Name: Vikash Mishra

Reg No.: RA1811003010344

Experiment: A\* Algorithm

Statement : A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Algorithm :

1. Initialize the open list

2. Initialize the closed list

put the starting node on the open

list (you can leave its f at zero)

3. while the open list is not empty

a) find the node with the least f on

the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their

parents to q

d) for each successor

i) if successor is the goal, stop search

successor.g = q.g + distance between

successor and q

successor.h = distance from goal to

successor (This can be done using many

ways, we will discuss three heuristics-

Manhattan, Diagonal and Euclidean

Heuristics)

successor.f = successor.g + successor.h

ii) if a node with the same position as

successor is in the OPEN list which has a

lower f than successor, skip this successor

iii) if a node with the same position as

successor is in the CLOSED list which has

a lower f than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

Code :

tree = {'Z': [['A', 2], ['B', 6], ['C', 9]],

'A': [['Z', 2], ['D', 5], ['F', 8]],

'B': [['Z', 6], ['F', 3]],

'C': [['Z', 9], ['F', 4]],

'D': [['A', 5]]}

tree2 = {'S': [['A', 1], ['B', 2]],

'A': [['S', 1]],

'B': [['S', 2], ['C', 3], ['D', 4]],

'C': [['B', 2], ['E', 5], ['F', 6]],

'D': [['B', 4], ['G', 7]],

'E': [['C', 5]],

'F': [['C', 6]]

}

heuristic = {'Z': 8, 'A': 8, 'B': 4, 'C': 3, 'D': 5000, 'F': 0}

cost = {'Z': 0} # total cost for nodes visited

def AStarSearch():

global tree, heuristic

closed = [] # closed nodes

opened = [['Z', 8]] # opened nodes

'''find the visited nodes'''

while True:

fn = [i[1] for i in opened] # fn = f(n) = g(n) + h(n)

chosen\_index = fn.index(min(fn))

node = opened[chosen\_index][0] # current node

closed.append(opened[chosen\_index])

del opened[chosen\_index]

if closed[-1][0] == 'F': # break the loop if node G has been found

break

for item in tree[node]:

if item[0] in [closed\_item[0] for closed\_item in closed]:

continue

cost.update({item[0]: cost[node] + item[1]}) # add nodes to cost dictionary

fn\_node = cost[node] + heuristic[item[0]] + item[1] # calculate f(n) of current node

temp = [item[0], fn\_node]

opened.append(temp) # store f(n) of current node in array opened

'''find optimal sequence'''

trace\_node = 'F' # correct optimal tracing node, initialize as node G

optimal\_sequence = ['F'] # optimal node sequence

for i in range(len(closed)-2, -1, -1):

check\_node = closed[i][0] # current node

if trace\_node in [children[0] for children in tree[check\_node]]:

children\_costs = [temp[1] for temp in tree[check\_node]]

children\_nodes = [temp[0] for temp in tree[check\_node]]

'''check whether h(s) + g(s) = f(s). If so, append current node to optimal sequence

change the correct optimal tracing node to current node'''

if cost[check\_node] + children\_costs[children\_nodes.index(trace\_node)] == cost[trace\_node]:

optimal\_sequence.append(check\_node)

trace\_node = check\_node

optimal\_sequence.reverse() # reverse the optimal sequence

return closed, optimal\_sequence

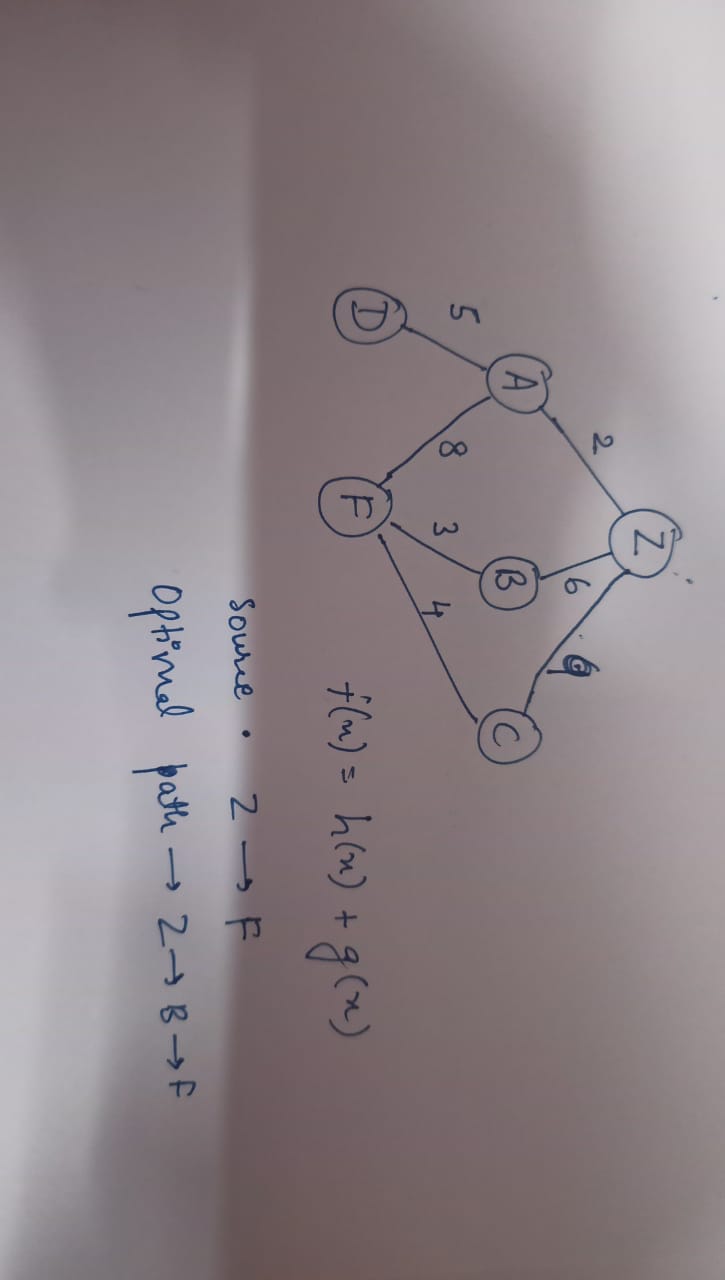
if \_\_name\_\_ == '\_\_main\_\_':

visited\_nodes, optimal\_nodes = AStarSearch()

print('visited nodes: ' + str(visited\_nodes))

print('optimal nodes sequence: ' + str(optimal\_nodes))

Calculation :



Output :

2

Result :

The give algorithm was implemented and thus optimal path was obtained.