***2018***



**算法实践 课程设计报告**



|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | ACM1601 |
| 学 号： | U201614756 |
| 姓 名： | 江易星 |
| 电 话： | 13995887596 |
| 邮 件： | [1093824239@qq.com](mailto:1093824239@qq.com) |
| 完成日期： | 2019/01/17 |

目 录

[1 课程设计任务的完成情况 错误!未定义书签。](#_Toc497213218)

[1.1 **错误!未定义书签。**](#_Toc497213219)

[1.2 **错误!未定义书签。**](#_Toc497213220)

[2 小组工作情况的汇报 错误!未定义书签。](#_Toc497213221)

[2.1 **错误!未定义书签。**](#_Toc497213222)

[2.2 **错误!未定义书签。**](#_Toc497213223)

[3 专题报告 错误!未定义书签。](#_Toc497213225)

[3.1 相关定义和概念 **错误!未定义书签。**](#_Toc497213226)

[3.2 基本理论和方法 **错误!未定义书签。**](#_Toc497213227)

[4 总结 错误!未定义书签。](#_Toc497213229)

[4.1 已完成的工作 **错误!未定义书签。**](#_Toc497213230)

[4.2 需要进一步完善的地方 **错误!未定义书签。**](#_Toc497213231)

[4.3 心得体会 **错误!未定义书签。**](#_Toc497213233)

# 完成情况

## 习题完成情况

POJ上习题共完成18道：如图1.1所示：



图1.1：习题完成情况

LCA\_RMQ章节完成 4题

并查集章节完成3题

差分约束系统章节完成2题

后缀数组章节完成3题

树状数组章节完成4题

线段树章节完成2题

## 小组工作贡献

小组选择报告专题为并查集专题。

我负责的工作是：

1负责poj1182食物链的题解，准备这部分ppt的内容。

2结合组内其他成员完成的ppt，理解poj1733 parity game的思路。

3负责作报告是对例题的讲解。

# 并查集

## 基本描述

### 并查集的概念：

并查集（Disjoint Set）是一种特殊的数据结构。其定义可以表述如下：将n个不同的元素划分成一组不相交的集合，开始时，每个元素各自成一个单元素的集合，然后按照一定的规律将归于同一组元素的集合合并，在每个集合中，选择其中一个作为这个集合的代表，适合描述这类问题的数据结构称为并查集。

### 并查集的实现

并查集最常见也是最基本的两种操作：

1合并两个集合 Union

2查找某个元素所在的集合 Find

通常用一颗“有根树”来实现并查集，用根节点来代表每一个集合。定义数组Set[n]，如果Set[i] = i，表示i节点是根节点，否则Set[i]表示i节点的父节点。如图2.1所示：

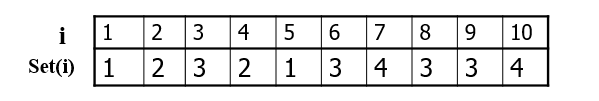


图2.1（a）数组形式（实际实现形式）

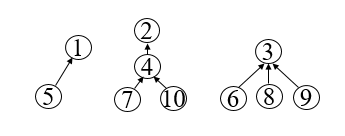


图2.1（b）树形式（逻辑形式）

find的代码实现：找到x所在的集合

find(x)

{

if(Set(x) == x)

{

return x;

}

else

{

return find(Set(x));

}

}

复杂度：Θ（N）

Union的代码实现：合并x和y所在的集合

Union(x,y)

{

int xroot = find(x);

int yroot = find(y);

Set(xroot) = yroot;

}

复杂度：Θ（N）

### 并查集的优化

优化的思路和方向：并查集由于要多次进行Find和Union的操作，所以主要是优化Find和Union 操作的复杂度。而影响Find和Union复杂度的主要因素为树的高度，所以优化的方向是降低树高。

