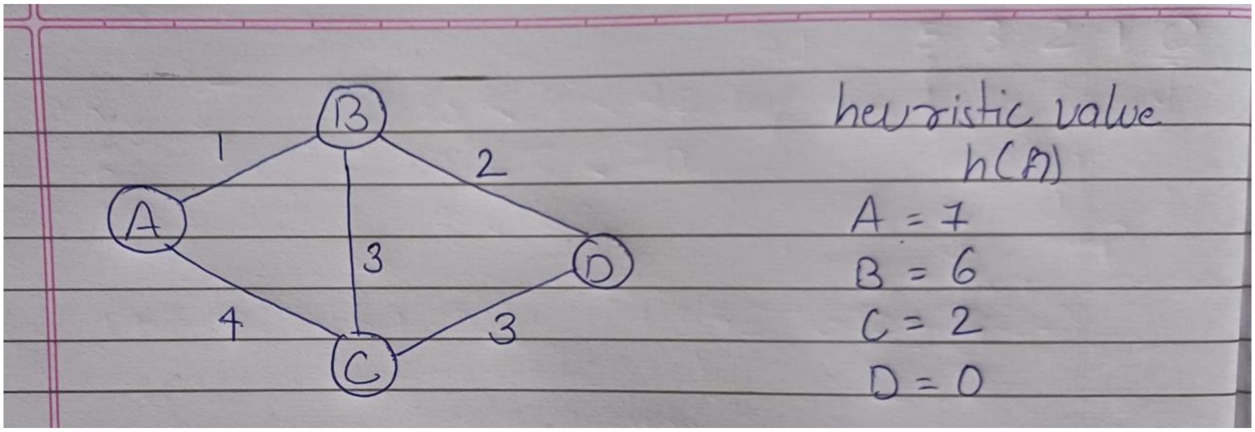
Name: Arpit Bhavesh Damani

Class & Batch: TY09-A

Roll no: 12

Aim: Implement A* informed search algorithm to reach the goal state.

Graph:



Source Code:

```
import heapq

def a_star(graph, start, goal, heuristic):
    open_list = []
    heapq.heappush(open_list, (0 + heuristic[start], start))

    g_costs = {start: 0}
    parent = {start: None}
    open_set = {start}

    while open_list:
        _, current_node = heapq.heappop(open_list)
        open_set.remove(current_node)

        if current_node == goal:
            path = []
            while current_node is not None:
                path.append(current_node)
                current_node = parent[current_node]
```

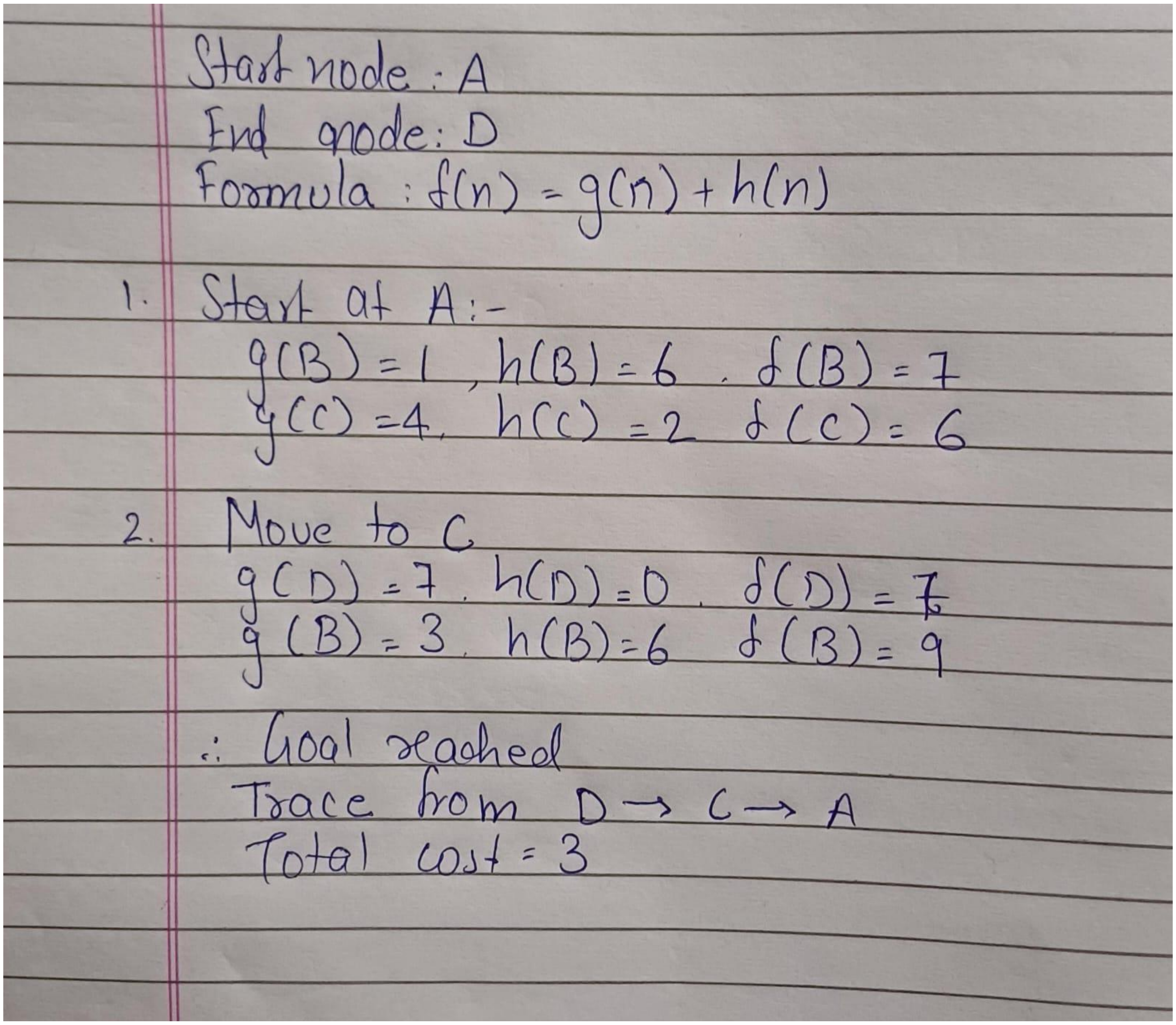
```python
            return path[::-1], g_costs[goal]

        for neighbor, cost in graph[current_node].items():
            tentative_g_cost = g_costs[current_node] + cost

            if neighbor not in g_costs or tentative_g_cost < g_costs[neighbor]:
                g_costs[neighbor] = tentative_g_cost
                f_cost = tentative_g_cost + heuristic.get(neighbor, 0)
                if neighbor not in open_set:
                    heapq.heappush(open_list, (f_cost, neighbor))
                    open_set.add(neighbor)
                parent[neighbor] = current_node

    return None, float('inf')

graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 3, 'D': 2},
    'C': {'A': 4, 'B': 3, 'D': 3},
    'D': {'B': 2, 'C': 3}
}

heuristic = {
    'A': 7,
    'B': 6,
    'C': 2,
    'D': 0
}

start = 'A'
goal = 'D'
path, cost = a_star(graph, start, goal, heuristic)

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

Output:

Path found: ['A', 'B', 'D']

Minimum cost: 3

Example solved in Notebook:


Start node : A
End node: D
Formula : $f(n) = g(n) + h(n)$

1. Start at A:-
$g(B) = 1$, $h(B) = 6$, $f(B) = 7$
$g(C) = 4$, $h(C) = 2$ $f(C) = 6$

2. Move to C
$g(D) = 7$, $h(D) = 0$, $f(D) = 7$
$g(B) = 3$, $h(B) = 6$ $f(B) = 9$

∴ Goal reached
Trace from D → C → A
Total cost = 3

Review Questions:

Q1) What are the components of the A* evaluation function f(n), and how do they contribute to the search process?

Ans: The A* (A-star) search algorithm evaluates each node *nn* in the search space using an evaluation function:

$$f(n)=g(n)+h(n)f(n) = g(n) + h(n)$$

Components of the A* Evaluation Function:

1. ***g(n)* - Cost from Start to Current Node:**
    a. Represents the actual cost incurred to reach node *n* from the start node.
    b. Tracks the path cost using the accumulated values of edge weights or step costs.
    c. Ensures that A* finds the lowest-cost path by considering the cost spent so far.
2. ***h(n)*- Heuristic Estimate to Goal:**
    a. Provides an estimate of the minimum cost to reach the goal from node *n*.
    b. Helps in guiding the search by prioritizing nodes that appear to be closer to the goal.
    c. A well-designed heuristic can significantly improve search efficiency.
    d. The heuristic should be **admissible** (never overestimates the true cost) and **consistent** (obeys the triangle inequality).

Contribution to the Search Process:

- **Balancing Exploration and Exploitation:**
    - *g(n)* ensures thorough exploration of the paths with minimal cost incurred.
    - *h(n)* directs the search towards the goal, helping to reduce unnecessary exploration.
- **Optimality and Completeness:**
    - If *h(n)* is admissible and consistent, A* guarantees finding the optimal path.
    - The algorithm is complete, meaning it will find a solution if one exists.
- **Performance:**

- A good heuristic reduces the number of nodes expanded, improving time and space efficiency.
- Poor heuristics can degrade A* to uninformed search methods like Dijkstra's algorithm.

Q2. How does A* search differs from BFS and DFS in terms of its approach to finding the goal?

Ans: Key Differences Between A, BFS, and DFS*

Feature	**A* Search**	**BFS**	**DFS**
Strategy	Uses cost + heuristic *f(n)=g(n)+h(n)f(n) = g(n) + h(n)*	Explores level by level	Explores depth-first
Optimality	Yes (if heuristic is good)	Yes (for equal cost paths)	No
Completeness	Yes	Yes	No (may get stuck in infinite paths)
Time Complexity	*O(b^d)*, faster with a good heuristic	*O(bd)O(b^d)*	*O(bd)O(b^d)*
Space Complexity	*O(b^d)* (high)	*O(bd)O(b^d)* (high)	*O(d)O(d)* (low)
Best For	Shortest path in weighted graphs (e.g., navigation, AI)	Unweighted graphs, shortest path	Exploring deep structures, backtracking problems