并查集

文章分类:C++编程

1,什么是并查集？

并查集，就是Union-Find Set，也称不相交集合 (Disjoint Set),用于处理一些不相交集合（Disjoint Sets）的合并及查询问题,如其求无向图的连通分量个数等.最完美的应用当属:实现Kruskar算法求最小生成树.

2,并查集的主要操作有:

(1)makeSet(x):把每一个元素初始化为一个集合

初始化后每一个元素的父亲节点是它本身，每一个元素的祖先节点也是它本身(也可以根据情况而变).

(2)findSet(x):查找一个元素所在的集合

查找一个元素所在的集合，其精髓是找到这个元素所在集合的祖先.这个才是并查集判断和合并的最终依据.

判断两个元素是否属于同一集合，只要看他们所在集合的祖先是否相同即可。

(3)Union(x,y):合并x,y所在的两个集合

合并两个不相交集合操作很简单：

利用Find\_Set找到其中两个集合的祖先，将一个集合的祖先指向另一个集合的祖先。

3,并查集的优化:

(1)Find\_Set(x)时 路径压缩

寻找祖先时我们一般采用递归查找，但是当元素很多亦或是整棵树变为一条链时，每次Find\_Set(x)都是O(n)的复杂度，有没有办法减小这个复杂度呢？

这就是路径压缩，即当我们经过"递推"找到祖先节点后，"回溯"的时候顺便将它的子孙节点都直接指向祖先，这样以后再次Find\_Set(x)时复杂度就变成O(1)了.

(2)Union(x,y)时 按秩合并

即合并的时候将元素少的集合合并到元素多的集合中，这样合并之后树的高度会相对较小。

实现代码:

Cpp代码

#defince VMAX 100 //节点个数

int father[VMAX]; //father[x]:元素x的父节点

int rank[VMAX]; //rank[x]:x元素的秩

//初始化操作

void makeSet(int n)

{

for(i = 0; i < n; i++)

{

father[i] = i; //根据实际情况指定的父节点可变化

rank[i] = 0; //根据实际情况初始化秩也有所变化

}

}

//查找操作:两种实现方式

//递归实现

int findSetR(int x)

{

if(x!=father[x])

//通过回溯,完成路径压缩

father[x]=findSetR(father[x]);

return father[x];

}

//非递归实现

int findSet(int x)

{

int r=x;

while(r!=father[r])

r=father[r];

//路径压缩:把x上面的所有节点的父节点都指向祖先节点

int temp;

while(x!=r)

{

temp=father[x];

father[temp]=r;

x=temp;

}

return r;

}

//合并操作

void UnionR(int x,int y)

{

x=findSetR(x);

y=findSetR(y);

if(x==y)

return;

if( rank[x]>rank[y] ) //通常将元素少的集合合并到元素多的集合

father[y]=x;

else

father[x]=y;

if( rank[x]==rank[y] )

rank[y]++;

}

void Union(int x,int y)

{

x=findSet(x);

y=findSet(y);

if(x==y)

return;

if( rank[x]>rank[y] ) //通常将元素少的集合合并到元素多的集合

father[y]=x;

else

father[x]=y;

if( rank[x]==rank[y] )

rank[y]++;

}

4,应用

(1)http://acm.pku.edu.cn/JudgeOnline/problem?id=1611

Cpp代码

#include<iostream>

using namespace std;

int n, m, i, j;

int father[30005], num[30005];

void makeSet(int n)

{

for(i = 0; i < n; i++)

{

father[i] = i; //使用本身做根

num[i] = 1; //记录集合的数目

}

}

int findSetR(int x)

{

if(father[x] != x) //合并后的树的根是不变的

{

father[x] = findSetR(father[x]);

}

return father[x];

}

//非递归实现

int findSet(int x)

{

int r=x;

while(r!=father[r])

r=father[r];

//路径压缩:把x上面的所有节点的父节点都指向祖先节点

int temp;

while(x!=r)

{

temp=father[x];

father[temp]=r;

x=temp;

}

return r;

}

void Union(int a, int b)

{

int x = findSet(a);

int y = findSet(b);

if(x == y)

{

return;

}

if(num[x] <= num[y])

{

father[x] = y;

num[y] += num[x];

}

else

{

father[y] = x;

num[x] += num[y];

}

}

void UnionR(int a, int b)

{

int x = findSetR(a);

int y = findSetR(b);

if(x == y)

{

return;

}

if(num[x] <= num[y])

{

father[x] = y;

num[y] += num[x];

}

else

{

father[y] = x;

num[x] += num[y];

}

}

int main()

{

while(scanf("%d %d", &n, &m)!=EOF && n != 0)

//学生数 小组数

{

makeSet(n);

for(i = 0; i < m; i++)

{

int count, first, b;

//小组成员数目 成员编号0到n-1

scanf("%d %d",&count, &first);

for(j = 1; j < count; j++)

{

scanf("%d",&b);

Union(first,b);

}

}

//0所在集合的数目,就是可能人群数

printf("%d\n",num[findSet(0)]);

}

return 0;

}

(2)http://acm.pku.edu.cn/JudgeOnline/problem?id=2524

分析:实质就是求连通分量的个数

思路:每合并一次,将个数减1,即可得到最后结果.

Cpp代码

#include<iostream>

using namespace std;

int father[50005], rank[50005];

int maxN;

void makeSet(int n)

{

for(int i = 0; i < n; i++)

{

father[i] = i;

rank[i] = 0;

}

}

int findSet(int x)

{

int r=x;

while(r!=father[r])

r=father[r];

int temp;

while(x!=r)

{

temp=father[x];

father[temp]=r;

x=temp;

}

return r;

}

void Union(int x,int y)

{

x=findSet(x);

y=findSet(y);

if(x==y)

return;

maxN--;

if( rank[x]>rank[y] )

father[y]=x;

else

father[x]=y;

if( rank[x]==rank[y] )

rank[y]++;

}

int main()

{

int Case = 1;

int m;

//总人数 问题数

while(scanf("%d %d", &maxN, &m),maxN!=0)

{

makeSet(maxN);

int a, b;

for(int i = 0; i < m; i++)

{

//a和b信仰相同

scanf("%d %d",&a, &b);

a--,b--;

Union(a,b);

}

printf("Case %d: %d\n",Case++,maxN);

}

return 0;

}

(3)http://acm.pku.edu.cn/JudgeOnline/problem?id=2236

思路:并查集。修复一台电脑就标记为修复过了，然后跟其他已经修复过的电脑进行合并，但是合并的条件必须是两台电脑的距离小于D。当要测试两台电脑时，就是判断他们的根结点是否一样.

Cpp代码

#include <iostream>

#include <math.h>

using namespace std;

#define size 1002

int father[size];

int rank[size];

bool isVisit[size];

struct coordinate

{

double x;

double y;

}locate[size];

void makeSet(int x)//创建集合

{

father[x] = x;

rank[x] = 1;

isVisit[x] = false;

}

int findSet(int x)//寻找根结点

{

int r = x;

int temp;

while(r!=father[r])

r = father[r];

while (x!=r)

{

temp = father[x];

father[temp] = r;

x = temp;

}

return r;

}

void Union(int a,int b)//合并集合

{

a = findSet(a);

b = findSet(b);

if(rank[a]>rank[b])

father[b] = a;

else

father[a] = b;

if(rank[a] == rank[b])

rank[b]++;

}

bool IsOk(int a,int b,long distance)//判断两台电脑是否可以连通

{

double k;

k = sqrt((locate[a].x-locate[b].x)\*(locate[a].x-locate[b].x)+(locate[a].y-locate[b].y)\*(locate[a].y-locate[b].y));

return k<=distance;

}

int main()

{

char commander;

int computer;

int p1,p2;

int i;

long distance;

cin>>computer>>distance; //pc数目,有效距离

for(i=1;i<=computer;i++)

makeSet(i);

for (i=1;i<=computer;i++) //输入每台pc的位置

cin>>locate[i].x>>locate[i].y;

while (cin>>commander)

{

if (commander == 'O')

{

cin>>p1;

isVisit[p1] = true;

/\*查看修复的电脑是否可以加入到并查集中\*/

for(i=1;i<=computer;i++)

{

if(isVisit[i]&&i!=p1)

if(IsOk(p1,i,distance)) //满足通讯条件,合并

if(findSet(p1)!=findSet(i))

Union(p1,i);

}

}

else

{

cin>>p1>>p2;

if(findSet(p1)==findSet(p2))

cout<<"SUCCESS"<<endl;

else

cout<<"FAIL"<<endl;

}

}

return 0;

}