优化1：带树高的Union操作。

每一个节点记录一个height值，该值表示以该节点为根的子树的高度，在Union 时将树高低的树合并到树高较高的树下。如果两棵树一样高，则随意将一棵树合并到另一颗树下。这样做可以避免在合并时，得到一颗高度为N的单向的树，避免最坏情况。

具体实现：

Union(x,y)

{

int xroot = find(x);

int yroot = find(y);

if(height[xroot] > height[yroot])

{

Set[yrooot] = xroot;

}

else

{

Set[xroot] = yroot;

}

}

合并之前两棵树的高度分别为h1 h2，合并后的树高为h。如果h1≠h2 那么h=max（h1,h2）如果h1=h2 则h = h1+1。任意顺序的合并操作之后，得到的包含k个节点的树，高不会超过O（logk）。

优化2：路径压缩

每次进行find操作时，如果路径较长，则修改信息，把路径上所有的节点都直接指向根节点，使得在下一次查找时能够更快找到根节点。示意图如图2.2所示：

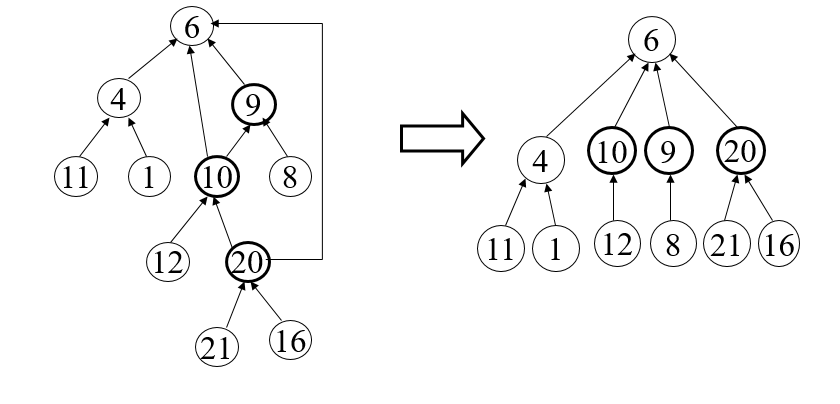


图2.2 路径压缩

具体实现

find(x)

{

r = x;

while(Set[r]!=r) r = Set[r];

i = x;

while(i!=r)

{

j = Set[i];

Set[i] = r;

i = j;

}

return r;

}

## 解题报告

### poj1182食物链

题目描述：

动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。A吃B， B吃C，C吃A。   
 现有N个动物，以1－N编号。每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。   
 有人用两种说法对这N个动物所构成的食物链关系进行描述：   
 第一种说法是"1 X Y"，表示X和Y是同类。   
 第二种说法是"2 X Y"，表示X吃Y。   
 此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。   
 1） 当前的话与前面的某些真的话冲突，就是假话；   
 2） 当前的话中X或Y比N大，就是假话；   
 3） 当前的话表示X吃X，就是假话。   
你的任务是根据给定的N（1 <= N <= 50,000）和K句话（0 <= K <= 100,000），输出假话的总数。

输入：

第一行是两个整数N和K，以一个空格分隔。   
以下K行每行是三个正整数 D，X，Y，两数之间用一个空格隔开，其中D表示说法的种类。   
若D=1，则表示X和Y是同类。   
若D=2，则表示X吃Y。

输出：

一个整数，表示假话的数目。

解题思路：

这题使用到的是并查集的一种变式，称为带权并查集。对于前面所说的一般形式的并查集，从子节点到父节点的边是没有权值的，或者可以认为权值都是一样的，它表示的关系是一种“同类”的含义，即子节点和父节点是属于同一个类。带权并查集中子节点到父节点的边是有权值的，权值代表子节点和父节点之间的关系。边的权值不同代表着子节点和父节点的不同关系。并且在同一棵树中的所有节点互相之间的关系都已经确定，可以由边的权值推导得到。

具体在此题中的做法：把相互关系已经确定的点放到同一颗树种，每个节点处存储有三个信息：

parent：父亲节点

height：以次节点为根的子树的高度

relation：与父节点的关系，也即边权。

对边权进行编码：

0表示：此节点倍父节点吃

1表示：此节点和父节点是同类

2表示：此节点吃父节点

初始化：每个节点单独建一棵树，初始每一个节点的信息都可以确定，parent是自己，height是0，relation是1.

