Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

Course No. : CSE4108
Course Title: Artificial Intelligence Lab

Date of Submission: 21/03/2021

Submitted By:

Lab Group: A1

Section: A

Name: Umme Habiba

ID: 170104004

Submitted To:

Dr. S.M.A Al Mamun

Tonmoy Hossain Dihan

Question: 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.

Answer:

Python Code:

```
# input to write file

fp = open("output.py", "w")

print("\n")

length = int(input('How many input: '))

for i in range(length):

    name = input("Enter Name:")

    dept = input("Enter Department:")

    cgpa = input("Enter CGPA:")

    std = name + "\t" + dept + "\t" + cgpa

    print(std, end="\n", file=fp)

    print("\n")

fp.close


# file to dictionary

fp = open("output.py", "r")

dictionary = {}

for i in fp:

    name, dept, cgpa = i.split("\t")

    cgpa = cgpa.replace('\n', '')
```

```
    dictionary[name] = [dept, cgpa]


print(dictionary)


# change cgpa and write back to file
fp = open("output.py", "w")
print("\n")
for key, value in dictionary.items():
    cg = input(f'Enter new cgpa for {key}: ')
    value[1] = cg
    std = key + "\t" + value[0] + "\t" + value[1]
    print(std, end="\n", file=fp)
    print("\n")
fp.close
# updated portion file to dictionary
fp = open("output.py", "r")
dictionary = {}
for i in fp:
    name, dept, cgpa = i.split("\t")
    cgpa = cgpa.replace('\n', '')
    dictionary[name] = [dept, cgpa]
print(dictionary)
```

Output:

```
= RESTART: P:\Artificial Intelligence Lab(CSE4108)\Session 3\170104004\dictionar
y.py


How many input: 3
Enter Name:Umme
Enter Department:CSE
Enter CGPA:3.75


Enter Name:Habiba
Enter Department:CSE
Enter CGPA:3.48


Enter Name:Prity
Enter Department:CSE
Enter CGPA:3.23


{'Umme': ['CSE', '3.75'], 'Habiba': ['CSE', '3.48'], 'Prity': ['CSE', '3.23']}


Enter new cgpa for Umme: 3.85


Enter new cgpa for Habiba: 3.75


Enter new cgpa for Prity: 3.48


{'Umme': ['CSE', '3.85'], 'Habiba': ['CSE', '3.75'], 'Prity': ['CSE', '3.48']}
>>>
```

Explanation:

Firstly we have create a file in write mood then insert input to write file ,then we have store file data into dictionary .After that, changing the CPGA for each and write back to file and then write back to dictionary. Dictionary is used to store the values .Here name is used as a dictionary key and CGPA & Department as dictionary values.

Question: Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First algorithms in Python.

Answer:

Python Code:

```
class NodePriority(object):
    def __init__(self, node_name, h_value):
        self.h_value = h_value
        self.node_name = node_name
        #return
    def __lt__(self, other):
        return self.h_value < other.h_value
import queue as Q
import moduleforGBFS as GBFSQ
Neighbours = [('i', 'a', 35), ('a', 'i', 35), ('i', 'b', 45), ('b', 'i', 45), ('a', 'c', 22), ('c', 'a', 22),
            ('a', 'd', 32), ('d', 'a', 32),
            ('b', 'd', 28), ('d', 'b', 28), ('b', 'e', 36), ('e', 'b', 36), ('b', 'f', 27), ('f', 'b', 27),
            ('c', 'd', 31), ('d', 'c', 31),
            ('c', 'g', 47), ('g', 'c', 47), ('d', 'g', 30), ('g', 'd', 30), ('e', 'g', 26), ('g', 'e', 26)]
heuristicFunc = [('i', 80), ('a', 55), ('b', 42), ('c', 34), ('d', 25), ('e', 20), ('f', 17), ('g', 0)]
priorityQ = Q.PriorityQueue()
treenode = []
```

```
path = []
startNode = str(input('Enter starting node:'))
goalNode = str(input('Enter goal node:'))
treenode.append((startNode, 'root'))
isvisited = {}
neighborNode = False
for i in range(len(heuristicFunc)):
    isvisited[heuristicFunc[i][0]] = False
for i in range(len(heuristicFunc)):
    if heuristicFunc[i][0] == startNode:
        priorityQ.put(GBFSQ.NodePriority(startNode,heuristicFunc[i][1]))
while not (priorityQ.empty()):
    v = priorityQ.get()
    node = v.node_name
    isvisited[node] = True
    if node == goalNode:
        path.append(node)
        break
    else:
        for i in range(len(Neighbours)):
            if Neighbours[i][0] == node:
                nextValue = Neighbours[i][1]
```

```
            if isvisited[nextValue] == False:
                neighborNode = True
                treenode.append((node, nextValue))
                for j in range(len(heuristicFunc)):
                    if heuristicFunc[j][0] ==nextValue:
                        priorityQ.put(GBFSQ.NodePriority(nextValue,heuristicFunc[j][1]))
            else:
                neighborNode = False
        if neighborNode == True:
            path.append(node)
print('The path is:', end=' ')
for x in path:
    print(x,end=' ')
```

