
    queue<int> q;
    rep(i, 1, m){
        int u, v; cin >> u >> v;
        graph[u].push_back(v);
        in[v]++;
    }
    // q.push({0, 1});
    rep(i, 1, n){
        if(in[i] == 0) q.push(i);
    }
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(auto nxt : graph[u]){
            if(dis[nxt] < dis[u] + 1){
                dis[nxt] = dis[u] + 1;
                par[nxt] = u;
            }
            in[nxt]--; if(in[nxt] == 0) q.push(nxt);
        }
        vis[u] = 1;
    }
    if(dis[n] < 0){
        cout << "IMPOSSIBLE";
    }
    else print_ans(n, par);
}
```

4.7 PlanetQueriesII

```
int n, q;
int dp[30][maxn];
vector<vector<int>> cycles;
int no[maxn]; //
                            cyclecycle
int cycle_idx[maxn];
bool vis[maxn];
void set_out_of_cycle_no(int now, unordered_set<int> &done){
    if (done.find(now) != done.end())
        return;
    set_out_of_cycle_no(dp[0][now], done);
    done.insert(now);
    no[now] = no[dp[0][now]] - 1;
}
int will_go_to(int u, int k){ //             k
    rep(i, 0, 18){
        if (k & (1 << i)){
            u = dp[i][u];
        }
    }
    return u;
}
void find_cycle(int now){
    unordered_set<int> appear;
    vector<int> vec;
    bool flag = true;
    while (appear.find(now) == appear.end()){
        appear.insert(now);
        vec.push_back(now);
        if (vis[now]){ //
            flag = false;
            break;
        }
        now = dp[0][now];
    }
    for (auto i : vec) vis[i] = true;
    if (!flag) return;
```

```
    int z = find(vec.begin(), vec.end(), now) - vec.begin();
        //        now
    int m = vec.size();
    vector<int> cycle;
    for (int i = z; i < m; i++){
        cycle.push_back(vec[i]);
    }
    cycles.push_back(cycle);
}
void solve(){
    cin >> n >> q;
    rep(u, 1, n){
        cin >> dp[0][u];
    }
    rep(i, 1, 18){ //
        rep(u, 1, n){
            dp[i][u] = dp[i - 1][dp[i - 1][u]];
        }
    }
    rep(i, 1, n){
        if(!vis[i]) find_cycle(i);
    }
    ll idx = 0;
    memset(no, -1, sizeof(no));
    memset(cycle_idx, -1, sizeof(cycle_idx));
    unordered_set<int> done;
    for (auto &i : cycles){
        ll c = 0;
        for (auto &j : i){
            no[j] = c++;
            cycle_idx[j] = idx;
            done.insert(j);
        }
        idx++;
    }
    rep(i, 1, n) set_out_of_cycle_no(i, done);
    rep(i, 1, q){
        int u, v; cin >> u >> v;
        //
        if(cycle_idx[u] == cycle_idx[v] && cycle_idx[u] != -1
              && cycle_idx[v] != -1){
            int cyc_size = cycles[cycle_idx[u]].size();
            cout << (no[v] - no[u] + cyc_size) % cyc_size <<
                 endl;
        }
        else if (cycle_idx[u] == -1 && cycle_idx[v] == -1){
             //
            if(no[u] > no[v]){
                cout << -1 << endl;
                continue;
            }
            ll jump = no[v] - no[u];
            if(will_go_to(u, jump) == v){
                cout << jump << endl;
            }
            else cout << -1 << endl;
        }
        else if (cycle_idx[u] == -1 && cycle_idx[v] != -1){
             // v
            int l = -1, r = n;
            while(l <= r){
                int m = (l + r) / 2;
                if(cycle_idx[will_go_to(u, m)] == cycle_idx[v
                     ]){
                    r = m - 1;
                }
                else l = m + 1;
            }
            if(l != -1 && l <= n){
                int in_cycle_of_u = will_go_to(u, l);
                int cycle_size = cycles[cycle_idx[v]].size();
                cout << l + (no[v] - no[in_cycle_of_u] +
                     cycle_size) % cycle_size << endl;
            }
            else cout << -1 << endl;
        }
        else { // u
            cout << -1 << endl;
        }
    }
}
```

4.8 PlanetsCycles

```
vi dis, v;
vector<bool> vis;
ll step;
queue<ll> path;
void dfs(ll x){
    path.push(x);
    if(vis[x]){
        step += dis[x];
        return;
    }
    vis[x] = true;
    step++;
    dfs(v[x]);
}
// count pathdis to rep
int main(){
    v.assign(n+1, 0);
    dis.assign(n+1, 0);
    vis.assign(n+1, false);
    for(int i = 1; i <= n; i++){
        cin >> v[i];
    }
    for(int i = 1; i <= n; i++){
        step = 0;
        int is_outof_cycle = 1;
        dfs(i);
        while(!path.empty()){
            if(path.front() == path.back()){
                is_outof_cycle = 0;
            }
            dis[path.front()] = step;
            step -= is_outof_cycle;
            path.pop();
        }
    }
    for(int i = 1;i <= n;i++){
        cout << dis[i] << ' ';
    }
    cout << '\n';
}
```

4.9 Prim

```
int n, m;
ll ans = 0;
vii adj[maxn];
bool Prim(){
    int node_sz = 0;
    priority_queue<pii, vii, greater<pii>> pq;
    pq.push({0, 1});
    bool vis[maxn]; init(vis, false);
    while(!pq.empty()){
        auto [cost, u] = pq.top(); pq.pop();
        if(vis[u]) continue;
        vis[u] = true;
        ans += cost;
        node_sz++;
        for(auto [v, cost] : adj[u]){
            if(!vis[v])
                pq.push({cost, v});
        }
    }
    if(node_sz == n) return true;
    return false;
```

```
}
void solve(){
    cin >> n >> m;
    rep(i, 1, m){
        int u, v, cost; cin >> u >> v >> cost;
        adj[u].push_back({v, cost});
        adj[v].push_back({u, cost});
    }
    if(Prim()) cout << ans;
    else cout << "IMPOSSIBLE";
}
```

4.10 SCC

4.11 TopologicalDP

```
//        DAG
```

5 init

```
#include <bits/stdc++.h>
using namespace std;
#define all(x) (x).begin(), (x).end()
#define endl "\n"
#define lrep(i, st, n) for(int i = st; i < n; i++)
#define rep(i, st, n) for(int i = st; i <= n; i++)
#define sz size()
#define pb(x) push_back(x)
#define ppb pop_back()
#define IO ios_base::sync_with_stdio(0); cin.tie(0);
#define init(x, k) memset(x, k, sizeof(x));
#define vec_init(x, k) x.assign(x.size(), k);
#define lc 2*now
#define rc 2*now+1
#define mid (L+R)/2
typedef long long int ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<pii> vii;
typedef pair<ll, ll> pll;
typedef vector<ll> vl;
typedef vector<pll> vll;
typedef struct {
    int from; int to;
    ll weight;
} edge;
typedef struct {
    ll sum;
} Node;
const ll llinf = 1e18;
const int inf = 1e9;
const int MOD = 1e9+7;
const int maxn = 2e5+5;

void solve(){

}
int main(){
    IO;
    int t = 1;
    cin >> t;
    while(t--){
        solve();
    }
}
```

6 Math

7 Queries

7.1 BIT

```
typedef struct {
    int set_val, add, sum, val;
} node;
node tree[100];
int n, q, nums[100], _1D_BIT[100], _2D_BIT[100][100];

// 1D-BIT
void modify(int x, int mod){
    for(; x <= n; x += (x&-x)){
        _1D_BIT[x] += mod;
    }
}
ll query(int x, int y){
    ll ans = 0;
    for(; x; x -= (x&-x)){
        ans += _1D_BIT[x];
    }
    return ans;
}
// 2D-BIT // Forest Queries (Area)
void modify(int x, int y, int mod){
    for(; x <= n; x += (x&-x)){
        for(int tmp = y; tmp <= n; tmp += (tmp&-tmp)){
            _2D_BIT[x][tmp] += mod;
        }
    }
}
ll query(int x, int y){
    ll ans = 0;
    for(; x; x -= (x&-x)){
        for(int tmp = y; tmp; tmp -= (tmp&-tmp)){
            ans += _2D_BIT[x][tmp];
        }
    }
    return ans;
}
```

7.2 Mo's *Algorithm*

```
typedef struct {
    int l, r, ind;
} query;
query queries[100];
int n, block, nums[100];
bool cmp(query a, query b){
    int block_a = a.l / block;
    int block_b = b.l / block;
    if(block_a != block_b) return block_a < block_b;
    return a.r < b.r;
}
void Mo(){
    // sort
    int cl = 1, cr = 0;
    for(auto i : queries){
        while(cl < i.l){} // remove
        while(cr > i.r){} // remove
        while(cl > i.l){} // add
        while(cr < i.r){} // add
    }
}
// Compress too big numsgives new nums to them
void compress(){
    vector<pair<int, int>> compress(n);
    rep(i, 1, n){
        cin >> nums[i];
        compress[i-1] = {nums[i], i};
    }
    sort(all(compress));
```

```
    int pre = compress[0].first, new_num = 0;
    nums[compress[0].second] = 0;
    for(auto it = compress.begin() + 1, end = compress.end();
          it != end; it++){
        if((*it).first != pre){
            pre = (*it).first;
            new_num++;
        }
        nums[(*it).second] = new_num;
    }
}
```

7.3 Segment$_{Tree}$

