桥

王曦 2021192010

数学与统计学院

算法设计与分析

2023 年 05 月 23 日

1. 实验内容

实验内容

1. 给定一张无向图, 求其桥数.
2. 实现基准算法.
3. 用并查集设计一个比基准算法更高效的算法.

2. 实验环境与约定

实验环境与约定

- 实验环境: GNU C++17 (O2).
- 约定所有实际运行时间都不包含数据生成和输入的时间.

3. 暴力法

暴力法

- 思路:

(1) 先求原图的连通块数, 再枚举删除的边, 将其删除后求图的连通块个数, 若连通块个数增加, 则该边为桥.

(2) 求图的连通块个数可用 DFS 或 BFS 实现, 其中前者为递归实现, 效率较低, 故实验中采用非递归实现的后者.

(3) 若用邻接矩阵存图, 则删边、加边操作的时间复杂度为 $O(1)$, 单次 BFS 的时间复杂度为 $O(n^2)$; 若用邻接表存图, 为支持快速删边、加边操作, 可用 vector 套 set 作为邻接表, 则删边、加边操作的时间复杂度为 $O(\log m)$, 单次 BFS 的时间复杂度为 $O(n+m)$, 本次实验采用后者.

- 总时间复杂度: $O(m(n+m))$.

小数据验证

- 用如下两图分别验证算法的正确性, 得到答案分别为 0、6, 正确.

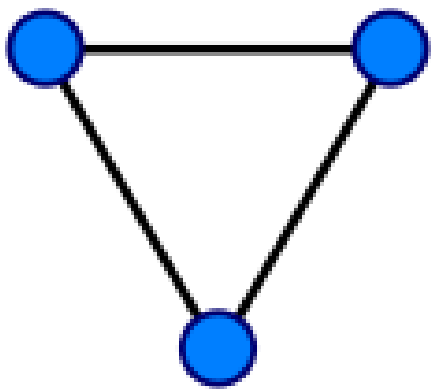

Figure 1: 无桥的无向连通图

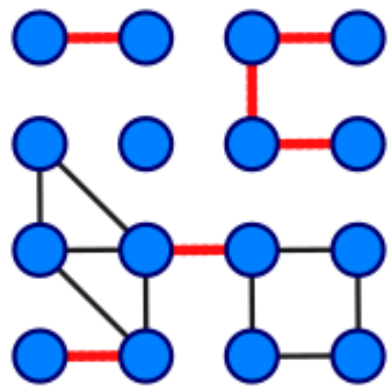

Figure 2: 包含 6 座桥 (红色) 的图

实际运行效率

样例	mediumG.txt	largeG.txt
节点数 n	50	1e6
边数 m	147	7586063
实际运行效率 (ms)	42	3 h+
答案	0	–

- 分析: 基准算法未注意到每次删边和加边对原图的影响范围很小, 每次删边后都遍历整个图, 效率低下.

4. 朴素并查集与路径压缩优化

朴素并查集与路径压缩优化

- 思路:

(1) 用并查集快速求连通块的个数. 具体地, 将边逐条加入, 用并查集维护图中的连通块, 使得 $fa[u] = u$ 的节点 u 的个数即图中连通块的个数.

(2) 朴素并查集 find() 时一步一步向上跳至集合的根节点, 时间复杂度为 $O(h)$, 其中 h 为树高, 最坏 $h = n$. 考虑优化, 注意到此处只关心元素的从属关系, 而不关心同个集合中元素的拓扑关系, 则 find() 时可将元素直接连接到集合的根节点上, 即路径压缩.

(3) 论文 "Yao, A. C. (1985). On the expected performance of path compression algorithms.SIAM Journal on Computing, 14(1), 129-133" 中证明了路径压缩后每次 find() 的时间复杂度为 $O(\alpha(n))$, 其中 $\alpha(n)$ 为反 Ackermann 函数, $n \leq 2^{2^{2^{1987}}}$ 时, 有 $\alpha(n) \leq 5$, 即可认为每次 find() 的时间复杂度为 $O(1)$.

- 总时间复杂度: $O(m(n+m))$.