Output:

```
= RESTART: P:\Artificial Intelligence Lab(CSE4108)\Session 3\170104004\GBFS.py =
Enter starting node:i
Enter goal node:g
The path is: i b e g
>>>
```

Explanation: Firstly we initialize the given graph neighbors and heuristic functions .Then taking input as starting node and goal node. Declare a priority queue which will contain nodes in ascending order. Then based on heuristic function select nodes and its neighbor and then select the neighbor which has least heuristics value. After that select them visited and print the node and go for next nodes, will continue the procedure until the priority queue become empty.

Question: Implement in generic ways (as multi-modular and interactive systems) the A* search algorithm in Python.

Answer:

Python Code:

```
class NodePriority(object):
    def __init__(self, name, index, parent,h_value):
        self.h_value = h_value
        self.name = name
        self.index = index
        self.parent = parent
        return
    def __lt__(self, other):
        return self.h_value < other.h_value


import queue as Q
import moduleforAstar as astar


Neighbours = [('i', 'a', 35), ('a', 'i', 35), ('i', 'b', 45), ('b', 'i', 45), ('a', 'c', 22), ('c', 'a',
22),
            ('a', 'd', 32), ('d', 'a', 32),
            ('b', 'd', 28), ('d', 'b', 28), ('b', 'e', 36), ('e', 'b', 36), ('b', 'f', 27), ('f', 'b', 27),
            ('c', 'd', 31), ('d', 'c', 31),
```

```
        ('c', 'g', 47), ('g', 'c', 47), ('d', 'g', 30), ('g', 'd', 30), ('e', 'g', 26), ('g', 'e', 26)]
heuristicFunc = [('i', 80), ('a', 55), ('b', 42), ('c', 34), ('d', 25), ('e', 20), ('f', 17), ('g', 0)]
priority_queue = Q.PriorityQueue()
t_nodes = []
path = []
# main
s = str(input('Enter starting node:'))
g = str(input('Enter goal node:'))
visited = {}
tn_index = {}
parrent_node = {}
h_value = 0
next_node = False
for i in range(len(heuristicFunc)):
   visited[heuristicFunc[i][0]] = False
for i in range(len(heuristicFunc)):
   if heuristicFunc[i][0] ==s:
      h_value = heuristicFunc[i][1]
      priority_queue.put(astar.NodePriority(s, 0, 'root', h_value)) #
t_nodes.append((s, 0, 'root', h_value))
index = 0
```

```
tn_index[s] = 0
parrent_node[0] = s
parrent_node['root'] = 'root'
n_h_value = 0
while not (priority_queue.empty()):
    v = priority_queue.get()
    node = v.name
    visited[node] = True
    if (node == g):
        path.append(node)
        break
    else:
        i = 0
        for i in range(len(Neighbours)):
            if Neighbours[i][0] == node:
                next_v = Neighbours[i][1]
                index = index + 1
                tn_index[next_v] = index
                parrent_node[index] = node
                cost = Neighbours[i][2]
                if visited[next_v] == False:
                    next_node = True
```

```
                t_nodes.append((node, next_v))
                for j in range(len(heuristicFunc)):
                    if heuristicFunc[j][0] ==next_v:
                        priority_queue.put(astar.
NodePriority(next_v,tn_index[next_v],tn_index[node],heuristicFunc[j][1]+cost))
            else:
                next_node = False
        if next_node == True:
            path.append(node)
print('The path is:')
for i in path:
    print(i,end=' ')
```

Output:

```
= RESTART: P:/Artificial Intelligence Lab(CSE4108)/Session 3/170104004/Astar.py
Enter starting node:i
Enter goal node:g
The path is:
i b d g
>>>
```

Explanation: Basically it is similar to Greedy BFS, just it has some extra features. Here we store node index and parent index into the priority queue. For evaluation it's same as Greedy BFS just extra addition of cost being added. Here optimal solution is considered among all the possible solution.