Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

CSE4108

Artificial Intelligence

Lab Assignment 3

Submitted By:

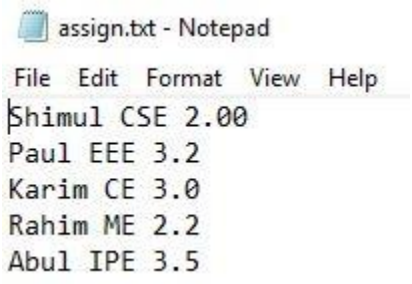
Name: Shimul Paul

ID: 16.02.04.014

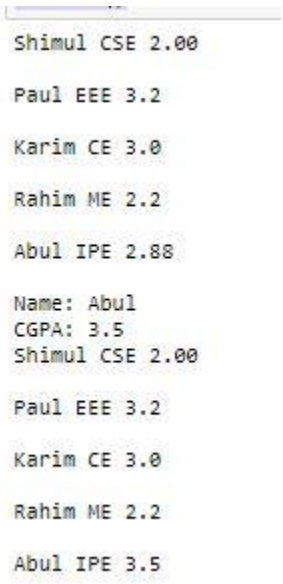
Section: A1

Assigment 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.

Input:



Output:



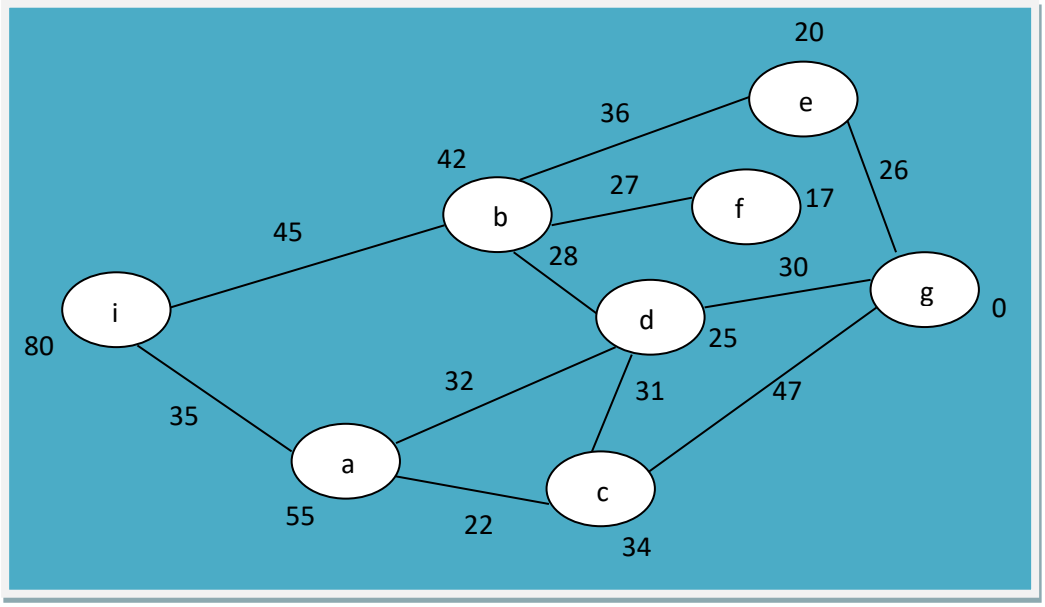
Python Code:

```
f=open('C:/Users/USER/Desktop/assign.txt','r')
dictionary = {}
for line in f:
    temp = line.strip().split()
    dictionary[temp[0]]=[temp[1],temp[2]]
    print(line)

x = input('Name: ')
for k,v in dictionary.items():
```

```
    if k==x:
        dictionary[k][1]=input('CGPA: ')
f.close()

f = open('C:/Users/USER/Desktop/assign.txt','w')
for k, v in dictionary.items():
    x = str(k) + ' ' + str(v[0]) +' ' + str(v[1]) + '\n'
    print(x)
    f.write(x)
f.close()
```

Assignment 2: Implement in generic ways (as multi-modular and interactive systems) A* search algorithms in Python.

Input:



graph = {'i': [['a', 35], ['b', 45]],

'a': [['d', 32], ['c', 22]],

'b': [['d', 28], ['f', 27],['e',36]],

'd': [['a',32],['b',28],['c', 31], ['g', 30]],

'c': [['a',22],['d',31],['g', 47]],

'e': [['b',36],['g',26]],

'f':[['b',27]]}

Output :

visited nodes: [['i', 80], ['b', 87], ['f', 89], ['a', 90], ['c', 91], ['d', 92], ['d', 98], ['e', 101], ['g', 104]]
optimal nodes sequence: ['i', 'b', 'e', 'g']

Python Code:

graph = {'i': [['a', 35], ['b', 45]],

'a': [['d', 32], ['c', 22]],

'b': [['d', 28], ['f', 27],['e',36]],

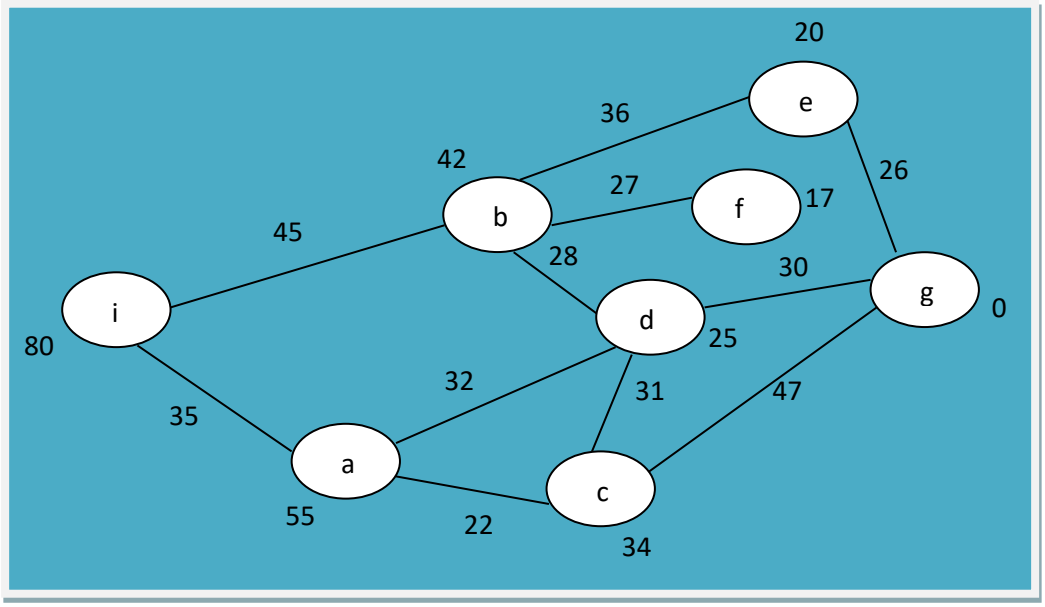
'd': [['a',32],['b',28],['c', 31], ['g', 30]],

```
        'c': [['a',22],['d',31],['g', 47]],
        'e': [['b',36],['g',26]],
        'f':[['b',27]]}
heuristic_values = {'i': 80, 'a': 55, 'b': 42, 'c': 34, 'd': 25, 'e': 20,'f':17, 'g': 0}
totalcost = {'i': 0}
def AStarSearch():
    closed_nodes = []
    opened_nodes = [['i', 80]]

    #finding the visited nodes
    while True:
        fn = [i[1] for i in opened_nodes]     # fn = f(n) = g(n) + h(n)
        chosen_index = fn.index(min(fn))
        #print(chosen_index)
        node = opened_nodes[chosen_index][0]  # current node
        #print(chosen_index,' ',node)
        closed_nodes.append(opened_nodes[chosen_index])
        del opened_nodes[chosen_index]
        if closed_nodes[-1][0] == 'g':
            break
        for item in graph[node]:
            if item[0] in [closed_item[0] for closed_item in closed_nodes]:
                continue
            totalcost.update({item[0]: totalcost[node] + item[1]})
            fn_node = cost[node] + heuristic_values[item[0]] + item[1]
            temp = [item[0], fn_node]
            opened_nodes.append(temp)
    #finding the optimal sequence
    trace_node = 'g'
    optimal_sequence = ['g']
```

```
    for i in range(len(closed_nodes)-2, -1, -1):
        check_node = closed_nodes[i][0]
        if trace_node in [children[0] for children in graph[check_node]]:
            children_costs = [temp[1] for temp in graph[check_node]]
            children_nodes = [temp[0] for temp in graph[check_node]]

            if totalcost[check_node] + children_costs[children_nodes.index(trace_node)] ==
totalcost[trace_node]:
                optimal_sequence.append(check_node)
                trace_node = check_node
    optimal_sequence.reverse()

    return closed_nodes, optimal_sequence
if __name__ == '__main__':
    visited_nodes, optimal_nodes = AStarSearch()
    print('visited nodes: ' + str(visited_nodes))
    print('optimal nodes sequence: ' + str(optimal_nodes))
```

Assignment 2: Implement in generic ways (as multi-modular and interactive systems) greedy best first search algorithms in Python.

Input:



graph = {'i': [['a', 35], ['b', 45]],

'a': [['d', 32], ['c', 22]],

'b': [['d', 28], ['f', 27],['e',36]],

'd': [['a',32],['b',28],['c', 31], ['g', 30]],

'c': [['a',22],['d',31],['g', 47]],

'e': [['b',36],['g',26]],

'f':[['b',27]]}

Output :

```
optimal nodes sequence: ['i', 'b', 'e', 'g']
```

Python code:

graph = {'i': [['a', 35], ['b', 45]],

'a': [['d', 32], ['c', 22]],

'b': [['d', 28], ['f', 27],['e',36]],

'd': [['a',32],['b',28],['c', 31], ['g', 30]],

```
        'c': [['a',22],['d',31],['g', 47]],
        'e': [['b',36],['g',26]],
        'f':[['b',27]]}
heuristic_values = {'i': 80, 'a': 55, 'b': 42, 'c': 34, 'd': 25, 'e': 20,'f':17, 'g': 0}

totalcost = {'i': 0}

def AStarSearch():
    closed_nodes = []
    opened_nodes = [['i', 80]]

    #finding the visited nodes
    while True:
        fn = [i[1] for i in opened_nodes]     # fn = f(n) = g(n) + h(n)
        chosen_index = fn.index(min(fn))
        #print(chosen_index)
        node = opened_nodes[chosen_index][0]  # current node
        #print(chosen_index,' ',node)
        closed_nodes.append(opened_nodes[chosen_index])
        del opened_nodes[chosen_index]
        if closed_nodes[-1][0] == 'g':
            break
        for item in graph[node]:
            if item[0] in [closed_item[0] for closed_item in closed_nodes]:
                continue
            totalcost.update({item[0]: totalcost[node] + item[1]})
            hn_node = heuristic[item[0]]
            temp = [item[0], hn_node]
            opened_nodes.append(temp)
    #finding the optimal sequence
```

```
    trace_node = 'g'
    optimal_sequence = ['g']
    for i in range(len(closed_nodes)-2, -1, -1):
        check_node = closed_nodes[i][0]
        if trace_node in [children[0] for children in graph[check_node]]:
            children_costs = [temp[1] for temp in graph[check_node]]
            children_nodes = [temp[0] for temp in graph[check_node]]

            if totalcost[check_node] + children_costs[children_nodes.index(trace_node)] ==
totalcost[trace_node]:
                optimal_sequence.append(check_node)
                trace_node = check_node
    optimal_sequence.reverse()

    return closed_nodes, optimal_sequence
if __name__ == '__main__':
    visited_nodes, optimal_nodes = AStarSearch()
    print('optimal nodes sequence: ' + str(optimal_nodes))
```