# Name: Soham Mukherjee

# Regd. No: 2301292114

# Section: A

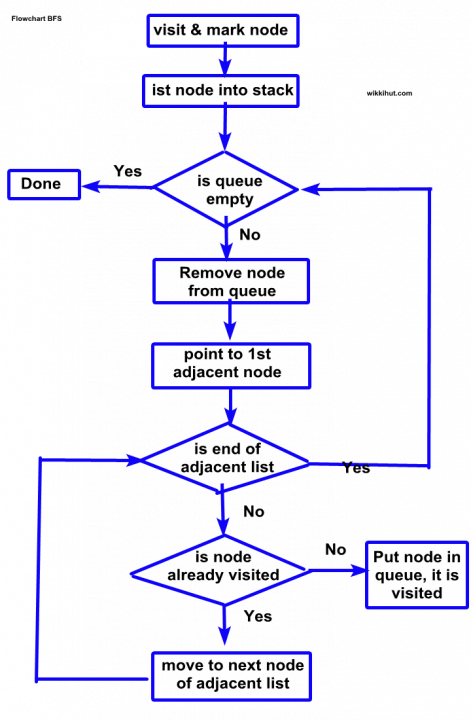
Uninformed Searches:

1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Breadth First Search (BFS) :

Algorithm:

1. **Initialize:**
   * Create a queue and add the starting node to it.
   * Mark the starting node as visited to prevent reprocessing.
2. **Traversal:**
   * While the queue is not empty:
     1. Dequeue a node from the queue.
     2. Process the node (e.g., print its value or record it).
     3. Enqueue all unvisited neighbours of the node and mark them as visited.

Flowchart:



Code:

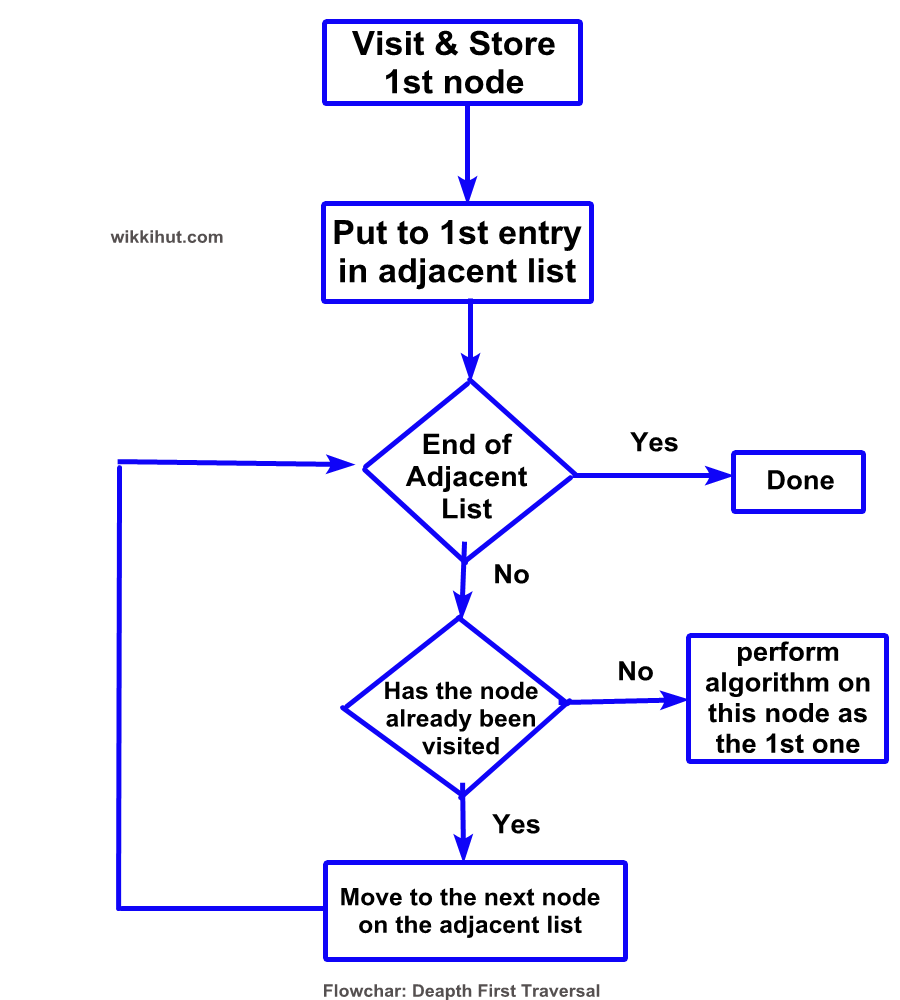


2. Depth First Search (DFS):

Algorithm:

1. **Initialization:**
   * Start from a given node.
   * Mark it as visited.
2. **Exploration:**
   * Recursively visit all unvisited neighbours of the current node.
   * Backtrack when all neighbours are visited.

