NOI notes

Xin Anxiaoyang

January 2023

0 stress testing

```
@echo off
:again
gen > input
main < input > output.a
brute < input > output.b
fc output.a output.b > nul
if not errorlevel 1 goto again
```

1 Template

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

#define fi first
#define se second
#define pii pair<int,int>
#define pll pair<long long,long long>
#define pb push_back
#define debug(x) cerr<<#x<<"="<<x<<endl
#define pq priority_queue
#define inf (1ll<<60)
#define rep(i,a,b) for (int i=a;i<(b);i++)
#define MP make_pair
#define SZ(x) (int(x.size()))
#define ll long long
#define mod 1000000007
#define ALL(x) x.begin(),x.end()
#define endl "\n"
void inc(int &a,int b) {a=(a+b)%mod;}
void dec(int &a,int b) {a=(a-b+mod)%mod;}
int lowbit(int x) {return x&(-x);}
ll expo(ll base,ll p) {ll ret=1;while(p>0){if (p%2ll==1ll)
    ret=ret*base%mod;base=base*base%mod;p/=2ll;}return ret;}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);

    return 0;
}
//~ note to use // instead of / after copying full path
```

```
//~ freopen("input.in","r",stdin);
```

2 Graph Theory

2.1 DFS

```
void dfs(ll u){
    vis[u]=1;
    for(auto x:alist[u]){
            if(vis[x])continue;

            dfs(x);
    }
}
```

2.2 BFS

```
ll dist[maxn];
void bfs(ll st){
    queue<ll>q;
    q.push(st);
    dist[st]=0;
    while(!q.empty()){
        ll u=q.front();
        q.pop();
        for(auto x:alist[u]){
            if(dist[x]>dist[u]+1){
                dist[x]=dist[u]+1;
                q.push(x);
            }
        }
    }
}
//to init: memset(dist,63,sizeof dist);
```

2.3 BFS on grid

```
//if only grid
ll dy[4]={0,0,-1,1};
ll dx[4]={1,-1,0,0};
//for diagonals as well
int dx[]={0,0,1,-1,1,1,-1,-1};
int dy[]={1,-1,0,0,1,-1,1,-1};
int distance[maxn][maxn];
memset(distance,-1,sizeof(distance));
distance[my][mx]=0;
q.push({my,mx});
while(!q.empty()){
    auto curr=q.front();q.pop();
    int x=curr.fi,y=curr.se;
    rep(i,0,8){
        int xx=x+dx[i];
```

```
        int yy=y+dy[i];
        if(!valid(xx,yy))continue;
        q.push({xx,yy});
        distance[xx][yy]=distance[x][y]+1;
    }
}
```

2.4 01 bfs

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

2.5 Checking bipartite graph

```
vector<ll>alist[100005];
ll visited[100005];
ll color[100005];
void dfs(ll node){
    visited[node]=true;
    for(auto x:alist[node]){
        if(!visited[x]){
            color[x]=color[node]%2+1;
            dfs(x);
        }else{
            if(color[x]==color[node]){
                cout<<"IMPOSSIBLE";exit(0);
            }
        }
    }
}
// to init:
dfs(1);
```

2.6 Cycle detection(directed)

```
vector<ll>alist[maxn];
```

```
ll color[maxn];
bool cycle=false;
void dfs(ll u){
    color[u]=1;
    for(auto x:alist[u]){
        if(color[x]==0){
            dfs(x);
        }
        else if(color[u]==1){
            cycle=true;
            return;
        }
    }
    color[u]=2;
}

// to init:
rep(i,1,n+1){
    if(color[i]==0)dfs(i);
}
cout<<cycle;
```

2.7 Toposort(DAG)

```
vector<int>alist[100000];
int visited[100000];
vector<int>topo;
void dfs(int v){
    visited[v]=true;
    for(auto x: alist[v]){
        if(!visited[x]){
            dfs(x);
        }
    }
    topo.pb(v);
}
void topsort(){
    memset(visited,0,sizeof visited);
    rep(i,0,n){
        if(!visited[i])dfs(i);
    }
    reverse(ALL(topo));
}
```

2.8 Dijkstra

```
vector<pll>alist[maxn];
pq<pll,vector<pll>,greater<pll>>s; //weight of edge,node
ll dist[maxn],big;
void dijkstra(){
    memset(dist,63,sizeof dist);
    big=dist[0];
    dist[1]=0;
    s.push({0,1});
    while(!s.empty()){
```

```
        pll u=s.top();
        s.pop();
        if(u.fi!=dist[u.se])continue;
        for(auto v:alist[u.se]){
            if(dist[v.se]>dist[u.se]+v.fi){
                dist[v.se]=dist[u.se]+v.fi;
                s.push({dist[v.se],v.se});
            }
        }
    }
}
```

2.9 Graph duplication

cyberland: Given an undirected weighted graph with N nodes and M edges. Some nodes can clear the current distance to 0, while some can divide the distance by 2. In each visit, you can only use the special ability at most once, and you must use the ability on the node. Find the shortest path from 0 to H, if you can not visit H twice, and the divide-by-2 ability can be used at most K times.

```
vector<pll>alist[111111];
double solve(int n, int m, int k, int h, std::vector<int> x, std::vector<int> y, std::vector<int>
    c, std::vector<int> arr) {
    double dist[n][k+1];
    double inf=1e18;
    rep(i,0,n){
        rep(j,0,k+1){
            dist[i][j]=inf;
        }
    }
    rep(i,0,m){
        alist[x[i]].pb(MP(c[i],y[i]));
        alist[y[i]].pb(MP(c[i],x[i]));
    }
    pq<pair<double,pll>,vector<pair<double,pll>>,greater<pair<double,pll>>>s;
    dist[0][0]=0;
    s.push(MP(0,MP(0,0)));
    while(!s.empty()){
        auto tmp=s.top();
        s.pop();
        double d=tmp.fi;
        ll u=tmp.se.fi;
        ll kk=tmp.se.se;
        if(abs(d-dist[u][kk])>0.0000000001 or u==h)continue;
        for(auto x:alist[u]){
            if(arr[x.se]==0 and dist[x.se][0]!=0){//set 0
                dist[x.se][0]=0;
                s.push(MP(0,MP(x.se,0)));
                continue;
            }
            if(arr[x.se]==2 and kk<k){//half time
                if(dist[x.se][kk+1]>(d+x.fi)/2){
                    dist[x.se][kk+1]=(d+x.fi)/2;
                    s.push(MP(dist[x.se][kk+1],MP(x.se,kk+1)));
                }
            }
            if(dist[x.se][kk]>d+x.fi){
                dist[x.se][kk]=d+x.fi;
```

```
                s.push(MP(dist[x.se][kk],MP(x.se,kk)));
            }

        }
    }
    double ans=inf;
    rep(i,0,k+1)ans=min(ans,dist[h][i]);
    rep(i,0,n)alist[i].clear();
    if(ans!=inf)return ans;
    return -1;
}
```

2.10 Floyd Warshall

