华东师范大学数据科学与工程学院实验报告

课程名称：当代人工智能	**年级**：2020	**上机实践成绩**：
指导教师：李翔	**姓名**：李昕原	**学号**：10205501425
上机实践名称：A*算法		**上机实践日期**：3.24
上机实践编号：	**组号**：	**上机实践时间**：

一、实验任务

使用 A*算法解决两道题目：

Q1:森林宝石的秘密通道

在一个神秘的森林里，有一块由 9 个小方格组成的魔法石板。石板上有 8 个宝石，每个宝石上刻有 1-8 中的一个数字（每个数字都不重复）。石板上还有一个空位，用 0 表示。通过移动空位周围的宝石，你可以改变宝石的位置。传说中，当宝石按照某个特定的顺序排列时（本题设为 1 3 5 7 0 2 6 8 4），魔法石板将会显露出通往一个宝藏的秘密通道。现在，你站在这块魔法石板前，需要找到一种最少步骤的移动方法，将宝石排列成目标顺序。为了解开这个谜题，请使用 A*算法来设计一个程序，帮助你从初始状态成功解锁秘密通道。

Q2:杰克的金字塔探险

在一个神秘的王国里，有一个名叫杰克的冒险家，他对宝藏情有独钟。传说在那片广袤的土地上，有一座名叫金字塔的奇迹，隐藏着无尽的财富。杰克决定寻找这座金字塔，挖掘隐藏在其中的宝藏。金字塔共有 N 个神秘的房间，其中 1 号房间位于塔顶，N 号房间位于塔底。在这些房间之间，有先知们预先设计好的 M 条秘密通道。这些房间按照它们所在的楼层顺序进行了编号。杰克想从塔顶房间一直探险到塔底，带走其中的宝藏。然而，杰克对寻宝路线有着特别的要求：（1）他希望走尽可能短的路径，但为了让探险更有趣和挑战性，他想尝试 K 条不同的较短路径。（2）他希望在探险过程中尽量节省体力，所以在选择通道时，他总是从所在楼层的高处到低处。现在问题来了，给你一份金字塔内房间之间通道的列表，每条通道用（X_i, Y_i, D_i）表示，表示房间 X_i 和房间 Y_i 之间有一条长度为 D_i 的下行通道。你需要计算出杰克可以选择的 K 条最短路径的长度，以便了解他在探险过程中的消耗程度。

二、实验环境

Python

三、算法设计

两道题目均通过 Python 使用 A*算法解决，基于的数据结构为堆，调用 Python 的 heapq 包，通过处理堆来维护一个优先级队列（由于题目不涉及阻塞、多线程等，因此选用更加轻量级、高性能的 heapq 而不是 Priorityqueue），主要使用了 heappush 与 heappop 两个方法：

- heapq.heappush(heap, item)：将一个元素 item 添加到堆 heap 中，保持堆的性质不变。
- heapq.heappop(heap)：弹出并返回堆 heap 中的最小元素，保持堆的性质不变

题目一的代码如下：

```
import heapq
class Puzzle:
    def __init__(self, state, goal):
        self.state = state
        self.goal = goal

    # 排序方法
    def __lt__(self, other):
        return self.priority() < other.priority()

     # 计算当前状态和目标状态之间的曼哈顿距离作为启发式函数
     def heuristic(self, current, goal):
         distance = 0
         for i in range(3): #3*3
             for j in range(3):
                 current_pos = current.index(str(i * 3 + j))
                 goal_pos = goal.index(str(i * 3 + j))
                 current_x, current_y = current_pos // 3, current_pos
  % 3
                 goal_x, goal_y = goal_pos // 3, goal_pos % 3
                 distance += abs(current_x -
  goal_x) + abs(current_y - goal_y)
         return distance

     def priority(self):
         # 计算优先级
         return len(self.state_history) + self.heuristic(self.state,
 self.goal)

     def solved(self):
         # 判断当前状态是否为目标状态
         return self.state == self.goal

     def get_next(self):
         # 获取所有可能的下一步状态
         next_puzzles = []
         zero_pos = self.state.index('0')
         for offset in [-3, -1, 1, 3]:
             new_pos = zero_pos + offset
             if new_pos < 0 or new_pos >= 9:
                 continue
             if zero_pos in [2, 5, 8] and offset == 1:
                 continue
             if zero_pos in [0, 3, 6] and offset == -1:
                 continue
```

```
43.                 new_state = list(self.state)
44.                 new_state[zero_pos], new_state[new_pos] = new_state[new_
   pos], new_state[zero_pos]
45.                 next_puzzles.append(Puzzle(''.join(new_state), self.goal))

46.             return next_puzzles
47.
48.         def a_star(self):
49.             self.state_history = [] #记录状态历史
50.             frontier = [self]
51.             while frontier:
52.                 current = heapq.heappop(frontier)
53.                 if current.solved():
54.                     return len(current.state_history)
55.                 for puzzle in current.get_next():
56.                     if puzzle.state not in current.state_history:
57.                         puzzle.state_history = current.state_history + [
   current.state]
58.                         heapq.heappush(frontier, puzzle)
59.             return -1
60.
61.     if __name__ == '__main__':
62.         init = input().strip()
63.         goal = '135702684'
64.         # 创建puzzle对象并调用A*算法得到结果
65.         puzzle = Puzzle(init, goal)
66.         result = puzzle.a_star()
67.         print(result)
```

通过 Class Puzzle：定义了该八数码问题的状态和操作，包含以下方法：

- init(self, state, goal)：初始化 Puzzle 对象的状态和目标状态。
- lt(self, other)：定义了对象之间的比较方法，将对象按照优先级从小到大排序。
- heuristic(self, current, goal)：计算曼哈顿距离作为启发式函数。
- priority(self)：计算当前状态的优先级，即 f(n) = g(n) + h(n)
- solved(self)：判断当前状态是否为目标状态。
- get_next(self)：获取所有可能的下一步状态。
- a_star(self)：使用 A*算法求解八数码问题。

其中的 get_next()方法通过将空格 0 向上下左右移动来生成当前状态的所有可能的下一步状态。该方法检查移动是否有效（即 0 越过边界），并创建一个新的 Puzzle 对象，更新状态。a_star()方法实现了 A*算法，使用了一个优先队列 frontier 来存储待扩展的节点。在每次迭代中，从优先队列中取出优先级最高的节点进行扩展(f(n) = g(n) + h(n)最小)。如果该节点为目标状态，则返回状态历史的长度，即从起始状态到达目标状态的步数。如果该节点不是目标状态，则将其所有可能的下一步状态加入到优先队列中，并记录状态历史。通过这种方式，A* 算法可以保证总代价最小的节点被先访问，从而找到最优解。

题目二代码如下：

```python
import heapq

class Node:
    def __init__(self, room, cost, estimate):
        self.room = room
        self.cost = cost
        self.estimate = estimate

    def __lt__(self, other):
        if self.cost + self.estimate == other.cost + other.estimate:
            # 当估计距离相等时，按照节点编号升序排序
            return self.room < other.room
        else:
            return self.cost + self.estimate < other.cost + other.estimate

def a_star(N, M, K, edges):
    # 邻接表表示图
    graph = [[] for _ in range(N+1)]
    for x, y, d in edges:
        graph[x].append((y, d))

    # 启发式函数
    def heuristic(room):
        # 如果节点没有出边，估计距离设为0
        if not graph[room]:
            return 0
        else:
            # 否则使用每条边的权重作为估计距离
            return min(edge[1] for edge in graph[room])

    # A*算法
    k_shortest_paths = []
    start = Node(1, 0, heuristic(1))
    open_set = [start]
    while open_set and len(k_shortest_paths) < K:
        visited = set()
        current = heapq.heappop(open_set)
        if current.room == N:
            k_shortest_paths.append(current.cost)
        for neighbor, cost in graph[current.room]:
            if neighbor in visited:
                continue
            next_node = Node(neighbor, current.cost + cost, heuristic(neighbor))
            heapq.heappush(open_set, next_node)
        visited.add(current.room)

```

```
47.      # 补充不足K条路径的情况
48.      while len(k_shortest_paths) < K:
49.          k_shortest_paths.append(-1)
50.
51.      return k_shortest_paths
52.
53.  if __name__ == '__main__':
54.      N, M, K = map(int, input().split())
55.      edges = [tuple(map(int, input().split())) for _ in range(M)]
56.      result = a_star(N, M, K, edges)
57.      for path_length in result:
58.          print(path_length)
```

首先通过输入的内容构建有向图，然后定义了一个 Node 类，用于表示图中的节点。该类有三个属性：room 表示节点编号，cost 表示到该节点的路径长度，estimate 表示该节点到终点的估计距离。

在 a_star()方法中，用邻接表表示图后定义启发式函数：选择该节点的所有出边中权重最小的那条边的权重作为估计距离（没有出边则估计距离为 0）。接下来开始使用 A* 算法求解前 K 短路径问题，先定义一个列表 k_shortest_paths，用于存储前 K 短路径的长度，接着初始化起点 start，并将其加入到优先队列 open_set 中。然后进入一个循环，不断从优先队列中取出距离起点最短的节点 current，并根据该节点的出边更新优先队列，即遍历 current 的所有出边，对于每个邻居节点 neighbor，计算从起点到该节点的路径长度 current.cost + cost，并将该节点作为下一个节点 next_node 的父节点，并通过先前定义的启发式函数计算 next_node 到终点的估计距离 next_node.estimate，并将 next_node 加入到优先队列 open_set 中，从而完成一次节点扩展。

