地图填色

王曦 2021192010

数学与统计学院

算法设计与分析

2023 年 04 月 11 日

1. 实验内容

实验内容

1. 将地图转化为平面图, 每个地区变成一个节点, 相邻地区用边相连. 为该地图的顶点染色, 使得任一边相连的两节点异色.
2. 对下图所示的小规模数据, 用四色填色测试验证算法的正确性.
3. 对附件中的三个地图数据分别尝试用 5 个 (le450_5a)、15 个 (le450_15b)、25 个 (le450_25a) 颜色为地图着色.
4. 随机产生不同规模的图, 用 4 种颜色染色, 分析算法效率与图规模的关系.

Figure 1: 小数据示意图

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. 将地图转化为图

将地图转化为图

- 给地图种的各区域编号后视为一个节点, 在相邻的两区域间对应的节点间连边, 将地图转化为图.
- 因图中各边不交叉, 故它是平面图. 下面的染色问题都针对平面图.

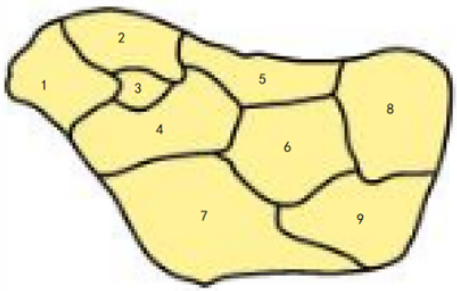

Figure 2: 给地区编号

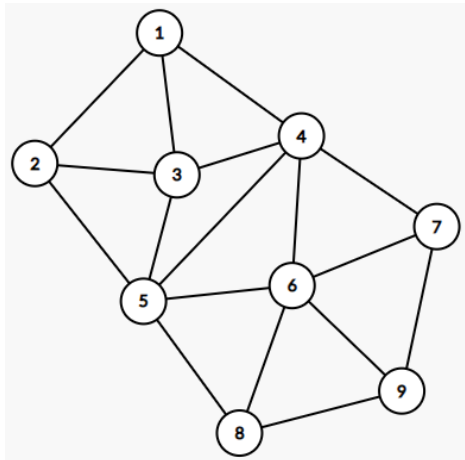

Figure 3: 将地图转化为图

- 地图填色问题的运行效率的影响因素猜想: 地图的节点数 n、地图的边数 m、颜色数 $MAXC$、节点的平均度数 $avgd$.

4. 朴素回溯法

朴素回溯法 (brute.cpp)

- 思路: 从某个节点开始染色, 先检查该节点能否染色. 若能染色, 则继续递归染下一个节点, 否则回溯.
- 错误解法: 从某个节点开始染色, 先检查该节点能否染色. 若能染色, 则染色并递归到该节点的邻居节点继续染色; 否则回溯. 重复该过程直至找到一组解, 或搜索完所有情况后证明无解.

朴素回溯法 (brute.cpp)

- 上述解法是错误的, 反例如下图所示. 若从节点 2 开始搜索, 则下一个搜索的节点可能为节点 1 或节点 3, 不妨设下一个搜索的节点是节点 1. 搜索节点 1 时, 因它无其他邻居节点, 则将回溯到节点 2, 此时节点 1 的颜色会被清空. 下一个搜索的节点为节点 3, 同理它也将回溯到节点 2. 故上述解法未找到一组合法解, 但事实上合法解显然存在.
- 上述解法错误的原因是错误地假定了图的形态, 即误认为按给定的边的顺序搜索就能找到一组合法解.

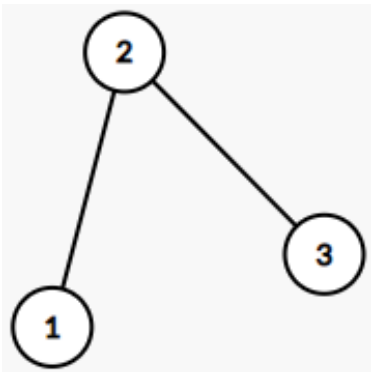

Figure 4: 错误解法的反例

朴素回溯法 (brute.cpp)