执行过程：每次遇到一句话 1 x y或者2 x y。首先判断x和y是否在同一颗树中，如果在同一棵树中，则说明x和y的关系已经由前面所说的真话所确定。这时推导出x和y的关系并和这句话说的关系比较，如果不同说明这句话是假话。如果x和y不在同一颗树中，那么xy之间并没有确定关系，这是这句话就要背当作真话，并且xy的关系就是所说的关系。根据这个关系我们要把x和y所在的两棵树合并。

路径压缩：在find操作时应同时实现路径压缩，由于路径压缩会改变节点之间的父子关系，因为relation边权表示的是子节点和父节点之间的关系，所以改变了父子关系的同时也需要对relation边权进行更新。

边权更新的规律：

考虑两种最简单的情况：只有三个点a、b和root，

（1）情形1：他们之间的父子关系如图2.3所示：

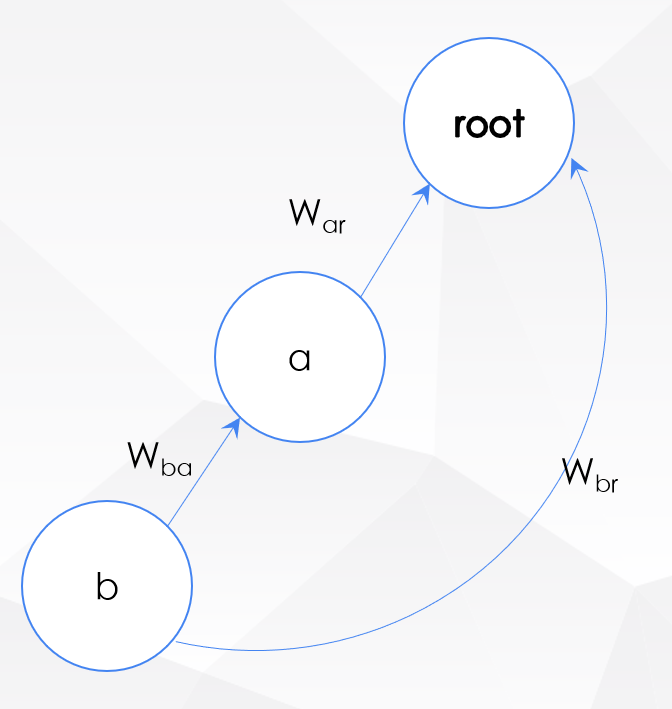


图2.3 一种最简单的情况

节点b的relation记为Wba，节点a的relation记为War，我们需要做的是推导出b个root的relation Wbr.列举出所有可能的情况如表2.4所示：

表2.4：Wba、War与Wbr的关系

|  |  |  |
| --- | --- | --- |
| Wba | War | Wbr |
| 0 | 0 | 2 |
| 0 | 1 | 0 |
| 0 | 2 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 2 | 2 |
| 2 | 0 | 1 |
| 2 | 1 | 2 |
| 2 | 2 | 0 |