```
typedef struct {
    int set_val, add, sum, val;
} node;
int n, q; node tree[4*maxn]; int nums[maxn];
#define lc 2*now
#define rc 2*now+1
#define mid (L+R)/2 // LR is now range, lr is target range
// Pull
void pull(int now){ // update now with 2 children
    // use  lcrc  to undate now
    // tree[now].sum = tree[lc].sum + tree[rc].sum;
    // tree[now].prefix = max(tree[lc].sum+tree[rc].prefix,
         tree[lc].prefix);
    // tree[now].suffix = max(tree[lc].suffix+tree[rc].sum,
         tree[rc].suffix);
    // tree[now].middle_max = max(max(tree[lc].middle_max,
         tree[rc].middle_max), tree[lc].suffix+tree[rc].
         prefix);
    // tree[now].middle_max = max(max(tree[now].middle_max,
         tree[now].prefix), tree[now].suffix);
}
// Lazy
void push(int now, int child){
    if(tree[now].set_val){
        tree[child].set_val = 1;
        tree[child].val = tree[now].val;
        tree[child].add = tree[now].add;
    }
    else {
        tree[child].add += tree[now].add;
    }
}
void apply_tag(int now, int L, int R){
    if(tree[now].set_val)
        tree[now].sum = (R-L+1)*tree[now].val;
    tree[now].sum += (R-L+1)*tree[now].add;
    if(L != R){ // can go lower
        push(now, lc);
        push(now, rc);
    }
    tree[now].add = tree[now].set_val = 0; // Reset
}
// Build
void build(int L, int R, int now){
    if(L == R){
        // init tree[now];
        return;
    }
    int M = mid;
    build(L, M, lc);
    build(M + 1, R, rc);
    pull(now);
}
// modify
void modify(int l, int r, int L, int R, int now){
    if(R < l || r < L || L > n) // invalid range
        return;
    if(l <= L && R <= r){
        // modify tree[now];
        // tree[now].add += add; // modify_add
        // tree[now].set_val = 1; // modify_mod
  // tree[now].val = mod;
  // tree[now].add = 0; // Set is more prior
        return;
    }
    int M = mid;
    apply_tag(now, L, R);
    modify(l, r, L, M, lc);
    modify(l, r, M+1, R, rc);
    apply_tag(lc, L, M); // need
    apply_tag(rc, M+1, R); // need
    pull(now); // update now with 2 children
}
// query
ll query(int l, int r, int L, int R, int now){
    int M = mid;
    if(R < l || r < L || L > n){
        return 0;
    }
    // apply_tag(now, L, R); // Lazy to uncomment
    if(l <= L && R <= r){
        return tree[now].sum;
    }
 return query(l, r, L, M, lc) + query(l, r, M+1, R, rc);
}
// pizza_queries
// Left(s < t): dis_l = (pizza[s] - s) + t;
// Right(t < s): dis_r = (pizza[s] + s) - t;

// List Removals
// Use seg_tree to maintain how many nums have been selected
     in the range
// Use binary_Search to find "mod" nums have been selected
     before ans
// if ans - mod == posnums[ans] is the answerand we modify
     tree[pos]

// polynomial queries
//  Lazy_segset  under and distance
```

7.4 Treap

```
struct Treap {
    Treap *l, *r;
    int pri, subsize; char val; bool rev_valid;
    Treap(int _val){
        val = _val;
        pri = rand();
        l = r = nullptr;
        subsize = 1; rev_valid = 0;
    }
    void pull(){  // update subsize or other information
        subsize = 1;
        for(auto i: {l,r}){
            if(i) subsize += i->subsize;
        }
    }
};
int size(Treap *treap) {
    if (treap == NULL) return 0;
    return treap->subsize;
}
// lazy
void push(Treap *t){
    if(!t) return;
    if(t->rev_valid){
        swap(t->l, t->r);
        if(t->l) t->l->rev_valid ^= 1;
        if(t->r) t->r->rev_valid ^= 1;
    }
    t->rev_valid = false;
}
Treap *merge(Treap *a, Treap *b){
    if(!a || !b) return a ? a : b;
```

```
    // push(a); push(b);  // lazy
    if(a->pri > b->pri){
        a->r = merge(a->r, b);
        a->pull();
        return a;
    }
    else {
        b->l = merge(a, b->l);
        b->pull();
        return b;
    }

}
pair<Treap*, Treap*> split(Treap *root, int k) { // find 1~k
 if (root == nullptr) return {nullptr, nullptr};
    // push(root); // lazy
 if (size(root->l) < k) {
  auto [a, b] = split(root->r, k - size(root->l) - 1);
  root->r = a;
  root->pull();
  return {root, b};
 }
   else {
  auto [a, b] = split(root->l, k);
  root->l = b;
  root->pull();
  return {a, root};
 }
}
void Print(Treap *t){
    if(t){
        // push(t);   // lazy
        Print(t->l);
        cout << t->val;
        Print(t->r);
    }
}
void substring_rev(){
    int n, m; cin >> n >> m;
    Treap *root = nullptr;
    string str; cin >> str;
    for(auto c : str){
        root = merge(root, new Treap(c));
    }
    rep(i, 1, m){
        int x, y; cin >> x >> y;
        auto [a, b] = split(root, x-1); // a: 1~x-1, b: x~n
        auto [c, d] = split(b, y-x+1); // Use b to split
        // c->rev_valid ^= true;
        // push(c);
        b = merge(a, d);  // Notice the order
        root = merge(b, c);
    }
    Print(root);
}
```

8 TreeThm