**4.Implement A\* algorithm**

**Aim:** To Implement A\* algorithm

**Code:**

import heapq

def astar(graph, start, goal, heuristic):

open\_set = [(0 + heuristic[start], start)]

heapq.heapify(open\_set)

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

g\_score[start] = 0

came\_from = {}

while open\_set:

current\_f, current\_node = heapq.heappop(open\_set)

if current\_node == goal:

path = []

while current\_node in came\_from:

path.append(current\_node)

current\_node = came\_from[current\_node]

path.append(start)

path.reverse()

return path, g\_score[goal]

if current\_node in graph:

for neighbor, weight in graph[current\_node].items():

tentative\_g\_score = g\_score[current\_node] + weight

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

came\_from[neighbor] = current\_node

g\_score[neighbor] = tentative\_g\_score

heapq.heappush(open\_set, (g\_score[neighbor] + heuristic[neighbor], neighbor))

return None, None

sample\_graph = {

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

'B': {'D': 2, 'E': 5},

'C': {'F': 2},

'D': {'G': 3},

'E': {'G': 1},

'F': {'G': 1},

'G': {}

}

sample\_heuristic = {

'A': 7,

'B': 6,

'C': 6,

'D': 4,

'E': 2,

'F': 2,

'G': 0

}

start\_node = 'A'

goal\_node = 'G'

path, cost = astar(sample\_graph, start\_node, goal\_node, sample\_heuristic)

if path:

print(f"Shortest path from {start\_node} to {goal\_node}: {path}")

print(f"Cost: {cost}")

else:

print(f"No path found from {start\_node} to {goal\_node}")

**OUT PUT:**

**sample\_graph = {**

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

**'B': {'D': 2, 'E': 5},**

**'C': {'F': 2},**

**'D': {'G': 3},**

**'E': {'G': 1},**

**'F': {'G': 1},**

**'G': {}**

**}**