Experiment-2

Write a Program to Implement A* Search

Date: 16/8/ 2024

AIM

Write a Program to Implement A* Search.

ALGORITHM

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

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

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

SOURCE CODE

```
cubile len(open-set) >0:
  n = None
   for v in open-set:
     if n == None or g[v] + heuristic[v] <g[n]
                                  theunstic(n):
   if n == stop-node or Graph-nodes[n] == None:
       Paus
   else:
     for (m, weight) in get_neighbors(n):
         if m not in open_set and m not in closed
            open_set.add(m)
            Parents [m] = n
            g[m] =g[n] + weight
        else:
if g[ni] > g[ni] + weight:
                g[m] = g[n] + weight.
                parent [m] =n
                if min closed - set:
                     closed - set . remove (m)
                    open-set. add (m)
 if n == None:
   Print ( path does not exist! )
   return None
if n = = stop-node:
  path = [ ]
  while parents [n] = n:
     path. append (n)
      n = parents[n]
 Path-append (start-node)
Path apreverse ()
```

```
print ( Path found: ()'. format (path))
       return path
   open-set. remove (n)
   closed - set. add(n)
 Print ('Path does not exist!')
 return None
                   (3), (1, 'a'), (2, 9), (6)
def get_neighbors(v):
  if v in Graph_nodes:
     return Ecraph-nodes[v]
  else:
     return None
               (r. 14'). (1.5'). (s. 14') ] 13'
def heuristic(n):
  H-dist = {
                    · [(E; 'I), (E,'31)]; 'S'
     A: 110
                      . [(e,'I), (r,')) ] . H'
     B': 6,
     ~ (E. 5, ). (e. H). (E. 18), (E. 51)] (I
      'D' : 7,
      'E' : 3,
     'F':6,
      'G' : 5,
     'H' : 3,
      'I':1,
      '7':0
  4
```

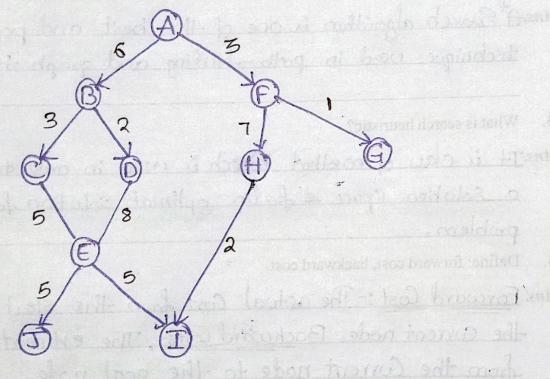
```
return H-distEn]
Erraph-noder = {
    ", :[('B', 6), ('F', 3)],
    B': [('A',6), ('c',3), ('p',2)],
     'C': [('B', 3), ('D', 1), (E', 5)],
     'b': [('B',2), ('c',1), ('E',8)],
     'E': [('c',5),('D',8),(T',5),('T',5)],
     'F': [('A',3), ('G',1), ('H',7)],
     G': [('F',1), ('I',3)],
     'H': [('F',7), (I',2)],
     'I': [('E',5), ('G',3), ('H',2), ('J',3)],
 astarAlgo ('A', 'J')
```

01 74

OUTPUT

Path found: {'A', 'F', G', 'I', 'J']

Graph:



vois for salt to others grow some a sant ad and

sound settle of storing. (b) also storing

the depity of the depth of the optimal of

Resulti 18/8/2 Using JOF (Python 3.9.64 -bit) we have succendily enoughly search.

VIVA QUESTIONS

1. What are the types of informed search algorithms?

Ans. BFS, A* Search, Genetic Algorithms

2. Define: A* Search.

Ans. A Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

3. What is search heuristic?

Ans. It is class of method which is used in order to search a Solution space of for an optimal solution for a problem.

4. Define: forward cost, backward cost.

Ans. Forward Cost: The actual Cost from the start node to the current node. Backward Cost: The estimated Cost from the current node to the goal node.

5. What is the time complexity and space complexity of A* search algorithm?

ans. The time & space Complexity of the A* search algorithm are both O(b^d), where b is the branching factor and d is the depth of the optimal Solution. These Complexities can vary depending on the quality of the heuristic used.

to the same places of

Week-2

AIM:-Write a program to Implement At Search.

Algorithm:

It is best-known form of Bert First Search. It avoids expanding paths that are already expensive, but expands most promising paths first.

+(n) = g(n) + h(n), where

- · g(n) the Cost (so far) to reach the node
- . h(n) extimated cost to get from the node to the
- . f(n) extimated total Cost of path through n to goal.
 It is implemented using priority queue by increasing f(n).

to log 1603 High taking

Source Code:=

def astarAlgo (start_node, stop_node):

open_set = set (start_node)

closed_set = set ()

9 = 6 3

Parent = 13

g [start_node] = 0
Parents [start_node] = start_node

while len (open-set)>0:

n = None

for vin open-set:

Paus

if n == None or g[v] + heunstic(v) < g[n] +
heunstic(n);

if n == stop_node or Graph_nodes [n] == None:

```
elses
     for (m, weight) in get-neighbors (n):
        if m not in open-set and m not in closed
            open-set-add (m)
           Parents [m] =n
            g [m] = g [n] + weight
         else:
          if g [m] > g [m] + weight:
              g [m] = g [n] + cue ighto
                                a) Al- Calle
              Parents [m] =n
             if min closed_set:
                  closed_set.remove(m)
     open-set-add (m)
    if n = = Hone: / my warm habyand skym is hi
       Print ('path does not exist;')
       return None
    if n = stop-node:
       Path = []
       while parents [n] h=n:
          Path. appenden)
          n = payents [n]
      Roth append (start-node)
      Path. renderse()
      Print ('path found: 29'. formed (path))
      return path
   open-set. remove(n)
   closed_set.add(n)
Print ('Path does not exist')
return None
```

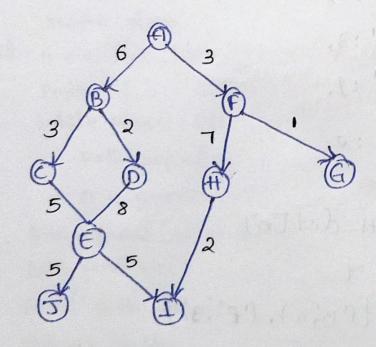
```
def get-neighbors(v):
   if v in Euraph_nodes:
      return Graph-nodes[v]
   else:
      return None
def heunstic (n):
  H-dist = {
        'A' & 11.
   (8, 8), 6, (14'), (8, 9'), (8, 9') (8, 9')
       'C':5,
                           ('T', 'A') oplanted
        'D';7,
        'E': B,
                7. I' 10 "1" 1 1 1 1 bounds att
        16:60
        G :5,
        141:3,
        T':1,
        7:0
    return-t1-dist [n]
Graph - nodes = {
     A' & [('B', 6), ('f', 3)]
    'B': [('A',6), ('c',3), ('D',2)),
    'c': [('B',3),('D',1),('E',5)],
    'O': [('B',2), ('C',1), ('E',8)],
```

'E':[(c',5), ('D',8), ('I',5), ('J',5)], F': [('A',3), ('G',1), ('H',7)], 'E': [(F',1),('I',3)], 'H':[('F',7),('I',3)], 'I': [('E,5), ('G',3), ('H',2), ('5',3)], aStarAlgo ('A', 'J')

Output :

Path found: {' A', 'f', 'G', 'I', 'J'}

Graph:



Result:

Using IDLE (Python 3.9.64 - bit) we have succendily executed A* search