Lesson02---并查集

【本节目标】

- **1. 并查集原理**
- **2. 并查集实现**
- **3. 并查集应用**

1. 并查集原理

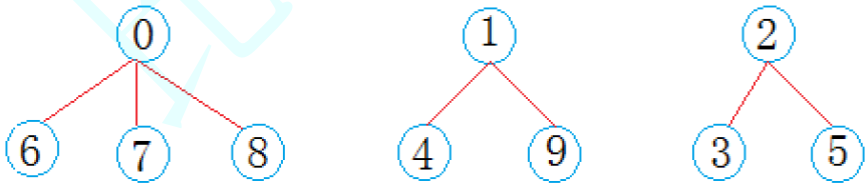
在一些应用问题中，需要**将n个不同的元素划分成一些不相交的集合。开始时，每个元素自成一个单元素集合，然后按一定的规律将归于同一组元素的集合合并。**在此过程中**要反复用到查询某一个元素归属于那个集合的运算。**适合于描述这类问题的抽象数据类型称为**并查集(union-find set)。**

比如：某公司今年校招全国总共招生10人，西安招4人，成都招3人，武汉招3人，10个人来自不同的学校，起先互不相识，每个学生都是一个独立的小团体，现给这些学生进行编号：{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; 给以下数组用来存储该小集体，数组中的数字代表：该小集体中具有成员的个数。(负号下文解释)

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

毕业后，学生们要去公司上班，每个地方的学生自发组织成小分队一起上路，于是：

西安学生小分队s1={0,6,7,8}，成都学生小分队s2={1,4,9}，武汉学生小分队s3={2,3,5}就相互认识了，10个人形成了三个小团体。假设右三个群主0,1,2担任队长，负责大家的出行。



集合的树形表示

一趟火车之旅后，每个小分队成员就互相熟悉，称为了一个朋友圈。

0	1	2	3	4	5	6	7	8	9
-4	-3	-3	2	1	2	0	0	0	1

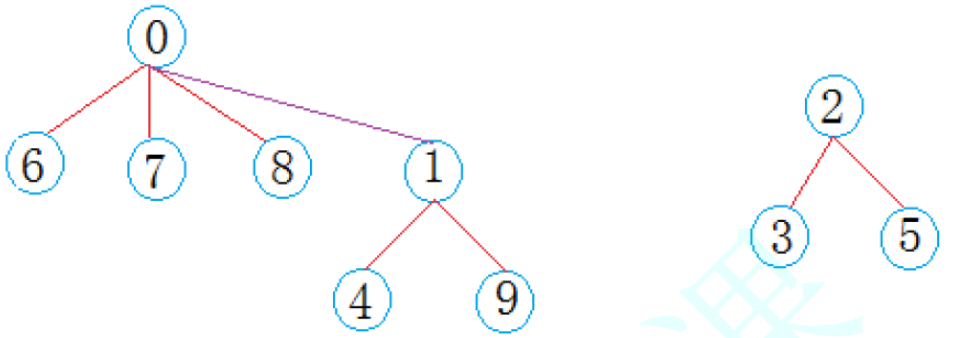
集合s1、s2、s3的森林父指针数组表示

从上图可以看出：编号6,7,8同学属于0号小分队，该小分队中有4人(包含队长0)；编号为4和9的同学属于1号小分队，该小分队有3人(包含队长1)，编号为3和5的同学属于2号小分队，该小分队有3个人(包含队长1)。

仔细观察数组中内融化，可以得出以下结论：

1. **数组的下标对应集合中元素的编号**
2. **数组中如果为负数，负号代表根，数字代表该集合中元素个数**
3. **数组中如果为非负数，代表该元素双亲在数组中的下标**

在公司工作一段时间后，西安小分队中8号同学与成都小分队1号同学奇迹般的走到了一起，两个小圈子的学生相互介绍，最后成为了一个小圈子：



0	1	2	3	4	5	6	7	8	9
-7	0	-3	2	1	2	0	0	0	1

现在0集合有7个人，2集合有3个人，总共两个朋友圈。

通过以上例子可知，并查集一般可以解决一下问题：

1. **查找元素属于哪个集合**

   沿着数组表示树形关系以上一直找到根(即：树中中元素为负数的位置)