```
rep(i,0,n){
        rep(j,0,n){
            if(i==j)f[i][j]=0;
            else f[i][j]=inf;
        }
}
rep(i,0,e){
        ll a,b,c;cin>>a>>b>>c;
        f[a][b]=c;
        f[b][a]=c;
}
rep(k,0,n){
        rep(i,0,n){
            rep(j,0,n){
                    f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
            }
        }
}
```

2.11 Bellman ford

```
memset(dist,63,sizeof dist);
ll big=dist[0];
queue<pii>q;q.push({1,0});
dist[1]=0;
while(!q.empty()){
        pii v=q.front();
        q.pop();
        for(auto x:alist[v.fi]){
            if(dist[v.fi]+x.se<dist[x.fi]){
                  dist[x.fi]=dist[v.fi]+x.se;
                  q.push(x);
            }
        }
}
```

2.12 SCC+dp on dag :clown

A game has n rooms and m tunnels between them. Each room has a certain number of coins. What is the maximum number of coins you can collect while moving through the tunnels when you can freely choose your starting and ending room?

```
const ll maxn=111111;
vector<ll>edge[maxn],egde[maxn],edge_scc[maxn];
vector<ll>topo,component;
ll vis[maxn],val[maxn],val_scc[maxn];
ll roots[maxn],memo[maxn];
ll num=0,tot=0;
void dff(ll u){
    vis[u]=1;
    for(auto x:edge[u]){
        if(!vis[x]){
            dff(x);
        }
    }
    topo.pb(u);
}
void dfs(ll u){
    vis[u]=1;
    tot+=val[u];
    component.pb(u);
    for(auto x:egde[u]){
        if(!vis[x]){
            dfs(x);
        }
    }
}
ll dp(ll u){
    if(memo[u]!=-1)return memo[u];
    memo[u]=val_scc[u];
    for(auto x:edge_scc[u]){
        memo[u]=max(memo[u],dp(x)+val_scc[u]);
    }
    return memo[u];
}
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    //freopen("i.txt","r",stdin);
    ll n,m;cin>>n>>m;
    rep(i,1,n+1)cin>>val[i];
    rep(i,0,m){
        ll a,b;cin>>a>>b;
        edge[a].pb(b);// a can reach b
        egde[b].pb(a);// b can be reached from a;
    }
    memset(vis,0,sizeof vis);
    rep(i,1,n+1){
        if(!vis[i])dff(i);
    }
    memset(vis,0,sizeof vis);
    reverse(ALL(topo));//by decreasing endtime
    //debug(333);
    for(auto u:topo){
        if(!vis[u]){
```

```
            tot=0;
            dfs(u);
            ll root=component.front();
            val_scc[root]=tot;
            for(auto x:component)roots[x]=root;
            component.clear();
        }
    }
    //debug(222);
    rep(i,1,n+1){
        for(auto x:edge[i]){
            ll u=roots[i];
            ll v=roots[x];
            if(u!=v){
                //edge_scc[u].pb(v);
                edge_scc[v].pb(u);//reverse edges
            }
        }
    }
    //debug(111);
    //rep(i,1,5)cout<<val_scc[i]<<" ";
    //cout<<endl;
    /*
    rep(i,0,5){
        cout<<i<<": ";
        for(auto x:edge_scc[i]){
            cout<<x<<" ";
        }cout<<endl;
    }
    */
    memset(memo,-1,sizeof memo);
    rep(i,1,n+1){
        dp(i);
    }
    ll mx=0;
    rep(i,1,n+1){
        mx=max(mx,memo[i]);
    }
    cout<<mx<<endl;
    return 0;
}
```

3 Trees

3.1 Tree traversal

```
void dfs(ll u,ll p){
    for(auto x:alist[u]){
         if(x==p)continue;

         dfs(x,u);
    }
}
```

3.2 Tree flattening

```
ll tot=0;
void dfs(ll u,ll p){
    pre[u]=++tot;
    for(auto v:edge[u]){
        if(v==p)continue;
        dfs(v,u);
    }
    post[u]=tot;
}
//to do subtree updates queries just query(pre[u],post[u])
```

3.3 Subtree size and depth of node

```
vector<int>alist[111111];
bool vis[111111];
ll depth[111111],sz[111111];
void dfs(ll u, ll p){
    sz[u]=1;
    for(auto x:alist[u]){
        if(x==p)continue;
        depth[x]=depth[u]+1;
        dfs(x,u);
        sz[u]+=sz[x];
    }
}
//to init:
dfs(1,-1);
```

3.4 Kth par

```
int kpar(int x,int k){
    for (int i=0;i<20;i++){
        if (k & (1<<i)) x = twok[x][i];
        if (x==-1) return x;
    }
    return x;
}

memset(twok, -1, sizeof twok);
dfs(0,-1);
// initialize twok[x][0] = par[x]
for (int k=1;k<20;k++){
    for (int x=1;x<n;x++){
        if (twok[x][k-1]!=-1)
            twok[x][k] = twok[twok[x][k-1]][k-1];
    }
}
```

3.5 LCA

```
const ll maxn=200005;
const ll lg=20;

ll up[lg][maxn];
ll depth[maxn];
vector<ll>alist[maxn];

void dfs(ll u,ll p){
    up[0][u]=p;
    rep(i,1,lg){
        if(up[i-1][u]==-1)break;
        up[i][u]=up[i-1][up[i-1][u]];
    }
    for(auto x:alist[u]){
        if(x!=p){depth[x]=depth[u]+1;dfs(x,u);}
    }
}
ll lca(ll a,ll b){
    if(depth[a]<depth[b])swap(a,b); // let a be the deeper one
    ll k=depth[a]-depth[b];
    rep(i,0,lg){
        if((1<<i)&k)a=up[i][a];
    }
    if(a==b)return a;
    for(ll i=19;i>=0;i--){
        if(up[i][a]!=up[i][b]){
            a=up[i][a];
            b=up[i][b];
        }
    }
    return up[0][a];
}
//to initialise:
memset(up,-1,sizeof up);
dfs(1,-1);
```

3.6 distance between nodes on a tree

```
distance(a,b) = dist[a]+dist[b]-2*dist[lca(a,b)];
```

3.7 path max queries

```
void dfs(ll u,ll p,ll e){
    up[0][u]=p;
    mx[0][u]=e;
    rep(i,1,20){
        if(up[i-1][u]==-1)break;
        up[i][u]=up[i-1][up[i-1][u]];
        mx[i][u]=max(mx[i-1][u],mx[i-1][up[i-1][u]]);
    }
    for(auto x:tree[u]){
        if(x.fi==p)continue;
        depth[x.fi]=depth[u]+1;
        dfs(x.fi,u,x.se);
```

