Babaahmadi-narges-610398102

States: Different combination of numbers on 3.3 board. States are also nodes of the search tree.
Actions: moves of the empty block in 3.3 board.(up, down, left, right)
Goal state: The goal state is the state in which all numbers are in order.
(1,2,3,4,5,6,7,8,0)
Heuristic function: Manhattan distance of each block to its correct position.(Summed up)

This is an implementation of BFS, DFS, UCS, IDS, and A* for solving the 8-puzzle game. The game consists of a 3x3 grid that is filled with tiles numbered 1 through 8, and a blank space represented by the number 0. The player can move the tiles by sliding them into the blank space. The goal is to rearrange the tiles to a specific final configuration.

The code defines functions for performing the various moves (push_up, push_down, push_left, and push_right) and functions for generating the children of a given node in the search tree (get_children).

BFS function:
This function implements a Breadth First Search (BFS) algorithm to find the solution of a problem represented as a tree. The problem is to find the path from a root node to a goal node in the tree. The algorithm makes use of a queue to keep track of the nodes to be visited and a marked set to keep track of the nodes that have been visited so that they are not revisited.
In each iteration, the algorithm pops a node from the queue, gets its children using the **get_children** function and adds them to the queue if they haven't been marked. The algorithm continues this process until the goal node is found or the time limit of 120 seconds is reached.
If the goal node is found, the algorithm traces back the path from the goal node to the root node and **returns the path**, **the depth** of the goal node and the **time taken** to find the solution. If the goal node is **not found**, the function returns **-1**.

DFS function:
This function implements a Depth First Search (DFS) algorithm to find a solution of a problem represented as a tree. It uses a stack to keep track of the nodes to be visited and a **marked** set to keep track of the visited nodes. The algorithm visits nodes in the order of their height in the tree. It **returns the path** from the root node to the goal node, **the depth** of the goal node, and the time taken to find the solution if the goal node is found. If the goal node is not found, the algorithm returns -1. The code also has a time limit of 120 seconds, after which the algorithm terminates and returns -1 if the solution has not been found.

Uni-cost function:
The function **uni_cost** implements the uniform cost search algorithm. The algorithm takes the initial state of the board (root) and the goal state as inputs. It initializes a queue, adds the root node to the queue, and creates a set of marked boards to keep track of already visited boards. Then, while the current board is not equal to the goal state, it gets the children of the current node, adds the children to the queue, sorts the queue based on the depth of the nodes, and removes the node with the lowest depth from the queue to make it the current node. The algorithm continues this process until the current node is equal to the goal state or the time taken exceeds 120 seconds. If the current node is equal to the goal state, the function finds the path from the root to the current node and **returns the path**, **the depth**, and the time taken. If the current node is not equal to the goal state, the function returns -1.

Ids function:
The function **ids** implements the Iterative Deepening Search algorithm to solve a puzzle represented as a two-dimensional array, root.The algorithm has a default depth limit of 20 but can be changed to a different value through the **depth_lim** parameter. The function **returns a tuple** that consists of the **path from the root** node to the goal node, **the number of moves** taken to reach the goal, and the time taken to solve the puzzle. The function uses a stack to keep track of nodes to be explored, and it adds the children of a node to the stack only if the depth of the node is less than the depth limit. The function also keeps track of the nodes that have already been explored using a marked set to avoid exploring the same node again.

A_star:
The function uses the concept of the **A*** search algorithm.The algorithm works by first adding the initial state (root) to a priority queue (queue). The algorithm then repeatedly selects the state with the lowest estimated cost from the queue, checks if it is the goal state, and generates its children. The generated children are added to the queue if they haven't been marked yet, and the cost of reaching the state is estimated by summing the manhattan distance between the board and the goal state. The algorithm continues until the goal state is found or it takes more than 120 seconds to complete. The function **returns a path** to the goal state, **the depth** of the solution, and the time it took to find the solution.

Comparing algorithms:
Due to the statistics we got from the code, Uniform cost and bfs were giving us similar result and the other algorithms from best to worst, respectively, are:
A* , BFS, Uniform Cost, IDS, DFS
(Memory and time usage are in out.txt)