**[并查集总结](http://www.cnblogs.com/DiaoCow/archive/2010/04/22/1718180.html)**

并查集的基本操作：合并两个不相交的集合，查找某个元素所在的集合。

//初始化

void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i < N ; i++)  
 {  
 p[i] = i;  
 rank[i] = 0; 　　//rank[]的意义随情况而定，其一般意义是i的子树深度  
 }  
}

//查找(压缩路径)  
  
int Find(int x)  
{  
 if(x != p[x])  
 {  
 p[x] = Find(p[x]);  
 }  
 return p[x];  
}

//合并（加权规则）  
  
void Union(int x , int y)  
{  
 int a , b;  
  
 a = Find(x); b = Find(y);  
 if(a == b) return ; 　　//x,y已经在一个集合中了

if(rank[a] > rank[b]) //选择子树深度大的为根节点  
 {  
 p[b] = a;  
 }  
 else  
 {  
 p[a] = b;  
 if(rank[a] == rank[b]) rank[b]++;  
 }  
}

在实际做题中，发现对于较难的题目并不采用加权规则合并（只是简单合并），但是却在Find的压缩路径中“大作文章”。

1.并查集最经典使用(Kruscal最小生成树算法)：

用来判断添加边<v,w>到集合中去是否会造成回路。理由就是：若需要添加边(v , w) , 但发现Find(v) == Find(w),说明v,w已经在一个集合中(v,w是连通的)，加入(v , w)就必然会产生回路。

这类题目有小希的迷宫，还是畅通工程，Is It A Tree?等等

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//小希的迷宫

#include <stdio.h>  
#define N 100001  
  
int p[N] , rank[N] , b[N] , nFlag , n;  
  
void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i < N ; i++)  
 {  
 p[i] = i;  
 rank[i] = 0;  
 b[i] = 0;  
 }  
}  
  
int Find(int x)  
{  
 if(x != p[x])  
 {  
 p[x] = Find(p[x]);  
 }  
 return p[x];  
}  
  
void Union(int x , int y)  
{  
 int a , b;  
  
 a = Find(x); b = Find(y);  
 if(a == b)   
 {  
 nFlag = 1;  
 return ;  
 }  
  
 n--;　　　　　　　　　　　　　　　　　//计数保证最后一个集合  
 if(rank[a] > rank[b])  
 {  
 p[b] = a;  
 }  
 else  
 {  
 p[a] = b;  
 if(rank[a] == rank[b]) rank[b]++;  
 }  
}  
  
int main(void)  
{  
 int i , j;  
  
   
 while(scanf("%d%d", &i ,&j) && i + j > 0)  
 {   
 n = nFlag = 0;  
 MakeSet();  
 while(i + j > 0)  
 {  
 if(!b[i]) {b[i] = 1; n++;};  
 if(!b[j]) {b[j] = 1; n++;};  
 if(!nFlag) Union(i , j);  
 scanf("%d%d", &i ,&j);  
 }  
 nFlag || n != 1 ? printf("No\n") : printf("Yes\n");  
 }  
 return 0;  
}

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//还是畅通工程代码  
#include <stdio.h>  
#define N 5000  
  
typedef struct Node  
{  
 int i , j;  
 int d;  
}Node;  
  
Node v[N];  
int p[N] , rank[N];  
  
void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i < N ; i++)  
 {  
 p[i] = i;  
 rank[i] = 0;  
 }  
}  
  
int Find(int x)  
{  
 if(x != p[x])  
 {  
 p[x] = Find(p[x]);  
 }  
 return p[x];  
}  
  
int Union(int x , int y)  
{  
 int a , b;  
  
 a = Find(x); b = Find(y);  
 if(a == b) return 0;  
  
 if(rank[a] > rank[b])  
 {  
 p[b] = a;  
 }  
 else  
 {  
 p[a] = b;  
 if(rank[a] == rank[b]) rank[b]++;  
 }  
 return 1;  
}  
  
//快排  
void QuickSort(Node \*arr , int left , int right)  
{  
 int i , j;  
 Node x , nTemp;  
  
 if(left >= right) //边界条件检查  
 return;  
 else  
 {   
 //Partition  
 i = left; j = right + 1; x = arr[i];  
 while(1)  
 {  
 do i++; while(i < j && arr[i].d < x.d);  
 do j--; while(arr[j].d > x.d);  
 if(i > j) break;  
 //swap(i,j)  
 nTemp = arr[i]; arr[i] = arr[j]; arr[j] = nTemp;  
 }  
 //swap(left,j)  
 nTemp = arr[left]; arr[left] = arr[j]; arr[j] = nTemp;  
  
 QuickSort(arr,left,j-1);  
 QuickSort(arr,j+1,right);  
 }  
}  
int main(void)  
{  
 int z , i , j , k , n , sum;  
  
   
 while(scanf("%d", &z))  
 {   
 if(!z) break;  
 k = sum = 0; n = z = z \* (z - 1) / 2;  
 while(z-- > 0)  
 {  
 scanf("%d%d%d", &v[k].i , &v[k].j , &v[k].d);  
 k++;  
 }  
 QuickSort(v , 0 , n - 1);  
 MakeSet();  
 for(k = 0 , z = 0 ; k < n ; k++)  
 {  
 i = v[k].i;   
 j = v[k].j;  
 if(Union(i , j))  
 {  
 sum += v[k].d;  
 if(++z == n - 1) break;  
 }  
   
