## **AIM**

To implement the **A\* Search Algorithm** in Python to find the shortest path from a source to a destination in a weighted graph using a heuristic function.

## **PROCEDURE**

1. Represent the graph using an **adjacency dictionary**.
2. Define a **heuristic** value for each node.
3. Initialize:  
   * A **priority queue** (based on f = g + h)
   * g\_score for path cost from the start
   * came\_from to reconstruct the path
4. While the queue is not empty:  
   * Pick the node with the **lowest f-score**
   * For each neighbor:  
     + If a **better path** is found, update scores and queue
5. Reconstruct the path once the goal is reached.

## **CODE :**

from queue import PriorityQueue

def a\_star(start, goal, graph, heuristic):

open\_list = PriorityQueue()

open\_list.put((0, start))

came\_from = {}

g\_score = {node: float('inf') for node in graph}

g\_score[start] = 0

while not open\_list.empty():

\_, current = open\_list.get()

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for neighbor in graph[current]:

temp\_g = g\_score[current] + graph[current][neighbor]

if temp\_g < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = temp\_g

f = temp\_g + heuristic[neighbor]

open\_list.put((f, neighbor))

return None

# Sample weighted graph

graph = {

'A': {'B': 1, 'C': 4},

'B': {'D': 1, 'E': 3},

'C': {'F': 1},

'D': {'G': 5},

'E': {'G': 2},

'F': {'E': 1},

'G': {}

}

# Heuristic values (estimated cost to goal 'G')

heuristic = {

'A': 7,

'B': 6,

'C': 4,

'D': 2,

'E': 1,

'F': 3,

'G': 0

}

start\_node = 'A'

goal\_node = 'G'

path = a\_star(start\_node, goal\_node, graph, heuristic)

# Output

print("Shortest Path from", start\_node, "to", goal\_node, "using A\*:")

if path:

print(" → ".join(path))

else:

print("No path found.")

## **OUTPUT :**

Shortest Path from A to G using A\*:

A → B → E → G

## **CONCLUSION**

* The A\* algorithm correctly computes the shortest path by combining actual cost (g) and heuristic (h) to prioritize nodes.
* This implementation is **flexible**, easy to visualize and test, and is commonly used in:  
  + GPS route finding
  + Game AI navigation
  + Robot motion planning