2020 年 春 季学期研究生课程考核

实验报告（二）

考 核 科 目： 高级算法设计与分析

学生所在院（系）： 计算机科学与技术学院

学 生 所 在 学 科： 计算机科学与技术

学 生 姓 名： 于晟健

学 号： 19S003037

学 生 类 别： 全日制学术型硕士研究生

考 核 结 果 阅 卷 人

实验 2　搜索算法

2.1 实验目的

（1）掌握搜索算法的基本设计思想与方法；
（2）掌握 *A**算法的设计思想与方法；
（3）熟练使用高级编程语言实现搜索算法；
（4）利用实验测试给出的搜索算法的正确性；

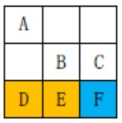
2.2 实验学时

4 学时。

2.3 实验问题

寻路问题。以图 1 为例，输入一个方格表示的地图，要求用 *A**算法找到并输出从起点（在方格中标示字母 *S*）到终点（在方格中标示字母 *T*）的代价最小的路径。有如下条件及要求：

（1）每一步都落在方格中，而不是横竖线的交叉点；
（2）灰色格子表示障碍，无法通行；
（3）在每个格子处，若无障碍，下一步可以达到八个相邻的格子，并且只可以到达无障碍的相邻格子。其中，向上、下、左、右四个方向移动的代价为 1，向四个斜角方向移动的代价为√2。
（4）在一些特殊格子上行走要花费额外的地形代价。比如，黄色格子代表沙漠，经过它的代价为 4；蓝色格子代表溪流，经过它的代价为 2；白色格子为普通地形，经过它的代价为 0。
（5）经过一条路径总的代价为移动代价+地形代价。其中移动代价是路径上所做的所有移动的代价的总和；地形代价为路径上除起点外所有格子的地形代价的总和。比如，在下图的示例中，路径 *A*→*B*→*C* 的代价为√2+1(移动)+0(地形)，而路径 *D*→*E*→*F* 的代价为 2(移动)+6(地形)。



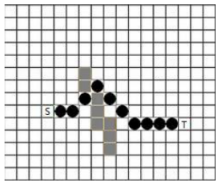


图 1

2.4 实验步骤

2.4.1 实现单向和双向的 *A**搜索算法

以图 1 为样例，实现单向和双向 *A**算法并测试输出。

（1）单向 *A**搜索算法

算法规则如下：

a. 单向 *A**算法使用最佳优先搜索策略求解优化问题；

b. 算法中代价函数 $f(x)$定义为 $g(n)+h(n)$，$g(n)$是起点 S 到点 n 的优化路径代价，$h^*(n)$是从点 n 到终点 T 的优化路径代价，则 $h(n)$是 $h^*(n)$的估计值且对于所有点 n 有 $h(n)\leq h^*(n)$；在本问题中，设定 $h(n)$为点 n 到终点 T 的欧氏距离；

c. 当选择的点是终点时，算法停止，该路径即为优化解。

我们首先给出对于 *A**搜索树结点 *node* 和点 *point* 的定义：

```python
class point(object):
    def __init__(self, row, column):
        self.row, self.column = row, column


class node(object):
    def __init__(self, parent, n: point, topography: int, end: point, reverse: bool):
        self.row, self.column = n.row, n.column
        self.parent = parent
        self.reverse = reverse  # whether the path is reverse
        if parent == None:
            self.G = 0
            self.H = 0
        else:
            self.G = parent.G + math.sqrt(
                (self.row - parent.row) ** 2 + (self.column - parent.column) ** 2) + topography
            # self.H=abs(self.row-end.row)+abs(self.column-end.column)
            self.H = math.sqrt((self.row - end.row) ** 2 + (self.column - end.column) ** 2)
        self.F = self.G + self.H

    def __lt__(self, other):
        return self.F < other.F

    def __str__(self):
        return '(' + str(self.row) + ',' + str(self.column) + ')'
```

故算法代码如下：

```python
def singleAStarSearch(Map: np.array, start: point, end: point) -> node:
    """

    :param Map: the topography of Map
    :param start: the start point
    :param end: the end point
    :return: the end node with the information of path
    """
    visitedMap = np.zeros(Map.shape)
    heap = []
    n = node(None, start, 0, end, False)
    # if the point n is the end point then over
    while (n.row != end.row or n.column != end.column):
        # find the possible area
        row_low = n.row - 1 if n.row > 0 else n.row
        row_high = n.row + 1 if n.row < Map.shape[0] - 1 else n.row
        column_low = n.column - 1 if n.column > 0 else n.column
        column_high = n.column + 1 if n.column < Map.shape[1] - 1 else n.column
        # extend the path from point n
        for i in range(row_low, row_high + 1):
            for j in range(column_low, column_high + 1):
                # skip the point n ,visited point and unreachable point
                if (i == n.row and j == n.column) or visitedMap[i][j] or Map[i][j] < 0: continue
                # push the new node into heap and mark it as visited
                heapq.heappush(heap, node(n, point(i, j), Map[i][j], end, False))
                visitedMap[i][j] = 1
        # if no path can arrive at the end point then return None
        if len(heap) == 0: return None
        n = heapq.heappop(heap)

    return n
```

