Week 23: Advanced Data Structures

Topics: - Segment Trees (Range Sum, Range Minimum/Maximum, Lazy Propagation) - Fenwick Tree (Binary Indexed Tree) - Sparse Table (RMQ for immutable arrays) - Treap, Splay Tree, and AVL Trees - Disjoint Set Union (DSU) / Union-Find with path compression and union by size/rank - Persistent Data Structures

Weekly Tips: - Segment trees allow efficient range queries and updates, O(log n) per operation. - Lazy propagation is crucial for range updates. - Fenwick Trees are simpler alternatives for prefix sum queries. - Sparse tables are perfect for static range minimum/maximum queries, O(1) per query. - Treaps and Splay Trees balance BSTs probabilistically or via rotations. - DSU is essential for connected components and Kruskal's algorithm. - Persistent structures allow access to previous versions efficiently.

Problem 1: Range Sum Query with Segment Tree Link: CSES Range Queries **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
vector<long long> seg;
int n;
void build(vector<long long>& a,int v,int tl,int tr){
    if(tl==tr) seg[v]=a[tl];
    else{
        int tm=(tl+tr)/2;
        build(a,2*v,tl,tm); build(a,2*v+1,tm+1,tr);
        seg[v]=seg[2*v]+seg[2*v+1];
    }
}
long long sum(int v,int tl,int tr,int l,int r){
    if(l>r) return 0;
    if(l==tl && r==tr) return seg[v];
    int tm=(tl+tr)/2;
    return sum(2*v,tl,tm,l,min(r,tm))+sum(2*v+1,tm+1,tr,max(l,tm+1),r);
}
void update(int v,int tl,int tr,int pos,long long new_val){
    if(tl==tr) seg[v]=new_val;
    else{
        int tm=(tl+tr)/2;
        if(pos<=tm) update(2*v,tl,tm,pos,new_val);
        else update(2*v+1,tm+1,tr,pos,new_val);
        seg[v]=seg[2*v]+seg[2*v+1];
    }
}
int main(){
    int q; cin>>n>>q;
    vector<long long> a(n);
```

```
    for(int i=0;i<n;i++) cin>>a[i];
    seg.assign(4*n,0);
    build(a,1,0,n-1);
    while(q--){
        int t; cin>>t;
        if(t==1){ int i; long long x; cin>>i>>x; update(1,0,n-1,i-1,x); }
        else{ int l,r; cin>>l>>r; cout<<sum(1,0,n-1,l-1,r-1)<<endl; }
    }
}
```

Explanation Comments: - Build segment tree recursively for sum of ranges. - Query and update are O(log n). - Modify recursive calls for different operations (min, max, gcd).

Problem 2: DSU / Union-Find Link: CSES Road Construction **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
vector<int> parent,size;
int find(int u){ return parent[u]==u?u:parent[u]=find(parent[u]); }
void unite(int u,int v){
    u=find(u); v=find(v);
    if(u==v) return;
    if(size[u]<size[v]) swap(u,v);
    parent[v]=u; size[u]+=size[v];
}
int main(){
    int n,m; cin>>n>>m;
    parent.resize(n+1); size.resize(n+1,1);
    for(int i=1;i<=n;i++) parent[i]=i;
    for(int i=0;i<m;i++){
        int u,v; cin>>u>>v;
        unite(u,v);
        cout<<n-i-1<<endl;
    }
}
```

Explanation Comments: - Path compression optimizes find operation. - Union by size keeps tree shallow for efficiency. - Essential for connected components, Kruskal's MST, and dynamic connectivity.

End of Week 23 - Master advanced data structures for efficient query and update operations. - Practice segment trees, DSU, sparse tables, and persistent structures. - These are vital for many ACM-ICPC and online judge problems.