|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **华南农业大学数学与信息学院**  程序设计与算法综合实习Ⅱ   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **班级** | | 软件工程六班 | **学号** | 202425220607 | **姓名** | 廖信同 | | **实**  **验**  **题**  **目** | **并查集在Java中的实现**  ■设计性 □综合性 | | | | | | | | **自**  **我**  **评**  **价** | 通过互联网视频教程【并查集 - 基本操作, 路径压缩, 按秩合并】 <https://www.bilibili.com/video/BV1zZRSYUEWV>的学习获得了有关并查集的基础实现，在使用Java完成作业的过程中，按照面向对象的编程思维，较为完好地实现了并查集的基础算法实现，并与数据结构的“图”相结合，使用并查集归类不同的连通分量。自我感觉良好。 | | | | | | | | **教**  **师**  **评**  **语** | 能够实现实验要求的功能 □全部 □部分  算法有新意 □有 □一般  程序运行通过 □全部 □部分  算法注释说明 □完善 □仅有功能说明  接口参数说明 □有 □无  按期上交文档资料及源程序 □所有 □部分  独立完成实验 □能 □不能  体现团队合作精神。 □能够 □不能 | | | | | | | | **成绩** |  | | | | | | | |

并查集在图中连通分量的实践——用Java实现

# 一、要求分析

是并查集是一种**数据结构**，主要用来管理一些**不相交的集合**，支持快速地判断两个元素是否属于同一个集合，以及合并两个集合。通常用于解决**连通性问题**。在我编写的程序中，我希望通过并查集的数据结构及其具体算法，来获取连通图的连通分量的相关信息。

我使用的是Java实现，这是面向对象编程语言，实验报告中要求的函数在此处应表示为“类和方法”。

## 变量信息

各个类的信息如下：

|  |  |
| --- | --- |
| 类Class | 用途Function |
| UnionFindSet | 顶层类，包括下面的两个子类 |
| UFset | 用于封装实现并查集的方法 |

各个方法的信息如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 方法Method | 所属类Class | 返回值Returned | 用途Function |
| UFset | UFset | 无返回值 | UFset的构造方法 |
| Find | UFset | int | 查找一个节点的根结点 |
| Union | UFset | void | 合并两个结点 |
| main | UnionFindSet | void | 程序入口vv |

不同类或方法包含的，存放数据的参数如下，不包括临时变量：

|  |  |  |
| --- | --- | --- |
| 类或方法Class or Method | 参数或变量Value | 用途Function |
| UFset | int[] parent | 节点的父节点 |
|  | int[] size | 根节点对应的大小 |
| Union | int rootx/int rooty | 存放根节点 |
| main | Scanner sc | Scanner类型，用于读取输入 |
|  | int vertex | 结点数，默认标号为0~n-1 |
|  | int edge | 边数 |
|  | UFset graph | UFset类的一个对象 |
|  | (HashSet)Set<Integer> CCset | 哈希集合类型，即Connected Components Set，存放连通分量的个数 |
|  | (HashMap)Map<Integer,ArrayList<Integer>> CComponent | 哈希映射类型，即Connected Component，Intrger类型的键存储根节点，ArrayList类型的值存储根节点所对应的所有连通分量 |

## 输入与输出

根据我写的程序，输入应当为：

顶点数 边数

边1

边2

边3

...

输入值的范围理论上应当满足0~2,147,483,647

在输入完顶点数和边数后，接下来应当输入与边数相等的边，使用x y来表示两个结点连通。

以下为一个输入示例：

6 4

0 1

1 2

3 4

4 5

以下为一个输出示例：

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

2->1

3->4

4->4

5->4

连通分量的个数有：2

2个不同连通分量的成员如下：

[0, 1, 2]

[3, 4, 5]

## 程序所能达到的功能

通过输入的边，判断其根节点是否一致，若不一致，根据形成的树的大小合并。

最终输出为：

1. 经过路径压缩的优化算法后，节点对应的根结点
2. 连通分量的个数
3. 连通分量各自的成员

## 解题思路

并查集的实现比较简单，通过循环或递归的方式访问两个节点各自的根节点，然后判断其对应的树的大小，将小树合并到大树。以下给出查找Find和合并Union的算法代码：

//这里给出递归实现的Find方法，用于查找一个节点的根节点，并实现路径压缩  
 public int Find(int x){  
 if(x!=parent[x])  
 parent[x]=Find(parent[x]);  
 return parent[x];  
 }  
  
 //这里给出Union方法，比较输入边的两个端点x,y各自的根节点的树的大小，将小数合并到大树  
 public void Union(int x,int y){  
 int rootx=Find(x);  
 int rooty=Find(y);  
 if(rootx!=rooty){  
 if(size[rootx]<=size[rooty]){  
 parent[rootx]=rooty;  
 size[rooty]+=size[rootx];  
 }  
 else{  
 parent[rooty]=rootx;  
 size[rootx]+=size[rooty];  
 }  
 }  
 }  
}

此处使用了路径压缩和按秩合并的优化方法，优化程序运行效率。

在实现算法之后，只需要根据输入存储数据，执行算法，最后将结果打印到终端上即可。

以下给出不同的输出对应的算法：

打印每个节点对应的根：

System.*out*.println("经过路径压缩后，每个节点对应的根如下：");  
for(int i=0;i<vertex;i++){  
 System.*out*.println(i+"->"+graph.Find(i));  
}

打印连通分量的个数：

//使用哈希集读取连通分量个数  
Set<Integer> CCset=new HashSet<>();  
for(int i=0;i<vertex;i++){  
 CCset.add(graph.Find(i));  
}  
System.*out*.println("连通分量的个数有："+CCset.size());

打印连通分量各自的成员：

//使用哈希映射来展示不同连通分量成员  
Map<Integer,ArrayList<Integer>> CComponent=new HashMap<>();  
for(int i=0;i<vertex;i++){  
 CComponent.putIfAbsent(graph.Find(i),new ArrayList<>());  
 CComponent.get(graph.Find(i)).add(i);  
}  
System.*out*.println(CCset.size()+"个不同连通分量的成员如下：");  
for(ArrayList<Integer> group:CComponent.values()){  
 System.*out*.println(group);  
}

# 调试分析

代码撰写比较顺利，没有遇到过多的问题。

## 测试数据

以下给出不同的测试数据与输出结果：

### 数据1

常规数据

6 4

0 1

1 2

3 4

4 5

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

2->1

3->4

4->4

5->4

连通分量的个数有：2

2个不同连通分量的成员如下：

[0, 1, 2]

[3, 4, 5]

### 数据2

测试孤立节点

5 0

经过路径压缩后，每个节点对应的根如下：

0->0

1->1

2->2

3->3

4->4

连通分量的个数有：5

5个不同连通分量的成员如下：

[0]

[1]

[2]

[3]

[4]

### 数据3

全连通的情况

4 6

0 1

0 2

0 3

1 2

1 3

2 3

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

2->1

3->1

连通分量的个数有：1

1个不同连通分量的成员如下：

[0, 1, 2, 3]

### 数据4

仅一条边的情况

2 1

0 1

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

连通分量的个数有：1

1个不同连通分量的成员如下：

[0, 1]

### 数据5

稀疏图

7 3

0 1

2 3

4 5

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

2->3

3->3

4->5

5->5

6->6

连通分量的个数有：4

4个不同连通分量的成员如下：

[0, 1]

[2, 3]

[4, 5]

[6]

### 数据6

自环与重复边

5 5

0 1

1 2

2 2

3 4

0 1

经过路径压缩后，每个节点对应的根如下：

0->1

1->1

2->1

3->4

4->4

连通分量的个数有：2

2个不同连通分量的成员如下：

[0, 1, 2]

[3, 4]

## 效率与分析

经过查阅资料，经过路径压缩和按秩排序后，该函数的时间效率为O(α(n))

即反阿克曼函数。基本可以认定为线性复杂度。

空间效率取决于输入数据量的大小。

## 经验与体会

教学视频中已经把该知识点讲解地是十分明白，剩下的就是运用OOP的编程思维进行开发。源代码中使用了类似于cpp的STL库的数据结构，比如哈希集，哈希映射等。查阅官方那个文档即可得到使用方式。

最大的体会算是OOP编程思维的一次实践，在此之前并没有多少编写的经验。

# 附录

