S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 2

Aim: Make use of A* Algorithm to find out goal state (Shortest path cost) in the given Graph. Make use of Breadth First Search algorithm to find the goal state in the above graph and compare the performance of both algorithms in terms of time taken.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: V/III

Academic Session: 2024-2025

Date of Performance:

Date of Submission:

AIM: Make use of A* algorithm to find out goal state in the given problem scenario of Practical No.2.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To be able to understand the concept of search algorithms.
- To be able to implement the concept of A* algorithm.

THEORY:

Many traditional search algorithms are used in AI applications.

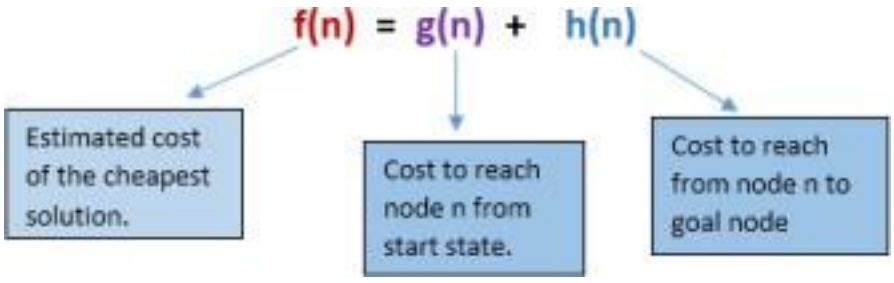
The Search Algorithms Are Classified in Two Types:

Uninformed Search: Also called blind, exhaustive or brute-force search, it uses no information about the problem to guide the search and therefore may not be very efficient. **Informed Search:** Also called heuristic or intelligent search, this uses information about the problem to guide the search—usually guesses the distance to a goal state and is therefore efficient, but the search may not be always possible.

A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state. It has combined features of Uniform Cost Search and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



Algorithm of A* Search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

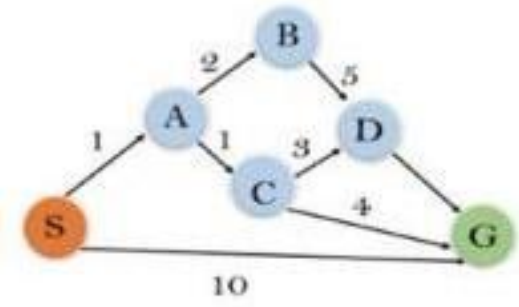
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

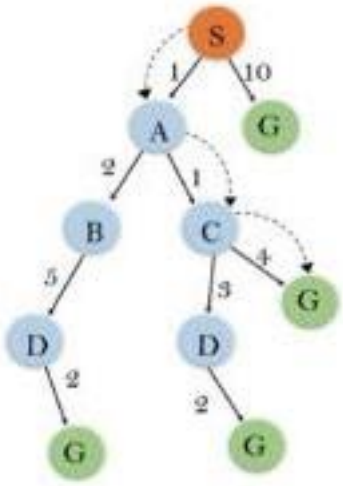
Consider the graph for the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



PSEUDOCODE:

```
1  Create a node containing the goal state node_goal
2  Create a node containing the start state node_start
3  Put node_start on the open list
4  while the OPEN list is not empty
5  {
6  Get the node off the open list with the lowest f and call it node_current
7  if node_current is the same state as node_goal we have found the solution; break from
   the while loop
8     Generate each state node_successor that can come after node_current
9     for each node_successor of node_current
10    {
11       Set the cost of node_successor to be the cost of node_current plus the cost to get
         to node_successor from node_current
12       find node_successor on the OPEN list
13       if node_successor is on the OPEN list but the existing one is as good or better
         then discard this successor and continue
14       if node_successor is on the CLOSED list but the existing one is as good or better
         then discard this successor and continue
15       Remove occurrences of node successor from OPEN and CLOSED
16       Set the parent of node_successor to node_current
17       Set h to be the estimated distance to node_goal (Using the heuristic function)
18       Add node_successor to the OPEN list
19    }
20    Add node_current to the CLOSED list
```

PROGRAM CODE:

```
""" Unidirectional """
import queue

class PriorityQueue:
    def __init__(self):
        self.queue = queue.PriorityQueue()

    def push(self, item, priority):
        self.queue.put((priority, item))

    def pop(self):
        _, item = self.queue.get()
        return item

    def is_empty(self):
```

```
        return self.queue.empty()

# Unidirectional graph representation
graph = {
    "S": [(1, "A"), (4, "B")],
    "A": [(2, "B"),(12,"D")],
    "B": [(2, "C")],
    "C": [(3, "D")],
    "D": []
}

heuristic_values = {
    "S": 7,
    "A": 6,
    "B": 2,
    "C": 1,
    "D": 0
}

def a_star(graph, start, goal, heuristic):
    open_list = PriorityQueue()
    open_list.push(start, 0)
    g_score = {node: float('inf') for node in graph.keys()}
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph.keys()}
    f_score[start] = heuristic[start]
    came_from = {}

    while not open_list.is_empty():
        current = open_list.pop()

        if current == goal:
            return reconstruct_path(came_from, current)

        for cost, neighbor in graph[current]:
            tentative_g_score = g_score[current] + cost
            if tentative_g_score < g_score[neighbor]:
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic[neighbor]
                open_list.push(neighbor, f_score[neighbor])
                came_from[neighbor] = current

    return None

def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
```

```
    path.reverse()
    return path

if __name__ == "__main__":
    start_node = "S"
    goal_node = "D"
    path = a_star(graph, start_node, goal_node, heuristic_values)

    if path:
        print("Shortest path:", path)
    else:
        print("No path found.")
```

INPUT & OUTPUT:

Sr. No.	INPUT	OUTPUT
1.	graph = "S": [(1, "A"), (4, "B")], "A": [(2, "B"),(12,"D")], "B": [(2, "C")], "C": [(3, "D")], "D": [] heuristic_values = "S": 7, "A": 6, "B": 2, "C": 1, "D": 0	[‘S’ , ‘A’ , ‘B’ , ‘C’ , ‘D’]

COMPARISON WITH BREADTH FIRST SEARCH:

A* is an efficient search algorithm that uses a heuristic to find the shortest path more quickly, offering optimal solutions if the heuristic is good. BFS is simpler and guarantees the shortest path in unweighted graphs but can be slower and use more memory, as it explores all nodes level by level. Both have time and space complexities of O(b^d).

CONCLUSION:

Thus, successfully make use of A* algorithm to find out goal state in the given problem scenario.

DISCUSSION QUESTIONS:

Q.1) What do you mean by Search Algorithm?

Q.2) What are the differences between informed and uninformed search strategies?

Q.3) What is A* algorithm?

Q.4) What are the pros and cons of A* algorithm?

REFERENCES:

- *https://www.javatpoint.com/ai-informed-search-algorithms*
- *https://www.educative.io/answers/what-is-the-a-star-algorithm*
- *https://www.geeksforgeeks.org/a-search-algorithm/*