

2020牛客国庆集训派对

Day 1

[比赛链接](#)

I Saba1000kg

题意

给定一张图 q 个询问，每次询问只考虑图中 s_i 个点构成的连通块数。数据保证 $\sum s_i \leq 10^5$

题解

考虑分块。当 $s_i \leq k$ 时暴力 $O(s_i^2)$ 枚举所有点对同时维护并查集 $s_i \geq k$ 时暴力 $O(m)$ 枚举所有边同时维护并查集。

从均摊复杂度考虑，消耗每点 s_i 的复杂度为 $O(\max(\frac{s_i^2}{s_i}(s_i \leq k), \frac{m}{s_i}(s_i \geq k)) \log s_i) = O(\max(k, \frac{mk}{\log k}))$

取 $k = O(\sqrt{m})$ 时总时间复杂度为 $O(\sum s_i \sqrt{m} \log m)$

```
const int MAXN=1e5+5;
set<int>g[MAXN];
struct Edge{
    int u,v;
}edge[MAXN];
int p[MAXN],b[MAXN];
bool vis[MAXN];
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
void Merge(int x,int y){
    int xx=Find(x),yy=Find(y);
    if(xx!=yy)
        p[xx]=yy;
}
int main()
{
    int n=read_int(),m=read_int(),q=read_int(),limt=sqrt(m+10);
    _for(i,0,m){
        edge[i].u=read_int(),edge[i].v=read_int();
        g[edge[i].u].insert(edge[i].v);
        g[edge[i].v].insert(edge[i].u);
    }
    while(q--){
```

```
int sz=read_int(),ans=0;
_for(i,0,sz){
    b[i]=read_int();
    vis[b[i]]=true;
    p[b[i]]=b[i];
}
if(sz<limt){
    _for(i,0,sz)
        _for(j,i,sz){
            if(g[b[i]].count(b[j]))
                Merge(b[i],b[j]);
        }
}
else{
    _for(i,0,m){
        if(vis[edge[i].u]&&vis[edge[i].v])
            Merge(edge[i].u,edge[i].v);
    }
    _for(i,0,sz){
        ans+=(Find(b[i])==b[i]);
        vis[b[i]]=false;
    }
    enter(ans);
}
return 0;
}
```

Day 4

[比赛链接](#)

B Arithmetic Progressions

题意

给定一个不可重集，求最大子集满足子集中数构成等差数列。

题解

先将不可重集排序得到序列 A 然后令 $\text{dp}(i,j)$ 表示等差数列最后一项为 a_j 且倒数第二项为 a_i 时的等差数列的长度。

考虑枚举 i 同时双指针 k 和 j 查找满足 $a_k + a_j = a_i$ 的数，有状态转移 $\text{dp}(i,j) = \text{dp}(k,i) + 1$ 时间复杂度 $O(n^2)$

```

const int MAXN=5e3+5;
int a[MAXN],dp[MAXN][MAXN];
int main()
{
    int n=read_int();
    _for(i,0,n)a[i]=read_int();
    sort(a,a+n);
    _for(i,0,n)_for(j,i+1,n)dp[i][j]=2;
    int ans=2;
    _for(i,0,n){
        int pos1=i-1,pos2=i+1;
        while(pos1>=0&&pos2<n){
            if(a[pos1]+a[pos2]<(a[i]<<1))pos2++;
            else if(a[pos1]+a[pos2]>(a[i]<<1))pos1--;
            else{
                dp[i][pos2]=max(dp[i][pos2],dp[pos1][i]+1);
                ans=max(ans,dp[i][pos2]);
                pos1--;pos2++;
            }
        }
    }
    enter(ans);
    return 0;
}

```

H Colorful Tree

题意

给定一棵树，树上每点有一种颜色。接下来两种操作：

1. 询问包含所有颜色为 c 的结点的最小子图的边数
2. 将某点的颜色修改为 c

题解

将无根树转为以 1 为根的有根树，同时处理得到每个点的 dfs 序。

不难发现，对颜色为 c 的结点的最小子图的边数等价于所有该颜色结点所有 dfs 序相邻结点距离加上 dfs 序最大最小两点距离和除以 2 。

考虑 set 维护即可，时间复杂度 $O((n+q)\log n)$

```

const int MAXN=1e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];