观察上表可以得出三者之间的关系；

Wbr=（Wba+War+2）mod 3

用向量的观点来看我们可以发现：

所以可以得到：两条边如果满足

那么对应的边权就可以用

Wbr=（Wba+War+2）mod 3

来计算。

（2）情形2：他们之间的关系如图2.5所示：

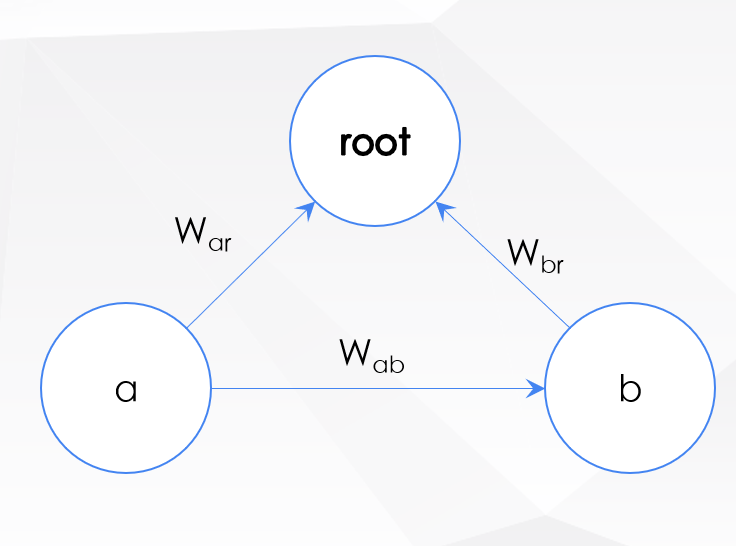


图2.5另一种情形

这里我们发现：

也可以转化成：

由于我们之前已经知道了当一个向量等于另外两个向量相加时的权值关系，所以这里我们只需要知道Wba与Wab的关系就可以带入原来的式子中得到新的关系式。

Wab与Wba的关系如表2.6所示：

表2.6：W­ab与Wba的关系

|  |  |
| --- | --- |
| Wab | Wba |
| 0 | 2 |
| 1 | 1 |
| 2 | 0 |

观察上表很容易发现：W­ab = 2 - Wba

所以我们可以得到 ：当两条边的向量满足

对应边的权值可以由：

Wbr = (War-Wab+4) mod 3

得到。这个关系是可以用来求同一棵树中任意两个节点之间的关系，我们只需要先对这两个节点a、b进行find的操作就可以得到他们和root和关系，在利用这个式子就可以得到Wab

合并操作：Union(x,y)时会把x和y所在的两颗树合并成一棵树，合并时为了保持树的高度为logn，我们需要按照之前的方式，将高度较小的树合并到较高的树下面，同时由于这里用到带权的并查集，所以我们需要确定两个根节点之间的关系。假设x和y不在同一棵树中时，由于在Union之前我们需要先进行find操作，find操作已经进行了路径压缩，所在Union的时候x和y必定都已经直接和各自的根节点相连。

