



AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Course Name: Artificial Intelligence Lab

Course No: CSE 4108

Assignment No: 03

Name: Ahmad Subaktagin Jabir

ID: 16.02.04.061

Section: B

Group: B1

Date of Submission: August 25, 2020

Ques 1: Write a Python program that reads the file created as demonstrated into a dictionary taking 'name' as the key and a list consisting of 'dept' and 'cgpa' as the value for each line. Make changes in some 'cgpa' and then write back the whole file.

Ans:

```
d=dict()
with open("fl1.py") as f:
    for line in f:
        (name, dept, cgpa) = line.split('\t')
        cgpa=cgpa.rstrip("\n")
        d[str(name)]=[dept,cgpa]
f.close
print(d)
new_key=str(input("Enter the Name: "))
for key, value in d.items():
    if (new_key == key):
        new_cgpa= str(input("Enter the New CGPA: "))
        value[1] = new_cgpa
f1=open("fl1.py", "w")
for key,value in d.items():
    std=key+"\t"+value[0]+" \t"+value[1]
    print(std, end="\n", file=f1)
    print("\n")
f1.close
```

```
{'Ayon': ['CSE', '2.80'], 'Rakib': ['EEE', '3.25'], 'Shanto': ['Civil', '3.90']}
Enter the Name: Ayon
Enter the New CGPA: 2.70
```

Figure: Input

Ayon	<u>CSE</u>	2.70
Rakib	EEE	3.25
Shanto	Civil	3.90

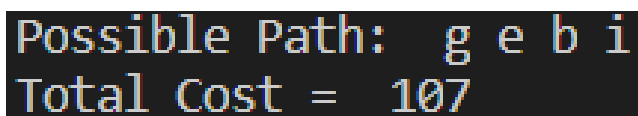
Figure: Output

Ques 2: Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First and A* search algorithms.

Ans:

Greedy Best First Search:

```
def execute_gbfs():
    source=8
    goal=6
    parent = [-1] * len(graph)
    visited = [False] * len(graph)
    push((heuristics_[source], source))
    visited[source]=True
    while(empty()==False):
        x=pop()
        if(x[1]==goal):
            break
        for adj in graph[x[1]]:
            v = adj[0]
            if(visited[v]==False):
                push((heuristics_[v],v))
                visited[v]=True
                parent[v]=x[1]
    temp = goal
    cost = 0
    print("Possible Path: ", end=' ')
    while (parent[temp]!=-1):
        x = parent[temp]
        v = temp
        print(chr(ord('a')+temp), end=' ')
        for adj in graph[x]:
            if(adj[0]==v):
                cost+=adj[1]
                break
        temp = parent[temp]
    print(chr(ord('a')+temp))
    print("Total Cost = ", cost)
execute_gbfs()
```



Possible Path: g e b i
Total Cost = 107

Figure: Output

A* Search:

```
def executeAStar(self, initial_node, goal_node):
    if not self.edges:
        print('Error: graph not contains edges!!')
    else:
        if initial_node in self.nodes and goal_node in self.nodes:
            if initial_node == goal_node:
                return 0

            queue = PriorityQueue()

            distance_vector, antecessors = {}, {}
            for node in self.nodes:
                distance_vector[node.getKey()] = None
                antecessors[node.getKey()] = None
            distance_vector[initial_node.getKey()] = 0

            g_cost, h_cost = 0, initial_node.getForwardCost()
            f_cost = g_cost + h_cost
            queue.insert((initial_node, g_cost, h_cost), f_cost)
            total_cost = None

            while True:
                current_node, g_cost, h_cost = queue.remove()
                successors = self.successors[current_node.getKey()]
                for successor in successors:
                    destination, weight = successor
                    new_g_cost = g_cost + weight
                    h_cost = destination.getForwardCost()
                    f_cost = new_g_cost + h_cost
                    queue.insert((destination, new_g_cost, h_cost),
                                f_cost)

                    if distance_vector[destination.getKey()]:
                        if distance_vector[destination.getKey()] >
new_g_cost:
                                distance_vector[destination.getKey()] =
new_g_cost
                                antecessors[destination.getKey()] =
current_node.getKey()
                    else:
                        distance_vector[destination.getKey()] =
new_g_cost
                        antecessors[destination.getKey()] =
current_node.getKey()

                    if destination.getKey() == goal_node.getKey():
                        if not total_cost:
                            total_cost = f_cost
                        elif f_cost < total_cost:
                            total_cost = f_cost

                if queue.isEmpty():
```

```
        curr_node = goal_node.getKey()
        while curr_node:
            self.path.append(curr_node)
            curr_node = antecessors[curr_node]
        self.path = self.path[::-1]
        return total_cost
    else:
        print('Error: the node(s) not exists in the graph!!')
```

```
Total Cost: 97.
Possible Path: i a d g
```

Figure: Output