CSE4185 기초 인공지능 과제 1 보고서

20190554 송재현

사용한 라이브러리

```
from collections import deque, defaultdict
import heapq
```

1. deque
BFS와 DFS에서 큐와 스택을 보다 효율적인 deque를 이용하여 구현함

2. defaultdict
MST를 만들 때의 필요한 그래프의 정보를 저장하기 위해 사용함

3. heapq
MST를 만들 때 prim 알고리즘을 사용하였는데 이 때 사용함

BFS Method

Code

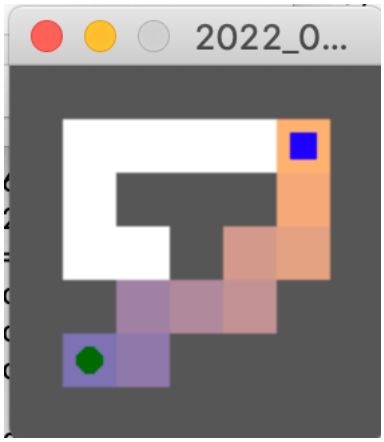
```
def bfs(maze):
    """
    [Problem 01] 제시된 stage1 맵 세 가지를 BFS Algorithm을 통해 최단 경로를
return하시오.
    """
    start_point=maze.startPoint()
    path=[]
    ######################### Write Your Code Here
##############################
    end_point = maze.circlePoints()[0]

    queue = deque()
    queue.append(start_point)

    visited = []
    parent = {}

    while queue:
        y, x = queue.popleft()

        if (y, x) == end_point:
            break

        for next_y, next_x in maze.neighborPoints(y, x):
            if (next_y, next_x) not in visited:
                visited.append((next_y,next_x))
                queue.append((next_y,next_x))
                parent[(next_y, next_x)] = (y,x)
```

```
    path = [(y,x)]
    while path[-1] != start_point:
        path.append(parent[path[-1]])
    path.reverse()

    result = maze.isValidPath(path)
    if result != 'Valid':
        print(result)

    return path
```

```
##############################################################################
###
```

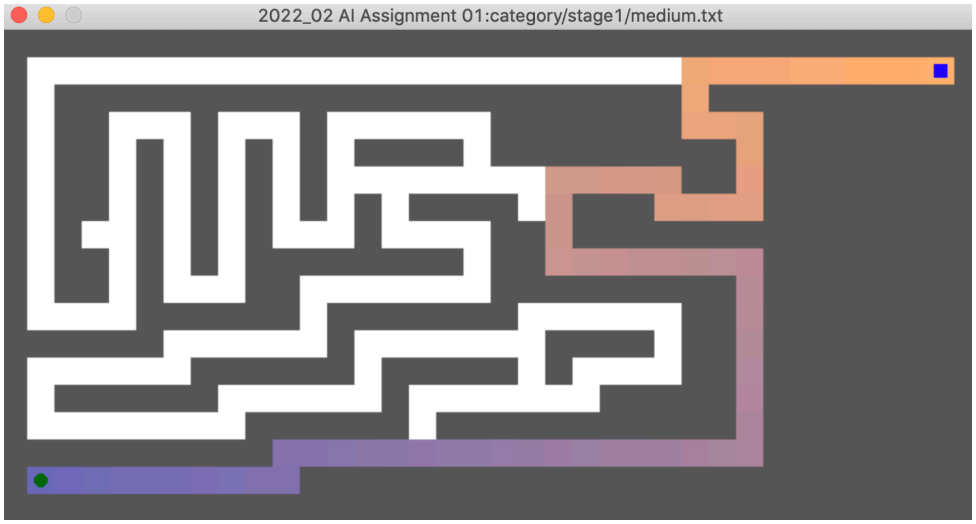
1. Small.txt

```
==========================================
[ bfs results ]
(1) Path Length: 9
(2) Search States: 16
(3) Execute Time: 0.0001709461 seconds
------------------------------------------
```



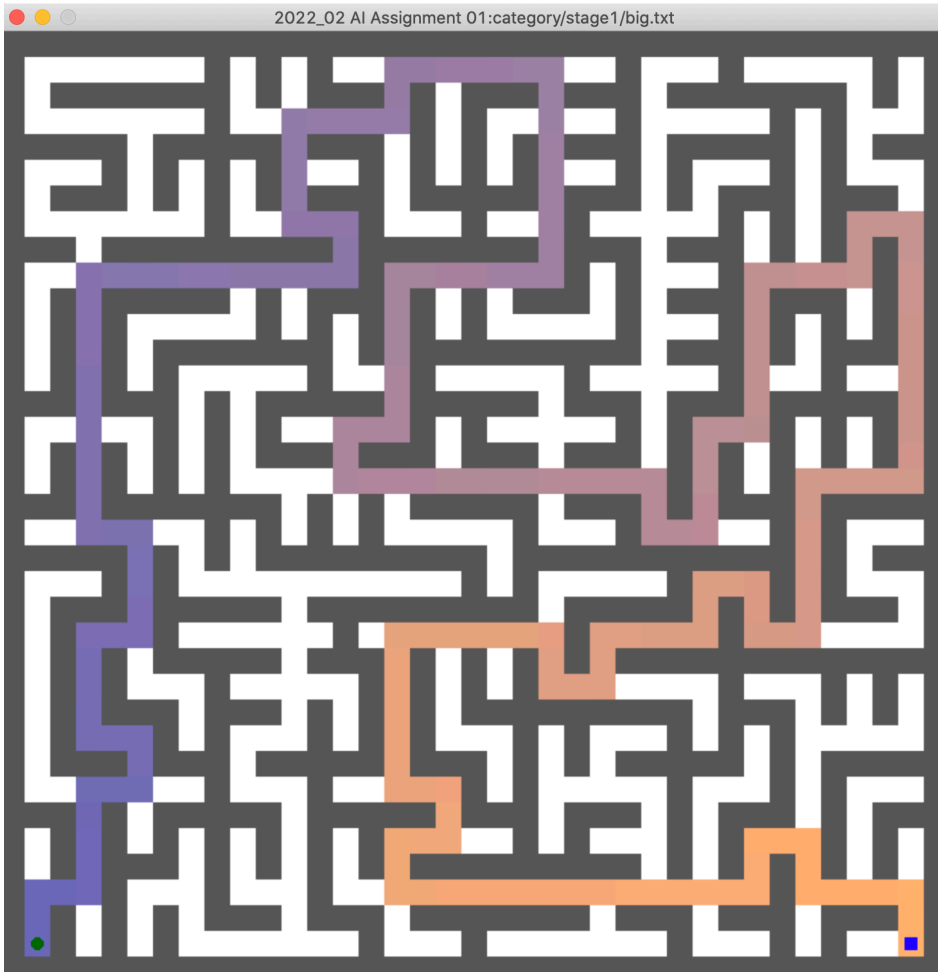
2. Medium.txt

```
==========================================
[ bfs results ]
(1) Path Length: 69
(2) Search States: 220
(3) Execute Time: 0.0019221306 seconds
------------------------------------------
```



3. Big.txt

```
==========================================
[ bfs results ]
(1) Path Length: 211
(2) Search States: 620
(3) Execute Time: 0.0093398094 seconds
==========================================
```



구현 방법

deque 라이브러리를 이용해 이를 queue로 활용하여 bfs를 구현하였다. Visited 리스트는 현재 좌표가 이미 방문된 좌표인지를 기록하기 위하여 사용하였고 visited 리스트에 좌표가 없다면 이를 queue에 append하여 방문하도록 하였다. 이 때 현재 위치에서 이동할 수 있는 좌표들은 maze.py에 있는 neigborPoints함수를 이용하여 구하였다. 이렇게 bfs를 구현하였고 이 때 경로를 return하기 위해서 각 좌표가 어디서 왔는지 parent를 따로 기록하였는데 dict를 이용하여 구현하였다. Key 값을 현재 좌표로 value 값을 parent 좌표로 하여 기록하였다. 이를 통해 path를 역추적할 수 있었다. 마지막으로 maze.py파일의 isValidPath 함수를 통해 이 path가 유효한지 end_point에 도달했는지를 확인하였다.

DFS Method

Code

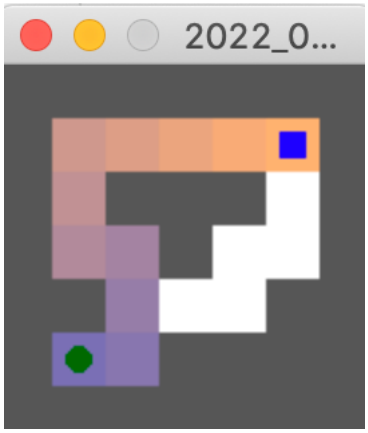
```
def dfs(maze):
    """
    [Problem 02] 제시된 stage1 맵 세 가지를 DFS Algorithm을 통해 최단 경로를
return하시오.
    """
    start_point = maze.startPoint()
    path = []
    ######################## Write Your Code Here
################################

    end_point = maze.circlePoints()[0]

    stack = deque()
    stack.append(start_point + (-1,-1))

    visited = []
    parent = {}

    while stack:
        y, x, prev_y, prev_x = stack.pop()

        if (y, x) == end_point:
            parent[(y,x)] = (prev_y, prev_x)
            break

        if (y, x) not in visited:
            visited.append((y, x))
            parent[(y, x)] = (prev_y, prev_x)
            for next_y, next_x in maze.neighborPoints(y, x):
                stack.append((next_y,next_x, y, x))

    path = [(y,x)]
    while path[-1] != start_point:
        path.append(parent[path[-1]])
    path.reverse()

    result = maze.isValidPath(path)
```

```
    if result != 'Valid':
        print(result)

    return path
```

```
##############################################################################
###
```

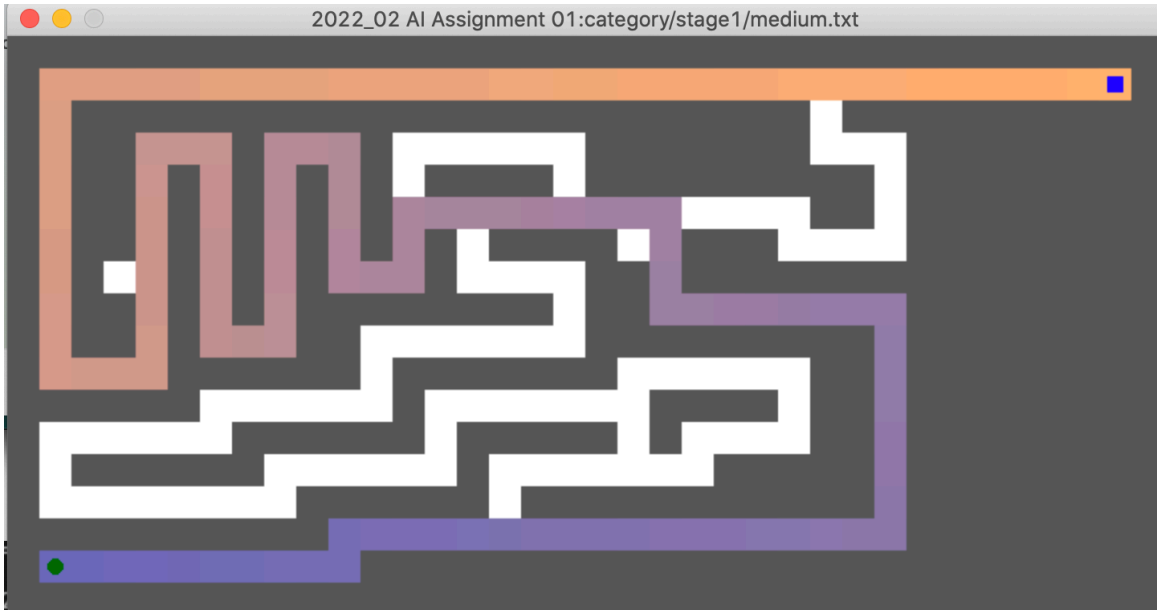
1. small.txt

```
==========================================
[ dfs results ]
(1) Path Length: 11
(2) Search States: 15
(3) Execute Time: 0.0001740456 seconds
==========================================
```



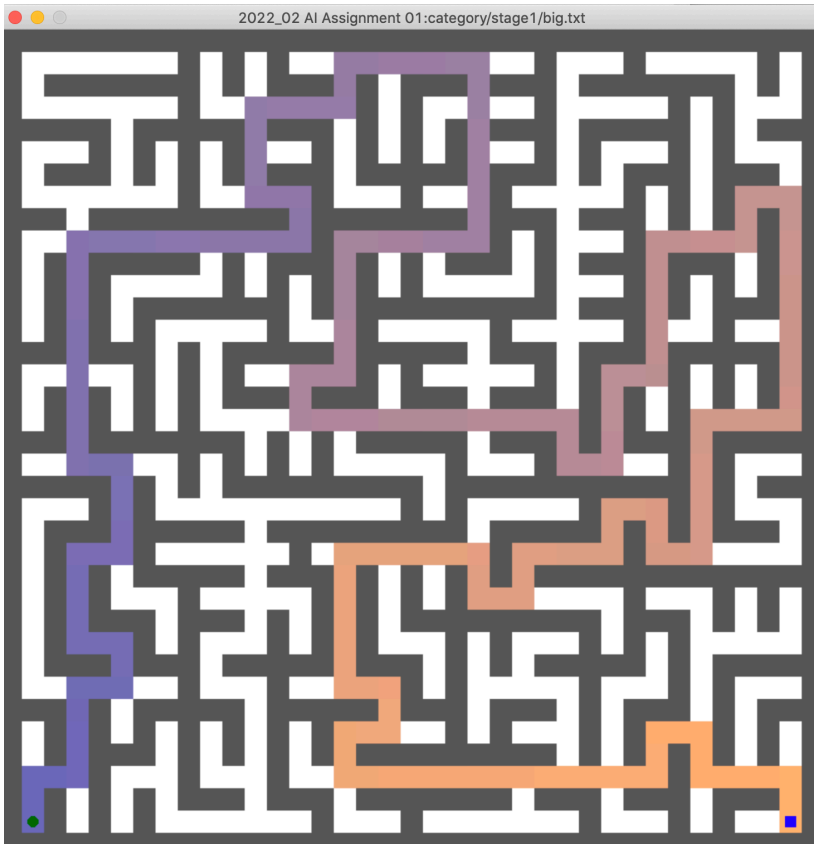
2. medium.txt

```
==========================================
[ dfs results ]
(1) Path Length: 131
(2) Search States: 146
(3) Execute Time: 0.0009977818 seconds
==========================================
```



3. big.txt

```
==========================================
[ dfs results ]
(1) Path Length: 211
(2) Search States: 426
(3) Execute Time: 0.0048677921 seconds
==========================================
```



구현 방법

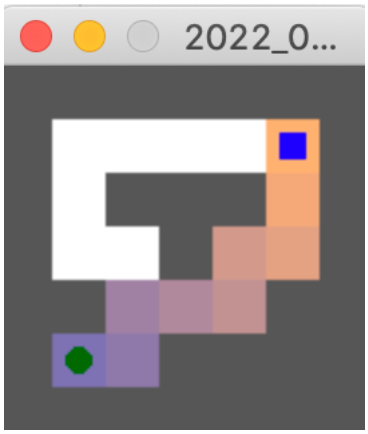
deque 라이브러리를 이용해 이를 stack으로 활용하여 dfs를 구현하였다. Visited 리스트는 현재 좌표가 이미 방문된 좌표인지를 기록하기 위하여 사용하였고 visited 리스트에 좌표가 없다면 주변 좌표들을 neighborPoints를 이용하여 구한 뒤에 stack에 append하여 search 하도록 하였다. 이렇게 dfs를 구현하였고 이 때 경로를 return하기 위해서 각 좌표가 어디서 왔는지 parent를 따로 기록하였는데 dict를 이용하여 구현하였다. Key 값을 현재 좌표로 value 값을 parent 좌표로 하여 기록하였다. 이를 통해 path를 역추적할 수 있었다. 또한 stack에 parent의 좌표를 같이 append 해줘서 parent 좌표를 기록할 수 있도록 하였다. 마지막으로 maze.py파일의 isValidPath 함수를 통해 이 path가 유효한지 end_point에 도달했는지를 확인하였다.

Astar Method

Code

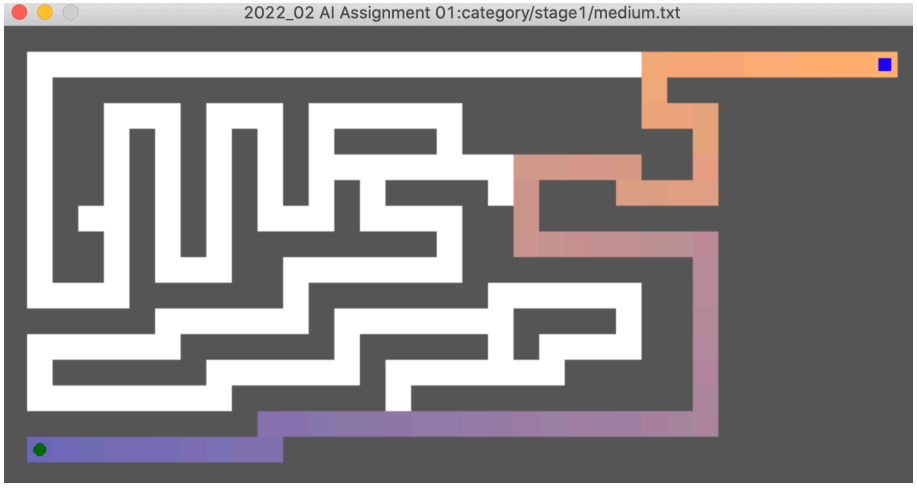
```
def astar(maze):
    """
    [Problem 03] 제시된 stage1 맵 세가지를 A* Algorithm을 통해 최단경로를 return하시오.
    (Heuristic Function은 위에서 정의한 manhattan_dist function을 사용할 것.)
    """
    start_point = maze.startPoint()
    path = []
    ######################## Write Your Code Here ################################

    end_point = maze.circlePoints()[0]

    start_node = Node(None, start_point,0, 0)
    end_node = Node(None, end_point,0, 0)

    open_list = []
    closed_list = []

    open_list.append(start_node)

    while open_list:
        cur_node = open_list[0]
        cur_idx = 0

        for idx, item in enumerate(open_list):
            if item.fscore < cur_node.fscore:
                cur_node = item
                cur_idx = idx

        open_list.pop(cur_idx)
        closed_list.append(cur_node)

        if cur_node == end_node:
            temp = cur_node
            while temp:
                path.append(temp.item)
                temp = temp.parent
```

```
            path.reverse()
            break

        for next_y,next_x in maze.neighborPoints(cur_node.item[0],
cur_node.item[1]):
            new_node = Node(cur_node, (next_y,next_x), 0, 0)
            if new_node in closed_list:
                continue

            new_node.gscore = cur_node.gscore + 1
            new_node.heuri = manhattan_dist(new_node.item, end_node.item)
            new_node.fscore = new_node.gscore + new_node.heuri

            flag = 0
            for open_node in open_list:
                if open_node == new_node:
                    if new_node > open_node:
                        flag = 1
                        break
            if(flag != 1):
                open_list.append(new_node)

    result = maze.isValidPath(path)
    if(result != 'Valid'):
        print(result)

    return path


##########################################################################
###


1. small.txt

 ==========================================
 [ astar results ]
 (1) Path Length: 9
 (2) Search States: 14
 (3) Execute Time: 0.0002245903 seconds
 ==========================================
```



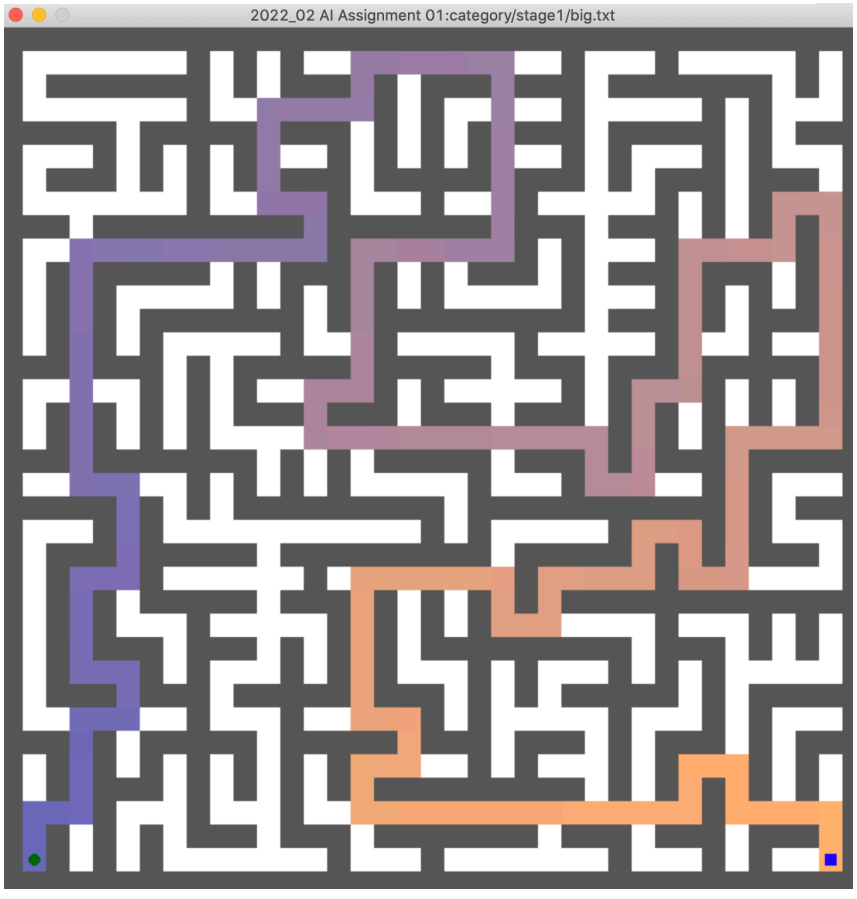
2. medium.txt

```
==========================================
[ astar results ]
(1) Path Length: 69
(2) Search States: 184
(3) Execute Time: 0.0058841705 seconds
==========================================
```



3. big.txt

```
==========================================
[ astar results ]
(1) Path Length: 211
(2) Search States: 549
(3) Execute Time: 0.0441658497 seconds
==========================================
```



구현 방법

Code

```
class Node(object):
    def __init__(self, parent, item, gscore = 0, heuri = 0):
        self.parent = parent
        self.item = item
        self.fscore = 0
        self.gscore = gscore
        self.heuri = heuri

    def __eq__(self, other):
        return self.item == other.item

    def __le__(self, other):
        return self.gscore + self.heuri <= other.gscore + other.heuri

    def __lt__(self, other):
        return self.gscore + self.heuri < other.gscore + other.heuri

    def __ge__(self, other):
        return self.gscore + self.heuri >= other.gscore + other.heuri

    def __gt__(self, other):
        return self.gscore + self.heuri > other.gscore + other.heuri
```

```
def manhattan_dist(p1, p2):
    return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])
```

Astar 알고리즘에선 새롭게 Node 클래스를 정의하여서 구현하였다. Node의 클래스에는 gscore과 heuristic의 합인 fscore 그리고 heuristic, gscore 값이 있고 현재 노드의 parent와 좌표(item)이 있다. 이 때 코딩을 원할하게 하기 위해서 ==, >, >=, <, <=를 정의해주었다.

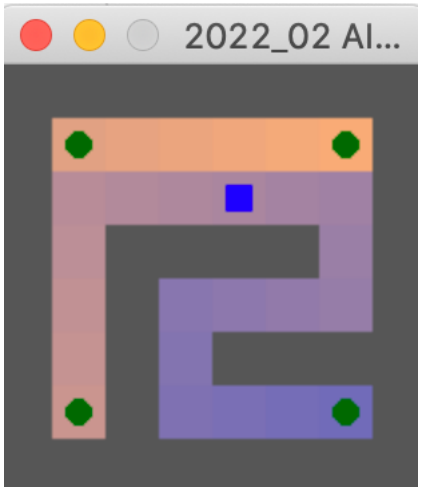
open_list와 closed_list를 통해서 open_list에는 방문하지 않은 node들을 그리고 closed_list에는 방문한 node들을 기록해둔다. 이 때 open_list에서 값을 pop할 때 값이 가장 작은 index를 구하여서 pop하고 closed_list에 기록하였다. 이후 neighborPoints 함수를 이용하여 주변 좌표들 중 closed_list에 없는 좌표의 fscore 값을 계산한 뒤 자신보다 작은 fscore 값을 가지고 있다면 이를 open_list에 삽입하고 그렇지 않다면 삽입하지 않는 과정을 반복한다. 최종적으로 end_node에 도달하게 되면 parent를 이용해 path를 역추적하여 path를 계산하였다.

Astar Four Circles Method

Code

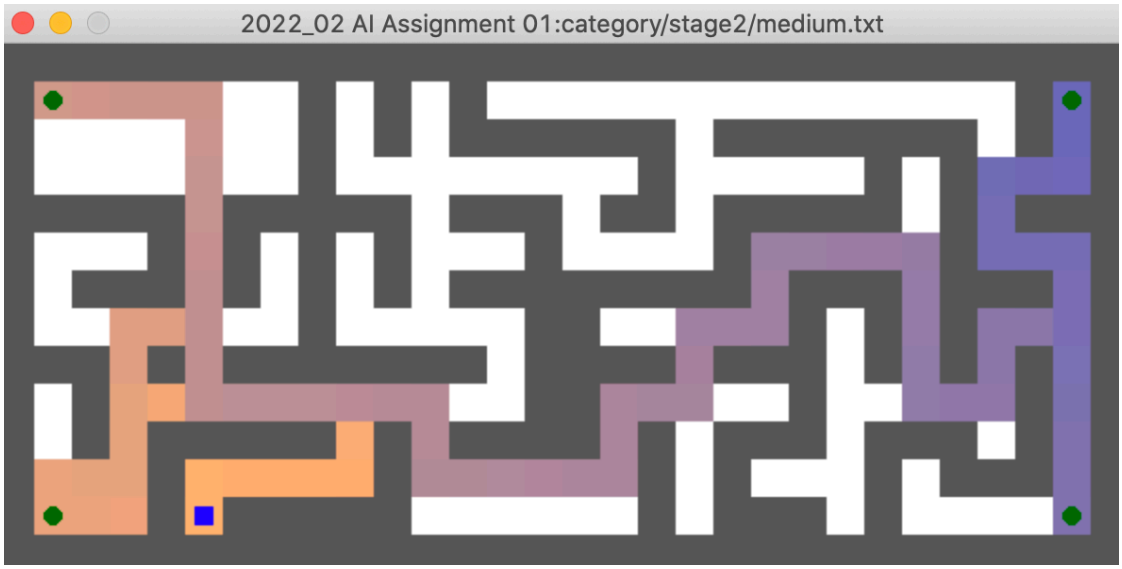
```
def stage2_heuristic(cur_point, end_nodes, visited):

    min_dist = float("inf")
    for i in range(4):
        if not visited[i]:
            min_dist = min(min_dist, manhattan_dist(cur_point,
end_nodes[i].item))

    return min_dist



def astar_four_circles(maze):
    """
    [Problem 04] 제시된 stage2 맵 세 가지를 A* Algorithm을 통해 최단경로를 return
하시오.
    (Heuristic Function은 직접 정의할것 )
    """
    start_point = maze.startPoint()
    path = []
    ######################## Write Your Code Here
################################

    end_points = maze.circlePoints()

    start_node = Node(None, start_point, 0, 0)
    end_nodes = [Node(None, end_point, 0, 0) for end_point in end_points]

    visited = [False, False, False, False]


    for i in range(4):
        open_list = []
        closed_list = []

        if not path:
            open_list.append(start_node)
        else:
            open_list.append(Node(None,path[-1],0,0))

        while open_list:
            cur_node = open_list[0]
            cur_idx = 0

            for idx, item in enumerate(open_list):
                if item.fscore < cur_node.fscore:
                    cur_node =  item
                    cur_idx = idx

            open_list.pop(cur_idx)
            closed_list.append(cur_node)
```

```
            if cur_node in end_nodes and not
visited[end_nodes.index(cur_node)]:
                visited[end_nodes.index(cur_node)] = True
                temp = cur_node
                path_temp = []
                while temp:
                    path_temp.append(temp.item)
                    temp = temp.parent

                path = path[0:len(path)-1] + path_temp[::-1]
                break

            for next_y, next_x in maze.neighborPoints(cur_node.item[0],
cur_node.item[1]):
                new_node = Node(cur_node, (next_y, next_x), 0, 0)
                if new_node in closed_list:
                    continue

                new_node.gscore = cur_node.gscore + 1
                new_node.heuri = stage2_heuristic(new_node.item,
end_nodes, visited)
                new_node.fscore = new_node.gscore + new_node.heuri

                flag = 0
                for open_node in open_list:
                    if new_node == open_node:
                        if new_node > open_node:
                            flag = 1
                            break
                if flag != 1:
                    open_list.append(new_node)


    result = maze.isValidPath(path)
    if result != 'Valid':
        print(result)

    return path


##########################################################################
###

1. small.txt


==========================================
[ astar_four_circles results ]
(1) Path Length: 33
(2) Search States: 49
(3) Execute Time: 0.0007789135 seconds
==========================================
```



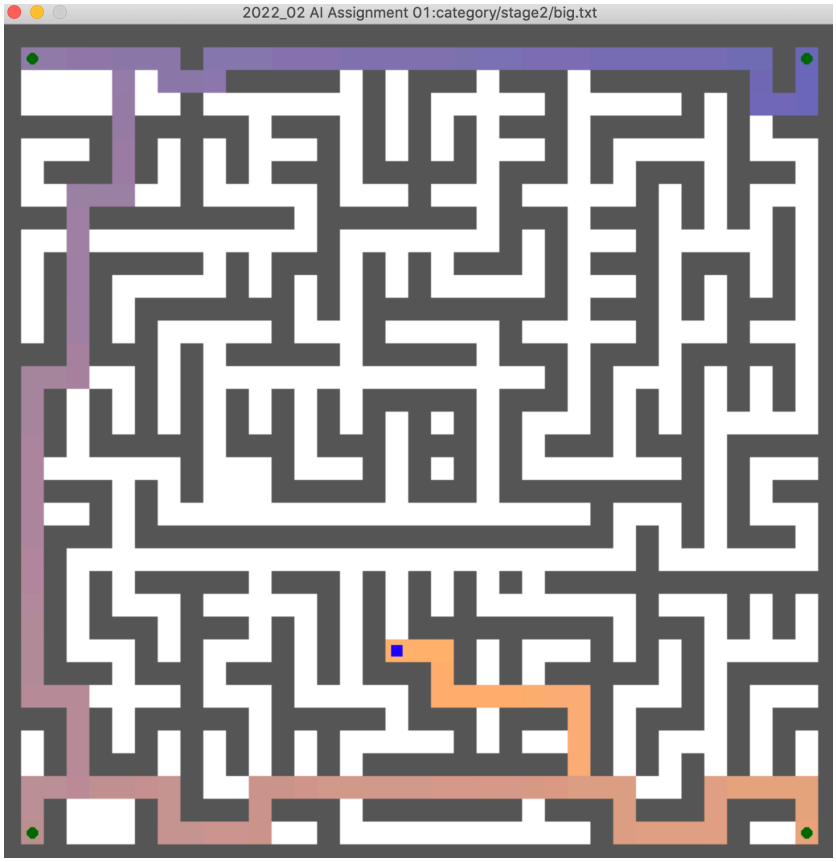
2. medium.txt

```
==========================================
[ astar_four_circles results ]
(1) Path Length: 107
(2) Search States: 767
(3) Execute Time: 0.0601582527 seconds
==========================================
```



3. big.txt

```
==========================================
[ astar_four_circles results ]
(1) Path Length: 163
(2) Search States: 485
(3) Execute Time: 0.0144009590 seconds
==========================================
```



구현 방법

사용한 자료구조는 앞에서 설명한 하나의 목표점만을 갖는 Astar 알고리즘과 동일하다. 다른 점은 heuristic function과 end_point들을 찾는 과정이다.

Heuristic function은 네 개의 목적지 중 Manhattan_dist 함수의 결과 중에서 가장 작은 값을 반환하도록 하였다.

```
def stage2_heuristic(cur_point, end_nodes, visited):

    min_dist = float("inf")
    for i in range(4):
        if not visited[i]:
            min_dist = min(min_dist, manhattan_dist(cur_point,
end_nodes[i].item))

    return min_dist
```

에서 현재 좌표에서 end_node들 중에서 이미 visited 되지 않은 좌표들 중 가장 Manhattan_dist가 작은 값을 return 하도록 하였다.
이후 목적지를 한 번 도착할 때마다 open_list와 closed_list를 초기화해줘서 다시 다른 목적지를 찾아가도록 하였다. path는 이렇게 구한 path들 중 목적지 path는 겹치기 때문에 이를 제외하고 각 for문에서 구한 path를 이어붙여서 구하였다.

Astar Many Circles Method

Code

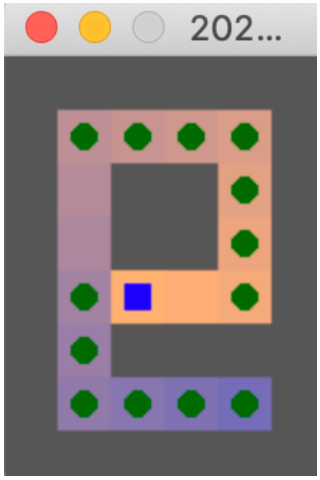
```
ef mst(start_point, remain_points):
    total_weight = 0

    graph = defaultdict(list)

    for y, x in remain_points:
        graph[start_point].append(((manhattan_dist(start_point,(y,x))),
(y,x)))
        graph[(y,x)].append(((manhattan_dist(start_point,(y,x))),
start_point))

    for y1, x1 in remain_points:
        for y2, x2 in remain_points:
            if y1 != y2 and x1 != x2:
                graph[(y1,x1)].append(((manhattan_dist((y1,x1),(y2,x2))),
(y2,x2)))
                graph[(y2,x2)].append(((manhattan_dist((y1,x1),(y2,x2))),
(y1,x1)))

    visited = set([start_point])
    candidate = graph[start_point]
    heapq.heapify(candidate)

    while(candidate):
        weight, u = heapq.heappop(candidate)

        if u in visited:
            continue
        visited.add(u)
        total_weight += weight

        for edge in graph[u]:
            if edge[1] not in visited:
                heapq.heappush(candidate, edge)

    return total_weight

def stage3_heuristic(cur_point, end_nodes, visited):
    remain_end_points = []

    for i in range(len(visited)):
        if not visited[i]:
            remain_end_points.append(end_nodes[i].item)

    return mst(cur_point, remain_end_points)


def astar_many_circles(maze):
    """
    [Problem 04] 제시된 stage3 맵 다섯 가지를 A* Algorithm을 통해 최단 경로를
return하시오.
```

```
    (Heuristic Function은 직접 정의 하고, minimum spanning tree를 활용하도록 한
다.)
    """
    start_point = maze.startPoint()
    path = []
    ######################### Write Your Code Here
################################

    end_points = maze.circlePoints()

    start_node = Node(None, start_point, 0, 0)
    end_nodes = [Node(None, end_point) for end_point in end_points]

    visited = [False for i in range(len(end_points))]

    for i in range(len(end_points)):
        open_list = []
        closed_list = []

        if not path:
            open_list.append(start_node)
        else:
            open_list.append(Node(None,path[-1],0,0))

        while open_list:
            cur_node = open_list[0]
            cur_idx = 0

            for idx, item in enumerate(open_list):
                if item.fscore < cur_node.fscore:
                    cur_node = item
                    cur_idx = idx

            open_list.pop(cur_idx)
            closed_list.append(cur_node)

            if cur_node in end_nodes and not
visited[end_nodes.index(cur_node)]:
                visited[end_nodes.index(cur_node)] = True

                temp = cur_node
                path_temp = []
                while temp:
                    path_temp.append(temp.item)
                    temp = temp.parent

                path = path[0:len(path)-1] + path_temp[::-1]
                break

            for next_y, next_x in maze.neighborPoints(cur_node.item[0],
cur_node.item[1]):
                new_node = Node(cur_node, (next_y,next_x), 0, 0)
                if new_node in closed_list:
                    continue
```

```
                new_node.gscore = cur_node.gscore + 1
                new_node.heuri = stage3_heuristic(new_node.item,
end_nodes, visited)
                new_node.fscore = new_node.gscore + new_node.heuri

                flag = 0
                for open_node in open_list:
                    if open_node == new_node:
                        if open_node < new_node:
                            flag = 1
                            break
                if flag != 1:
                    open_list.append(new_node)

    result = maze.isValidPath(path)
    if(result != 'Valid'):
        print(result)


    return path


#########################################################################
###
```

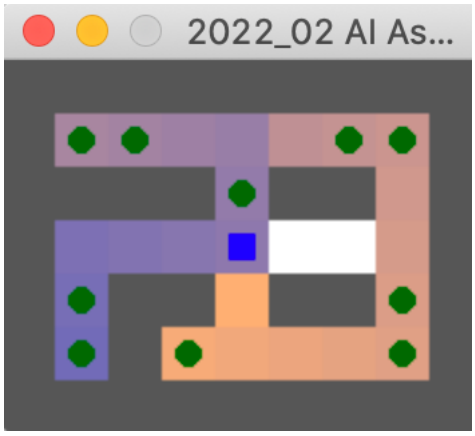
1. tiny.txt

```
==========================================
[ astar_many_circles results ]
(1) Path Length: 17
(2) Search States: 17
(3) Execute Time: 0.0026099682 seconds
==========================================
```



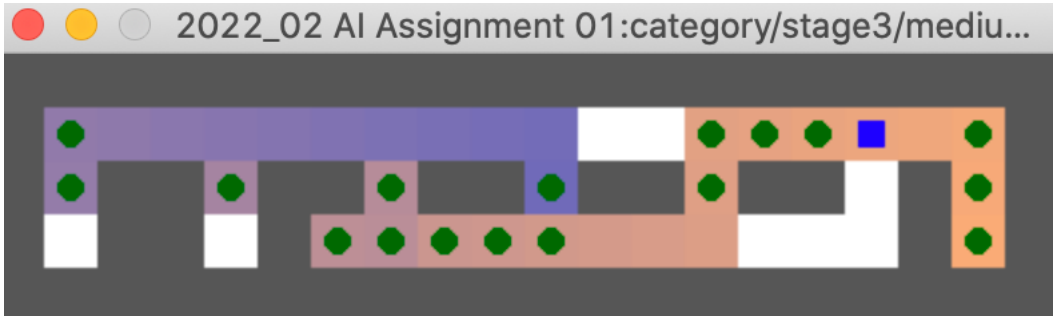
2. small.txt

```
==========================================
[ astar_many_circles results ]
(1) Path Length: 28
(2) Search States: 29
(3) Execute Time: 0.0027978420 seconds
==========================================
```



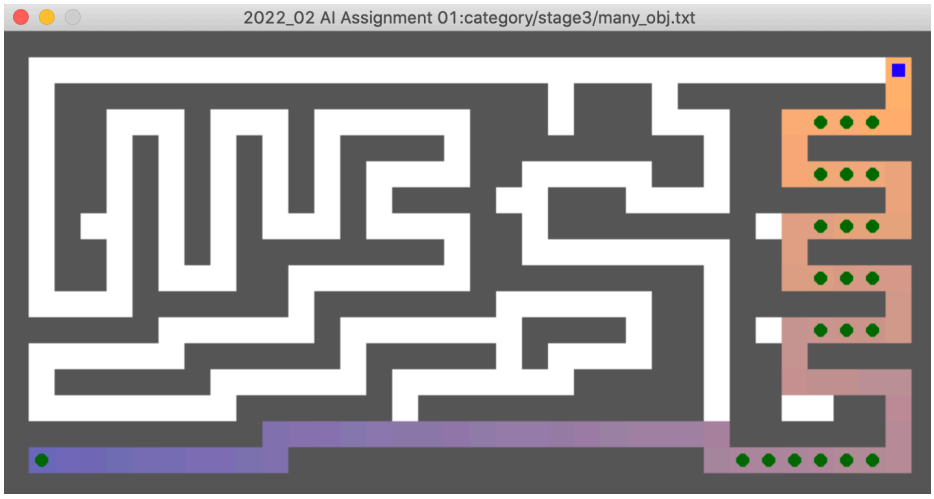
3. medium.txt

```
==========================================
[ astar_many_circles results ]
(1) Path Length: 46
(2) Search States: 50
(3) Execute Time: 0.0077998638 seconds
==========================================
```



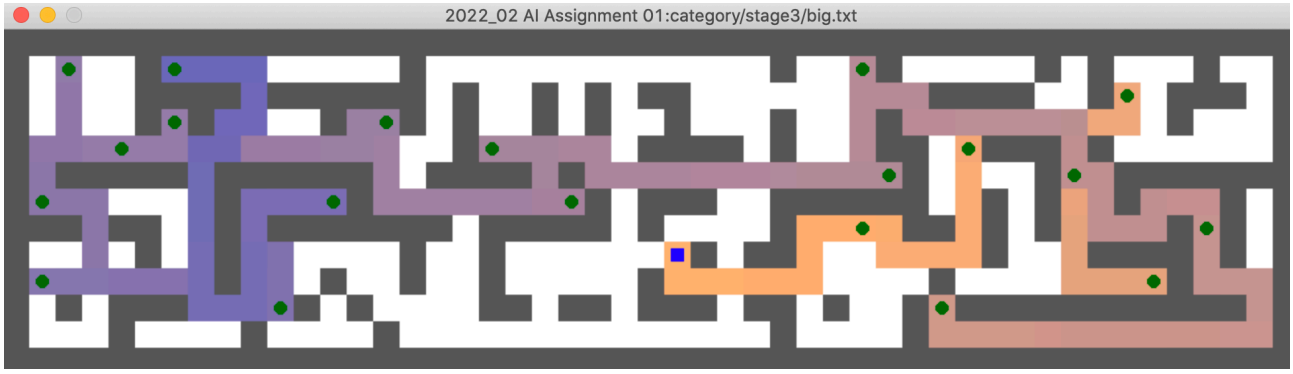
4. many_obj.txt

```
==========================================
[ astar_many_circles results ]
(1) Path Length: 75
(2) Search States: 87
(3) Execute Time: 0.0143349171 seconds
==========================================
```



5. big.txt

```
==========================================
[ astar_many_circles results ]
(1) Path Length: 223
(2) Search States: 761
(3) Execute Time: 0.2641711235 seconds
==========================================
```



구현 방법

MST를 구하는데 있어서 heapq라이브러리와 defaultdict라이브러리를 사용하였다. heapq를 이용하여 open_list 중 값이 가장 작은 node를 pop하도록 하였고 defaultdict는 MST에서 그래프를 구축할 때 사용하였다. 기존의 좌표로만 있던 정보들을 이용하여 graph로 만들어주었다. 이외 다른 자료구조들은 기존 astar 알고리즘과 동일하다.
MST를 구축할 때는 prim 알고리즘을 사용하였다. 매개변수로 받은 시작 노드를 기준으로 아직 도달하지 못한 end_point들을 이용하여 MST를 구축하엿다. 이 때 노드 사이의 cost는 기존의 manhattan_dist를 이용하였다.
Heuristic function은 위에서 구한 MST를 이용하여서 구하였다. 이전의 Astar four circles에서는 MST를 이용하지 않았지만 many circles에서는 아직 방문하지 않은 end_node들과 cur_node를 이용하여 MST를 이용한 total_weight을 반환받아 이것을 heuristic function으로 이용하였다.

앞에서는 4개의 end_point가 있었지만 여기서는 end_point가 정해지지 않았으므로 end_point의 개수만큼 for문을 이용하여 반복시켜주었으며 이외의 방식은 four circles와 동일하게 진행하였다.