（2）双向 *A**搜索算法

算法规则如下：

a. 双向 *A**算法分别从起点 S 和终点 T 同时进行 *A**搜索，我们同时对两个 *A**搜索

使用最佳优先搜索策略求解优化问题；

b. 对于从起点 S 出发的 A*搜索，代价函数 $f(x)$定义为 $g(n)+h(n)$，$g(n)$是起点 S 到点 n 的优化路径代价，$h^*(n)$是从点 n 到终点 T 的优化路径代价，则 $h(n)$是 $h^*(n)$的估计值且对于所有点 n 有 $h(n)\leq h^*(n)$；在本问题中，设定 $h(n)$为点 n 到终点 T 的欧氏距离；

c. 对于从终点 T 出发的 A*搜索，代价函数 $f'(x)$定义为 $g'(n)+h'(n)$，$g'(n)$是终点 T 到点 n 的优化路径代价，$h'^*(n)$是从点 n 到起点 S 的优化路径代价，则 $h'(n)$是 $h'^*(n)$的估计值且对于所有点 n 有 $h'(n)\leq h'^*(n)$；在本问题中，设定 $h'(n)$为点 n 到起点 S 的欧氏距离；

d. 当选择的点在另一个 A*搜索待扩展队列(堆)中，算法停止，该交点包含的 2 条路径即为优化解。

故算法代码如下：

```python
def bidirectionalAStarSearch(Map: np.array, start: point, end: point) -> tuple:
    """

    :param Map:
    :param start:
    :param end:
    :return:
    """
    visitedMap = np.zeros(Map.shape)
    visitedMap_reverse = np.zeros(Map.shape)

    heap, heap_reverse = [node(None, start, 0, end, False)], [node(None, end, 0, start, True)]
    n = heapq.heappop(heap) if heap[0] < heap_reverse[0] else heapq.heappop(heap_reverse)
    # if the point is visited in the another path then break
    while True:
        # if point belong to the path from end to start
        if n.reverse:
            if visitedMap[n.row][n.column]:
                break
            # find the possible area
            row_low = n.row - 1 if n.row > 0 else n.row
            row_high = n.row + 1 if n.row < Map.shape[0] - 1 else n.row
            column_low = n.column - 1 if n.column > 0 else n.column
            column_high = n.column + 1 if n.column < Map.shape[1] - 1 else n.column
            # extend the path from point n
            for i in range(row_low, row_high + 1):
                for j in range(column_low, column_high + 1):
                    # skip the point n ,visited point and unreachable point
                    if (i == n.row and j == n.column) or visitedMap_reverse[i][j] or Map[i][j] < 0: continue
                    new_node = node(n, point(i, j), Map[i][j], start, True)
                    heapq.heappush(heap_reverse, new_node)
                    visitedMap_reverse[i][j] = 1
        # if point belong to the path from start to end
        else:
            if visitedMap_reverse[n.row][n.column]:
                break
            # find the possible area
            row_low = n.row - 1 if n.row > 0 else n.row
            row_high = n.row + 1 if n.row < Map.shape[0] - 1 else n.row
            column_low = n.column - 1 if n.column > 0 else n.column
            column_high = n.column + 1 if n.column < Map.shape[1] - 1 else n.column
            # extend the path from point n
            for i in range(row_low, row_high + 1):
                for j in range(column_low, column_high + 1):
                    # skip the point n ,visited point and unreachable point
                    if (i == n.row and j == n.column) or visitedMap[i][j] or Map[i][j] < 0: continue
                    new_node = node(n, point(i, j), Map[i][j], end, False)
                    heapq.heappush(heap, new_node)
                    visitedMap[i][j] = 1
        # if no path can arrive at the end point then return None
        if len(heap) == 0 or len(heap_reverse) == 0: return None, None
        n = heapq.heappop(heap) if heap[0] < heap_reverse[0] else heapq.heappop(heap_reverse)

    # if point belong to the path from end to start
    if n.reverse:
        for m in heap:
            if m.row == n.row and m.column == n.column:
                return m, n
    # if point belong to the path from start to end
    else:
        for m in heap_reverse:
            if m.row == n.row and m.column == n.column:
                return n, m
```

2.4.2 测试 *A**算法的应用

精灵王子 *Haposola* 一大早从城堡中起床，要去沙漠中河流消失的地方寻找失落的宝藏。长路漫漫，处处险恶，精灵王子要尽快到达目标地点。

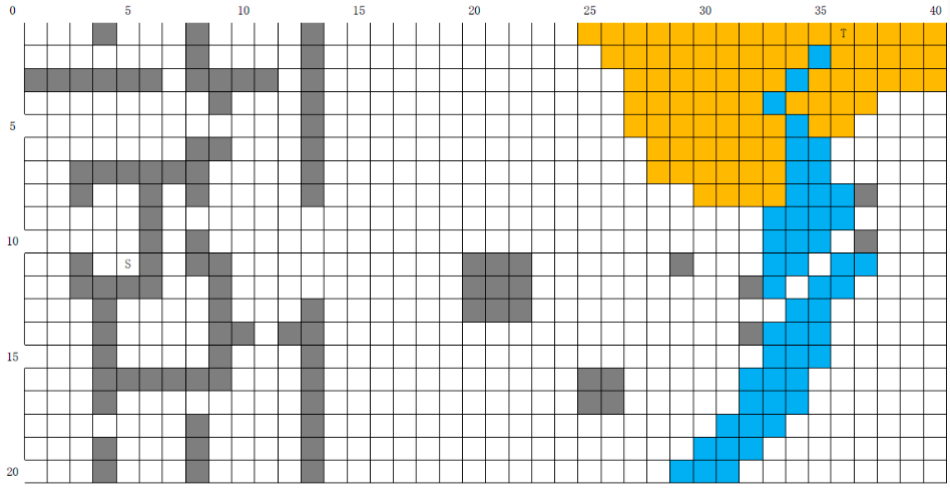
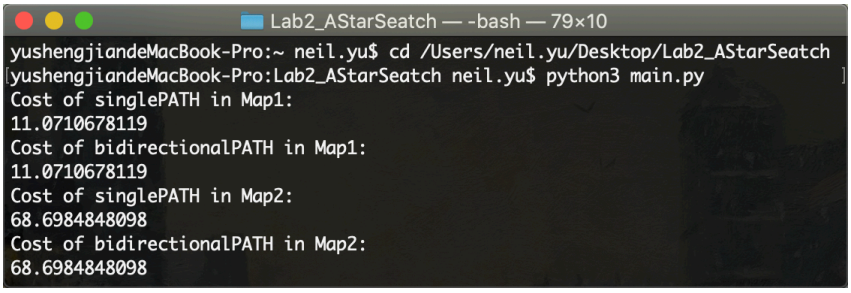


图 2

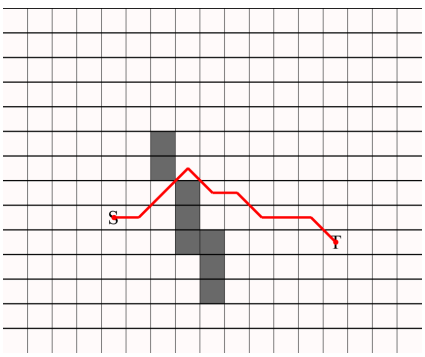
以图 2 为输入，采用单向和双向的 *A**算法，寻找地图上给出的起点和终点间的最小代价路径。规则和条件如第 2.3 节所述。

2.5 实验结果

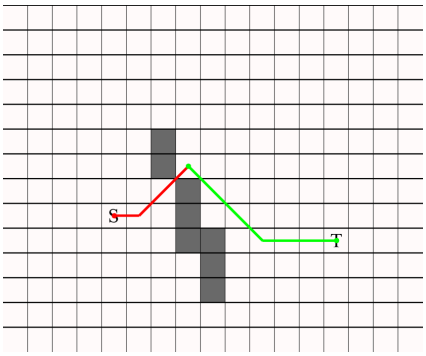
2.5.1 结果可视化



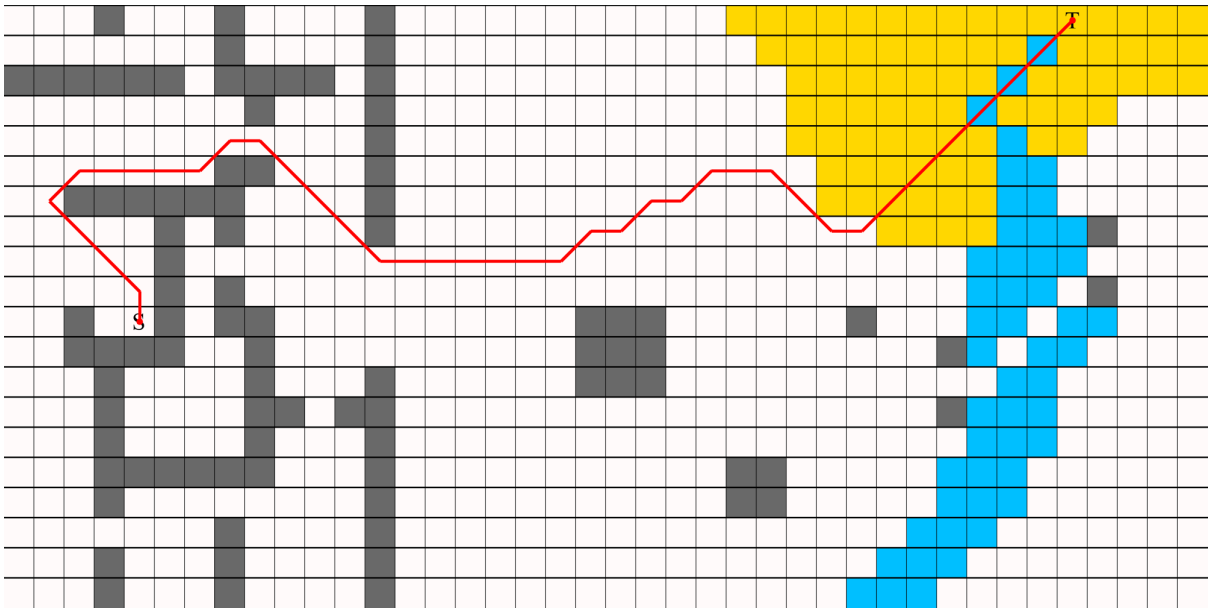
单向 *A**算法和双向 *A**算法分别在图 1 和图 2 寻找的路径最小代价分别相同，且路径如下（红色路径和绿色路径分别表示从起点 *S* 出发和从终点 *T* 出发的 *A**搜索）：



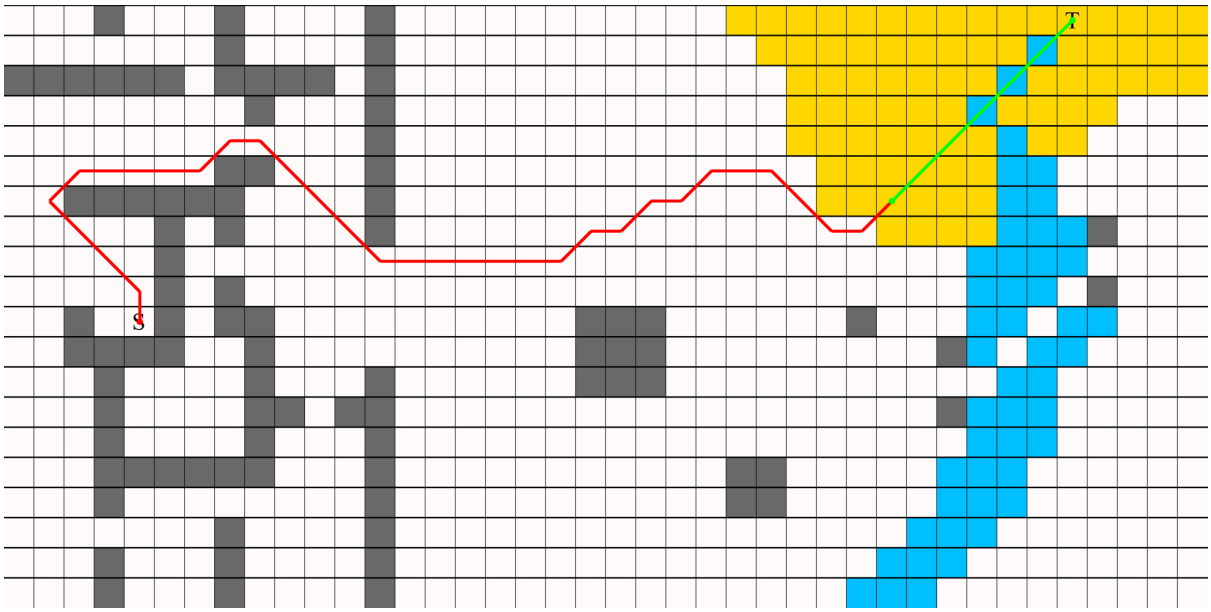
单向 *A**算法路径图（图 1）



双向 *A**算法路径图（图 1）



单向 *A**算法路径图（图 2）



双向 *A**算法路径图（图 2）

2.6 实验心得

通过本次实验，掌握了 A*算法的设计思想和方法（包含单向 A*算法和双向 A*算法）。同时，对于不同的启发式方法选择会有不同的效果。