并查集

Disjoint Sets

例题： 亲戚(Relations)

或许你不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子。如果能得到完整的家谱，就可以判断两个人是否亲戚，但如果两个人的最近公共祖先与他们相隔好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。你将得到一些亲戚关系的信息，如同Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben 是亲戚。请写一个程序，对于亲戚关系的提问，快速给出答案。输入由两部分组成：

第一部分以N，M开始。N为问题涉及的人的个数(1 ≤ N ≤ 20000)。这些人的编号为1,2,3,…,N。下面有M行(1 ≤ M ≤ 100000)，每行有两个数ai, bi，表示已知ai和bi是亲戚。

第二部分以Q开始。以下Q行有Q个询问(1 ≤ Q ≤ 1 000 000)，每行为ci, di，表示询问ci和di是否为亲戚。对于每个询问ci, di，若ci和di为亲戚，则输出Yes，否则输出No。

样例输入：

```
10 7
2 4
5 7
1 3
8 9
1 2
5 6
2 3
3
3 4
7 10
8 9
```

样例输出：

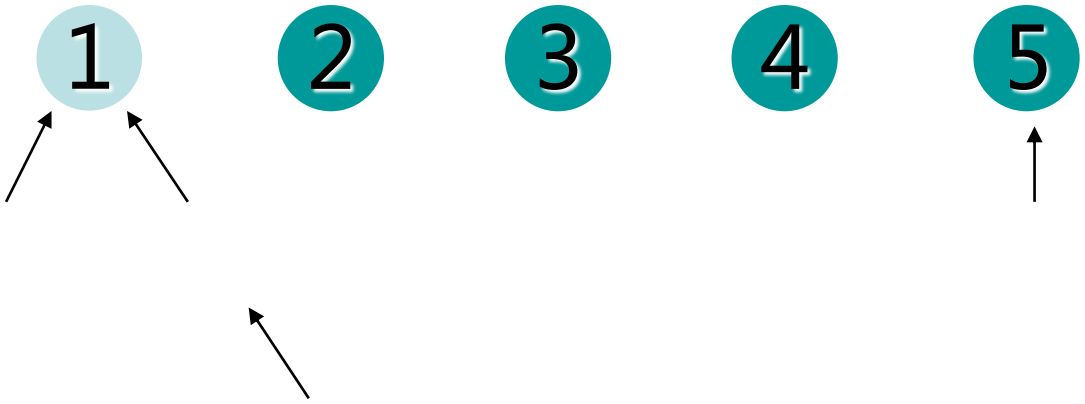
```
Yes
No
Yes
```

floyd 时间复杂度为$O(n^3)$ n<=20000 不可行

并查集Disjoint Sets

- 并查集是一种树型的数据结构，用于处理一些不相交集合的合并问题。
- 并查集的主要操作有
- 1 - 合并两个不相交集合
- 2 - 查看两个元素是否属于同一个集合

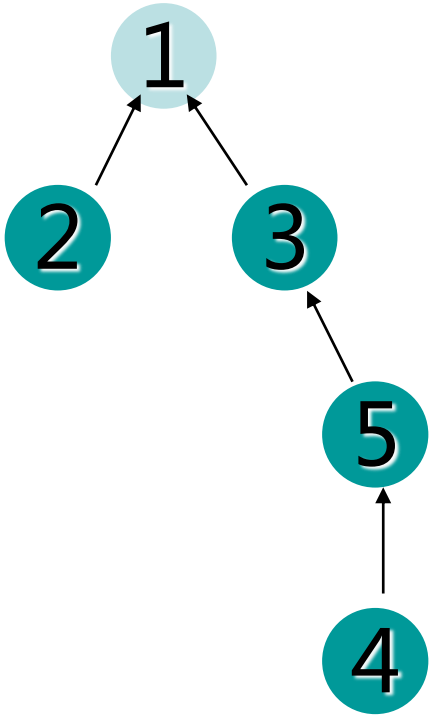
元素的合并图示



- 合并1和2
- 合并1和3
- 合并5和4
- 合并5和3

判断元素是否属于同一集合

- 用father[i]表示元素i的父亲结点，如刚才那个图所示



faher[1]=1

faher[2]=1

faher[3]=1

faher[4]=5

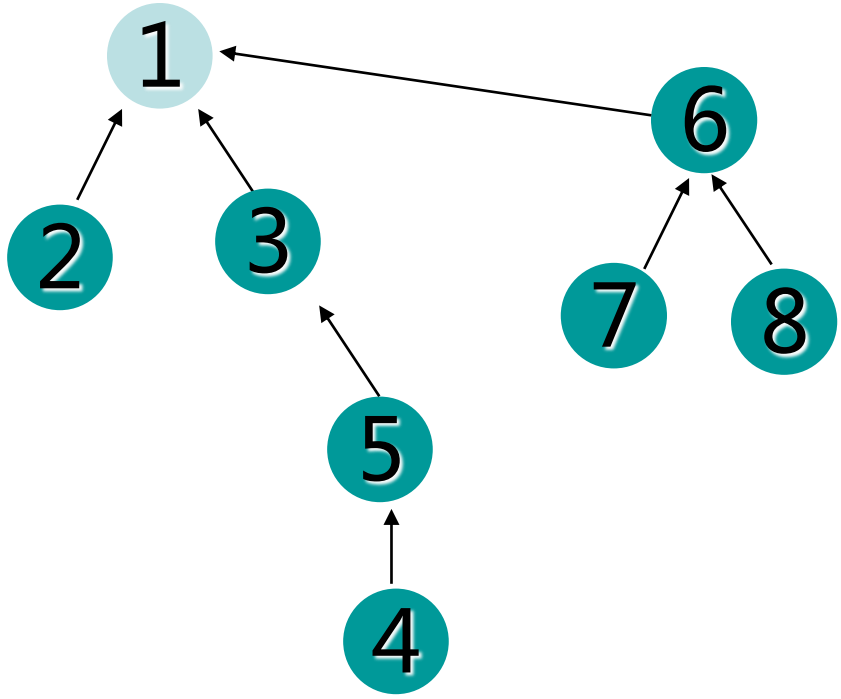
faher[5]=3

判断元素是否属于同一集合

- 用某个元素所在树的根结点表示该元素所在的集合
- 判断两个元素时候属于同一个集合的时候，只需要判断他们所在树的根结点是否一样即可
- 当我们合并两个集合的时候，只需要在两个根结点之间连边即可

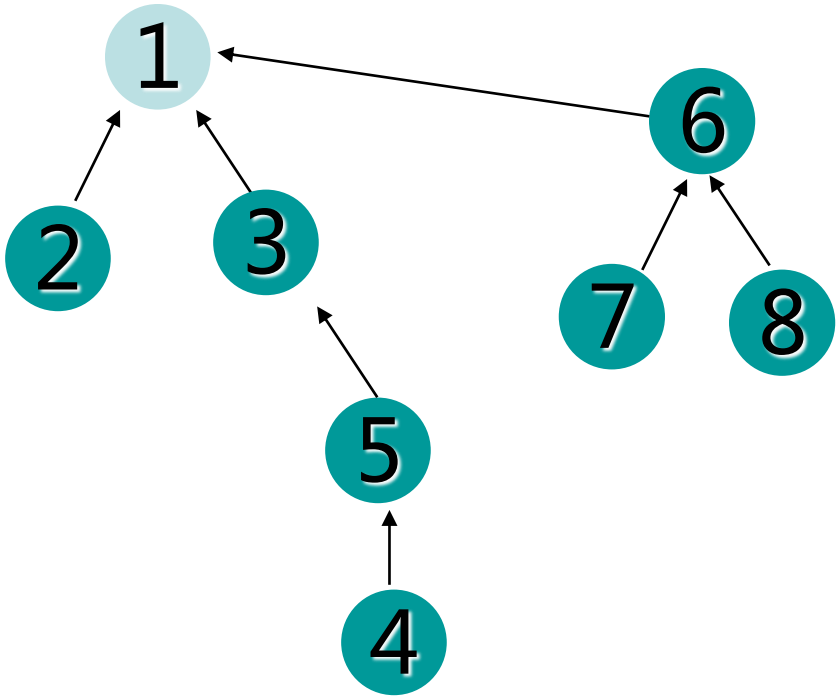
元素的合并图示

- 用某个元素所在树的根结点表示该元素所在的集合
- 判断两个元素时候属于同一个集合的时候，只需要判断他们所在树的根结点是否一样即可
- 当我们合并两个集合的时候，只需要在两个根结点之间连边即可



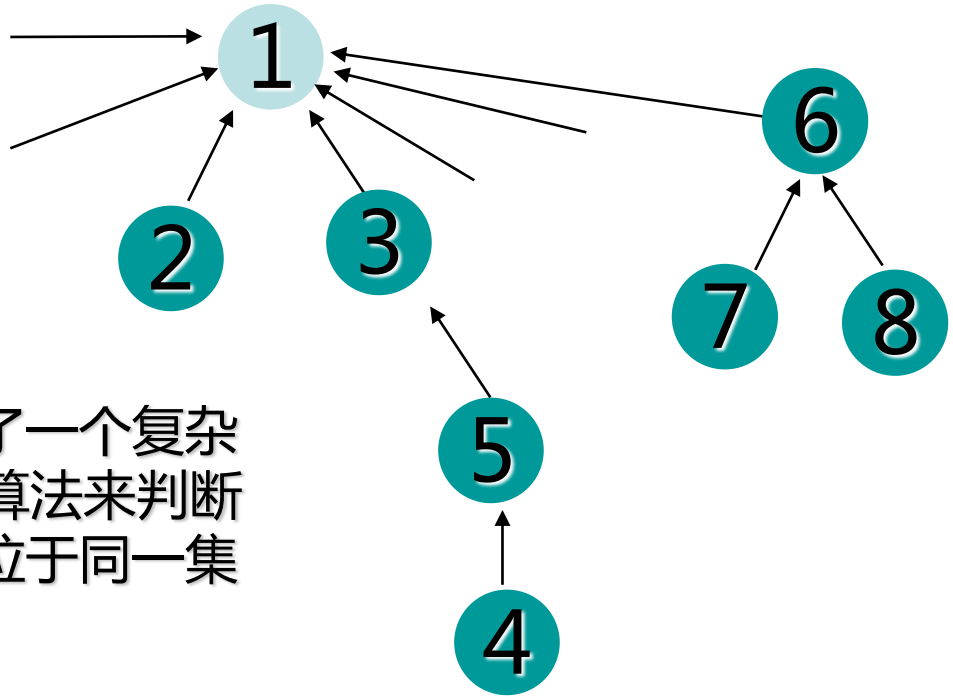
路径压缩

- 判断两个元素是否属于同一集合只需要判断它们所在的树根是否相同，需要O(N)的时间来完成，于是路径压缩产生了作用
- 路径压缩实际上是把一棵树的根节点设置为所有节点的父亲。在找完根结点之后，在递归回来的时候顺便把路径上元素的父亲指针都指向根结点



路径压缩

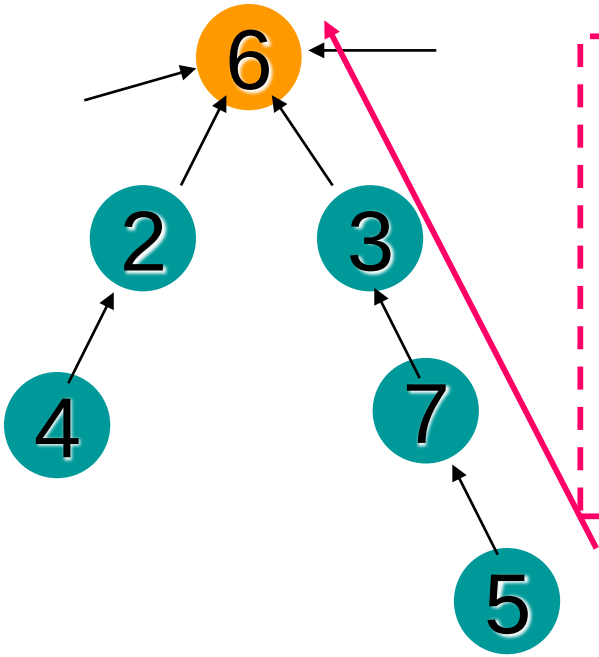
- 判断两个元素是否属于同一集合只需要判断它们所在的树根是否相同，需要O(N)的时间来完成，于是路径压缩产生了作用
- 路径压缩实际上是把一棵树的根节点设置为所有节点的父亲。在找完根结点之后，在递归回来的时候顺便把路径上元素的父亲指针都指向根结点



- 由此我们得到了一个复杂度只是O(1)的算法来判断两个元素是否位于同一集合

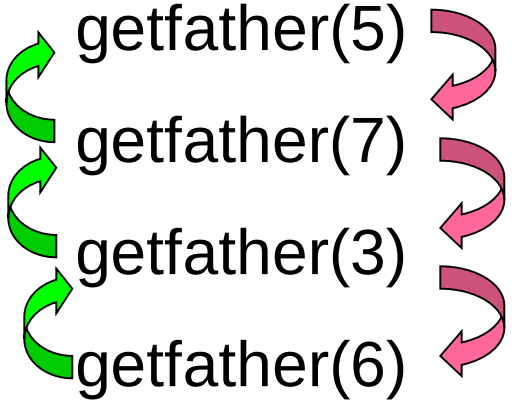
查询同时进行路径压缩

```
int getfather(int v) //查阅元素v的父亲，即查看元素v所在集合的根节点
{
    if (father[v]==v) return v;  //v本身为根节点
    else
    {
        father[v]=getfather(father[v]);
        return father[v];
    }
}
```



通过一次getfater,将5到根之间路径上的所有点的父亲都置为了根节点。

faher[6]=6
faher[4]=2
faher[2]=6
faher[3]=6
faher[7]=6
faher[5]=6




```
int getfather(int v)
{
    if (father[v]==v) return v;
    else
    {
        father[v]=getfather(father[v]);
        return father[v];
    }
}
```

判断两个元素是否位于同一集合,如果不在同一集合，**合并**两个集合

```
bool Merge(int x,int y)
{
        int  fx,fy ;
        fx = getfather(x);
        fy = getfather(y);
        if (fx==fy) return true;
        else
        {
                father[fx] = fy; //合并两个集合
                return false;
        }
}
```

例题： 亲戚(Relations) nkoj 1205

或许你不知道，你的某个朋友是你的亲戚。他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子。如果能得到完整的家谱，就可以判断两个人是否亲戚，但如果两个人的最近公共祖先与他们相隔好几代，使得家谱十分庞大，那么检验亲戚关系实非人力所能及。在这种情况下，最好的帮手就是计算机。你将得到一些亲戚关系的信息，如同Marry和Tom是亲戚，Tom和Ben是亲戚，等等。从这些信息中，你可以推出Marry和Ben 是亲戚。请写一个程序，对于亲戚关系的提问，快速给出答案。输入由两部分组成：

第一部分以N，M开始。N为问题涉及的人的个数(1 ≤ N ≤ 20000)。这些人的编号为1,2,3,...,N。下面有M行(1 ≤ M ≤ 100000)，每行有两个数ai, bi，表示已知ai和bi是亲戚。

第二部分以Q开始。以下Q行有Q个询问(1 ≤ Q ≤ 1 000 000)，每行为ci, di，表示询问ci和di是否为亲戚。对于每个询问ci, di，若ci和di为亲戚，则输出Yes，否则输出No。

样例输入：

```
10 7
2 4
5 7
1 3
8 9
1 2
5 6
2 3
3
3 4
7 10
8 9
```

Merge(x,y)

getfahter(x)

getfather(y)

样例输出：

```
Yes
No
Yes
```

初始化：

```
for(i=1;i<=N;i++)fahter[i]=i;
```

读入关系

```
for(i=1;i<=M;i++)
{
    cin>>x>>y;
    Merge(x,y);
}
```

回答询问

```
for(i=1;i<=Q;i++)
{
    cin>>x>>y;
    if(getfather(x)==getfater(y))cout<<"Yes";else cout<<"No";
}
```

例2：方块游戏 USACO OFEN 2004　nkoj 2281

FJ 和贝茜 用 N (1 <= N <= 30,000)块相同的小立方块玩游戏，小方块编号为1..N。开始时，小方块都单独分开的， 每个看成一个柱子， 即有 N 柱子。 FJ 要 贝茜做 P(1 <= P <= 100,000) 个操作，操作有两种类型：

(1) FJ 要求贝茜把X 号方块所在的柱子放到 Y号所在的柱子上面，成一个新柱子。

(2) FJ 要求贝茜计算 X 号方块所在柱子，它下面有多少个小方块。

请编个程序，帮助贝茜计算。

Input

第一行：一个整数 P

第 2..P+1行：第 i+1行表示第 i个FJ要求的合法操作。

如果这行以'M'开头，后面有两个整数 X,Y 表示要进入(1)操作。

如果这行以'C'开头，后面有一个整数 X，表示要求计算 X所在柱子下面的方块个数。

Output

依次要求计算的值，每次一行。

Sample Input

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

Sample Output

```
1
0
2
```

分析：

判断方块X和Y是否位于同一根柱子：并查集getfather(X,Y)操作

合并方块X和Y所在的柱子：并查集Merge(X,Y)操作

如何统计方块x所在柱子总的方块数？ Count[ ]数组

如何统计方块x所下方的方块数？ Before[ ]数组

Count[x]记录x所在的柱子总的方块数

Before[x]记录从方块x出发到根(最上面)的方块数(也就是x上方的方块数)。
则 **x下方的方块数=Count[x]-Before[x]-1**

```
int getfather(int x)
{
    int dad;
    if(x==father[x]) return x;
    dad=getfather(Father[x]);
    Before[x]=Before[x]+Before[Father[x]];
    Father[x]=dad;
    return Father[x];
}

void Merge(int x,int y)
{
    x=getfather(x);
    y=getfather(y);
    Father[y]=x;  //x放在y的上方，y合并到x所在集合
    Before[y]=Before[y]+Count[x];
    Count[x]=Count[x]+Count[y];
}
```

```
int getfather(int x)//朴素的路径压缩
{
    if (Father[x]==x) return x;
    Father[x]=getfather(Father[x]);
    return Father[x];
}
```