Flowchart:



Code:



Uninformed Searches:

1. Best First Search

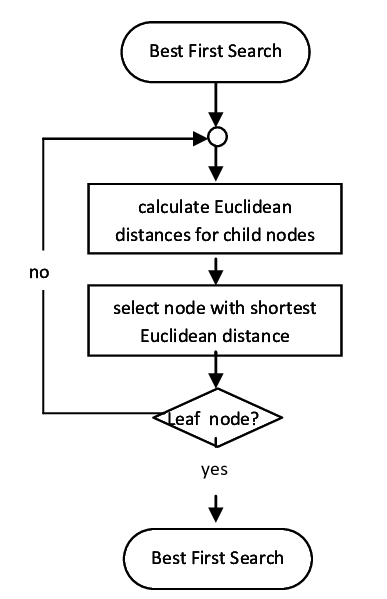
2. A-star Search (A\*)

1. Best First Search:

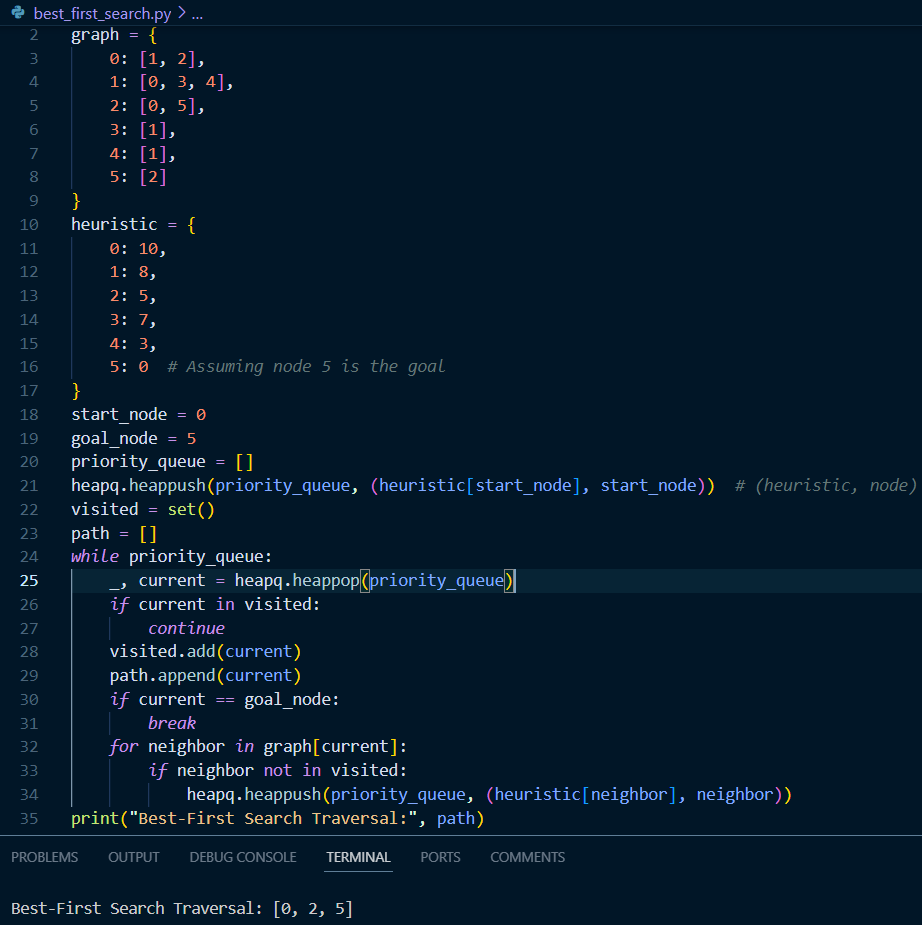
Algorithm:

1. **Initialize:**
   * Start with the initial node and add it to the priority queue.
2. **Loop:**
   * Remove the node with the smallest h(n)h(n)h(n) from the priority queue.
   * If the node is the goal, terminate and return the solution.
   * Otherwise, expand the node and add its neighbours to the priority queue, evaluating each neighbour using the heuristic.
3. **Repeat:**
   * Continue until the goal is found or the priority queue is empty (failure).

Flowchart:



Code:



2. A-star Search (A\*):

Algorithm:

1. **Initialize:**

* Place the start node in a priority queue (commonly called the open list). Initialize its g(n)=0g(n) = 0g(n)=0.

2. **Expand Nodes:**

* At each step, select the node nnn from the priority queue with the lowest f(n)f(n)f(n) value.
* If nnn is the goal, terminate and reconstruct the path.
* Otherwise, move nnn to the closed list (visited nodes) and evaluate its neighbours.

3. **Update Costs:**

* For each neighbour of nnn:
  + Compute g(neighbour)g(neighbour)g(neighbour).
  + If the neighbour is not in the open list or has a lower ggg-cost, update its g(n)g(n)g(n) and calculate f(n)f(n)f(n).
  + Add the neighbour to the priority queue if not already present.

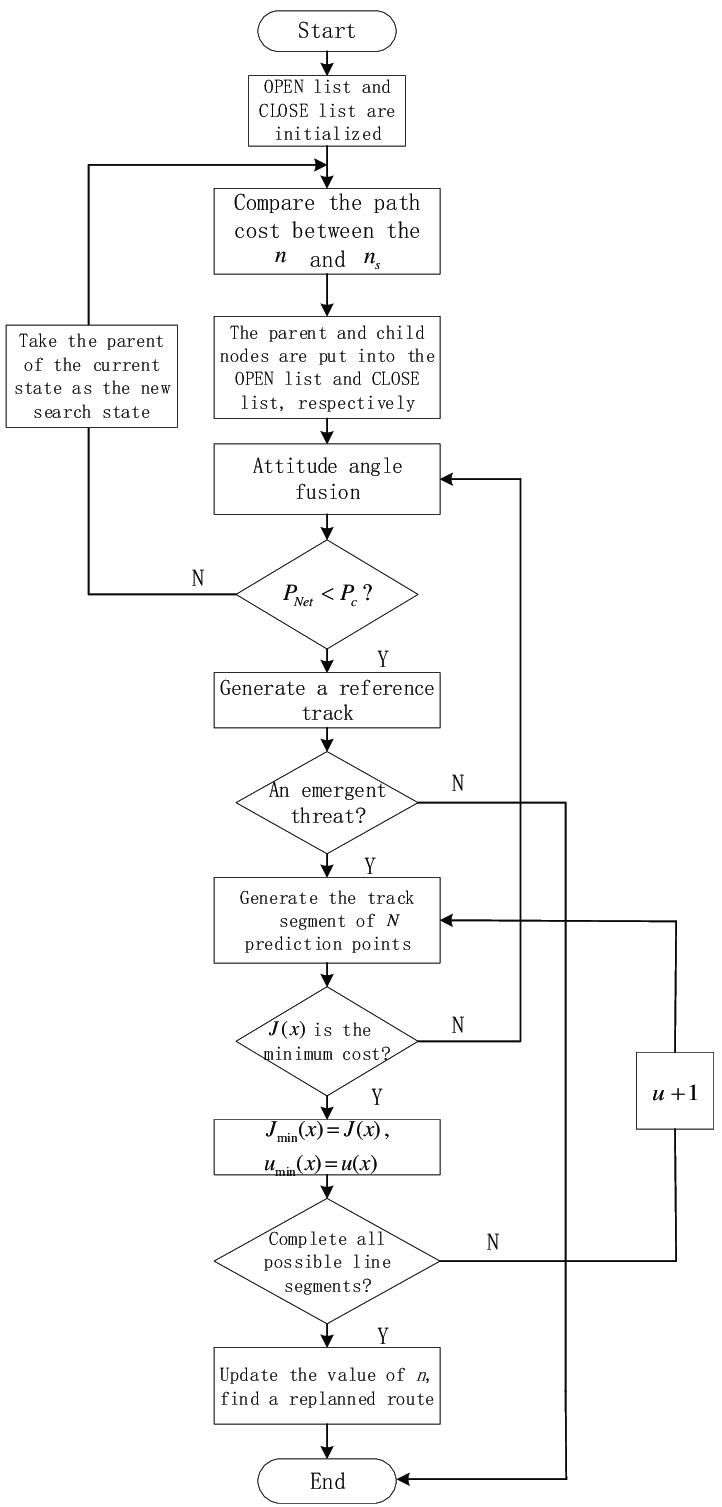
4. **Repeat:**

* Continue until the goal is found or the priority queue is empty (failure).

Features:

1. **Optimal**: Finds the shortest path if h(n)h(n)h(n) is admissible (never overestimates the actual cost).
2. **Complete**: Always finds a solution if one exists.
3. **Efficient**: Reduces the number of nodes explored by guiding the search using h(n)h(n)h(n).

Flowchart:



Code:

