**基准算法：基于BFS**

**思想**：删除某一条边，通过BFS计算图的连通分量个数，如果个数增加则为桥

**方法**：BFS搜索过程中跳过要搜索的边，如果搜索完连通，说明两点间可通过其他点到达，不是桥

**时间复杂度：O(E \* (V + E))**

每条边执行一次 BFS/DFS，时间复杂度为 O(V + E)，实际运行中如果连通提前终止

总共 E 条边，所以总时间复杂度为 O(E \* (V + E))

**固定点 O(E²) 固定边 O(V)**

**基准算法优化：并查集+路径压缩**

**思想**：并查集是一个树形的数据结构，维护同一个祖先的集合，利用并查集判断两个点是否连通

**方法**：去除测试边，重新构造一个并查集，如果两个顶点不在一个集合中，说明是桥

**路径压缩：**

1. 因为我们只关心一个点的祖先，因此可以在并查集加入边时直接连到祖先，减少搜索时间

2. 多个集合合并，将高度小的合并到高度大的

**时间复杂度**：O(E² \* α(V))

每条边处理需要：O(V) 初始化 + O(E) 次并查集操作

每次并查集查询时间复杂度为 O(α)（α 是阿克曼反函数）

总共 E 条边，所以总时间复杂度为 O(E \* (V + E \* α(V))) ≈ O(E² \* α(V))

**固定点 O(E²) 固定边 O(V)**

**算法2：标记环边**

**思想：性质 因为环边一定不是桥，所以桥一定是树边，**因此可以将问题转化为找到所有环边的数量

**找环边——构建DFS生成树找到反向边**

**流程：构建树—存储反向边—逐边处理—标记环边至LCA**

**DFS生成树：记录深度，前驱，存储反向边**

depth记录节点首次访问的深度，father数组记录前驱，visited记录访问状态，is\_Bridge表示树边

**并查集的用途在于标记环边时把节点加入并查集中，使每次只标记新的边**

**算法3：LCA （Lowest Common Ancestor）是指在一棵树中，两个节点的最近公共祖先节点**

**思想：一旦找到反向边，两个点到他们的最近公共祖先的边都是环边**

**朴素法：**将较深的节点向上移动，知道两个节点达到同一深度，然后两个节点共同上移直到到达公共祖先

时间复杂度：

1.DFS构建生成树 O(V + E)

2.并查集：初始化：O(V) 每次查找(find)：接近O(1)（路径压缩优化）

每次合并(unite)：接近O(1)（路径压缩优化）

3.标记环边：每条非树边处理接近O(1) 时间复杂度：O(E × α(V))

**总时间复杂度**：O(V + E × α(V))

**新问题1：生成树该如何构建？ 为什么反向边不是预期的值反而能达到更快的速度？**

**OK!**

**新问题2**： 标记过程是否能够优化：是否需要每次都交换深度最低？ 并查集路径压缩后它还是按原来的那样进行简单的深度比较吗？

没有改变深度，只是通过并查集来维护这一个集合，保证集合的根节点是深度较小的元素从而不会陷入回退的问题； 深度比较依然是按照树中的深度进行