```
    }
}
ll lca(ll u,ll v){
    if(depth[v]>depth[u])swap(u,v);
    ll diff=depth[u]-depth[v];
    ll ret=0;
    rep(i,0,20){
          if((1<<i)&diff){
            ret=max(ret,mx[i][u]);
            u=up[i][u];
          }
    }
    if(u==v)return ret;
    for(ll i=19;i>=0;i--){
          if(up[i][u]!=up[i][v]){
               ret=max({mx[i][u],mx[i][v],ret});
            u=up[i][u];
            v=up[i][v];
          }
    }
    return max({ret,mx[0][u],mx[0][v]});
}
// initialise:
memset(up,-1,sizeof up);
memset(up,-1,sizeof up);
dfs(1,-1,0);
```

3.8 Finding centriod

```
const ll maxn=2e5+5;
vector<ll>alist[maxn];
ll sz[maxn],wt[maxn];
ll n,centriod;
//
//all subtree size from root are <=n/2
//can have up to 2 centriod
//sum of distances to every node from centriod is the lowest
//if 2 trees are joined together, the new centriod is on the simple path from one centriod to the
    other from another tree
//removing or adding an leaf node will let centriod move by at most one unit
void dfs(ll u,ll p){
    sz[u]=1;
    for(auto x:alist[u]){
        if(x==p)continue;
        dfs(x,u);
        sz[u]+=sz[x];
        wt[u]=max(wt[u],sz[x]);
    }
    wt[u]=max(wt[u],n-sz[u]);
    if(wt[u]<=n/2)centriod=u;
}
//to init:
dfs(1,-1);
```

3.9 Kruskal mst

UFDS not included in code

```
struct edge{
    ll a,b,c;
};
vector<edge>vec;
vector<pii>alist[maxn];
bool cmp(edge x,edge y){
    return x.c<y.c;
}

rep(i,0,e){
        edge x;
        cin>>x.a>>x.b>>x.c;
        vec.pb(x);
}
sort(ALL(vec),cmp);
for(auto x:vec){
        if(!sameset(x.a,x.b)){
            mergeset(x.a,x.b);
            alist[x.a].pb({x.b,x.c});
            alist[x.b].pb({x.a,x.c});
        }
}
```

3.10 Set merging

black magic (below is just an example)

```
void dfs(ll u,ll p){
    for(auto x:alist[u]){
        if(x==p)continue;
        dfs(x,u);
        if(SZ(s[x])>SZ(s[u]))swap(s[x],s[u]);//merge small to large
        for(auto c:s[x]){
            s[u].insert(c);
        }
    }
    ans[u]=SZ(s[u]);
}
```

3.11 Tree diameter

dp version:

```
const ll maxn=3e5+5;
ll f[maxn][2]; // 0 for shortest path, 1 for second shortest path
vector<ll>alist[maxn];
ll d=0;
void dfs(ll u,ll p){
    f[u][0]=f[u][1]=0;
    for(auto x:alist[u]){
        if(x==p)continue;
        dfs(x,u);
        ll tmp=f[x][0]+1;
```

```
        if(tmp>f[u][0]){
            f[u][1]=f[u][0];
            f[u][0]=tmp;
        }
        else if(tmp>f[u][1]){
            f[u][1]=tmp;
        }
    }
    d=max(d,f[u][0]+f[u][1]);
}
// to init:
dfs(1,-1);
```

dfs version:

```
const ll maxn=3e5+5;
vector<ll>alist[maxn];
ll d=0,deepest=-1,depth[maxn];
void dfs(ll u, ll p){
    for(auto x:alist[u]){
        if(x==p)continue;
        depth[x]=depth[u]+1;
        if(deepest==-1 or depth[x]>depth[deepest]){
            deepest=x;
        }
        dfs(x,u);
    }
}
// init:
dfs(1,-1);
depth[deepest]=0;
dfs(deepest,-1);
cout<<depth[deepest];
```

3.12 HLD

```
const ll maxn=500005;
vector<ll>edge[maxn];
ll f[maxn],dep[maxn],sz[maxn],son[maxn],id[maxn],tp[maxn];

ll n,q;
ll tot=1;
ll tree[maxn<<2],add[maxn<<2];
void dfs1(ll u,ll fa){
    dep[u]=dep[fa]+1;
    sz[u]=1;
    f[u]=fa;
    son[u]=0;
    for(auto v:edge[u]){
        if(v==fa)continue;
        dfs1(v,u);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]])son[u]=v;
    }
    return;
}
void dfs2(ll u,ll t){
```

```
    tp[u]=t;
    id[u]=tot++;
    if(son[u]!=0){
        dfs2(son[u],t);
    }
    for(auto v:edge[u]){
        if(v==f[u] or v==son[u])continue;
        dfs2(v,v);
    }
    return;
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>n>>q;
    rep(i,0,n-1){
        ll a,b;cin>>a>>b;
        edge[a].pb(b);
        edge[b].pb(a);
    }
    dfs1(1,-1);
    dfs2(1,1);
    while(q--){
        char c;cin>>c;
        ll u,v;cin>>u>>v;
        if(c=='P'){
            while(tp[u]!=tp[v]){
                if(dep[tp[u]]<dep[tp[v]])swap(u,v);
                update(1,1,n,id[tp[u]],id[u],1);
                u=f[tp[u]];
            }
            if(id[u]>id[v])swap(u,v); //same chain
            if(u!=v)update(1,1,n,id[u]+1,id[v],1); //upd from u to v
        }else{
            ll sum=0;
            while(tp[u]!=tp[v]){
                if(dep[tp[u]]<dep[tp[v]])swap(u,v);
                sum+=query(1,1,n,id[tp[u]],id[u]);
                u=f[tp[u]];
            }
            if(id[u]>id[v])swap(u,v); //same chain
            if(u!=v)sum+=query(1,1,n,id[u]+1,id[v]); //qry from u to v
            //if its on nodes and not edges, do:
            //sum+=query(1,1,n,id[u],id[v]);
            cout<<sum<<endl;
        }
    }
    return 0;
}
```

4 Greedy

4.1 Exchange arguments

```
Try consider cases where abcdeXYfghijkl works and abcdeYXfghijkl doesn't work, what's the
    condition such that XY works and YX doesn't.

Or try considering when once case is better than another case and then do algebra to figure out
    why and sort in that order
```

5 Dynamic programming

5.1 Knapsack

```
ll dp[105][100005]; // number, weight, stores the max val
//if want opt memory do dp[2][100005](ref to below knapsack)
ll wt[105],val[105];
rep(i,1,n+1){
       rep(j,0,w+1){
           dp[i][j]=dp[i-1][j];
           if(j-wt[i]>=0)
               dp[i][j]=max(dp[i][j],dp[i-1][j-wt[i]]+val[i]);
       }
}
cout<<dp[n][w];
```

5.2 Knapsack 1d

```
rep(i,1,n+1){
       for(ll j=w;j>=wt[i];j--){
           f[j]=max(f[j],f[j-wt[i]]+val[i]);
       }
}
cout<<f[w];
```

5.3 unbounded knapsack

```
rep(i,1,m+1)cin>>ti[i]>>va[i];
ll f[m+5][t+5];
memset(f,0,sizeof f);
rep(i,1,m+1){
       rep(j,1,t+1){
           rep(k,0,(j/ti[i])+1){
                  f[i][j]=max(f[i][j],f[i-1][j- k*ti[i]]+k*va[i]);
           }
       }
}
cout<<f[m][t];
```

5.4 knapsack

```
ll s,n;cin>>s>>n;
vector<ll>wt,val;
rep(i,0,n){
```

```
    ll v,w,k;cin>>v>>w>>k;
    ll p=1;
    while(k>=p){
        wt.pb(p*w);
        val.pb(p*v);
        k-=p;
        p*=2;
    }
    if(k>0){
        wt.pb(k*w);
        val.pb(k*v);
    }
}
rep(i,1,SZ(val)+1){
    rep(j,0,s+1){
        dp[i%2][j]=dp[(i-1)%2][j];
          if(j-wt[i-1]>=0)dp[i%2][j]=max(dp[i%2][j],dp[(i-1)%2][j-wt[i-1]]+val[i-1]);
    }
}
cout<<dp[SZ(val)%2][s];
```

5.5 LIS

```
//dp[i] is the longest increasing subsequence including i as the last element
rep(i,0,n){
       dp[i]=1;
       rep(j,0,i){
           if(alist[j]<alist[i]){
                  dp[i]=max(dp[j]+1,dp[i]);
           }
       }
}
cout<<*max_element(dp,dp+n);
```

5.6 LIS seg tree

```
ll ans=0;
//dp[i]=max(dp[j])+1; 1<=j<i
rep(i,1,n+1){
       dp=query(1,1,mx,1,a[i]-1);//range max query
       update(rt,1,mx,a[i],dp+1);//point update
       ans=max(dp,ans);
}
cout<<ans;
```

5.7 LIS binary search

```
ll f[maxn],a[maxn];
// f[i]   will be the smallest element at which an increasing subsequence of length i ends
memset(f,63,sizeof f);
ll big=f[0];
f[0]=-inf;
rep(i,0,n){
```

```
    auto j=upper_bound(f,f+n,a[i])-f;
    if(f[j-1]<a[i] and a[i]<f[j])f[j]=a[i];
}
ll ans=0;
rep(i,1,n+1){
        if(f[i]<big)ans=i;
}
cout<<ans;
```

6 Data structures

6.1 UFDS

```
const ll maxn=1e5+5;
ll p[maxn],sz[maxn];
void init(){
    rep(i,0,maxn)sz[i]=1,p[i]=i;
}
ll findset(ll u){
    if(p[u]==u)return u;
    return p[u]=findset(p[u]);
}
bool sameset(ll u,ll v){
    return findset(u)==findset(v);
}
void mergeset(ll a,ll b){
    a=findset(a);
    b=findset(b);
    if(a!=b){
        if(sz[a]<sz[b])swap(a,b);
        p[b]=a;
        sz[a]+=sz[b];
    }
}
```

6.2 Fenwick

```
const ll maxn=1e6+5;
ll tree[maxn];
void update(ll p,ll val){
    while(p<maxn){
        tree[p]+=val;
        p+=lowbit(p);
    }
}
ll query(ll p){
    ll ret=0;
    while(p){
        ret+=tree[p];
        p-=lowbit(p);
    }
    return ret;
}
```

6.3 Segment tree

```
ll tree[maxn<<2];
void update(int c,int cl,int cr,int pos, int val){
    if(cl==cr){
        tree[c]=val;
        return;
    }
    ll mid=cl+cr>>1;
    if(pos<=mid)update(c<<1,cl,mid,pos,val);
    if(mid<pos)update(c<<1|1,mid+1,cr,pos,val);
    tree[c]=min(tree[c<<1],tree[c<<1|1]);
}
ll query(ll c,ll cl,ll cr,ll l,ll r){
    if(l<=cl and cr<=r)return tree[c];
    ll mid=cl+cr>>1;
    ll ret=2e9;
    if(l<=mid)ret=min(ret,query(c*2,cl,mid,l,r));
    if(r>mid)ret=min(ret,query(c*2+1,mid+1,cr,l,r));
    return ret;
}
```

6.4 Lazy prop seg tree

```
const ll maxn=100005;
ll tree[maxn<<2],lazy[maxn<<2];
//note that prop for range add and range max is diff
void pushdown(ll c,ll cl,ll cr){
    lazy[c<<1]+=lazy[c];
    lazy[c<<1|1]+=lazy[c];
    tree[c<<1]+=lazy[c];
    tree[c<<1|1]+=lazy[c];
    lazy[c]=0;
}
void update(ll c,ll cl,ll cr,ll l,ll r,ll val){
    if(l<=cl and cr<=r){
        tree[c]+=val;
        lazy[c]+=val;
        return;
    }
    pushdown(c,cl,cr);
    ll mid=(cl+cr)>>1;
    if(l<=mid)update(c<<1,cl,mid,l,r,val);
    if(r>mid)update(c<<1|1,mid+1,cr,l,r,val);
    tree[c]=max(tree[c<<1],tree[c<<1|1]);
}

ll query(ll c,ll cl,ll cr,ll l,ll r){
    if(l<=cl and cr<=r){
        return tree[c];
    }
    pushdown(c,cl,cr);
    ll mid=(cl+cr)>>1;
    ll ret=-inf;
    if(l<=mid)ret=max(ret,query(c<<1,cl,mid,l,r));
    if(r>mid)ret=max(ret,query(c<<1|1,mid+1,cr,l,r));
```

```
    return ret;
}
```

6.5 AP queries

```
rep(i,1,n+1)diff[i]=a[i]-a[i-1];
build(1,1,n);
ll ans=0;
rep(i,1,n+1){
    ans+=abs(query(1,1,n,1,i));
    ll l=i,r=n;
    ll d=-query(1,1,n,1,i),k=-query(1,1,n,1,i);
    update(1,1,n,l,l,k);
    if(l+1<=r)update(1,1,n,l+1,r,d);
    if(r+1<=n)update(1,1,n,r+1,r+1,-(k+d*(r-l))); rep(i,1,n+1)cout<<(query(1,1,n,1,i))<<" ";
    cout<<endl;
}
//do operations on the diff array instead
```

6.6 Lazy create seg tree