在节点扩展完毕后，检查是否已经找到了 K 条路径，若是则直接返回 k_shortest_paths，否则补充不足 K 条路径的情况（设为-1，表示路径不存在）。

四、案例测试

首先将测试用例输入检查代码正确性：

对于题目一，输入 150732684，输出为 2（5 右移 3 上移），代码输出与此一致：

```
150732684
2
```

输入所有测试样例，得到输出：

Input	Output
135720684	1
105732684	1
015732684	2
135782604	1
715032684	3

```
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
top\当代人工智能\实验二\A1.py"
135720684
1
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
top\当代人工智能\实验二\A1.py"
105732684
1
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
top\当代人工智能\实验二\A1.py"
015732684
2
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
top\当代人工智能\实验二\A1.py"
135782604
1
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二>
top\当代人工智能\实验二\A1.py"
715032684
3
```

对于题目二，代码对于测试案例的输出与答案一致：

```
5 7 3
1 2 1
1 3 4
2 4 3
3 4 2
3 5 1
4 5 2
5 1 5
5
6
8
```

输入所有测试样例，得到输出：

Input	Output
5 6 4 1 2 1 1 3 1 2 4 2 2 5 2 3 4 2 3 5 2	3 3 -1 -1
6 9 4 1 2 1 1 3 3 2 4 2 2 5 3 3 6 1 4 6 3	4 5 6 7

5 6 3 1 6 8 2 6 4	
7 12 6 1 2 1 1 3 3 2 4 2 2 5 3 3 6 1 4 7 3 5 7 1 6 7 2 1 7 10 2 6 4 3 4 2 4 5 1	5 5 6 6 7 7
5 8 7 1 2 1 1 3 3 2 4 1 2 5 3 3 4 2 3 5 2 1 4 3 1 5 4	4 4 5 -1 -1 -1 -1
6 10 8 1 2 1 1 3 2 2 4 2 2 5 3 3 6 3 4 6 3 5 6 1 1 6 8 2 6 5 3 4 1	5 5 6 6 6 8 -1 -1

```
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> top\当代人工智能\实验二\A2.py"
5 6 4
1 2 1
1 3 1
2 4 2
2 5 2
3 4 2
3 5 2
3
3
-1
-1
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> 代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> top\当代人工智能\实验二\A2.py"
6 9 4
1 2 1
1 3 3
2 4 2
2 5 3
3 6 1
4 6 3
5 6 3
1 6 8
2 6 4
4
5
6
7
```

```
7 12 6
1 2 1
1 3 3
2 4 2
2 5 3
3 6 1
4 7 3
5 7 1
6 7 2
1 7 10
2 6 4
3 4 2
4 5 1
5
5
6
6
7
7
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> 代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> top\当代人工智能\实验二\A2.py"
5 8 7
1 2 1
1 3 3
2 4 1
2 5 3
3 4 2
3 5 2
1 4 3
1 5 4
4
4
5
-1
-1
-1
-1
```

```
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> 代人工智能\实验二"
PS C:\Users\lxyAM\Desktop\当代人工智能\实验二> top\当代人工智能\实验二\A2.py"
6 10 8
1 2 1
1 3 2
2 4 2
2 5 3
3 6 3
4 6 3
5 6 1
1 6 8
2 6 5
3 4 1
5
5
6
6
6
8
-1
-1
```

五、总结

本次实验通过 A*算法解决了两道题目，通过解决具体问题巩固了 A*算法的相关知识，成功将理论落实到应用，但过程中也遇到了不少问题：

1. 在实现第二道题目的时候，误解题意为无向图，最后输出了 5 5 6，最后经过仔细查看题意修改为有向图，然后咨询老师才知道给定条件中的“5 1 5”为冗余条件（因为只能下行）

2. 在实现第二道题目时会出现无法找到部分路径问题（可以找到最短路径，但其他路径有时找不到），检查代码并调试时发现是节点的状态标记出现了问题，将已访问节点的 set 设置在了搜索节点的循环之外，导致部分节点加入集合（被访问过）后被设为永久被访问，每次搜索节点都会将该节点排除在外，因此在寻找最优路径时没有影响，但查找其他路径时可能会出现找不到的情况（比如测试案例输出了 5 6 -1）

3. 对于第二道题目的第二个测试案例，代码输出了“5 4 6 7”，经过排查调试后发现是启发式函数的选取有问题，最初基于贪心策略选取了曼哈顿距离作为启发式函数(N-room)，但在一些情境下会先探索估计离目标节点近，但实际上代价更高的节点，从而导致错误的发生，修改启发式函数为选择节点的所有出边中权重最小的边的权重作为估计距离（没有出边则估计距离为 0），修改后成功解决该问题。

综上，在不断解决问题的过程中进一步加深了对 A*算法的理解，认识到了启发式函数选择的重要性。