```

```
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
namespace LCA{
    int d[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],node_id[MAXN],dfs_t;
    int h_son[MAXN],mson[MAXN],p[MAXN];
    void dfs_1(int u,int fa,int depth){
dfn[u]=++dfs_t;sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;node_id[dfs_t]=u;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
                continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u])
                h_son[u]=v,mson[u]=sz[v];
        }
    }
    void dfs_2(int u,int top){
        p[u]=top;
        if(mson[u])dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])
                continue;
            dfs_2(v,v);
        }
    }
    void init(int root){dfs_1(root,0,0);dfs_2(root,root);}
    int query_lca(int u,int v){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])swap(u,v);
            u=f[p[u]];
        }
        return d[u]<d[v]?u:v;
    }
    int query_dis(int u,int v){
        u=node_id[u],v=node_id[v];
        return d[u]+d[v]-2*d[query_lca(u,v)];
    }
};
typedef set<int>::iterator iter;
set<int>s[MAXN];
int c[MAXN];
LL ans[MAXN];
void del(int c,int u){
    iter it=s[c].find(u);
    int l=0,r=0;
```

```

    if(++it!=s[c].end())
        r=*it;
    if(--it!=s[c].begin())
        l=*(--it);
    if(l&&r)ans[c]+=LCA::query_dis(l,r);
    if(l)ans[c]-=LCA::query_dis(l,u);
    if(r)ans[c]-=LCA::query_dis(u,r);
    s[c].erase(u);
}

void add(int c,int u){
    s[c].insert(u);
    iter it=s[c].find(u);
    int l=0,r=0;
    if(++it!=s[c].end())
        r=*it;
    if(--it!=s[c].begin())
        l=*(--it);
    if(l&&r)ans[c]-=LCA::query_dis(l,r);
    if(l)ans[c]+=LCA::query_dis(l,u);
    if(r)ans[c]+=LCA::query_dis(u,r);
}

int main()
{
    int n=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    LCA::init(1);
    _rep(i,1,n){
        c[i]=read_int();
        add(c[i],LCA::dfn[i]);
    }
    int q=read_int();
    while(q--){
        char opt=get_char();
        if(opt=='Q'){
            int v=read_int();
            if(s[v].empty())
                enter(-1);
            else
                enter((ans[v]+LCA::query_dis(*s[v].begin(),*(--s[v].end())))/2);
        }
        else{
            int u=read_int(),v=read_int();
            del(c[u],LCA::dfn[u]);
            add(v,LCA::dfn[u]);
            c[u]=v;
        }
    }
}

```

```
return 0;  
}
```

Day 7

[比赛链接](#)

C Expect to wait

题意

给定 n 个事件，时间分两种：

1. t 时刻有 k 人借车
2. t 时刻新加入 k 辆车

接下来 q 个询问，每次询问当初始时有 b 辆车时所有人的等待时间和。

题解

先假设初始时没有车，考虑根据初始事件构造“时间-等待人数”图，则答案恰好为该图形围成的面积。

考虑初始时有 b 辆车的情况，易知这等价于将图形向下移动 b 个单位，同时删去小于 0 的部分。

不难发现这同时等价于直线 $y=b$ 与原图形上方曲线围成面积。考虑将询问根据 b 排序，利用扫描线法维护答案，时间复杂度 $O(n\log n)$

需要注意无解情况的特判。另外发现也可以吉司机线段树区间置 $\max$ 再询问 $\sum$ 无脑维护。

```
const int MAXN=1e5+5;  
struct Seg{  
    int len,v;  
    bool operator < (const Seg &b)const{  
        return v>b.v;  
    }  
}seg[MAXN];  
struct Query{  
    int v,idx;  
    bool operator < (const Query &b)const{  
        return v>b.v;  
    }  
}que[MAXN];  
LL ans[MAXN];  
int tim[MAXN],k[MAXN];  
int main()  
{
```

```
int n=read_int(),q=read_int();
_rep(i,1,n){
    char c=get_char();
    tim[i]=read_int(),k[i]=read_int();
    if(c=='+')k[i]=-k[i];
}
int st=0;
_for(i,1,n){
    seg[i].len=tim[i+1]-tim[i];
    seg[i].v=st+=k[i];
}
st+=k[n];
st=max(0,st);
sort(seg+1,seg+n);
_for(i,0,q){
    que[i].v=read_int();
    que[i].idx=i;
}
sort(que,que+q);
int pos=1,slen=0;
LL sum=0;
_for(i,0,q){
    if(que[i].v<st){
        ans[que[i].idx]=-1;
        continue;
    }
    if(i)sum+=1LL*(que[i-1].v-que[i].v)*slen;
    while(pos<n&&que[i].v<seg[pos].v){
        slen+=seg[pos].len;
        sum+=1LL*(seg[pos].v-que[i].v)*seg[pos].len;
        pos++;
    }
    ans[que[i].idx]=sum;
}
_for(i,0,q){
    if(~ans[i])
        enter(ans[i]);
    else
        puts("INFINITY");
}
return 0;
}
```

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:contest:2020%E7%89%9B%E5%AE%A2%E5%9B%BD%E5%BA%86%E9%9B%86%E8%AE%AD%E6%B4%BE%E5%AF%B9

Last update: 2020/10/07 21:26