East West University

Project

Title: Maze solver using A* algorithm

CSE366

Submitted by-

Name: Md. Shadman Sakib
Id: 2019-2-60-026

Name: Md Shorif hossain
Id: 2019-2-60-039

Name: Md. Shajibul Islam
Id: 2019-1-60-031

Name: Mehzabin Meem
Id: 2019-2-60-019

Name: Iftekhar Ahmed Shohan
Id: 2018-2-60-103

Submitted to-

Dr. Md Rifat Ahmed Rashid
Assistant Professor
Department of Computer Science & Engineering
East West University

Introduction:

The A*(A-star) search algorithm is one of the most commonly used algorithms for path planning. It relies mainly on heuristics to find the least costly route between the two points.

A* search algorithm will find the path between source and destination. Source and destination location must be selected. Then select one of the three elements that can move on the map. Maze solving is considering the image as black and white image. There is one element that can only move on the maze white area. According to the information, the algorithm will find the shortest path. Without knowing which element is moving on the map, the system will find the path supposing that the element can move anywhere on the map.

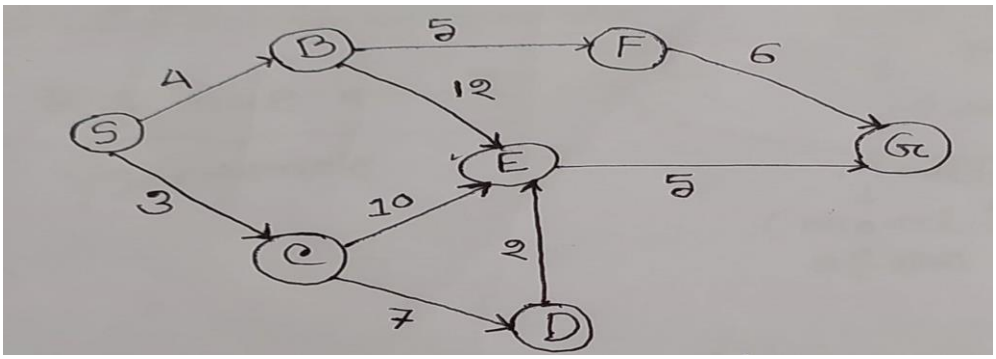
$f(n)=g(n)+h(n)$

Here,

$g(n)$=Actual cost from start to node n.

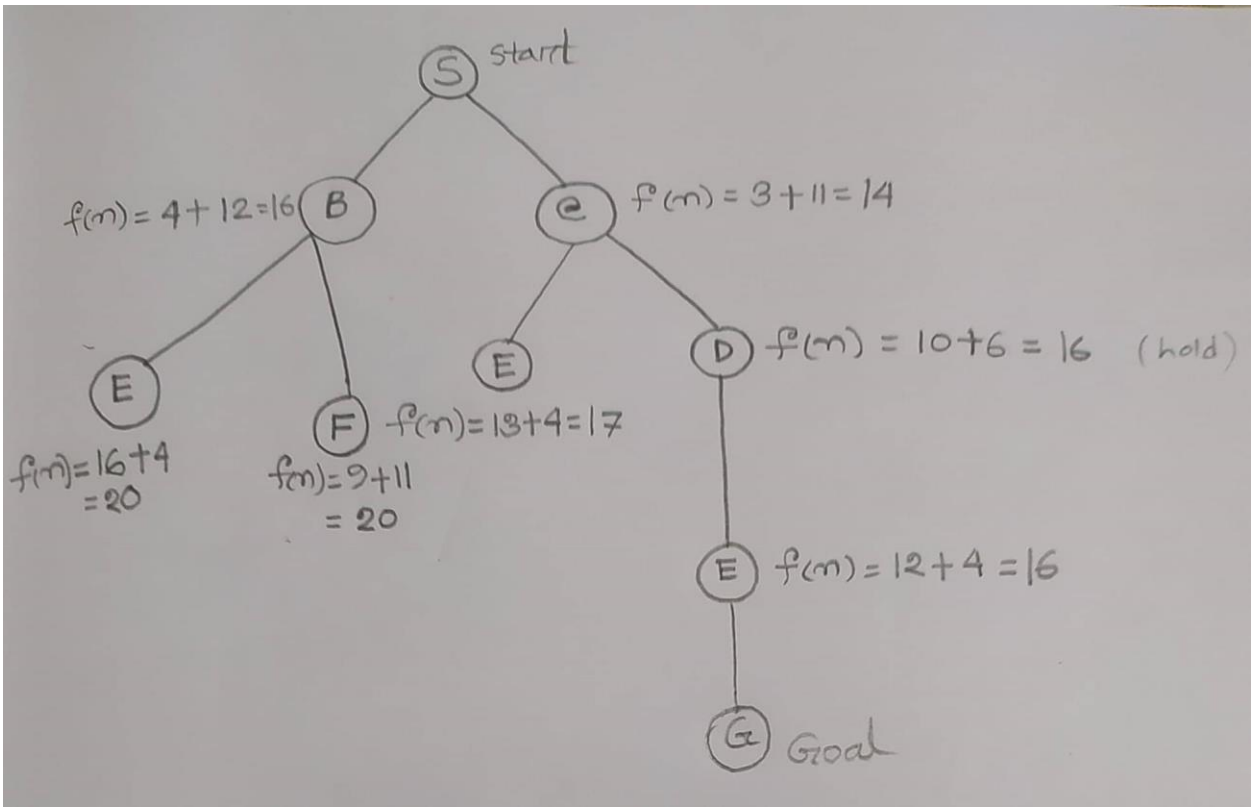
$h(n)$=Estimation cost from node n to goal node.

