Week 40: Advanced Data Structures – Persistent & Dynamic Trees

Topics: - Persistent Segment Tree (versioned queries) - Link-Cut Trees for dynamic connectivity - Euler Tour Trees for maintaining dynamic forests - Dynamic Tree Queries (path sum, subtree queries) - Applications: Offline Queries, Dynamic Graphs, Version Control

Weekly Tips: - Persistent segment trees store previous versions while supporting updates. - Link-Cut Trees allow changing parent-child relationships dynamically. - Euler Tour Trees maintain forest properties with dynamic splits/joins. - Useful in dynamic graph problems with updates and queries. - Focus on memory optimization with persistence.

Problem 1: Persistent Segment Tree Link: CSES Hotel Queries **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Node{ int val; Node* left; Node* right; Node(int
v):val(v),left(NULL),right(NULL){} };
vector<Node*> version;

Node* build(vector<int>&a,int l,int r){
    if(l==r) return new Node(a[l]);
    int m=(l+r)/2;
    Node* node=new Node(0);
    node->left=build(a,l,m);
    node->right=build(a,m+1,r);
    node->val=max(node->left->val,node->right->val);
    return node;
}

Node* update(Node* prev,int l,int r,int idx,int val){
    if(l==r) return new Node(val);
    int m=(l+r)/2;
    Node* node=new Node(0);
    if(idx<=m){
        node->left=update(prev->left,l,m,idx,val);
        node->right=prev->right;
    } else {
        node->left=prev->left;
        node->right=update(prev->right,m+1,r,idx,val);
    }
    node->val=max(node->left->val,node->right->val);
    return node;
}
```

```cpp
int query(Node* node,int l,int r,int ql,int qr){
    if(r<ql||l>qr) return -1e9;
    if(ql<=l&&r<=qr) return node->val;
    int m=(l+r)/2;
    return max(query(node->left,l,m,ql,qr), query(node->right,m+1,r,ql,qr));
}

int main(){
    int n,q; cin>>n>>q; vector<int>a(n);
    for(int i=0;i<n;i++) cin>>a[i];
    Node* root=build(a,0,n-1);
    version.push_back(root);
    while(q--){
        int t; cin>>t;
        if(t==1){ // update new version
            int idx,val; cin>>idx>>val; idx--;
            version.push_back(update(version.back(),0,n-1,idx,val));
        } else { // query old version
            int ver,l,r; cin>>ver>>l>>r; l--; r--;
            cout<<query(version[ver],0,n-1,l,r)<<"\n";
        }
    }
}
```

Explanation Comments: - Each update creates a new version, old versions remain accessible. - Query any version efficiently. - Useful for rollback queries or offline tasks.

Problem 2: Link-Cut Tree (Conceptual Overview) - Dynamic tree structure that supports: - `link(u,v)`: connect two nodes. - `cut(u,v)`: remove edge. - `findRoot(u)`: find root of tree containing u. - Path queries using splay trees.

Applications: - Dynamic connectivity in graphs. - Maintaining MST under edge updates. - Dynamic tree path queries.

End of Week 40 - Learn persistent and dynamic trees. - Practice persistent segment trees for rollback/version queries. - Understand link-cut trees for dynamic connectivity problems in ACM-ICPC.