- 正确解法: 按节点的编号顺序搜索, 此时不受图的形态的影响.
- 为加入后续的各种优化, 按节点的编号顺序搜索显然是不合理的, 故实现时 dfs 记录当前已染的节点数.

brute.cpp 的实际运行效率

- 对样例地图染色, 耗时 0 ms, 得到的一组解如下图所示:

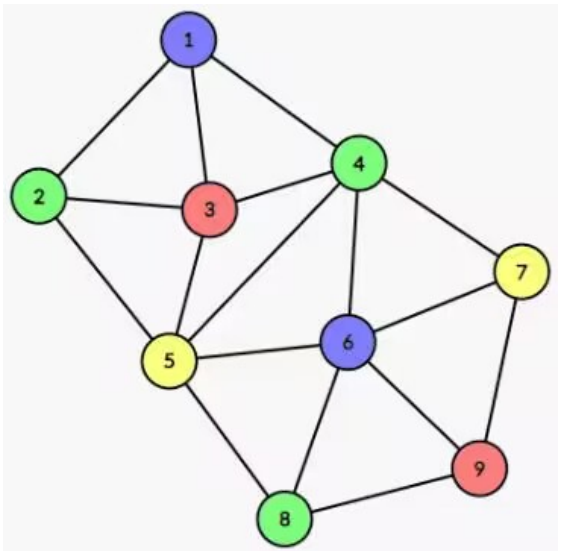

Figure 5: 对样例地图的染色

- 对样例地图染色, 耗时 1 ms, 得到所有解, 解数为 480.

brute.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	0
450	5714	5	42	1	3 h+
450	8169	15	94	1	3 h+
450	8260	25	128	3 h+	3 h+

- 分析:

(1) 对样例的图, 回溯法给出的解恰用 4 种颜色进行合法染色, 验证了四色定理在该图上成立.

(2) 数据范围较小时, 回溯法运行效率高.

(3) 数据范围较大时, 朴素的回溯法运行效率低, 需引入其他优化.

5. 优化 I: 剪枝——优化搜索顺序 I

优化 I: 剪枝——优化搜索顺序 I

- 优化搜索顺序 I: 优先搜索可填颜色数较少的节点 (op1.cpp).
- 思想:

(1) 大部分情况下, 优先搜索分支较少的节点能更快地找到一组合法解.

(2) 在本题中, 应优先搜索当前节点的可填颜色数较少的邻居节点, 因为可填颜色数越少的节点的搜索空间越小.

(3) 实现时, 类似于朴素的 Dijkstra 算法, 贪心地选择一个可用颜色的 set 的元素个数较小的节点搜索.

(4) 颜色数较少时, set 可用二进制数代替.

op1.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	2 (↓)
450	5714	5	42	4544 (↓)	3 h+
450	8169	15	94	1	3 h+
450	8260	25	128	1 (↑)	3 h+

op1.cpp 的实际运行效率

- 分析:

(1) 与 brute.cpp 相比, 加入优化 I 后, 第一组数据稍有负优化, 第二组数据有明显的负优化, 但在第四组数据出现明显的正优化, 原因可能是第二组数据中, 按节点编号升序染色的顺序可以找到一组合法解, 而第四组数据的一组解并非如此.

(2) 加入剪枝后在小数据出现负优化, 但在大数据出现明显的正优化, 但这可能与图本身较特殊有关.

(3) 加入该优化后仍无法在较短时间内得到第二组数据的所有解, 需引入其他优化.

6. 优化 II: 剪枝——可行性剪枝

优化 II: 剪枝——可行性剪枝

- 可行性剪枝: 及时停止无法得到合法解的搜索路径 (op2.cpp).
- 思想:

(1) 若搜到不合法的方案, 或确定当前方案继续搜索下去无法得到合法解, 则不再向下搜索, 直接回溯.

(2) 在本题中, 给一个节点染色后, 更新其邻居节点的可用颜色. 若出现邻居节点的可用颜色为空, 则直接回溯.

op2.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	2
450	5714	5	42	4483 (↑)	3 h+
450	8169	15	94	1	3 h+
450	8260	25	128	1	3 h+

op2.cpp 的实际运行效率

- 分析:

(1) 与 op1.cpp 相比, 在第二组数据中稍有正优化.

