比赛链接

补题情况

题目	蒋贤蒙	王赵安	王智彪
B	0	0	0
C	2	0	1
F	2	0	1
G	0	0	0
H	0	0	0
I	0	0	0
J	2	0	0

题解

C. Fuzzy Graph

题意

给定一个连通图，要求将图上每个点染成 RGB 三种颜色。对应边 (u,v)如果 u,v 同色，则该边染成 u,v 的颜色，否则该边染成黑色。

要求构造一种染色方案，使得删除图中所有非黑边仍然可以使图连通，同时满足如下两个条件之一：

1. RGB 三种颜色的点一样多，数据保证 $3\mid n$
2. X 是点集中出现次数最多的颜色(允许有其他颜色出现次数和他相同)，且不存在边的颜色为 X

题解

首先跑一遍 dfs 树，然后用 RG 两种颜色进行二染色，这样所有树边都是黑色边，图保证连通。

接下来，设 c_0 表示 R 颜色在点集中出现的次数，设 c_1 表示 G 颜色在点集中出现的次数。

若 $c_0\le \frac n3$ 或者 $c_1\le \frac n3$则考虑构造第二个条件：将所有叶子结点染成颜色 B

事实上，由于叶子结点在 dfs 树上只有返祖边，所以所有叶子节点之间一定没有连边，即不存在颜色为 B 的边。

另一方面，不妨设 $c_0\le \frac n3$则所有颜色为 G 的非叶子结点至少是一个颜色为 R 的结点的父亲，所以所有颜色为 G 的非叶子结点个数不超过 $\frac n3$

由于颜色为 R 的非叶子结点个数不超过颜色为 R 的结点总个数 c_0所有颜色为 R 的非叶子结点个数也不超过 $\frac n3$

所以叶子结点总个数一定不小于 $\frac n3$且将叶子结点染成 B 后 B 出现次数一定最多。

若 $c_0\ge \frac n3$ 且 $c_1\ge \frac n3$则考虑构造第一个条件：从深到浅贪心染色，如果当前结点颜色出现个数大于 $\frac n3$ 且相邻结点颜色不为 B则染成 B

事实上，设 $c_0=\frac n3+d_0,c_1=\frac n3+d_1$根据上述构造，易知 G 被染色成 B 的个数不超过 d_1

而由于是从深到浅染色，所以每个从 G 被染色成 B 的结点只会 ban 他的父结点，不考虑 ban 儿子结点。

所有 R 结点的可选染色位置最多被 ban d_1 个位置，于是要从余下 $\frac n3+d_0-d_1$ 个位置中选出 d_0 个位置，这一定是可行的。

同理，所有 G 结点一定也可以选出 d_1 个位置染成 B

总时间复杂度 $O(n+m)$

```
const int MAXN=3e5+5,MAXM=5e5+5;
const char s[]="RGB";
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int ans[MAXN],fa[MAXN],d[MAXN],cnt[2];
bool fib[MAXN];
vector<int> node[MAXN],leaf;
void dfs(int u,int f){
    ans[u]=ans[f]^1;
    cnt[ans[u]]++;
    d[u]=d[f]+1;
    fa[u]=f;
    node[d[u]].push_back(u);
    bool flag=false;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f||d[v])continue;
        flag=true;
        dfs(v,u);
    }
    if(!flag)
    leaf.push_back(u);
}
void solve(){
    int n=read_int(),m=read_int();
    _rep(i,1,n){
        head[i]=0;
```

```
        d[i]=0;
        fib[i]=false;
        node[i].clear();
    }
    cnt[0]=cnt[1]=0;
    leaf.clear();
    edge_cnt=0;
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    dfs(1,0);
    if(cnt[0]<=n/3||cnt[1]<=n/3){
        for(int v:leaf)
        ans[v]=2;
    }
    else{
        for(int i=n-1;i;i--){
            for(int v:node[i]){
                if(fib[v]||cnt[ans[v]]==n/3)continue;
                cnt[ans[v]]--;
                ans[v]=2;
                fib[fa[v]]=true;
            }
        }
    }
    _rep(i,1,n)
    putchar(s[ans[i]]);
    putchar('\n');
}
int main(){
    int T=read_int();
    while(T--)
    solve();
    return 0;
}
```

J. Tree

题意

给定一棵树和两个人的初始位置 s,t，两个人在树上轮流移动，两个人走过的点都不能再走且两个人不能在同一个点。

每个人的分数为自己无法移动时走过的点数，两个人都希望最大化自己和另一个人的分数差，问最终先手的分数减去后手分数的值。

题解

首先以先手为根建树，树形 dp 求出每个点 u 不经过 s,t 链上的点能到达的最远距离 $a(u)$。设链上的点为 $v_1,v_2\cdots v_k$，其中 $v_1=s,v_k=t$。

定义 $\text{solve}(l,r,0/1)$ 表示第一个人位于 v_l 第二个人位于 v_r，$0/1$ 表示轮到先手/后手行动的答案。

对 $\text{solve}(l,r,0)$，先手如果选择进入子树，此时答案为 $a(v_l)+l-1-\max_{l\lt x\le r}(a(v_x)+k-x)$。

如果选择不进入子树，则答案为 $\text{solve}(l+1,r,1)$，所以两种情况取 $\max$ 即可，后手情况类似。

上述转移可以使用 ST 表，时间复杂度 $O(n\log n)$。

```
const int MAXN=5e5+5,MAXM=20;
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;
}
int p[MAXN];
multiset<int> dp[MAXN];
void dfs(int u,int fa){
    dp[u].insert(1);
    p[u]=fa;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs(v,u);
        dp[u].insert(*dp[v].rbegin()+1);
    }
}
struct ST{
    int d[MAXN][MAXM],lg2[MAXN];
    void build(int *a,int n){
        lg2[1]=0;
        _rep(i,2,n)
        lg2[i]=lg2[i>>1]+1;
        _for(i,0,n)
        d[i][0]=a[i];
        for(int j=1;(1<<j)<=n;j++){
            _for(i,0,n+1-(1<<j))
            d[i][j]=max(d[i][j-1],d[i+(1<<(j-1))][j-1]);
        }
    }
    int query(int lef,int rig){
        int len=rig-lef+1;
```

```
        return max(d[lef][lg2[len]],d[rig-(1<<lg2[len])+1][lg2[len]]);
    }
}st1,st2;
int mv1[MAXN],mv2[MAXN];
int solve(int l,int r,int pos){
    if(l+1==r)
    return mv1[l]-mv2[r];
    if(pos==0)
    return max(mv1[l]-st2.query(l+1,r),solve(l+1,r,!pos));
    else
    return min(st1.query(l,r-1)-mv2[r],solve(l,r-1,!pos));
}
int main(){
    int n=read_int(),s=read_int(),t=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    dfs(s,0);
    vector<int> vec;
    int pos=t;
    while(pos){
        vec.push_back(pos);
        pos=p[pos];
    }
    reverse(vec.begin(),vec.end());
    _for(i,0,vec.size()){
        int v=vec[i];
        if(i!=vec.size()-1){
            int v2=vec[i+1];
            dp[v].erase(dp[v].find(*dp[v2].rbegin()+1));
        }
    }
    _for(i,0,vec.size()){
        mv1[i]=*dp[vec[i]].rbegin()+i;
        mv2[i]=*dp[vec[i]].rbegin()+vec.size()-i-1;
    }
    st1.build(mv1,vec.size());
    st2.build(mv2,vec.size());
    enter(solve(0,vec.size()-1,0));
    return 0;
}
```

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

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%BB%84%E9%98%9F%E8%AE%AD%E7%BB%83%E6%AF%94%E8%B5%9B%E8%AE%B0%E5%BD%95:contest13&rev=1628850807

Last update: **2021/08/13 18:33**