Experiment 3

Objective

To be able to model a given problem in terms of state space search problem and solve the same using informed search techniques.

Problem Statement

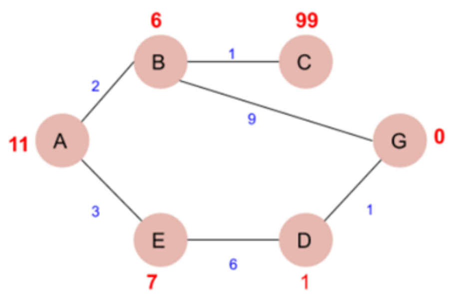
To apply an informed search technique (A* search algorithm) for finding out the minimum cost path from a source node to a goal node in a graph (see the figure given below for the graph).

Lab Exercise

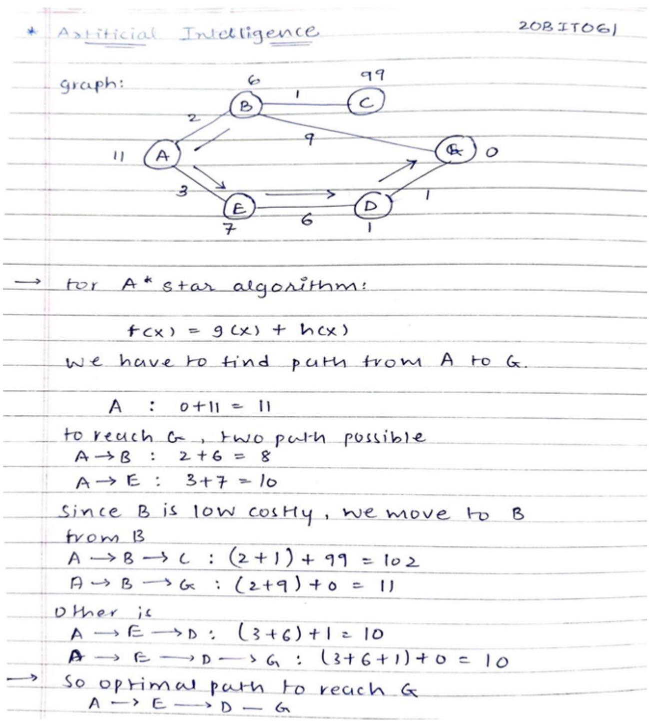
Write the program for finding out the path from a source node to a goal node in a graph. Ask the user to provide the details of the graph number of nodes, number of edges, weights of edges, h(n) values, etc.

Program Code

Example:



Solution



* Artificial Intelligence 20BIT061

graph:

6 99 B 1 C 2 9 11 A G 0 3 1 E 6 D 7 1

→ for A* star algorithm:

f(x) = g(x) + h(x)

we have to find path from A to G.

A : 0+11 = 11

to reach G, two path possible

A → B : 2 + 6 = 8

A → E : 3+7 = 10

Since B is low costly, we move to B

from B

A → B → C : (2+1) + 99 = 102

A → B → G : (2+9) + 0 = 11

Other is

A → E → D : (3+6) + 1 = 10

A → E → D → G : (3+6+1) + 0 = 10

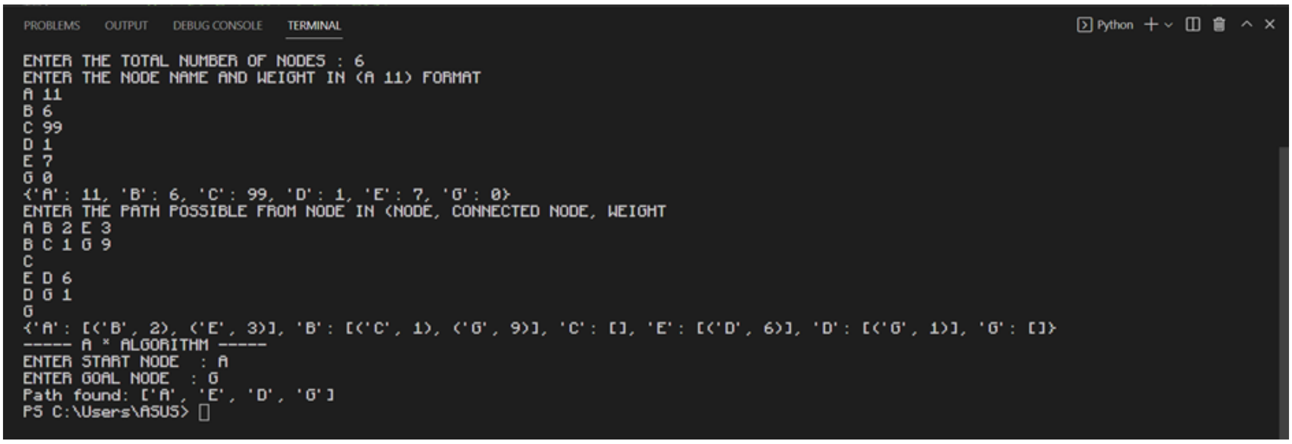
→ So optimal path to reach G

A → E → D → G

```python
def aStarAlgo(start_node, stop_node):
        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all nodes
        #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node
        while len(open_set) > 0:
            n = None
            #node with lowest f() is found
            for v in open_set:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                    n = v
            if n == stop_node or Graph_nodes[n] == None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    #nodes 'm' not in first and last set are added to first
                    #n is set its parent
                    if m not in open_set and m not in closed_set:
                        open_set.add(m)
                        parents[m] = n
                        g[m] = g[n] + weight
                    #for each node m,compare its distance from start i.e g(m) to the
                    #from start through n node
                    else:
                        if g[m] > g[n] + weight:
                            #update g(m)
                            g[m] = g[n] + weight
                            #change parent of m to n
                            parents[m] = n
                            #if m in closed set,remove and add to open
                            if m in closed_set:
                                closed_set.remove(m)
                                open_set.add(m)
            if n == None:
                print('Path does not exist!')
                return None
            # if the current node is the stop_node
            # then we begin reconstructin the path from it to the start_node
            if n == stop_node:
                path = []
                while parents[n] != n:
                    path.append(n)
                    n = parents[n]
                path.append(start_node)
                path.reverse()
                print('Path found: {}'.format(path))
                return path
            # remove n from the open_list, and add it to closed_list
            # because all of his neighbors were inspected
            open_set.remove(n)
            closed_set.add(n)

        print('Path does not exist!')
        return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
# n is total number of nodes in graph, graph node and weight
n= int(input("ENTER THE TOTAL NUMBER OF NODES : "))
d ={}
print("ENTER THE NODE NAME AND WEIGHT IN (A 11) FORMAT")
for i in range(n):
    text = input().split()
    d[text[0]] = int(text[1])
```

```python
print(d)

def heuristic(n):
        H_dist = d
        return H_dist[n]

#Describe your graph here
# Graph_nodes = {
#      'A': [('B', 2), ('E', 3)],
#      'B': [('C', 1),('G', 9)],
#      'C': None,
#      'E': [('D', 6)],
#      'D': [('G', 1)],
#      'G': None
# }
  # {
        #      'A': 11,
        #      'B': 6,
        #      'C': 99,
        #      'D': 1,
        #      'E': 7,
        #      'G': 0,
        # }
# graph inputs
print("ENTER THE PATH POSSIBLE FROM NODE IN (NODE, CONNECTED NODE, WEIGHT")
Graph_nodes ={}
for i in range(n):
        p = input().split()
        # p = [a,b,'2',e,4]
        Graph_nodes[p[0]] = [tuple((p[j],int(p[j+1]))) for j in range(1,len(p)-1,2)]
print(Graph_nodes)
print("----- A * ALGORITHM -----")
start = input('ENTER START NODE  : ')
goal = input('ENTER GOAL NODE  : ')
aStarAlgo(start, goal)
```

Output



```
ENTER THE TOTAL NUMBER OF NODES : 6
ENTER THE NODE NAME AND WEIGHT IN (A 11) FORMAT
A 11
B 6
C 99
D 1
E 7
G 0
{'A': 11, 'B': 6, 'C': 99, 'D': 1, 'E': 7, 'G': 0}
ENTER THE PATH POSSIBLE FROM NODE IN (NODE, CONNECTED NODE, WEIGHT
A B 2 E 3
B C 1 G 9
C
E D 6
D G 1
G
{'A': [('B', 2), ('E', 3)], 'B': [('C', 1), ('G', 9)], 'C': [], 'E': [('D', 6)], 'D': [('G', 1)], 'G': []}
----- A * ALGORITHM -----
ENTER START NODE  : A
ENTER GOAL NODE  : G
Path found: ['A', 'E', 'D', 'G']
PS C:\Users\ASUS>
```

Conclusion

Program for finding the path using A* Algorithm from taking input from user for weight, number of edges, number of nodes, h(n) etc. has been successfully observed. The solution using program code and calculation solution are same and verified.