Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

CSE4108
Artificial Intelligence Lab

Class Assignment: 03

Submitted By:

Name : Devopriya Tirtho
Id :16.02.04.033

Date of Submission: **25th August, 2020**

Lab Exercise 1: Explore thoroughly the supplementary material provided for this session.

A. We have analyzed Major components of Prolog code for GBF and A* search.
B. We have run a python code which can write in and read from a file.

Lab Exercise 2: Run and analyze the codes demonstrated in this session.

We have run and analyzed the codes demonstrated in this session.

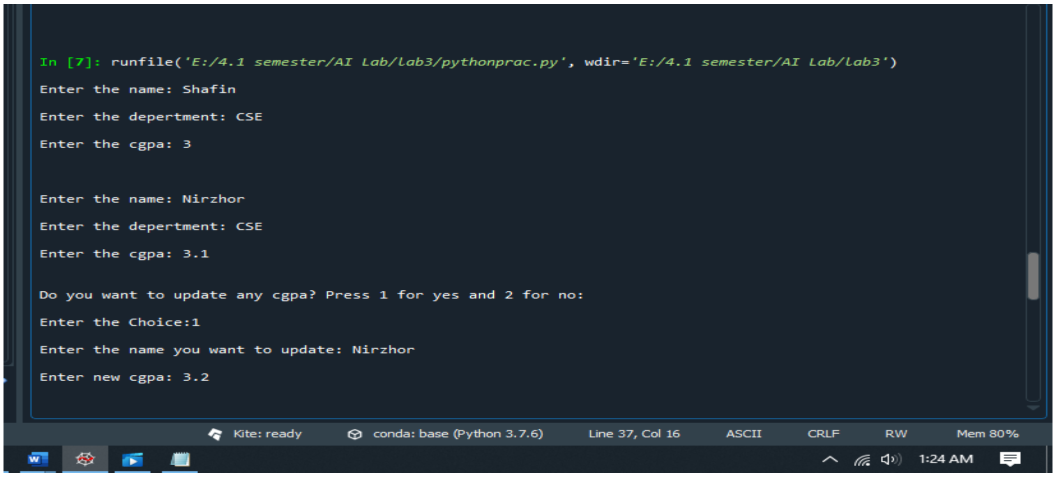
Lab Exercise 3: Write a Python program that reads the file created as demonstrated into a dictionary taking ‘name’ as the key and a list consisting of ‘dept’ and ‘cgpa’ as the value for each line. Make changes in some ‘cgpa’ and then write back the whole file.

Python Code:

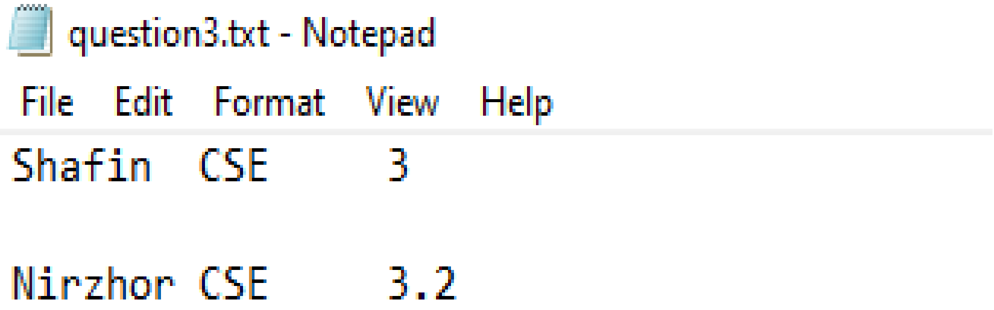
```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 24 19:03:28 2020

@author: Hp
"""
class_info = dict()
def update(class_info):
    name = str(input('Enter the name you want to update: '))
    cgpa = str(input('Enter new cgpa: '))
    store = class_info[name]
    store = str(store)
    sl = store.split("\t")
    sl[1] = cgpa
    up = sl[0] +"\t"+ sl[1]
    class_info[name] = up

f1 = open("assm.txt", "w")
for i in range(3):
    name = str(input("Enter the name:"))
    dept = str(input("Enter the depertment: "))
    cgpa = str(input("Enter the cgpa: "))
    std = name + "\t" + dept + "\t" + cgpa
    print(std, end="\n", file=f1)
    print("\n")
f1.close


f1 = open("assm.txt","r")
for i in f1:
    name, dept, cgpa = i.split("\t")
    class_info[name] = dept + "\t" + cgpa
```

```
f1.close



print("Do you want to update any cgpa? Press 1 for yes and 2 for no: ")

choice=(str(input("Enter the Choice:")))

if choice=='1':
   update(class_info)
   f1 = open("assm.txt", "w")
   for key,values in class_info.items():
      name = key
      t = values
      t = str(t)
      t = t.split("\t")
      std = name +"\t" +t[0] +"\t" + t[1]
      print(std, end="\n", file=f1)
      print("\n")
   f1.close

elif choice=='2':
   f1=open("assm.txt", "r")
   for l in f1:
      name, dept, cgpa =l.split("\t")
      print(name, dept, float(cgpa), end="\n")
   f1.close
```

