人工智能实验报告 LAB4

（2021学年秋季学期）

课程名称：Artificial Intelligence

教学班级	计科2班	专业（方向）	计算机科学与技术
学号	19335174	姓名	施天予

无信息搜索

一、 实验题目

无信息搜索，本人选择的是一致代价搜索

二、 实验内容

1、算法原理

搜索问题的定义

解决搜索问题时，首先需要对搜索问题进行形式化表述。搜索问题从以下几个方面表述：

- 状态空间：表示需要进行搜索的空间
- 动作：表示从一个状态到达另一个状态
- 初始状态：表示当前的状态
- 目标：表示需要达到的目标的状态
- 启发方法：用于指挥搜索的前进方向的方法
- 问题的解：一个从初始状态到达目标状态的动作序列

搜索问题可以用状态空间树表示，每个节点对应着状态空间中的一种状态。节点的父节点表示产生该状态的上一个状态，父节点生成子节点时需要记录生成节点所采取的行动与代价。

搜索算法的性能

- 完备性：当问题有解时是否一定能找到解

- 最优性：搜索策略是否一定能找到最优解
- 时间复杂度：找到解所需要的时间，又称为搜索代价
- 空间复杂度：执行搜索过程中需要多少内存空间

一致代价搜索

UCS算法是BFS算法的变种。边界中，按路径的成本升序排列，总是扩展成本最低的那条路径。当每种动作的成本是一样的时候，和宽度优先是一样的。假设每个动作的成本都大于某个大于0的常量，所有成本较低的路径都会在成本高的路径之前被扩展。给定成本，该成本的路径数量是有限的；成本小于最优路径的路径数量是有限的，最终，我们可以找到最短的路径，因此具备完备性和最优性。

