Aim: Write a program to implement on 8 puzzle

a)Best first Search

b)A* algorithm

Theory:

The 8-puzzle problem involves arranging tiles numbered from 1 to 8 on a 3x3 grid, with one blank space that allows tiles to slide into the blank space. The goal is to rearrange the tiles from a given initial state into the goal state, where the tiles are ordered from 1 to 8, with the blank space at the bottom-right corner. The challenge is to determine the most efficient way to solve this puzzle, which is where search algorithms like Best-First Search and A* Algorithm come into play.

1. Best-First Search Algorithm

Best-First Search is a search algorithm that uses a heuristic to decide which node to explore next. In the case of the 8-puzzle, the heuristic evaluates the "closeness" of a given state to the goal state. The main idea is to explore nodes that appear to be closest to the goal based on some heuristic function, often referred to as **h(n)**, where **n** is the current node (state).

Heuristic Function for the 8-Puzzle:

The heuristic in Best-First Search for the 8-puzzle can be one of the following:

- **Misplaced Tiles**: The number of tiles that are not in their goal position. For example, in a goal state, the tile numbered 1 should be in the top-left corner, tile 2 in the top-middle, and so on. If any tile is not in its correct position, it is counted as a misplaced tile.

  Heuristic:
  h(n)=Number of misplaced tilesh(n) = \text{Number of misplaced tiles}
- **Manhattan Distance**: The sum of the distances each tile must travel to reach its goal position. Each tile can move horizontally or vertically, but not diagonally. The Manhattan distance for a tile is the sum of the horizontal and vertical steps required to move it to its correct position.

  Heuristic:
  h(n)=∑(abs(x1−x2)+abs(y1−y2))h(n) = \sum \left(\text{abs}(x_1 - x_2) + \text{abs}(y_1 - y_2) \right)
  where (x1,y1)(x_1, y_1) are the current coordinates of the tile, and (x2,y2)(x_2, y_2) are its goal coordinates.

Best-First Search Steps:

1. **Initial State**: Start from the initial configuration of the puzzle.
2. **Heuristic Calculation**: For each possible move, calculate the heuristic function (e.g., Manhattan distance or misplaced tiles) for the resulting state.
3. **Node Selection**: Select the node (state) with the lowest heuristic value (most promising move) to explore next.
4. **Expand the Node**: Generate all possible moves (states) from the current node by sliding a tile into the blank space.
5. **Repeat**: Continue expanding nodes based on their heuristic values until the goal state is reached.
6. **Termination**: The algorithm terminates once the goal state is reached or if there are no more nodes to explore.

Characteristics of Best-First Search:

- **Greedy Approach**: The algorithm selects the state that seems closest to the goal, but it does not consider the path taken to reach the state.
- **No Optimality Guarantee**: Since Best-First Search only focuses on the heuristic, it may not find the optimal (shortest) solution. It may get stuck in local minima or explore inefficient paths.
- **Efficiency**: Best-First Search can sometimes explore fewer nodes than exhaustive search algorithms, but it is not guaranteed to be efficient in all cases.

2. A* Algorithm

The A* algorithm is an improvement over Best-First Search, as it combines the benefits of **both the cost to reach a node** (denoted **g(n)**) and the **heuristic estimate of the cost to reach the goal** (denoted **h(n)**). This allows A* to consider both the effort spent to reach the current state and the estimated future effort to reach the goal. The function used to decide which node to expand next is called **f(n)**, and it is defined as:

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

where:

- **g(n)**: The actual cost to reach the current state from the start (this is often the number of moves made so far).
- **h(n)**: The heuristic estimate of the cost to reach the goal from the current state.

The A* algorithm uses this combined evaluation to choose the most promising node to expand, balancing between exploring the current path and anticipating the future steps to reach the goal.

Heuristic for A*:

For the 8-puzzle, A* typically uses one of the following heuristics:

- **Misplaced Tiles**: As in Best-First Search, A* can use the number of misplaced tiles as the heuristic.
- **Manhattan Distance**: A* can also use the Manhattan distance as the heuristic.

A* Algorithm Steps:

1. **Initial State**: Start with the initial configuration of the puzzle, and initialize the cost g(n)=0g(n) = 0 for the initial node.
2. **Heuristic Calculation**: For each possible move, calculate the heuristic function h(n)h(n) for the resulting state.
3. **Node Evaluation**: For each possible move, calculate the total evaluation function f(n)=g(n)+h(n)f(n) = g(n) + h(n). This combines the path cost and the estimated future cost.
4. **Node Selection**: Select the node with the lowest f(n)f(n) value to expand next.
5. **Expand the Node**: Generate all possible moves (states) from the current node by sliding a tile into the blank space, and update g(n)g(n) for each new state.
6. **Repeat**: Continue expanding nodes based on their f(n)f(n) values until the goal state is reached.
7. **Termination**: The algorithm terminates once the goal state is reached, or if there are no more nodes to explore.

Characteristics of A* Algorithm:

- **Optimality**: If the heuristic function h(n)h(n) is admissible (i.e., it does not overestimate the cost to reach the goal), A* is guaranteed to find the optimal solution.
- **Efficiency**: A* is generally more efficient than Best-First Search because it combines both the path cost and the heuristic to avoid inefficient paths.
- **Complete**: A* is complete, meaning it will always find a solution if one exists, provided there is no error in the implementation or constraints.

Differences Between Best-First Search and A* Algorithm

Feature	Best-First Search	A* Algorithm
Evaluation Function	Uses only the heuristic function h(n)h(n)	Uses both the cost to reach the node g(n)g(n) and the heuristic h(n)h(n)
Optimality	Does not guarantee the optimal solution	Guarantees the optimal solution if the heuristic is admissible
Efficiency	May get stuck in local minima and explore inefficient paths	More efficient as it considers both the cost and the heuristic
Exploration	May explore irrelevant paths, focusing only on the heuristic	Explores paths more judiciously by considering both cost and heuristic

Conclusion

- **Best-First Search** is useful for problems like the 8-puzzle where you want a fast solution without necessarily guaranteeing optimality. It is more focused on how "close" a state is to the goal but may not always find the shortest solution.

- *A Algorithm** is superior for solving the 8-puzzle problem optimally. By considering both the current path and future possibilities, it ensures that the solution found is the shortest possible and is guaranteed to find the goal if the heuristic is admissible.

Both algorithms are applicable to the 8-puzzle, but A* is generally more reliable in finding an optimal and efficient solution.