人工智能作业

学生姓名：董安宁

班 学 号：111171-05

指导教师：叶雅琴

中国地质大学信息工程学院

2020 年 2 月 23 日

目录

作业一 吃豆人寻路

1、实现 DFS 和 BFS

要优化之前首先要完成最基本的版本，因此首先完成了最基础的 dfs 和 bfs 的算法，算法的流程如下图：

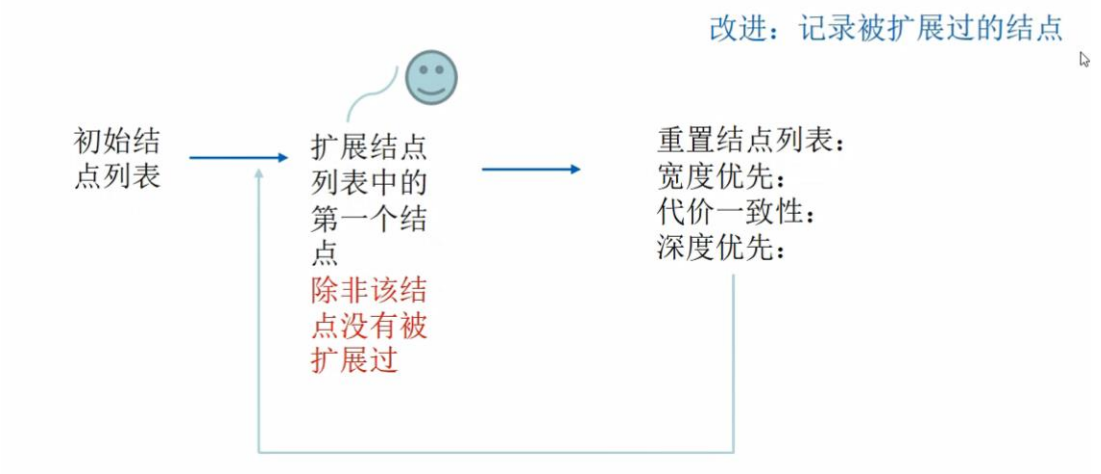


图 1 算法流程

需要注意的是，dfs 扩展节点的时候会把节点放在队列首位，并在下次立刻取出；bfs 则会把节点放在队列尾部，并且当前面的遍历完后才会取出，分别对应的是“先入后出”和“先入先出”，因此使用的数据结构分别是“栈”和“队列”。

1.1 伪代码

接下来展示 dfs 和 bfs 在该问题中的伪代码：

1.1.1 DFS 伪代码

```
定义 DFS 栈
定义结果栈
将起点放入栈
While 栈非空：
    取出栈顶节点，再放入
    标记该点被访问
    If 到达终点：
        返回结果栈
    对该节点进行扩展
    For 每个扩展节点：
        If 扩展节点没被访问：
            入栈顶
    If 该节点没有扩展节点:
        DFS 栈中删除该节点
        结果栈中删除该节点
```

1.1.2 BFS 伪代码

```
定义 BFS 队列
定义父亲节点字典
将起点放入队列
While 队列非空：
    取出队列头节点，并删掉
    标记该点被访问
    If 到达终点：
        根据字典，返回结果
    对该节点进行扩展
    For 每个扩展节点：
        If 扩展节点没被访问：
            入队尾
```

1.2 结果展示

分别对 dfs 和 bfs 进行测试，可以得到下面的结果

1.2.1 DFS 扩展结果

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 24
Pacman emerges victorious! Score: 500
```

图 1 普通 dfs 小地图

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 274
Pacman emerges victorious! Score: 380
```

图 2 普通 dfs 中地图

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 601
Pacman emerges victorious! Score: 300
```

图 3 普通 dfs 大地图

1.2.2 BFS 扩展结果

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 24
Pacman emerges victorious! Score: 502
```

图 4 普通 bfs 小地图

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 343
Pacman emerges victorious! Score: 442
```

图 5 普通 bfs 中地图

```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 830
Pacman emerges victorious! Score: 300
```

图 6 普通 bfs 大地图

接下来是地图遍历过程的展示

1.2.3 地图展示：

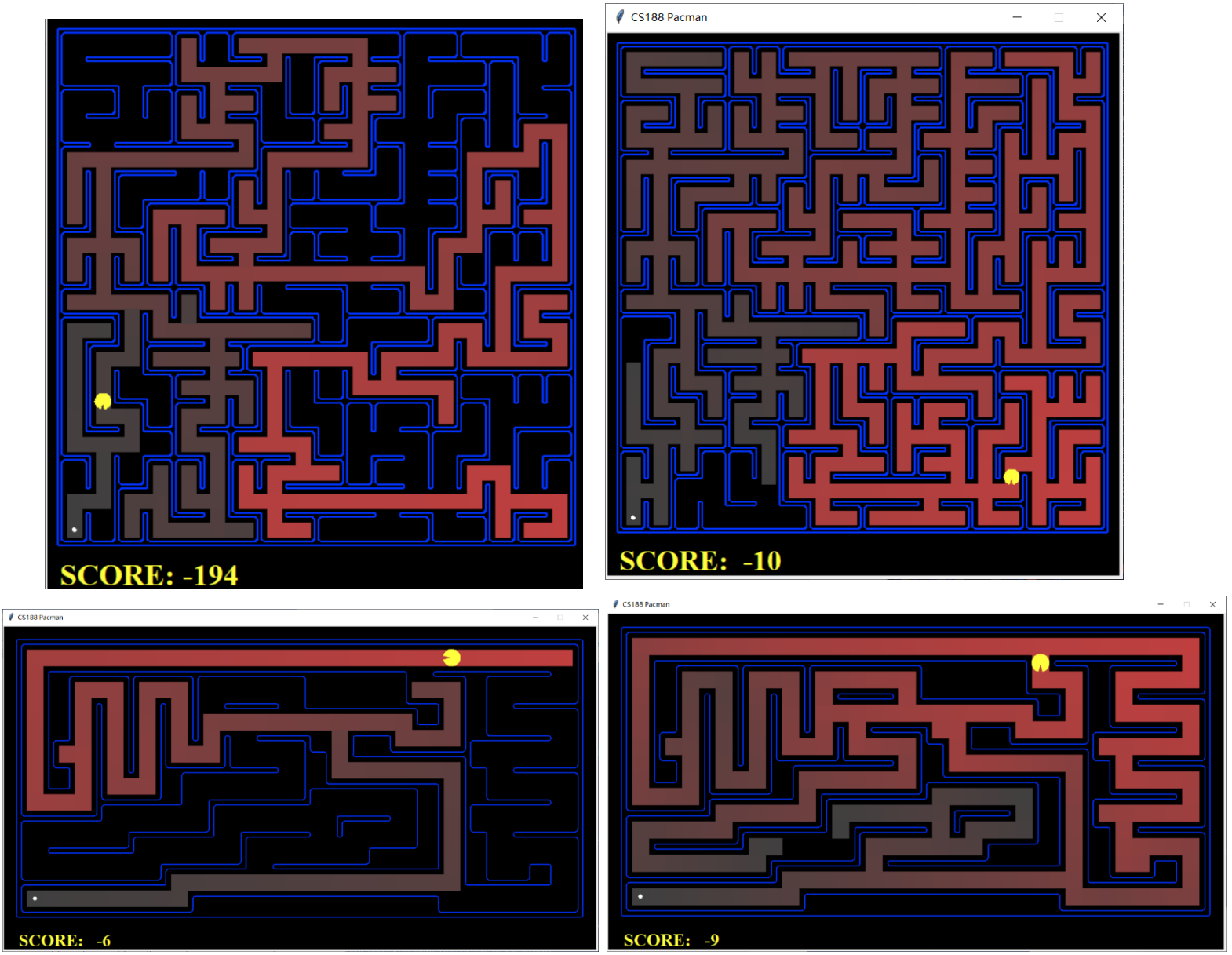


图 7 地图可视化结果

上图展示的地图结果，左边都是 dfs 的搜索方式，右边都是 bfs 的搜索方式，可以看到 dfs 的搜索策略经常是一条路走到黑，直到找到终点。而 bfs 会搜索全部可以到达的地点，直到找到终点。

这里需要说明的是，无论是 dfs 还是 bfs 我都只是单纯的设置“找到终点后就退出”，因此没有遍历完全部的路径，因为从道理上来说，dfs 和 bfs 的搜索方式都属于全部搜索的暴力式搜索，理应搜索出全部能到达终点的路径，并存起来，判断得到最优路径的。

2、对算法进行优化

在优化的过程中我询问了 111172 班的汪圣翔关于 DFS 优化的问题，最终达成共识 DFS 似乎没法进行优化…我的理解是这样的：

由于 DFS 的定义是在每次扩展的时候，直接对第一个被扩展到的节点进行向下搜索，因此 DFS 没有选择的余地。而对于 BFS 来说，它会列出所有被扩展到的节点，这样一来，就有了使用评价函数进行优化的余地了，可以在这一步对节点进行排序，将评价更高的节点进行扩展，这样就能比原来更快地找到到达终点的路径。

我一共写了三种优化方式，分别是贪心和 A*，用到了 util 中提供的优先队列的定义。这里只需要将各自的评价函数进行定义，就可以得到符合该算法的优先队列了，我的函数定义如下：

```
# 启发式函数 -- 贪心
def fun_greedy(item):
    return util.manhattanDistance(item[0],problem.goal)

# 启发式函数 -- A*
def fun_A_star(item):
    return util.manhattanDistance(item[0],problem.goal)+util.manhattanDistance(item[0],problem.

#dfs_stack = util.PriorityQueueWithFunction(fun_greedy)    # 贪心优先队列
#dfs_stack = util.PriorityQueueWithFunction(fun_A_star)    # A*优先队列
#dfs_stack = util.Stack()    # 普通dfs栈
```

图 8 启发式函数以及优先队列的定义，由于是在 DFS 的函数里面写的这个算法，因此变量名沿用了 dfs_stack 这个叫法

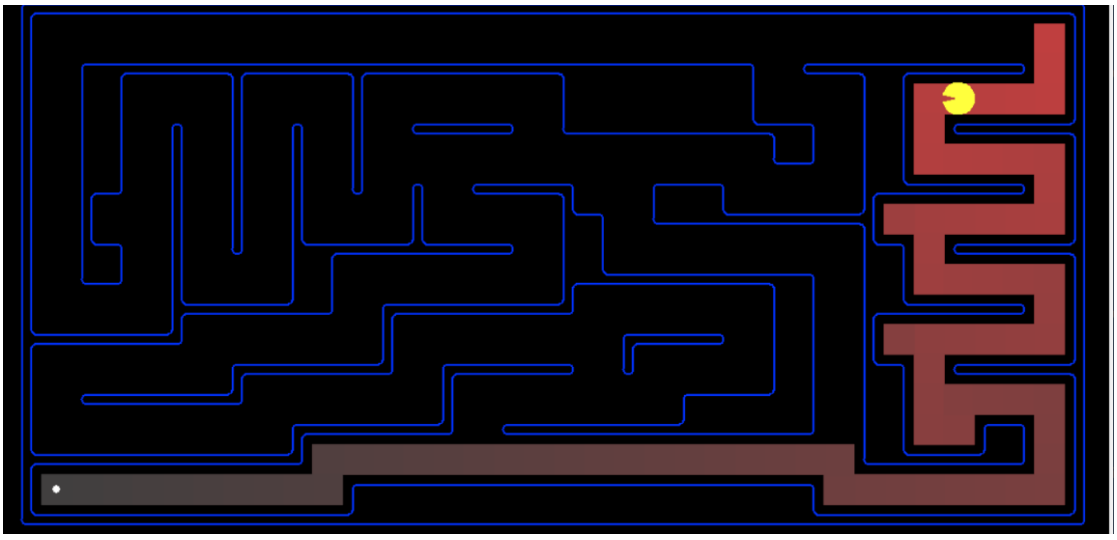
将启发函数定义翻译成中文，如下：

① 贪心：从该节点到终点的曼哈顿距离

② A*：　从该节点到终点的曼哈顿距离 + 从该节点到起点的曼哈顿距离

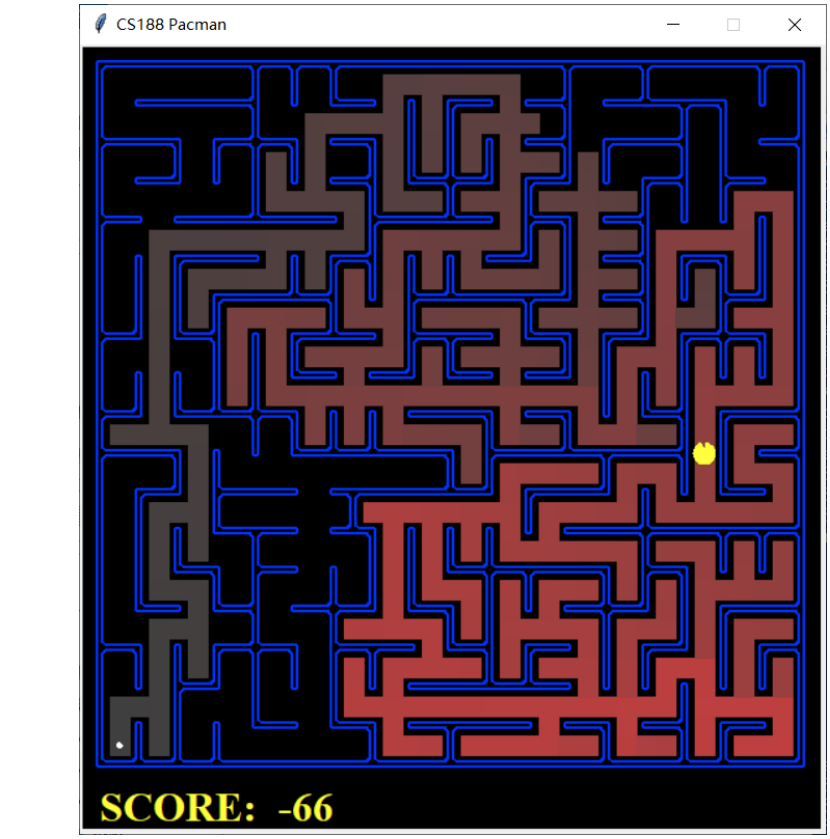
接下来是三种优化的结果展示，如下图所示：

2.1 贪心优化