If we take a graph,



In this graph starting node S and Goal node G. The heuristic cost of the graph's node are given below-

Node	Cost
S → G	14
B → G	12
C → G	11
F → G	11
O → G	6
E → G	4
G → G	0



Shortest Path: S → C → D → E → G

A* Search Algorithm:

Code:

Function of A* algorithm:

This is the code of A* algorithm that we used in the lab.

```
import math
path = []

def aStar(start, stop):
    open_set = {start}
    closed_set = set()
    g = {}
    parents = {}
    g[start] = 0
    parents[start] = start
    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop or graph_nodes[n] == None:
            pass
        else:
            neighbours=get_neighbors(n)
            ng=neighbours.split()
            for i in range(0,len(ng),2):
                if ng[i] not in open_set and ng[i] not in closed_set:
                    open_set.add(ng[i])
                    parents[ng[i]] = n
                    g[ng[i]] = g[n] + float(ng[i+1])
                else:
                    if g[ng[i]] > g[n] + float(ng[i+1]):
                        g[ng[i]] = g[n] + float(ng[i+1])
                        parents[ng[i]] = n
        if n == None:
            print("Path does not exist")
            return None
```

```
        if n == stop:
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start)
            path.reverse()
            return path
        open_set.remove(n)
        closed_set.add(n)
    print("Path does not exist")
    return None
```

Function for getting neighbor nodes and Heuristic values:

get_neighbors() function gives the adjacent nodes of a node. Heuristic function gives the heuristics values by calculating the distance of two nodes.

```
def get_neighbors(v):
    if v in graph_nodes:
        return graph_nodes[v]


def heuristic(n):
    n=n.split(',')
    gl=goal.split(',')
    return math.sqrt((int(n[0]) - int(gl[0])) ** 2 + (int(n[1]) - int(gl[1])) ** 2)
```

Input part, Start and Goal:

Taking input line by line of the maze from user. Then, assigning the start node and goal node from the maze.

```
Map_List=[]
while True:
    ln=input()
    if ln:
        Map_List.append(list(ln))
    else:
        break

for i in range(0,len(Map_List)):
    for j in range(0,len(Map_List[i])):
        if(Map_List[i][j]=='o'):
            start=str(i)+','+str(j)
for i in range(0,len(Map_List)):
    for j in range(0,len(Map_List[i])):
        if(Map_List[i][j]=='x'):
            goal=str(i)+','+str(j)
```

Adjacent nodes, Valid nodes, Move costs:

Selecting adjacent nodes of a node and storing it in a dictionary in python. Selecting only valid nodes for moving which is not #. Also, adding the move costs with the nodes.

```
graph_nodes = {}
for i in range(1,len(Map_List)-1):
    for j in range(0,len(Map_List[i])):
        if(Map_List[i][j]!='#'):
            key=str(i)+','+str(j)

            Values=''
            Values1=''
            Values2=''
            Values3=''
            Values4=''
            Values5=''
            Values6=''
            Values7=''
            Values8=''

            if(Map_List[i][j-1]!='#'):
                Values1=str(i)+','+str(j-1)+" "+"1.0" +" "

            if(Map_List[i][j+1]!='#'):
               Values2=str(i)+','+str(j+1)+" "+"1.0"+" "

            if(Map_List[i-1][j]!='#'):
                Values3=str(i-1)+','+str(j)+" "+"1.0"+" "

            if(Map_List[i+1][j]!='#'):
                Values4=str(i+1)+','+str(j)+" "+"1.0"+" "

            if(Map_List[i-1][j+1]!='#'):
                Values5=str(i-1)+','+str(j+1)+" "+"1.7"+" "
```

```
            if(Map_List[i-1][j-1]!='#'):
                Values6=str(i-1)+','+str(j-1)+" "+"1.7"+" "

            if(Map_List[i+1][j+1]!='#'):
                Values7=str(i+1)+','+str(j+1)+" "+"1.7"+" "

            if(Map_List[i+1][j-1]!='#'):
                Values8=str(i+1)+','+str(j-1)+" "+"1.7"+" "

            Values=(Values1+Values2+Values3+Values4+Values5+
                    Values6+Values7+Values8)
            graph_nodes[key]=Values
```

Function of A* algorithm calling and Printing the output:

Calling A* algorithm function to run the maze and printing the shortest path. Here, “path” is a list which is a global variable and used in “aStar” function to store the shortest path.

```
aStar(start,goal)
Map_List
for i in range(1,len(path)-1):
    a=path[i]
    a=a.split(',')
    Map_List[int(a[0])][int(a[1])]='.'
for x in Map_List:
    print("".join(map(str, x)))
```

Input and Output: