HW1

آرین جلالیان 99243026

1. با توجه به اینکه branching factor در بدترین حالت 4 است (up, dpwn, right, left) پیچیدگی bfs از $O(4^d)$ است که d عمق درخت در node هدف است. پیچیدگی زمانی dfs برابر $O(4^m)$ است که m برابر ماکزیمم طول درخت است. پیچیدگی زمانی A* هم مانند bfs است.
یک کلاس node داریم که در آن state و father و path cost و total cost داریم. در این کلاس یک تابع is_goal وجود دارد که بررسی می کند node ما node هدف است یا خیر.
یک تابع calculate_heuristic داریم که فاصله node کنونی تا node هدف را حساب می کند.

```python
class Node:
    def __init__(self, state, father):
        """
        args =
        state : an dict containing board as a np array of 0s and 1s
        a agent as a tuple of cordinates and a end as a tuple of cordinates
        father : node
        """
        self.state = state
        self.father = father
        self.path_cost = 0 if father is None else father.path_cost + 1
        self.total_cost = self.path_cost + self.calculate_heuristic()



    def is_goal(self):
        return self.state['agent'] == self.state['end']

    def calculate_heuristic(self):
        x_start, y_start = self.state['agent']
        x_end, y_end = self.state['end']

        return math.sqrt((x_start-x_end)**2 + (y_start-y_end)**2)
```

یک تابع make_child داریم که یک father و یک action میگیرد و node حاصل از این action را برمیگرداند.

```python
def make_child(father, action):
    """
    args =
    father : node
    action : tuple of direction

    returns a node
    """
    new_state = {}
    new_state['board'] = father.state['board']
    new_state['end'] = father.state['end']

    new_agent_pos_x, new_agent_pos_y = father.state['agent'][0] + action[0], father.state['agent'][1] + action[1]
    new_state['agent'] = (new_agent_pos_x, new_agent_pos_y)

    child = Node(new_state, father)
    return child
```

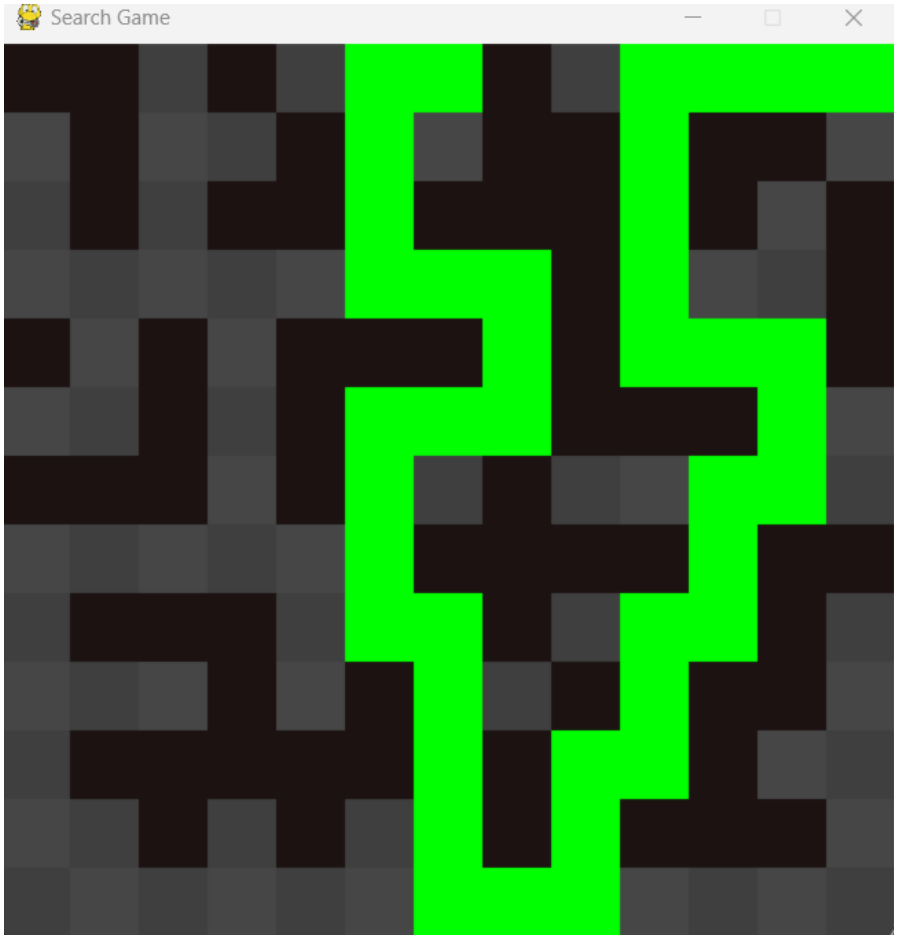
یک تابع get_action داریم که بررسی می کند از node فعلی به کدام node ها می توانیم برویم و یک لیست از tuple های دوتایی برمی گرداند.

```python
def get_actions(state):
    """
    args =
    state : an dict containing board as a np array of 0s and 1s
    a agent as a tuple of cordinates and a end as a tuple of cordinates

    retruns a list of tuples that are corresponding directions
    """
    actions = []

    x, y = state['agent']
    if x + 1 < 13 and state['board'][y, x + 1] == 1 :
        actions.append((1,0))

    if x - 1 >= 0 and state['board'][y, x - 1] == 1 :
        actions.append((-1,0))

    if y + 1 < 13 and state['board'][y + 1, x] == 1 :
        actions.append((0,1))

    if y - 1 >= 0 and state['board'][y - 1, x] == 1 :
        actions.append((0,-1))

    return actions
```

تابع solution هم یک لیست از مختصات node هایی که تا رسیدن به هدف از آن ها عبور کردیم را برمی گرداند. تابع color_path هم لیستی که در تابع solution برگردانده شده است را با استفاده از تابع colorize رنگ می کند.

```python
def solution(node, curr_path):
    """
    args = node

    returns a list of nodes cordinates  (tuple)
    """
    curr_path.append(node.state['agent'])

    if node.father == None :
        return curr_path

    return solution(node.father, curr_path)

def path_color(board, path):
    """
    args =
    board : a Board object
    path : list of cordinations from start to end
    """
    for cor in path :
        board.colorize(cor[0], cor[1], colors.green)
```

یک کلاس agent جدید تعریف شده است که تابع fs الگوریتم های bfs و dfs را پیاده سازی می کند.
یک آرگومان kind داریم که اگر "b" باشد bfs است است و از اول لیست node ها را خارج می کند.
اگر "d" باشد dfs است و از انتهای لیست node ها را بررسی می کند.
در تابع a_star هر دفعه لیست frontier را بر اساس total cost هر node سورت می کنیم و از اول لیست برای expand کردن می خوانیم.
کلاس board هم تغییر کرده است و به جای اینکه در آن فایل Maze.npy را load کنیم در تابع main این کار را انجام می دهیم و آرایه numpy این maze را به آن پاس می دهیم.
اجرای bfs روی maze به این صورت است :



اجرای dfs هم به این صورت است :

2. اجرای A* هم به این صورت است :

در bfs برای expand کردن node ها از ابتدای لیست frontier می خوانیم و node های فرزند را به انتهای لیست frontier اضافه می کنیم در نتیجه node ها در یک level درخت اول expand می شوند و در dfs برای expand کردن از انتهای لیست frontier می خوانیم و در نتیجه پایین ترین node اول expand می شود. در A* اولویت گسترش expand کردن با node ای است که total cost کمتری دارد که این total cost از جمع path cost و heuristic به دست می آید.

3. اجرای الگوریتم dfs برای maze جدید به این شکل است :