Sample Output:
When the choice is 1:

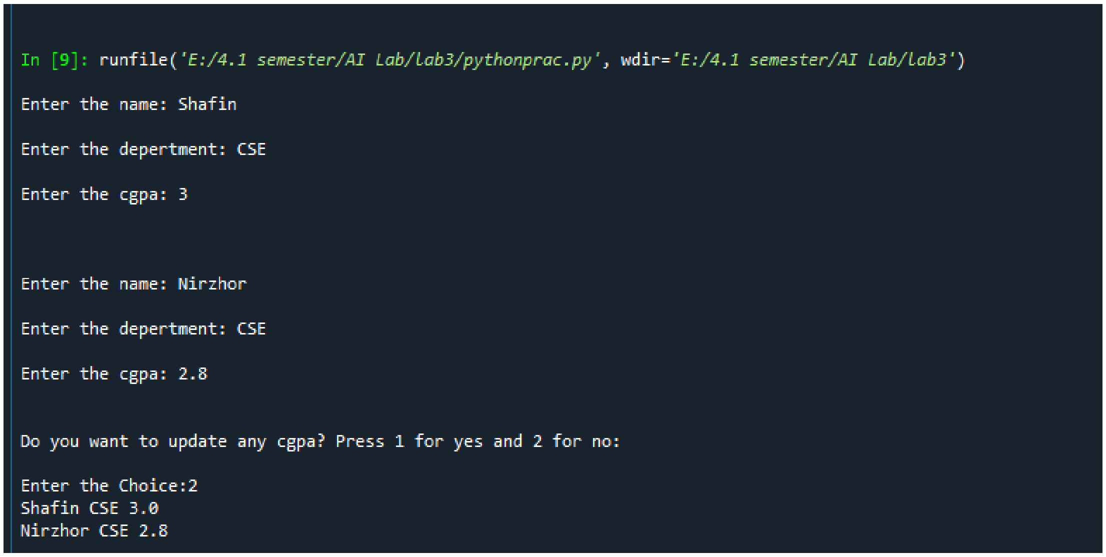


```
In [7]: runfile('E:/4.1 semester/AI Lab/lab3/pythonprac.py', wdir='E:/4.1 semester/AI Lab/lab3')

Enter the name: Shafin

Enter the depertment: CSE

Enter the cgpa: 3


Enter the name: Nirzhor

Enter the depertment: CSE

Enter the cgpa: 3.1

Do you want to update any cgpa? Press 1 for yes and 2 for no:

Enter the Choice:1

Enter the name you want to update: Nirzhor

Enter new cgpa: 3.2
```



```
Shafin  CSE     3

Nirzhor CSE     3.2
```

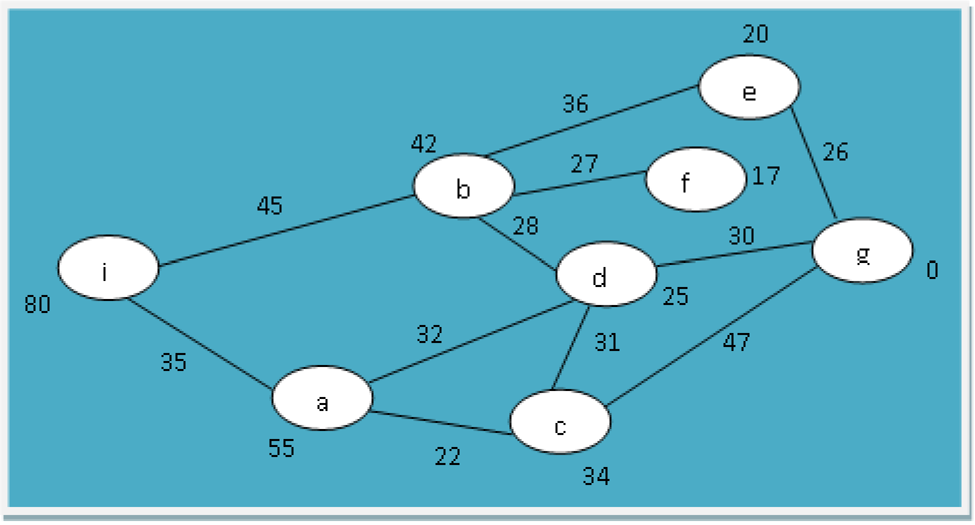
When the choice is 2:



```
In [9]: runfile('E:/4.1 semester/AI Lab/lab3/pythonprac.py', wdir='E:/4.1 semester/AI Lab/lab3')

Enter the name: Shafin

Enter the depertment: CSE

Enter the cgpa: 3


Enter the name: Nirzhor

Enter the depertment: CSE

Enter the cgpa: 2.8


Do you want to update any cgpa? Press 1 for yes and 2 for no:

Enter the Choice:2
Shafin CSE 3.0
Nirzhor CSE 2.8
```

Lab Exercise 4(1): Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First in Python.

The given graph:



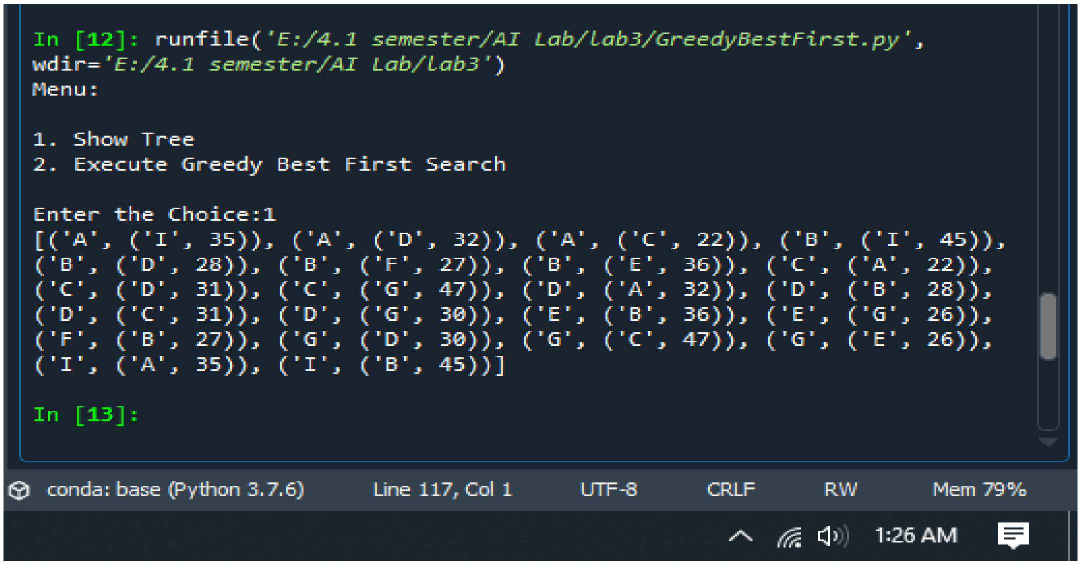
Python Code:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 24 19:03:28 2020

@author: Hp
"""
class Graph:

   def __init__(self, adjacency_list):
      self.adjacency_list = adjacency_list

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

   def hr(self, n):
      H = {
         'A': 55,
         'B': 42,
         'C': 34,
         'D': 25,
         'E': 20,
         'F': 17,
         'G': 0,
         'I': 80
      }
```

```
        return H[n]

    def a_star_algorithm(self, start_node, stop_node):

        open_list = set([start_node])
        closed_list = set([])

        #in g , storing the values of distances from start node to all other nodes
        g = {}

        g[start_node] = 0

        parents = {}
        parents[start_node] = start_node

        while len(open_list) > 0:
            n = None

            # find a node with the lowest value of f() - evaluation function
            for v in open_list:
                if n == None or self.hr(v) < self.hr(n):
                    n = v;

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

            if n == stop_node:
                reconst_path = []

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

                reconst_path.append(start_node)

                reconst_path.reverse()

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

            # for all neighbors of the current node do
            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
                    #print(g[m])

                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)

            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')

        return None





# GRAPH MENTIONED IN THE LAB 3 SESSION

adjacency_list = {
    'A': [('I', 35), ('D', 32), ('C', 22)],
    'B': [('I', 45), ('D', 28), ('F', 27),('E',36)],
    'C': [('A', 22), ('D', 31), ('G', 47)],
    'D': [('A', 32), ('B', 28), ('C', 31),('G',30)],
    'E': [('B', 36), ('G', 26)],
    'F': [('B', 27)],
    'G': [('D', 30), ('C', 47), ('E', 26)],
    'I': [('A', 35), ('B', 45)]

}


print("Menu:\n")
print("1. Show Tree")
print("2. Execute Greedy Best First Search")

choice=(str(input("Enter the Choice:")))

if choice=='1':

    def generate_edges(adjacency_list):
        edges = []
```

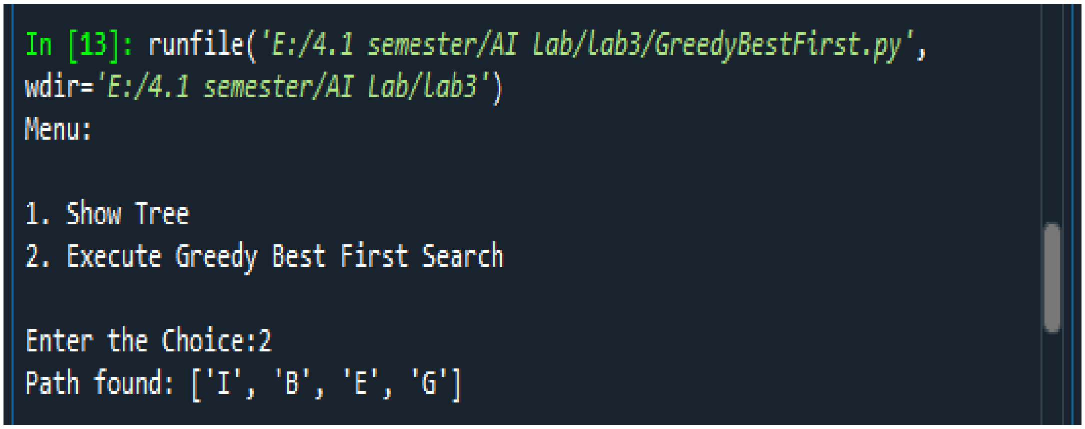
```
                  for node in adjacency_list:
                      for neighbour in adjacency_list[node]:
                          edges.append((node, neighbour))
                  return edges
              print(generate_edges(adjacency_list))


          elif choice=='2':
              graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('I', 'G')
```

Sample Output:
When the choice is 1:



```
In [12]: runfile('E:/4.1 semester/AI Lab/lab3/GreedyBestFirst.py',
wdir='E:/4.1 semester/AI Lab/lab3')
Menu:

1. Show Tree
2. Execute Greedy Best First Search

Enter the Choice:1
[('A', ('I', 35)), ('A', ('D', 32)), ('A', ('C', 22)), ('B', ('I', 45)),
('B', ('D', 28)), ('B', ('F', 27)), ('B', ('E', 36)), ('C', ('A', 22)),
('C', ('D', 31)), ('C', ('G', 47)), ('D', ('A', 32)), ('D', ('B', 28)),
('D', ('C', 31)), ('D', ('G', 30)), ('E', ('B', 36)), ('E', ('G', 26)),
('F', ('B', 27)), ('G', ('D', 30)), ('G', ('C', 47)), ('G', ('E', 26)),
('I', ('A', 35)), ('I', ('B', 45))]

In [13]:
```

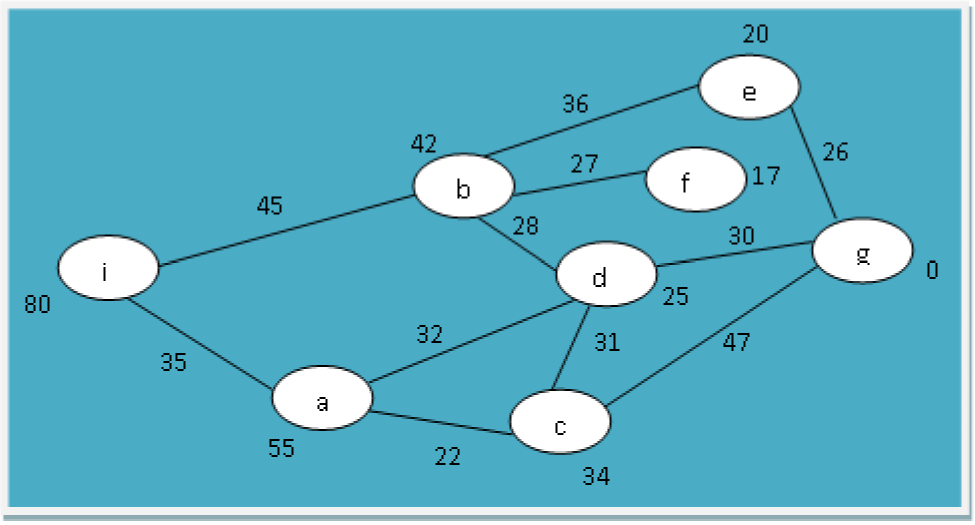
When the choice is 2:



```
In [13]: runfile('E:/4.1 semester/AI Lab/lab3/GreedyBestFirst.py',
wdir='E:/4.1 semester/AI Lab/lab3')
Menu:

1. Show Tree
2. Execute Greedy Best First Search

Enter the Choice:2
Path found: ['I', 'B', 'E', 'G']
```

Lab Exercise 4(2): Implement in generic ways (as multi-modular and interactive systems) the A* Search in Python.

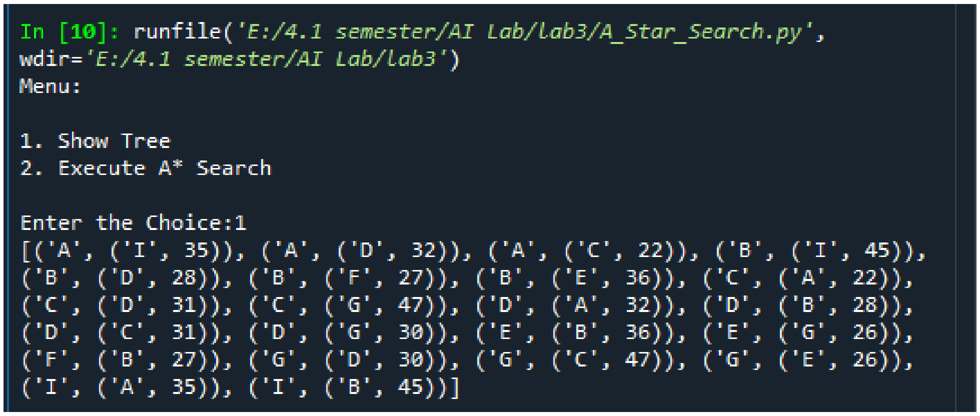
The given graph:



Python Code:

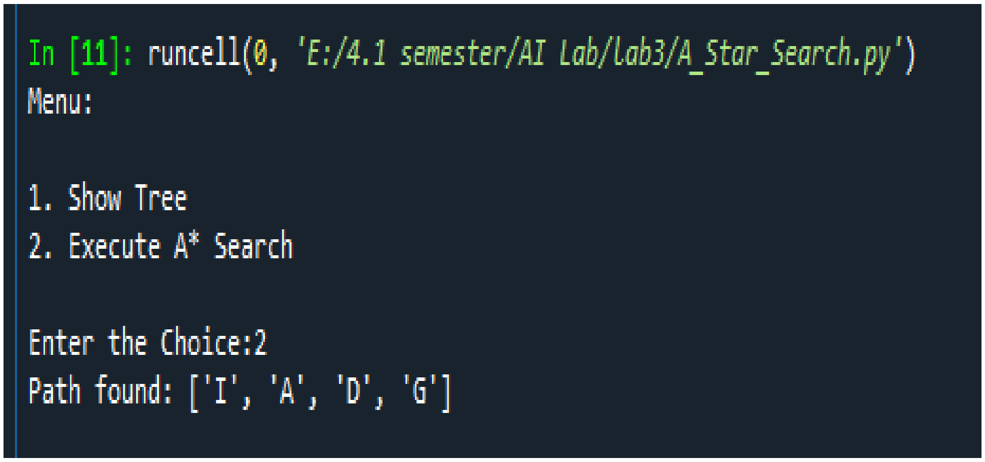
```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 24 19:03:33 2020

@author: Hp
"""
class Graph:

  def __init__(self, adjacency_list):
    self.adjacency_list = adjacency_list

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

  def hr(self, n):
    H = {
      'A': 55,
      'B': 42,
      'C': 34,
      'D': 25,
      'E': 20,
      'F': 17,
      'G': 0,
      'I': 80
    }
    return H[n]

def a_star_algorithm(self, start_node, stop_node):

    open_list = set([start_node])
    closed_list = set([])

    #in g , storing the values of distances from start node to all other nodes
    g = {}

    g[start_node] = 0

    parents = {}
    parents[start_node] = start_node

    while len(open_list) > 0:
        n = None

        # find a node with the lowest value of f() - evaluation function
        for v in open_list:
            if n == None or g[v] + self.hr(v) < g[n] + self.hr(n):
                n = v;

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

        if n == stop_node:
            reconst_path = []

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

            reconst_path.append(start_node)

            reconst_path.reverse()

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

        # for all neighbors of the current node do
        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)


            open_list.remove(n)
            closed_list.add(n)
            #print(closed_list)
        print('Path does not exist!')
        return None





# GRAPH MENTIONED IN THE LAB 3 SESSION

adjacency_list = {
    'A': [('I', 35), ('D', 32), ('C', 22)],
    'B': [('I', 45), ('D', 28), ('F', 27),('E',36)],
    'C': [('A', 22), ('D', 31), ('G', 47)],
    'D': [('A', 32), ('B', 28), ('C', 31),('G',30)],
    'E': [('B', 36), ('G', 26)],
    'F': [('B', 27)],
    'G': [('D', 30), ('C', 47), ('E', 26)],
    'I': [('A', 35), ('B', 45)]

}


print("Menu:\n")
print("1. Show Tree")
print("2. Execute A* Search")

choice=(str(input("Enter the Choice:")))

if choice=='1':

    def generate_edges(adjacency_list):
        edges = []
        for node in adjacency_list:
            for neighbour in adjacency_list[node]:
                edges.append((node, neighbour))
        return edges
    print(generate_edges(adjacency_list))


elif choice=='2':
    graph1 = Graph(adjacency_list)
    graph1.a_star_algorithm('I', 'G')
```

Sample Output:
When the choice is 1:



```
In [10]: runfile('E:/4.1 semester/AI Lab/lab3/A_Star_Search.py',
wdir='E:/4.1 semester/AI Lab/lab3')
Menu:

1. Show Tree
2. Execute A* Search

Enter the Choice:1
[('A', ('I', 35)), ('A', ('D', 32)), ('A', ('C', 22)), ('B', ('I', 45)),
('B', ('D', 28)), ('B', ('F', 27)), ('B', ('E', 36)), ('C', ('A', 22)),
('C', ('D', 31)), ('C', ('G', 47)), ('D', ('A', 32)), ('D', ('B', 28)),
('D', ('C', 31)), ('D', ('G', 30)), ('E', ('B', 36)), ('E', ('G', 26)),
('F', ('B', 27)), ('G', ('D', 30)), ('G', ('C', 47)), ('G', ('E', 26)),
('I', ('A', 35)), ('I', ('B', 45))]
```

When the choice is 2:



```
In [11]: runcell(0, 'E:/4.1 semester/AI Lab/lab3/A_Star_Search.py')
Menu:

1. Show Tree
2. Execute A* Search

Enter the Choice:2
Path found: ['I', 'A', 'D', 'G']
```