

**A\* SEARCH ALGORITHM****AIM:**

To implement a A\* heuristic algorithm to find the least-cost path in a graph using node weights and heuristic approximations for efficient traversal.

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.

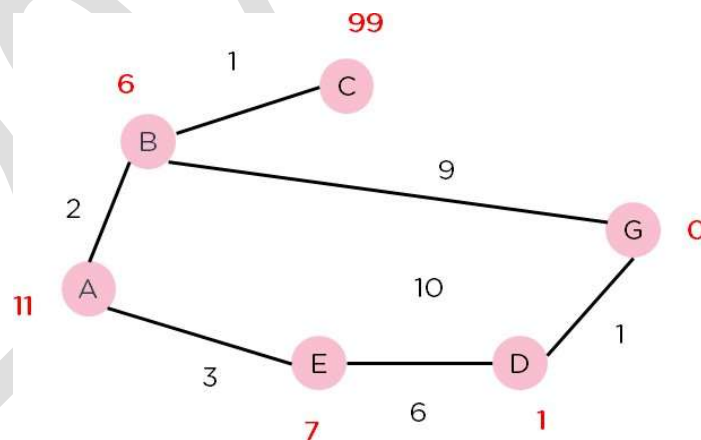
Initially, the Algorithm calculates the cost to all its immediate neighboring nodes,  $n$ , and chooses the one incurring the least cost. This process repeats until no new nodes can be chosen and all paths have been traversed. Then, you should consider the best path among them. If  $f(n)$  represents 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.

**PROGRAM:**

```

import heapq
class Node:
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent

```

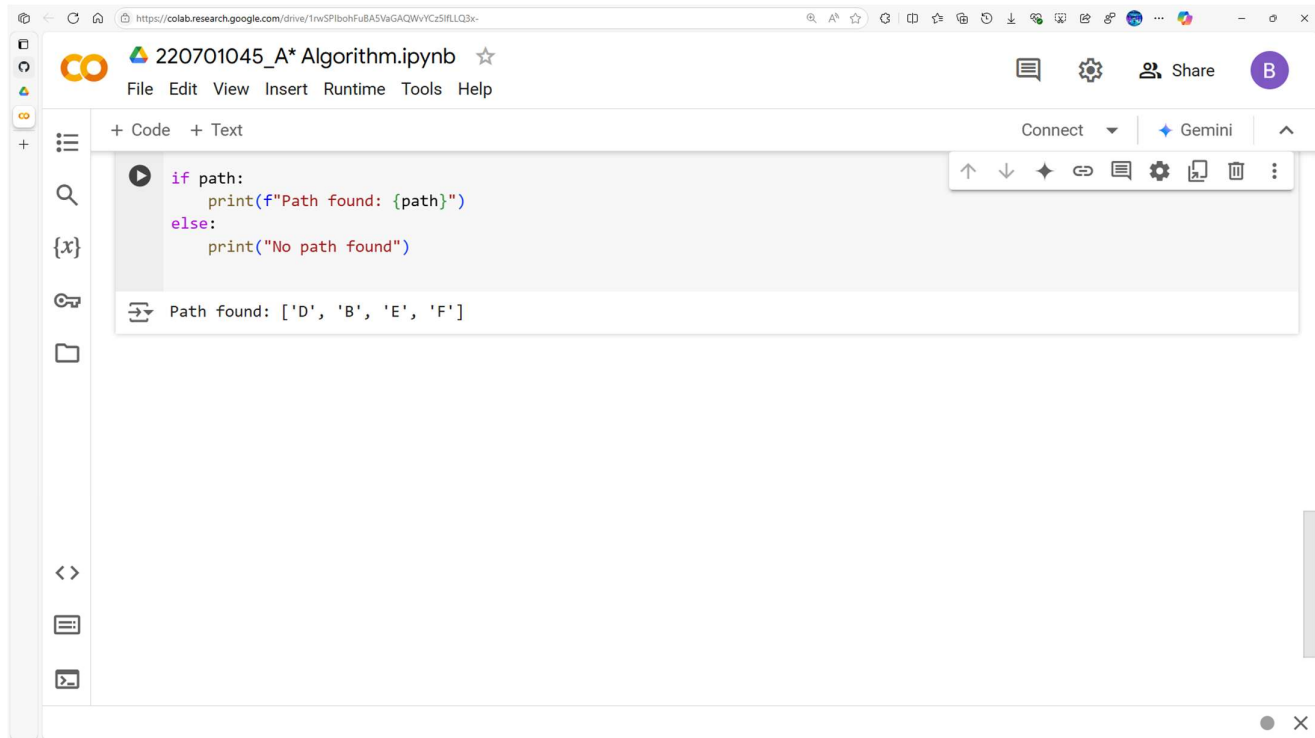
```

    self.g = g
    self.h = h
    self.f = g + h
    def __lt__(self, other):
        return self.f < other.f
def a_star(graph, start, goal, h_values):
    open_list = []
    heapq.heappush(open_list, Node(start, None, 0, h_values[start]))
    closed_list = set()
    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.get(current_node.name, []):
            if neighbor in closed_list:
                continue
            g_new = current_node.g + cost
            h_new = h_values[neighbor]
            f_new = g_new + h_new
            neighbor_node = Node(neighbor, current_node, g_new, h_new)
            heapq.heappush(open_list, neighbor_node)
    return None
graph = {}
h_values = {}
print("Enter heuristic values for each node. Type 'nil' to stop.")
while True:
    node = input("Enter node name (or 'nil' to finish): ")
    if node.lower() == 'nil':
        break
    h_value = int(input(f"Enter heuristic value for {node}: "))
    h_values[node] = h_value
print("Enter edges and their costs. Type 'nil' to stop.")
while True:

```

```
node1 = input("Enter the start node (or 'nil' to finish): ")
if node1.lower() == 'nil':
    break
node2 = input("Enter the end node: ")
cost = int(input(f"Enter the cost from {node1} to {node2}: "))
if node1 not in graph:
    graph[node1] = []
graph[node1].append((node2, cost))
start_node = input("Enter the start node: ")
goal_node = input("Enter the goal node: ")
path = a_star(graph, start_node, goal_node, h_values)
if path:
    print(f"Path found: {path}")
else:
    print("No path found")
```

## OUTPUT:



The screenshot shows a Google Colab notebook interface. The browser address bar at the top displays the URL: <https://colab.research.google.com/drive/1nwSPibohFu8ASVaGAQWwYcZ5IFLLQ3x->. The notebook title is "220701045\_A\* Algorithm.ipynb". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". On the left sidebar, there are icons for file management and a search bar. The main code editor area contains the following Python code:

```
if path:
    print(f"Path found: {path}")
else:
    print("No path found")
```

Below the code editor, the output is displayed: "Path found: ['D', 'B', 'E', 'F']". The output is shown in a white box with a small icon on the left. The right sidebar of the notebook shows a "Connect" button and a "Gemini" chat interface.

## RESULT:

Thus , the heuristic algorithm successfully identifies an efficient, least-cost path in the graph by evaluating node weights and heuristic estimates.