EX.NO: 04
DATE:

A* Search

AIM:-

Create a program to solve A* search algorithm using python.

ALGORITHM:-

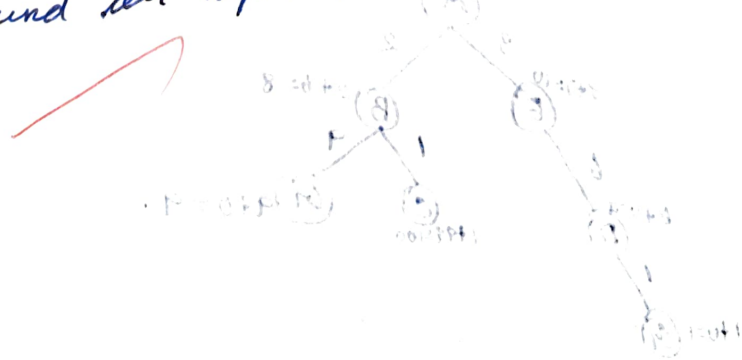
1. Initialization:
Add the start node to the open_list with g(start) = 0 and f(start) g + h (where h is heuristic).

2. Node selection:
Select the node from open_list with the list with the lowest f(n). If the node is the goal, reconstruct and return the path.

3. Neighbour Exploration:-
for each neighbour of the current node, calculate the tentative g(n). (cost to reach the neighbour).

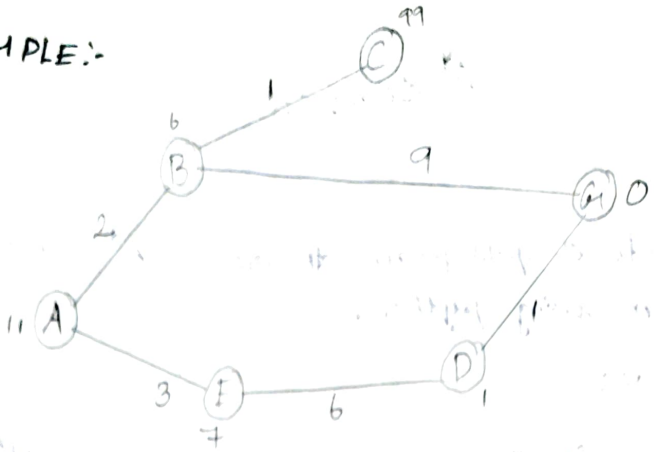
4. update cost:-
If a better path to a neighbour is found update g(n) and f(n) and set the current node as the parent of this neighbour.

5. Repeat or Terminus:-
continue the process until the goal is found or open_list becomes empty.



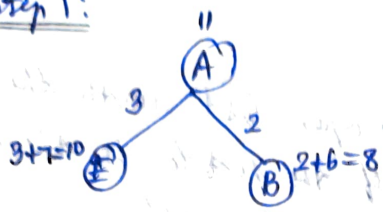
EXAMPLE:-

1.

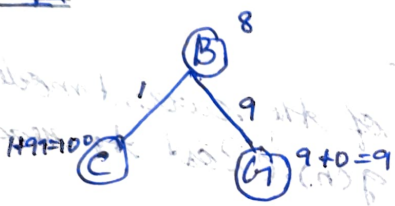


Soln:

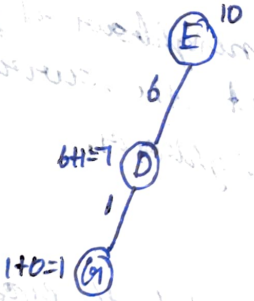
Step 1:



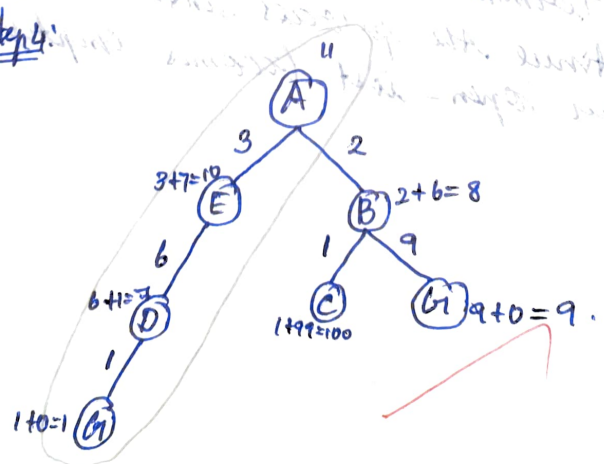
Step 2:



Step 3:



Step 4:



A → E → D → G1.

CODE:-

```
graph = {
  'A': [('B',2), ('E', 3)],
  'B': [('A',2), ('C', 1)],
  'C': [('B', 1), ('G', 9)],
  'D': [('E', 6), ('G', 1)],
  'E': [('A', 3), ('D', 6)],
  'G': [('C', 9), ('D', 1)] }

heuristic = {'A': 11, 'B': 6, 'C': 99, 'D': 1, 'E': 7, 'G': 0}

def a_star(graph, start, goal):
    from Queue import PriorityQueue
    open_list = PriorityQueue()
    open_list.put((0, start))
    g_costs = {start: 0}
    came_from = {start: None}
    f_costs = {start: heuristic[start]}
    while not open_list.empty():
        current = open_list.get()[1]
        if current == goal:
            return reconstruct_path(came_from, current)
        for neighbour, cost in graph[current]:
            tentative_g_cost = g_costs[current] + cost
            if neighbour not in g_cost or tentative_g_cost < g_cost:
                f_cost = tentative_g_cost + heuristic[neighbour]
                f_costs[neighbour] = f_cost
                open_list.put((f_cost, neighbour))
                came_from[neighbour] = current
    return None
def reconstruct_path(came_from, current):
    path = []
    while current:
        path.append(current)
        current = came_from[current]
    path.reverse()
```

```
return path
start_node = 'A'
goal_node = 'G'
path = a_star(egraph, start_node, goal_node)
print('shortest path:', path)
```

Output:

```
shortest path: ['A', 'E', 'D', 'G']
```

Result:-

Thus A* problem is executed and output is verified successfully.