Ahsanullah University of Science and Technology

Department of Computer Science & Engineering

Course No.	CSE 4108
Course Name	Artificial Intelligence Lab
Assignment No.	05

Submitted To:

Md. Siam Ansary	Tonmoy Hossain
Department of CSE, AUST	Department of CSE, AUST

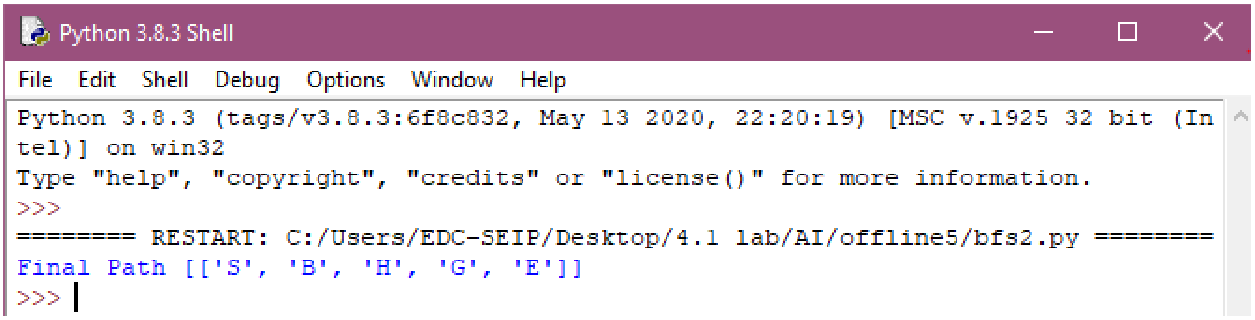
Submitted By:

Name	Tahiya Ahmed Chowdhury
ID No.	17.02.04.048
Session	Fall – 2020
Section	A (A2)
Date of Submission:	September 19,2021

Best first search code:

```
import copy

heuristic = {'S': 10, 'A': 9, 'B': 7, 'C': 8, 'D': 8, 'H': 6, 'L': 6, 'F': 6, 'G': 3, 'I': 4, 'J': 4, 'K': 3, 'E': 0}

traversal = {
    'S': ['A', 'B', 'C'],
    'A': ['B', 'D'],
    'B': {'H', 'D'},
    'C': ['L'],
    'D': ['F'],
    'H': ['F', 'G'],
    'L': ['I', 'J'],
    'G': ['E'],
    'I': ['K'],
    'J': ['K'],
    'K': ['E']
}


def Best_First_Search(start, end):
    path = []
    Q = []
    priorityQueue = [[[start], heuristic[start]]]
    visited = []

    while priorityQueue != []:
        Q = priorityQueue.copy()
        path.append(priorityQueue.pop(0))
        node = path[-1][0][-1]
        visited.append(node)

        if node == end:
            finalPath = copy.deepcopy(path[-1])
            print("Final Path", finalPath[0:1])
            return "Found"

        for neighbor in traversal[node]:
            if neighbor not in visited:
                newPath = copy.deepcopy(path[-1])
                newPath[0].append(neighbor)
                newPath[1] = heuristic[neighbor]
                priorityQueue.append(newPath)

        priorityQueue.sort(key=lambda x: x[1])
    print("Visited", visited)

Best_First_Search('S', 'E')
```

Best first search output:



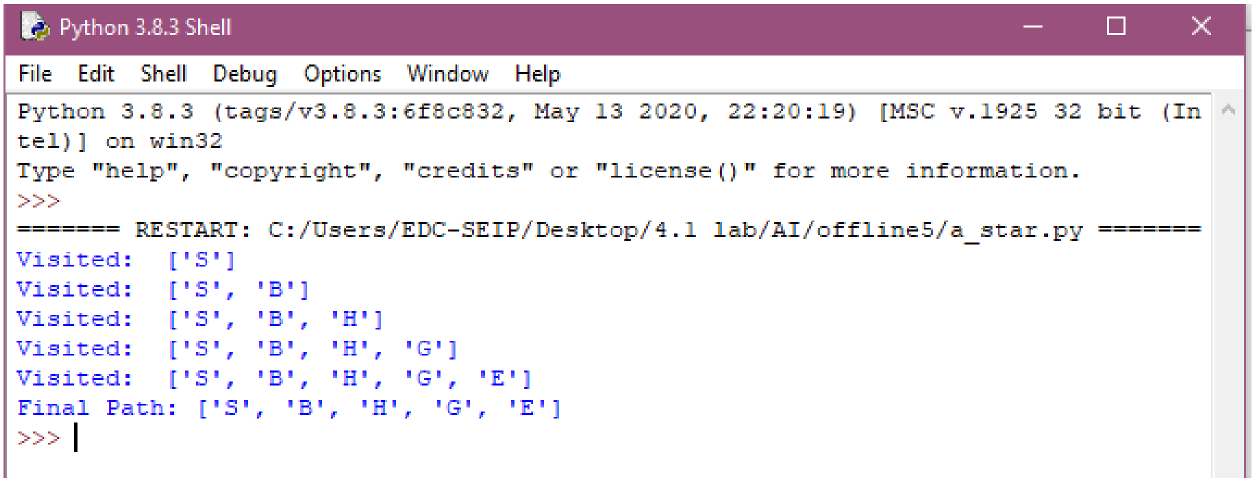
```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
======== RESTART: C:/Users/EDC-SEIP/Desktop/4.1 lab/AI/offline5/bfs2.py ========
Final Path [['S', 'B', 'H', 'G', 'E']]
>>> 
```

A star search code:

```
class Graph:
    def __init__(self, graph):
        self.graph = graph

    def get_neighbors(self, v):
        return self.graph[v]

    def h(self, n):
        Heuristic = {'S': 10, 'A': 9, 'B': 7, 'C': 8, 'D': 8, 'H': 6, 'L':
6, 'F': 6, 'G': 3, 'I': 4, 'J': 4, 'K': 3, 'E': 0}

        return Heuristic[n]

    def A_star_algorithm(self, start_node, end_node):
         open_list = set([start_node])
         closed_list = set([])
         li = []
         g = {}
         g[start_node] = 0
         parents = {}
         parents[start_node] = start_node

         while len(open_list) > 0:
             n = None

             for v in open_list:
                 if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                     n = v

             if n == None:
                 print('Path does not exist!')
                 return None

             if n == end_node:
                 li.append(n)

                 print("Visited: ", li)
                 reconst_path = []
```

```
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]

                reconst_path.append(start_node)

                reconst_path.reverse()

                print('Final Path: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight

                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)

            li.append(n)
            print("Visited: ", li)
            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')
        return None

graph = {
    'S': [('A', 7), ('B', 2), ('C', 3)],
    'A': [('B', 3), ('D', 4)],
    'B': [('D', 4), ('H', 1)],
    'C': [('L', 2)],
    'D': [('F', 5)],
    'H': [('F', 3), ('G', 2)],
    'L': [('I', 4), ('J', 4)],
    'F': [],
    'G': [('E', 2)],
    'I': [('K', 4)],
    'J': [('K', 4)],
    'K': [('E', 5)],
    'E': []
}
graph1 = Graph(graph)
graph1.A_star_algorithm('S', 'E')
```

A star search output:



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
======= RESTART: C:/Users/EDC-SEIP/Desktop/4.1 lab/AI/offline5/a_star.py =======
Visited:  ['S']
Visited:  ['S', 'B']
Visited:  ['S', 'B', 'H']
Visited:  ['S', 'B', 'H', 'G']
Visited:  ['S', 'B', 'H', 'G', 'E']
Final Path: ['S', 'B', 'H', 'G', 'E']
>>> 
```