深 圳 大 学 实 验 报 告

课程名称 算法设计与分析

项目名称 图论--桥

学　　院 计算机与软件学院

专　　业 软件工程

指导教师 卢亚辉

报 告 人 郑彦薇 学号 2020151022

实验时间 2022/5/31~2022/6/7

提交时间 2022/6/4

教务处制

一、实验目的与要求

1. 掌握图的连通性。
2. 掌握并查集的基本原理和应用。

二、实验内容与方法

1. 桥的定义

在图论中，一条边被称为“桥”代表这条边一旦被删除，这张图的连通块数量会增加。等价地说，一条边是一座桥当且仅当这条边不在任何环上。一张图可以有零或多座桥。

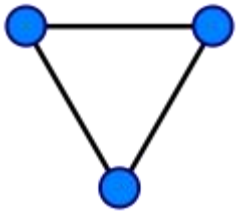


图 1 没有桥的无向连通图

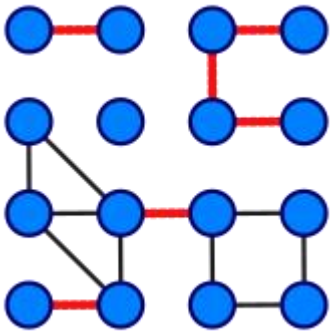


图 2 这是有 16 个顶点和 6 个桥的图（桥以红色线段标示）

2. 求解问题

找出一个无向图中所有的桥。

3. 算法

（1）基准算法

For every edge (u, v), do following

a) Remove (u, v) from graph

b) See if the graph remains connected (We can either use BFS or DFS)

c) Add (u, v) back to the graph.

（2）应用并查集设计一个比基准算法更高效的算法。不要使用 Tarjan 算法，如果使用 Tarjan 算法，仍然需要利用并查集设计一个比基准算法更高效的算法。

三、实验步骤与过程

（一）基准法

1. 解题思路
   1) 计算图的连通分量
   2) 删除边并统计当前图的连通分量
   3) 复原
   4) 判断是否是桥，如果是，桥的数量加 1

2. 采用的数据结构

在该方法中，使用邻接表存储图信息。原因是实验数据多为稀疏数据，如果采用邻接矩阵，则会得到稀疏矩阵，在大数据情况下是非常消耗空间的；另外，对于 DFS 而言，使用邻接表存储时，时间复杂度为 $O(E+V)$，而使用邻接矩阵存储，DFS 的时间复杂度为 $O(v^2)$。因此选择邻接表进行存储，在编程实现中，使用 vector 存储顶点以及相邻点信息（即边信息）。

3. 伪代码

根据思路，编写每一个模块功能的伪代码如下：

算法 1 基准法计算桥的数量

输入: $vexnum$ 顶点数，$arcnum$ 边数，$edge$ 边信息

输出: $Count$ 桥的数量

```
1: function INIT
2:     for i = 0 → vexnum do
3:         visit[i] ← false
4:     end for
5: end function
6:
7: function DFS(v)
8:     visit[i] ← true
9:     for i = 0 → Node[v].size do
10:        if visit[Node[v][i]] == false and !(v == Delstartand Node[v][i] == Delend) and !(v ==
       Delendand Node[v][i] == Delstart) then
11:            DFS(Node[v][i])
12:        end if
13:    end for
14: end function
15:
16: function DFSTraverse
17:    ans ← 0
18:    INIT
19:    for i = 0 → vexnum do
20:        if visit[Node[v][i]] == false then
21:            DFS(i)
22:            ans++
23:        end if
24:    end for
25: end function
26:
27: function DFSCount
28:    ansbefore ← DFSTraverse
29:    for i = 0 → arcnum do
30:        Delstart ← egde[i][0]
31:        Delend ← edge[i][1]
32:        ansafter ← DFSTraverse
33:        if ans_after > ans_before then
34:            Count++
35:        end if
36:    end for
37: end function
```

4. 算法正确性验证

使用图 2 验证算法正确性，在该图中，有 16 个顶点、15 条边、6 条桥。根据信息创建相应的测试文本文档 test.txt，得到运行结果如下所示：

```
该图中桥的数量为：6
the time cost is: 8ms
```

算法正确性得以验证。

5. 性能分析

读取给定文本文档中的数据，统计实际运行所用时间，得到结果如下：

运行数据	时间统计
mediumDG	8ms
largeG	?

可以看到基准法并不能够在有限时间内解决大数据。

（二）基准+并查集

1. 解题思路

该方法与基准法基本相同。在该方法中，引入对并查集的应用，通过删除边并计算连通分枝数判断桥，即在计算连通分支时使用并查集而不是 DFS。

并查集统计连通分量具体思路：

1) 添加全局变量 father 用于记录顶点所在集合，访问边信息，对连通的顶点进行合并
2) 若该边顶点合并后集合数量发生变化，说明该边为桥，桥的数量加 1

2. 伪代码

根据思路，编写每一个模块功能的伪代码如下：

算法 2 基准法 + 并查集计算桥的数量

输入: $vexnum$ 顶点数，$arcnum$ 边数，$edge$ 边信息

输出: $Count$ 桥的数量

```
function JoinSetInit
    for i = 0 → vexnum do
        father[i] ← i
    end for
end function

function JoinSetFind(v)
    if father[v] == v then
        return v
    end if
    father[v] ← JoinSetFind(father[v])
    return father[v]
end function

function JoinSetCount
    ansbefore ← DFSTraverse
    for i = 0 → arcnum do
        INIT
        JoinSetInit
        res ← vexnum
        for j = 0 → arcnum do
            if i == j then
                continue
            end if
            f1 ← JoinSetFind(edge[j][0])
            f2 ← JoinSetFind(edge[j][1])
            if f1! = f2 then
                father[f2] ← f1
                res − −
            end if
        end for
        if res! = ansbefore then
            Count + +
        end if
    end for
end function
```

3. 算法正确性验证

   使用已创建的 test 文本文档对算法正确性进行验证。得到运行结果如下：

```
该图中桥的数量为: 6
the time cost is: 1ms
```

   算法正确性得以验证。

4. 性能分析

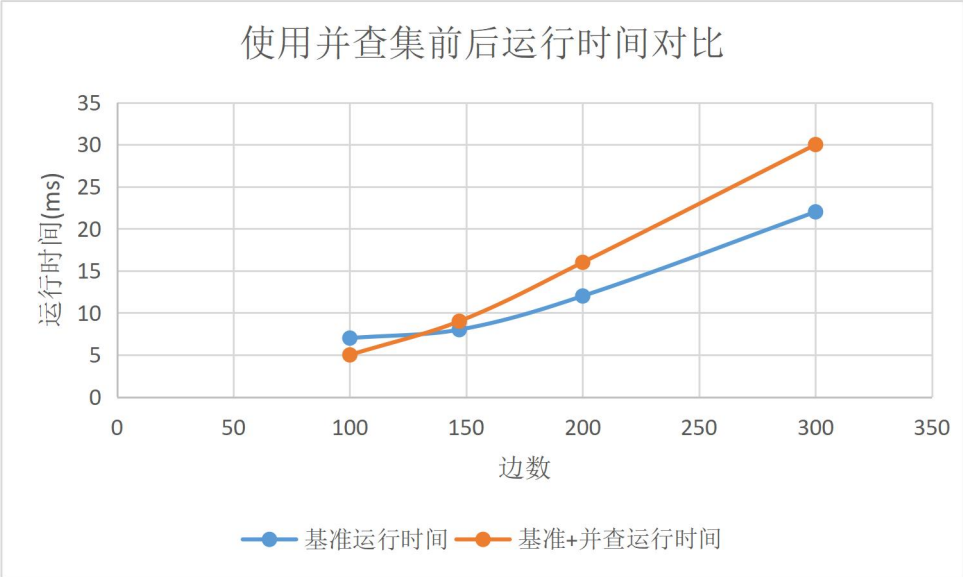
   读取给定文本文档中的数据，统计实际运行所用时间，得到结果如下：

运行数据	时间统计
mediumDG	10ms
largeG	?

（三）对上述两种方法性能进行分析

随机生成指定顶点数和边数的图数据，对引入并查集前后基准算法统计不同规模图的桥数量时的实际运行时间进行对比。

运行数据	基准法运行时间	基准+并查运行时间
mediumDG(50-147)	8ms	9ms
test50-100	7ms	5ms
test50-200	12ms	16ms
test50-300	22ms	30ms



根据上述数据展示可以看到，引入并查集前后，算法的实际运行效率相差并不大，仅仅只有几毫秒之差。因此我们知道，在基准法的基础上简单引入对并查集的应用，不能够真正提高算法效率，以处理大规模数据。因此，需要引入更高效的方法解决问题。

（四）LCA

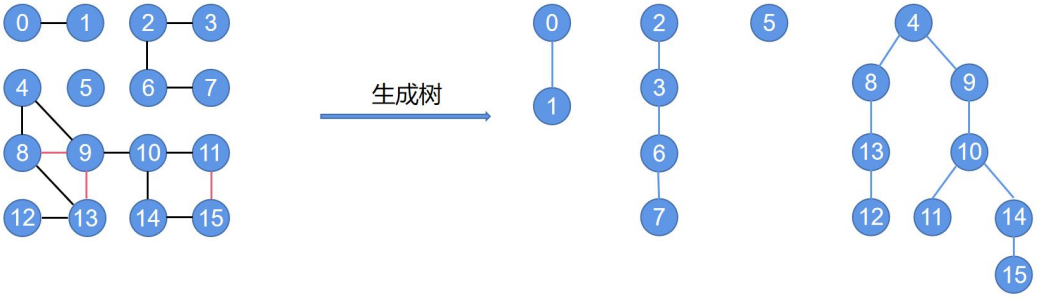
1. 解题思路

根据桥的等价定义“一条边是一座桥当且仅当这条边不在任何环上”，引入 LCA 算法解决问题。

LCA 算法具体思路：

1) 逆向思维：根据桥的等价定义，我们只需将图中所有环边找出来，那么剩下的边就是桥
2) 最近公共祖先概念引入：两个点能找到的最近的公共祖先节点
3) 寻找环边：使用 BFS 建立生成树，对于图中不在生成树上的边，我们称之为“重边”。对于重边上的两点 v1、v2，找到 v1、v2 的最近公共祖先，再从 v1、v2 出发，找到他们的最近公共祖先的路径上的边，将这些边标记为环边
4) 根据图中所有非环边的边为桥，将上述所找环边进行排除即可得到桥

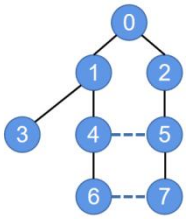
用图 2 进一步解释说明该逆向思维：



根据上图可以看到有三条非生成树边即重边：（8，9）、（9，13）、（11，15）；分别寻找这三条重边的两个顶点的最近公共祖先：4、4、10；再从两个顶点出发，找到它们到最近公共祖先路径上的边：对于边（8，9），8 和 9 到最近公共祖先 4 的路径为 8→4、9→4，同理可以对其他两条重边的顶点到最近公共祖先的路径的边进行统计，这些边都为环边；最后可以从所有边中排除环边，得到图中的非环边有：（0，1）、（2，3）、（2，6）、（2，7）、（9，10）、（12，13）共六条边，正对应着图中的六条桥。

对路径压缩的解释：

例如下面的生成树中，边（4，5）和边（6，7）为重边。边（4，5）中两个顶点寻找 lca 的两条路径为：4→1→0 以及 5→2→0；边（6，7）中两个顶点寻找 lca 的两条路径为：6→4→1→0 以及 7→5→2→0；可以看到上述路径均有重复的部分，路径压缩则是消除这样的重复。



路径压缩的方法是将路径上所有节点都挂到 lca 上，表示该点到 lca 的路径已经走过，如下图所示。

可以看到压缩后，顶点 7 到 lca 的路径由 7→5→2→0 缩短至 7→5→0，即 5 到 0 的路径被压缩。

2. 采用的数据结构

在该方法中，同样使用邻接表存储图的信息。但在该方法下，程序使用 BFS 完成树的生成等工作，原因是处理大数据时，如果使用 DFS 可能会出现栈溢出的现象，程序无法正常运行。

3. 伪代码

根据思路，编写每一个模块功能的伪代码如下：

算法 3 LCA 计算桥的数量

输入： $vexnum$ 顶点数，$arcnum$ 边数，$edge$ 边信息

输出： $Count$ 桥的数量

```
function LCA
    if x == father[y]ory == father[x] then
        return
    end if
    if depth[x] < depth[y] then
        x ← x^y
        y ← y^x
        x ← x^y
    end if
    while depth[x] > depth[y] do
        Loop[x] ← 1
        x ← set[x]
    end while
    FindLCA
    pathcompression                                  ▷ 路径压缩
end function

function LcaCount
    BFSTraverse                                      ▷ 通过 BFS 建立生成树
    for xinBFSTree do
        for yinNode[x] do
            LCA(x, y)
        end for
    end for
end function
```

4. 算法正确性验证

使用已创建的 test 文本文档对算法正确性进行验证。得到运行结果如下：

```
数据载入...
载入完成...
该图中桥的数量为：6
the time cost is: 0ms
```

算法正确性得以验证。

5. 性能分析

读取给定文本文档中的数据，统计实际运行所用时间，得到结果如下：

运行数据	运行时间
mediumDG	6ms (1000 times)
largeG	3677ms

可以看到使用 lca 算法统计给定图的桥数量时，运行效率大大提高。

四、实验结论或体会

该实验是对图中的桥的数量进行统计。通过基准法、基准法中引入并查集、LCA 方法三种思路对数据进行处理，学习和掌握了对桥数量统计的不同思路。对三种方法的运行时间进行对比，可以知道 lca 方法在处理该问题时具备高效性。同时了解了单纯引入对并查集的使用并不能真正实现高效，需要在基准法的基础上转变处理图数据的思路，如采用逆向思维，通过寻找环边反过来得到桥的数量。我们在解决其他问题时，也应学会转变思路，以更好的实现实验目的。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。