1.16NOIP套题3

T1黑匣子

```cpp
#include<bits/stdc++.h>
using namespace std;
#include<stdio.h>
priority_queue<int ,vector<int>,greater<int>> q1;//graeter 对应着上面的小顶堆
priority_queue<int ,vector<int>,less<int>> q2;//less对应着下面的大顶堆
//大顶堆是正三角，数值小，对应less
//小顶堆是倒三角， 数值大，对应greater
int a[200005];
int main(){
    int i=0;
    int m,n;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
    }
    int j=1;
    for(int i=1;i<=m;i++){//既代表get的次数，也代表get要输出的从小到大第i个值
        int t;
        scanf("%d",&t);
        for(int k=j;k<=t;k++){
            //push 插入并排序
            q2.push(a[k]);//输入大根堆，按从下到上依次增大的顺序排
            if(q2.size()==i) q1.push(q2.top()),q2.pop();
            //如果大根堆排满了，将堆顶即最大的数值给小顶堆
        }
        j=t+1;//避免没必要的运算，加快速度
        printf("%d\n",q1.top());//要求输出从小到大第i个
        //大顶堆一共有i-1个数值，故q2.top（）对应着从小到大第n-1个
        //q1.top（）对应着从小到大第n个
        q2.push(q1.top());
        q1.pop(); //将小根堆堆顶数值即最小数给大顶堆，这步是为了应对相邻两次t相同的情况
        //相邻两次t相同时，上面以k为自变量的for循环不会进行
        //这部保证了q1.top（）为从小到大第i个
    }
    return 0;
}
```

T2 联合权值

【noip2014】联合权值首先枚举每一个点，找他周围的最大与第二大的点，相乘的联合权值则为以这个点为中心的最大值，最后把所有点的最大权值扫一遍，然后就得到ans2.至于权值之和，我们举一个例子：　　i 这个点连了 1 2 3 三个点 ，那么权值之和即：w1*w2*+w1*w3+w2*w3,技巧来了，根据乘法分配律，原式=w1*0+w2*w1+w3*(w1+w2)

根据这个，可以设一个数组存括号内的累和，然后与当前相乘，加入ans1；

T3 最优贸易

首先本题题意为：从节点1出发到一个价格最低的节点购买水晶球，再去一个价格最高的节点卖出水晶球，前提是买卖两个节点使可以到达的。因此可以考虑动态规划的思路，我们假设买卖由节点k分隔，即在节点1-k购买，在节点k-n卖出，当然也可以不买不卖，那么有：

从 1 走到 i 的过程中，买入水晶球的最低价格 dmin[i]；
从 i 走到 n 的过程中，卖出水晶球的最高价格 dmax[i]；
那么显然最大差值为max(dmax[i] - dmin[i])，接下来考虑算法实现。求dmax[i] 和dmin[i]的过程显然使用最短路算法，由于这次求的最值是节点的权值最值，数值不累计，因此不能使用Dijkstra算法，于是采用spfa求最值，朴素的实现应该是先求节点1到各个节点最小值，然后分别求节点i到节点n的最大值，这样就等于执行了n次spfa，这样复杂度显然太高，于是考虑反向建图，即将图路径取反，然后求节点n到各个节点的最大距离，这样节点n到节点i的最大距离，就等于原图中节点i到节点n的最大距离，这样只需要执行两次spfa，节省了很多时间，可参考下边代码实现，代码参考yxc

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <queue>

using namespace std;

const int N = 100010, M = 2000010;

int n, m;
int price[N];
int h[N], rh[N], e[M], ne[M], idx;
int dmin[N], dmax[N];
bool st[N];

void add(int *h, int a, int b)
{
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

//寻找最小距离，flag来区分正反搜索
void spfa(int *d, int start, int *h, bool flag)
{
    queue<int> q;
    memset(st, 0, sizeof st);

    if (flag) memset(d, 0x3f, sizeof dmin);

    q.push(start);
    st[start] = true;
    d[start] = price[start];

    while (q.size())
    {
        int t = q.front();
```

```
        q.pop();
        st[t] = false;

        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            //正搜求最小值，反搜求最大值
            if (flag && d[j] > min(d[t], price[j]) || !flag && d[j] < max(d[t],
price[j]))
            {
                if (flag) d[j] = min(d[t], price[j]);
                else d[j] = max(d[t], price[j]);

                if (!st[j])
                {
                    st[j] = true;
                    q.push(j);
                }
            }
        }
    }

}

int main()
{
    scanf("%d%d", &n, &m);

    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);

    for (int i = 1; i <= n; i ++ ) scanf("%d", &price[i]);
    while (m -- )
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        //add(rh, b, a)等是反向建图
        add(h, a, b), add(rh, b, a);
        if (c == 2) add(h, b, a), add(rh, a, b);
    }

    //分别进行一次spfa
    spfa(dmin, 1, h, true);
    spfa(dmax, n, rh, false);

    //枚举求最大差值
    int res = 0;
    for (int i = 1; i <= n; i ++ ) res = max(res, dmax[i] - dmin[i]);

    printf("%d\n", res);

    return 0;

}
```

T4 friend

并查集