实际运行效率

样例	mediumG.txt	largeG.txt
节点数 n	50	1e6
边数 m	147	7586063
实际运行效率 (ms)	1	3 h+
答案	0	–

- 分析: 引入路径压缩优化的并查集后, 实际运行效率有所提高, 但时间复杂度未改变, 效率仍较低, 无法在有限时间内得到 largeG.txt 的答案.

5. 可撤销并查集与线段树分治

可撤销并查集与线段树分治

- 思路:

(1) 合并时, 用栈记录每个版本的并查集的父亲节点和集合大小, 则可回退上一步操作. 注意回退依赖于集合中节点的拓扑关系, 故不可使用路径压缩优化.

(2) 因并查集的 find() 的时间复杂度与树高成正比, 考虑如何降低树高. 每次合并时将树高较小的集合合并到树高较大的集合中, 即按秩合并. 实践中, 按秩合并的实现比启发式合并的实现繁琐, 故一般采用启发式合并代替按秩合并, 可用势能分析证明启发式合并优化后每次 find() 的时间复杂度为 $O(\log n)$.

可撤销并查集与线段树分治

(3) 论文 “Tarjan, R. E., & Van Leeuwen, J. (1984). Worst-case analysis of set union algorithms. Journal of the ACM (JACM), 31(2), 245-281.” 中证明了按秩合并后每次 find() 的时间复杂度为 $O(\log n)$; 对朴素的并查集, 同时使用路径压缩和按秩合并优化后每次 find() 的时间复杂度为 $O(\alpha(n))$, 可认为每次 find() 的时间复杂度为 $O(1)$.

TABLE I. WORST-CASE RUNNING TIMES OF SET UNION ALGORITHMS IF $m \geq n$

	Naive linking	Linking by rank or size
Naive find	$\Theta(mn)$	$\Theta(m \log n)$
Compression	$\Theta(m \log_{(1+m/n)} n)$	$\Theta(m\alpha(m,n))$
Splitting	$\Theta(m \log_{(1+m/n)} n)$	$\Theta(m\alpha(m,n))$
Halving	$\Theta(m \log_{(1+m/n)} n)$	$\Theta(m\alpha(m,n))$
Type zero reversal	$\Theta(m \log n)$	$\Theta(m \log n)$
Type one reversal	$\Theta(m \log n)$	$\Theta(m \log n)$
Type two reversal	$\Theta(m \log(2 + n^2/m))$	$\Theta\left(m \log\left(2 + \frac{n \log n}{m}\right)\right)$
Collapsing	$\Theta(m + n^2)$	$\Theta(m + n \log n)$
Naive splicing	$\Theta(m \log_{(1+m/n)} n)$	—
Splicing by rank	—	$\Theta(m\alpha(m,n))$

TABLE II. WORST-CASE RUNNING TIMES OF SET UNION ALGORITHMS IF $m < n$

	Naive linking	Linking by rank or size
Naive find	$\Theta(mn)$	$\Theta(n + m \log n)$
Compression	$\Theta(n + m \log n)$	$\Theta(n + m\alpha(n,n))$
Splitting	$\Theta(n \log m)$	$\Theta(n + m\alpha(n,n))$
Halving	$\Omega(n + m \log n), O(n \log m)$	$\Theta(n + m\alpha(n,n))$
Type zero reversal	$\Theta(n + m \log n)$	$\Theta(n + m \log n)$
Type one reversal	$\Theta(n + m \log n)$	$\Theta(n + m \log n)$
Type two reversal	$\Theta(n + m \log n)$	$\Theta(n + m \log \log n)$
Collapsing	$\Theta(n^2)$	$\Theta(n \log n)$
Naive splicing	$\Theta(n + m \log m)$	—
Splicing by rank	—	$\Theta(n + m\alpha(m,m))$

Figure 3: 并查集的优化的最坏时间复杂度

(4) 将所有边加入图中后, 对每条边, 将其删除后, 统计图中连通块的个数, 再将该边加回, 总时间复杂度 $O(n + m \log n + m^2)$. 考虑优化, 注意查询连通块个数的操作可离线, 考虑对时间分治. 具体地, 考察每条边存在的时间段. 如下图所示, 其中不同颜色表示不同的权值, 方块表示在对应时刻删除对应颜色的边, 线段表示对应的权值的边存在的时间段:

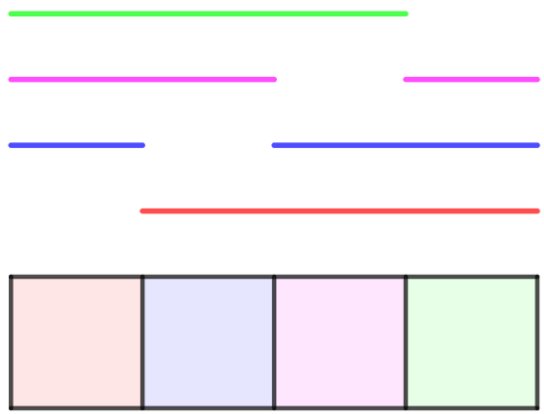

Figure 4: 对时间分治示意图

用线段树维护各边存在的时间段, 在统计答案 (查询) 时加边和回退即可. 总时间复杂度 $O(n + m \log^2 m)$.

实际运行效率

样例	mediumG.txt	largeG.txt
节点数 n	50	1e6
边数 m	147	7586063
实际运行效率 (ms)	1	59169
答案	0	8

- 分析:

(1) 引入可撤销并查集和线段树分治后, 实际运行效率大大提高, 时间复杂度有本质性的降低, 可在有限时间内获得 largeG.txt 的答案.

(2) 本算法的常数较大.

6. 生成森林与 LCA

生成森林与 LCA

- 思路:

(1) 用 DFS 求图的生成森林, 遍历时, 若当前边的两端点都未被搜过, 则该边为生成树边.

(2) 因一条边是桥 iff 它不在环上, 则桥是生成森林上的非环边. 故只需检查生成树上的边是否为桥即可. 最坏情况该图连通, 则有 $(n-1)$ 条生成树边, 此时时间复杂度为 $O(n(n+m))$.

(3) 考虑优化, 注意到无需输出方案, 只需输出桥数, 而非环边不易统计, 考虑统计环边, 则桥数 $=$ 总边数 $-$ 环边数. 显然非生成森林上的边都是环边, 只需统计生成森林上的环边. 将非生成树边逐条加入, 用朴素并查集检查是否成环, 若成环, 则该边连同所在的生成树构成一棵基环树, 进而在环上的生成树边为环边.

生成森林与 LCA

(4) 考虑成环时如何找到生成树上的环边. 如下图, 除边 $(4,6)$ 以外的边构成一棵生成树, 不妨设节点 1 为根节点. 加入边 $(4,6)$ 后, 节点 $2,4,6,5$ 成环, 则边 $(2,4),(2,5),(5,6)$ 都为环边. 显然成环后, 节点 4 和节点 6 到它们的 LCA 节点 2 的简单路径上的边都是环边.

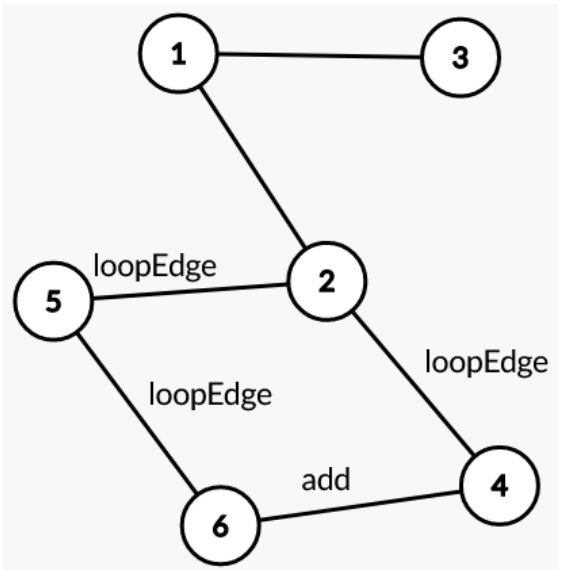

Figure 5: 环边示意图

生成森林与 LCA

(5) 因生成森林中可能不止一棵树, 且 LCA 与生成树的根节点有关, 则不易用优化的方法求 LCA. 考虑暴力求 LCA, 具体地, DFS 时记录每个节点在 DFS 树上的深度和前驱节点. 求节点 4 与节点 6 的 LCA 时, 先将深度较大者沿前驱回跳至两节点在同一深度, 再将两节点同时往上跳, 跳的过程中给环边打上标记. 最坏情况图为一条链, 加入连接两链端的边, 此时单次求 LCA 的时间复杂度为 $O(n)$.