```
const ll maxn=5e6+5;
//just init super big for lazy create
ll lc[maxn],rc[maxn],tree[maxn],add[maxn];
ll tot=1,rt=1;

void pushdown(ll c,ll cl, ll cr){
    if(lc[c]==-1)lc[c]=++tot;
    if(rc[c]==-1)rc[c]=++tot;
    ll mid=cl+cr>>1;
    tree[lc[c]]+=add[c]*(mid-cl+1);
    add[lc[c]]+=add[c];
    tree[rc[c]]+=add[c]*(cr-mid);
    add[rc[c]]+=add[c];
    add[c]=0;
}
void update(ll &c,ll cl, ll cr, ll l, ll r, ll val){
    if(c==-1)c=++tot;
    if(l<=cl and cr<=r){
        tree[c]+=val*(cr-cl+1);
        add[c]+=val;
        return;
    }
    pushdown(c,cl,cr);
    ll mid=cl+cr>>1;
    if(l<=mid)update(lc[c],cl,mid,l,r,val);
    if(r>mid)update(rc[c],mid+1,cr,l,r,val);
    tree[c]=tree[lc[c]]+tree[rc[c]];
    return;
}
ll query(ll c,ll cl, ll cr, ll l, ll r){
    if(c==-1)return 0;
    if(l<=cl and cr<=r)return tree[c];
    pushdown(c,cl,cr);
    ll mid=cl+cr>>1;
```

```
    ll ans=0;
    if(l<=mid)ans+=query(lc[c],cl,mid,l,r);
    if(r>mid)ans+=query(rc[c],mid+1,cr,l,r);
    return ans;
}
//  memset(lc,-1,sizeof lc);
//  memset(rc,-1,sizeof rc);
```

6.7 PBDS

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<long long, null_type, less_equal<long long>,rb_tree_tag,
    tree_order_statistics_node_update>ordered_set;

//find_by_order()
//returns an iterator to the k-th largest element (counting from zero)

//order_of_key()
//the number of items in a set that are strictly smaller than our item

ordered_set X;
X.insert(1);
X.insert(2);
X.insert(4);
X.insert(8);
X.insert(16);

cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<*X.find_by_order(4)<<endl; // 16
cout<<(end(X)==X.find_by_order(6))<<endl; // true

cout<<X.order_of_key(-5)<<endl; // 0
cout<<X.order_of_key(1)<<endl; // 0
cout<<X.order_of_key(3)<<endl; // 2
cout<<X.order_of_key(4)<<endl; // 2
cout<<X.order_of_key(400)<<endl; // 5
```

7 Scams

7.1 Brute force + clock + rng

If qn is output just YES or NO, we could run bruteforce, pray that it covers enough cases, then right before tle, if no solution go do rng() and hope ACs all remaining tcs

```
mt19937 rng(chrono::system_clock::now().time_since_epoch().count());
int st=clock(); //put as global variable at start

//place in bruteforce solution
if(clock()-st>0.90*CLOCKS_PER_SEC){// adjust according to time limit
    if(rng()%2)cout<<"YES";
    else cout<<"NO";
    return 0;
```

```
}
```

7.2 Pragmas

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// note: may not work locally depending on compiler and CE
```

7.3 fast input template

```
void fastscan(int &x){
    bool neg=false;
    register int c;
    x =0;
    c=getchar();
    if(c=='-'){
        neg = true;
        c=getchar();
    }
    for(;(c>47 && c<58);c=getchar())
        x = (x<<1) + (x<<3) +c -48;
    if(neg)
        x *=-1;
}
```

8 Search

8.1 All permutations

```
vector<ll>alist;
rep(i,1,n+1)alist.pb(i);
do{
    //check if permutation valid
}while(next_permutation(ALL(alist)));
//Note:
make sure the list is sorted before applying next perm
```

8.2 Bitmask

```
rep(mask,0,(1<<n)){// all subsets
    rep(i,0,n){
        if((1<<i)&mask){//if ith bit is 1
            //do smt
        }
    }
}
```

8.3 Backtracking

Isn't really library code, just here for inspiration

```
void solve(int anything){
    if(condition){
        // do smt;
        return;
    }
    for(int i=0;i<something;i++){
        take[i]=1;
        solve(whatever);
        take[i]=0;
    }
}
```

8.4 Binary search

Hint: If minimise the maximum or maximise the minimum, try bsta

```
ll lo=0,hi=n;
while(lo<hi){
    ll mid=hi+lo>>1;
    //if inf loop try hi+lo+1>>1, may be off by 1 error;
    if(solve())
    else
}
cout<<lo;
```

9 Others

9.1 2d prefix sums

```
rep(i,1,r+1){
    rep(j,1,c+1){
        f[i][j]=f[i][j]+f[i-1][j]+f[i][j-1]-f[i-1][j-1];
    }
}
```

9.2 2 pointers

O(n^2) to O(n) many diff variants(one pointer inc/dec as another inc), below just one e.g.

```
ll l=1,r=n;
while(l<r){
    if()r--;
    l++;
}
```

9.3 Monotonic stack/deque

```
//monotonic stack
rep(i,0,n){
       ll a;cin>>a;
       while(!s.empty() and s.top()<=a){
           s.pop();
       }
       s.push(a);
}

//monotonic deque
deque<ll>q;q.pb(0);//stores indices of elements
rep(i,1,n+2){
       while(!q.empty() and q.front()<i-k)q.pop_front();
       // do smt
       while(!q.empty() and arr[q.back()]>arr[i])q.pop_back();
       q.pb(i);
}
```

9.4 sieve of eratosthenes

```
memset(sieve,0,sizeof sieve);
rep(i,2,maxn)sieve[i]=1;
rep(i,2,maxn){
       if(sieve[i]){
           for(int j=i+i;j<maxn;j+=i){
               sieve[j]=0;
           }
       }
}
```

9.5 Exponentiation

```
ll expo(ll a,ll b, ll m){
    if(b==1)return a;
    else{
        if(b%2==0)return (expo(a,b/2,m)%m*expo(a,b/2,m)%m)%m;
        else return ((expo(a,b/2,m)%m*expo(a,b/2,m)%m)%m*a%m)%m;
    }
}
//OR
int a,b,m;
cin>>a>>b>>m;
int res=1;
while(b>0){
    if(b&1)res=(res%m*a%m)%m;
    a=(a%m*a%m)%m;
    b>>=1;
}
cout<<res;
```

9.6 Discretize

```
vector<ll> discretize(vector<ll> V){
    vector<ll> dis = V;
    vector<ll> v(V.size());
    sort(dis.begin(), dis.end());
    dis.resize(unique(dis.begin(), dis.end()) - dis.begin());
    for (ll i = 0; i < V.size(); i++){
        v[i] = lower_bound(dis.begin(), dis.end(), V[i]) - dis.begin() + 1;
    }
    return v;
}
```

9.7 Tree Gen

note the tree generated here kinda sucks, but its the simplest code

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll n=20;
    cout<<n<<endl;
    for(long i = 2; i <= n; ++i) cout << (rng() % (i - 1) + 1) << " " << i << "\n";
    return 0;
}
```

