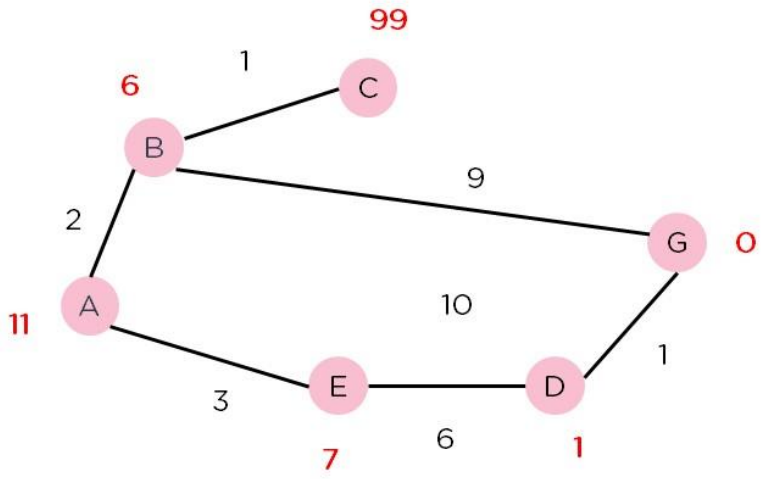
EX.No : 05 **DATE :**
REG.NO: 220701065

A* SEARCH ALGORITHM

A heuristic algorithm sacrifices optimality, with precision and accuracy for speed, to solve problems faster and more efficiently.
All graphs have different nodes or points which the algorithm has to take, to reach the final node. The paths between these nodes all have a numerical value, which is considered as the weight of the path. The total of all paths transverse gives you the cost of that route.
Initially, the Algorithm calculates the cost to all its immediate neighboring nodes,n, and chooses the one incurring the least cost. This process repeats until no new nodes can be chosen and all paths have been traversed. Then, you should consider the best path among them. If f(n) represents the final cost, then it can be denoted as : f(n) = g(n) + h(n), where : g(n) = cost of traversing from one node to another. This will vary from node to node h(n) = heuristic approximation of the node's value. This is not a real value but an approximation cost.



CODE:

```python
from queue import PriorityQueue
def a_star_search(graph, heuristic, start, goal):

    open_list = PriorityQueue()
    open_list.put((0, start))  # f(n), node

    came_from = {}

    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    f_score = {node: float('inf') for node in graph}
    f_score[start] = heuristic[start]

    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], g_score[goal]

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

    return None, float('inf')

graph = {
    'A': {'B': 1, 'C': 3},
    'B': {'D': 1, 'E': 4},
    'C': {'F': 6},
    'D': {'G': 5},
    'E': {'G': 1},
    'F': {'G': 2},
    'G': {}
}

heuristic = {
    'A': 6,
    'B': 4,
    'C': 4,
    'D': 2,
    'E': 1,
    'F': 2,
    'G': 0
}

start_node = 'A'
goal_node = 'G'
path, cost = a_star_search(graph, heuristic, start_node, goal_node)

print(f"Shortest path: {path}")
print(f"Total cost: {cost}")
```

OUTPUT:

```
Shortest path: ['A', 'B', 'E', 'G']
Total cost: 6
```


RESULT:

Thus, the A* search program has been implemented successfully.