(2) 加入该优化后仍无法在较短时间内得到第二组数据的所有解, 需引入其他优化.

7. 优化 III: 剪枝——优化搜索顺序 II

优化 III: 剪枝——优化搜索顺序 II

- 优化搜索顺序 II: 优先搜索度数较大的节点 (op3.cpp).
- 思想:

(1) 大部分情况下, 优先搜索分支较少的节点能更快找到一组合法解.

(2) 在本题中, 应优先搜索剩余节点中度数最大的节点, 因为度数较大的节点对其邻居节点可染颜色的限制越多, 对搜索空间的减小越明显.

op3.cpp 的实际运行效率

- 大规模数据实验
- 第二组数据的方案数为 3840.

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	3 (↓)
450	5714	5	42	2063 (↑)	337509 (↑)
450	8169	15	94	20 (↓)	3 h+
450	8260	25	128	11 (↓)	3 h+

op3.cpp 的实际运行效率

- 分析:

(1) 与 op2.cpp 相比, 求第二组数据的一组解和求所有解的时间都明显缩短, 且能在可接受的时间内求得所有解.

(2) 与 op2.cpp 相比, 求第三组、第四组数据的一组解产生负优化, 其中第三组的负优化更大, 原因可能是第三组数据对节点的颜色限制多而可用颜色少, 按加入优化后的搜索顺序难找到一组合法解; 而第四组数据的可用颜色多, 更易找到一组合法解.

(3) 加入该优化后仍无法在较短时间内得到第三、四组数据的所有解, 需引入其他优化.

8. 优化 IV：剪枝——优化搜索顺序 III

优化 IV：剪枝——优化搜索顺序 III

- 优化搜索顺序 III: 优先染当前节点可染但邻居节点不可染的颜色 (op4.cpp).
- 思想:

(1) 染色时, 应减小当前节点所染颜色对邻居节点可染颜色的影响, 尽量保证优先搜索合法方案. 该优化有利于加快找到一组解的速度, 对于找到所有解的优化不明显.

(2) 在本题中, 应优先染当前节点可染但邻居节点不可染的颜色.

(3) 实现时, 将当前节点可染的颜色按该颜色在该节点的邻居节点的可染颜色中出现的次数非降序排列, 优先染出现次数最少的节点.

op4.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	3
450	5714	5	42	1797 (↑)	482416 (↓)
450	8169	15	94	12 (↑)	3 h+
450	8260	25	128	20 (↓)	3 h+

op4.cpp 的实际运行效率

- 分析:

(1) 与 op3.cpp 相比, 求第二组数据的一组解的时间明显缩短, 但求所有解产生负优化, 原因可能是运行过程中需大量调用 sort() 函数, 导致时间复杂度增大.

(2) 与 op3.cpp 相比, 求第二组数据的一组解稍有正优化, 求第三组数据的一组解稍有负优化, 这与图本身较为特殊有关.

(3) 加入该优化后仍无法在较短时间内得到第三、四组数据的所有解, 需引入其他优化.

9. 优化 V：剪枝——排除等效冗余

优化 V：剪枝——排除等效冗余

- 排除等效冗余：用组合计数的方法统计方案数 (op5.cpp).
- 思想：

(1) 若不考虑顺序, 应尽量用组合的方式搜索, 因为大部分情况下用组合的方式搜索比用排列的方式搜索效率更高.

(2) 在本题中, 若要统计总方案数, 则搜索时对每个节点, 可不枚举所有颜色, 而规定使用的颜色在模加法意义下不小于前面使用的颜色, 即让求得的第一组染色方案的颜色分布有一定的字典序. 得到一系列互不同构的合法解后, 用组合计数的方法, 即考察颜色的全排列, 即可求得总方案数.

优化 V：剪枝——排除等效冗余

- 按节点数分为如下四种情况:

  (1) 若 $n = 1$, 则 $ans = MAXC$, 其中 $MAXC$ 为颜色数.
  (2) 若 $n = 2$, 则 $ans = MAXC \cdot (MAXC - 1)$.

优化 V：剪枝——排除等效冗余

(3) 若 $n = 3$, 则图有两种不同构的情况. 设节点 1 染 1 色得到的方案数为 res.

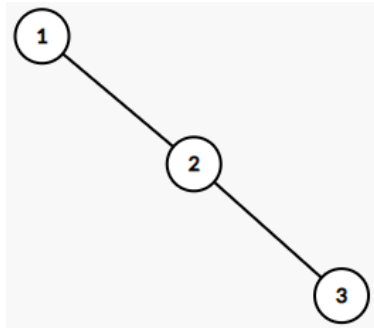

Figure 6: 链状

- 若三个节点成链状, 按节点 1 和节点 3 是否同色分类.

  (i) 若同色, 则 $ans = res \cdot MAXC \cdot (MAXC - 1)$.

  (ii) 若异色, 则

$ans = res \cdot MAXC \cdot (MAXC - 1) \cdot (MAXC - 2)$.

- 显然上述两种分类是不重不漏的.

优化 V：剪枝——排除等效冗余

(3) 若 $n = 3$, 则图有两种不同构的情况. 设节点 1 染 1 色得到的方案数为 res.

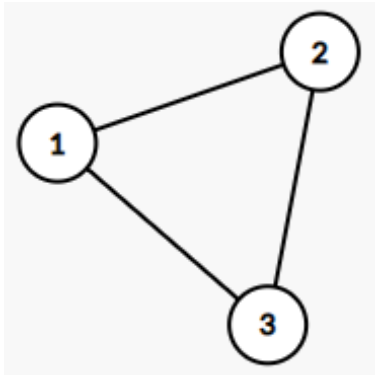

Figure 7: 环状

- 若三个节点成链状, 则
  $ans = res \cdot MAXC \cdot (MAXC - 1) \cdot (MAXC - 2)$.

优化 V：剪枝——排除等效冗余

(4) 若 $n \geq 4$, 则有多种不同构的图. 虽然可通过置换群和 Burnside 引理求不同构的图的个数, 但对每种不同构的图讨论其染色方案是复杂的, 故对 $n \geq 4$ 的情况, 染完前 3 个节点后, 用优化 III 继续搜索即可. 这种方法实现简单, 实际运行效率良好.

op5.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	/	1 (↑)
450	5714	5	42	/	1343 (↑)
450	8169	15	94	/	3 h+
450	8260	25	128	/	3 h+

op5.cpp 的实际运行效率

- 分析:

(1) 与 op4.cpp 相比, 求第一组和第二组数据的所有解的时间明显缩短.

(2) 加入该优化后仍无法在较短时间内得到第三、四组数据的所有解, 需引入其他优化.

10. 优化 VI: 用邻接表存图

优化 VI: 用邻接表存图

- 因上述算法涉及到多处遍历邻居节点的操作, 而用邻接矩阵存图时, 遍历邻居节点需枚举所有节点, 故改用邻接表存图可大大提高效率.
- 除遍历邻居节点外, 上述算法还涉及到检查边是否存在的操作, 此时用朴素的邻接表效率不如邻接矩阵高. 对此, 用 vector 套 set 作为邻接表, 可结合邻接表和邻接矩阵的优点, 两种操作的时间复杂度都为 $O(logn)$.

  (1) 求一组解使用 op6.cpp: 将 op4.cpp 换为邻接表存图.
  (2) 求所有解使用 op7.cpp: 将 op5.cpp 换为邻接表存图.

op6.cpp, op7.cpp 的实际运行效率

- 大规模数据实验

节点数	边数	颜色数	节点最大度数	一组解 (ms)	所有解 (ms)
9	17	4	4	0	0 (↑)
450	5714	5	42	4329 (↓)	6352 (↓)
450	8169	15	94	21 (↑)	3 h+
450	8260	25	128	44 (↓)	3 h+

op6.cpp, op7.cpp 的实际运行效率

- 分析:

(1) op6.cpp 与 op4.cpp 相比, 求一组解都产生负优化, 可能的原因:

(i) 邻接表中 set 带来的时间复杂度. 具体地, 虽然 set 每次修改、查询的时间复杂度都是 $O(logn)$, 但过程中涉及到大量操作, 导致运行时间变长.

