**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与实现**

**实验项目名称： 无向图求桥**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 杨烜老师**

**报告人： 李若龙 学号：2018171028 班级： 计科02**

**实验时间： 2020/6/1**

**实验报告提交时间：**

**教务处制**

1. 问题描述：

**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。

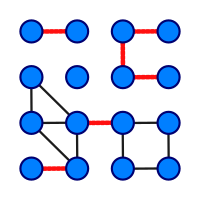
 

图 1 没有桥的无向连通图 图 2 这是有16个顶点和6个桥的图

（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.

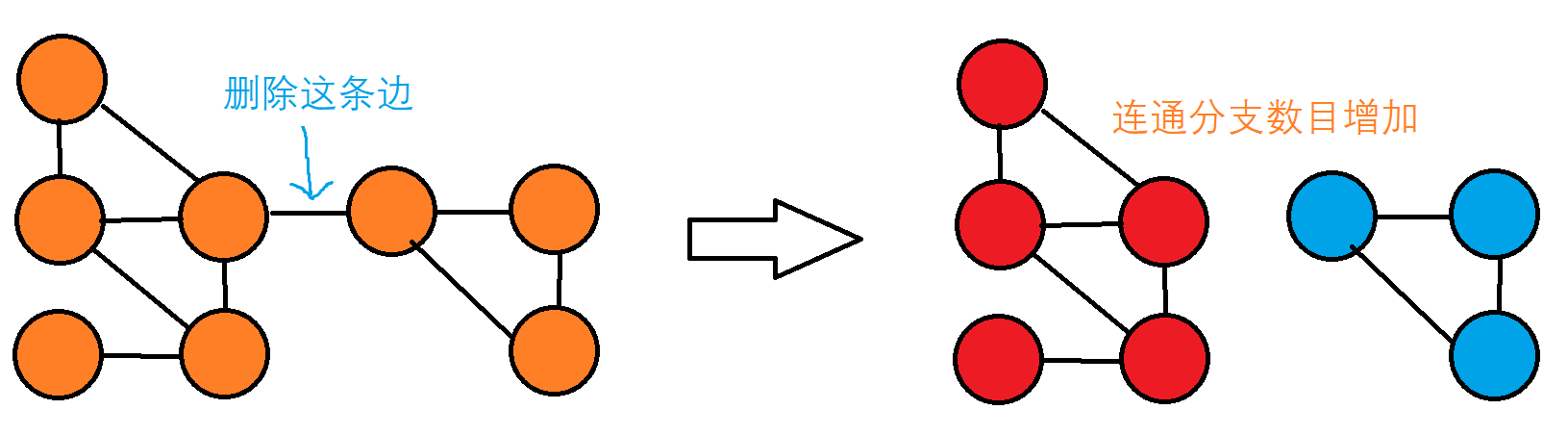
（2）应用并查集设计一个比基准算法更高效的算法。不要使用Tarjan算法，如果使用Tarjan算法，仍然需要利用并查集设计一个比基准算法更高效的算法。

求解问题的算法原理描述

1. 基准法
2. 基准法+并查集
3. 基准法+生成树优化
4. 基准法+并查集+生成树优化
5. lca求环边求桥
6. Tarjan算法直接求桥

**基准算法**

穷举边并且删除这条边，然后计算连通分支数目，使用dfs计算连通分支数目



**伪代码描述：**



复杂度分析：其中 n为顶点数，e为边数

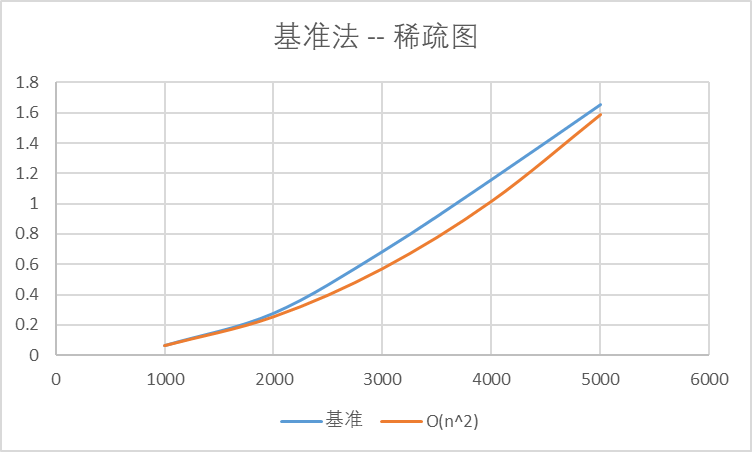
穷举删除的边需要e次，每次删除都要dfs判断连通分支数目，需要O(n+e)，复杂度O(e)

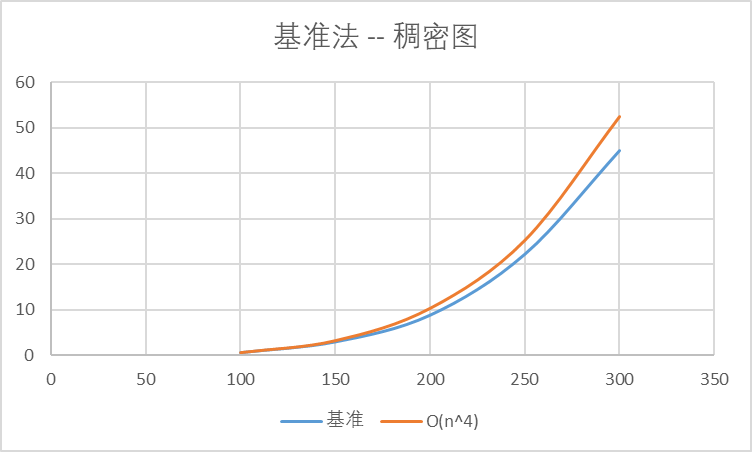
对于稀疏图（e=n），复杂度为O(n2)

对于稠密图(e=n2)，复杂度为O(n4)



分别使用稀疏图和稠密图测量其运行时间，其中横坐标是顶点数n，纵坐标是运行时间





可以看到稀疏图中，基准法符合O(n2)的复杂度，而稠密图中，基准法符合O(n4)的复杂度

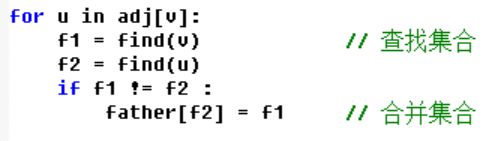
**基准法+并查集**

和基准法思路相同，通过删除边并计算连通分支数目来查找桥，只是计算连通分支时使用并查集而不是dfs

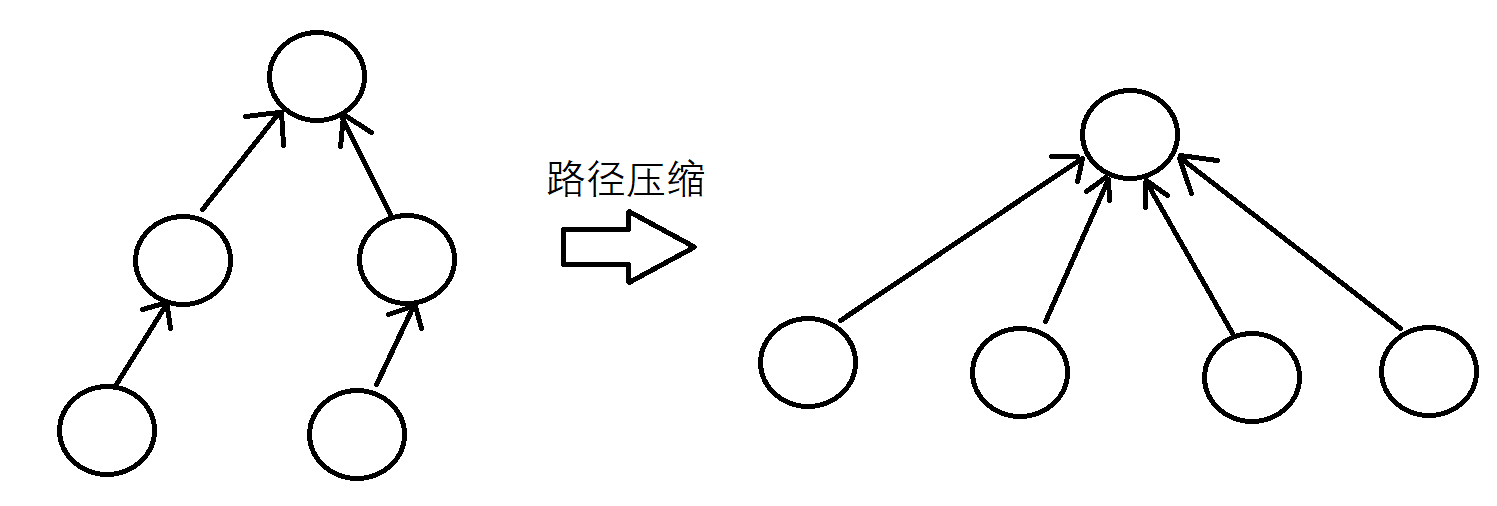
**并查集计算连通分支数目：**

枚举边，对每个边上的两点v1和v2，查询v1和v2所属的集合f1,f2，如果v1和v2不在同一个集合则合并v1和v2所属的两个集合，最后统计集合的个数，即为连通分支数目

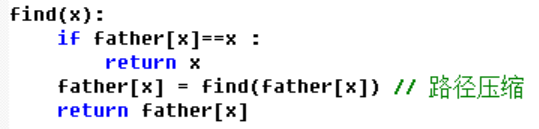
**并查集伪代码描述：**



使用路径压缩策略，使得并查集的查询复杂度均摊下来为O(1)



**路径压缩伪代码描述：**

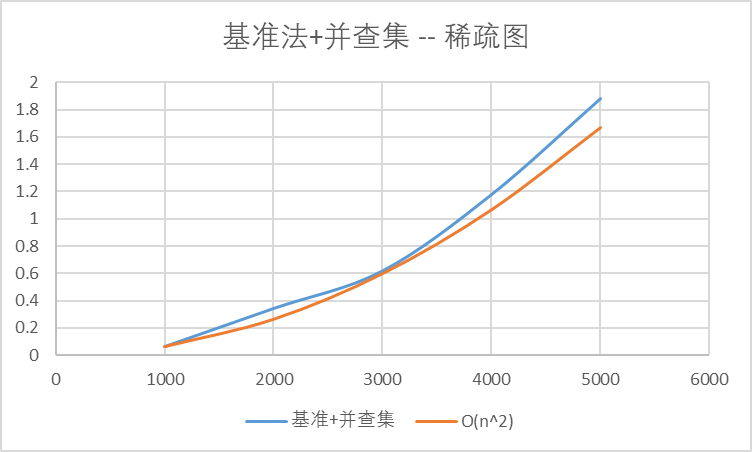


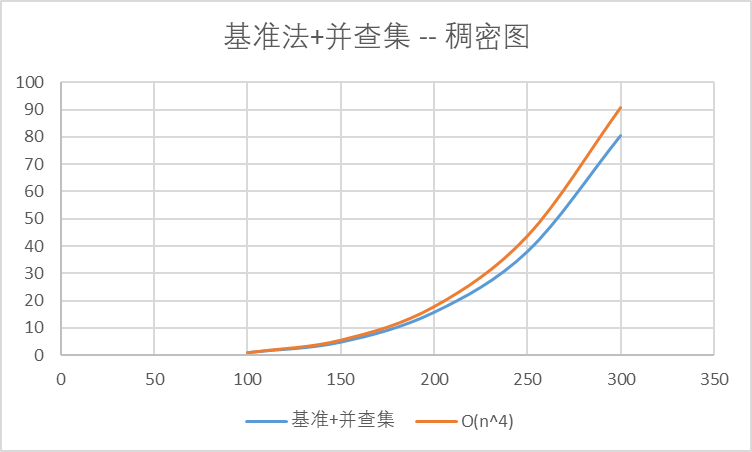
复杂度分析：其中 n为顶点数，e为边数

穷举删除的边需要e次，每次删除都用并查集判断连通分支数目，需要O(e)，复杂度O(e)

对于稀疏图（e=n），复杂度为O(n2)，对于稠密图(e=n2)，复杂度为O(n4)



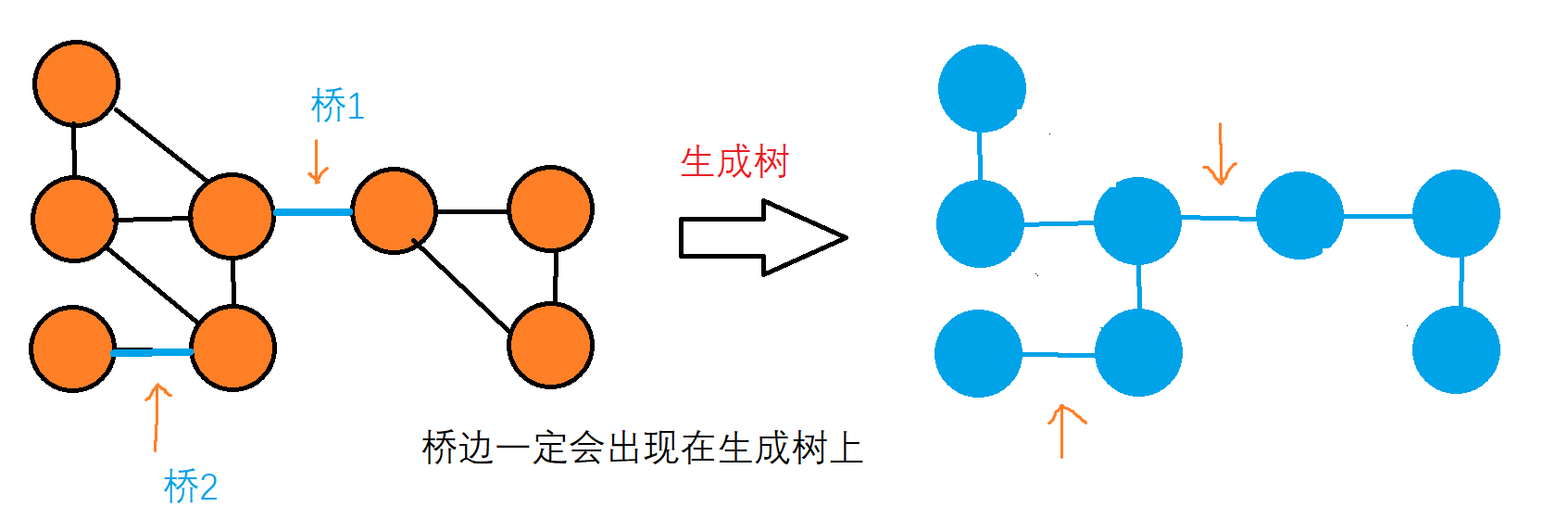




可以看到稀疏图中，算法符合O(n2)的复杂度，而稠密图中，算法符合O(n4)的复杂度

**引入生成树优化**

因为桥边一定会出现在生成树上，所以对于基准法，我们只需要枚举生成树上的边，而不需要枚举所有的边，一样能找到答案

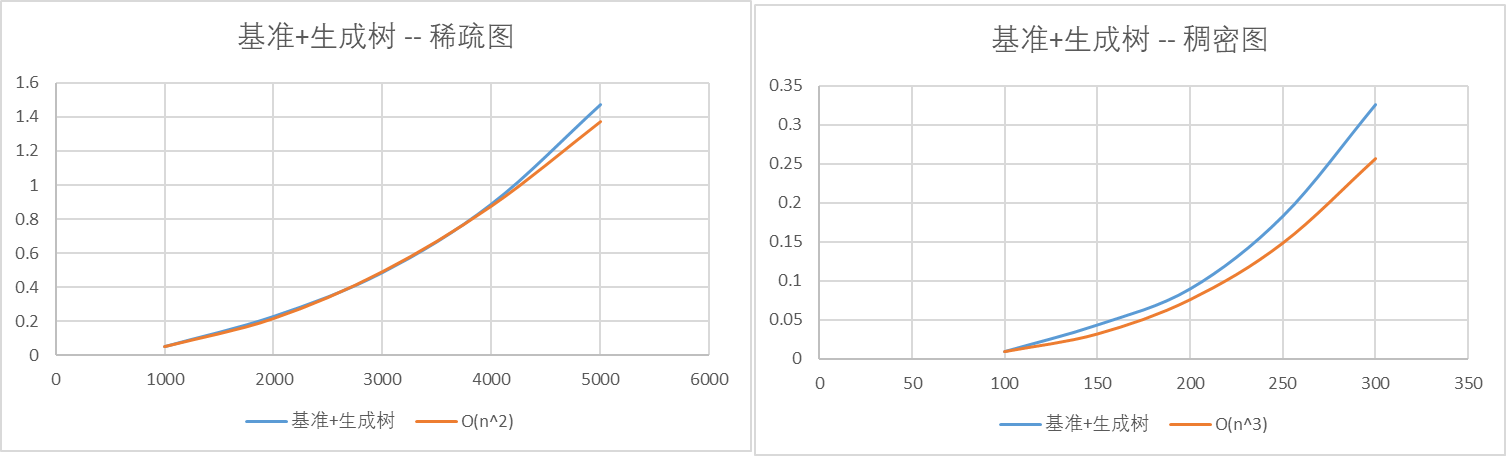


生成树优化能够使得枚举边的代价从O(e)变为O(n)，我们可以利用这个特性优化基准法

**基准法+生成树优化**

生成树优化能够使得枚举边的代价从O(e)变为O(n)，对于稀疏图，复杂度为O(n2)，对于稠密图，复杂度为O(n3)



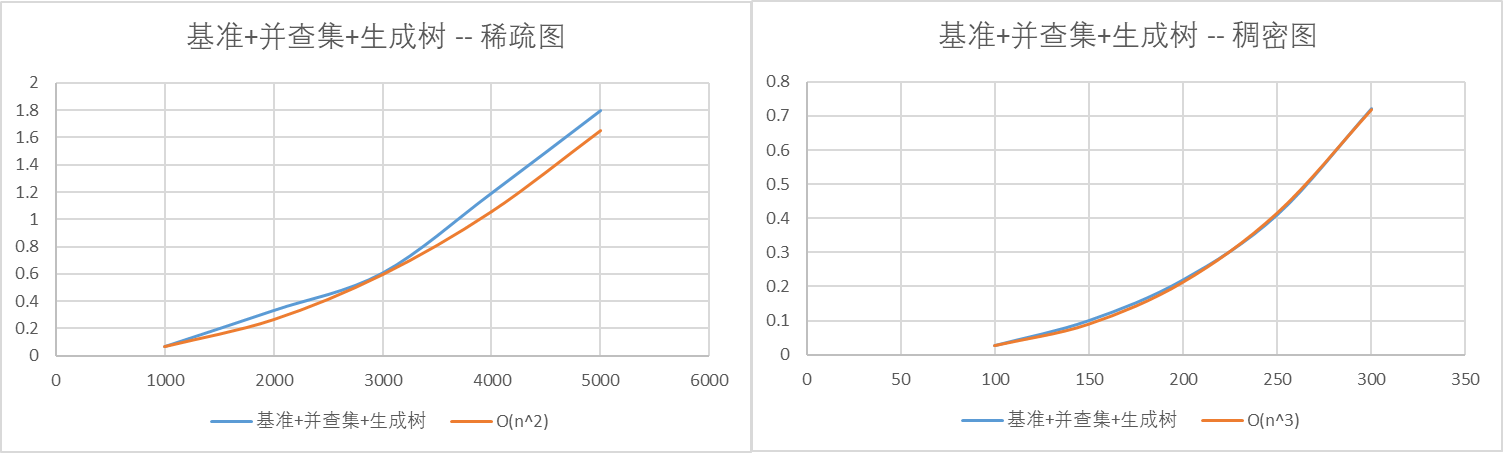


可以看到在图像和数据上，稀疏图的情况下算法时间复杂度符合O(n2)增长，在稠密图的情况下算法的时间复杂度符合O(n3)增长

**基准法+并查集+生成树优化**

生成树优化能够使得枚举边的代价从O(e)变为O(n)，对于稀疏图，复杂度为O(n2)，对于稠密图，复杂度为O(n3)





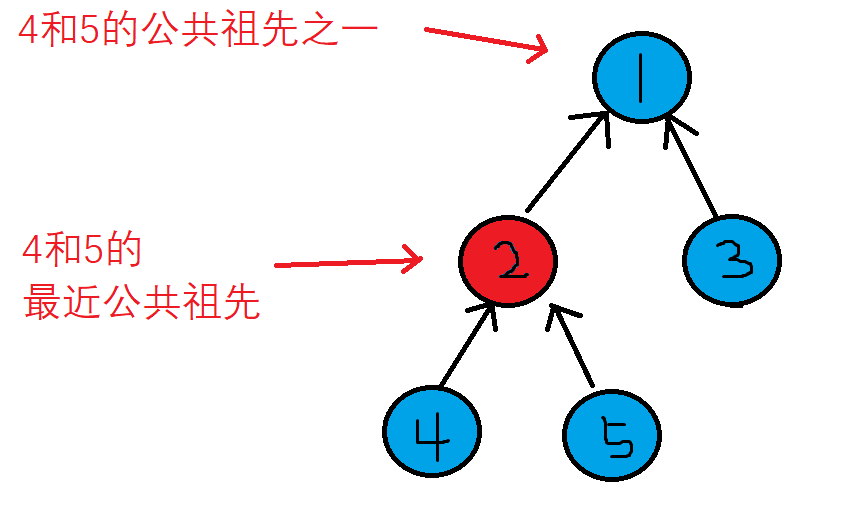
可以看到在图像和数据上，稀疏图的情况下算法时间复杂度符合O(n2)增长，在稠密图的情况下算法的时间复杂度基本符合O(n3)增长

**Lca求环边求桥**

逆向思维：我们不去主动找桥，因为定义【一条边是一座桥当且仅当这条边不在任何环上】，我们把环边找出来，然后剩下的不是环边的边，就是桥了

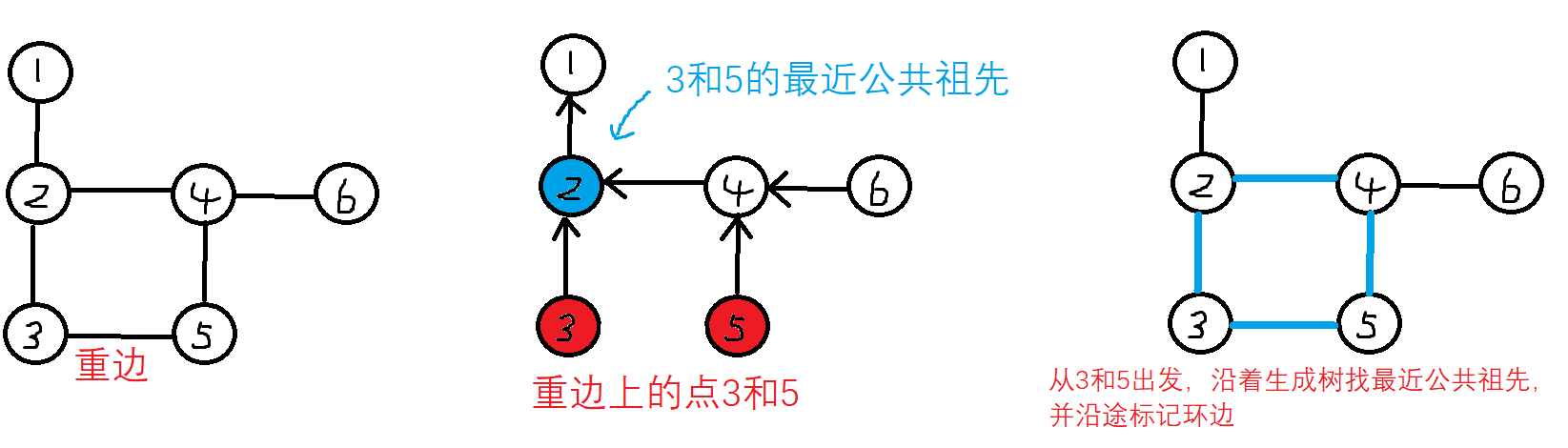
**引入lca问题：最近公共祖先**

最近公共祖先，顾名思义即两个节点的最近的能找到的公共的祖先



先利用dfs建立生成树，对于图中那些不在生成树上的边，我们叫它【重边】

对于重边上的两点v1和v2，我们找v1和v2的最近公共祖先，并且把从v1和v2出发，到他们的最近公共祖先的路径上的边，全部标记为环边



**dfs配合并查集求解lca问题：**

lca[x]表示集合x最高顶点的编号，father[]为并查集数组

收集所有lca查询并且使用dfs建立生成树，然后按照生成树顺序对图做dfs

每次递归开始前：x节点自成一个集合

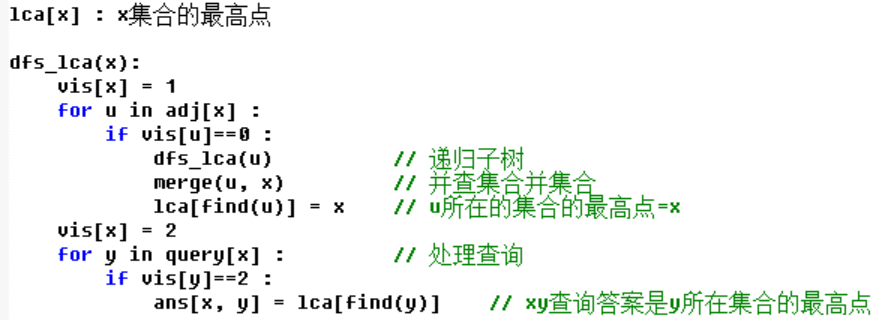
递归所有子树

一个子树递归结束：将子树并到x节点集合

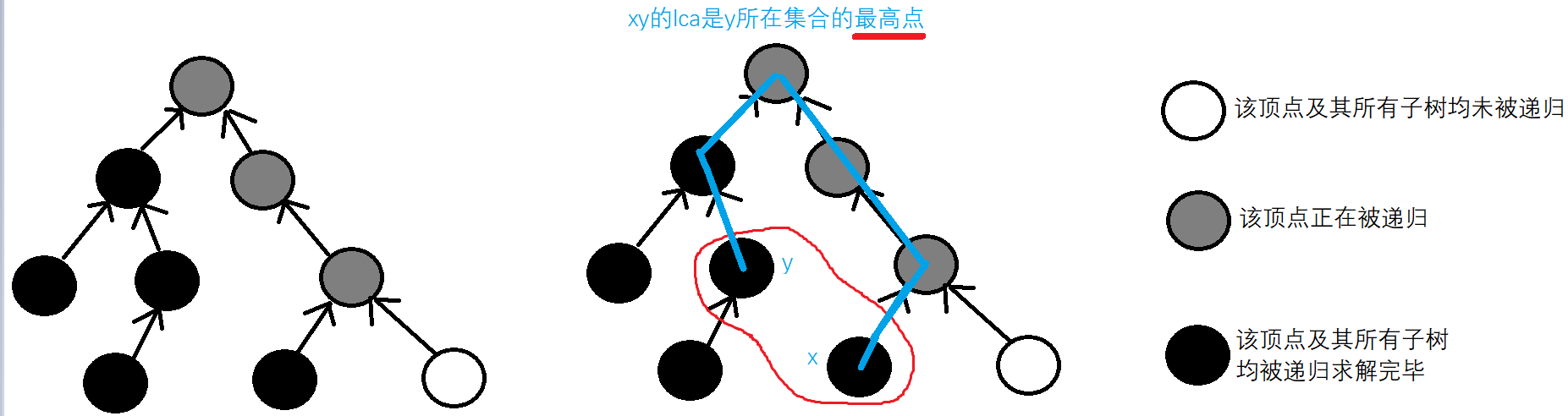
x所有子树递归结束时：遍历和x有关的查询，并尝试回答这些查询

假设查询为 (x, y) 如果y节点及其所有子树已经完成递归，那么答案是y所在的集合的最高点编号，如果y未完成所有的递归，那么先搁置这个查询（等一下y完成递归的时候会再次尝试回答）

**伪代码描述：**



**图形描述：**

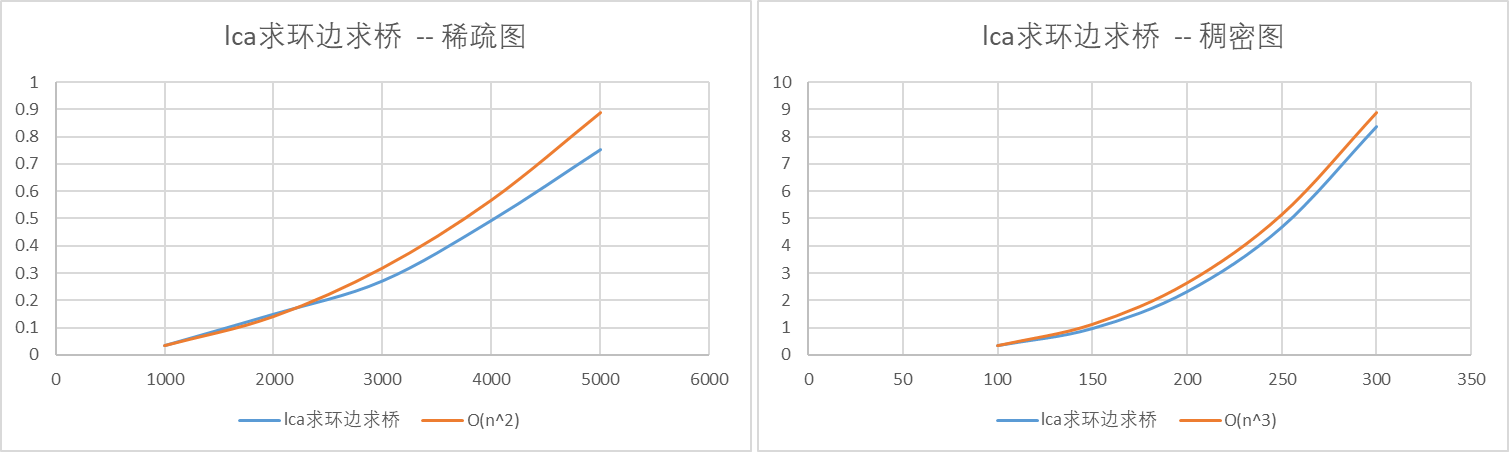


**复杂度分析：**

以dfs为模板求lca的时间复杂度是O(n+e+q)其中q为查询次数，对于稀疏图，查询次数q为n，对于稠密图，查询次数q为n2，因为除了生成树上的n-1条边，其他的都要查询

对于每个查询，都需要沿着生成树，往回找他们的最近公共祖先，并且标记沿途的边，这个操作的时间复杂度是不确定的，但是其上界是O(n)，因为走完所有的顶点，故整个算法的时间复杂度上界是：稀疏图O(n2)，稠密图O(n3)

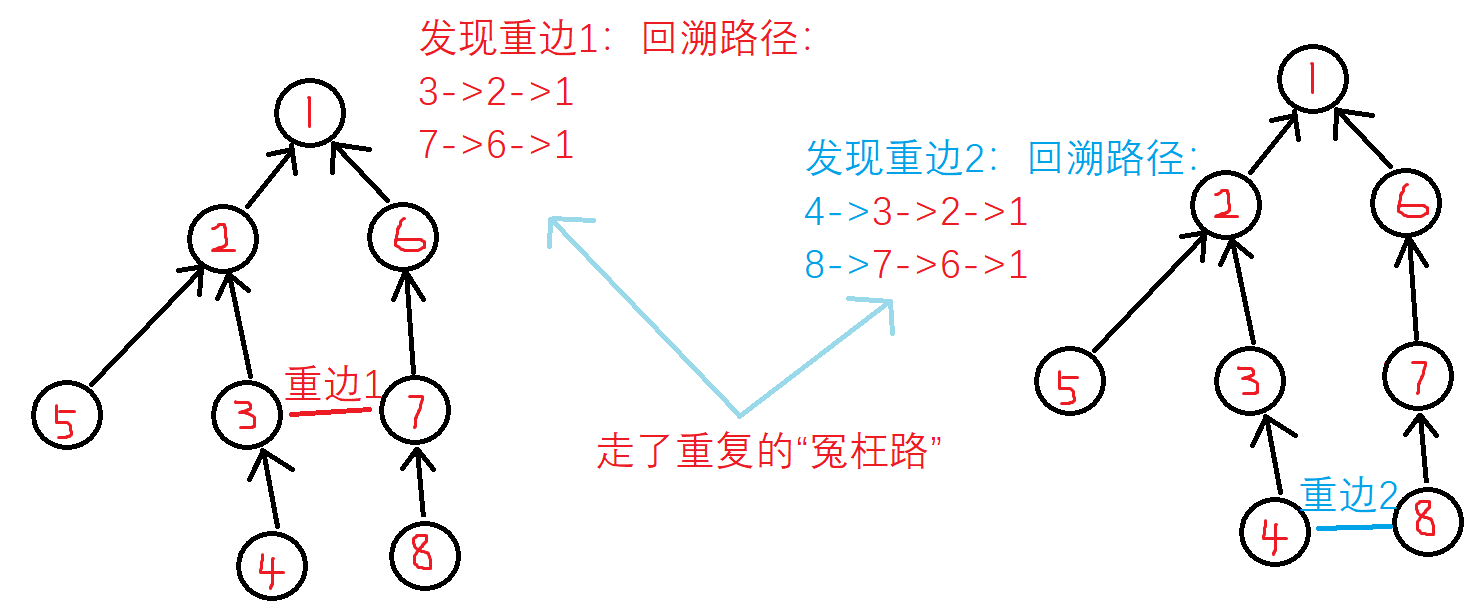




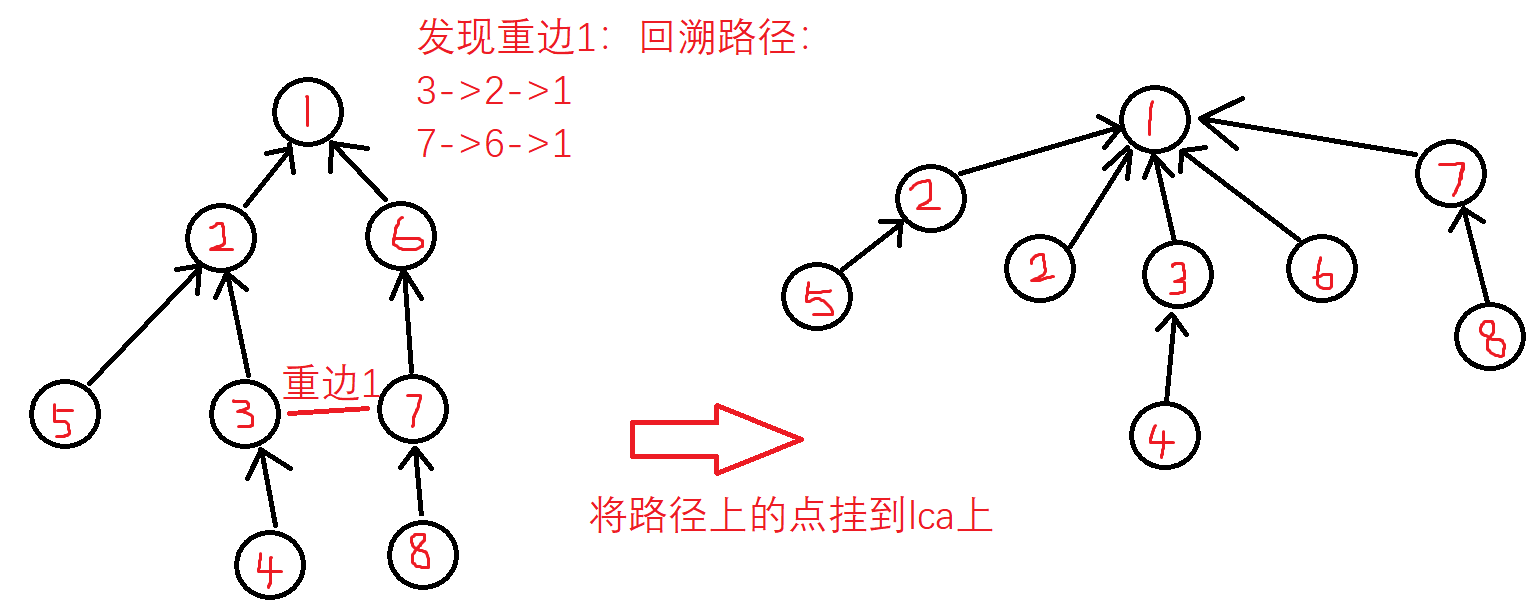
可以看到在图像和数据上，稀疏图的情况下算法时间复杂度符合O(n2)增长，在稠密图的情况下算法的时间复杂度基本符合O(n3)增长

**Lca求环边：路径压缩策略**

如图，因为边非常多，我们有时候可能会走很多冤枉路



我们在沿着生成树向上找lca的时候，找到lca并记录它，然后不妨将路径上的所有节点挂到lca上面（即查找的终点），表示该点和lca点之间的路径被记录过了，在下次向上查找的时候，我们就会直接跳到lca上，不必走冤枉路

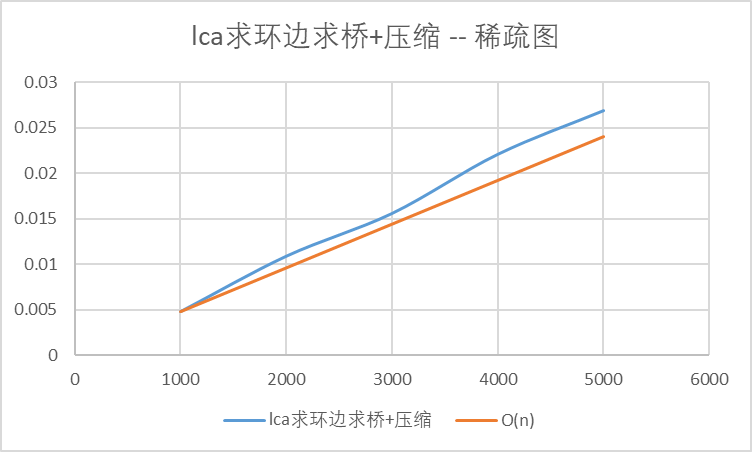


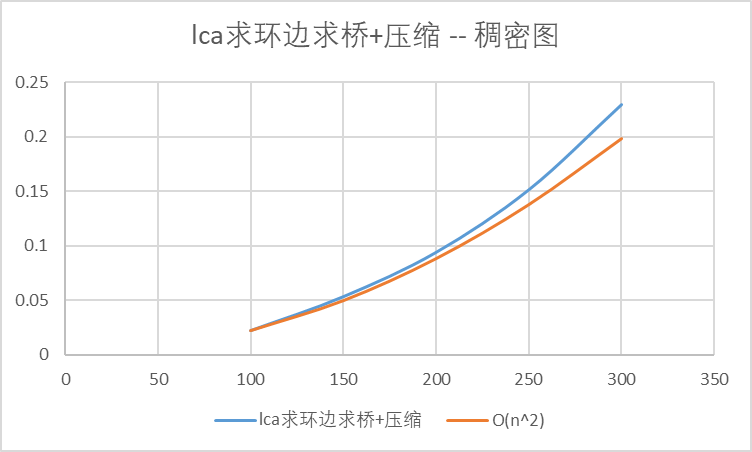
**Lca+路径压缩：复杂度分析：**

因为加入了路径压缩策略，和并查集类似，我们最多把n个顶点挂到父节点上面，每次找路径都要挂，总共找e次路径，均摊下来使得找路径的复杂度为O(1)，于是我们算法的复杂度变为：稀疏图：O(e)\*O(1) = O(n) 稠密图：O(e)\*O(1) = O(n2)

Lca+路径压缩：时间测试：





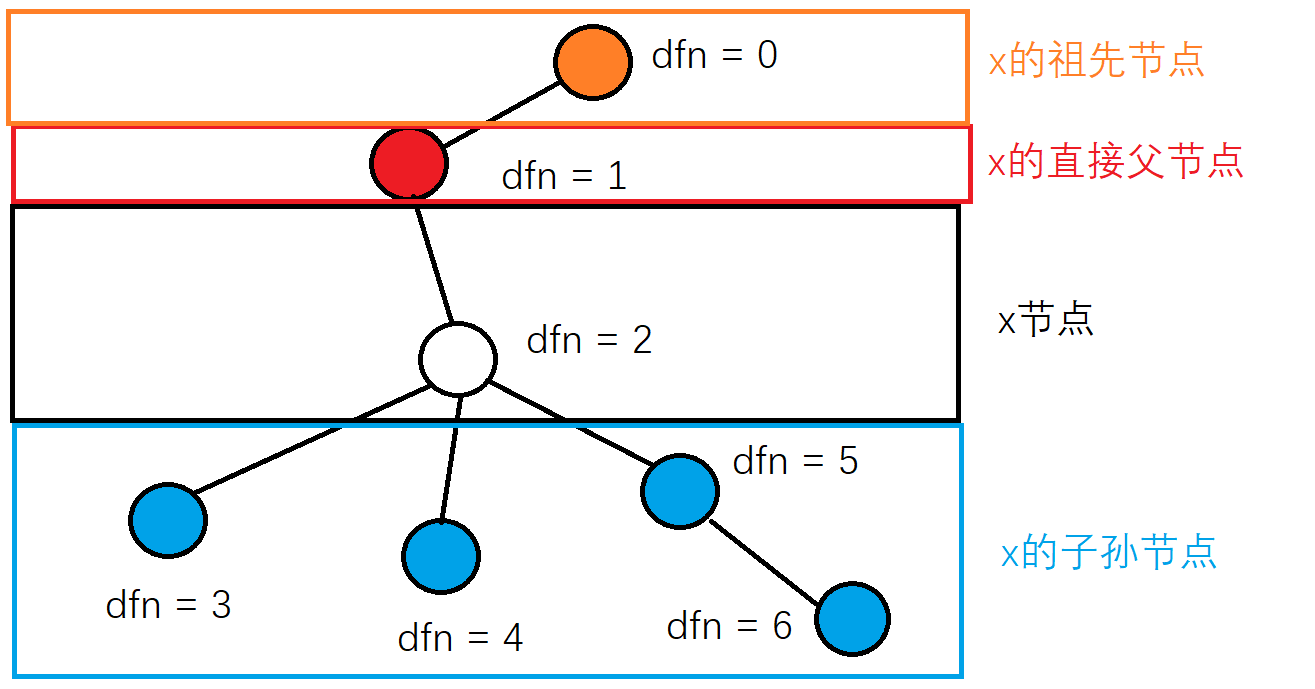


可以看到算法在稀疏图下的复杂度和我们预想的一样保持在O(n)，稠密图则为O(n2)

**Tarjan（塔扬）算法直接求桥：**

Tarjan算法基于dfs，一次dfs即可求解问题。

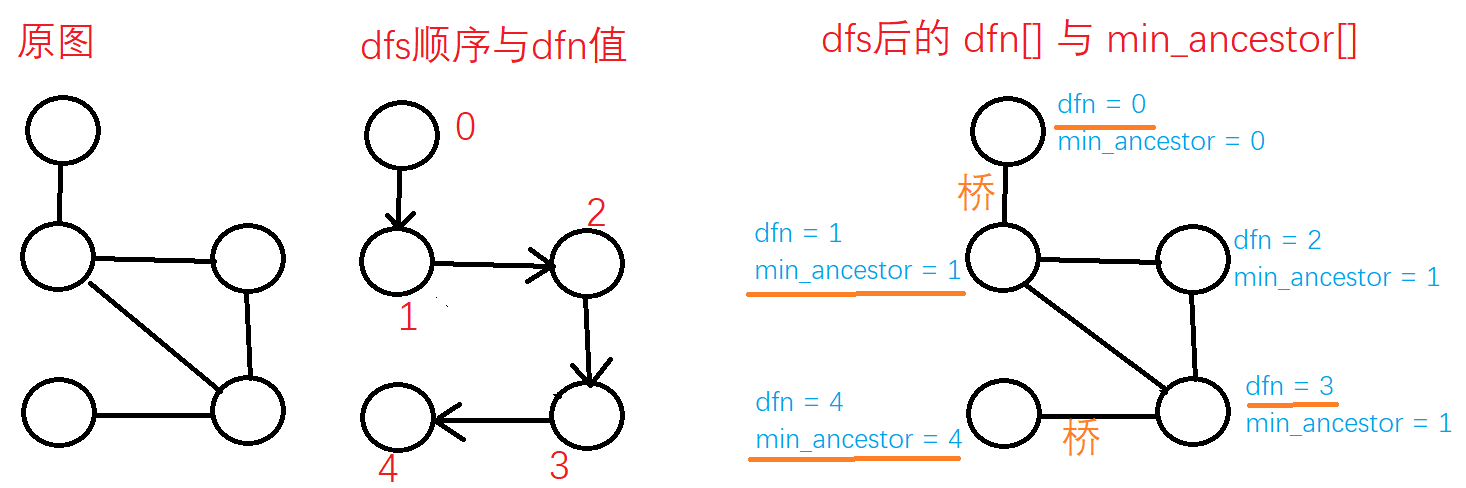
在一次dfs中维护两个数组，分别是dfn[]即深度优先数，和min\_ancestor[]即最小可达祖先，dfn[]数组描述了dfs进行的顺序，而min\_ancestor[x]则记录了x及其所有子孙节点能够到达的【dfn值最小的祖先节点（不包括x的直接父节点）】的dfn数值



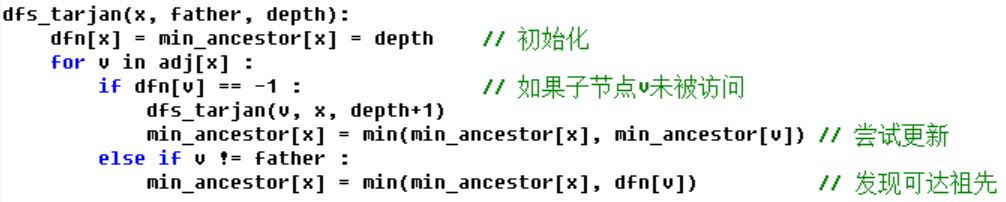
如果一条边的两个顶点v1和v2，满足dfn[v1] < min\_ancestor[v2]，那么这条边是桥

可以显然的从定义得知这个事实：因为除了和x相连的边，没有任何途径可以到达x之前的节点（体现在最小可达祖先大于x直接父节点的dfn值），也就是说x是唯一通路，是桥

图片描述：



伪代码描述：



**Tarjan算法时间复杂度分析：**

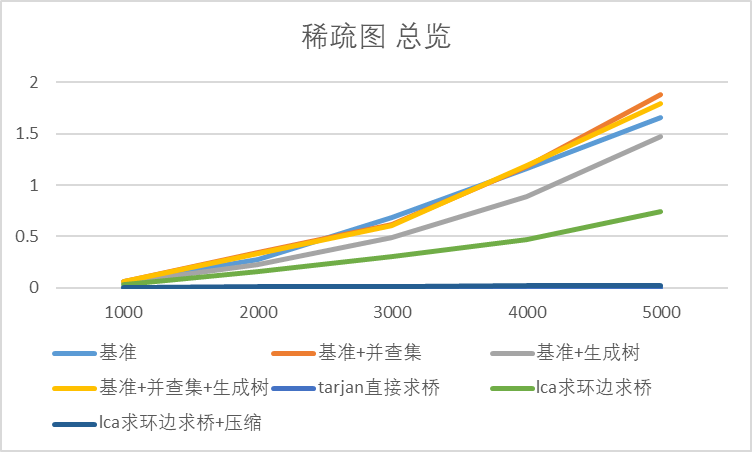
共需要一次dfs，故稀疏图中复杂度为O(n)，稠密图中复杂度为O(n2)

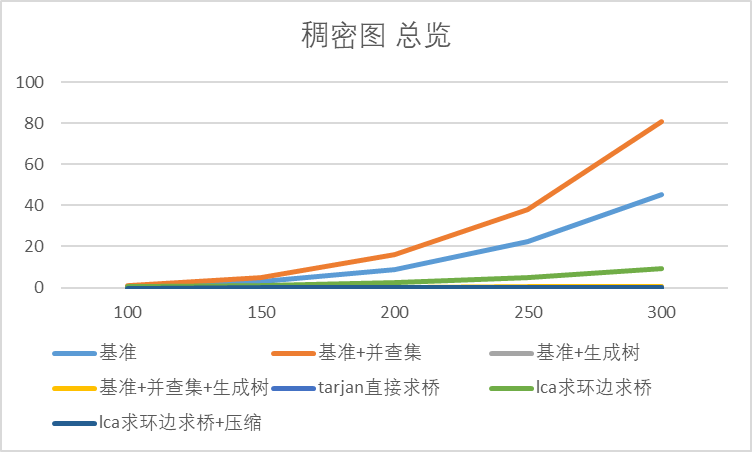


可以看到在稀疏图中tarjan算法几乎完全符合O(n)，是不可多得的线性算法，而稠密图中也完全符合O(n^2)的复杂度

**总览**（注：tarjan算法在这个规模过于快了，以至于测出来都是0或者0.0001不好对比）







**结论：**

**稀疏图**

复杂度：

Tarjan = lca+压缩 < lca = 基准+生成树 = 基准+并查集+生成树 = 基准 = 基准+并查集

用时：

Tarjan < lca+压缩 < lca < 基准+生成树 = 基准+并查集+生成树 = 基准+并查集 = 基准

**稠密图**

复杂度：

Tarjan = lca+压缩 < lca = 基准+生成树 = 基准+并查集+生成树 < 基准 = 基准+并查集

用时：

Tarjan < lca+压缩 < 基准+生成树 = 基准+并查集+生成树 < lca < 基准+并查集 = 基准

结论：

1. 一次dfs解决，tarjan（塔扬）算法是最快的算法
2. 引入路径压缩的lca第二快（复杂度低），但是操作语句较为繁琐所以慢于tarjan
3. Lca求环边和带生成树优化的基准法远远地快于普通基准法
4. Lca求环边在边情况少的图中速度远远快于基准法，而在稠密图中，慢于基准法+生成树优化，虽然同为O(n3)的复杂度，但是因为代码的语句数量明显增多（先生成生成树，然后反选重边，然后查询重边的lca，递归沿着路径标记环边，然后通过环边反选桥），而且使用多种复杂的数据结构，比如并查集，哈希set，vector，并且沿着环边标记路径使用的是递归操作，导致算法的常数开销过大，在边多的情况下，不理想
5. Lca求环边在稠密图中达到最坏情况，即复杂度上限O(n3)，从图中曲线也可以看出
6. 基准法用并查集计算连通分支数目，虽然有路径压缩策略，但总体执行语句数量多于dfs法，因此代码的常数开销过大，导致并查集慢于dfs，但其实它们拥有相同的复杂度

给定数据测试：



在测试数据样例和mediumDG中测试时间，因为规模过小导致时间不好测量

在largeG里面测试，只有lca+压缩和tarjan可以跑完整个过程，但是因为tarjan算法代码实现和语句较为简单，而相对来说lca+压缩语句比较复杂，所以很慢

Lca+压缩，用到较多的复杂数据结构，并且未使用任何针对C++STL的编译优化，所以是代码常数开销很大慢于tarjan

**总结：**

并查集数据结构要尽量使用路径压缩策略，能够大大加快查询的速度

在求解大规模问题的时候尽量选择简单的数据结构

编写代码时要注意代码语句不能过长，否则同样复杂度的算法，执行用时会有差距

在大规模递归的时候要修改编译器的栈空间，否则数据容量一大，容易爆栈

|  |
| --- |
| 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。