 }  
 printf("%d\n",sum);  
 }  
 return 0;  
}

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//Is it a tree

#include <stdio.h>  
#define N 100  
  
int p[N] , rank[N] , b[N] , cnt ,nFlag;  
void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i < N ; i++)  
 {  
 p[i] = i;  
 rank[i] = 0;  
 b[i] = 0;  
 }  
}  
  
int Find(int x)  
{  
 if(x != p[x])  
 {  
 p[x] = Find(p[x]);  
 }  
 return p[x];  
}  
  
void Union(int x , int y)  
{  
 int a , b;  
  
 a = Find(x); b = Find(y);  
 if(a == b)  
 {  
 nFlag = 1;  
 return;  
 }  
 cnt++;   
 if(rank[a] > rank[b])  
 {  
 p[b] = a;  
 }  
 else  
 {  
 p[a] = b;  
 if(rank[a] == rank[b])  
 rank[b]++;  
 }  
}  
  
int main(void)  
{  
 int i , j , n , m;  
  
 n = 1;  
 while(scanf("%d%d", &i ,&j) && i + j >= 0)  
 {  
 cnt = nFlag = m = 0;  
 MakeSet();  
 while(i + j)  
 {  
 if(!nFlag) //是否有环  
 {  
 if(!b[i]) {b[i] = 1; m++;}  
 if(!b[j]) {b[j] = 1; m++;}  
 Union(i, j);  
 }  
 scanf("%d%d", &i ,&j);  
 }  
 if(m && m -1 != cnt) nFlag = 1; //是否只有一个根  
 nFlag ? printf("Case %d is not a tree.\n" , n) : printf("Case %d is a tree.\n" , n);  
 n++;  
 }  
 return 0;  
}

2.划分集合(连通子图)：把“直接认识”(直接有关系)或“间接认识”(间接有关系)的元素划分到一个集合，判断最后集合数量或者某个指定元素所在集合中元素个数。(这里统计有个技巧，刚开始的n个元素可以看做是n个集合，以后每成功合并一次，集合数-1)

The Suspects , Ubiquitous Religions , More is better , 畅通工程等等都是这样的题目

3. 扩展使用

1).Cube Stacking , 2).食物链

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//Cube Stacking

#include <stdio.h>  
#define N 30001  
  
int p[N] , rank[N] , up[N];

void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i < N ; i++)  
 {  
 p[i] = i;  
 rank[i] = 1; //i下面的方块数  
 up[i] = 0; //i上面的方块数（不含本身）  
 }  
 rank[0] = 0;  
}  
  
int Find(int x)  
{  
 int k;  
  
 if(x != p[x])   
 {  
 k = p[x];  
 p[x] = Find(p[x]);  
 up[x] += up[k];  
 }  
 return p[x];  
}  
  
void Union(int x , int y)   
{  
 int a , b;  
  
 a = Find(x); b = Find(y); p[b] = a;   
 up[b] += rank[a];   
 rank[a] += rank[b]; //b的子树在下次Find()时更新  
}  
  
  
int main(void)  
{  
 int z , i , j;  
 char c;  
  
 scanf("%d", &z);  
 MakeSet();  
 while(z-- > 0)  
 {   
 scanf(" %c", &c);  
 if(c == 'M')  
 {  
 scanf("%d%d", &i , &j);  
 Union(i,j);  
 }  
 else  
 {  
 scanf("%d", &i);  
 j = Find(i);  
 printf("%d\n",rank[j] - up[i] - 1);  
 }  
 }  
 return 0;  
}

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//食物链

#include <stdio.h>  
#define N 50001  
  
int p[N] , rank[N] , n , k;  
  
void MakeSet()  
{  
 int i;  
  
 for(i = 0 ; i <= n ; i++)  
 {  
 p[i] = i;  
 rank[i] = 0;  
 }  
}  
  
int Find(int x)  
{  
 int f;  
  
 if(x != p[x])  
 {  
 f = p[x];  
 p[x] = Find(p[x]);  
 rank[x] = (rank[x] + rank[f]) % 3;  
 }  
 return p[x];  
}  
  
void Union(int a , int b , int x , int y , int d)  
{  
 p[a] = b;  
 rank[a] = (rank[y] - rank[x] + 2 + d) % 3;  
}  
  
  
int main(void)  
{  
 int d , x , y , a , b , sum;  
  
 scanf("%d%d" , &n ,&k);  
 MakeSet();  
 sum = 0;  
 while(k-- > 0)  
 {  
 scanf("%d%d%d", &d , &x , &y);  
 if(x > n || y > n || (d == 2 && x == y))   
 {  
 sum++;  
 continue;  
 }  
 a = Find(x); b = Find(y);  
 if(a == b)  
 {  
 if((rank[x] - rank[y] + 3) % 3 != d - 1)  
 {  
 sum++;  
 }  
 }  
 else  
 {  
 Union(a,b,x,y,d);  
 }  
 }  
 printf("%d\n", sum);  
 return 0;  
}

这两个题目都是并查集的扩展使用，特别是食物链那题我很难想到用并查集= =! , 这个两个题目都是在Find中做文章，所以要好好理解Find中的递归过程，至于食物链中那些递推式，你把所有情况列一遍(也不是很多)，就一目了然了。