目录

1 并查集

1.1 应用场景

并查集是一种树型数据结构，用于处理一些不相交集合的合并与查询。

1.2 核心思想

每个集合选择一个代理节点 (集合中所有节点所构成树的根节点) 作为集合身份标号。

通过查询两个节点集合的代理节点是否一致来判断两个节点是否属于同一集合。

通过将一个集合的代理节点指向另一个结合的代理节点的方式来合并两个结合。

1.3 问题定义

给定 N 个节点，初始每个节点单独为一个集合。

表 1: 符号及名称解释

符号及名词	含义
集合 U 的节点树	集合 U 中所有节点所构成的树形结构
集合 U 的代理节点	集合 U 的节点树的根节点，代理节点的 ID 为集合 U 的标识符
fa[v]	v 所在集合的节点树中 v 节点的父节点
w[v]	节点 v 与节点 fa[v] 之间边的权重

1.4 主要流程

1.4.1 初始化

最初每个节点的各自为独立的集合，所以每个节点所在集合的代理节点为其本身。

```
void init(int n){
    for (int i = 0 ;i<= n ;i++)
        fa[i] = i;
}
```

1.4.2 查询

查询节点 v 所在集合的代理节点，并在此过程进行路径压缩。

路径压缩即为在查询代理节点的过程中，将所有经过的节点的 fa 值都修改为代理节点，加速之后的查询操作。

```
int Find(int v){
    if (v == fa[v])
```

```
        return v;
    fa[v] = Find(fa[v]);
    return fa[v];
}
```

1.4.3 合并

将节点 v 与节点 u 所在的集合合并

```
void update(int u, int v){
    int fu = Find(u);
    int fv = Find(v);
    fa[fu] = fv;
}
```

1.5 带权并查集

在原始的并查集中，每个集合树中的边是没有权重的。然而在某些应用场景中，同一个集合中的元素是有相对关系的 (比如集合中有三类节点 A，B，C，其中 A 吃 B，B 吃 C，C 吃 A)，此时可以通过为集合树中的边加权的方式用以维护各个节点之间的关系。

带权并查集的难点主要在于权重含义的设计以及权重的维护。

边权意义为如下表所示 主要流程变更如下:

表 2: 边权值含义表

边权值	含义
w[v]=0	v 节点与其父节点为同一类型节点
w[v]=1	v 节点会吃掉其父节点
w[v]=2	v 节点会被其父节点吃掉

1.5.1 初始化

初始化边权为 0，表示当前节点与其父节点为相同类型的节点。

```
void init(int n){
   for (int i = 0 ;i<= n ;i++){
       fa[i] = i;
       w[i] = 0;
   }
}
```

1.5.2 查询

当执行 w[v] 的时候，fa[v] 的父亲节点即为代理节点。由于节点之间的关系是可以递推的，因此我们可以通过 w[v](v 与父节点之间的关系) 与 w[fa[v]](fa[v] 与代理节点之间的关系) 推断出 v 节点与代理节点之间的关系，具体实现如下所示。

```
int Find(int v){
    if (v == fa[v])
        return v;
    int fv = Find(fa[v]);
    w[v] = (w[v] + w[fa[v]])%3;
    fa[v] = fv;
    return fa[v];
}
```

其中关系递推可以通过公式 w[v] = (w[v] + w[fa[v]])%3 来计算

1.5.3 合并

在执行合并操作前，我们要先查找节点 u 和节点 v 的代理节点，在查找完代理节点后，u 与 v 的父节点必然会成为代理节点。

我们已知 u 与 v 之间的关系 typ，u 与代理节点之间的关系 w[u] 以及 v 与代理节点之间的关系 w[v]，即可推断出 u 代理节点与 v 代理节点之间的关系。

在合并操作的过程中，我们先将 u 节点的代理节点作为 v 节点代理节点的子节点，并根据两个代理节点之间的关系，更新对应边的权重，具体实现如下所示:

```
int update(int u,int v, int typ){
    //type 0 表示u与v类型相同; 1 表示 u吃v; 2 表示 v吃u
    int fu = Find(u);
    int fv = Find(v);
    if (fu == fv){
        //当两个节点处于同一个集合，看看当前添加的关系与原有关系是否产生冲突
        //产生冲突返回1，不产生冲突返回0
        if ((w[u]-w[v]+3)%3!=typ)
            return 1;
        else
            return 0;
    }
    fa[fu] = fv;
    //计算fu与fv之间的关系
    w[fu] = (-w[u] + typ + w[v] + 3)%3;
    return 0;
}
```

1.6 相关题目

1.6.1 POJ 1611

标称

1.6.2 HDU 1232

标称

1.6.3 POJ 1182

标称

1.6.4 POJ 1456

标称