B. Best Subgraph

题意

定义 $k-\text{degree}$ 子图为每个点在子图中度数至少为 k 的连通极大子图。

定义每个 G 的子图 S 的分数为 $M\times m(S)-N\times n(S)+B\times b(S)$。

其中 $m(S)$ 表示 S 的边数，$n(S)$ 表示 S 的点数，$b(S)$ 表示 $|\{(u,v)|(u,v)\in G,u\in S,v\not\in S\}|$。

求分数最大的 $k-\text{degree}$ 子图，并求出分数最大的 $k-\text{degree}$ 子图中 k 的最大值。

输入保证图连通。

题解

首先，定义 $V(k)$ 表示所有 $k-\text{degree}$ 子图的并集，易知 $V(k+1)\subseteq V(k)$。

构建分层图，其中第 k 层的点集为 $V(k)-V(k+1)$，同时对每个点 $u\in V(k)-V(k+1)$，$w(u)=k$。

然后考虑按层加点，动态维护当前每个 $k-\text{degree}$ 子图的答案。对于每个新加入的点 u，首先他本身产生贡献 $-N$。

对于 u 的所有连边 (u,v)，如果 $w(v)\gt w(u)$，则 (u,v) 会被加入 $m(S)$，同时从 $b(S)$ 删除，产生贡献 $M-B$。

如果 $w(v)=w(u)$，则 (u,v) 也会被加入 $m(S)$，但为了贡献被计算两次，于是每个点的贡献为 $\frac M2$。

如果 $w(v)\lt w(u)$，则 (u,v) 会被加入 $b(S)$，产生贡献 B。

于是上述过程可以将所有贡献都转化为每个点的点权。考虑加入点时并查集维护每个连通块的点权和。

每层点全部加完后查询每个连通块的点权和的最大值即为所有 $k-\text{degree}$ 子图的最大答案。

至于如果计算每个点的 $w(u)$，首先输入保证图连通，所有图 G 本身就是 $1-\text{degree}$ 子图。

考虑先删去所有度数为 1 的点，删点后可能会导致原来度数不为 1 的点度数不大于 1，继续删除，直到无点可删，得到 $2-\text{degree}$ 子图。

将删去的点的 $w(u)$ 全部设为 1，然后再求 $3-\text{degree}$ 子图不断重复上述过程，即可得到所有 $w(u)$。

时间复杂度 $O\left(\text{Na}(n+m)\right)$。

```
const int MAXN=1e6+5;
const LL inf=1e18;
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;
}
bool vis[MAXN];
vector<int> q[MAXN],vec[MAXN];
int deg[MAXN],w[MAXN],p[MAXN];
LL sum[MAXN];
int Find(int x){
    return x==p[x]?x:p[x]=Find(p[x]);
}
int main()
{
    int n=read_int(),m=read_int(),M=read_int(),N=read_int(),B=read_int();
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
        deg[u]++;
        deg[v]++;
    }
    _rep(i,1,n)
    q[deg[i]].push_back(i);
    _for(k,1,MAXN){
        _for(j,0,q[k].size()){
            int u=q[k][j];
            if(vis[u])continue;
            vis[u]=true;
            w[u]=k;
            vec[k].push_back(u);
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                deg[v]--;
                q[deg[v]].push_back(v);
            }
        }
    }
    _rep(u,1,n){
        p[u]=u;
        sum[u]=-N;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(w[u]<w[v]){
                sum[u]+=M;
                sum[u]-=B;
            }
            else if(w[u]==w[v]&&u<v)
            sum[u]+=M;
            else if(w[u]>w[v])
```

```
            sum[u]+=B;
        }
    }
    int st=MAXN-1;
    while(vec[st].empty())st--;
    pair<LL,int> ans=make_pair(-inf,0);
    for(int k=st;k;k--){
        for(int u:vec[k]){
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(w[v]<k)continue;
                int x=Find(u),y=Find(v);
                if(x!=y){
                    p[x]=y;
                    sum[y]+=sum[x];
                }
            }
        }
        for(int u:vec[k])
        ans=max(ans,make_pair(sum[Find(u)],k));
    }
    space(ans.second);enter(ans.first);
    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:%E7%BC%93%E5%86%B2%E5%8C%BA&rev=1629684993

Last update: **2021/08/23 10:16**