DSAL II Mini-Project Write-up

Group Members: Poorva Bhalerao, Asawari Badkundri, Chaitanyasuma Jain, Aaliyah Ahmed

1) Problem Statement

Implement solution for eight puzzle solver using breadth first search and evaluate efficiency.

2) Details about Designing (Classes and Functions)

- **Class Node:**
  -Gives attributes of each node
  -Extends State class

  Functions in Node:
  -LinkedList<Node> getChildren(Node parent, State goalState) -
  Returns all children of node parent.
  After 1 swapping is done in parent, we obtain it's child.
  After all possible single swaps are made, we get all the children.

- **Enum Move:**
  Has all possible moves (up,down,left,right) given state can perform for one swap.

- **Class State:**
  -Creates state of the board.
  -Calculates manhattan distance (returns the heuristic cost (here, Manhattan distance) between current state and goal state)

  Functions in State:
  -State createState(State state, Move move)
  -public int manhattanDist(State curr, State goal)
  -boolean statesMatch(State testState, State goalState)

  [Note: We have not used the heuristic cost in the algorithm but it is required for implementation of other algorithms using artificial intelligence to solve the n-puzzle.]

- **<u>Class eightPuzzle:</u>**
  -Main function
  -Takes input state, output state
  -gives number of moves from initial state to goal state.

  -void welcomeUser()-
  -void printInstructions()-
  -void inputState() and void finalState()
    - called twice from main, one for initial and one for final state
  -void printBoard(int board[][])-
  -void printSolution() - This function interprets numerical instructions of the move to make, to its verbal counterpart to be displayed to the screen. Here, path is the solution path consisting of a list of nodes from the root to the goal.

4) Data Structures/Algorithms Used

Graph
Graph G=(V,E) is a non-linear data structure comprising of a finite set of vertices V (also called nodes or points) and a finite set of edges (also called links) E. A graph, in computer science, is an abstract data type to implement the directed or undirected graph.

We have used the graph data structure, dynamically creating nodes denoting the state of the board, its heuristic distance and other details.

Graph Traversal
Traversal is the process of visiting the nodes of a graph by travelling along its edges. There are many graph traversal of graph search algorithms available depending on their order of node traversal, some of which are better than the other. Examples: Breadth-First Search, Depth-First Search.

We have used breadth first search algorithm to evaluate if state at each node matches the goal state of the puzzle.

Breadth-First Search Algorithm
It traverses a finite graph by visiting all vertices at the same level/depth before proceeding to the next levels. It is implemented using a queue that stores all sibling vertices. BFS can be used to find the shortest path from one vertex to the other.

Here, we have used BFS to arrive at an optimum solution for the 8-puzzle.

5) Operations performed

class eightPuzzle

1. void welcomeUser()- This displays the '8-Puzzle Solver' ASCII art on the screen

2. void printInstructions()- This displays the input instructions for the user

3. void inputState() and void finalState()
   - gets the initial and goal state of the board from the user (non-repeated integer between 0-9). Here, 0 depicts the blank space
   - called twice from main, once for initial and then for final state

4. void printBoard(int board[][])- gives the standard ouput display

5. void printSolution()- this function interprets numerical instructions of the move to make, to its verbal counterpart to be displayed to the screen. Here, path is the solution path consisting of a list of nodes from the root to the goal

class Move

- enum Move{}- defines the possible moves

class State

1. State createState(State state, Move move)- returns the resultant state if move is a valid move, checks if the coordinates are valid and make changes in the new board configuration to show the new move made

2. int manhattanDist(State curr, State goal)- returns the heuristic cost (here, Manhattan distance) between current state and goal state

3. boolean statesMatch(State testState, State goalState)

- testState - state to be matched
- goalState - state to be matched with
- returns true if states match, else returns false

class Node

1. LinkedList<Node> getChildren(Node parent, State goalState)-

- attempts to create states for each move, and add to the list of children if true