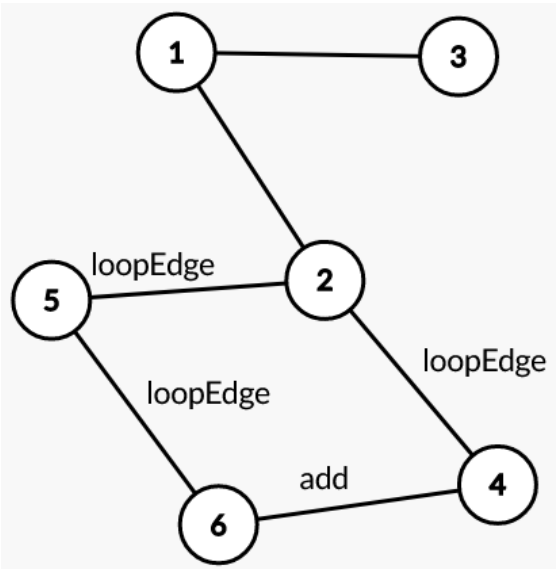

Figure 6: 环边示意图

实际运行效率

样例	mediumG.txt	largeG.txt
节点数 n	50	1e6
边数 m	147	7586063
实际运行效率 (ms)	0	3 h+
答案	0	8

- 分析: 引入生成森林和 LCA 后, 实际运行效率相较于朴素并查集的解法有所提高, 时间复杂度也少了一项 m^2, 但时间复杂度仍较低, 无法在有限时间内得到 largeG.txt 的答案.

7. Tarjan 算法求边双连通分量

Tarjan 算法求边双连通分量

- 思路:

(1) 无向图的边双连通分量 e-DCC 是双连通分量之一, 指无向图中极大的不含桥的连通块, 它有性质: 删除 e-DCC 中的任一边不改变其连通性.

(2) 对每个节点 u 定义两个时间戳:

(i) $dfn[u]$, 表示 u 的 DFS 序.

(ii) $low[u]$, 表示以 u 为根节点的子树中所有节点能追溯到的最小时间戳的节点.

与有向图的不同之处: 无向图无横插边. 注意搜索时不能往回搜, 否则每个节点沿其反向边都能追溯到其前驱节点, 进而图不存在桥.

(3) 对边 (x, y), 若节点 y 无法追溯到节点 x 或其祖先节点, 即 $dfn[x] < low[y]$ 时, 该边是桥.

- 总时间复杂度: $O(n + m)$.

实际运行效率

样例	mediumG.txt	largeG.txt
节点数 n	50	1e6
边数 m	147	7586063
实际运行效率 (ms)	0	2003
答案	0	8

- 分析: Tarjan 算法是无向图求边双连通分量和桥的标准算法, 它有线性的时间复杂度, 效率极高, 可迅速得到 largeG.txt 的答案.

8. 动态连通性问题简介

动态连通性问题简介

- 定义: 动态连通性问题 (Dynamic Connectivity Problem, DCP) 指在存在加边、删边操作时, 动态维护图的关于连通性的信息, 如连通块个数、连通块大小、双连通分量等.
- 分类:

(i) 离线: 操作和询问均已知, 可离线后排序等, 即允许不按顺序处理操作和询问.

(ii) 在线: 操作和询问均未知, 或后一个操作或询问依赖于前一个操作或询问的答案, 即必须按顺序处理操作和询问.

- 解法:

(i) 离线: 可撤销并查集 + 线段树分治.

(ii) 在线: 动态树 (如 Link Cut Tree、Euler Tour Tree 等) 维护图的生成森林.

- 例题: Dynamic connectivity contest (https://codeforces.com/gym/100551).

9. 不同算法的时间效率对比

随机生成简单图

- 用 set 记录加过的边, 防止出现重边.

  (1) 要求连通时, 可先生成一棵树, 再随机加边.
  (2) 不要求连通时, 直接随机加边.

不同算法的时间效率对比

- 固定节点数 $n = 1e4$, 取边数 $m = 1e3, 5e3, 1e4, 1.5e4, 2e4$.

节点 n	1e4	1e4	1e4	1e4	1e4
边数 m	1e3	5e3	1e4	1.5e4	2e4
基准算法	3132	13097	35675	94821	199008
朴素并查集	117	1151	6173	14745	25304
可撤销并查集 + 线段树分治	10	50	85	116	140
生成森林 + LCA	3	8	67	309	698
Tarjan 算法求 e-DCC	2	3	4	5	6

各算法的实际运行效率与边数的关系

- 固定节点数 $n = 1e4$, 取边数 $m = 1e3, 5e3, 1e4, 1.5e4, 2e4$.

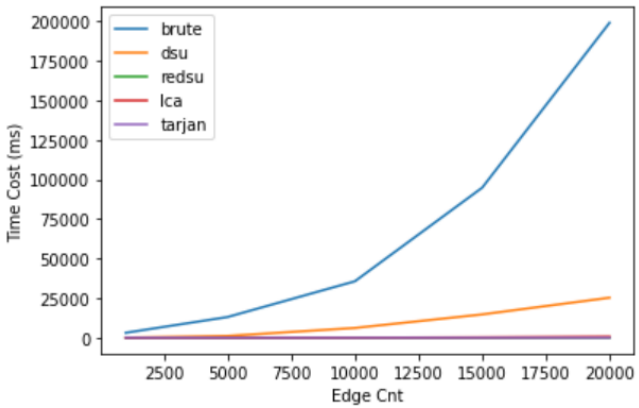

Figure 7: 各算法的实际运行效率 I

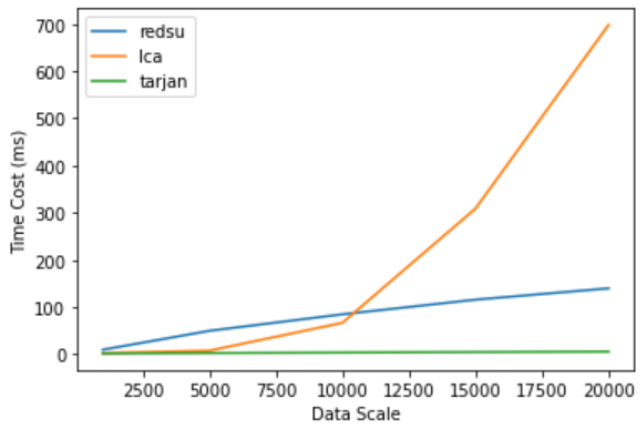

Figure 8: 各算法的实际运行效率 II

- 对固定的节点数, 各算法的实际运行时间随边数的增加而增大.
- 基准算法和朴素并查集 + 路径压缩的效率较低, 可撤销并查集 + 线段树分治、生成森林 + LCA、Tarjan 算法的效率较高.

