B 前进者

出题人:梅子酒

考察点:并查集,STL容器,线段树

题意

图 G 有 n 个点，权值为 v_i，并逐步给出权值 w_i 递增的 m 条道路。每次给出新道路后求出从任意一点出发，遍历该连通块后的最大值。每次新增道路后，第一次通过城市可以获得价值 v_i，第一次经过道路需要付出价值 w_i。

思路

考虑到题目所说的无论从任意点出发都要遍历一整个连通块，并且点权和边权都只计算一次，**所以可以知道一个连通块的价值和遍历该连通块的路径无关，只和连通块本身的点权和，边权和有关。**可以用带权并查集来维护一个连通块的点权和。逐步给出边并且边权是递增的，这恰好就是一个最小生成树的过程， 同样可以用并查集维护，将边权和视为负数和点权和一起维护。

这样我们可以用并查集近似 $O(n)$ 的时间复杂度维护任意连通块的点权和，边权和。考虑如何维护其中的最大值，有很多方法。其中标程给出的方法是线段树，维护的就是上述的点权和，边权和，只需要单点修改，最终递归回根节点取最大值后，根节点的价值就是答案。还有很多其他方法，很多STL容器都可以维护，例如`priority_queue`，`multiset` 等，可以自行了解。其中无论是线段树还是STL容器维护该数据的时间复杂度都是 $O(nlogn)$，线段树的常数更加优秀一些，总时间复杂度为 $O(n+nlogn)$。注意答案可能是负数，所以极值的设置要合适。

代码

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ls p << 1
#define rs p << 1 | 1
const int N = 1e5 + 10;
const ll INF = 1e18; // 极大值
int n, m, p[N];
ll v[N], c[N]; // v_i 点权和，c_i 边权和
struct seg{
    int l, r;
    ll val; // 一个连通块的点权和 - 边权和
}tr[N * 4];
void pushup(int p){
    tr[p].val = max(tr[ls].val, tr[rs].val); // 将答案向上更新
}
void build(int p, int l, int r){
    tr[p] = {l, r, 0};
    if(l == r){
        tr[p].val = v[l] - c[l]; // 维护
        return ;
    }
    int mid = (l + r) >> 1;
```

```
    build(ls, l, mid); build(rs, mid + 1, r);
    pushup(p);
}
void update(int p, int loc, ll k){ // 单点修改
    if(tr[p].l == tr[p].r){
        tr[p].val = k;
        return ;
    }
    int mid = tr[ls].r;
    if(loc <= mid) update(ls, loc, k);
    else update(rs, loc, k);
    pushup(p);
}

int find(int x){ // 找到祖先
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
}
void merge(int x, int y, int w){
    int fx = find(x), fy = find(y);
    if(fx != fy){ // 如果之前不连通，就需要合并
        p[fx] = fy;
        v[fy] += v[fx]; // 点权和相加
        c[fy] += c[fx] + w; // 边权和相加
        v[fx] = c[fx] = 0; // 将之前的置为 0
        update(1, fy, v[fy] - c[fy]); // 更新
        update(1, fx, -INF); // 将原来连通块置为无穷小
    }
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);

    cin >> n >> m;
    for(int i = 1; i <= n; i ++) cin >> v[i];

    for(int i = 1; i <= n; i ++) p[i] = i;

    build(1, 1, n);
    for(int i = 1; i <= m; i ++){
        int u, v, w;
        cin >> u >> v >> w;
        merge(u, v, w);
        cout << tr[1].val << "\n"; // 直接输出根节点的答案
    }
    return 0;
}
```