对于一致代价搜索，当最优解的成本为C*，上述最低代价为 s，则有：

- 时间复杂度：$O(b^{[C^*/s]+1})$
- 空间复杂度：$O(b^{[C^*/s]+1})$

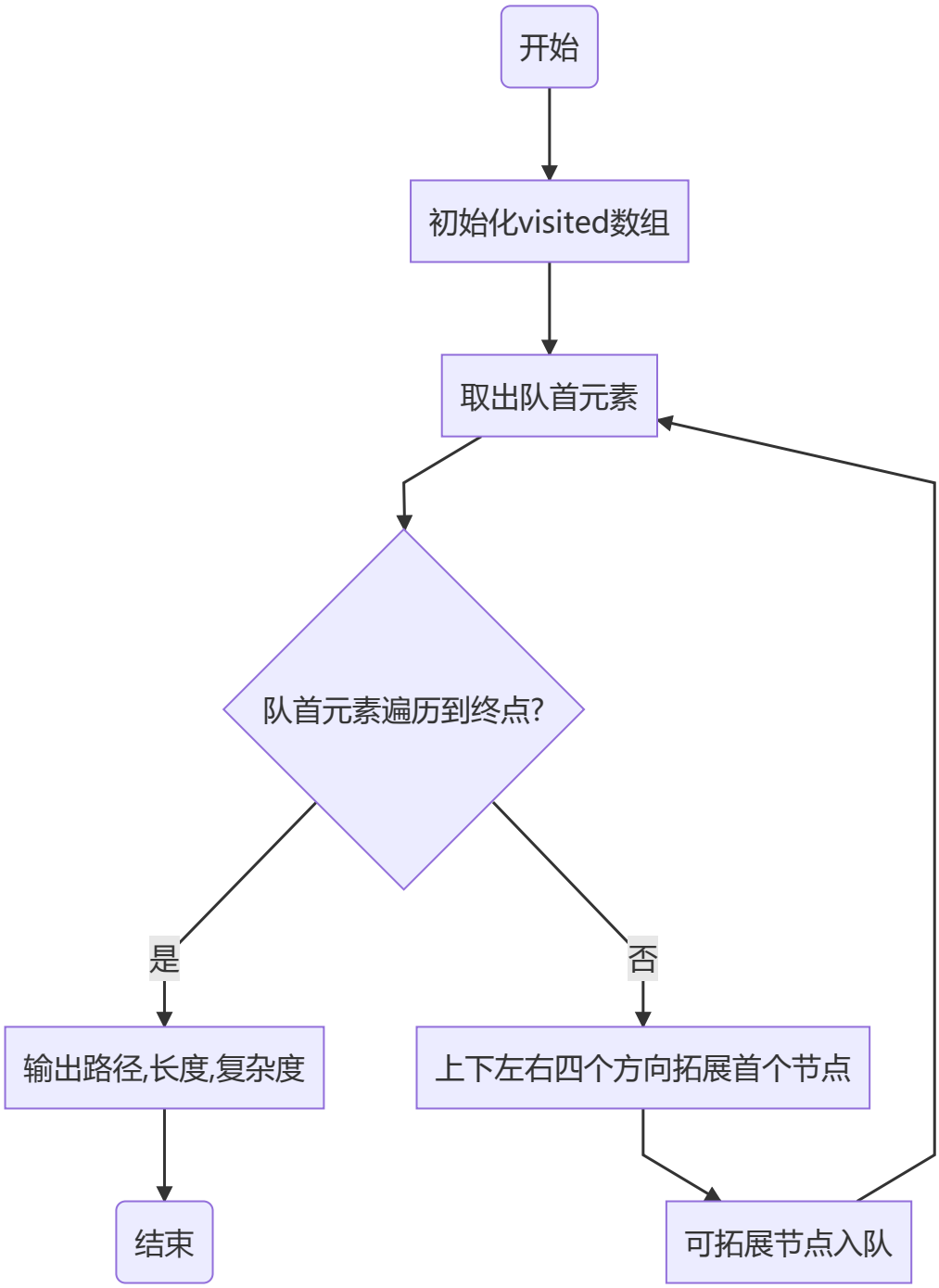
宽度优先搜索

这次的一致代价搜索其实是一个宽度优先搜索。宽度优先搜索用一个先进先出的队列实现，节点的拓展顺序与目标节点的位置无关。从状态空间树上看，宽度优先搜索的拓展顺序是按树的层次顺序来进行的。这样一来，短的路径会在任何比它长的路径之前被遍历，因此宽度优先搜索具有完备性和最优性。

定义b为问题中一个状态最大的后继状态个数，d为最短解的动作个数，则有：

- 时间复杂度：$1+b+b^2+\ldots+b^d+b(b^d-1)=O(b^{d+1})$
- 空间复杂度：$b(b^d-1)=O(b^{d+1})$

2、流程图和伪代码




```
Function USC
Input: maze, s, e   /*迷宫，起点，终点*/
Output: best, shortestLength, complex/*路径图，最短距离，时空复杂度（时间和空间相同）*/
    初始化best，shortestLength，x，y（起始位置），complex，visited数组
    将(x,y)加入队列q
    while 队列q不为空:
        (x,y)从队列q取出
        if (x,y) == e then break end if
        count += 1  /*增加复杂度*/
        /*遍历上下左右，visited[x][y]保存该点向起点移动路径的方向*/
        explore(x-1, y, 'D', visited, q)
        explore(x+1, y, 'U', visited, q)
        explore(x, y-1, 'R', visited, q)
```

```
        explore(x, y+1, 'L', visited, q)
    end while
    /*回到起点生成最短路径*/
    (x, y) = e
    while (x, y)不为起点s:
        shortestLength += 1/*增加最短路径值*/
        /*根据方向往回走*/
        if visited[x][y] == 'D' then x = x + 1 end if
        if visited[x][y] == 'U' then x = x - 1 end if
        if visited[x][y] == 'L' then y = y - 1 end if
        if visited[x][y] == 'R' then y = y + 1 end if
    end while
    将路径从maze写入字符串best
    return best, shortestLength, complex/*返回路径图，最短距离，时空复杂度*/
```

3、关键代码展示

测试(x, y)位置是否可走

```
def validQ(x,y):
    if x < 0 or x >= len(maze) or y < 0 or y >= len(maze[0]) or maze[x][y] == '.':
        return False
    else:
        return True
```

判断下一步(x,y)返回起点的路径的方向，加入队列q

```
def explore(x, y, d, visited, q):
    if validQ(x,y) and visited[x][y] == 0:
        visited[x][y] = d
        q.append((x,y))
```

一致代价搜索，因为每一步代价相同所以其实是BFS

```
def UCS(maze, s, e):
    best = ""
    shortestLength = 0
    x, y = s[0], s[1]
    complex = 1 # 复杂度
    # 初始化
    q = [(x,y)]
    visited = [[0] * (len(maze[0])) for _ in range(len(maze))]
    visited[x][y] = -1
    # 遍历上下左右，visited[x][y]保存该点向起点移动路径的方向
```

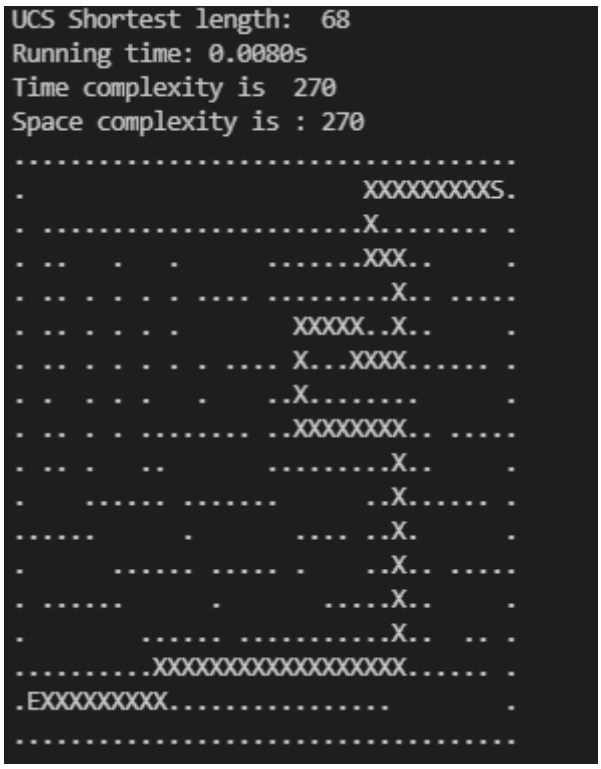
```
    while len(q) > 0:
        (x,y), q = q[0], q[1:]
        if (x,y) == e:
            break
        complex += 1
        explore(x-1, y, 'D', visited, q)
        explore(x+1, y, 'U', visited, q)
        explore(x, y-1, 'R', visited, q)
        explore(x, y+1, 'L', visited, q)
    # 回到起点生成最短路径
    (x, y) = e
    while (x, y) != s:
        shortestLength += 1
        if visited[x][y] == 'D':
            x = x + 1
        elif visited[x][y] == 'U':
            x = x - 1
        elif visited[x][y] == 'L':
            y = y - 1
        elif visited[x][y] == 'R':
            y = y + 1
        maze[x] = maze[x][:y] + 'X' + maze[x][y+1:]
    # 将路径写入字符串best
    maze[s[0]] = maze[s[0]][:s[1]] + 'S' + maze[s[0]][s[1]+1:]
    for i in range(len(maze)):
        best += maze[i] + "\n"
    # 返回路径图、最短距离、复杂度
    return best, shortestLength, complex
```

三、 实验结果及分析

1、实验结果展示示例

为显示清晰，1用点号表示，0用空格表示，路径用X表示

如图最短路径为68，运行时间0.0080s，时间复杂度和空间复杂度270



2、算法性能分析

完备性	最优性	时间复杂度	空间复杂度
是	是	270	270

完备性：如果每个路径的成本s都大于0，由于如果存在路径到达目标点，就一定能搜索出一条路径。

最优性：如果每个路径的成本s都大于0，若将任意点作为目标点，UCS都能保证第一次探索到目标点的时候是最小的路径，否则当前边界会先选择探索另外一个点而不是目标点。

时间复杂度和空间复杂度： 按照公式这两者应该是相同的。在实验的迷宫的问题，每个点最多只能向四个方向拓展，也就是b为4，则时空复杂度为$O(4^{d+1})$（d为到终点的最优解需要的步数），但其实因为有些路堵住了所以不会4个方向都搜索。本次搜索算法的复杂度（拓展节点数）是270，之后会与启发式搜索的算法进行比较。

启发式搜索

一、 实验题目

启发式搜索，本人选择的是$A*$搜索

二、 实验内容

1、算法原理

启发式搜索

又称有信息的搜索，它利用问题所拥有的启发信息来引导搜索，达到减少搜索范围，降低问题复杂度的目的。对于一个具体的问题，构造一个专用于该领域的启发式函数$h(n)$, 该函数用于估计从节点n到达目标节点的成本 ， 要求对于所有满足目标条件的节点n有：$h(n) = 0$。