如图2.7所示：

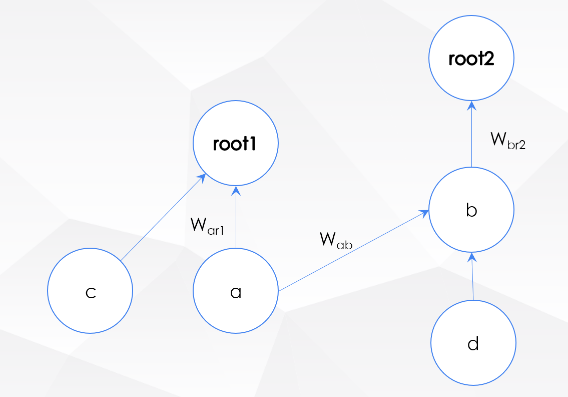


图2.7 Union(a,b)示例

我们需要确定Wr1r2。在途中添加辅助线：一条由root1指向b的向量，如图2.8所示：

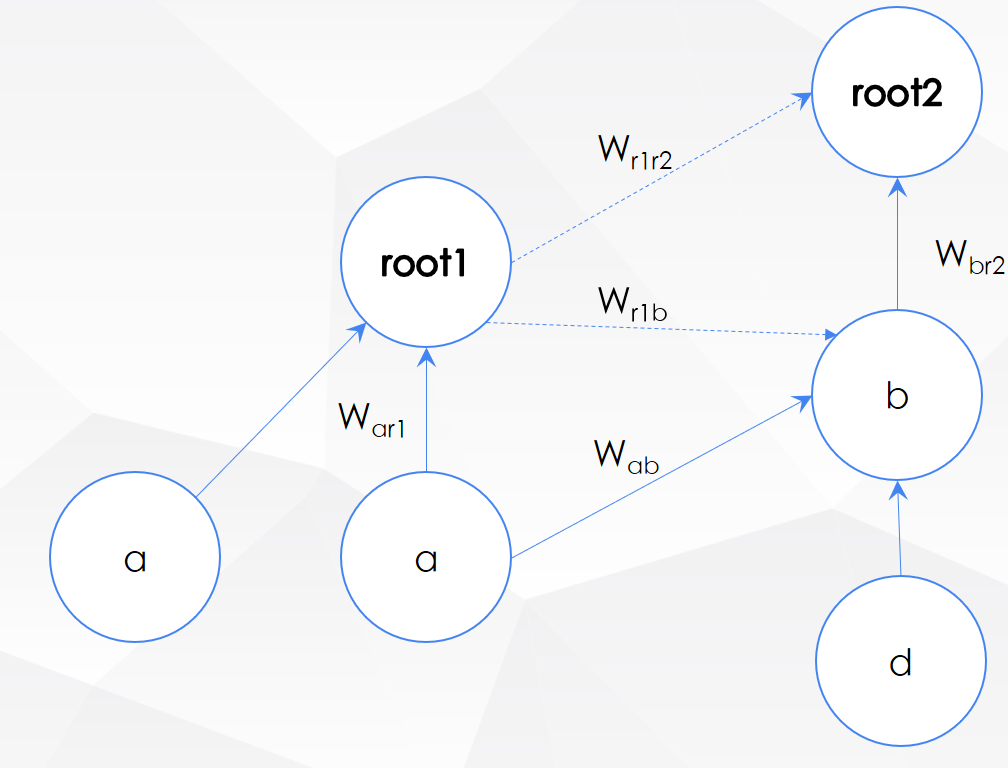


图2.8：加入辅助线

由向量的关系我们可以知道

其中 War1 Wab Wbr2已知，所以根据我们之前推出的关系式，可以求出Wr1r2。

算法设计

伪代码描述

find(x)的伪代码描述：

find(x)

{

int pp = parent[p];

if(parent[p] == p) return p;

else

{

parent[p] = find(parent[p]);//递归实现路径压缩

relation[p] = (relation[p]+relation[pp]+2)%3//实现边权relation的更新

return parent[p];

}

}

Union(x,y)的伪码实现

Union(x,y)

{

int xroot = find(x);

int yroot = find(y);

if(xroot == yroot) return;//在同一棵树中

else

{

if(height[xroot] > height[yroot])

{

parent[yroot] = xroot;//高度低的合并到高度高的下面

relation[yroot] = (relation[x]-((re+relation[y]+2)%3)+1)%3;//计算边权relation

}

else

{

if(height[xroot] == height[yroot])

{

height[xroot] ++;

}

parent[xroot] = yroot;

relation[xroot] = (((re+relation[y]+2)%3)-relation[x]+1)%3;

}

}

}

算法的时间和空间复杂度：

find的时间复杂度：由于Union按照height合并，并且find实现了路径压缩，所以每次find的时间复杂度不会超过树高O(logn)。

Union 的时间复杂度：由于Union先调用了find，时间复杂度为O(logn),之后合并的发操作只需要O(1)即可完成，所以综合起来 Union的时间复杂度为O(logn).

整个算法在判断每一条语句时，都只会使用两次find或者一次Union，时间复杂度均为O(logn)所以最终的时间复杂度为O(mlogn),其中m为语句条数，n为动物数量。

空间复杂度：在实际实现算法时，存储了每个节点的parent height和relation，以及每一话，所以需要的总的空间复杂度为3n+3m = O(N+M).

实现技术：

为了解决这个问题主要运用了下面三个方面的技术：

1带权并查集

2带秩合并：具体体现在每个节点维护一个height，根据height来决定两棵树如何合并

3路径压缩：具体体现在find操作同时实现，进一步压缩树高。

实验结果：

测试用例1如图2.9所示：

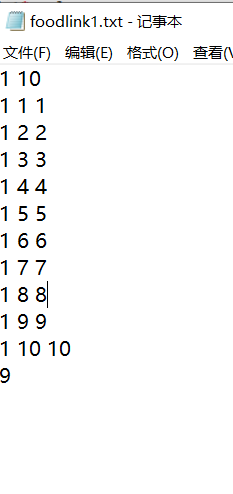


图2.9：测试样例1

测试结果如图2.10所示：

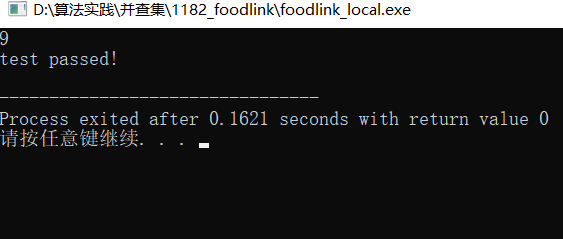


图2.10 测试样例1 的测试结果

说明：这是本地测试的样例，为了方便测试采用了文件读写的方式来输入测试数据，测试样例的最后一行为正确答案，得到正确答案后会输出tess passed！字样，如果错误会输出tess failed！ answer：xxx your answer：xxx的提示。测试输出第一行为程序的结果，第二行是提示信息。

样例分析：由于第一个数字指定的动物只有1种，所以从第二句话到第九句话都是错误的，所以结果应该为9.

测试样例2：如图2.11所示：

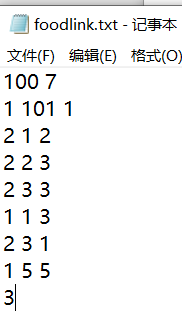


图2.11：测试样例2

测试样例2的测试结果如图2.12所示：

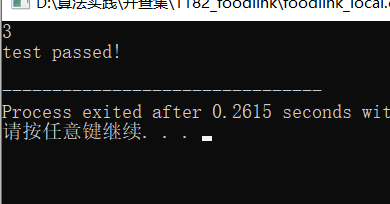


图2.12：测试样例2 的测试结果

样例分析：

错误1：1 101 1 101超过范围

错误2：2 3 3 自己吃自己

错误3：1 1 3：由 2 1 2和 2 2 3可以知道3吃1。

### 总结

工作情况：

本次总共在POJ上完成了18道题目，每个专题都至少完成了2道。不够完善的地方是感觉整个课程时间比较短，题目的难度稍微有点偏高，最开始的时候想着完全靠自己思考把这些题目写完，发现时间不够。所以独立思考的时间不是很够。最开始时尝试自己寻找思路，往往自己想到一个思路，并觉得似乎是正确。但是继续往下就会发现走不通。这时候就觉得有点难受，虽然想到一个思路的感觉还是蛮好的。

心得体会：

上完这门课之后觉得这是一门非常好的课，很实在，干货十足。首先是见识到了很多新奇并且有趣的数据结构，这些数据结构和平时课堂上学习的数据结构不太一样。他们往往是针对一些特殊问题所提出来的。针对这些数据结构的操作往往有一些非常新奇的效果。第二是熟悉了算法和加强了动手。通过题目的训练，进一步的熟悉了一些经典的算法以及它们的实现，体会到了算法在减少运算量，减小时间复杂度和空间复杂度方面的作用。通过实际的去实现这些算法和数据结构，加强了动手能力。第三，最后团队合作完成一个ppt并且选派两人讲解，不仅锻炼了我的团队合作能力，一个团队里面一起做事情就应该相互多交流，才能更好的完成工作。而且上去讲解也是一个比较锻炼胆量的事情。

意见和建议：

我觉得这门课是一门非常好的课，学完之后感受收获了很多，强烈建议全年级推广。

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |

|  |
| --- |
| 二、对课程设计的学术评语（教师填写） |
|  |
| 三、对课程设计的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |