深 圳 大 学 实 验 报 告

课程名称：　　算法设计与分析

实验项目名称：　　实验六 桥

学院：　　计算机与软件学院

专业：　　计算机科学与技术

指导教师：　　杨烜

报告人：沈晨玙　学号：2019092121　班级：19 计科国际

实验时间：　　2021.5.25

实验报告提交时间：　　2021.5.25

教务部制

实验六 桥

一、实验目的：

（1）掌握图的连通性。
（2）掌握并查集的基本原理和应用。

二、内容：

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。

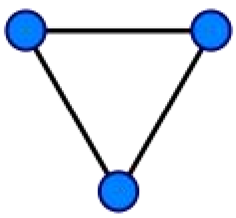


图 1 没有桥的无向连通图

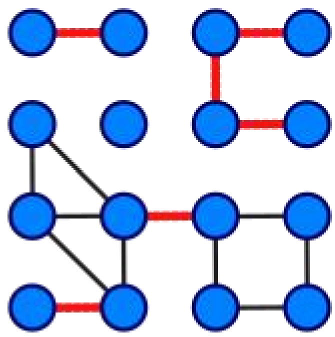


图 2 这是有 16 个顶点和 6 个桥的图（桥以红色线段标示）

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、实验要求

1. 实现上述基准算法。
2. 设计的高效算法中必须使用并查集，如有需要，可以配合使用其他任何数据结构。
3. 用图 2 的例子验证算法正确性。
4. 使用文件 mediumG.txt 和 largeG.txt 中的无向图测试基准算法和高效算法的性能，记录两个算法的运行时间。
5. 设计的高效算法的运行时间作为评分标准之一。
6. 提交程序源代码。
7. 实验报告中要详细描述算法设计的思想，核心步骤，使用的数据结构。

四、实验过程及结果

一：基准法求解

1. 算法原理
   (1) 由桥的定义可知，删除桥之后，图的连通分支数量就会增加。所有只需要比较删除前后连通分支数量即可判断该边是否为桥。
   (2) 利用 DFS 算法计算连通分支数量。
   (3) 计算连通分支数量方法：
      ① 首先创建长度大小等于节点个数的访问数组，并对每个元素初始化为 false，然后对每个节点进行以下操作。
      ② 若当前节点已经被访问过，则遍历下一个节点
      ③ 若当前节点没被访问，连通分量个数加 1，同时对该节点进行 DFS 遍历，将遍历过程中的点对应访问元素的值设置成 1

2. 伪代码

```
BASIC：
    A = Count_connected_components
    For every edge (u, v):
        Remove (u, v) from graph
        B = Count_connected_components
        If B > A
            (u,v) is a bridge
        Add (u, v) back to the graph.
```

```
Count_connected_components:
    Count = 0
    For every vertex (v):
        If visited[v] = false:
            DFS (v)
            Count++
    return Count
```

```
DFS(v):
    visited[v] = true
    for every adjacent points (next) of (v):
        if visited[next] = false:
            DFS (next)
```

3. 时间复杂度分析

对全图做 DFS 需要 $O(n+e)$

有 e 条边，需要对全图做 DFS，总时间复杂度为 $O(en+e^2)$

对于稀疏图 e = n，对于稠密图 e = n^2

稀疏图：$O(n^2)$，稠密图：$O(n^4)$

4. 优化模型

(1) 算法原理：

① 在基准法的基础上，删除边后不需要对全图进行 DFS。

② 对删除边的其中一个结点做 DFS，如果在深度优先搜索过程中，搜索到另一结点则说明在同一连通分支，提前结束搜索。

(2) 伪代码：

```
IMPROVED_BASIC：
    For every edge (u, v):
        Remove (u, v) from graph
        DFS (u)
        If v is visited during DFS
            (u,v) is a bridge
        Add (u, v) back to the graph.
```

(3) 时间复杂度分析：

有 e 条边，需要做 1 次 DFS，总时间复杂度上限为 $O(en+e^2)$

对于稀疏图 e = n，对于稠密图 e = n^2

稀疏图：$O(n^2)$，稠密图：$O(n^4)$

(4) 优化分析：

图越稀疏时，连通分支增多，优化后只需要对一个连通分支进行 DFS，可以极大程度上增快搜索速度。

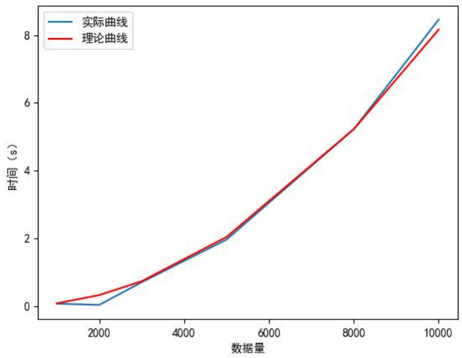
对于 10000 个节点，对比算法效率。

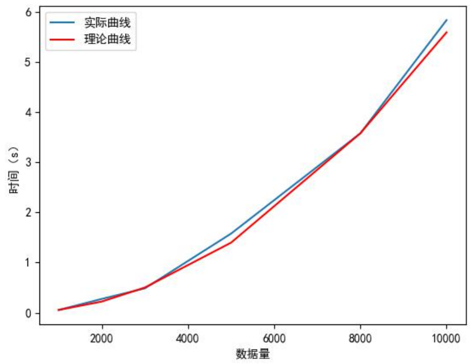
V : E	1 ： 1	1.2 : 1	1.4 : 1	2 : 1
优化前（s）	8.457	5.915	4.043	2.32
优化后（s）	5.833	3.253	1.453	0.412
提升效率	31%	45%	64%	82%

5. 数据分析

稀疏图：$e = n$　　时间复杂度：$O(n^2)$

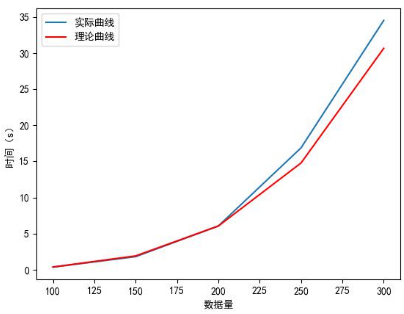
节点个数	1000	2000	3000	5000	8000	10000
优化前运行时间(s)	0.074	0.0361	0.706	1.965	5.224	8.457
优化后运行时间(s)	0.056	0.276	0.487	1.578	3.577	5.833

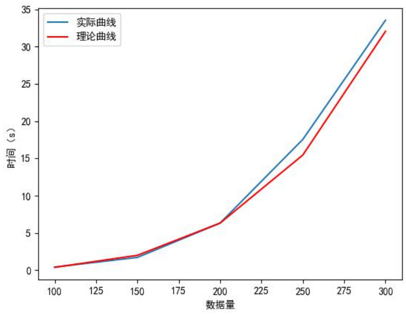




稠密图：$e = n^2$　　时间复杂度：$O(n^4)$

节点个数	100	150	200	250	300
优化前运行时间(s)	0.36	1.81	6.059	16.888	34.542
优化后运行时间(s)	0.428	1.726	6.333	17.556	33.553





二：并查集求解

1. 算法原理

(1) 主要思路与基准法类似，通过比较删除边前后连通分支数来判断是否为桥。差别在于对于连通分支数的计算利用并查集。

(2) 连通分支数计算方法：初始状态下，图中有 v 个连通分支。遍历所有边，利用并查集，如果两节点所在集合不属于同一集合，则将两个集合合并，连通分支数减一。

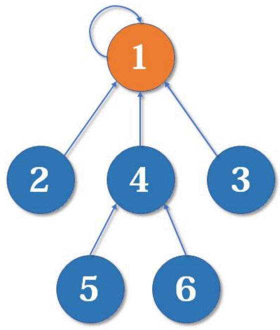
*并查集　　*图源自：https://zhuanlan.zhihu.com/p/93647900*

A. 并查集原理

① 并查集的重要思想在于，用集合中的一个元素代表集合。

② 这是一个树状的结构，要寻找集合的代表元素，只需要一层一层往上访问父节

点（图中箭头所指的圆），直达树的根节点（图中橙色的圆）即可。根节点的父节点是它自己。我们可以直接把它画成一棵树



B. 伪代码

初始化：

Init()

```
    for i = 0 to n
        parent[i] = i;
```

查询

Get_Parent(x)

```
    if(parent[x] == x)
        return x;
    else
        return Get_Parent(parent[x]);
```

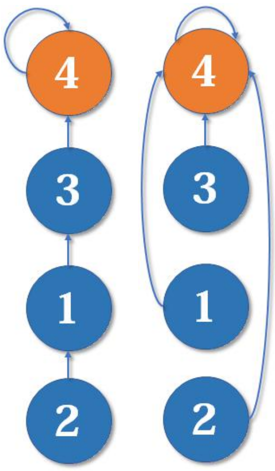
合并

Union(i, j)

```
    parent[Get_Parent(i)] = Get_Parent(j);
```

C. 优化方案

a. 路径压缩



随着链越来越长，我们想要从底部找到根节点会变得越来越难。可以使用路径压缩的方法。既然只关心一个元素对应的根节点，那就将每个元素直接指向根节点。

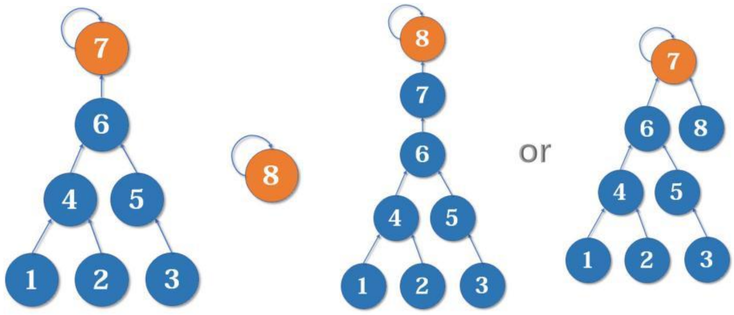
实现的方式是在查询的过程中，把沿途的每个节点的父节点都设为根节点即可。下一次再查询时，就可以省很多事。

Get_Parent(x)

```
    If (x == parent[x])
        return x;
    else
        parent[x] = Get_Parent(parent[x]);//父节点设为根节点
    return parent[x];                  //返回父节点
```

b. 平衡处理

由于路径压缩只在查询时进行，也只压缩一条路径，所以并查集最终的结构仍然可能是比较复杂的。



显然应该把简单的树往复杂的树上合并，而不是相反。因为这样合并后，到根节点距离变长的节点个数比较少。

实现方法是用一个数组 rank[]记录每个根节点对应的树的深度(如果不是根节点，其 rank 相当于以它作为根节点的子树的深度)。一开始，把所有元素的 rank 设为 1。合并时比较两个根节点，把 rank 较小者往较大者上合并。

```
Union(i, j)
    x = Get_Parent(i), y = Get_Parent(j);       //先找到两个根节点
    if (rank[x] <= rank[y])
        parent[x] = y;
    else
        parent[y] = x;
    if (rank[x] == rank[y] && x != y)
        rank[y]++;                    //如果深度相同且根节点不同，则新的根节点的深度+1
```

2. 伪代码

```
BASIC_Union：
    A = Count_connected_components
    B = v
    For every edge (e1, e2):
        Remove (e1, e2) from graph
        For every edge (e3, e4):
            Union (e3, e4)                # if parent[e3] = parent[e4], B = B - 1
        If B > A
            (e1, e2) is a bridge
        Add (e1, e2) back to the graph.
```

3. 复杂度分析

初始化为求连通分支数，时间复杂度 $O(e^2)$

外层循环需要遍历 e 次删除情况

内层循环中每条边需要做 2 次 Get_Parent 操作，每次时间复杂度为 $O(\alpha(n))$

时间复杂度表格

当并查集中的元素个数为 n 时，下面的表格给出了单次并查集操作的时间复杂度：

优化	平均时间复杂度	最坏时间复杂度
无优化	$O(\log n)$	$O(n)$
路径压缩	$O(\alpha(n))$	$O(\log n)$
按秩合并	$O(\log n)$	$O(\log n)$
路径压缩 + 按秩合并	$O(\alpha(n))$	$O(\alpha(n))$

这里 α 表示阿克曼函数的反函数，在宇宙可观测的 n 内（例如宇宙中包含的粒子总数），$\alpha(n)$ 不会超过 5。

循环总时间复杂度为 $\mathrm{O}\,(e*(e*2*\alpha(n))) = O(e^2)$

总体时间复杂度为 $\mathrm{O}(e^2)$

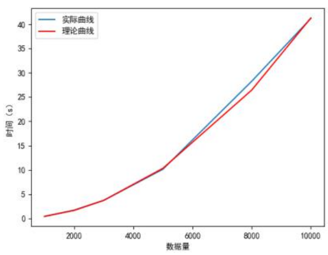
对于稀疏图 e = n，对于稠密图 $e = n^2$

稀疏图：$\mathrm{O}(n^2)$，稠密图：$\mathrm{O}(n^4)$

4. 数据分析

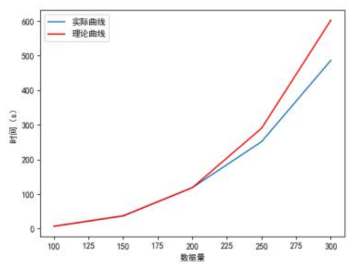
稀疏图：e = n　　时间复杂度：$\mathrm{O}(n^2)$

节点个数	1000	2000	3000	5000	8000	10000
运行时间(s)	0.413	1.704	3.718	10.132	28.262	41.22



稠密图：$e = n^2$　　时间复杂度：$\mathrm{O}(n^4)$

节点个数	100	150	200	250	300
运行时间(s)	7.217	36.959	119.089	251.919	486.795



二：生成森林优化基准法

1. 算法原理

   (1) 利用 DFS 算法，获得图的一个生成森林。

   (2) 因为桥边是连接两个连通分支的唯一边，所以桥边一定会出现在原图的生成森林之中。只需要遍历生成森林上的边，就可以找到所有桥边。

   (3) 生成生成森林的方法是对全图进行 DFS，如果前后两节点都没有被搜索到过，说明是一条生成树边并进行标记。

2. 伪代码

MST_Create：

```
Init visited[]
For i = 1 to v:
    if (visited[i] == false)
        MST_DFS(i, -1);
```

MST_DFS (cur, pre):

```
If (pre != -1)
    Edge (cur, pre) is a bridge
visited[v] = true
for every adjacent points (next) of (v):
    if visited[next] = false:
        MST_DFS (next, cur)
```

3. 时间复杂度分析

大体上与基准法类似，区别在于只需要确认生成树上的边是否为桥。

对全图做 DFS 需要 O（n+e），生成树至多有有 n-1 条边，需要做一次 DFS，总时间复杂度为$\mathrm{O}(\mathrm{n}*(\mathrm{n}+\mathrm{e}))=\mathrm{O}(\mathrm{n}^2+ne)$。（DFS 优化算法）

初始化为求连通分支数，时间复杂度$O(e^2)$，外层循环需要遍历之多 n-1 次删除情况

内层循环中每条边需要做 2 次 Get_Parent 操作，每次时间复杂度为$\mathrm{O}(\alpha(n))$,循环总时间复杂度为$\mathrm{O}(e*(\mathrm{n}*2*\alpha(n)))=O(ne)$。（并查集算法）

对于稀疏图 e = n，对于稠密图 e = n^2

稀疏图：$O(\mathrm{n}^2)$，稠密图：$O(\mathrm{n}^3)$

4. 数据分析

对于 10000 个节点，对比 DFS/并查集加入生成树后的算法效率。

DFS+生成树算法

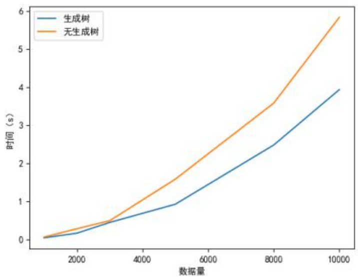
V : E	1 ： 1	1 ： 2	1 ： 5	1 ： 10
提升效率	17%	54%	79%	83%

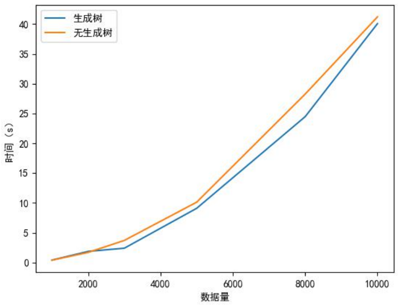
并查集+生成树算法

V : E	1 ： 1	1 ： 2	1 ： 5	1 ： 10
提升效率	23%	50%	82%	86%

稀疏图：e = n　　时间复杂度 O（n^2）

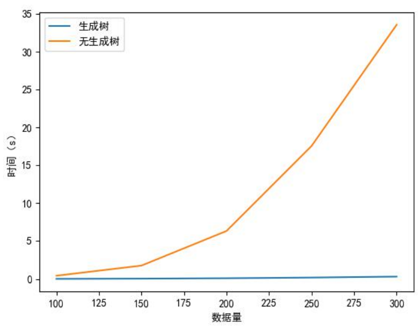
节点个数	1000	2000	3000	5000	8000	10000
DFS 运行时间(s)	0.037	0.159	0.441	0.92	2.476	3.932
并查集运行时间(s)	0.426	1.728	3.818	12.458	26.89	41.62

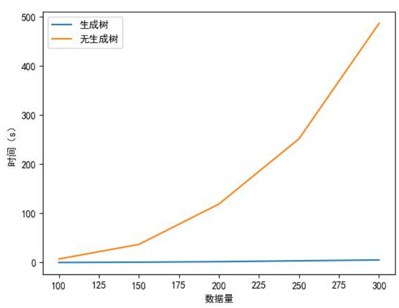




稠密图：e = n^2　　时间复杂度 O（n^3）

节点个数	100	150	200	250	300
DFS 运行时间(s)	0.012	0.04	0.095	0.184	0.31
并查集运行时间(s)	0.128	0.763	1.768	3.568	5.22





三：LCA 算法求解

1. 算法原理

(1) 由桥的定义，“一条边是一座桥当且仅当这条边不在任何环上”。桥边的计算可以通过排除法得到，即‘总边数 - 环边数’。首先非生成树边一定是环边（该边连

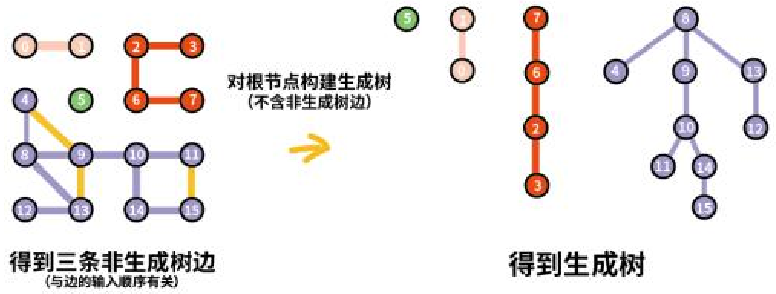
接的两点之间存在不唯一路径，否则该边一定在生成树中）。所以问题就转化成了求生成树边中的环边。

(2) 实现：首先通过 DFS 获得生成森林。依次将非生成树边加入生成树之中，如果新增了环，则将未标记的环进行记录。最后得到环边总数。桥数 = 总边数 - 环边数。

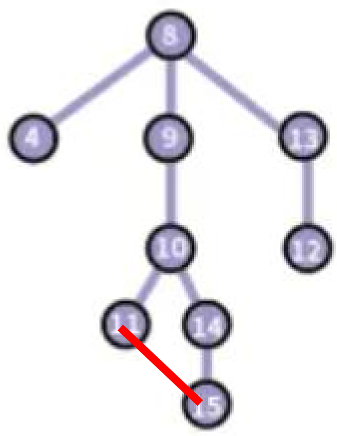
(3) 此处引入 LCA 算法用于求生成树中的环边数。

*** LCA 算法** *图源自：https://www.codenong.com/cs106674857/*

1. LCA 算法用于求两结点的共同祖先，如果两个结点拥有相同的祖先，则必能形成闭环。



2. 得到生成树之后，我们需要将每条非生成树边依次加入生成树。对于这条边的两个节点，寻找它们的最小公共祖先节点，并且将寻找过程中遇到的边记录下来。下面我们拿（11，15）这条边进行解释。



显然 10-11-14-15 可以形成闭环，在这颗生成树中 11、15 的公共祖先为 10，向上遍历的过程中 11-10，10-14，14-15 均被标记为环边。对于其他非生成树边同理。

3. LCA 具体步骤：（以边 11-15 为例）

(1) 获得两节点的深度，11：4，15：5。首先需要对节点 15 沿着父亲节点遍历，直到其高度为 3（与 11 的高度一样）。于是我们首先找到 14-15 边。

(2) 此时节点 11 和节点 14 的高度都为 3，然后让它们同时寻找父亲节点 10，记录下来 10-11 边和 10-14 边。

(3) 找到共同父亲节点后结束搜索。

最终找到三条环边。

4. 部分数据结构解析：

(1) Depth[]：记录节点在生成树中的深度

(2) Parent[]：记录父节点

(3) Edges_visited[]：因为生成树中的每一条边都有且仅有一个父节点，所以可以以子节点序号作为边数组的索引（例如 Edges[15]代表边 14-15）

2. 伪代码

```
LCA():
    // 以 DFS 为模板，构建生成树
    For i = 1 to v:
        If visited[i] = false:
            LCA_DFS (i, -1, 0)
    // 检查非生成树边
    Lca_num = 0
    For every edge (l, r) not in MST:
        Find_Lca (l, r, lca_num)
    Return Edges_Sum - Edges_Not_In_MST - Lca_num

LCA_DFS(cur, pre, dep):
    If pre != -1
        Edge (cur, pre) is in MST
    Visited[cur] = true
    Parent[cur] = pre
    Depth[cur] = dep
    for every adjacent points (next) of (v):
        if visited[next] = false:
            LCA_DFS (next, cur, dep + 1)


Find_Lca (l, r):
    left_dep = Depth[l], right_dep = Depth[r]
    // 右边深度大于左边深度
    If left_dep < right_dep:
        delta = right_dep - left_dep
        // 向上搜索到同一高度
        While delta >= 0:
            If Edges_visited[r] == false:
                Edges_visited[r] = True
                Lca_num++
            r = Parent[r]
        // 同时向上搜索
        while l != r:
            if Edge_visited[right] == false
                Edge_visited[right] = true
```

```
                Lca_num++
            if Edge_visited[l] == false:
                Edge_visited[l] = true
                Lca_num++
            l = parent[l]
            r = parent[r]
    Else if left_dep > right_dep:
        ...
    Else if left_dep == right_dep:
        ...
```

3. 时间复杂度分析

DFS+并查集的时间复杂度为 $O(n+e)$

依次加入找到的非生成树边进行 LCA 算法，其中 LCA 算法与并查集的搜索类似，因此查询时间复杂度为 $O(e*\log(n))$

算法复杂度为 $O(n+e+e\log(n))$

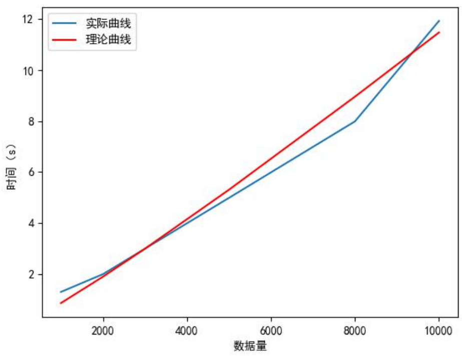
对于稀疏图 e = n，对于稠密图 $e = n^2$

稀疏图：$O(n\log(n))$，稠密图：$O(n^2\log(n))$

4. 数据分析

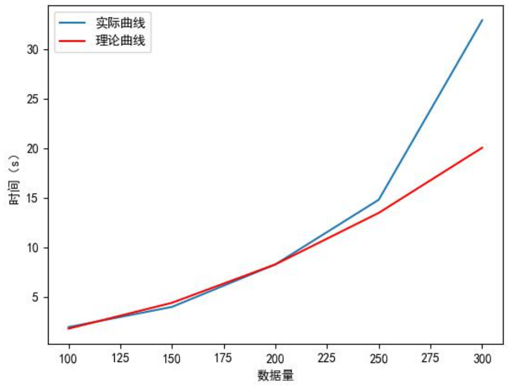
稀疏图：e = n　　时间复杂度：$O(n\log(n))$

节点个数	1000	2000	3000	5000	8000	10000
运行时间(ms)	0.997	1.994	2.992	4.986	7.987	11.927



稠密图：$e = n^2$　　时间复杂度：$O(n^2\log(n))$

节点个数	100	150	200	250	300
运行时间(s)	1.953	3.989	8.276	14.8	32.905



MediumDG.txt 数据集：0s

LargeG 数据集：5.489s

五、经验总结

本次实验完成了图论中对于桥边的搜索。其中不同算法对于稀疏图、稠密图有着不同程度的优化，经过测试后可以发现图的稠密程度会有较大影响。

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。