(ii) 邻接表中 set 按照节点编号升序排列, 可能与数据中合法方案的搜索路径不同, 故产生较大的负优化.

(2) 实验表明: 在同时涉及遍历邻居节点和查询边是否存在, 且操作数量大时, 邻接矩阵的效率可能比用 vector 套 set 实现的邻接表效率高.

op6.cpp, op7.cpp 的实际运行效率

- 分析:

(3) op7.cpp 与 op5.cpp 相比, 求第一组数据的所有解的时间稍有优化, 求第二组数据的所有解产生负优化, 原因一部分同 (1), 另一部分是因为可染颜色数较多时易于求一组合法解, 但明显增大了求所有解的难度.

(4) op7.cpp 已加入足量的优化, 仍无法求得第三、第四组数据的所有解, 说明回溯法效率仅在小数据范围可行, 对于较大的数据范围, 回溯法的效率极低.

(5) 在上述的所有优化中, 求一组解运行效率最高的为 op4.cpp, 求所有解运行效率最高的为 op5.cpp.

11. 其他优化 I: 剪枝——最优性剪枝

其他优化 I: 剪枝——最优性剪枝

- 最优性剪枝: 及时停止搜不到最优解的搜索路径.
- 思想:

(1) 搜索所有方案求最小花费时, 若当前方案的花费已 $>$ 当前的最小花费, 则当前方案不是最优解, 直接回溯.

(2) 在本题中, 若要求染色所用的颜色 A 的次数不高于颜色 B 的次数, 则搜索到颜色 A 的次数 $>$ 颜色 B 的次数的方案时可直接回溯.

(3) 因本题无颜色使用次数的限制, 故不再赘述.

12. 其他优化 II: 剪枝——记忆化搜索

其他优化 II: 剪枝——记忆化搜索

- 记忆化搜索: 动态规划.
- 思想:

(1) 若不同次搜索中, 搜索到同一位置时的信息相同, 则该位置的信息可在搜索一次后记录, 后续无需重复搜索. 这也是动态规划的一种实现.

(2) 事实上, 这种思想在 op5.cpp 中有所体现, 即不同的起始颜色搜索到同一位置时的状态本质相同, 故可只搜索一次后, 用组合计数的方式统计答案.

(3) 若本题中的地图对应的图是 DAG, 则可按拓扑序、用 DAG 上 DP 的方式统计染色方案数. 因本题中的地图只是平面图, 未必是 DAG, 故不再赘述.

13. 平面图生成器

平面图生成器 (gen.cpp)

- 定义:

(1) 平面图: 若图 G 可画在平面上, s.t. G 除顶点外无边相交, 则称 G 为平面图.

(2) 图的同胚: 若图 G_1 与图 G_2 同构, 或两图可通过反复添加或删除度数为 2 的节点后变为同构的, 则称两图同胚.

- 平面图的判定: Kuratowki 定理

(1) 图 G 的平面图 iff G 不含与 K_5 或 $K_{3,3}$ 同胚的子图.
(2) 图 G 是平面图 iff G 不含可收缩到 K_5 或 $K_{3,3}$ 的子图.

- 直接用定义或 Kuratowski 定理判定平面图是困难的. 为此, 改用在坐标平面上随机生成点, 并检查是否与已有的线段相交的方式来生成平面图.

线段相交的判定

- 思路:

(1) 特殊情况: 若两线段平行, 则显然它们不相交. 若两线段重合或部分重合, 则它们相交, 且有多个交点.

线段相交的判定

- 思路:

(2) 快速排斥实验: 定义一条线段的区域为以该线段为对角线的、各边平行于坐标轴的矩形区域. 若两条线段有公共区域, 则称它们通过快速排斥实验; 否则称它们未通过快速排斥实验. 显然未通过快速排斥实验的两线段不相交, 如下图所示的两条线段.

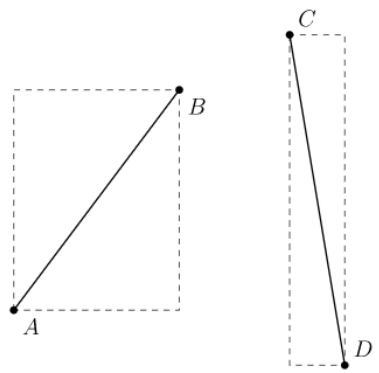

Figure 8: 未通过快速排斥实验的两线段

快速排斥实验只是两线段不相交的充分不必要条件, 还需引入其他判断方法.

