并查集

前言

并查集应用广泛，用于判断元素是否属于同一个集合以及合并集合。

复杂度

最坏到 $O(n)$□单独使用路径压缩达到 $O(n+f(1+\log_{2+f/n}n))$□单独使用按秩合并可达到 $O(\log{n})$□两者同时使用可以达到 $O(\alpha (n))$□其中 $\alpha (n)$ 是阿克曼函数的反函数，可以认为非常小。按秩合并即每次将小的集合合并到大的集合中去，可以减少修改次数。

概念

每个集合有一个代表元素，每一个元素通过一个数组记录其所属集合的代表元素。判断两个元素是否属于同一个集合时查询它们的代表元素是否相同，合并集合时将一个集合的代表元素更替为另一个集合的代表元素。

操作

初始化

每一个元素的代表元素是它自身，每一个元素的高度是0（高度在合并时用到）。用了 fa 数组表示代表元素□m 表示集合元素个数。

```
for(int i=1;i<=n;i++)
{
    fa[i]=i;
    m[i]=1;
}
```

合并

这里把合并放到查询前面，有助于理解（也许），用到的 find 函数就是查找代表元素的意思。把一个集合的代表元素更替为零一个集合的代表元素，这里体现为将这个代表元素的 fa 更换为另一个代表元素，以后再进行其他元素 fa 的更迭，有 lazy_tag 的感觉。这里将高度小的合并到高度大的里面去，从而减少了以后更改的工作量。

```
x=find(x),y=find(y);
if(x==y)
    return;
if(m[x]>m[y])
```

```
    std::swap(x,y)
m[y]+=m[x];
fa[x]=y;
```

查找

查找元素的代表元素。根据上面的合并的过程，查找的方式就是递归查找 fa[x] 的 fa 的 fa 的 fa……直某个元素的 fa 是它自己。在这里用了一个叫路径压缩的方法，即在查询过程中顺带将所有涉及到的祖辈元素的 fa 都改成集合的代表元素，这样只更新了用到的点，并且为后来的操作提供了便利。

```
return fa[u] == u ? u : (fa[u] = find(fa[u]));
```

权值

元素与代表元素之间的联系可以带有权值，这样代表元素相同的两个元素可以通过各自与代表元素之间的权值来判断关系。~~比如负负得正~~

可持久化

有一些问题要求回溯到之前的某一个版本进行查询或修改，这个时候就要求我们的并查集可持久化。
这个地方的前置知识点就是主席树。因为主席树只支持单点修改，在可持久化的版本里我们就不能使用路径压缩了，因此我们要尽量减少在某一个版本中查询代表元素的时间的话就应当仍然使用按秩合并。
具体的操作就是用主席树把 fa 数组和 m 数组存起来，每一次查询或更改的时候先找一下这个值对应的版本里的位置。
代码就不贴了，就是主席树和并查集的结合。

例题

hdu 1232 畅通工程

hdu1232

题意：某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

这一题就是模板题了。进行并查集的操作，最后扫一遍看看有多少元素的代表元素是它本身，即多少块未相互连通的城市集合，也就是说要铺设这个数减一条道路。 这里面由于m已经被使用了，因此集合中元素个数用 h 数组表示

```
#include<cstdio>
#include<algorithm>
using namespace std;
```

```
int n,m,fa[1005],h[1005];
int Find(int x)
{
    return x==fa[x]?x:(fa[x]=Find(fa[x]));
}
void Merge(int x,int y)
{
    x=Find(x); y=Find(y);
    if(x==y) return;
    if(h[x]>h[y]) swap(x,y);
    h[y]+=h[x];
    fa[x]=y;
}
int main()
{
    while(scanf("%d%d",&n,&m)&&n)
    {
        for(int i=1;i<=n;i++)
        {
            fa[i]=i;
            h[i]=1;
        }
        for(int i=1,a,b;i<=m;i++)
        {
            scanf("%d%d",&a,&b);
            Merge(a,b);
        }
        int ans=-1;
        for(int i=1;i<=n;i++)
            if(fa[i]==i)
                ans++;
        printf("%d\n",ans);
    }
    return 0;
}
```

poj 2492 A Bug's Life

poj2492

题意：有若干虫子，分为两种性别。虫子们只会与异性交流。给出虫子们之间一些两两交流的情况，判断是否有同性恋虫子。

这一题是带权值的题目。通过数据不能得知虫子的性别，但能得知虫子之间性别的关系。假如通过这样的关系推出两只虫子性别相同，但在数据中又有两只虫子联系的记录，就说明存在同性恋虫子。 用并查集记录这样的关系，对于每一对虫子的联系都是一次合并。

```
#include<cstdio>
#include<algorithm>
using namespace std;
```

```
int t,n,intr,fa[2005][2],m[2005];
int Find(int x)
{
    if(fa[x][0]==x) return x;
    int f=Find(fa[x][0]);
    fa[x][1]=!(fa[x][1]^fa[fa[x][0]][1]);
    return fa[x][0]=f;
}
int Merge(int x,int y)
{
    int fx=Find(x),fy=Find(y);
    if(fx==fy)
    {
        if(fa[x][1]==fa[y][1])
            return 0;
        return 1;
    }
    if(m[fx]>m[fy]) swap(fx,fy),swap(x,y);
    m[fy]+=m[fx];
    fa[fx][0]=fy;
    fa[fx][1]=!(fa[x][1]^(!fa[y][1]));
    return 1;
}
int main()
{
    scanf("%d",&t);
    for(int T=1;T<=t;T++)
    {
        scanf("%d%d",&n,&intr);
        for(int i=1;i<=n;i++)
        {
            fa[i][0]=i;
            m[i]=fa[i][1]=1;
        }
        int v=1;
        for(int i=1,a,b;i<=intr;i++)
        {
            scanf("%d%d",&a,&b);
            v&=Merge(a,b);
        }
        if(T>1) printf("\n\n");
        printf("Scenario #%d:\n",T);
        if(v) printf("No suspicious bugs found!");
        else printf("Suspicious bugs found!");
    }
    return 0;
}
```

poj 1182 食物链

poj1182

题意：有三种动物，处在一个捕食循环上$1\to2\to3\to1$给出若干组捕食或者同类关系，判断假话的数量。假话即是与之前的非假话矛盾的话或不符合数据范围条件的话（这条不是重点）。

这一题也是带权值，处理方式和上一题相似，关键在于处理好权值之间的计算关系。

```
#include<cstdio>
#include<algorithm>
using namespace std;
int n,k,d,x,y,ans=0,fa[50005][2],m[50005];
int Find(int x)
{
    if(fa[x][0]==x) return x;
    int f=Find(fa[x][0]);
    fa[x][1]=(fa[x][1]+fa[fa[x][0]][1])%3;
    return fa[x][0]=f;
}
int Merge(int d,int x,int y)
{
    if(x>n||y>n||(d&&x==y)) return 1;
    int fx=Find(x),fy=Find(y);
    if(fx==fy)
    {
        if((fa[x][1]+d+3)%3==fa[y][1]) return 0;
        return 1;
    }
    if(m[fx]>m[fy]) swap(fx,fy),swap(x,y),d=-d;
    m[fy]+=m[fx];
    fa[fx][0]=fy;
    fa[fx][1]=(fa[y][1]-d-fa[x][1]+3)%3;
    return 0;
}
int main()
{
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;i++)
    {
        fa[i][0]=i;
        fa[i][1]=0;
        m[i]=1;
    }
    for(int i=1;i<=k;i++)
    {
        scanf("%d%d%d",&d,&x,&y);
        ans+=Merge(d-1,x,y);
    }
    printf("%d",ans);
```

```
    return 0;
}
```

参考

https://www.nowcoder.com/profile/2315431/note/detail/316067