*Ahsanullah University of Science and Technology*

**Department of Computer Science and Engineering**

**CSE4108: Artificial Intelligence Lab**

**Name:** Rakib Hossain Rifat

**ID:** 16.02.04.099

**Group:** B2

Assignment **# 03**

**Date of Submission:** 25/08/2020

**QUESTION-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.**

**ANSWER:**

- File opened in write mode and Name, DEPT & CGPA taken as user input.
- File opened in read mode and Name, DEPT & CGPA are read from the file split by tab.
- Dept and CGPA are stored in a list.
- A dictionary is used to store the values.
- Name is taken as the key and list is the value.
- All CGPA are updated and stored in the file.

Code:

```
#taking input to write in the file
f1=open("test.py", "w")
print("\n")
for i in range(3):
        name=str(input("Enter the name:"))
        dept=str(input("Enter the department:"))
        cgpa=str(input("Enter the cgpa:"))
        std=name+"\t"+dept+"\t"+cgpa
        print(std, end="\n", file=f1)
        print("\n")
f1.close

#reading from a file to dictionary
f1 = open("test.py", "r")
dict = {}
for l in f1:
    list = []
    name, dept, cgpa =l.split("\t")
    cgpa = cgpa.replace('\n','')
    list.append(dept)
    list.append(cgpa)
    (key , val) = name , list
    dict[key] = val

print(dict)

#changing  cgpa and writing back to file
f1=open("test.py", "w")
print("\n")
```

```
for key,value in dict.items():

    value[1] = 4.0
    name = str(key)
    dept = str(value[0])
    cgpa = str(value[1])
    std = name + "\t" + dept + "\t" + cgpa
    print(std, end="\n", file=f1)
    print("\n")

f1.close

# after change reading from a file to dictionary
f1 = open("test.py", "r")
dict = {}
for l in f1:
    list = []
    name, dept, cgpa =l.split("\t")
    cgpa = cgpa.replace('\n','')
    list.append(dept)
    list.append(cgpa)
    (key , val) = name , list
    dict[key] = val

print(dict)
```

Output:

```
(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>python CGPA.py


Enter the name:Rakib
Enter the department:CSE
Enter the cgpa:2.0


Enter the name:Hossain
Enter the department:EEE
Enter the cgpa:2.5


Enter the name:Rifat
Enter the department:CSE
Enter the cgpa:3.0


{'Rakib': ['CSE', '2.0'], 'Hossain': ['EEE', '2.5'], 'Rifat': ['CSE', '3.0']}









{'Rakib': ['CSE', '4.0'], 'Hossain': ['EEE', '4.0'], 'Rifat': ['CSE', '4.0']}

(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>
```

## QUESTION-2:

**Implement in generic ways (as multi-modular and interactive systems) the Greedy best-first algorithms in Python**.

## ANSWER:

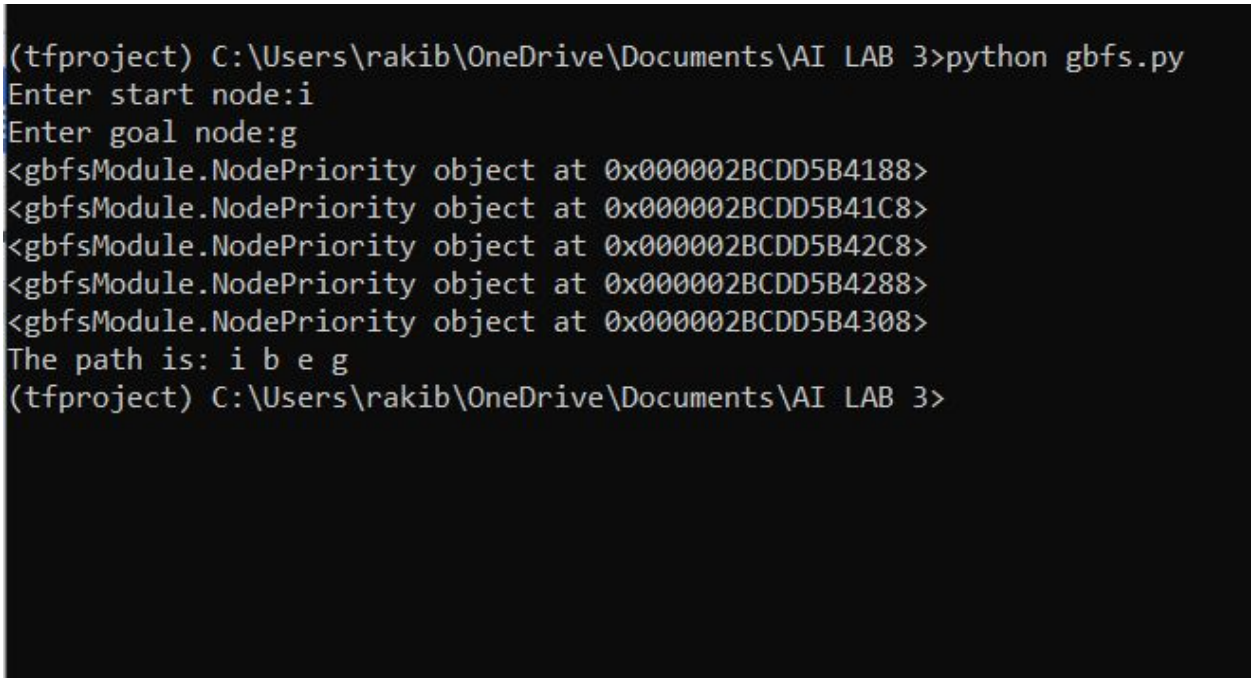
- For the given graph neighbor nodes name path value and heuristic functions are initialized.
- User input is taken for source(s) and goal nodes(g).
- A Priority Queue (PQ), which contains nodes in ascending order(modified in module) of h-values, is maintained.
- A Possible Path (Path) is maintained that will be output.
- Based on the heuristic function a node is selected and is visited to find its neighbors.
- The process begins with placing the source node in empty PQ and initiating a tree with source as root and ends when the goal node is placed in the PQ and selected for visit.
- The first node from the PQ is selected repeatedly and deleted from PQ. Each time the tree, the PQ, and the Path are updated.
- The node in the tree is marked visited and its neighbors from the graph are added to the tree as its children.

## 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 gbfsModule as gbfs_q
neighbour = [('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)]

heu_fn = [('i', 80), ('a', 55), ('b', 42), ('c', 34), ('d', 25), ('e', 20), ('f', 17), ('g', 0)]


pq = q.PriorityQueue()
t_nodes = []
path = []


s = str(input('Enter start node:'))
g = str(input('Enter goal node:'))
t_nodes.append((s, 'root'))
visited = {}
next_node = False
for i in range(len(heu_fn)):
    visited[heu_fn[i][0]] = False


for i in range(len(heu_fn)):
    if heu_fn[i][0] ==s:
        pq.put(gbfs_q.NodePriority(s,heu_fn[i][1]))

while not (pq.empty()):
    v = pq.get()
    print(v)
    node = v.node_name
    visited[node] = True
    if (node == g):
        path.append(node)
        break
    else:
        for i in range(len(neighbour)):
            if (neighbour[i][0] == node):
                next_v = neighbour[i][1]
                if (visited[next_v] == False):
                    next_node = True
                    t_nodes.append((node, next_v))
                    for j in range(len(heu_fn)):
                        if heu_fn[j][0] ==next_v:
                            pq.put(gbfs_q. NodePriority(next_v,heu_fn[j][1]))
                else:
                    next_node = False
        if(next_node == True):
            path.append(node)

print('The path is:',end=' ')
for x in path:
    print(x,end=' ')
```

Output:


(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>python gbfs.py
Enter start node:i
Enter goal node:g
<gbfsModule.NodePriority object at 0x000002BCDD5B4188>
<gbfsModule.NodePriority object at 0x000002BCDD5B41C8>
<gbfsModule.NodePriority object at 0x000002BCDD5B42C8>
<gbfsModule.NodePriority object at 0x000002BCDD5B4288>
<gbfsModule.NodePriority object at 0x000002BCDD5B4308>
The path is: i b e g
(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>

**QUESTION-3:**

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

**ANSWER:**

- It is similar to GBFS. However, it has some other features.
- The index for all nodes is maintained.All Node index and parent index are also added to PQ.
- The evaluation function is f(n) = g(n) + h(n), where g(n) = an actual path cost from initial node to node n, and h(n) = estimated cost of the cheapest path from n to the goal.
- If a path is there then the process generates all the neighbors repeatedly and adds in PQ.
- The optimal solution is considered and sub-optimal ones are avoided.

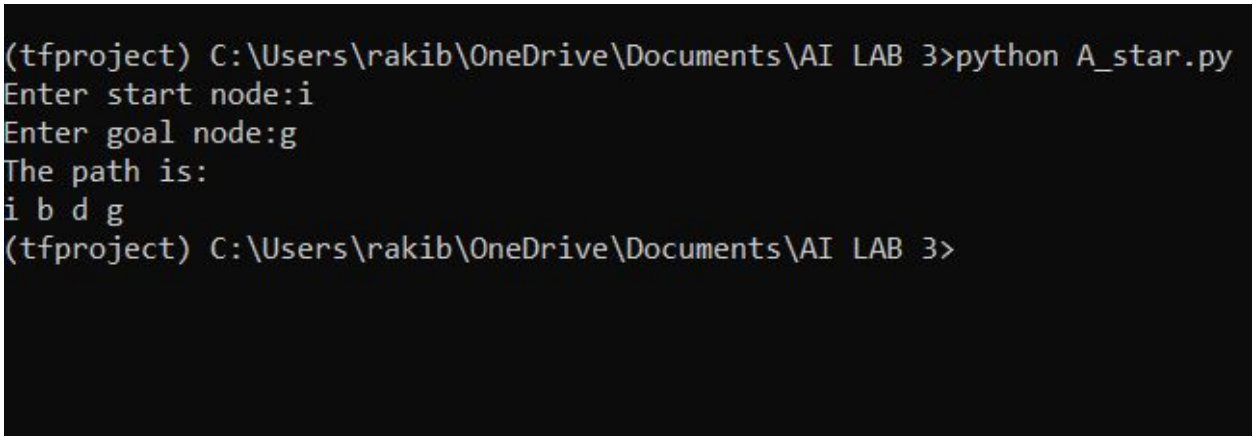
Code:

```
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 sModule as pri_queue

neighbour = [('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)]

heu_fn = [('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 start node:'))
g = str(input('Enter goal node:'))


visited = {}
tn_index = {}
parrent_node = {}
h_value = 0
next_node = False
```

```
for i in range(len(heu_fn)):
   visited[heu_fn[i][0]] = False
for i in range(len(heu_fn)):
   if heu_fn[i][0] ==s:
      h_value = heu_fn[i][1]
      priority_queue.put(pri_queue.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(neighbour)):
         if (neighbour[i][0] == node):
            next_v = neighbour[i][1]
            index = index + 1
            tn_index[next_v] = index
            parrent_node[index] = node
            cost = neighbour[i][2]
            if (visited[next_v] == False):
               next_node = True
               t_nodes.append((node, next_v))
               for j in range(len(heu_fn)):
                  if heu_fn[j][0] ==next_v:
                     priority_queue.put(pri_queue. NodePriority(next_v,tn_index[next_v],tn_index[node],heu_fn[j][1]+cost))

            else:
               next_node = False
      if(next_node == True):
         path.append(node)

print('The path is:')
for x in path:
   print(x,end=' ')
```

Output:


(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>python A_star.py
Enter start node:i
Enter goal node:g
The path is:
i b d g
(tfproject) C:\Users\rakib\OneDrive\Documents\AI LAB 3>