# 南斯拉夫

## 题目背景

在 $kr$ 世界线中，同盟国赢得了一战，而作为协约国的一员的塞尔维亚，遭受了严厉的惩罚——他们的尼什和马其顿地区被保加利亚夺取，黑山并入奥匈后来成立的伊利里亚自治王国，经济受到严重束缚，而《瓦尔托茨条约》限制了塞尔维亚的军备。

但是这样的日子没有持续多久。在 $1927$ 年，共和派和准军事组织"黑手"推翻了亚历山大国王政府，成立了塞尔维亚共和国，开始秘密扩军备战。在 $1936$ 年资本主义股市崩盘，奥匈陷入民族、经济二重危机中，塞尔维亚得以撕毁屈辱条约，退出奥匈经济圈，有机会复仇雪耻。同年塞尔维亚大选中，共和党在民族危机下联合了社会民主工人党确保政治稳定。在维德节演讲的鼓动下，充满愤怒的塞尔维亚联合巴尔干半岛的兄弟——希腊、罗马尼亚，越过大摩拉瓦河，并把保加利亚揍回了本位面的领土，彻底毁灭了大保加利亚的理想。

后来，在二战爆发后，塞尔维亚得以得到东方斯拉夫兄弟、西方工团主义的战略援助，联合怀着大罗马尼亚理想的罗马尼亚铁卫团，渴望夺回加利西亚的波兰和亚平宁半岛上的奥地利宿敌意大利社会主义共和国，越过德里纳河，对奥战争。最终，工团主义者和东边的俄罗斯民族主义者在二战中获胜，作为胜利者的一员，塞尔维亚实现了伟大理想——成立南斯拉夫联邦共和国。

## 题目描述

南斯拉夫国家中有 $n$ 个民族。由于南斯拉夫民族问题复杂，南斯拉夫成立了民族团结署。现在，该署接到了 $m$ 个要求，每个要求是这样的形式：给 $l_i$ 或 $r_i$ 民族整个活。换句话说，你**只能且必须**选择 $l_i$ 和 $r_i$ 其中一个民族，**只能且必须**选择为其不满值 $+1$ 或 $-1$。现在为了维护民族平等，你需要满足每个需求并且使得最后每个民族的不满值**都**为 $0$，或者输出不可行。

## 输入格式

第一行一个数字 $T$。这里 $T$ 的意思是，这是第 $T$ 个测试点。
第二行两个整数 $n, m$。
接下来 $m$ 行，每行两个数字 $l_i, r_i$，表示第 $i$ 个要求。
含义如上所述。

## 输出格式

如果不可行，那么输出一行"$Yugoslavia\ is\ unstable!$"（不含引号）。
如果可行，输出 $1$ 行，包含 $m$ 个整数 $z_i, i \in [1, m]$， $z_i \in \{1, 2, 3, 4\}$。
$1$ 代表选择 $l_i$ 并且增加不满值；
$2$ 代表选择 $r_i$ 并且增加不满值；
$3$ 代表选择 $l_i$ 并且减少不满值；
$4$ 代表选择 $r_i$ 并且减少不满值。
有多组解输出任意一组。

## 样例输入1

```
1
5 8
1 2
3 5
4 5
2 3
1 4
2 4
1 3
1 5
```

## 样例输出1

```
4 3 4 1 4 2 2 2
```

## 样例输入2

```
1
15 20
7 15
10 12
2 10
11 14
1 15
4 6
10 14
6 9
10 11
7 10
6 12
3 14
4 11
4 13
5 7
5 14
1 5
3 4
7 9
3 15
```

## 样例输出2

```
3 4 4 3 4 3 4 4 2 2 2 2 3 1 2 3 2 2 2 2
```

## 数据规模和约定

对于所有数据，保证 $n, m \leq 10^6, l_i, r_i \in [1, n], l_i < r_i$。
对于测试点 $1, 2$，保证 $n \leq 5$。
对于测试点 $3$，保证 $m \leq 10$。保证合理的暴力你可以过。
对于测试点 $4$，保证 $m \leq 20$。
对于测试点 $5$，保证 $l_i = r_{i-1}, r_i = l_i + 1$。

对于测试点 $6$，保证 $l_i = 1$。
对于测试点 $7, 8$，保证 $n, m \leq 10^3$。
对于测试点 $9, 10$，无特殊保证。
数据随机生成。但是不可行的次数不多于 $2$。

# 题解

## 题意解释

我们来提取题目的关键信息。

有 $n$ 个民族，每个民族拥有一个不满值，初始均为 $0$ . 现在要进行 $m$ 次操作，每次操作给出两个民族，我们需要在给出的两个民族中选择一个民族并将其不满值增加 $1$ 或减少 $1$ 。我们需要给出操作方案使得m次操作后所有民族的不满值仍然均为 $0$ ， 若无解则告知无解。

## 知识点提炼

深度优先搜索、图、并查集（可不需要）

## 核心解题思路

从输入格式来看，我们可以发现，对给出的民族 $l_i$ 和 $r_i$ 每次操作有四种选择：

$1$ . 选择 $l_i$ 并且增加不满值
$2$ . 选择 $r_i$ 并且增加不满值
$3$ . 选择 $l_i$ 并且减少不满值
$4$ . 选择 $r_i$ 并且减少不满值

根据这样的操作方式，若要某个民族最终的不满值为0，它一定需要满足以下的性质：

1、在偶数个操作中被选择改变不满值

2、在上述偶数个操作中，一半的操作增加其不满值，一半的操作减少其不满值

否则，操作后其不满值就非 $0$.

显然，每个民族都需要满足上述两个性质。我们还可以发现，在满足性质1的条件下，我们只要任意分配操作，使堆某个民族一半的操作增加其不满值，一半的操作减少其不满值即可满足性质 $2$。

### 测试点 1-4

此时，操作的数量较少，对于每个操作，我们可以枚举其具体选择，在枚举时，我们可以不关心本次操作是增加还是减少不满值，而只关注其是否满足性质 $1$ ——每个民族都只被偶数个操作选中。因为根据上面的讨论，在满足性质1的条件下一定存在满足性质2的构造。

如此，每个操作便只需要枚举两种可能，在每次枚举后判断其是否满足性质 $1$，便可求出结果，枚举后的验证是 $O(m)$ 的，$m$ 次操作至多涉及 $2m$ 个民族。

时间复杂度为 $O(m \cdot 2^m)$ , 预期得分 $40$ 分

参考代码如下：

```
#include<iostream>
#include<cstdio>
#include<vector>
```

```cpp
using namespace std;

int T,n,m;
int l[25],r[25];
vector<int> cnt[1000010];
bool choice[25],flag;    // choice为false表示选择第一个民族，否则表示选择第二个民族
int ans[25]; // ans存储答案，中间计算过程中用于记录其增加还是减少不满值
void dfs(int x)
{
    if(!flag)
        return;
    if(x==m+1)
    {
        for(int i=1;i<=m;i++)
            if(cnt[l[i]].size()&1||cnt[r[i]].size()&1)
                return;
        for(int i=1;i<=m;i++)
        {
            for(int j=0;j<cnt[l[i]].size()/2;j++)
                ans[cnt[l[i]][j]]=1;
            for(int j=0;j<cnt[r[i]].size()/2;j++)
                ans[cnt[r[i]][j]]=1;
        }

        for(int i=1;i<=m;i++)
            if(choice[i]&&ans[i])// 选择第二个民族，增加不满值
                printf("2 ");
            else if(!choice[i]&&ans[i])// 选择第一个民族，增加不满值
                printf("1 ");
            else if(choice[i]&&!ans[i])// 选择第二个民族，减少不满值
                printf("4 ");
            else // 选择第一个民族，减少不满值
                printf("3 ");
        flag=false;
        return;
    }
    cnt[l[x]].push_back(x);
    choice[x]=false;
    dfs(x+1);
    cnt[l[x]].pop_back();  //回溯

    cnt[r[x]].push_back(x);
    choice[x]=true;
    dfs(x+1);
    cnt[r[x]].pop_back();   //回溯
}

int main()
{
    freopen("yugo.in","r",stdin);
    freopen("yugo.out","w",stdout);
    scanf("%d%d%d",&T,&n,&m);
    while(T--)
    {
        flag=true;
```

```
        for(int i=1;i<=m;i++)
            scanf("%d%d",&l[i],&r[i]);
        dfs(1);
        // 恢复变量，便于下次计算
        for(int i=1;i<=m;i++)
            cnt[l[i]].clear(),cnt[r[i]].clear(),ans[i]=0;
        if(flag)
            printf("Yugoslavia is unstable!\n");
        else
            printf("\n");
    }

    return 0;
}
```

### 测试点5：

由测试点 5 的性质，我们可以进行特殊构造，若 $m$ 不为偶数，则其不满足性质1，无解。否则，第奇数个操作回答 2，第偶数个操作回答 3 即可，如此便可满足性质1与性质2.

例如，假如操作序列是：

```
1 2

2 3

3 4

4 5
```

那么第一个和第二个操作使第二个民族的不满值增加 1再减少 1 ，第三个和第四个操作使第四个民族的不满值增加 1再减少 1 ，最终所有民族的不满值均为 0 .

预期得分 50 分

参考代码关键片段：

```
if(m&1)
    printf("Yugoslavia is unstable!\n");
else
{
    for(int i=1;i<=(m>>1);i++)
        printf("2 3 ");
    printf("\n");
}
```

### 测试点6：

由测试点 6 的性质，我们可以进行特殊构造，若 $m$ 不为偶数，则其不满足性质 1 ，无解。否则，第奇数个操作回答 1，第偶数个操作回答 3 即可。如此便可满足性质 1 与性质 2 .

例如，假如操作序列是：

```
1 2

1 3

1 4
```

```
1 5
```

那么第一个和第三个操作使第一个民族的不满值增加 1，第二个和第四个操作使第一个民族的不满值减少 1 ，最终所有民族的不满值均为 0 .

预期得分 60 分.

参考代码关键片段:

```
if(m&1)
    printf("Yugoslavia is unstable!\n");
else
{
    for(int i=1;i<=(m>>1);i++)
        printf("1 3 ");
    printf("\n");
}
```

## 测试点7-10:

我们来考虑正解。

由于每次操作需要且一定要在 $l_i$和 $r_i$ 之间作出选择。将 $n$ 个民族视为 $n$ 个节点，考虑在 $l_i$ 和 $r_i$建立未定向的有向边。我们可以边的出节点视为该操作选择的民族对应的节点。我们只需要寻找满足性质1的定向方案，使得每个点入度为偶数。

注意到原图可能有多个联通部分，分别考虑即可。且容易发现如果一个连通块边数为奇数则无解。若为偶数则考虑构造答案。对于联通块的判断，我们可以使用并查集来实现。

为了满足性质1，我们只需要关注每个节点入度的奇偶性，只要改变一条边的方向，相应点的奇偶性便被改变，这启发我们利用联通该联通子图的最简单结构——树。我们跑出这个子图的 $dfs$ 树。对于非树边我们任意定向。接下来沿着 $dfs$ 树递归考虑:

对于一个节点，先定连向子节点的边。具体来说，如果子节点目前入度为奇数，就从该节点连向子节点。否则从子节点连向该节点自己。这样能满足除了根之外的每个节点。由于边数为偶数，根节点在其他节点满足情况下一定满足入度为偶数。

如此，我们满足了性质1，在统计过程中记录节点与对应边，对于任意一点，连向他的边的个数一定是偶数。把一半设为减，一半为
加就可以了。

时间复杂度为$O(n+m\log n)$，预期得分 100 分。

参考代码如下:

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
                                            // 快速读入函数
inline int read(){
    int x=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9'){if (ch=='-') f=-1;ch=getchar();}
    while (ch>='0'&&ch<='9'){x=x*10+ch-48;ch=getchar();}
    return x*f;
}

struct edge{                                    //将民族抽象为点，操作抽象为边
```

```
    int u,v,id;
    edge(int a=0,int b=0,int c=0){u=a,v=b,id=c;}
};
vector<edge> G;
vector<int> e[maxn],Ref[maxn];
//记录dfs过程中是否访问的信息，vis记录的是操作（边）是否访问，zfy记录民族（点）是否访问
bool vis[maxn],zfy[maxn];
int n,m,out[maxn],sta[maxn],T,fa[maxn];


