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

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = { }
    Parents = { }
    g[start_node] = 0
    Parents[start_node] = start_node
```

```
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_node or Graph_nodes[n] == None:
        pass
    else:
        for (m, weight) in get_neighbors(n):
            if m not in open_set and m not in closed_set:
                open_set.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_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.reverse()
```

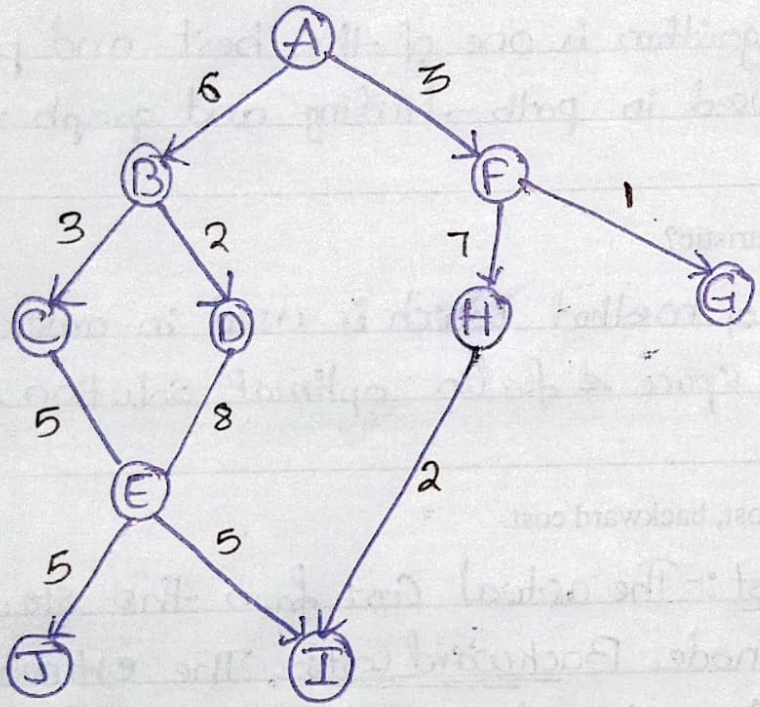
```
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A' : 11,
        'B' : 6,
        'C' : 5,
        'D' : 7,
        'E' : 3,
        'F' : 6,
        'G' : 5,
        'H' : 3,
        'I' : 1,
        'J' : 0
    }
```

```
        return H_dist[n]
Graph_nodes = {
    'A' : [('B', 6), ('F', 3)],
    'B' : [('A', 6), ('C', 3), ('D', 2)],
    'C' : [('B', 3), ('D', 1), ('E', 5)],
    'D' : [('B', 2), ('C', 1), ('E', 8)],
    'E' : [('C', 5), ('D', 8), ('I', 5), ('J', 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')
```

OUTPUT

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

Graph:-



Result:-

Using IDE (Python 3.9.64 -bit) we have Succesfully created A* 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 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.

Week-2

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:-

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = { }
    Parents = { }
    g[start_node] = 0
    Parents[start_node] = start_node
    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_node or Graph_nodes[n] == None:
            pass
```

```
else:
    for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in closed_set:
            open_set.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_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.reverse()
        Print('Path found: {}'.format(path))
        return path
    open_set.remove(n)
    closed_set.add(n)
Print('Path does not exist')
return None
```

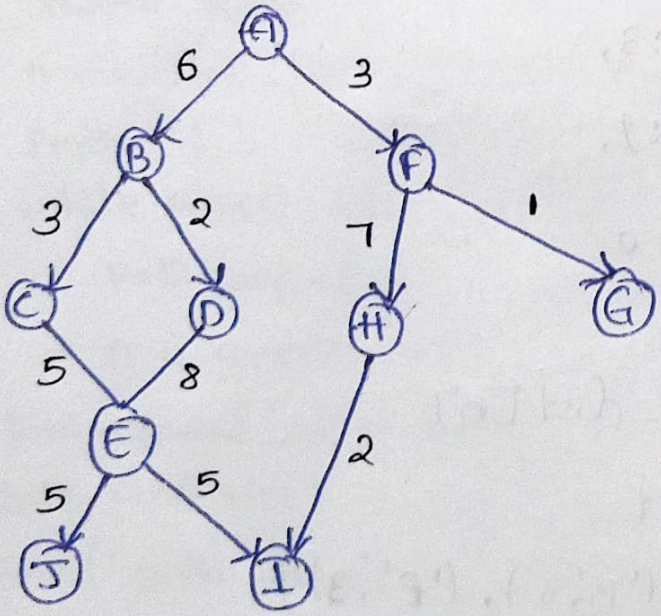
```
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)]
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
```

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

Output :-

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

Graph :-



Results :-

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