2. **查看两个元素是否属于同一个集合**

   沿着数组表示的树形关系往上一直找到树的根，如果根相同表明在同一个集合，否则不在

3. **将两个集合归并成一个集合**
   - 将两个集合中的元素合并
   - 将一个集合名称改成另一个集合的名称
4. **集合的个数**

   遍历数组，数组中元素为负数的个数即为集合的个数。

2. 并查集实现

```
class UnionFindSet
{
public:
 // 初始时，将数组中元素全部设置为1
 UnionFindSet(size_t size)
     : _ufs(size, -1)
 {}

 // 给一个元素的编号，找到该元素所在集合的名称
 int FindRoot(int index)
 {
```

```
        // 如果数组中存储的是负数，找到，否则一直继续
        while(_ufs[index] >= 0)
        {
            index = _ufs[index];
        }

        return index;
    }

    bool Union(int x1, int x2)
    {
        int root1 = FindRoot(x1);
        int root2 = FindRoot(x2);

        // x1已经与x2在同一个集合
        if(root1 == root2)
            return false;

        // 将两个集合中元素合并
        _ufs[root1] += _ufs[root2];

        // 将其中一个集合名称改变成另外一个
        _ufs[root2] = root1;
        return true;
    }

    // 数组中负数的个数，即为集合的个数
    size_t Count()const
    {
        size_t count = 0;
        for(auto e : _ufs)
        {
            if(e < 0)
                ++count;
        }

        return count;
    }

private:
    vector<int> _ufs;
};
```

3. 并查集应用

省份数量

```
class Solution {
public:
    int findCircleNum(vector<vector<int>>& isConnected) {
        // 手动控制并查集
        vector<int> ufs(isConnected.size(), -1);
        // 查找根
        auto findRoot = [&ufs](int x)
        {
            while(ufs[x] >= 0)
                x = ufs[x];
```

```
            return x;
        };

        for(size_t i = 0; i < isConnected.size(); ++i)
        {
            for(size_t j = 0; j < isConnected[i].size(); ++j)
            {
                if(isConnected[i][j] == 1)
                {
                    // 合并集合
                    int root1 = findRoot(i);
                    int root2 = findRoot(j);
                    if(root1 != root2)
                    {
                        ufs[root1] += ufs[root2];
                        ufs[root2] = root1;
                    }
                }
            }
        }

        int n = 0;
        for(auto e : ufs)
        {
            if(e < 0)
                ++n;
        }

        return n;
    }
};
```

[等式方程的可满足性]

```
/*
解题思路：
  1. 将所有"=="两端的字符合并到一个集合中
  2. 检测"!=" 两端的字符是否在同一个结合中，如果在不满足，如果不在满足
*/

class Solution {
public:
    bool equationsPossible(vector<string>& equations) {
        vector<int> ufs(26, -1);
        auto findRoot = [&ufs](int x)
        {
            while(ufs[x] >= 0)
                x = ufs[x];

            return x;
        };

        // 第一遍，先把相等的值加到一个集合中
        for(auto& str : equations)
        {
            if(str[1] == '=')
```

```
            {
                int root1 = findRoot(str[0] - 'a');
                int root2 = findRoot(str[3] - 'a');
                if(root1 != root2)
                {
                    ufs[root1] += ufs[root2];
                    ufs[root2] = root1;
                }
            }
        }

        // 第一遍，先把不相等在不在一个集合，在就相悖了
        // 返回false
        for(auto& str : equations)
        {
            if(str[1] == '!')
            {
                int root1 = findRoot(str[0] - 'a');
                int root2 = findRoot(str[3] - 'a');
                if(root1 == root2)
                {
                    return false;
                }
            }
        }

        return true;
    }
};
```