**Ex:4 A\* SEARCH ALGORITHM**

**Name: A.Ramya**

**Reg No.:231801135**

PROGRAM:

from collections import deque

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': 1, 'B': 1, 'C': 1, 'D': 1 }

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

OUTPUT:

