Assignment-2

Name: Aachal Dange

Roll No.:CO3010

```
class Graph:     def __init__(self,
adjacency_list):         self.adjacency_list
= adjacency_list


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


   def h(self, n):
H = {
          'A': 11,
          'B': 6,
          'C': 99,
          'D': 1,
          'E': 7,
          'G': 0
       }

       return H[n]


   def a_star_algorithm(self, start_node, stop_node):
       open_list = set([start_node])
closed_list = set([])

       g = {}

       g[start_node] = 0
```

```
        parents = {}
parents[start_node] = start_node
while len(open_list) > 0:
            n = None

            for v in open_list:                 if n == None or g[v] +
self.h(v) < g[n] + self.h(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 (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('Path does not exist!')
return None

adjac_lis = {
   'A': [('B', 2), ('E', 3)],
   'B': [('C', 1), ('G', 9)],
   'C': None,
   'D': [('G', 1)],
   'E': [('D', 6)]
}

graph = Graph(adjac_lis)
graph.a_star_algorithm('A', 'G')
```

OUTPUT :