PART A

Experiment No.03

- **Aim:** To Implement informed A* search methods using C,python or Java.
- **Prerequisite:** Data Structure, Searching Techniques
- **Outcome:**

After successful completion of this experiment students will be able to

- Ability to analyse the local and global impact of computing in searching techniques.
- Understand, identify, analyse and design the problem, implement and validate the solution for the A* search method.
- Ability to applying knowledge of computing in search technique areas.

Tools Required: C /Java

- **Theory:**

An informed search strategy-one that uses problem-specific knowledge-can find solutions more efficiently.A key component of these algorithms is a heuristic function *h(n)*.
h(n) = estimated cost of the cheapest path from node *n* to a goal node.Admissible /heuristic never over estimated i.e. h(n)$\leq$ Actual cost.

For example, Distance between two nodes(cities)=> straight line distance and for 8-puzzel problem- Admissible heuristic can be number of misplaced tiles h(n)= 8.

A* Search technique:

It is informed search technique. It uses additional information beyond problem formulation and tree. Search is based on Evaluation function f(n). Evaluation function is based on both heuristic function h(n) and g(n).

$$\mathbf{f(n)=g(n) + h(n)}$$

It uses two queues for its implementation: open, close Queue. Open queue is a priority queue which is arranged in ascending order of f(n)

Algorithm:

- Create a single member queue comprising of Root node
- If FIRST member of queue is goal then goto step 5
- If first member of queue is not goal then remove it from queue and add to close queue.
- Consider its children if any, and add them to queue in ascending order of evaluation function f(n).
- If queue is not empty then goto step 2.

- If queue is empty then goto step 6
- Print ‘success’ and stop
- Print ‘failure’ and stop.

Performance Comparison:

- Completeness: yes
- Optimality: yes

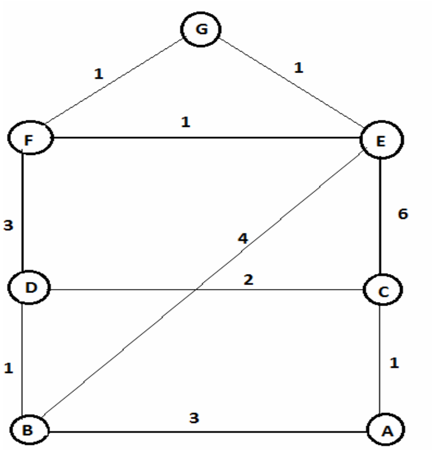
Limitation:

- It generate same node again and again
- Large Memory is required

Observation

Although A*generate many nodes it never uses those nodes for which f(n)> c* where c* is optimum cost.

Consider an example as below



OPEN/FRINGE	**CLOSE**
[A]	[ ]
[C,B]	[A]
[D,B,E,A]	[A,C]
[F,E,B,C,A]	
[G,E,B,C,A,D] SUCCESS	

Node A: Node C:

Node D:

Node F:

f(B)=g(B) + h(B)=3+5=8 f(C) =g(C) + h(C)=1 + 6=7

f(A) = g(A) + h(A)=2+7=10 f(D) = g(D) + h(D)=3+4=7 f(E) = g(E) + h(E)=7+1=8

f(F) = g(F) + h(F)=6+1=7 f(C) = g(C) + h(C)= 5+6=11 f(B) = g(B) + h(B)=4+5=9

f(E) = g(E) + h(E)=7+1=8 f(D) = g(D) + h(D)= 9+4=13 f(G) = g(G) + h(G)=7+0=7

[A,C,D]
[A,C,D,F]

Final path: A → C → D→F → G Total
cost= 7

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll. No. B30	Name: Bhatt Pranjal Deepak
Class : TE - Comps B	Batch: B2
Date of Experiment: 27/01/2025	Date of Submission:10/2/2025
Grade:	

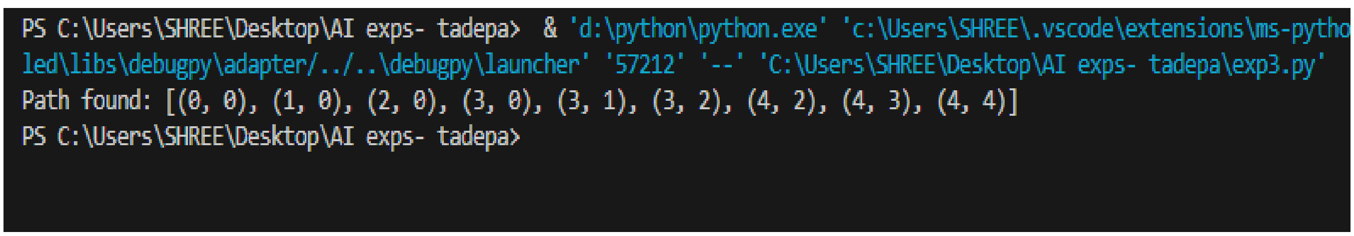
- **Software Code written by student:**

```
import heapq

class Node:
    def __init__(self, state, g, h, parent=None):
        self.state = state
        self.g = g
        self.h = h
        self.f = g + h
        self.parent = parent
```

```
    def __lt__(self, other):
        return self.f < other.f

def a_star(start, goal, heuristic, get_neighbors):
    open_list = []
    closed_list = set()
    start_node = Node(start, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)


        if current_node.state == goal:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return path[::-1]

        closed_list.add(current_node.state)


        for neighbor in get_neighbors(current_node.state):
            if neighbor in closed_list:
                continue

            g = current_node.g + 1
            h = heuristic(neighbor, goal)
            neighbor_node = Node(neighbor, g, h, current_node)

            if not any(open_node.state == neighbor and open_node.f <= neighbor_node.f for
open_node in open_list):
                heapq.heappush(open_list, neighbor_node)

    return None


def manhattan_heuristic(state, goal):
    return abs(state[0] - goal[0]) + abs(state[1] - goal[1])


def get_neighbors(state):
    x, y = state
    neighbors = []


    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dx, dy in directions:
        new_state = (x + dx, y + dy)
```

```
        if 0 <= new_state[0] < 5 and 0 <= new_state[1] < 5:
            neighbors.append(new_state)

    return neighbors


start = (0, 0)
goal = (4, 4)


path = a_star(start, goal, manhattan_heuristic, get_neighbors)


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

- ### Input and Output:



```
PS C:\Users\SHREE\Desktop\AI exps- tadepa>  & 'd:\python\python.exe' 'c:\Users\SHREE\.vscode\extensions\ms-pytho
led\libs\debugpy\adapter/../..\debugpy\launcher' '57212' '--' 'C:\Users\SHREE\Desktop\AI exps- tadepa\exp3.py'
Path found: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (4, 2), (4, 3), (4, 4)]
PS C:\Users\SHREE\Desktop\AI exps- tadepa>
```

- ### Observations and learning:

The A* search algorithm efficiently finds the optimal path by combining actual costs (g) and heuristic estimates (h).

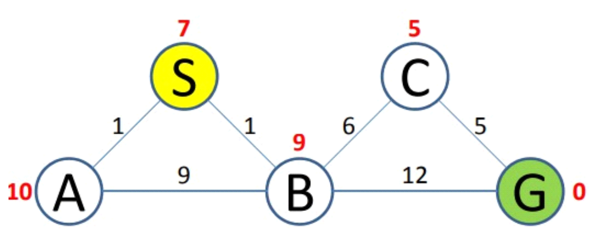
Correct heuristic values significantly impact the algorithm's performance and ensure the shortest path is found.

- ### Conclusion:

A* search successfully finds the optimal path from A to G with a total cost of 17, demonstrating its effectiveness in graph traversal with admissible heuristics.

- ### Question of Curiosity

Q1) Apply A* algorithm in the following example and find the cost



The shortest path from S to G is S-A-G and cost is 11.

Q2) What is the other name of informed search strategy?

- Simple search
- heuristic search
- online search
- none of the above