Experiment No. 04

Aim: Implementation of Informed Search Techniques.

4.1 Implement Path finding Problem Using A* Algorithm.

Objectives:

To impart basic proficiency in representing difficult real life problems in a state space representation so as to solve them using AI techniques.

Outcomes:

Analyze and formalize the problem as a state space, graph, design heuristics and select amongst different search or game based techniques to solve them.

Theory:

4.1 Implement Path finding Using A* Algorithm.

a) **Introduction to Path finding Problem and A* Algorithm.**

Path finding algorithms are important because they are used in applications like google maps, satellite navigation systems, routing packets over the internet. The usage of pathfinding algorithms isn’t just limited to navigation systems.

A* Search algorithm is one of the best and popular techniques used in path-finding and graph traversals.It is really a smart algorithm which separates it from the other conventional algorithms.A* (pronounced as "A star") is a computer algorithm that is widely used in pathfinding and graph traversal. The algorithm efficiently plots a walkable path between multiple nodes, or points, on the graph.the A* algorithm introduces a heuristic into a regular graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made.

b) **Algorithm/Pseudocode (A* 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:

```
from queue import heappop, heappush
from math import inf

class Graph:
    def __init__(self, directed=True):
        self.edges = {}
        self.huristics = {}
        self.directed = directed

    def add_edge(self, node1, node2, cost = 1, __reversed=False):
```

```
        try: neighbors = self.edges[node1]
        except KeyError: neighbors = {}
        neighbors[node2] = cost
        self.edges[node1] = neighbors
        if not self.directed and not __reversed: self.add_edge(node2, node1, cost, True)

    def set_huristics(self, huristics={}):
        self.huristics = huristics

    def neighbors(self, node):
        try: return self.edges[node]
        except KeyError: return []

    def cost(self, node1, node2):
        try: return self.edges[node1][node2]
        except: return inf


    def a_star_search(self, start, goal):
        found, fringe, visited, came_from, cost_so_far = False, [(self.huristics[start], start)],
set([start]), {start: None}, {start: 0}
        print('{:11s} | {}'.format('Expand Node', 'Fringe'))
        print('--------------------')
        print('{:11s} | {}'.format('-', str(fringe[0])))
        while not found and len(fringe):
            _, current = heappop(fringe)
            print('{:11s}'.format(current), end=' | ')
            if current == goal: found = True; break
            for node in self.neighbors(current):
                new_cost = cost_so_far[current] + self.cost(current, node)
                if node not in visited or cost_so_far[node] > new_cost:
                    visited.add(node); came_from[node] = current; cost_so_far[node] = new_cost
                    heappush(fringe, (new_cost + self.huristics[node], node))
            print(', '.join([str(n) for n in fringe]))
        if found: print(); return came_from, cost_so_far[goal]
        else: print('No path from {} to {}'.format(start, goal)); return None, inf

    @staticmethod
    def print_path(came_from, goal):
        parent = came_from[goal]
```

```
        if parent:
            Graph.print_path(came_from, parent)
        else: print(goal, end='');return
        print(' =>', goal, end='')


    def __str__(self):
        return str(self.edges)

graph = Graph(directed=True)
graph.add_edge('A', 'B', 4)
graph.add_edge('A', 'C', 1)
graph.add_edge('B', 'D', 3)
graph.add_edge('B', 'E', 8)
graph.add_edge('C', 'C', 0)
graph.add_edge('C', 'D', 7)
graph.add_edge('C', 'F', 6)
graph.add_edge('D', 'C', 2)
graph.add_edge('D', 'E', 4)
graph.add_edge('E', 'G', 2)
graph.add_edge('F', 'G', 8)
graph.set_huristics({'A': 8, 'B': 8, 'C': 6, 'D': 5, 'E': 1, 'F': 4, 'G': 0})
start, goal = 'A', 'G'
traced_path, cost = graph.a_star_search(start, goal)
if (traced_path): print('Path:', end=' '); Graph.print_path(traced_path, goal); print('\nCost:', cost)
```

OUTPUT:

```
Python 3.7.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/impre/AppData/Local/Programs/Python/Python37/pathfinding.py
Expand Node | Fringe
--------------------
-           | (8, 'A')
A           | (7, 'C'), (12, 'B')
C           | (11, 'F'), (13, 'D'), (12, 'B')
F           | (12, 'B'), (13, 'D'), (15, 'G')
B           | (12, 'D'), (13, 'E'), (13, 'D'), (15, 'G')
D           | (12, 'E'), (13, 'D'), (15, 'G'), (13, 'E')
E           | (13, 'D'), (13, 'E'), (15, 'G'), (13, 'G')
D           | (13, 'E'), (13, 'G'), (15, 'G')
E           | (13, 'G'), (15, 'G')
G           |
Path: A => B => D => E => G
Cost: 13
>>>
```

Conclusion: From this experiment, we understood the path finding problem and A* algorithm. Also the path finding problem using A* algorithm was implemented successfully.