各算法的实际运行效率与边数的关系

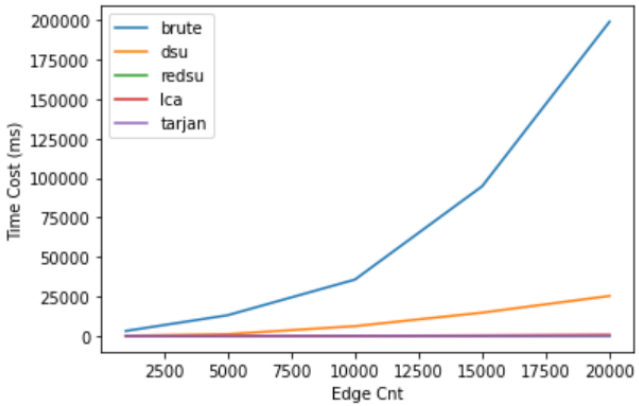

Figure 9: 各算法的实际运行效率 I

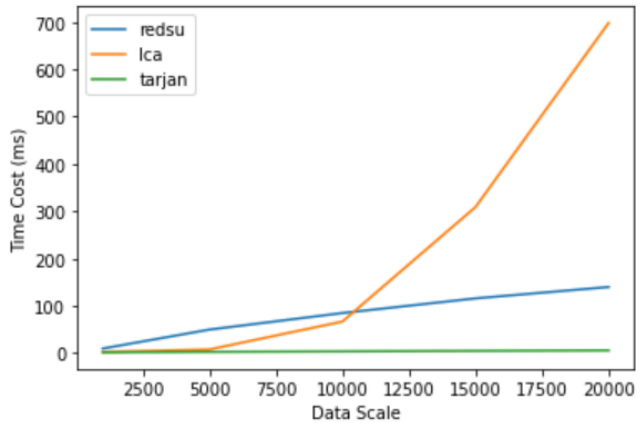

Figure 10: 各算法的实际运行效率 II

- 虽基准算法和朴素并查集 + 路径压缩的时间复杂度相同, 但朴素并查集实际运行效率远高于基准算法, 这是因为基准算法未注意到每次加边和删边只会影响到小范围的节点的连通性, 而并查集将同一连通块中的节点作为一个集合整体考虑, 并支持快速的合并和查询.

各算法的实际运行效率与边数的关系

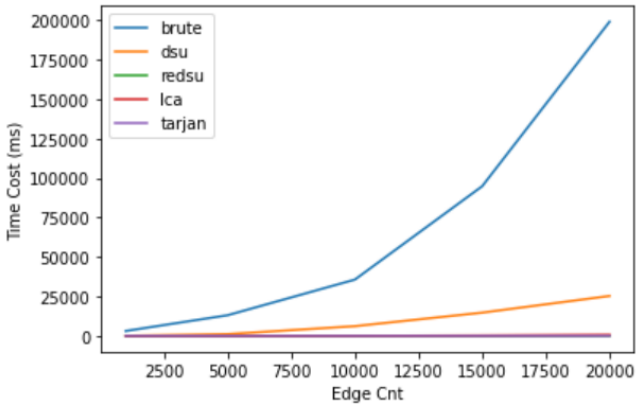

Figure 11: 各算法的实际运行效率 I

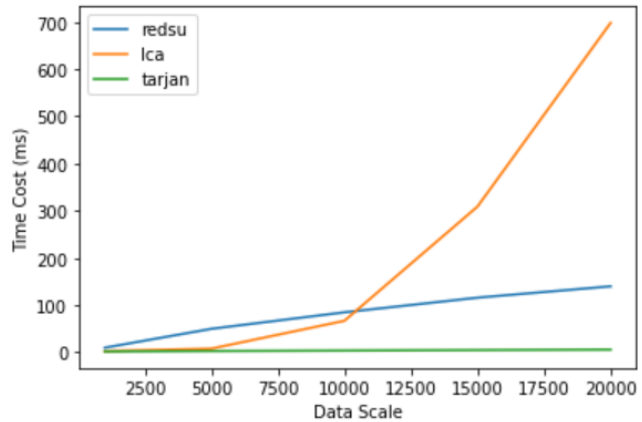

Figure 12: 各算法的实际运行效率 II

- 在可撤销并查集 + 线段树分治、生成森林 + LCA、Tarjan 算法中, Tarjan 算法的效率最高, 生成森林 + LCA 的效率最低.
- 数据范围较小时, 可撤销并查集 + 线段树分治的实际运行效率低于生成森林 + LCA 的实际运行效率, 这是因为前者的时间复杂度带有较大的常数. 数据范围较大时, 前者的实际运行效率明显高于后者.

各算法的实际运行效率与节点数的关系

- 固定边数 $m = 1e4$, 取节点数 $n = 1e3, 5e3, 1e4, 1.5e4, 2e4$.

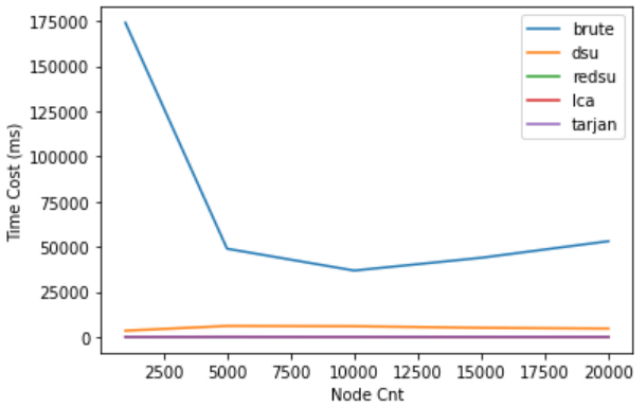

Figure 13: 各算法的实际运行效率 I

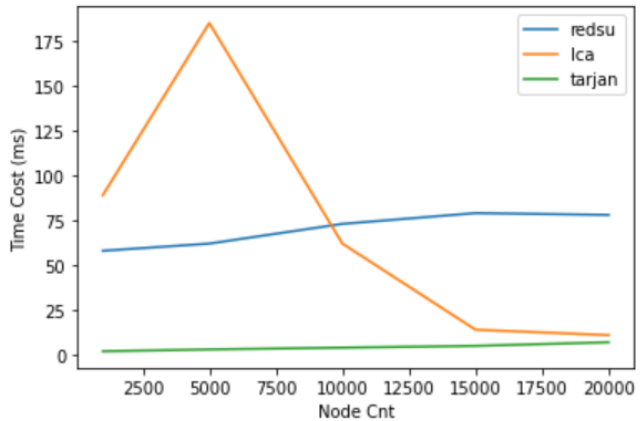

Figure 14: 各算法的实际运行效率 II

- 对固定的边数, 可撤销并查集 + 线段树分治、Tarjan 算法的实际运行时间随节点数的增加而增大, 但另外 3 个算法的实际运行时间不严格增大, 其中基准算法的实际运行时间先减小再增大, 朴素并查集 + 路径压缩、生成森林 + LCA 的实际运行时间先增大再减小, 这可能与图的稀疏性、连通性, 以及不同算法的策略不同有关.

各算法的实际运行效率与节点数的关系

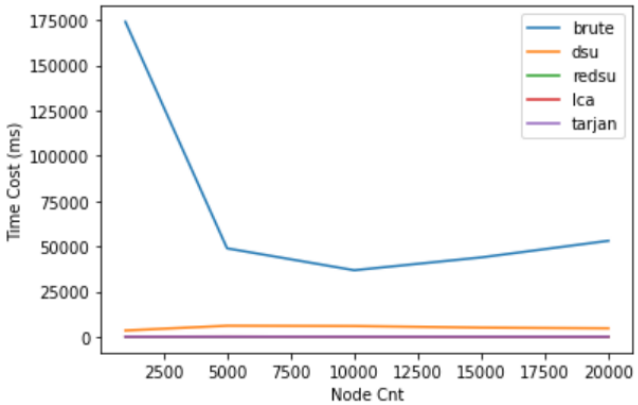

Figure 15: 各算法的实际运行效率 I

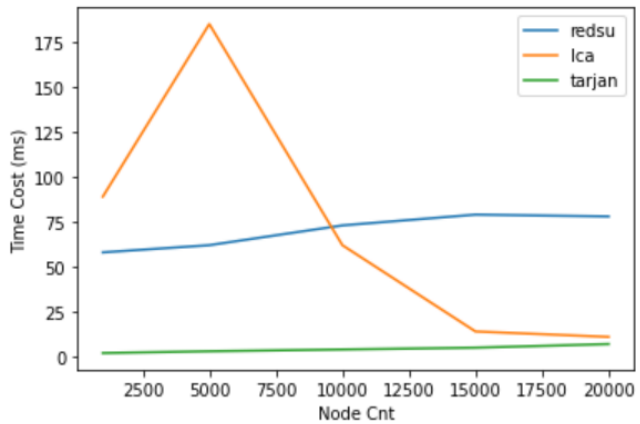

Figure 16: 各算法的实际运行效率 II

- 与固定节点数而变化边数相比, 固定边数而变化节点数对图的稀疏性、连通性影响较大, 导致一些算法的实际运行时间不随节点数的增加而严格增大.
- 基准算法和朴素并查集 + 路径压缩的效率较低, 可撤销并查集 + 线段树分治、生成森林 + LCA、Tarjan 算法的效率较高.

各算法的实际运行效率与节点数的关系

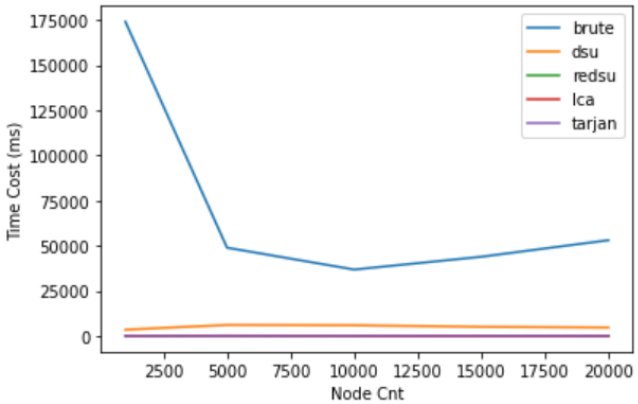

Figure 17: 各算法的实际运行效率 I

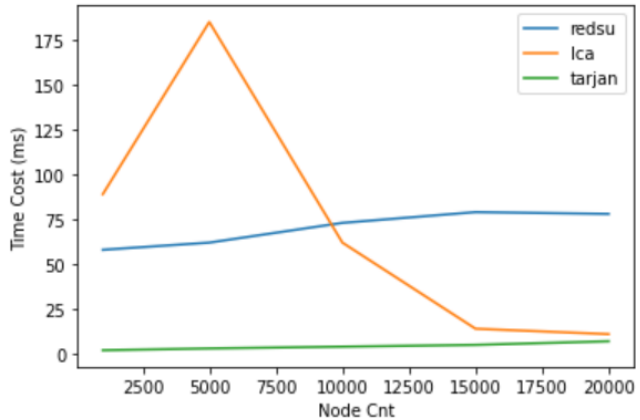

Figure 18: 各算法的实际运行效率 II

- 朴素并查集 + 路径压缩的实际运行效率远高于基准算法.
- 在可撤销并查集 + 线段树分治、生成森林 + LCA、Tarjan 算法中, Tarjan 算法的效率最高, 生成森林 + LCA 的效率最低.
- 生成森林 + LCA 的实际运行时间先增大后减小, 可能是因为数据范围较小时图较稠密, 桥数更多, 需暴力求更多次 LCA, 进而效率较低. 进行实验, 发现数据范围较小时桥数与边数为同一数量级, 进而需求较多次 LCA.

各算法的实际运行效率与节点数的关系

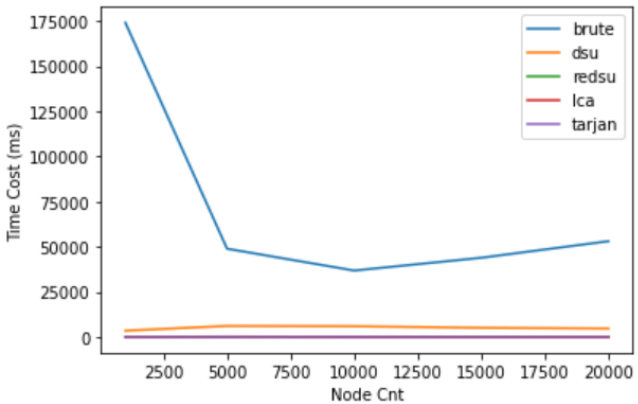

Figure 19: 各算法的实际运行效率 I

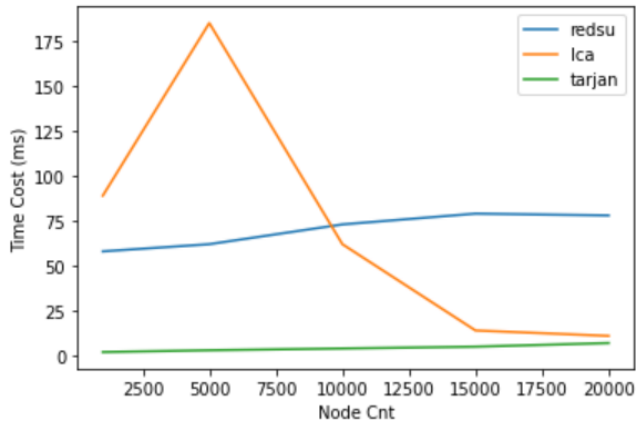

Figure 20: 各算法的实际运行效率 II

- 数据范围较大时, 生成森林 + LCA 的实际运行时间低于可撤销并查集 + 线段树分治, 可能是因为图本身较为稀疏, 前者求 LCA 的次数较少, 而后者无论图是否稀疏, 都需将所有边加入线段树后进行删边、询问、加边操作, 且常数较大.

谢　谢!

Thank you!