可持久化并查集

Persistent Disjoint Set Union

Idea：所谓并查集，在实现上无非就是两个数组 `fa[]` 和 `sz[]`（按秩合并），只需要将这两个数组可持久化即可。把 y 合并到 x 上时，我们需要在某个历史版本上更改 `fa[findfa(y)]` 为 `findfa(x)`，并将 `sz[findfa(x)]` 加上 `sz[findfa(y)]`。而所谓可持久化数组，仍旧是可持久化线段树来实现。

Complexity：$O(n\lg^2n)$

Code：

```cpp
#include<algorithm>
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 200005;
int n, m;

struct segTree{
    int lson, rson;
    int fa, sz;
}tr[N * 30];
int cnt, root[N];
void build(int id, int l, int r){
    if(l == r){
        tr[id].fa = l, tr[id].sz = 1;
        return;
    }
    tr[id].lson = ++cnt, tr[id].rson = ++cnt;
    int mid = (l + r) >> 1;
    build(tr[id].lson, l, mid);
    build(tr[id].rson, mid+1, r);
}
int queryID(int id, int l, int r, int pos){ // query the ID of pos'th leaf in tr[id]
    if(l == r)    return id;
    int mid = (l + r) >> 1;
    if(pos <= mid)  return queryID(tr[id].lson, l, mid, pos);
    else    return queryID(tr[id].rson, mid+1, r, pos);
}
void modify(int cur, int pre, int l, int r, int pos, int fa){
    tr[cur] = tr[pre];
    if(l == r){ tr[cur].fa = fa; return; }
    int mid = (l + r) >> 1;
    if(pos <= mid){
        tr[cur].lson = ++cnt;
        modify(tr[cur].lson, tr[pre].lson, l, mid, pos, fa);
    }
    else{
        tr[cur].rson = ++cnt;
        modify(tr[cur].rson, tr[pre].rson, mid+1, r, pos, fa);
    }
}
void add(int id, int l, int r, int pos, int val){
    if(l == r){ tr[id].sz += val; return; }
    int mid = (l + r) >> 1;
    if(pos <= mid)  add(tr[id].lson, l, mid, pos, val);
    else    add(tr[id].rson, mid+1, r, pos, val);
}

int findfa(int cur, int x){
    int xid = queryID(cur, 1, n, x); // x's id in tr[cur]
    return tr[xid].fa == x ? x : findfa(cur, tr[xid].fa);
}
void unionn(int cur, int pre, int x, int y){
    tr[cur] = tr[pre]; // ATT!
    x = findfa(pre, x), y = findfa(pre, y);
    if(x == y) return;
    int xid = queryID(pre, 1, n, x), yid = queryID(pre, 1, n, y);
    if(tr[xid].sz < tr[yid].sz) swap(x, y), swap(xid, yid);
    modify(cur, pre, 1, n, y, x);
    add(cur, 1, n, x, tr[yid].sz);
}

int main(){
    scanf("%d%d", &n, &m);
    build(root[0] = 0, 1, n);
    for(int i = 1; i <= m; i++){
        int opt, a, b;
        scanf("%d", &opt);
        if(opt == 1){
            scanf("%d%d", &a, &b);
            root[i] = ++cnt;
            unionn(root[i], root[i-1], a, b);
```

```c
        }
        else if(opt == 2){
            scanf("%d", &a);
            root[i] = root[a];
        }
        else{
            scanf("%d%d", &a, &b);
            root[i] = root[i-1];
            puts(findfa(root[i], a) == findfa(root[i], b) ? "1" : "0");
        }
    }
    return 0;
}
```