$A*$搜索

定义评价函数$f(n) = g(n) + h(n)$，其中$g(n)$是从初始节点到达节点n的路径成本，$h(n)$是从n节点到达目标节点的成本的启发式估计值。 因此，$f(n)$是经过节点n从初始节点到达目标节点的路径成本的估计值。利用节点对应的$f(n)$来对边界上的节点进行排序。

$A*$搜索速度较快，但需要维护开启列表和关闭列表，并且需要反复查询状态，因此空间复杂度是指数级的。其具体步骤为：

1. 将起点当做待处理点加入开启列表
2. 搜索所有可以拓展的节点并加入开启列表，计算这些节点的$f(x)$
3. 将起点从开启列表中移动到关闭列表
4. 从开启列表中寻找$f(x)$最小的节点，将相邻节点加入开启列表，并将该节点移动到关闭列表。若相邻节点已经在开启列表则更新它的$g(x)$值
5. 若找到目标节点（找到解）或开启列表为空（无解）则退出，否则重复步骤 4 。

启发式函数的设计

- 可采纳性：假设每个状态转移（每条边）的成本是非负的，而且不能无穷地小，假设$h^*(n)$是从节点n到目标节点的最优路径成本（不连通则为无穷大）。当对于所有的节点n，满足$h(n) <= h^*(n)$时，则称$h(n)$是可采纳的。也就是说，可采纳的启发式函数低估了当前节点到达目标节点的成本，使得实际成本最小的最优路径能够被选上。因此，对于任何目标节点g，有$h(g) = 0$
- 一致性（单调性）：对任意节点$n1$和$n2$有：$h(n1) \leq cost(n1 \rightarrow n2) + h(n2)$。满足一致性的启发式函数也一定满足可采纳性，大部分的可采纳的启发式函数也满足一致性/单调性。

可采纳性意味着最优性，最优解一定会在所有成本大于最优开销的路径之前被扩展到。因为一致性，搜索第一次扩展到某个状态，就是沿着最小成本的路径进行扩展的，而且在遍历节点n时，所有f值小于f(n)的节点都已经被遍历过了 。

也就是说，启发式函数的可采纳性和一致性是启发式搜索完备性和最优性的保证。

不同的启发式函数

曼哈顿距离：在正方形网络中，允许向4邻域移动。其中，D可设置为方格间移动的最短距离，`s`表示当前节点，`e`表示目标节点。

$$h(n) = D * (abs(s.x - e.x) + abs(s.y - e.y))$$

欧式距离：在正方形网络中，允许想任意角度移动：

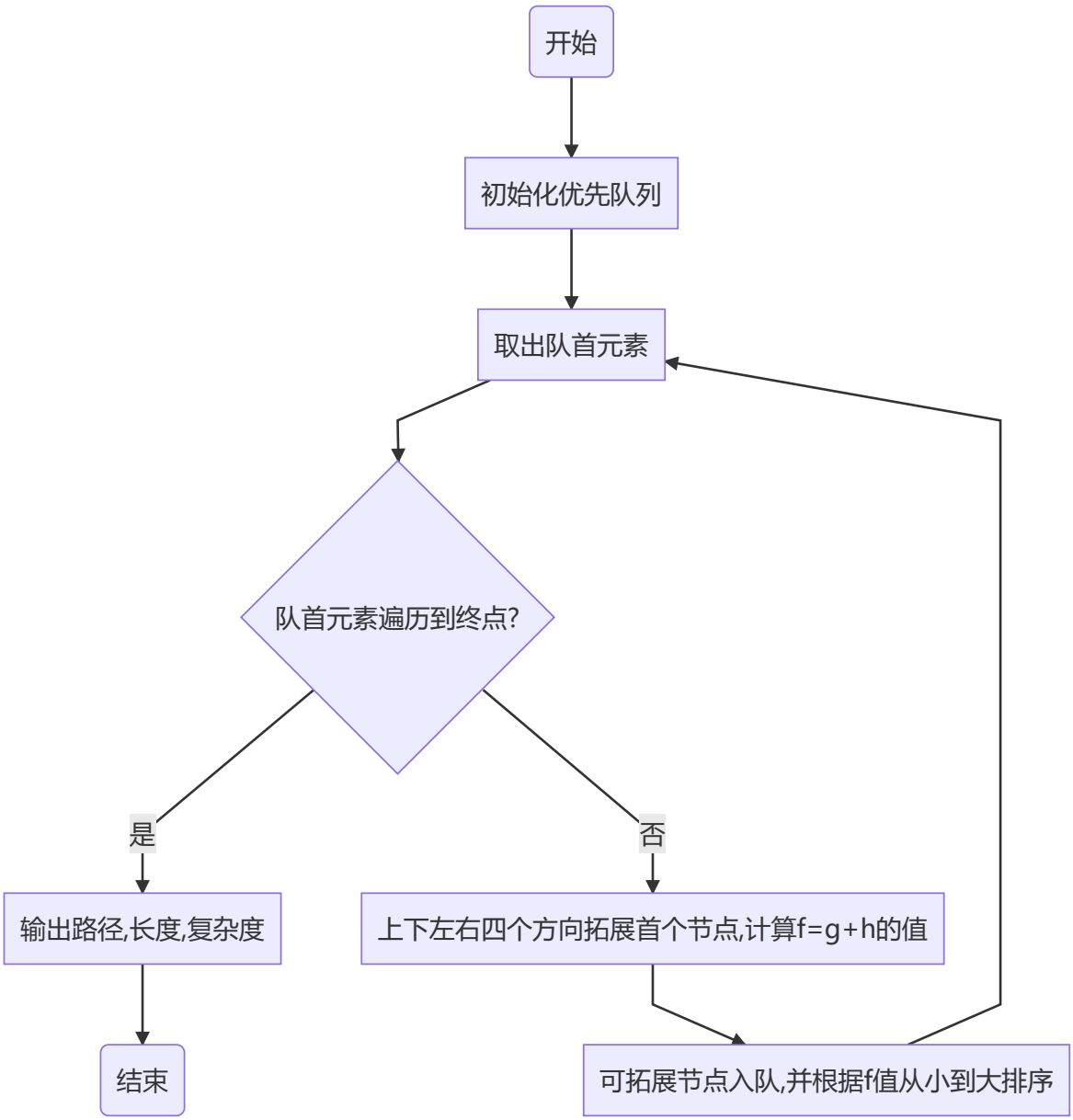
$$h(n) = D * \sqrt{(s.x - e.x)^2 + (s.y - e.y)^2}$$

对角线距离（切比雪夫距离）：在正方形网络中，允许朝着对角线方向移动（适合8邻域）：

$$h(n) = D * max(abs(s.x - e.x), abs(s.y - e.y))$$

在本次迷宫问题中，只能向4邻域方向移动，所以只能用曼哈顿距离或者欧氏距离来作启发式函数，不可使用对角线距离。

2、流程图和伪代码




```
Function AStar
Input: maze, s, e   /*迷宫，起点，终点*/
Output: best, shortestLength, complex/*路径图，最短距离，时空复杂度（时间和空间相同）*/
    初始化best，shortestLength，x，y（起始位置），complex
    初始化prev(节点的前一个节点)和prevCost(前一个节点的路径长度)
    将(x,y)加入优先队列q
    prev[s] = None
    prevCost[s] = 0
    while 优先队列q不为空:
        (x,y)从队列q取出队首元素
        if (x,y) == e then break end if
        count += 1  /*增加复杂度*/
        /*遍历4邻域节点*/
```

```
        for pos in (x,y)的4邻域):
            g = prevCost[(x,y)] + 1
            if pos节点可走 and (pos未遍历 or g < prevCost[pos]) then
                prevCost[pos] = g
                f = g + h(pos, e, 1)
                将pos加入到优先队列q
                prev[pos] = (x, y)
            end if
        end for
    end while
    # 回到起点生成最短路径
    p = e
    while prev[p] != None:
        shortestLength += 1
        x, y = p[0], p[1]
        maze[x] = maze[x][:y] + '#' + maze[x][y+1:]
        p = prev[p]
    end while
    将路径从maze写入字符串best
    return best, shortestLength, complex/*返回路径图，最短距离，时空复杂度*/
```

3、关键代码展示

A*搜索部分代码与一致代价搜索相同，只展示新的部分

启发式函数：Lp距离，可根据p的不同选择不同的距离，一般取p=1曼哈顿距离

```
def h(pos, e, p=1):
    return ((abs(pos[0] - e[0]))**p + (abs(pos[0] - e[0]))**p)**(1/p)
```