线段相交的判定

- 思路:

(3) 跨立实验: 对线段 a 和线段 b, 若 b 的两端点在 a 的异侧, 且 a 的两端点在 b 的异侧, 则称线段 a 和线段 b 通过跨立实验.

注意两线段贡献但不相交时也可通过跨立实验, 故需结合快速排斥实验判定线段是否相交.

(4) 两点是否在线段的异侧可用向量的叉乘的正负判定.

线段相交的判定

- 实现:

(1) 初始时的想法是每次在平面上随机生成整点, 并与之前的整点尝试连边, 以一定概率连一定的边数, 且边数越多的概率越小. 但实际运行效果不佳.

(2) 改进: 先随机生成平面上的 n 个整点, 再给定边数 m, 每次随机两个已有的整点尝试连边. 实际运行效果良好.

(3) 为保证生成的地图连通, 以节点 1 为超级源点, 向其他节点连 $(n-1)$ 条边, 再随机连其余的 $(m-n+1)$ 条边.

(4) 为保证地图的限制不过少或过多, 要求节点数与边数的关系满足 $n-1 \leq m \leq 4n$.

14. 回溯法效率与图规模的关系

回溯法效率与图规模的关系

- 用 gen.cpp 生成不同节点数和不同边数的平面图, 用 4 种颜色染色. 用 op4.cpp 求一组合法解, 用 op5.cpp 求所有解, 统计回溯法 + 优化的效率与图规模的关系.

回溯法运行时间与图的节点数的关系

- 固定边数 $m = 2n$, 生成节点数分别为 5、10、15、20、25、30、35 的平面图 (gen1.cpp).

节点数	边数	节点最大度数	一组解 (ms)	所有解 (ms)
5	10	4	0	0
10	20	9	0	0
15	30	14	0	2
20	40	19	0	34
25	50	24	0	144
30	60	29	0	13241
35	70	34	0	260519

回溯法效率与图规模的关系

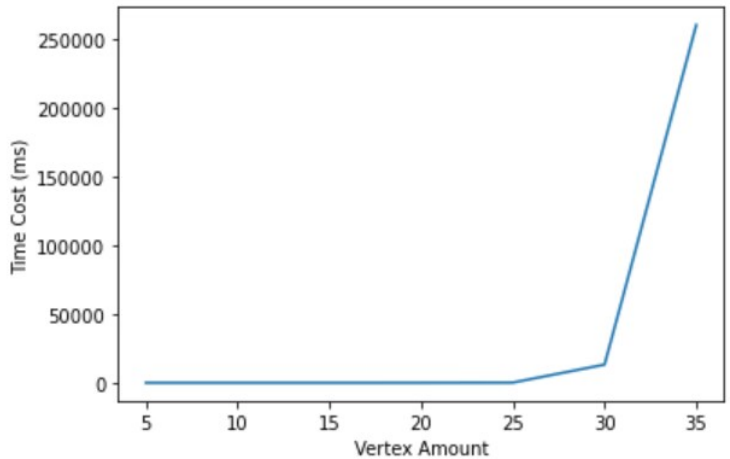

Figure 9: 回溯法的运行时间与节点数的关系

- 分析:

(1) 由图象知: 回溯法的运行时间随节点数的增大而增大, 且节点数越大, 曲线的斜率越大.

(2) 节点数增大时, 需暴力搜索的节点数增多, 计算量以阶乘级别增长.

回溯法运行时间与图的边数的关系

- 固定 $n = 20$, 生成边数分别为 20、25、30、35、40、45、50 的平面图 (gen2.cpp).

