独辟蹊径

队名：寻幽(迪杰斯特拉派)
成员：季冲
王颖
学校：南京理工大学

内容提要

算法原理： 图深度优先搜索 + 剪枝 + 启发式搜索

程序实现： *C++ (Windows 10 DevC++ 5.11)*

程序说明： 输入输出格式说明、数据存储方式

结果分析： 根据实验结果分析效率

总结： 对比及可优化部分

算法原理

- 深度优先搜索
  - 从起始点开始，按照深度优先的顺序进行路径搜索
  - 记录结果：搜索到终点则得到一条路径，而后评估当前路径是否符合题目的各项约束条件，如果满足，则与当前最优解进行比较、更新。
  - 边界条件：当前路径步数>=最大步数限制
  - 细节：当访问过一个节点后并不像传统*DFS*标记节点为*visited*来防止重复访问。

算法原理

- 剪枝策略
  - 搜索过程中维护当前的最优解
  - 每次进行下一步搜索前判断一下

    If cost of 当前已走路径(不完整路径) > cost of 当前最优解 *Then* 回溯
- 启发式搜索

  下一跳节点优先级排序策略：（从高到低）

  1. *End*节点
  2. 构成绿边
  3. 绿色节点
  4. *Cost*较小

程序实现

- 图存储　　采用邻接矩阵 *vector<vector<int>> board;*
- 约束条件存储　　采用数组 *vector<int> constraints;*

程序实现

- 约束条件存储

采用数组表示 vector<int> constraints;

用-1表示非约束，0表示是约束，大于0表示约束已满足

利用Cantor pairing function[1]: $\pi : \mathbb{N}^2 \to \mathbb{N}$

$$Define: \ \pi(k_1, k_1) = \frac{1}{2}(k_1 + k_2) \times (k_1 + k_2 + 1) + k2$$

从而将edge(i, j)编码成一个唯一的自然数

例如：

edge(2, 4)为一条绿边，则constraints[π(2, 4)+N] = 0;

node(7) 为一个绿点，则constraints[7] = 0;

[1] Reference http://en.wikipedia.org/wiki/Pairing_function

程序说明

- 输入输出
  - 输入：
    - 节点数目 N（所有节点编号从0~N-1）
    - 图的邻接矩阵 $board$（$board[i][j]=cost\ of\ edge(i,j)$, $board[i][j]=-1\ if\ blocked$）
    - 起点和终点（$start$, end）
    - 红边（禁止通行，$board[i][j] = -1$）
    - 绿边（必经边，$constraints[\pi(i, j)+N] = 0$）
    - 绿点（必经点，$constraints[i] = 0$）
    - 最大跳数 $maxHop$（最多经过的储物间个数）
  - 输出：
    - 最优路径，若无解则输出 $No\ Paths!$

结果分析

- 启发式搜索 VS. 非启发式搜索

耗时(s)	非启发式	启发式	效率 启发式 VS. 非启发式
输出第一个可行解	0.239	0.125	+47.7%
输出可行解，并且就是最优解	0.741	0.184	+75.1%
输出最优解	1.128	2.580	-128.7%

(以附件中给的样例为例，并设置maxHop=12,其中的运行时间为10次运行结果取平均值得到)

最优路径：

N0 -> N2 -> N4 -> N5 -> N6 -> N7 -> N8 -> N14 -> N13 -> N12 -> N16 -> N17

结果分析

- 约束条件存储方式： 数组存储 vs. Map存储

存储方式 / 耗时(s)	Map	数组	效率 数组 vs. Map
输出第一个可行解	0.272	0.125	+54.0%
输出可行解，并且就是最优解	0.375	0.184	+50.9%
输出最优解	5.306	2.580	+51.3%

（同为启发式搜索下，两种存储方式效率比较，其中的运行时间为10次运行结果取平均值得到）

  - map存储约束条件方式：

```
map<vector<int, int>, int> greenEdges;
map<int, int> greenNodes;
```

总结

- 利用启发式搜索策略可以较快地找到可行解，但是由于启发式搜索策略带来了额外的开销，相比不用启发式策略，得出最优解的效率降低了。
- 程序实现中利用*pair function*实现用数组方式存储约束条件的方法，相比使用其他数据结构去存储，大大提高了效率。

总结

- 进一步工作
  - 解决当$maxHop$值设置得很大时，程序运行效率较慢问题
  - 改进启发式搜索策略
  - 设计更多的测试样例
  - 考虑动态规划的实现方式，用$F(start, end, cons)$表示约束集为$cons$下从节点$start$ -> end的最优解

    状态转移方程：

$$F(start,end,cons) = min\left\{F(i,end,cons_i) + cost(start,i), i \in neighbor(st)\right\}$$

附录

- 队名： 寻幽
- 成员： 季冲
- 王颖
- 学校： 南京理工大学

- 分工情况
  - 算法：季冲，王颖
  - 程序：季冲
  - 报告：季冲，王颖