**Aim:**

To implement the **A\* Algorithm** in Python.

**CODE:**

**import heapq**

**class Node:**

**def \_\_init\_\_(self, name, g=0, h=0):**

**self.name = name\**

**self.g = g**

**self.h = h**

**self.f = g + h**

**self.parent = None**

**def \_\_lt\_\_(self, other):**

**return self.f < other.f**

**def a\_star(graph, start, goal, heuristics):**

**open\_list = []**

**closed\_list = set()**

**start\_node = Node(start, g=0, h=heuristics[start])**

**heapq.heappush(open\_list, start\_node)**

**while open\_list:**

**current\_node = heapq.heappop(open\_list)**

**if current\_node.name == goal:**

**path = []**

**while current\_node:**

**path.append(current\_node.name)**

**current\_node = current\_node.parent**

**return path[::-1]**

**closed\_list.add(current\_node.name)**

**for neighbor, cost in graph[current\_node.name]:**

**if neighbor in closed\_list:**

**continue**

**g\_cost = current\_node.g + cost**

**h\_cost = heuristics[neighbor]**

**neighbor\_node = Node(neighbor, g=g\_cost, h=h\_cost)**

**neighbor\_node.parent = current\_node**

**if not any(neighbor\_node.name == n.name and neighbor\_node.f >= n.f for n in open\_list):**

**heapq.heappush(open\_list, neighbor\_node)**

**return None**

**graph = {**

**'A': [('B', 1), ('C', 4)],**

**'B': [('A', 1), ('C', 2), ('D', 5)],**

**'C': [('A', 4), ('B', 2), ('D', 1)],**

**'D': [('B', 5), ('C', 1)],**

**}**

**heuristics = {**

**'A': 7,**

**'B': 6,**

**'C': 2,**

**'D': 0,**

**}**

**start\_node = 'A'**

**goal\_node = 'D'**

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

**print("Path from {} to {}:".format(start\_node, goal\_node))**

**print(" -> ".join(path) if path else "No path found")**

**RESULT:**

**The A\* program was successfully implemented.**