|  |  |
| --- | --- |
| **EX.N0:6** | **A\*SEARCH ALGORITHM** |
| **DATE:3//04/24** |

**AIM:**

To implement the A\* Search Algorithm using python

**ALGORITHM:**

Step1:Start

Step2:initiate open list and close list

Step3:While the open list is not empty, find the node with least form the openlist, call it “q”

Step 4:Stop working when you find the destination or you can’t find the destination going through all possible points from collections import deque

**PROGRAM:**

class Graph:

    def \_\_init\_\_(self, adjacency\_list):

        self.adjacency\_list = adjacency\_list

    def get\_neighbors(self, v):

        return self.adjacency\_list[v]

    def h(self, n):

        H = {'A': 1, 'B': 1, 'C': 1, 'D': 1}

        return H[n]

    def a\_star\_algorithm(self, start\_node, stop\_node):

        open\_list = set([start\_node])

        closed\_list = set([])

        g = {}

        g[start\_node] = 0

        parents = {}

        parents[start\_node] = start\_node

        while len(open\_list) > 0:

            n = None

            for v in open\_list:

                if n is None or (g[v] + self.h(v) < g[n] + self.h(n)):

                    n = v

            if n is None:

                print('Path does not exist!')

                return None

            if n == stop\_node:

                reconst\_path = [n]

                while parents[n] != n:

                    n = parents[n]

                    reconst\_path.append(n)

                reconst\_path.reverse()

                print('Path found: {}'.format(reconst\_path))

                return reconst\_path

            open\_list.remove(n)

            closed\_list.add(n)

            for m, weight in self.get\_neighbors(n):

                if m in closed\_list:

                    continue

                if m not in open\_list:

                    open\_list.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

        print('Path does not exist!')

        return None

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    adjacency\_list = {

        'A': [('B', 1), ('C', 3)],

        'B': [('A', 1), ('D', 5)],

        'C': [('A', 3), ('D', 2)],

        'D': [('B', 5), ('C', 2)]

    }

    graph = Graph(adjacency\_list)

    graph.a\_star\_algorithm('A', 'D')

**OUTPUT:**

Path found: ['A', 'C', 'D']

**RESULT:** Thus the experiment to solve A\* search algorithm by using python has been executed and verified Successfully.