9.8 MO's algo

```
const ll maxn=1e6+5;
const ll block=1000;
ll a[maxn],queries[maxn];
struct query{
    ll l,r,idx;
};
vector<query>qq;
bool cmp(query a,query b){
    if(a.l/block!=b.l/block){
        return a.l/block<b.l/block;
    }
    else if(a.r!=b.r){
        return a.r<b.r;
    }
}
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll n,q;cin>>n>>q;
    rep(i,1,n+1){
        cin>>a[i];
    }
    rep(i,1,q+1){
        query tmp;
        cin>>tmp.l>>tmp.r;
        tmp.idx=i;
        qq.pb(tmp);
    }
```

```
    sort(ALL(qq),cmp);
    ll curr_l=1,curr_r=0;
    ll ans=0;
    for(auto x:qq){
        while(curr_l>x.l){
            curr_l--;
            ans+=a[curr_l];
        }
            while (curr_r < x.r) {
                   curr_r++;
            ans+=a[curr_r];
            }
            while (curr_l < x.l) {
                   ans-=a[curr_l];
                   curr_l++;
            }
            while (curr_r > x.r) {
                   ans-=a[curr_r];
                   curr_r--;
            }
        queries[x.idx]=ans;
    }
    rep(i,1,q+1)cout<<queries[i]<<endl;
    return 0;
}
```

9.9 string hashing :clown

```
long long compute_hash(string const& s) {
    const int p = 31; //about the same as the number of different characters for the string
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}

//prefix sums of hash idea, no need mod inverse
//idea: number of distinct permutations of N which appear as a substring of H at least once
p_pow[0] = 1;
rep(i,1,m+1)p_pow[i] = (p_pow[i - 1] * p) % mod;
rep(i,0,m+5)h[i+1]=(h[i]+(M[i]-'a'+1)*p_pow[i])%mod;
rep(i,0,m-n+1){
       bool same=1;
       rep(j,0,26){
           if(cnt1[j]!=cnt2[j])same=false;
       }
       if(same){
           ll curr_h=0;
           inc(curr_h,h[i+n]);
           dec(curr_h,h[i]);
           curr_h=(curr_h*p_pow[m-i-1])%mod;
           if(f[curr_h]==0)ans++;
           f[curr_h]++;
```

```
        }
        cnt2[int(M[i]-'a')]--;
        cnt2[int(M[i+n]-'a')]++;
}


//if all fails use this
class HashedString {
  private:
    // change M and B if you want
    static const ll M = (1LL << 61) - 1;
    static const ll B;

    // pow[i] contains B^i % M
    static vector<ll> pow;

    // p_hash[i] is the hash of the first i characters of the given string
    vector<ll> p_hash;

    __int128 mul(ll a, ll b) { return (__int128)a * b; }
    ll mod_mul(ll a, ll b) { return mul(a, b) % M; }

  public:
    HashedString(const string &s) : p_hash(s.size() + 1) {
        while (pow.size() < s.size()) { pow.push_back(mod_mul(pow.back(), B)); }
        p_hash[0] = 0;
        for (int i = 0; i < s.size(); i++) {
            p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M;
        }
    }

    ll getHash(int start, int end) {
        ll raw_val =
            p_hash[end + 1] - mod_mul(p_hash[start], pow[end - start + 1]);
        return (raw_val + M) % M;
    }
};
```

9.10 Fermat's little thm

```
division with mod p (p is prime) : a/b (mod p) = (a%mod*expo(b,mod-2,mod))%mod
```

10 DS but more rare

10.1 CHT

slope is monotonic

```
struct line{
    ll m=0,c=-1e18;
    ll operator()(ll x){return m*x+c;}
};
deque<line>dq;
long double intersect(line a,line b){
    return (long double)(a.c-b.c)/(b.m-a.m);
```

```
}
void insert(line y){
    while(dq.size()>1){
        ll s=dq.size();
        if(intersect(dq[s-1],dq[s-2])>intersect(dq[s-1],y))dq.pop_back();
        else break;
    }
    dq.pb(y);
}
ll query(ll x){
    while(dq.size()>1){
        if(dq[0](x)<dq[1](x))dq.pop_front();
        else break;
    }
    return dq[0](x);
}
```

10.2 CHT dynamic ver

```
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

10.3 Lichao(lazy)

```
const ll maxn=1e6+5;
struct line{
    ll m=0,c=-1e18;//1e18 for min query
```

```
    ll operator()(ll x){return m*x+c;}
};
line tree[maxn];
ll lc[maxn],rc[maxn],tot=1;
void insert(ll &idx,ll l,ll r,line y){
    if(idx==-1)idx=++tot;
    if(l==r)return;
    ll mid=l+r>>1;
    if(tree[idx](mid)<y(mid))swap(tree[idx],y);
    if(y(l)>tree[idx](l))insert(lc[idx],l,mid,y);//< for min query
    else insert(rc[idx],mid+1,r,y);
}
ll query(ll idx,ll l, ll r, ll x){
    if(idx==-1)return -inf; // inf for min query
    if(l==r)return tree[idx](x);
    ll mid=l+r>>1;
    if(x<=mid)return max(tree[idx](x),query(lc[idx],l,mid,x));
    return max(tree[idx](x),query(rc[idx],mid+1,r,x));
}
//to init:
memset(lc,-1,sizeof lc);
memset(rc,-1,sizeof rc);
ll rt=1;
```

10.4 Kth largest

If wavelet is intended guess I will die $O(n \log^3 n)$ method

```
const ll maxn=1e5+5;
vector<ll>tree[maxn<<2];
ll a[maxn];
vector<ll>combine(vector<ll>a,vector<ll>b){
    ll p=0,pp=0;
    vector<ll>ret;
    while(p<SZ(a) and pp<SZ(b)){
        if(a[p]<b[pp]){
            ret.pb(a[p]);p++;
        }else{
            ret.pb(b[pp]);pp++;
        }
    }
    if(pp==SZ(b)){
        rep(i,p,SZ(a))ret.pb(a[i]);
    }else{
        rep(i,pp,SZ(b))ret.pb(b[i]);
    }
    return ret;
}
void build(ll c,ll cl,ll cr){
    if(cl==cr){
        tree[c].pb(a[cl]);
        return;
    }
    ll mid=cl+cr>>1;
    build(c<<1,cl,mid);
    build(c<<1|1,mid+1,cr);
    tree[c]=combine(tree[c<<1],tree[c<<1|1]);
}
```

```
ll query(ll c,ll cl,ll cr,ll l,ll r,ll k){
    if(r<cl or l>cr)return 0;
    if(l<=cl and cr<=r){
        return lower_bound(ALL(tree[c]),k)-tree[c].begin();
    }
    ll mid=cl+cr>>1;
    ll ret=0;
    if(l<=mid)ret+=query(c<<1,cl,mid,l,r,k);
    if(r>mid)ret+=query(c<<1|1,mid+1,cr,l,r,k);
    return ret;
}
ll n,p;cin>>n>>p;
rep(i,0,n)cin>>a[i];
build(1,0,n-1);
while(p--){
    ll l,r,k;cin>>l>>r>>k;
    l--;r--;
    //k=r-l+1-k; for kth largest
    ll lo=0,hi=1e9;// guess an element
    while(lo<hi){
        ll mid=lo+hi+1>>1;
        //check the position of the element in the range
        if(query(1,0,n-1,l,r,mid)<=k)lo=mid;
        else hi=mid-1;
    }
    cout<<lo<<endl;
}
```

10.5 Lazy create+Lazy prop seg tree

```
const ll maxn=5e6+5;
ll lc[maxn],rc[maxn],tree[maxn],add[maxn];
ll tot=1,rt=1;
void fill(ll c,ll cl,ll cr,ll val){
    tree[c]+=val*(cr-cl+1);
    add[c]+=val;
}
void pushdown(ll c,ll cl, ll cr){
    if(lc[c]==-1)lc[c]=++tot;
    if(rc[c]==-1)rc[c]=++tot;
    ll mid=cl+cr>>1;
    fill(lc[c],cl,mid,add[c]);
    fill(rc[c],mid+1,cr,add[c]);
    add[c]=0;
}
void update(ll &c,ll cl, ll cr, ll l, ll r, ll val){
    if(c==-1)c=++tot;
    if(l<=cl and cr<=r){
        fill(c,cl,cr,val);
        return;
    }
    pushdown(c,cl,cr);
    ll mid=cl+cr>>1;
    if(l<=mid)update(lc[c],cl,mid,l,r,val);
    if(r>mid)update(rc[c],mid+1,cr,l,r,val);
    tree[c]=tree[lc[c]]+tree[rc[c]];
    return;
```

```
}
ll query(ll c,ll cl, ll cr, ll l, ll r){
    if(c==-1)return 0;
    if(l<=cl and cr<=r)return tree[c];
    pushdown(c,cl,cr);
    ll mid=cl+cr>>1;
    ll ans=0;
    if(l<=mid)ans+=query(lc[c],cl,mid,l,r);
    if(r>mid)ans+=query(rc[c],mid+1,cr,l,r);
    return ans;
}
// to init:
memset(lc,-1,sizeof lc);
memset(rc,-1,sizeof rc);
```

10.6 Everything segtree

```
typedef long long ll;
struct node {
    int s, e;
    ll mn, mx, sum;
    bool lset;
    ll add_val, set_val;
    node *l, *r;
    node (int _s, int _e, int A[] = NULL): s(_s), e(_e), mn(0), mx(0), sum(0), lset(0),
        add_val(0), set_val(0), l(NULL), r(NULL) {
        if (A == NULL) return;
        if (s == e) mn = mx = sum = A[s];
        else {
            l = new node(s, (s+e)>>1, A), r = new node((s+e+2)>>1, e, A);
            combine();
        }
    }
    void create_children() {
        if (s == e) return;
        if (l != NULL) return;
        int m = (s+e)>>1;
        l = new node(s, m);
        r = new node(m+1, e);
    }
    void self_set(ll v) {
        lset = 1;
        mn = mx = set_val = v;
        sum = v * (e-s+1);
        add_val = 0;
    }
    void self_add(ll v) {
        if (lset) { self_set(v + set_val); return; }
        mn += v, mx += v, add_val += v;
        sum += v*(e-s+1);
    }
    void lazy_propagate() {
        if (s == e) return;
        if (lset) {
            l->self_set(set_val), r->self_set(set_val);
            lset = set_val = 0;
        }
```

```
        if (add_val != 0) {
            l->self_add(add_val), r->self_add(add_val);
            add_val = 0;
        }
    }
    void combine() {
        if (l == NULL) return;
        sum = l->sum + r->sum;
        mn = min(l->mn, r->mn);
        mx = max(l->mx, r->mx);
    }
    void add(int x, int y, ll v) {
        if (s == x && e == y) { self_add(v); return; }
        int m = (s+e)>>1;
        create_children(); lazy_propagate();
        if (x <= m) l->add(x, min(y, m), v);
        if (y > m) r->add(max(x, m+1), y, v);
        combine();
    }
    void set(int x, int y, ll v) {
        if (s == x && e == y) { self_set(v); return; }
        int m = (s+e)>>1;
        create_children(); lazy_propagate();
        if (x <= m) l->set(x, min(y, m), v);
        if (y > m) r->set(max(x, m+1), y, v);
        combine();
    }
    ll range_sum(int x, int y) {
        if (s == x && e == y) return sum;
        if (l == NULL || lset) return (sum / (e-s+1)) * (y-x+1);
        int m = (s+e)>>1;
        lazy_propagate();
        if (y <= m) return l->range_sum(x, y);
        if (x > m) return r->range_sum(x, y);
        return l->range_sum(x, m) + r->range_sum(m+1, y);
    }
    ll range_min(int x, int y) {
        if (s == x && e == y) return mn;
        if (l == NULL || lset) return mn;
        int m = (s+e)>>1;
        lazy_propagate();
        if (y <= m) return l->range_min(x, y);
        if (x > m) return r->range_min(x, y);
        return min(l->range_min(x, m), r->range_min(m+1, y));
    }
    ll range_max(int x, int y) {
        if (s == x && e == y) return mx;
        if (l == NULL || lset) return mx;
        int m = (s+e)>>1;
        lazy_propagate();
        if (y <= m) return l->range_max(x, y);
        if (x > m) return r->range_max(x, y);
        return max(l->range_max(x, m), r->range_max(m+1, y));
    }
    ~node() {
        if (l != NULL) delete l;
        if (r != NULL) delete r;
    }
} *root;
```

```
root = new node(0, N-1, array); //creates a segment tree with elements 0 to N - 1. The array
    parameter is optional.
root = new node(0, 1000000000); //this tree supports lazy node creation and propagation too,
    declare as much as you like :)


root->add(0, 5000, 3); //add 3 to range [0, 5000]
root->add(3000, 9000, -2); //minus 2 to range [3000, 9000]
root->set(7000, 10000, 5); //set range [7000, 10000] to 5

/* at this point, 0 to 2999 is 3, 3000 to 5000 is 1, 5001 to 6999 is -2, 7000 to 10000 is 5 */
root->range_max(0, 10000); //returns 5
root->range_min(0, 10000); //returns -2
root->range_sum(0, 10000); //returns 22008
```

10.7 median heap

```
ll n;
multiset<ll>lo,hi;
void maintain(){
    while(SZ(lo)>SZ(hi)+1)hi.insert(*(--lo.end())),lo.erase(--lo.end());
    while(SZ(hi)>SZ(lo))lo.insert(*hi.begin()),hi.erase(hi.begin());
}
void add(ll x){
    if(hi.empty() or *hi.begin()>x)lo.insert(x);
    else hi.insert(x);
    maintain();
}
void rem(ll x){//remove element
    if(lo.find(x)!=lo.end())lo.erase(lo.find(x));
    else if(hi.find(x)!=hi.end())hi.erase(hi.find(x));
    maintain();
}
ll med(){
    if((SZ(lo)+SZ(hi))%2==1)return *(--lo.end());
    else return (*(--lo.end())+*hi.begin())/2;
}
```

10.8 cakerun

```
unordered_map<ll,set<pll>>mp;
set<pair<pll,ll>>s;

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    //freopen("i.txt","r",stdin);
    ll n;cin>>n;
    rep(i,1,n+1){
        ll x;cin>>x;
        mp[x].insert(MP(i,i));
        s.insert(MP(MP(i,i),x));
    }
```

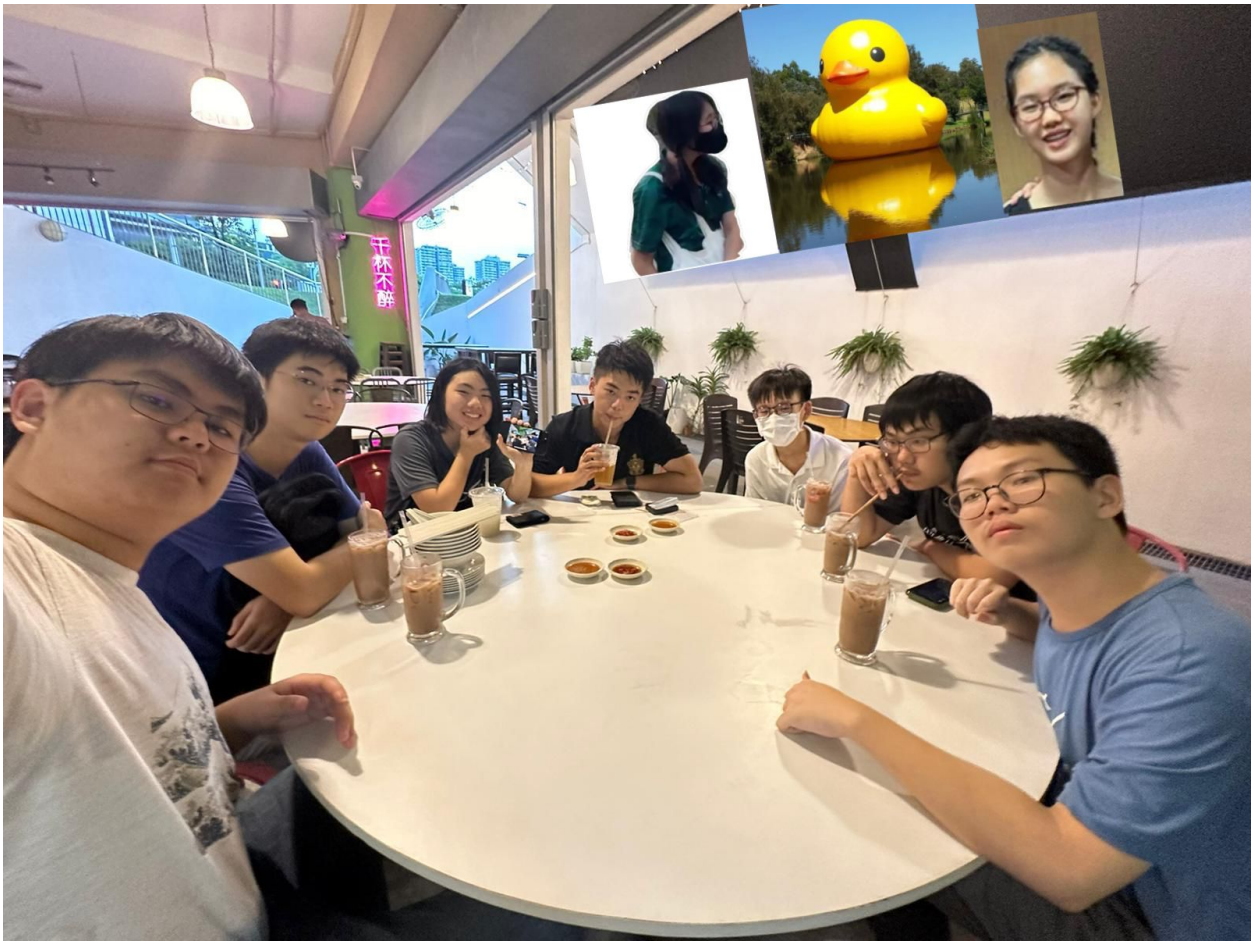
```
    ll q;cin>>q;
    while(q--){
        ll op;cin>>op;
        ll l,r,c;cin>>l>>r>>c;
        if(op==1){
            auto it=mp[c].lower_bound(MP(l,0ll));
            ll ans=inf;
            if(it!=mp[c].end() and it->fi<=r){
                ans=min(ans,it->fi);
            }
            if(it!=mp[c].begin() and (--it)->se>=l){
                ans=min(ans,l);
            }
            if(ans==inf)cout<<-1<<endl;
            else cout<<ans<<endl;
        }else{
            auto it=s.lower_bound(MP(MP(l,0ll),0ll));
            if(it->fi.fi!=l)it--;
            //find the first range containing l
            vector<pair<pll,ll>>ranges;
            while(it!=s.end() and it->fi.fi<=r){
                ranges.pb(*it);
                it++;
            }
            for(auto x:ranges){
                s.erase(x);
                mp[x.se].erase(x.fi);
            }
            if(!ranges.empty()){
                auto st=ranges[0];
                auto ed=ranges[SZ(ranges)-1];
                if(st.fi.fi<=l-1){
                    mp[st.se].insert(MP(st.fi.fi,l-1));
                    s.insert(MP(MP(st.fi.fi,l-1),st.se));
                }
                if(r+1<=ed.fi.se){
                    mp[ed.se].insert(MP(r+1,ed.fi.se));
                    s.insert(MP(MP(r+1,ed.fi.se),ed.se));
                }
            }
            mp[c].insert(MP(l,r));
            s.insert(MP(MP(l,r),c));
        }
    }
    return 0;
}
```

10.9 RY segtree

```
const int N = 1e5; // limit for array size
int n; // array size
int t[2 * N];

void build() { // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}
```

```
void modify(int p, int value) { // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}

int query(int l, int r) { // sum on interval [l, r)
  int 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;
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d", t + n + i);
  build();
  modify(0, 1);
  printf("%d\n", query(3, 11));
  return 0;
}
```

11 Motivation


千杯不醉