```
Path found with total cost of 74 in 0.0 seconds
Search nodes expanded: 152
Pacman emerges victorious! Score: 436
```

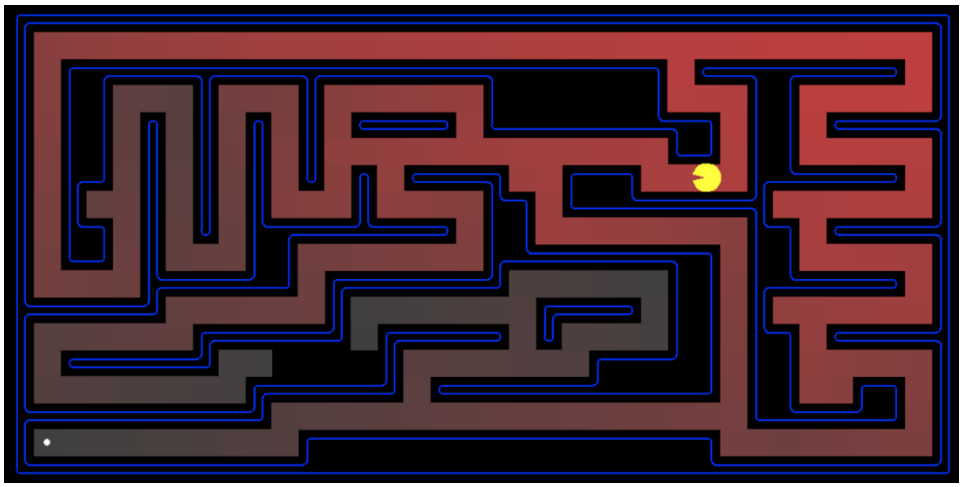
图 8 贪心优化 中地图




```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 677
Pacman emerges victorious! Score: 300
```

图 9 贪心优化 大地图

2.2 A*优化