节点数	边数	节点最大度数	一组解 (ms)	所有解 (ms)
20	20	19	0	197537
20	25	19	0	22929
20	30	19	0	2283
20	35	19	0	139
20	40	19	0	38
20	45	19	0	1
20	50	19	0	0

回溯法效率与图规模的关系

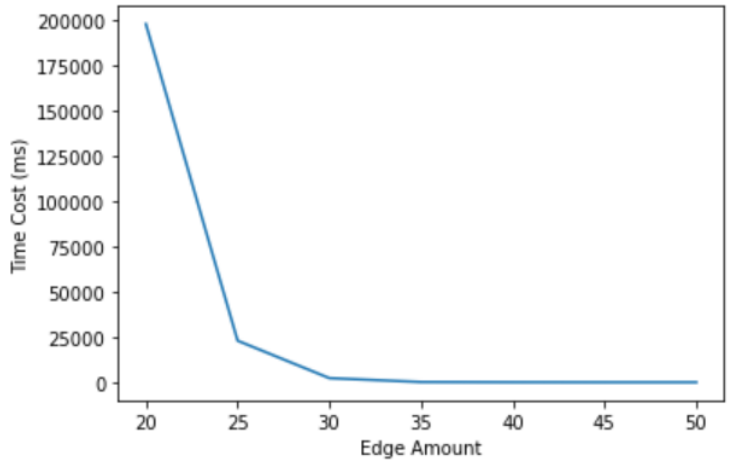

Figure 10: 回溯法的运行时间与边数的关系

- 分析:

(1) 由图象知: 对固定的节点数, 回溯法的运行时间随边数的增大而迅速减小, 且边数越接近节点数时, 运行时间越长.

(2) 对固定的节点数, 边数越少, 对节点可染的颜色的限制越少, 搜索空间越大, 运行时间越长; 边数越多, 对节点可染的颜色的限制越多, 搜索空间越小, 运行时间越短.

回溯法运行时间与图的节点的平均度数的关系

- 固定 $n = 20$, 生成随机边数的平面图 (gen3.cpp).

节点数	边数	节点平均度数	一组解 (ms)	所有解 (ms)
20	21	2.1	0	119579
20	30	3.0	0	1770
20	36	3.6	0	160
20	63	4.7	0	1
20	60	4.8	0	1
20	71	4.8	0	1
20	49	4.9	0	0

回溯法效率与图规模的关系

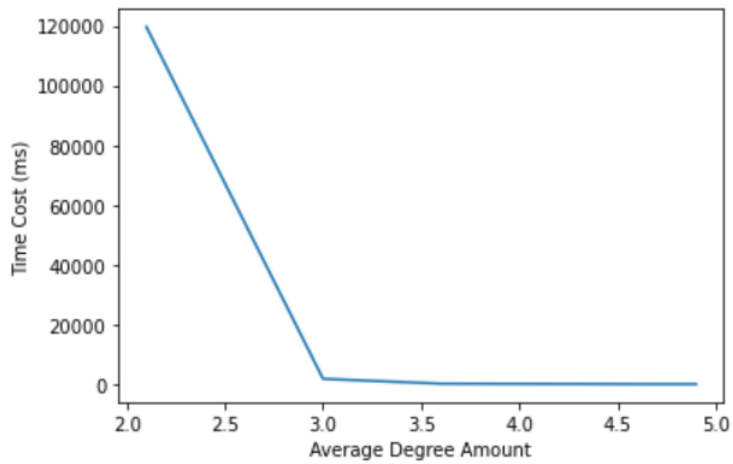

Figure 11: 回溯法的运行时间与边数的关系

回溯法效率与图规模的关系

- 分析:

(1) 由图象知: 对固定的节点数, 回溯法的运行时间随节点平均度数的增大而迅速减小, 且节点平均度数越接近 2.0 时, 运行时间越长.

(2) 节点平均度数反映图的密集程度. 对固定的节点数, 节点平均度数越小, 图越稀疏, 每个节点所连的平均边数越少, 对节点可染的颜色的限制越少, 搜索空间越大, 运行时间越长; 节点平均度数越大, 图越密集, 每个节点所连的平均边数越多, 对节点可染的颜色的限制越多, 搜索空间越小, 运行时间越短.

(3) 类似于物体的密度比其质量和密度更能反映物质本身的性质, 图的节点平均度数比其节点数和边数更能反映图本身的性质, 能更准确地反映回溯法运行效率与图的规模的关系.

谢　谢!

Thank you!