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$O(n)$使用路径压缩可达到$O(log{n})$使用启发式合并可以达到$O(\alpha (n))$其中$\alpha (n)$是阿克曼函数的反函数，可以认为非常小（这部分参照网上）。启发式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，每一个点通过一个数组记录代表元素。判断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零一个集合的代表元素。

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。

```
int fx=find(x),fy=find(y);
if(fx==fy)
    return;
if(m[fx]>m[fy])
{
    m[fx]+=m[fy];
```

```
    fa[fy]=fx;
}
else
{
    m[fy]+=m[fx];
    fa[fx]=fy;
}
```

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。

```
int find(int x)
{
    if(fa[x]==x)
        return x;
    return fa[x]=find(fa[x]);
}
```

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。

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启发式合并。
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。

例题

- 模板题 hdu1232
- 带权值 poj1073
- 带权值 poj2492
- 经典食物链 带权值 poj1182
- 可持久化 模板题

再难的题可以在各个oj上根据算法标签找

参考

牛客网