```
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 338
Pacman emerges victorious! Score: 442
```

图 10 A*优化 中地图

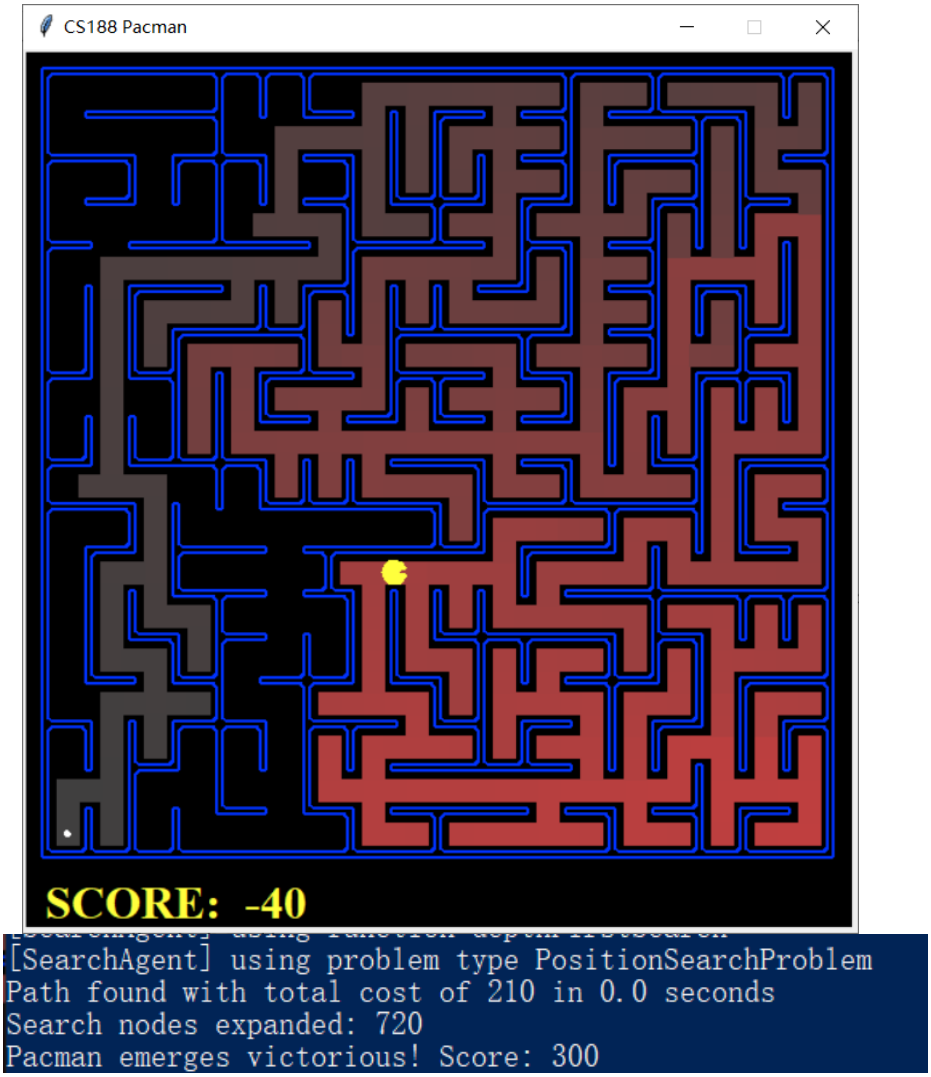


图 11 A*优化 大地图

最后附上表进行对比:

表 1 扩展节点个数

单位：个	DFS	BFS	贪心	A*
小地图	24	24	16	24
中地图	274	343	152	338
大地图	601	830	677	720

首先是扩展节点的个数。先对 DFS 和 BFS 进行对比，可以明显看到 BFS 扩展的节点都要大于等于 DFS 的节点个数，因为 BFS 的范围更广，更倾向于搜索全局的所有可能性，而 DFS 只需要一个正确的就够了。

贪心算法相比其他的算法来说，扩展节点的个数更少，因为在贪心的代价函数下，我们只会选取离目标越来越近的节点，不会去拓展多余的节点。

A*算法的拓展节点个数和 BFS 的差不多一样多，主要是因为 A*的任务是一定要找到最近的一条路径，因此会更多的探索可能到达终点的路径。而它比 BFS 少的原因是因为 A*存在一个选择过程，只会扩展更加靠近终点的节点，而不会想 BFS 一样一股脑的全都扩展。

表 2 路径代价

单位：长度	DFS	BFS	贪心	A*
小地图	10	8	8	8
中地图	130	68	74	68
大地图	210	210	210	210

接下来是路径代价的对比。可以看到和 DFS 相比，BFS 的算法更能找到距离短的路径（毕竟它探索的节点更多嘛）。贪心算法有时能找到最优的路径，有时找不到，因为贪心策略可能不会适合所有的问题情况。而 A*算法总是能找到最短的路径，这也是它的算法性质所导致的。

3、总结

本次实习虽然是我认为比较简单的 DFS 和 BFS，但在上手写代码的时候发现还是过于理论王者了…在实践的时候会面对很多在学理论知识的时候不会注意到的问题，比如出入栈的顺序，以及如何得到结果路径，等等。通过本次实习，我也对这两种搜索算法更加的熟练，并且更加理解了“启发式”搜索的意义，确实相比没有目标的 DFS 和 BFS 来说，启发函数将搜索的过程大大优化，能找到更好的方式来达到终点。