A*搜索算法，根据每个节点$f = g + h$值的不同，用一个优先队列从小到大排序存储，f值小的先遍历

```
def AStar(maze, s, e):
    best = ""
    shortestLength = 0
    x, y = s[0], s[1]
    complex = 1 # 复杂度
    # 初始化
    q = queue.PriorityQueue()
    q.put((0,(x,y)))
    prev = {}
    prevCost = {}
    prev[s] = None
    prevCost[s] = 0
```

```
    # 寻找终点
    while not q.empty():
        complex += 1
        _, (x, y) = q.get() # 取出优先队列队首元素
        if (x, y) == e:
            break
        # 遍历4邻域节点
        for pos in [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]:
            g = prevCost[(x,y)] + 1
            # 如果节课可走，并且节点未遍历或节点有更短的走法
            if validQ(pos[0], pos[1])and ((pos not in prev) or g < prevCost[pos]):
                prevCost[pos] = g
                f = g + h(pos, e, 1)
                q.put((f, pos)) # 入队
                prev[pos] = (x, y)
    # 回到起点生成最短路径
    p = e
    while prev[p] != None:
        shortestLength += 1
        x, y = p[0], p[1]
        maze[x] = maze[x][:y] + '#' + maze[x][y+1:]
        p = prev[p]
    # 将路径写入字符串best
    maze[e[0]] = maze[e[0]][:e[1]] + 'E' + maze[e[0]][e[1]+1:]
    maze[s[0]] = maze[s[0]][:s[1]] + 'S' + maze[s[0]][s[1]+1:]
    for i in range(len(maze)):
        best += maze[i] + "\n"
    # 返回路径图、最短距离、复杂度
    return best, shortestLength, complex
```

三、 实验结果及分析

1、实验结果展示示例

为显示清晰，1用点号表示，0用空格表示，路径用#表示

采用曼哈顿距离：最短路径为68，运行时间0.0100s，时间复杂度和空间复杂度244

采用欧式距离：最短路径为68，运行时间0.0140s，时间复杂度和空间复杂度248

2、算法性能分析

完备性	最优性	时间复杂度	空间复杂度
$h(x)$有一致性则有完备性	$h(x)$有可接纳性则有最优性	244	244

完备性：如果启发式函数具有一致性，则算法具有完备性，一定能搜索出一条路径。

最优性：如果启发式函数具有可接纳性，则算法具有最优性，保证第一次探索到目标点的时候是最小的路径。

时间复杂度和空间复杂度： 当启发式函数h(n)=0的时候，对于任何n这个启发式函数都是单调的，此时就变成了一致代价搜索。

因$A*$ 搜索的时间复杂度和空间复杂度低于一致代价搜索，但$A*$ 搜索的复杂度依然是指数级的，降低并不多。

3、算法性能对比

对比三种方法：

指标	UCS	$A*$搜索(曼哈顿距离)	$A*$搜索(欧式距离)
时间复杂度和空间复杂度	270	244	248
运行时间	0.0080s	0.0100s	0.0140s

- 时间复杂度和空间复杂度因为公式相同，所以设置的计算方法相同。每从队列取出一个队头元素，复杂度加1。
- 比较三种算法，$A*$搜索的复杂度低于一致代价搜索，说明启发式函数是有效的，但因为搜索的深度不大，所以复杂度的降低效果不是特别明显。因为本次迷宫只能在节点的4邻域移动，所以采用曼哈顿距离作为启发式函数效果更好，优于欧式距离。
- 对于实际的运行时间，因为本次迷宫较为简单，而启发式搜索需要计算启发式函数，并且维护一个优先队列，所以一致代价搜索反而稍微更快一些。

四、思考题

不同策略优缺点，适用场景的理解和认识。

	优点	缺点	适用场景
一致代价搜索	BFS的升级版，具有最优性和完备性。终点深度较小时速度较快	空间复杂度指数级，边界队列需要保存指数级的节点数量	最短路径问题
迭代加深搜索	DFS的升级版，具有最优性和完备性。空间复杂度小	开销很大，迭代数量为成本数值构成的集合大小。搜索深度不高时效率较低。	空间复杂度较高的问题
双向搜索	具有最优性和完备性，明显提高单项BFS的速度（深度减半）	空间复杂度指数级，要维护两个边界队列	最短路径问题
$A*$搜索	BFS的升级版，具有最优性和完备性（取决于启发式函数），速度较快	空间复杂度指数级，要维护"开启列表"和"关闭列表"，并反复检查状态	最短路径问题
$IDA*$搜索	迭代加深搜索的升级版，空间复杂度较低（不用维护表）	不具有最优性，需要对终点位置有已知有效信息	空间复杂度较高的问题