void dfs(int u){                            //初次dfs 得到dfs树
    for(int i=0;i<e[u].size();i++){
        int qsy=e[u][i];
        edge E=G[qsy];
        int v=E.v,id=E.id;
        if(zfy[v]||vis[id])continue;
        vis[id]=1;zfy[v]=1;
        dfs(v);
    }
}
                                            // 二次dfs 为边定向
void dfs2(int u){
    for(int i=0;i<e[u].size();i++){
        int qsy=e[u][i];
        edge E=G[qsy];
        int v=E.v,id=E.id;
        if(!vis[id]||zfy[v]==1)continue;
        zfy[v]=1;
        dfs2(v);
        if(out[v]&1){
            out[v]++;
            if(qsy&1)sta[id]=1;
            else sta[id]=2;
        }else{
            out[u]++;
            if(qsy&1)sta[id]=2;
            else sta[id]=1;
        }
    }
}
                                            // 并查集 判断连通块 使用路径压缩
int Find(int x){
    return x==fa[x]?x:fa[x]=Find(fa[x]);
}
int main(){
    freopen("yugo.in","r",stdin);
    freopen("yugo.out","w",stdout);
    T=read();
    n=read();m=read();
    bool spd=1;
    for(int i=1;i<=n;i++)fa[i]=i;
    for(int i=1;i<=m;i++){
        int l,r;
        l=read();r=read();
                                                    //维护并查集
```

```
        fa[Find(l)]=Find(r);
        G.push_back(edge(l,r,i));
        G.push_back(edge(r,l,i));
        e[l].push_back(G.size()-2);
        e[r].push_back(G.size()-1);
    }
    for(int i=1;i<=m;i++){
        out[Find(G[i*2-2].u)]++;
    }
                                                // 判断边的奇偶性，若为奇数条边则无解
    for(int i=1;i<=n;i++){
        if(out[i]&1){
            spd=0;
            break;
        }
    }
    memset(out,0,sizeof(out));
                                                //输出无解
    if(!spd){
        cout<<"Yugoslavia is unstable!"<<endl;
        return 0;
    }
                                                //构建dfs树
    for(int i=1;i<=n;i++){
        if(!zfy[i]){
            zfy[i]=1;
            dfs(i);
        }
    }
    for(int i=1;i<=m;i++){
        if(!vis[i]){
            out[G[2*i-2].v]++;
            sta[i]=2;
        }
    }
    memset(zfy,0,sizeof(zfy));
                                                // 二次dfs，记录答案，为边定向
    for(int i=1;i<=n;i++){
        if(!zfy[i]){
            zfy[i]=1;
            dfs2(i);
        }
    }
    for(int i=1;i<=m;i++){
        if(sta[i]==1)Ref[G[i*2-2].u].push_back(i);
        else Ref[G[i*2-2].v].push_back(i);
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<Ref[i].size()/2;j++){  // 选择一半的民族标注为减少  另一半已标注为增加
            if(sta[Ref[i][j]]==1)sta[Ref[i][j]]=3;
            else sta[Ref[i][j]]=4;
        }
    }
```

```
                                                    // 输出答案
    for(int i=1;i<=m;i++){
        putchar(sta[i]+'0');putchar(' ');
    }
    return 0;
}

```

## 更好的解法

事实上，并查集并非必须的步骤，使用并查集的目的是对连通块中的边进行计数并进行奇偶性判断。我们通过 $BFS$ 确定每个节点所在的连通块并进行统计，可以使时间复杂度降低至 $O(n+m)$

这里不能使用 $DFS$，存在栈溢出的风险

预计得分：$100$

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
                                                    // 快速读入函数
inline int read(){
    int x=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9'){if (ch=='-') f=-1;ch=getchar();}
    while (ch>='0'&&ch<='9'){x=x*10+ch-48;ch=getchar();}
    return x*f;
}

struct edge{                                        //将民族抽象为点，操作抽象为边
    int u,v,id;
    edge(int a=0,int b=0,int c=0){u=a,v=b,id=c;}
};
vector<edge> G;
vector<int> e[maxn],Ref[maxn];
queue<int> q;
//记录dfs过程中是否访问的信息，vis记录的是操作（边）是否访问，zfy记录民族（点）是否访问
bool vis[maxn],zfy[maxn];
int n,m,out[maxn],sta[maxn],T,belong[maxn],cnt[maxn],tot=0;


void dfs(int u){                                    //初次dfs 得到dfs树
    belong[u]=tot;
    for(int i=0;i<e[u].size();i++){
        int qsy=e[u][i];
        edge E=G[qsy];
        int v=E.v,id=E.id;
        if(zfy[v]||vis[id])continue;
        vis[id]=1;zfy[v]=1;
        dfs(v);
    }
}
                                                    // 二次dfs 为边定向
void dfs2(int u){
    for(int i=0;i<e[u].size();i++){
```

```
        int qsy=e[u][i];
        edge E=G[qsy];
        int v=E.v,id=E.id;
        if(!vis[id]||zfy[v]==1)continue;
        zfy[v]=1;
        dfs2(v);
        if(out[v]&1){
            out[v]++;
            if(qsy&1)sta[id]=1;
            else sta[id]=2;
        }else{
            out[u]++;
            if(qsy&1)sta[id]=2;
            else sta[id]=1;
        }
    }
}

int main(){
    freopen("yugo.in","r",stdin);
    freopen("yugo.out","w",stdout);
    T=read();
    n=read();m=read();
    bool spd=1;

    for(int i=1;i<=m;i++){
        int l,r;
        l=read();r=read();

        G.push_back(edge(l,r,i));
        G.push_back(edge(r,l,i));
        e[l].push_back(G.size()-2);
        e[r].push_back(G.size()-1);
    }

    memset(out,0,sizeof(out));

    for(int i=1;i<=n;i++){  //BFS
        if(!zfy[i]){
            tot++;
            q.push(i);
            while(q.size())
            {
                int uu=q.front();
                belong[uu]=tot;
                zfy[uu]=1;
                q.pop();

            for(int i=0;i<e[uu].size();i++){
                int qsy=e[uu][i];
                edge E=G[qsy];
                int v=E.v,id=E.id;
                if(zfy[v])continue;
                zfy[v]=1;
                q.push(v);
```

```
                }
            }
        }
    }

    for(int i=1;i<=m;i++){
        cnt[belong[G[i*2-2].u]]++;
    }
                                                    // 判断边的奇偶性，若为奇数条边则无解
    for(int i=1;i<=tot;i++){
        if(cnt[i]&1){
            spd=0;
            break;
        }
    }

                                                    //输出无解
    if(!spd){
        cout<<"Yugoslavia is unstable!"<<endl;
        return 0;
    }


    memset(zfy,0,sizeof(zfy));
                                                    //构建dfs树
    for(int i=1;i<=n;i++){
        if(!zfy[i]){
            zfy[i]=1;
            dfs(i);
        }
    }

    for(int i=1;i<=m;i++){
        if(!vis[i]){
            out[G[2*i-2].v]++;
            sta[i]=2;
        }
    }


    memset(zfy,0,sizeof(zfy));
                                                    // 二次dfs，记录答案，为边定向
    for(int i=1;i<=n;i++){
        if(!zfy[i]){
            zfy[i]=1;
            dfs2(i);
        }
    }
    for(int i=1;i<=m;i++){
        if(sta[i]==1)Ref[G[i*2-2].u].push_back(i);
        else Ref[G[i*2-2].v].push_back(i);
    }
    for(int i=1;i<=n;i++){
```

```
        for(int j=0;j<Ref[i].size()/2;j++){  // 选择一半的民族标注为减少  另一半已标注为增加
            if(sta[Ref[i][j]]==1)sta[Ref[i][j]]=3;
            else sta[Ref[i][j]]=4;
        }
    }
                                                // 输出答案
    for(int i=1;i<=m;i++){
        putchar(sta[i]+'0');putchar(' ');
    }
    return 0;
}

```

## 其他解法:

可以使用非递归的方式遍历，如此能降低常数，实际运行效率将会提高。对于部分分的构造可以采用其他方式，关键在于捕捉到满足题目条件所需要满足的性质。

并查集可以结合使用按秩合并与路径压缩，进一步降低时间复杂度。

## 易错点:

本题有多组数据，没有清空或恢复某些变量或数组可能导致后续计算错误。

进行 $dfs$ 时需要进行回溯，使用 $dfs$ 进行遍历时许明确需要记录或更新的信息。