به نام خدا

دانشگاه اصفهان

گروه مهندسی کامپیوتر

گزارش پروژه‌ی نهایی هوش مصنوعی

موضوع:

حل هزارتو (Maze) به کمک الگوریتم‌های کلاسیک

استاد:

خانم دکتر مرضیه حسینی

تهیه‌کنندگان:

حدیث گنجی 963613080

محمد امینازاده 973653002

زمستان 99

1 مقدمه

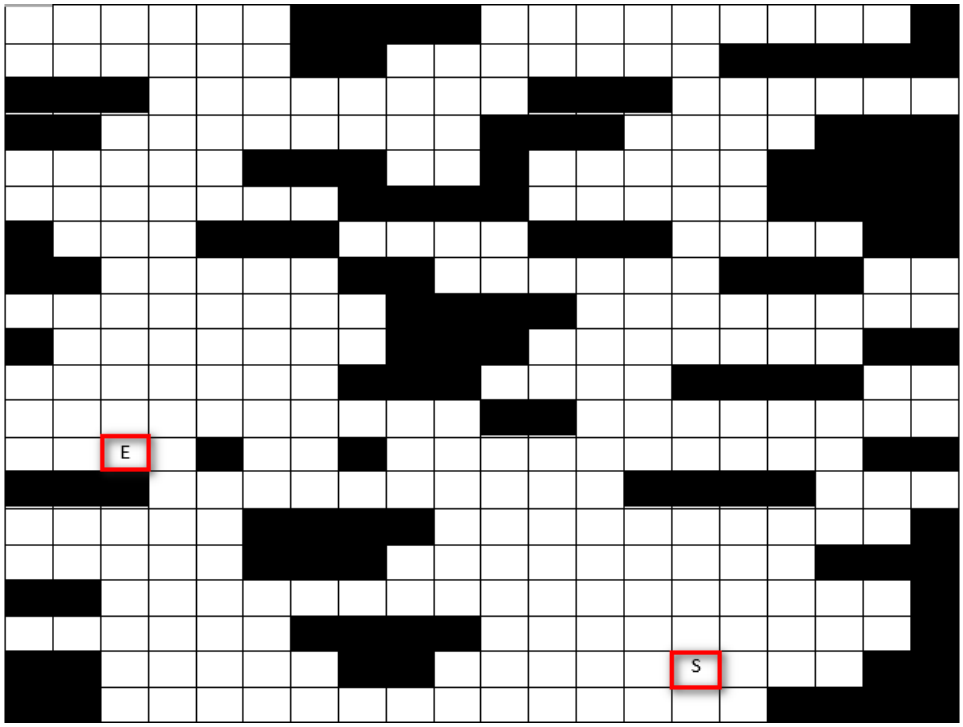
هدف ما از انجام این پروژه، پیاده‌سازی یک عامل برای حل هزارتو (maze) است. برای انجام بازی سه نوع روش جستجو ارائه کرده‌ایم که شامل جستجوی اول سطح (BFS) ، جستجوی عمیق شونده تکراری (IDFS) و جستجوی A^* است که جستجوی اول سطح و عمیق شونده تکراری از دسته جستجوی های ناآگاهانه و جستجوی A^* از دسته جستجوهای آگاهانه است. هرگاه عامل بازی را با هزینه کمتری با موفقیت به اتمام برساند، یعنی عملکرد بهینه تر و بهتری داشته است.

فضای حالت بازی به این گونه است که یک maze به صورت یک آرایه دو بعدی 20 در 20 در نظر می گیریم. هر سلول این آرایه یک موقعیت (x, y) دارد که از پایین چپ با شماره (0, 0) شروع شده و در بالا راست با (19, 19) خاتمه می‌یابد. هر سلول از این آرایه یا خالی یا پر است که به ترتیب میتوان با رنگ های سفید و سیاه نشان داد. همچنین دو خانه شروع و پایان نیز در آرایه وجود دارند که به صورت تصادفی انتخاب می‌شوند.

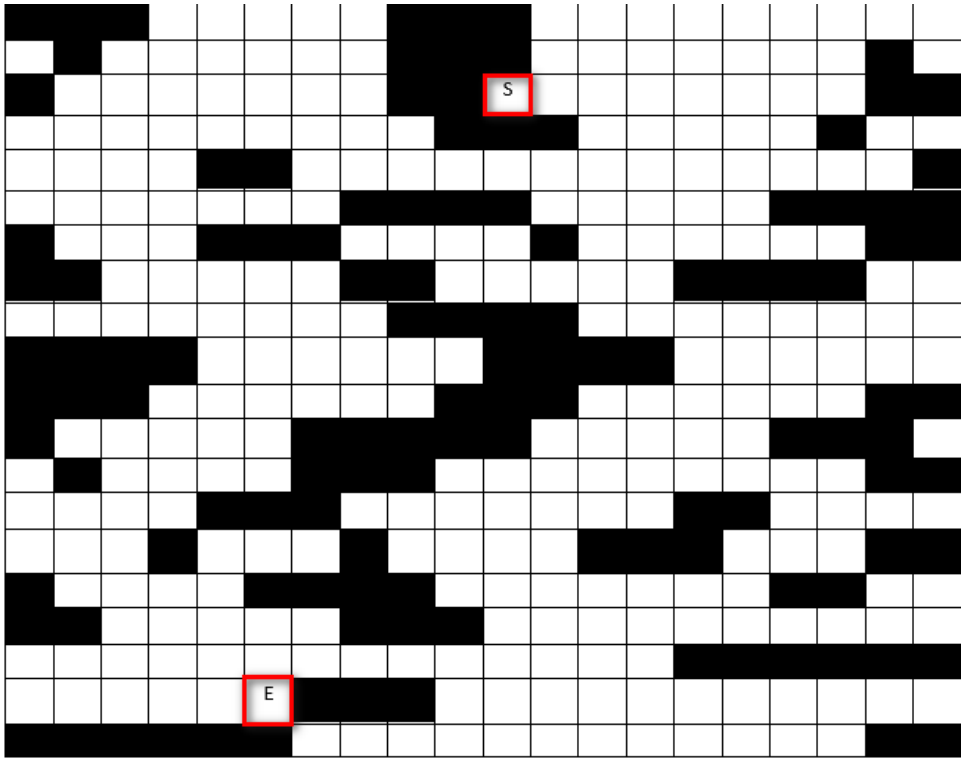
عامل می‌تواند به بالا، پایین، چپ و راست حرکت کند اما امکان حرکت قطری ندارد. همچمین نمی‌نواند به سلول‌های پر وارد شود یا به خارج از maze حرکت کند. وظیفه‌ی این عامل پیدا کردن مسیری از نقطه‌ی شروع به نقطه‌ی پایان است. هزینه‌ی مسیر تعداد حرکت‌های عامل در آن مسیر است.

از آن جا که انتخاب محل شروع و پایان بازی و شکل maze را به عهده کاربر گذاشتیم و در واقع هر سه پویا هستند، برای هر سه روش جستجویی که پیشنهاد کرده ایم، کاربر باید در ابتدا فضای حالت مورد نظر خود را در قالب یک فایل متنی که شامل 20 سطر است در اختیار برنامه نوشته شده قرار دهد. در این فایل هر سطر باید شامل 20 کاراکتر "." و یا "#" باشد و ما بین هر کدام یک "فاصله" قرار دهد. "." نشان دهنده خانه های سفید و "#" نشان دهنده خانه های سیاه است. علاوه بر این دو، کاربر باید در هر فایل متنی تنها یک خانه را به عنوان خانه شروع بازی و یک خانه را به عنوان خانه پایان بازی مشخص کند که برای این کار، "." و یا "#" را در خانه مورد نظرش به "X" برای نمایش خانه شروع (Start) و یا به "O" به عنوان خانه پایان بازی (End) تغییر دهد.

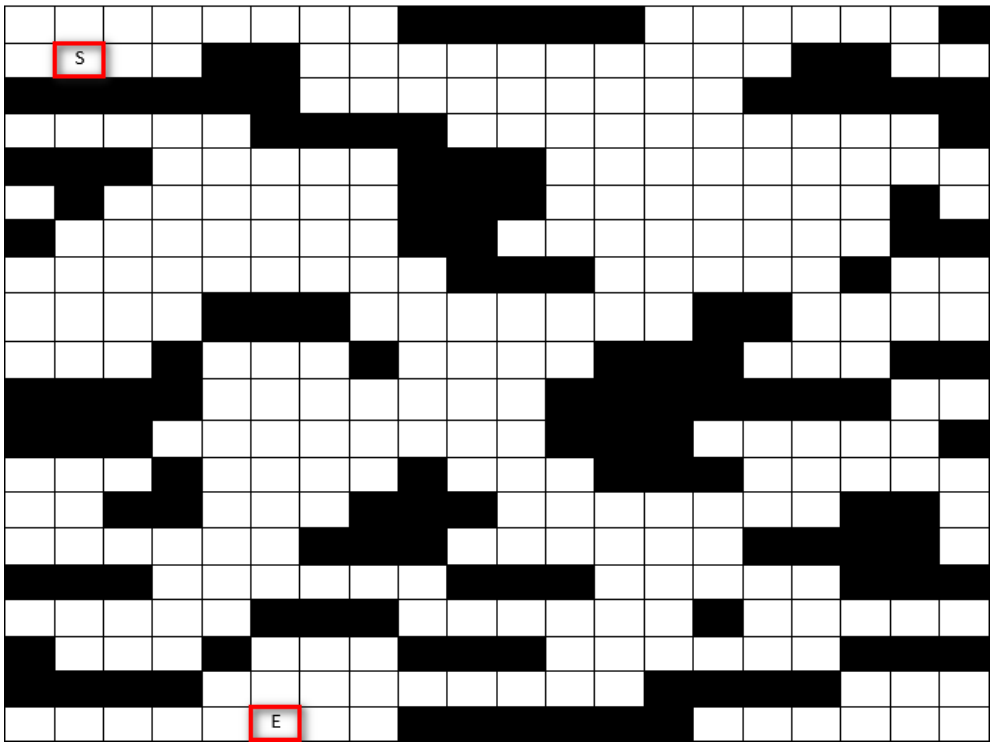
ما از سه تست کیس برای نمایش نوع کارکرد هر روش ارائه کرده ایم که فایل متنی آن ها به گزارش پیوست شده است (برای فهم بهتر قالب فایل و به درستی نوشتن فضای حالت مورد نظر خود، می توانید به آن ها مراجعه کنید). هر کدام از این تست کیس ها را در هر سه روش جستجو بررسی خواهیم کرد. این سه تست‌کیس به‌صورت زیر هستند.



TestCase1



TestCase2



TestCase3

در این گزارش، برای هر جستجو به صورت مجزا، ابتدا توضیحاتی کلی درمورد نحوه‌ی عملکرد آن جستجو می‌دهیم و سپس نحوه‌ی پیاده‌سازی عامل و الگوریتم، به‌همراه کد و توابع به‌کار رفته شده را به‌تفصیل شرح می‌دهیم. سپس خروجی واقعی کد در کنسول، به‌نمایش درمی‌آید و پس از آن مسیر طی شده توسط عامل را رسم می‌کنیم. همچنین هزینه و نودهای بسط داده شده به‌ترتیب بسط داده شدن لیست می‌شوند.

برای پیاده‌سازی این سه جستجو از زبان برنامه‌نویسی پایتون استفاده شده است و هر الگوریتم بر روی سه Testcase متفاوت اجرا می‌شوند و نتیجه شرح داده می‌شود.

2 جستجوی اول سطح (Breadth First Search)

2.1 نحوه‌ی عملکرد

شیوه کلی کار این نوع جستجوی ناآگاهانه به این صورت است که گره هایی با کمترین عمق را زودتر از سایر گره ها بسط می دهد. درنتیجه ابتدا گره های درون یک سطر کم عمق را بسط می دهد و سپس به سراغ بسط دادن گره های سطر عمیق تر بعدی می رود و آنقدر این کار را تکرار می کند تا در عمیق ترین سطری که در حال بسط دادن آن است به هدف برسد و سپس جستجو را متوقف می کند.

این الگوریتم جستجو از یک مجموعه تحت عنوان مجموعه Frontier برای نگه داری گره هایی که در مرحله بعد می تواند بسط دهد، استفاده می کند. این مجموعه یک صف FIFO است. یعنی قدیمی ترین گره ای که به این مجموعه اضافه شده را زودتر از سایر گره ها از مجموعه خارج می کند و بسط می دهد.

این روش جستجو یک روش جستجوی کامل (اگر ضریب انشعاب در هر گره محدود باشد، حتما هدف را در عمق موردنظر پیدا می کند.) و بهینه (در صورتی که با افزایش عمق، هزینه مسیر افزایش یابد.) است. پیچیدگی زمانی و فضایی آن در بدترین حالت $O(b^d)$ است. (که b ضریب انشعاب و d عمق هدف است).

2.2 الگوریتم و کد پیاده‌سازی

برای پیاده‌سازی جستجوی اول سطح به این ترتیب عمل می کنیم که یک صف (Queue) تعریف می کنیم که برای هر نود حالت FIFO را به‌وجود آوریم. هر نود به‌صورت دنباله‌ای از اعمال (Actions) مشخص می‌شود و تابعی مجزا برای نگاشت این اعمال به یک نود با مختصات خاص تعریف کرده‌ایم. ابتدا دنباله‌ی اعمال به‌صورت یک رشته (String) خالی است. سپس تا زمانی که هدف یافت نشده است، به ازای تمام اعمال ممکن (R, L, U, D) ، اعمالی که قابل قبول هستند (از صفحه خارج نمی‌شود یا به مانع برخورد نمی‌کند یا به خانه‌ای که قبلا بوده برنمی‌گردد) را به رشته‌ی عملی که در اول صف قرار دارد اضافه می‌کند و آن عمل قدیمی را از صف حذف می‌کند و سپس اعمال جدید تولید شده را به انتهای صف اضافه می‌کند. با این روش، نودهایی (به‌صورت دنباله‌ی اعمال) که زودتر به صف اضافه شده‌اند، زودتر بسط داده می‌شوند و نودها به‌صورت سطر به سطر بسط داده می‌شوند. در هر مرحله پس از اضافه شدن یک عمل جدید به دنباله‌ی اعمال گذشته، چک می‌شود که آیا به خانه‌ی هدف رسیده‌ایم یا خیر. اگر رسیده باشیم حلقه قطع می‌شود و آخرین رشته‌ی تولید شده، همان رشته‌ی اعمالی است که عامل را به هدف می‌رسانند.

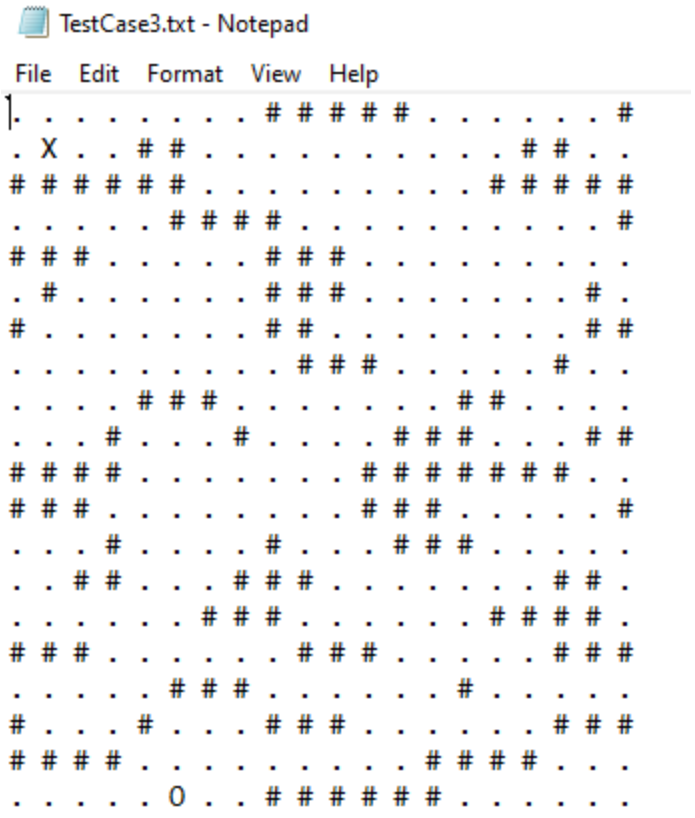
```python
if __name__ == '__main__':
    maze_board = build_board()
    path = queue.Queue()
    path.put("")
    previous_path = ""
    while not goal_test(maze_board, previous_path):
        previous_path = path.get()
        add_to_expanded_nodes(maze_board, previous_path)
        for move in ["R", "L", "U", "D"]:
            temp_path = previous_path + move
            if is_valid(maze_board, temp_path):
                if not_in_explored_set(maze_board, temp_path):
                    path.put(temp_path)
                    add_to_explored_set(maze_board, temp_path)
    cost = len(previous_path)
    print(f"The solution action sequence is {previous_path} and the cost is {cost}")
    print(f"The path nodes are:\n{map_moves_to_nodes(maze_board, previous_path)}")
    print(f"the expanded nodes are:\n{expanded_nodes}")

```

- اکنون توابع دیگری که در برنامه به‌کار رفته است را شرح می‌دهیم:

Build_board(): این تابع برای ساخت صفحه‌ی maze به‌کار می‌رود. صفحه به‌صورت یک فایل txt. است. پس از خواندن فایل هر خط به صورت یک لیست از کاراکترها ذخیره می‌شود. کل لیست‌های خطوط درون یک لیست بزرگتر قرار دارند که صفحه‌ی بازی را تشکیل می‌دهند.

```python
def build_board ():
    board = []
    with open('D:\\UNI\\AI\\Project\\MAZE TC\\TestCase3.txt') as board_text:
        for item in board_text.readlines():
            board.append(item.rstrip().rsplit())
    board.reverse()
    return board
```

نمونه ای از یک فایل متنی ورودی به این تابع، به صورت زیر است:



get_agent_node(board): این تابع وظیفه‌ی یافتن نقطه‌ی ابتدایی عامل بر روی صفحه را دارد.

```
14    def get_agent_node(board):
15        agent_node = []
16        for i in range(board_size):
17            for j in range(board_size):
18                if board[i][j] == "X":
19                    agent_node.append(i)
20                    agent_node.append(j)
21                    break
22        return agent_node
```

Half_update_agent_node(board, moves, agent_node): این تابع وظیفه دارد مکان فعلی عامل را بر اساس مکان اولیه و رشته‌ی اعمال تا آخرین عمل به‌روز رسانی کند، اما آخرین عمل را به‌حساب نمی‌آورد.

```
24  def half_update_agent_node(board, moves, agent_node):
25      for move in moves[:-1]:
26          if move == "R":
27              agent_node[1] = agent_node[1] + 1
28          elif move == "L":
29              agent_node[1] = agent_node[1] - 1
30          elif move == "U":
31              agent_node[0] = agent_node[0] + 1
32          else:
33              agent_node[0] = agent_node[0] - 1
34
```

Full_update_agent_node(board, moves, agent_node): این تابع نیز وظیفه‌ی به‌روزرسانی مختصات عامل را دارد اما آخرین حرکت او را نیز به‌حساب می‌آورد.

```
35  def full_update_agent_node(board, moves, agent_node):
36      for move in moves[:]:
37          if move == "R":
38              agent_node[1] = agent_node[1] + 1
39          elif move == "L":
40              agent_node[1] = agent_node[1] - 1
41          elif move == "U":
42              agent_node[0] = agent_node[0] + 1
43          else:
44              agent_node[0] = agent_node[0] - 1
45
```

Is_valid(board, moves): این تابع که یک مقدار True/False برمی‌گرداند نشان می‌دهد که اضافه شدن یک عمل به یک رشته از اعمال مجاز است یا خیر. مجاز بودن بر اساس بیرون نزدن از صفحه، یا برخورد نکردن با مانع یا برنگشتن به نودی که قبلا در آن بوده است، می‌باشد. عملکرد این تابع به‌این صورت است که مکان عامل را تا عمل آخر اما بدون احتساب آن حساب می‌کند. سپس بر اساس آخرین عمل که کدام یک از چهار عمل باشد مشخص می‌کند که آیا رفتن به آن نود مجاز است یا خیر.

```python
def is_valid(board, moves):
    agent_node = get_agent_node(board)
    half_update_agent_node(board, moves, agent_node)

    if moves[-1] == "R":
        if agent_node[1] + 1 > board_size - 1 or board[agent_node[0]][agent_node[1] + 1] == "#":
            return False
        return True
    elif moves[-1] == "L":
        if agent_node[1] - 1 < 0 or board[agent_node[0]][agent_node[1] - 1] == "#":
            return False
        return True
    elif moves[-1] == "U":
        if agent_node[0] + 1 > board_size - 1 or board[agent_node[0] + 1][agent_node[1]] == "#":
            return False
        return True
    else:
        if agent_node[0] - 1 < 0 or board[agent_node[0] - 1][agent_node[1]] == "#":
            return False
        return True
```

Add_to_explored_set(board, moves): وظیفه‌ی این تابع اضافه کردن یک نود به لیست نودهای ملاقات‌شده است.

```python
def add_to_explored_set(board, moves):
    agent_node = get_agent_node(board)
    full_update_agent_node(board, moves, agent_node)
    explored_set.append(agent_node)
```

Not_in_explored_set(board, moves): اگر یک نود در explored set نباشد مقدار True را برمی‌گرداند. در غیر این صورت False برمی‌گرداند.

```python
def not_in_explored_set(board, moves):
    agent_node = get_agent_node(board)
    full_update_agent_node(board, moves, agent_node)

    if agent_node not in explored_set:
        return True
    return False
```

Goal_test(board, moves): این تابع درصورتی مقدار True را برمی‌گرداند که رشته اعمال شده کنونی، او را از خانه‌ی ابتدایی به هدف برساند. درغیر این صورت False برمی‌گرداند.

```
81      def goal_test (board, moves):
82          agent_node = get_agent_node(board)
83          full_update_agent_node(board, moves, agent_node)
84
85          if board[agent_node[0]][agent_node[1]] == "0":
86              return True
87          return False
88
```

Map_moves_to_nodes(board, moves): این تابع وظیفه‌ی برگرداندن تمام نودهایی است که در نتیجه‌ی آن رشته اعمال پیمایش می‌شوند. در واقع وظیفه‌ی نگاشت رشته اعمال به دنباله‌ای از نودها را دارد.

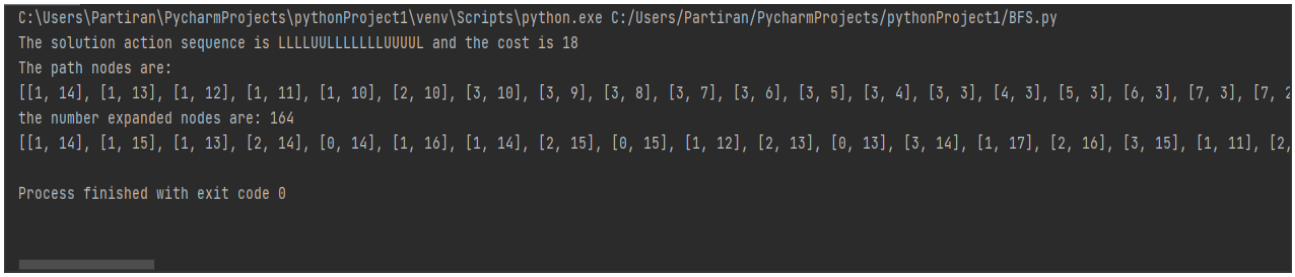
```
89      def map_moves_to_nodes(board, moves):
90          path_nodes = []
91          path_nodes.append(get_agent_node(board))
92          for move in moves:
93              if move == "R":
94                  path_nodes.append([path_nodes[-1][0], path_nodes[-1][1] + 1])
95              elif move == "L":
96                  path_nodes.append([path_nodes[-1][0], path_nodes[-1][1] - 1])
97              elif move == "U":
98                  path_nodes.append([path_nodes[-1][0] + 1, path_nodes[-1][1]])
99              else:
100                 path_nodes.append([path_nodes[-1][0] - 1, path_nodes[-1][1]])
101         return path_nodes
```

Add_to_expanded_nodes(board, moves): این تابع آن نودی که توسط رشته اعمال شده از نود ابتدایی حرکت عامل ایجاد می‌شود را به نودهای بسط داده شده اضافه می کند. (این تابع در راستای نمایش نودهای بسط داده شده در طول اجرای الگوریتم، در صورت پروژه است).

```
103     def add_to_expanded_nodes(board, moves):
104         agent_node = get_agent_node(board)
105         full_update_agent_node(board, moves, agent_node)
106         expanded_nodes.append(agent_node)
```

2.3 خروجی برنامه

- **خروجی TestCase1 به صورت زیر است:**



مشاهده می‌شود که رشته اعمال LLLLUULLLLLLUUUUL راه حل است که هزینه‌ی آن برابر 18 است.

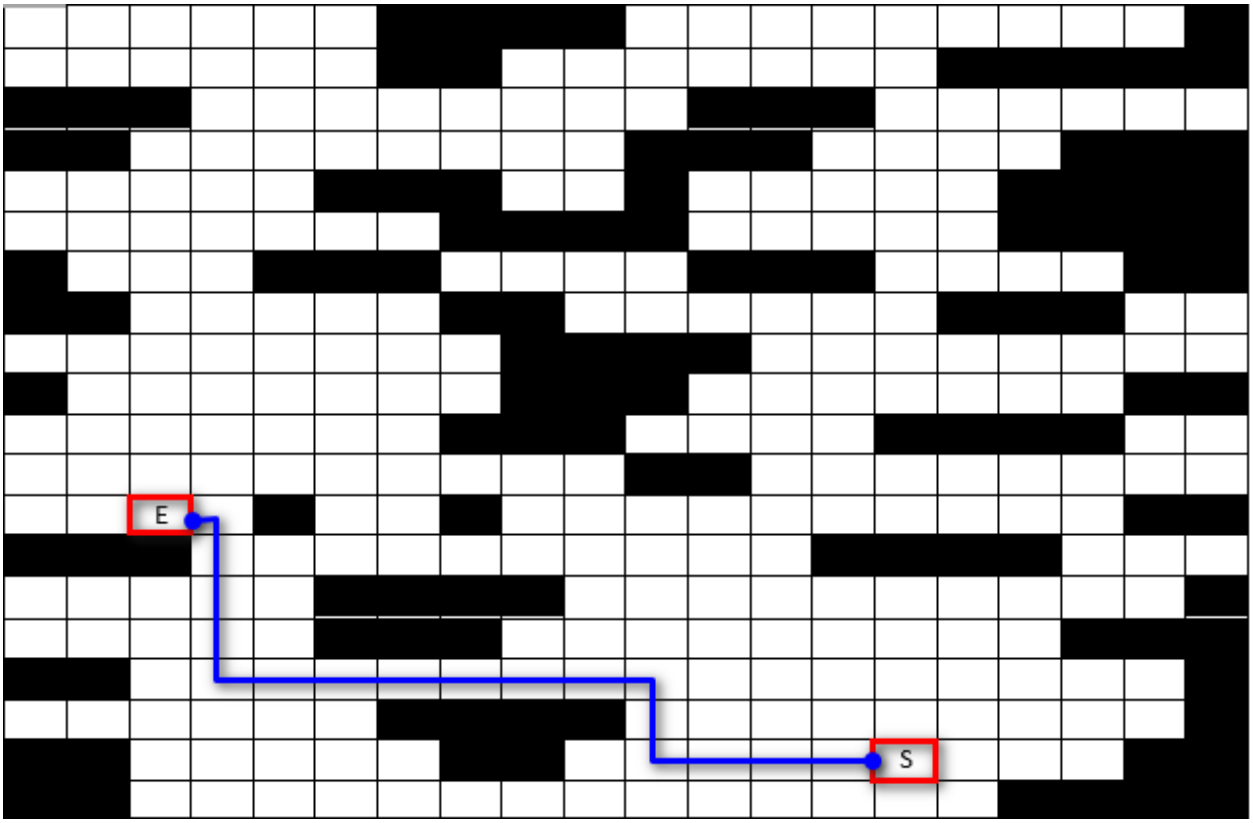
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[1, 14], [1, 13], [1, 12], [1, 11], [1, 10], [2, 10], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [4, 3], [5, 3], [6, 3], [7, 3], [7, 2]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 164 نود را شامل می‌شود.

[1, 14], [1, 15], [1, 13], [2, 14], [0, 14], [1, 16], [1, 14], [2, 15], [0, 15], [1, 12], [2, 13], [0, 13], [3, 14], [1, 17], [2, 16], [3, 15], [1, 11], [2, 12], [0, 12], [3, 13], [4, 14], [2, 17], [3, 16], [4, 15], [1, 10], [2, 11], [0, 11], [3, 12], [4, 13], [5, 14], [2, 18], [3, 17], [4, 16], [5, 15], [1, 9], [2, 10], [0, 10], [3, 11], [4, 12], [5, 13], [3, 18], [5, 16], [0, 9], [3, 10], [4, 11], [5, 12], [5, 17], [0, 8], [3, 9], [4, 10], [5, 11], [6, 12], [5, 18], [6, 17], [0, 7], [3, 8], [4, 9], [5, 10], [6, 11], [7, 12], [6, 18], [7, 17], [0, 6], [3, 7], [4, 8], [5, 9], [6, 10], [7, 11], [7, 13], [8, 12], [6, 19], [7, 16], [8, 17], [0, 5], [1, 6], [3, 6], [6, 9], [7, 10], [7, 14], [8, 13], [9, 12], [7, 15], [8, 16], [8, 18], [0, 4], [1, 5], [3, 5], [6, 8], [7, 9], [8, 14], [9, 13], [9, 11], [10, 12], [8, 15], [8, 19], [9, 18], [0, 3], [1, 4], [2, 5], [3, 4], [6, 7], [7, 8], [8, 9], [10, 13], [9, 10], [10, 11], [11, 12], [9, 19], [0, 2], [1, 3], [2, 4], [3, 3], [4, 4], [6, 6], [8, 8], [10, 14], [11, 13], [12, 12], [1, 2], [2, 3], [3, 2], [4, 3], [5, 4], [6, 5], [7, 6], [8, 7], [10, 15], [11, 14], [12, 13], [12, 11], [2, 2], [4, 2], [5, 3], [6, 4], [7, 5], [8, 6], [10, 16], [11, 15], [12, 14], [12, 10], [2, 1], [4, 1], [5, 2], [6, 3], [8, 5], [9, 6], [10, 17], [11, 16], [13, 14], [12, 9], [13, 10], [2, 0], [4, 0], [5, 1], [7, 3], [8, 4], [9, 5], [10, 6], [11, 17], [13, 15], [14, 14], [13, 9], [5, 0], [7, 2]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase1 طی می‌شود به‌صورت زیر است:



- خروجی **TestCase2** به‌صورت زیر است:

```
C:\Users\Partiran\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Partiran/PycharmProjects/pythonProject1/BFS.py
The solution action sequence is RRRDDDDDDRDDLLDLDLDDDDDLLLLLLD and the cost is 30
The path nodes are:
[[17, 10], [17, 11], [17, 12], [17, 13], [16, 13], [15, 13], [14, 13], [13, 13], [12, 13], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12],
the number expanded nodes are: 267
[[17, 10], [17, 11], [17, 12], [17, 10], [18, 11], [17, 13], [18, 12], [16, 12], [19, 11], [17, 14], [18, 13], [16, 13], [19, 12], [15, 12], [17, 15], [18, 1

Process finished with exit code 0
```

مشاهده می‌شود که رشته اعمال RRRDDDDDDRDDLLDLDLDDDDDLLLLLLD راه حل است که هزینه‌ی آن برابر 30 است.

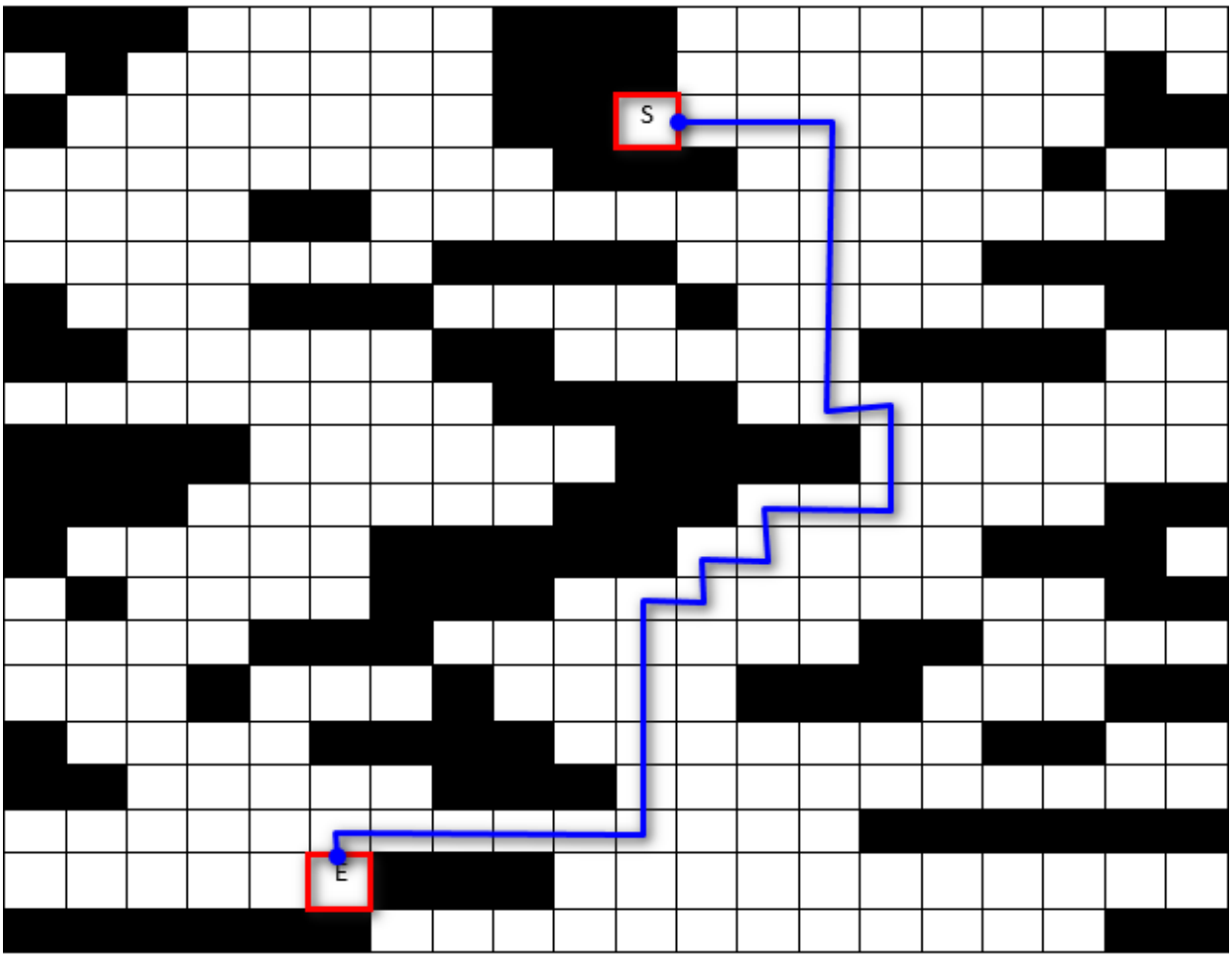
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [17, 13], [16, 13], [15, 13], [14, 13], [13, 13], [12, 13], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12], [8, 11], [7, 11], [7, 10], [6, 10], [5, 10], [4, 10], [3, 10], [2, 10], [2, 9], [2, 8], [2, 7], [2, 6], [2, 5], [2, 4], [1, 4]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 267 نود را شامل می‌شود.

[17, 10], [17, 11], [17, 12], [17, 10], [18, 11], [17, 13], [18, 12], [16, 12], [19, 11], [17, 14], [18, 13], [16, 13], [19, 12], [15, 12], [17, 15], [18, 14], [16, 14], [19, 13], [15, 13], [15, 11], [14, 12], [17, 16], [18, 15], [16, 15], [19, 14], [15, 14], [14, 13], [15, 10], [14, 11], [13, 12], [17, 17], [18, 16], [16, 16], [19, 15], [15, 15], [14, 14], [13, 13], [15, 9], [12, 12], [18, 17], [19, 16], [15, 16], [14, 15], [13, 14], [12, 13], [15, 8], [12, 11], [11, 12], [19, 17], [15, 17], [13, 15], [11, 13], [15, 7], [16, 8], [12, 10], [19, 18], [15, 18], [13, 16], [11, 14], [15, 6], [16, 7], [12, 9], [13, 10], [19, 19], [16, 18], [13, 17], [11, 15], [10, 14], [16, 6], [14, 6], [17, 7], [13, 9], [18, 19], [16, 19], [11, 16], [10, 15], [9, 14], [16, 5], [17, 6], [14, 5], [18, 7], [13, 8], [11, 17], [10, 16], [9, 15], [9, 13], [8, 14], [16, 4], [17, 5], [18, 6], [14, 4], [19, 7], [13, 7], [11, 18], [10, 17], [9, 16], [8, 15], [9, 12], [8, 13], [7, 14], [16, 3], [17, 4], [18, 5], [19, 6], [14, 3], [11, 19], [12, 18], [10, 18], [9, 17], [7, 15], [8, 12], [7, 13], [16, 2], [17, 3], [15, 3], [18, 4], [19, 5], [14, 2], [13, 3], [12, 19], [10, 19], [7, 16], [8, 11], [7, 12], [6, 13], [16, 1], [17, 2], [15, 2], [18, 3], [19, 4], [14, 1], [13, 2], [12, 3], [7, 17], [6, 16], [7, 11], [6, 12], [16, 0], [17, 1], [15, 1], [18, 2], [19, 3], [14, 0], [13, 1], [12, 2], [12, 4], [11, 3], [6, 17], [5, 16], [7, 10], [6, 11], [15, 0], [11, 2], [12, 5], [11, 4], [6, 18], [5, 17], [5, 15], [7, 9], [6, 10], [5, 11], [11, 1], [12, 6], [11, 5], [10, 4], [6, 19], [4, 15], [6, 9], [5, 10], [4, 11], [11, 0], [11, 6], [10, 5], [9, 4], [4, 14], [3, 15], [6, 8], [5, 9], [4, 10], [4, 12], [3, 11], [11, 7], [10, 6], [9, 5], [9, 3], [8, 4], [4, 13], [3, 14], [3, 16], [6, 7], [5, 8], [4, 9], [3, 10], [3, 12], [2, 11], [10, 7], [9, 6], [8, 5], [8, 3], [7, 4], [3, 13], [3, 17], [2, 10], [2, 12], [1, 11], [10, 8], [9, 7], [7, 5], [8, 2], [7, 3], [2, 13], [3, 18], [2, 9], [1, 10], [1, 12], [0, 11], [10, 9], [9, 8], [8, 1], [7, 2], [6, 3], [1, 13], [3, 19], [4, 18], [2, 8], [1, 9], [0, 10], [0, 12], [6, 2], [1, 14], [0, 13], [4, 19], [2, 7], [1, 8], [0, 9], [6, 1], [5, 2], [1, 15], [0, 14], [2, 6], [0, 8], [6, 0], [5, 1], [4, 2], [1, 16], [0, 15], [2, 5], [3, 6], [0, 7], [7, 0], [5, 0], [4, 1], [4, 3], [3, 2], [1, 17], [0, 16], [2, 4], [3, 5], [0, 6], [4, 4], [3, 3], [2, 2], [1, 18], [0, 17], [2, 3], [3, 4], [1, 4]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase2 طی می‌شود به‌صورت زیر است:



- خروجی **TestCase3** به صورت زیر است:

```
C:\Users\Partiran\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Partiran/PycharmProjects/pythonProject1/BFS.py
The solution action sequence is RRURRRRDRRRRRDDDDDDDLLDDDDRDRDDDLDDLLLLLLD and the cost is 42
The path nodes are:
[[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [17, 12], [16, 12], [15, 12
the number expanded nodes are: 256
[[18, 1], [18, 2], [18, 0], [19, 1], [18, 3], [18, 1], [19, 2], [19, 0], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [18, 8], [17

Process finished with exit code 0
```

مشاهده می‌شود که رشته اعمال RRURRRRDRRRRRDDDDDDDLLDDDDRDRDDDLDDLLLLLLD راه حل است که هزینه‌ی آن برابر 42 است.

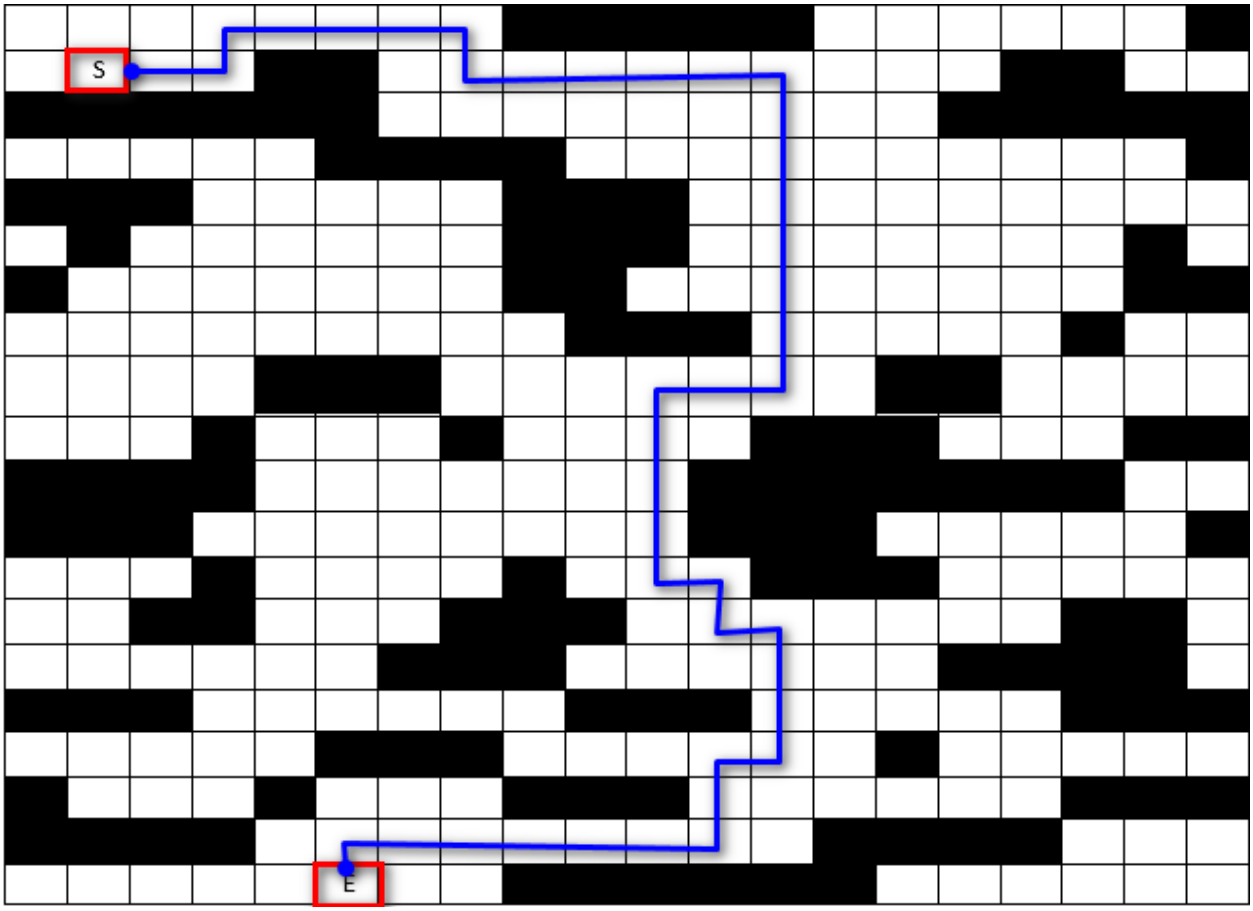
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [17, 12], [16, 12], [15, 12], [14, 12], [13, 12], [12, 12], [11, 12], [11, 11], [11, 10], [10, 10], [9, 10], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [5, 12], [4, 12], [3, 12], [3, 11], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [0, 5]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 256 نود را شامل می‌شود.

[18, 1], [18, 2], [18, 0], [19, 1], [18, 3], [18, 1], [19, 2], [19, 0], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [18, 8], [17, 7], [18, 9], [17, 8], [18, 10], [17, 9], [18, 11], [17, 10], [16, 9], [18, 12], [17, 11], [16, 10], [18, 13], [17, 12], [16, 11], [18, 14], [19, 13], [17, 13], [16, 12], [15, 11], [18, 15], [19, 14], [17, 14], [16, 13], [15, 12], [14, 11], [19, 15], [16, 14], [15, 13], [14, 12], [13, 11], [19, 16], [16, 15], [15, 14], [14, 13], [13, 12], [13, 10], [19, 17], [16, 16], [15, 15], [14, 14], [13, 13], [12, 12], [19, 18], [16, 17], [15, 16], [14, 15], [13, 14], [12, 13], [11, 12], [18, 18], [16, 18], [15, 17], [14, 16], [13, 15], [12, 14], [11, 13], [11, 11], [18, 19], [15, 18], [14, 17], [13, 16], [12, 15], [11, 10], [10, 11], [15, 19], [13, 17], [12, 16], [11, 9], [10, 10], [14, 19], [11, 16], [11, 8], [10, 9], [9, 10], [11, 17], [10, 16], [11, 7], [12, 8], [10, 8], [9, 9], [8, 10], [11, 18], [10, 17], [10, 15], [12, 7], [9, 8], [8, 9], [7, 10], [11, 19], [12, 18], [12, 6], [13, 7], [9, 7], [8, 8], [7, 9], [7, 11], [6, 10], [12, 19], [12, 5], [13, 6], [14, 7], [9, 6], [8, 7], [6, 11], [5, 10], [12, 4], [13, 5], [14, 6], [15, 7], [9, 5], [10, 6], [8, 6], [7, 7], [6, 12], [5, 11], [5, 9], [12, 3], [13, 4], [14, 5], [15, 6], [9, 4], [10, 5], [8, 5], [7, 6], [6, 13], [5, 12], [12, 2], [13, 3], [11, 3], [14, 4], [15, 5], [10, 4], [8, 4], [7, 5], [6, 6], [6, 14], [5, 13], [4, 12], [12, 1], [13, 2], [11, 2], [14, 3], [15, 4], [8, 3], [7, 4], [6, 5], [6, 15], [5, 14], [4, 13], [3, 12], [12, 0], [13, 1], [11, 1], [14, 2], [10, 2], [15, 3], [16, 4], [6, 4], [5, 5], [6, 16], [7, 15], [4, 14], [3, 13], [3, 11], [2, 12], [11, 0], [10, 1], [16, 3], [5, 4], [4, 5], [7, 16], [8, 15], [4, 15], [2, 13], [3, 10], [2, 11], [1, 12], [10, 0], [16, 2], [5, 3], [4, 4], [4, 6], [7, 17], [8, 16], [8, 14], [4, 16], [3, 15], [2, 14], [3, 9], [1, 11], [16, 1], [5, 2], [4, 3], [3, 4], [4, 7], [7, 18], [8, 17], [3, 16], [2, 15], [3, 8], [1, 10], [16, 0], [5, 1], [3, 3], [4, 8], [7, 19], [8, 18], [3, 17], [2, 16], [1, 9], [5, 0], [6, 1], [3, 2], [2, 3], [6, 19], [9, 18], [3, 18], [1, 8], [6, 0], [7, 1], [3, 1], [2, 2], [5, 19], [9, 19], [3, 19], [1, 7], [7, 0], [7, 2], [3, 0], [2, 1], [1, 6], [2, 7], [0, 7], [1, 5], [2, 6], [0, 6], [1, 4], [2, 5], [0, 5]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase3 طی می‌شود به‌صورت زیر است:



3 جستجوی عمیق‌شونده‌ی تکراری

3.1 نحوه‌ی عملکرد

این روش جستجوی ناآگاهانه، همان جستجوی اول عمق (DFS) است که عمق را به صورت تدریجی زیاد می کند تا در کم عمق ترین سطح ممکن، گره هدف را بیابد. این روش جستجو نیز از یک مجموعه تحت عنوان مجموعه Frontier برای گره های کاوش نشده استفاده می کند. با این تفاوت که یک صف LIFO است. یعنی در هر مرحله آخرین گره ای که به مجموعه اضافه شده را برای بسط دادن انتخاب می کند.

این روش جستجو یک روش کامل (در صورت متناهی بودن ضریب انشعاب) و بهینه (در صورت افزایش هزینه مسیر با افزایش عمق گره) است. پیچیدگی زمانی این الگوریتم $O(b^d)$ و پیچیدگی فضایی آن $O(bd)$ است. (که d عمق هدف و b ضریب انشعاب است.)

درواقع این الگوریتم مشکل نمایی بودن پیچیدگی فضایی روش جستجوی اول سطح که منجر به نیاز به حافظه بسیار زیاد می شد را حل کرد.

3.2 الگوریتم و کد پیاده‌سازی

برای پیاده‌سازی این الگوریتم ابتدا الگوریتم و تابع جستجوی عمق محدود را پیاده‌سازی کردیم (limited_DFS).
الگوریتم این تابع به‌این صورت است که ابتدا رشته‌ی اعمال خالی است. سپس به‌ازای تمام اعمال ممکن، آن عمل به آخرین رشته اعمال تولید شده اضافه می‌شود. سپس چک می‌شود که طول رشته (که بیانگر عمقی است که عامل پیش رفته است.) بیشتر از محدودیت عمق (limit) نشود. سپس چک می‌شود که آیا رشته اعمال تولید شده مجاز است یا خیر (شرط مجاز بودن در بخش قبل شرح داده شده است.). اگر مجاز بود، آن نود جدید به explored set و همچنین لیست نودهای بسط داده شده اضافه می‌شود. پس از آن چک می‌شود که آیا آن نود هدف است یا خیر. در صورت هدف بودن تابع خاتمه می‌یابد و یک تابع دیگر به‌نام solution(board, moves, limit) صدا زده می‌شود و راه حل چاپ می‌شود. درصورت هدف نبودن آن نود، تابع limited_DFS به صورت بازگشتی برای همان نود صدا زده می‌شود. به‌این صورت نودها به‌صورت عمقی به پیش می‌روند.

```
107  def solution(board, moves, expanded_nodes, cost):
108      print(f"The solution action sequence is {moves} and the cost is {cost}")
109      print(f"The path nodes are:\n{map_moves_to_nodes(board, moves)}")
110      print(f"The number of expanded nodes is:{len(expanded_nodes)}\n{expanded_nodes}")
111  # found = False
112  def limited_DFS(board, moves, limit):
113      best_moves = moves
114      for action in ["R","L","U","D"]:
115          temp_moves = moves + action
116          if len(temp_moves) > limit:
117              return None
118          if is_valid(board, temp_moves) and not_in_explored_set(board, temp_moves):
119              add_to_expanded_nodes(board, temp_moves)
120              add_to_explored_set(board, temp_moves)
121              if goal_test(board, temp_moves):
122                  # found = True
123                  return solution(board, temp_moves, expanded_nodes, len(temp_moves))
124              limited_DFS(board, temp_moves, limit)
125
```

سپس در تابع جستجوی عمیق‌شونده‌ی تکراری (Iterative_DFS) تابع عمق محدود را برای یک محدودیت عمق خاص (limit) صدا می‌زنیم. سپس main به ازای تمام عمق‌ها (معمولا از عمق 200 به بالا برای این محیط احتیاج نیست.) این تابع صدا زده می‌شود. در خروجی به ازای هر عمق اگر هدف یافت شود راه حل چاپ می‌شود. در غیر این صورت به سراغ عمق بعدی می‌رود.

```python
def iterative_DFS(board, moves, limit):
    limited_DFS(board, moves, limit)

if __name__ == "__main__":
    maze_board = build_board()
    for limit in range(150):
        print(f"Running with limit = {limit}")
        moves = ""
        expanded_nodes.clear()
        explored_set.clear()
        add_to_explored_set(maze_board, "")
        iterative_DFS(maze_board, moves, limit)

```

از دیگر توابع که در بخش اول سطح استفاده شده است نیز استفاده شده است.

3.3 خروجی برنامه

- خروجی **TestCase1** به‌صورت زیر است:

```
Running with limit = 54
The solution action sequence is RRRURULLLLLLLLLLLLLLLLURRULURRRRRRRRRULLLLULLLLLLLLDRR and the cost is 54
The path nodes are:
[[1, 14], [1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7
The number of expanded nodes is:110
[[1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6]
```

همانطور که مشاهده می‌شود هدف را در limit = 54 یافته است و برای عمق‌های قبلی به نتیجه نرسیده است. رشته اعمال RRRURULLLLLLLLLLLLLLLLURRULURRRRRRRRRULLLLULLLLLLLLDRR راه حل است که هزینه‌ی آن برابر 54 است.

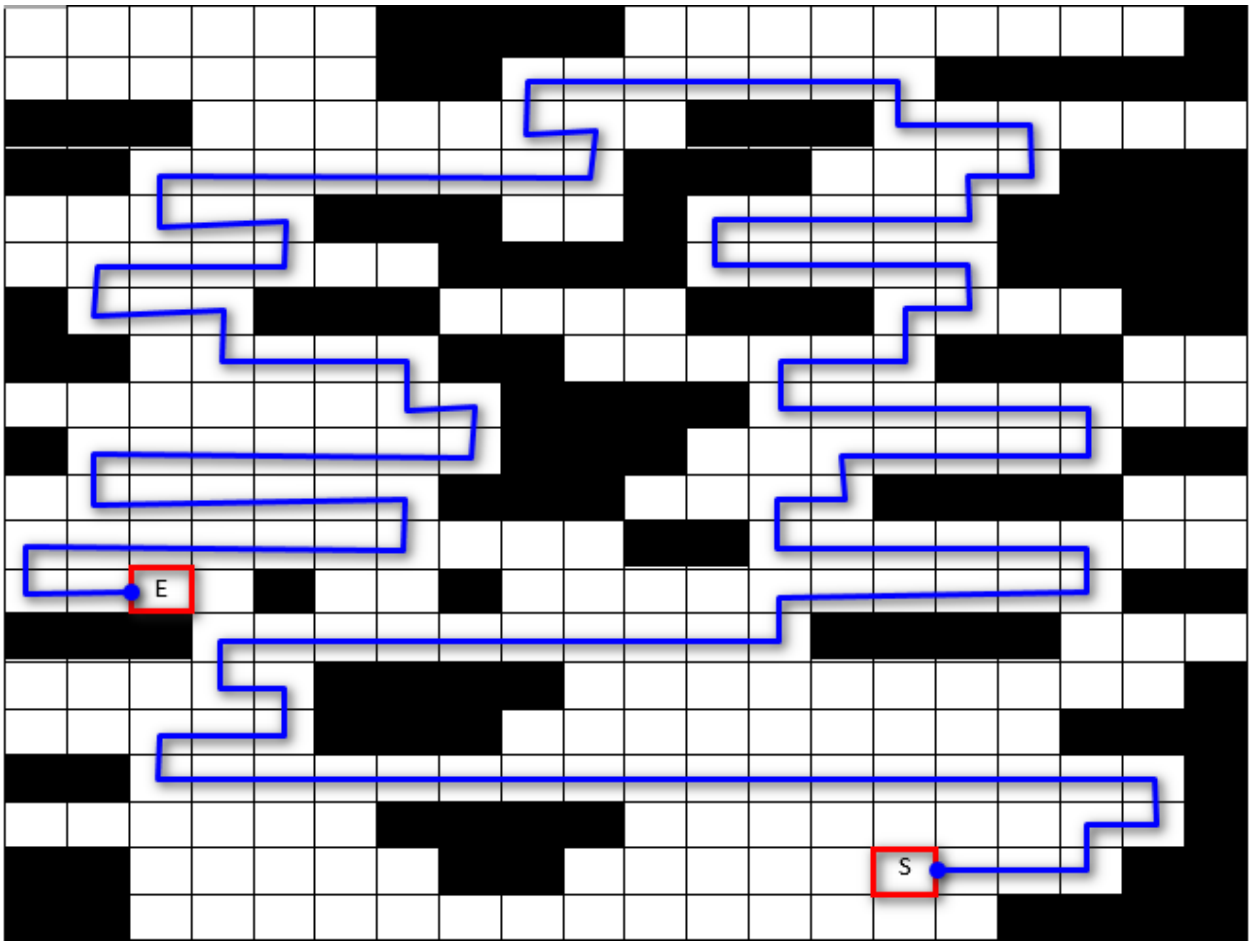
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[1, 14], [1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [3, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 3], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9], [6, 10], [6, 11], [6, 12], [7, 12], [7, 11], [7, 10], [7, 9], [7, 8], [8, 8], [8, 7], [8, 6], [8, 5], [8, 4], [8, 3], [8, 2], [8, 1], [8, 0], [7, 0], [7, 1], [7, 2]

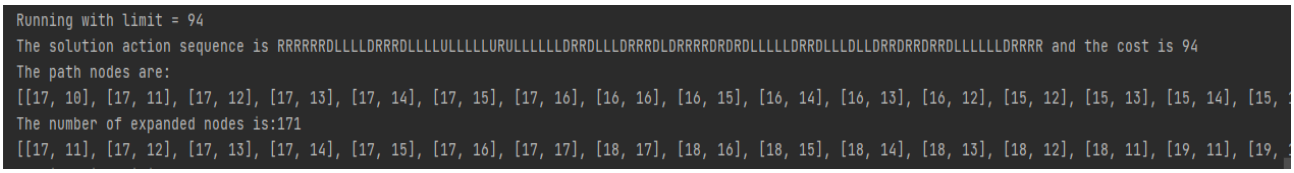
همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 110 نود را شامل می‌شود.

[1, 15], [1, 16], [1, 17], [2, 17], [2, 18], [3, 18], [3, 17], [3, 16], [3, 15], [3, 14], [3, 13], [3, 12], [3, 11], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [3, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 3], [5, 2], [5, 1], [5, 0], [4, 0], [4, 1], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6, 9], [6, 10], [6, 11], [6, 12], [7, 12], [7, 13], [7, 14], [7, 15], [7, 16], [7, 17], [8, 17], [8, 18], [8, 19], [9, 19], [9, 18], [8, 16], [8, 15], [8, 14], [8, 13], [8, 12], [9, 12], [9, 13], [10, 13], [10, 14], [10, 15], [11, 14], [10, 12], [10, 11], [11, 12], [11, 13], [12, 13], [9, 11], [9, 10], [6, 17], [6, 18], [6, 19], [5, 18], [5, 17], [5, 16], [5, 15], [5, 14], [5, 13], [5, 12], [5, 11], [5, 10], [4, 11], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [7, 11], [7, 10], [7, 9], [7, 8], [8, 8], [8, 9], [8, 7], [8, 6], [8, 5], [8, 4], [8, 3], [8, 2], [8, 1], [8, 0], [9, 0], [9, 1], [9, 2], [10, 1], [7, 0], [7, 1], [7, 2]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase1 طی می‌شود به‌صورت زیر است:



- **خروجی TestCase2 به‌صورت زیر است.**



همانطور که مشاهده می‌شود هدف را در limit = 94 یافته است و برای عمق‌های قبلی به نتیجه نرسیده است. رشته اعمال زیر راه حل است که هزینه‌ی آن برابر 94 است.

RRRRRRDLLLLDRRRDLLLLULLLLLURULLLLLLDRRDLLLDRRRDLDRRRRDRDRDLLLLLDRRDLLL
DLLDRRDRRDRRDLLLLLLDRRRR

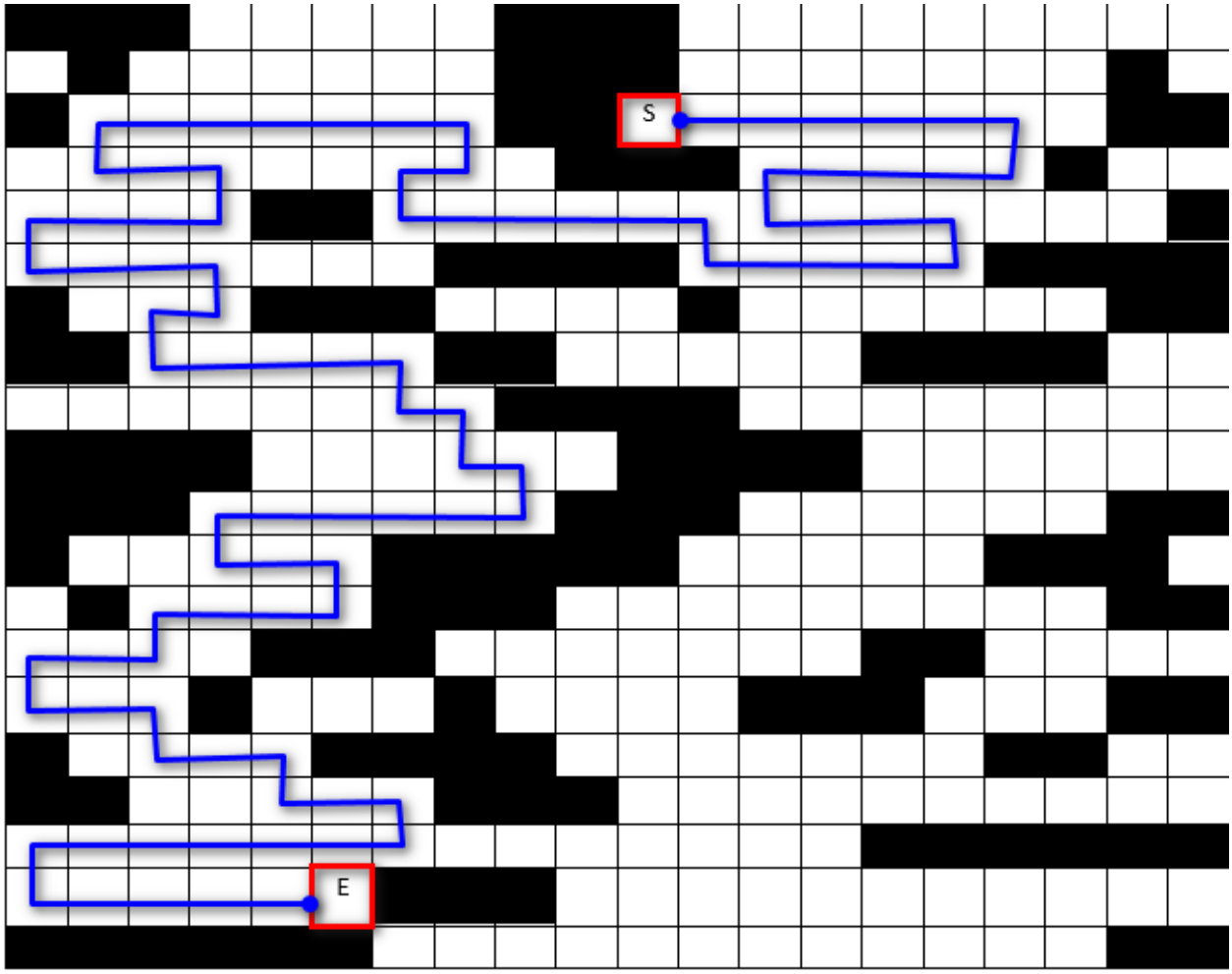
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [17, 13], [17, 14], [17, 15], [17, 16], [16, 16], [16, 15], [16, 14], [16, 13], [16, 12], [15, 12], [15, 13], [15, 14], [15, 15], [14, 15], [14, 14], [14, 13], [14, 12], [14, 11], [15, 11], [15, 10], [15, 9], [15, 8], [15, 7], [15, 6], [16, 6], [16, 7], [17, 7], [17, 6], [17, 5], [17, 4], [17, 3], [17, 2], [17, 1], [16, 1], [16, 2], [16, 3], [15, 3], [15, 2], [15, 1], [15, 0], [14, 0], [14, 1], [14, 2], [14, 3], [13, 3], [13, 2], [12, 2], [12, 3], [12, 4], [12, 5], [12, 6], [11, 6], [11, 7], [10, 7], [10, 8], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [9, 3], [8, 3], [8, 4], [8, 5], [7, 5], [7, 4], [7, 3], [7, 2], [6, 2], [6, 1], [6, 0], [5, 0], [5, 1], [5, 2], [4, 2], [4, 3], [4, 4], [3, 4], [3, 5], [3, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [2, 1], [2, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 171 نود را شامل می‌شود.

[17, 11], [17, 12], [17, 13], [17, 14], [17, 15], [17, 16], [17, 17], [18, 17], [18, 16], [18, 15], [18, 14], [18, 13], [18, 12], [18, 11], [19, 11], [19, 12], [19, 13], [19, 14], [19, 15], [19, 16], [19, 17], [19, 18], [19, 19], [18, 19], [16, 16], [16, 15], [16, 14], [16, 13], [16, 12], [15, 12], [15, 13], [15, 14], [15, 15], [15, 16], [15, 17], [15, 18], [16, 18], [16, 19], [14, 15], [14, 14], [14, 13], [14, 12], [14, 11], [15, 11], [15, 10], [15, 9], [15, 8], [15, 7], [15, 6], [16, 6], [16, 7], [16, 8], [17, 7], [17, 6], [17, 5], [17, 4], [17, 3], [17, 2], [17, 1], [16, 1], [16, 2], [16, 3], [16, 4], [16, 5], [15, 3], [15, 2], [15, 1], [15, 0], [16, 0], [14, 0], [14, 1], [14, 2], [14, 3], [14, 4], [14, 5], [14, 6], [13, 3], [13, 2], [13, 1], [12, 2], [12, 3], [12, 4], [12, 5], [12, 6], [11, 6], [11, 7], [10, 7], [10, 8], [10, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [9, 3], [8, 3], [8, 4], [8, 5], [7, 5], [7, 4], [7, 3], [7, 2], [8, 2], [8, 1], [6, 2], [6, 3], [6, 1], [6, 0], [7, 0], [5, 0], [5, 1], [5, 2], [4, 2], [4, 3], [4, 4], [5, 4], [5, 5], [5, 6], [3, 4], [3, 5], [3, 6], [2, 6], [2, 7], [2, 8], [2, 9], [2, 10], [2, 11], [2, 12], [2, 13], [3, 13], [3, 14], [3, 15], [3, 16], [4, 15], [4, 14], [4, 13], [3, 12], [3, 11], [3, 10], [4, 11], [4, 12], [1, 13], [1, 14], [1, 15], [1, 16], [0, 15], [0, 14], [0, 13], [1, 12], [1, 11], [1, 10], [0, 11], [0, 12], [1, 9], [1, 8], [0, 8], [0, 9], [0, 10], [0, 7], [0, 6], [2, 5], [2, 4], [2, 3], [2, 2], [2, 1], [2, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase2 طی می‌شود به‌صورت زیر است:



- **خروجی TestCase3 به‌صورت زیر است:**

```
Running with limit = 108
The solution action sequence is RRURRRRDRRRRRRRDLLLLLDRRRRRRRRRDLLLLLLLDRRRRRRDLLLLLDRDLLLLLDRRDLLLLLLDRRRRRRDRDRRRDLLDRRRRDLDLLLLDLLLLLLLDR and the cost is
The path nodes are:
[[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [17, 14
The number of expanded nodes is:222
[[18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [18, 15], [19, 1
```

همانطور که مشاهده می‌شود هدف را در limit = 108 یافته است و برای عمق‌های قبلی به نتیجه نرسیده است. رشته اعمال زیر راه حل است که هزینه‌ی آن برابر 108 است.

RRURRRRDRRRRRRRDLLLLLDRRRRRRRRRDLLLLLLLDRRRRRRDLLLLLDRDLLLLLDRRDLLLLLL
DRRRRRRDRDRRRDLLDRRRRDLDLLLLDLLLLLLLDR

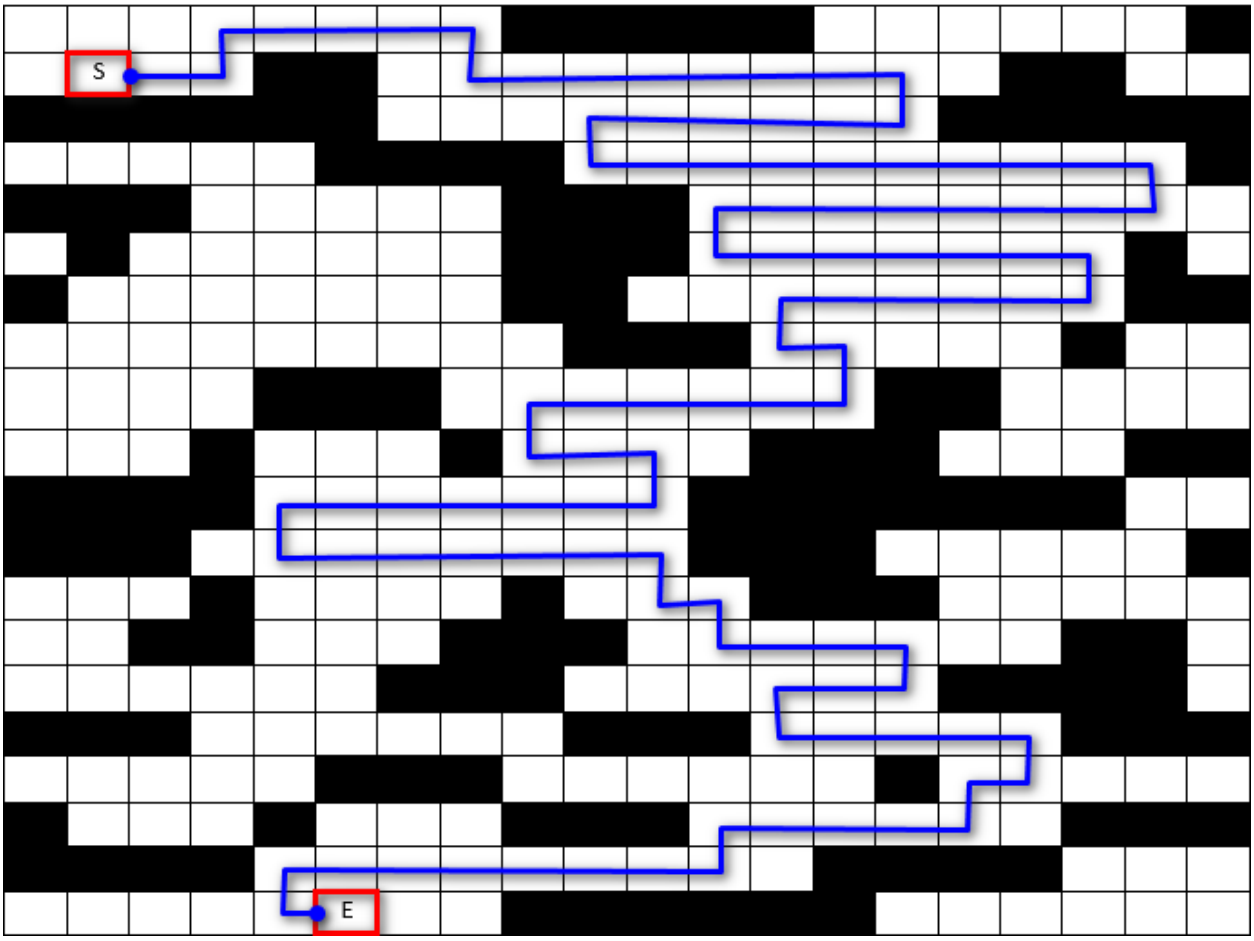
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [17, 14], [17, 13], [17, 12], [17, 11], [17, 10], [17, 9], [16, 9], [16, 10], [16, 11], [16, 12], [16, 13], [16, 14], [16, 15], [16, 16], [16, 17], [16, 18], [15, 18], [15, 17], [15, 16], [15, 15], [15, 14], [15, 13], [15, 12], [15, 11], [14, 11], [14, 12], [14, 13], [14, 14], [14, 15], [14, 16], [14, 17], [13, 17], [13, 16], [13, 15], [13, 14], [13, 13], [13, 12], [12, 12], [12, 13], [11, 13], [11, 12], [11, 11], [11, 10], [11, 9], [11, 8], [10, 8], [10, 9], [10, 10], [9, 10], [9, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [6, 13], [6, 14], [5, 14], [5, 13], [5, 12], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [3, 16], [3, 15], [2, 15], [2, 14], [2, 13], [2, 12], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [1, 4], [0, 4], [0, 5]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 222 نود را شامل می‌شود.

[18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [19, 7], [18, 7], [18, 8], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [18, 15], [19, 15], [19, 16], [19, 17], [19, 18], [18, 18], [18, 19], [19, 14], [19, 13], [17, 14], [17, 13], [17, 12], [17, 11], [17, 10], [17, 9], [17, 8], [17, 7], [17, 6], [18, 6], [16, 9], [16, 10], [16, 11], [16, 12], [16, 13], [16, 14], [16, 15], [16, 16], [16, 17], [16, 18], [15, 18], [15, 19], [14, 19], [15, 17], [15, 16], [15, 15], [15, 14], [15, 13], [15, 12], [15, 11], [14, 11], [14, 12], [14, 13], [14, 14], [14, 15], [14, 16], [14, 17], [13, 17], [13, 16], [13, 15], [13, 14], [13, 13], [13, 12], [13, 11], [13, 10], [12, 12], [12, 13], [12, 14], [12, 15], [12, 16], [11, 16], [11, 17], [11, 18], [11, 19], [12, 19], [12, 18], [10, 17], [10, 16], [10, 15], [11, 13], [11, 12], [11, 11], [11, 10], [11, 9], [11, 8], [11, 7], [12, 7], [12, 8], [12, 6], [12, 5], [12, 4], [12, 3], [12, 2], [12, 1], [12, 0], [11, 0], [11, 1], [11, 2], [11, 3], [10, 2], [10, 1], [10, 0], [13, 1], [13, 2], [13, 3], [13, 4], [13, 5], [13, 6], [13, 7], [14, 7], [14, 6], [14, 5], [14, 4], [14, 3], [14, 2], [15, 3], [15, 4], [15, 5], [15, 6], [15, 7], [16, 4], [16, 3], [16, 2], [16, 1], [16, 0], [10, 8], [10, 9], [10, 10], [10, 11], [9, 10], [9, 9], [9, 8], [9, 7], [9, 6], [9, 5], [9, 4], [10, 4], [10, 5], [10, 6], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8], [8, 9], [8, 10], [7, 10], [7, 11], [6, 11], [6, 12], [6, 13], [6, 14], [6, 15], [6, 16], [7, 16], [7, 17], [7, 18], [7, 19], [6, 19], [5, 19], [8, 18], [8, 17], [8, 16], [8, 15], [8, 14], [7, 15], [9, 18], [9, 19], [5, 14], [5, 13], [5, 12], [5, 11], [5, 10], [5, 9], [6, 10], [4, 12], [4, 13], [4, 14], [4, 15], [4, 16], [3, 16], [3, 17], [3, 18], [3, 19], [3, 15], [2, 15], [2, 16], [2, 14], [2, 13], [2, 12], [2, 11], [3, 11], [3, 12], [3, 13], [3, 10], [3, 9], [3, 8], [4, 8], [4, 7], [4, 6], [4, 5], [4, 4], [4, 3], [5, 4], [3, 4], [5, 5], [6, 5], [1, 11], [1, 12], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [1, 4], [0, 4], [0, 5]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase3 طی می‌شود به‌صورت زیر است:



4 جستجوی A*

4.1 نحوه‌ی عملکرد

این الگوریتم جستجوی آگاهانه از دو تابع برای پیدا کردن بهینه‌ترین مسیر استفاده می‌کند. اگر از اولین گره (ریشه درخت جستجو) شروع کرده باشد و اکنون در گره n باشد، یک تابع g(n) که هزینه مسیر از گره شروع تا گره n است و یک تابع h(n) که تخمینی از ارزان‌ترین مسیر از گره کنونی n تا گره هدف است که به آن تابع اکتشافی (heuristic) گفته می شود را ارائه می‌کند و برای گره n یک تابع f(n) به‌صورت زیر می‌سازد:

$$f(n) = g(n) + h(n)$$

و در هر مرحله گره‌ای را بسط می‌دهد که تابع f برای آن کمینه باشد. دلیل آگاهانه بودن این نوع جستجو، وجود همین تابع اکتشافی است که چشم انداز حدودی از مسیر آینده به عامل می دهد.

اگر تابع اکتشافی هیچگاه هزینه رسیدن به هدف را از مقدار واقعی آن تخمین نزند یک تابع اکتشافی قابل قبول است و تابع اکتشافی h(n) سازگار است اگر برای هر گره n و هر پسین آن گره، n' که با عمل a تولید شده است، هزینه‌ی برآورد شده رسیدن به هدف از n بزرگتر از هزینه گام رسیدن به n' به علاوه هزینه برآورد شده رسیدن به هدف از n' نباشد.

این الگوریتم یک روش جستجوی کامل (در صورتی که ضریب انشعاب متناهی باشد.) و بهینه (اگر تابع heuristic در جستجوی گراف سازگار و در جستجوی درخت قابل قبول باشد.) است. پیچیدگی زمانی و فضایی آن $O(b^{\epsilon d})$ است. (ϵ میزان خطای نسبی تابع اکتشافی است.)

4.2 الگوریتم و کد پیاده‌سازی

همانطور که در بالا شرح داده شد، الگوریتم به‌این صورت پیاده‌سازی شده که در هر مرحله نود با کمترین f(n) را بسط می‌دهد. در کد تابع f(n) را heuristic نامیده‌ایم. همچنین تابع g(n) را distance_cost و تابع h(n) را goal_distance نامیده‌ایم. تابع هیوریستیک پیشنهادی ما (goal_distance) فاصله‌ی منهتن نود n تا نود هدف برمی‌گرداند. پیاده‌سازی این سه تابع به‌شکل زیر بوده است.

```python
def manhattan_distance(firs_node, second_node):
    return abs(firs_node[0] - second_node[0]) + abs(firs_node[1] - second_node[1])

def get_goal_node(board):
    goal_node = []
    for i in range(board_size):
        for j in range(board_size):
            if board[i][j] == "0":
                goal_node.append(i)
                goal_node.append(j)
    return goal_node

#the distance of one node on the board to the goal node
def goal_distance(board, moves):
    node = map_moves_to_node(board, moves)
    goal_node = get_goal_node(board)
    return manhattan_distance(node, goal_node)

#the cost of a node from agent node
def distance_cost(board, moves):
    return len(moves)
    # node = map_moves_to_node(board, moves)
    # return manhattan_distance(get_goal_node(board), node)
```

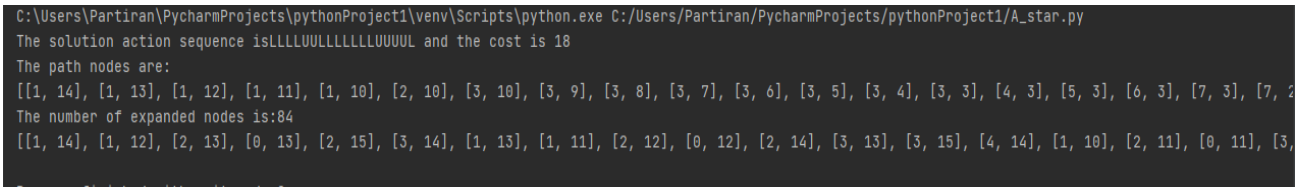
```python
def heuristic(board, moves):
    return goal_distance(board, moves) + distance_cost(board, moves)

```

تابع اصلی این جستجو a_star نام دارد. یک لیست از نودهای فعلی که آماده‌ی بسط داده شدن هستند، وجود دارد که در ابتدا نودهای ابتدایی بر اساس اعمال مجاز از نود اولیه‌ی حرکت عامل به این لیست اضافه می‌شود. سپس در یک حلقه‌ی بی‌نهایت هر سری از میان نودهای آماده‌ی بسط داده شدن بهترین نود که کمترین heuristic را داراست انتخاب می‌شود. سپس چک می‌شود که ایا نود هدف است یا خیر. اگر هدف بود حلقه break می‌شود. درغیر این صورت آن نود بسط داده می‌شود و از لیست نودهای آماده‌ی بسط داده شدن حذف می‌شود. سپس تمام نودهای فرزند این نود که با هرکدام از چهار عمل، قابل دسترسی هستند، به لیست نودهای آماده بسط داده شدن اضافه می‌شوند. این کار آنقدر ادامه می‌یابد تا هدف یافت شود و return شود.

```python
def a_star(board):
    best_node = ""
    current_nodes = []
    temp_node = ""
    add_to_explored_set(board, get_agent_node(board))
    #adding initial moves
    for action in ["R", "L", "U", "D"]:
        temp_node = ""
        temp_node = temp_node + action
        if is_valid(board, temp_node):
            current_nodes.append(temp_node)
    while True:
        best_node = current_nodes[0]
        for node in current_nodes:
            if heuristic(board, node) < heuristic(board, best_node):
                best_node = node
        if goal_test(board, best_node):
            break
        current_nodes.remove(best_node)
        for action in ["R", "L", "U", "D"]:
            temp_node = best_node + action
            if is_valid(board, temp_node) and not_in_explored_set(board, temp_node):
                current_nodes.append(temp_node)
                add_to_explored_set(board, temp_node)
                add_to_expanded_nodes(board, temp_node)
    return best_node
```

4.3 خروجی برنامه

- خروجی **TestCase1** به‌صورت زیر است:



مشاهده می‌شود که رشته اعمال LLLLUULLLLLLLUUUUL راه حل است که هزینه‌ی آن برابر 18 است.

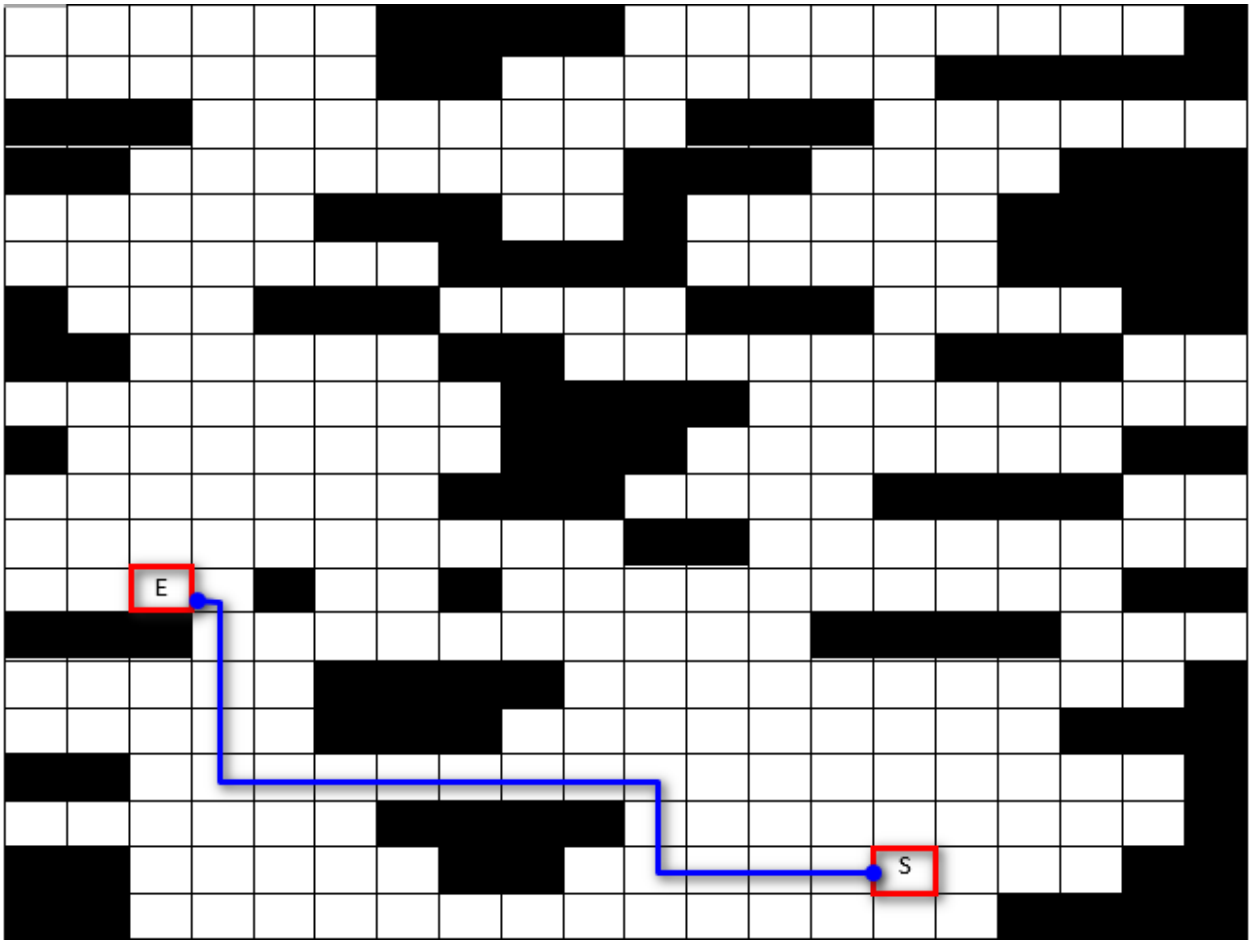
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[1, 14], [1, 13], [1, 12], [1, 11], [1, 10], [2, 10], [3, 10], [3, 9], [3, 8], [3, 7], [3, 6], [3, 5], [3, 4], [3, 3], [4, 3], [5, 3], [6, 3], [7, 3], [7, 2]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 84 نود را شامل می‌شود.

[1, 14], [1, 12], [2, 13], [0, 13], [2, 15], [3, 14], [1, 13], [1, 11], [2, 12], [0, 12], [2, 14], [3, 13], [3, 15], [4, 14], [1, 10], [2, 11], [0, 11], [3, 12], [4, 13], [4, 15], [5, 14], [1, 9], [2, 10], [0, 10], [3, 11], [4, 12], [5, 13], [5, 15], [0, 9], [3, 10], [4, 11], [5, 12], [3, 9], [4, 10], [5, 11], [6, 12], [3, 8], [4, 9], [5, 10], [6, 11], [7, 12], [3, 7], [4, 8], [5, 9], [6, 10], [7, 11], [7, 13], [8, 12], [3, 6], [6, 9], [7, 10], [3, 5], [6, 8], [7, 9], [3, 4], [2, 5], [6, 7], [7, 8], [8, 9], [3, 3], [4, 4], [2, 4], [6, 6], [8, 8], [3, 2], [4, 3], [2, 3], [5, 4], [6, 5], [7, 6], [4, 2], [2, 2], [5, 3], [6, 4], [7, 5], [8, 6], [4, 1], [5, 2], [6, 3], [8, 5], [5, 1], [7, 3], [7, 2], [8, 3]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase1 طی می‌شود به‌صورت زیر است:



- خروجی **TestCase2** به‌صورت زیر است:

```
C:\Users\Partiran\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Partiran/PycharmProjects/pythonProject1/A_star.py
The solution action sequence is RRDDDDDDRRDDLLDLDLDDDDDLLLLLLD and the cost is 30
The path nodes are:
[[17, 10], [17, 11], [17, 12], [16, 12], [15, 12], [14, 12], [13, 12], [12, 12], [11, 12], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12],
The number of expanded nodes is:103
[[17, 12], [17, 10], [18, 11], [17, 11], [17, 13], [18, 12], [16, 12], [19, 11], [16, 13], [15, 12], [15, 13], [15, 11], [14, 12], [14, 11], [14, 13], [13, 1
```

مشاهده می‌شود که رشته اعمال RRDDDDDDRRDDLLDLDLDDDDDLLLLLLD راه حل است که هزینه‌ی آن برابر 30 است.

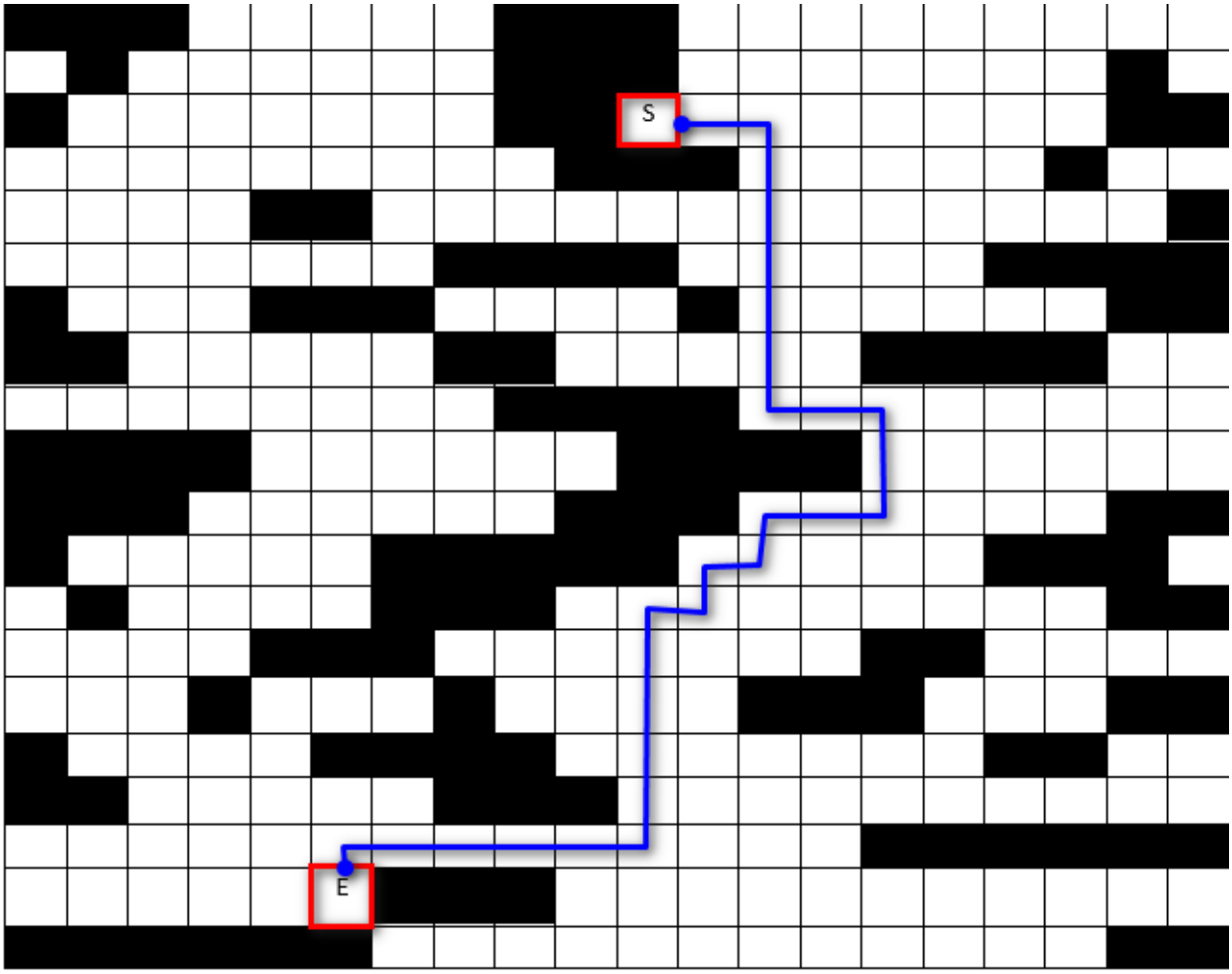
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[17, 10], [17, 11], [17, 12], [16, 12], [15, 12], [14, 12], [13, 12], [12, 12], [11, 12], [11, 13], [11, 14], [10, 14], [9, 14], [9, 13], [9, 12], [8, 12], [8, 11], [7, 11], [7, 10], [6, 10], [5, 10], [4, 10], [3, 10], [2, 10], [2, 9], [2, 8], [2, 7], [2, 6], [2, 5], [2, 4], [1, 4]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 103 نود را شامل می‌شود.

[17, 12], [17, 10], [18, 11], [17, 11], [17, 13], [18, 12], [16, 12], [19, 11], [16, 13], [15, 12], [15, 13], [15, 11], [14, 12], [14, 11], [14, 13], [13, 12], [13, 13], [12, 12], [12, 13], [12, 11], [11, 12], [12, 10], [11, 13], [12, 9], [13, 10], [13, 9], [17, 14], [18, 13], [19, 12], [16, 14], [15, 14], [14, 14], [13, 14], [11, 14], [13, 8], [13, 7], [17, 15], [18, 14], [19, 13], [16, 15], [15, 15], [14, 15], [13, 15], [11, 15], [10, 14], [10, 15], [9, 14], [9, 15], [9, 13], [8, 14], [9, 12], [8, 13], [8, 15], [7, 14], [8, 12], [7, 13], [7, 15], [8, 11], [7, 12], [6, 13], [7, 11], [6, 12], [7, 10], [6, 11], [7, 9], [6, 10], [5, 11], [6, 9], [5, 10], [4, 11], [6, 8], [5, 9], [4, 10], [4, 12], [3, 11], [6, 7], [5, 8], [4, 9], [3, 10], [3, 12], [2, 11], [2, 10], [2, 12], [1, 11], [2, 9], [1, 10], [1, 12], [0, 11], [2, 8], [1, 9], [0, 10], [2, 7], [1, 8], [0, 9], [2, 6], [0, 8], [2, 5], [3, 6], [2, 4], [3, 5], [2, 3], [3, 4], [1, 4]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase2 طی می‌شود به‌صورت زیر است:



- خروجی **TestCase3** به‌صورت زیر است:

```
C:\Users\Partiran\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Partiran/PycharmProjects/pythonProject1/A_star.py
The solution action sequence is RRURRRDDRRRDRRDDDRDDLLDDDDDDRRDDLDDLLLLLLD and the cost is 42
The path nodes are:
[[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [18, 6], [17, 6], [17, 7], [17, 8], [17, 9], [16, 9], [16, 10], [16, 11], [15, 11], [14, 11],
The number of expanded nodes is:162
[[18, 3], [18, 1], [19, 2], [18, 2], [19, 3], [19, 0], [18, 0], [19, 1], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [17, 7], [18, 8], [17
```

مشاهده می‌شود که رشته اعمال RRURRRDDRRRDRRDDDRDDLLDDDDDDRRDDLDDLLLLLLD راه حل است که هزینه‌ی آن برابر 42 است.

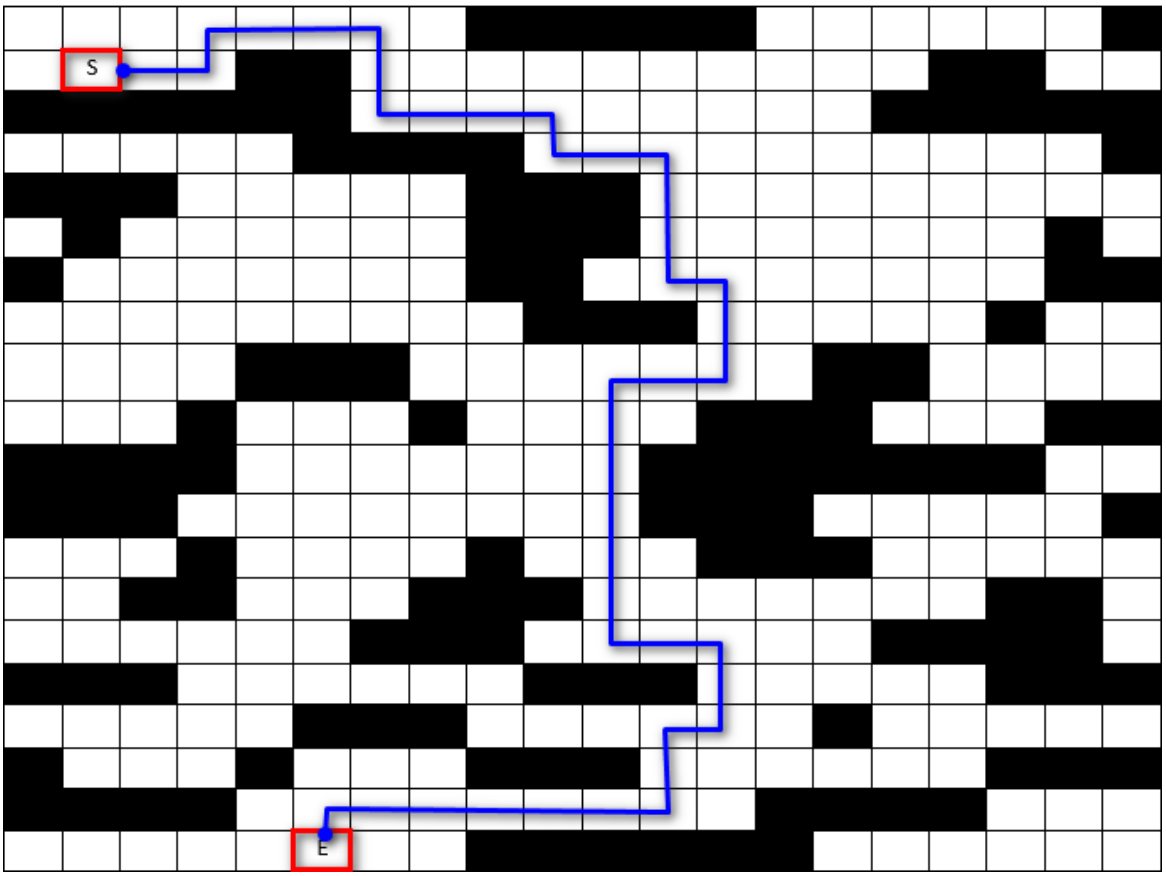
همچنین نودهای مسیر از ابتدا تا مقصد به‌صورت زیر هستند:

[18, 1], [18, 2], [18, 3], [19, 3], [19, 4], [19, 5], [19, 6], [18, 6], [17, 6], [17, 7], [17, 8], [17, 9], [16, 9], [16, 10], [16, 11], [15, 11], [14, 11], [13, 11], [13, 12], [12, 12], [11, 12], [11, 11], [11, 10], [10, 10], [9, 10], [8, 10], [7, 10], [6, 10], [5, 10], [5, 11], [5, 12], [4, 12], [3, 12], [3, 11], [2, 11], [1, 11], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [1, 5], [0, 5]

همچنین ترتیب نودهای بسط داده شده به‌صورت زیر است که 162 نود را شامل می‌شود.

[18, 3], [18, 1], [19, 2], [18, 2], [19, 3], [19, 0], [18, 0], [19, 1], [19, 4], [19, 5], [19, 6], [19, 7], [18, 6], [18, 7], [17, 6], [17, 7], [18, 8], [17, 8], [18, 9], [17, 9], [18, 10], [17, 10], [16, 9], [16, 10], [18, 11], [17, 11], [16, 11], [18, 12], [17, 12], [16, 12], [15, 11], [15, 12], [14, 11], [14, 12], [13, 11], [13, 12], [13, 10], [18, 13], [17, 13], [16, 13], [15, 13], [14, 13], [13, 13], [12, 12], [12, 13], [11, 12], [11, 13], [11, 11], [11, 10], [10, 11], [11, 9], [10, 10], [11, 8], [10, 9], [9, 10], [11, 7], [12, 8], [10, 8], [9, 9], [8, 10], [12, 7], [9, 8], [8, 9], [7, 10], [9, 7], [8, 8], [7, 9], [7, 11], [6, 10], [9, 6], [8, 7], [6, 11], [5, 10], [9, 5], [10, 6], [8, 6], [7, 7], [5, 11], [5, 9], [9, 4], [10, 5], [8, 5], [7, 6], [8, 4], [7, 5], [6, 6], [7, 4], [6, 5], [6, 4], [5, 5], [5, 4], [4, 5], [4, 6], [4, 4], [18, 14], [19, 13], [17, 14], [16, 14], [15, 14], [14, 14], [13, 14], [12, 14], [12, 6], [13, 7], [6, 12], [5, 12], [10, 4], [8, 3], [5, 3], [4, 7], [4, 3], [3, 4], [12, 5], [13, 6], [3, 3], [12, 4], [13, 5], [18, 15], [19, 14], [16, 15], [15, 15], [14, 15], [13, 15], [12, 15], [14, 7], [6, 13], [5, 13], [4, 12], [5, 2], [4, 8], [14, 6], [3, 2], [2, 3], [12, 3], [13, 4], [14, 5], [4, 13], [3, 12], [2, 2], [3, 13], [3, 11], [2, 12], [3, 10], [2, 11], [2, 13], [1, 12], [3, 9], [1, 11], [3, 8], [1, 10], [1, 9], [1, 8], [1, 7], [1, 6], [2, 7], [0, 7], [1, 5], [2, 6], [0, 6], [1, 4], [2, 5], [0, 5]

مسیری که توسط این الگوریتم در صفحه‌ی بازی TestCase3 طی می‌شود به‌صورت زیر است:



5 مقایسه

همانطور که گفتیم هر سه روش جستجو با رعایت کردن شرایطی، کامل و بهینه هستند و نیز گفتیم که پیچیدگی زمانی و فضایی هر دو روش جستجوی BFS و A^* نمایی و پیچیدگی زمانی IDFS نیز نمایی بود. بنابراین سوالی که پیش می آید این است که کدام یک بهینه تر و مناسب تر است؟

برای درک بهتر تفاوت های این سه روش جستجو، می توانیم نتایج حاصل شده برای هر سه Test Case را با هم مقایسه کنیم.

	BFS		IDFS		A^*	
	تعداد نود بسط داده شده	Cost	تعداد نود بسط داده شده	Cost	تعداد نود بسط داده شده	Cost
TestCase1	164	18	110	54	84	18
TestCase2	267	30	171	94	103	30
TestCase3	256	42	222	108	162	42

با توجه به داده هایی که در جدول می‌بینیم و نیز با علم به این که جستجو A^* آگاهانه است، **بهترین و بهینه ترین روش از بین سه روش پیشنهاد شده، روش جستجوی A^* است.**

اگر بخواهیم با جزییات بیشتر به مقایسه بپردازیم، می‌بینیم که از نظر هزینه صرف شده برای رسیدن به هدف، جستجوی A^* و BFS بهترین عملکرد را داشته و بهتر از IDFS عمل کرده‌اند.

از نظر تعداد نود بسط داده شده، A^* بهترین عملکرد را داشته و پس از آن IDFS از BFS بهتر عمل کرده است.

از نظر پیچیدگی زمانی در حین Run کردن کد دیدیم روش جستجوی A^* بهترین عملکرد را داشت و روش جستجوی IDFS و BFS تقریبا مشابه هم و بیشتر از روش A^* زمان برد تا به هدف برسند. البته این نتیجه با توجه به پیچیدگی زمانی کلی حساب شده برای دو روش ناآگاهانه که $O(b^d)$ و برای روش آگاهانه که $O(b^{\epsilon d})$ بود نیز واضح است. زیرا ϵ خطای نسبی است و مقداری اعشاری و بسیار کوچک دارد و در نتیجه یک عدد نمایی کوچکتر به نسبت دو روش دیگر ایجاد می‌کند.

از نظر پیچیدگی فضایی نیز با توجه به اینکه برای روش IDFS یک رابطه غیر نمایی به‌صورت $O(bd)$ داشتیم، بنابراین، این روش بهینه ترین است و بعد از آن A^* از BFS بهتر است زیرا برای روش A^* ، توان رابطه نمایی در یک اپسیلون اعشاری کوچک ضرب می شود و بنابراین پیچیدگی فضایی کمتری نسبت به BFS خواهد داشت.