A* Search Algorithm

CPSC 3620

Rohan Sharma ID# 001232930

Marcus Caparini ID# 001240882

Project Overview:

This project uses the A* search algorithm to solve the board game sliding tiles in a 3X3 matrix. The project takes in an initial and goal configuration and shows the user the most efficient move to solve the problem by displaying the move set and the amount of move needed to get to the goal configuration.

Board Tile class Implementation:

The board Tile class has three different private member variables. Firstly, there is an Int moves which hold the number of moves it took from the initial board to reach its current configuration. The second member variable is a string config which represents the 3X3 board Tile configuration. The last member variable is the string movesFromStart which represents the move that led to the current configuration from the initial configuration.

BoardTile uses two different constructors. One with no parameters and the other with a const string&. The constructor with no parameters initializes the following member variables to Moves = 0, config = "0000000" and movesFromStart = "". We use this constructor to implement BoardTile objects that are used within other parts of the implementation. The constructor that takes in a const string& also initializes the Moves = 0, movesFromStart = "" but initializes the config = const string&. We use this secondary constructor to represent a board Tile object which is the initial configuration for the problem. The run time of theses two constructors are O (1).

The member function nextConfigs () has no parameters and return a vector of board Tile objects that shows the next possible move sets to reach the goal configurations. The runtime of this function is O (N) where N is the number of elements within the matrix.

The member function int numMoves () is a getter function that returns the number of moves it took from the initial board to reach the current configuration. The runtime of this function is O (1) as it only return the number of moves.

The member function int Manhattan_Distance (const BoardTile& goalconfig) is a function that takes a boardTile object called goalconfig and return an int which is the value of the Manhattan distance.

The member function string GetPath () is a function that takes returns the movesFromStart member variable. The runtime of this function is O (1).

The member function void setMovesFromStart (string i) is a function that takes sets the movesFromStart member variable to equal i (movesFromStart = i). The runtime of this function is O (1).

The member function void setNumMoves (int i) is a function that takes sets the Moves member variable to equal i (Moves= i). The runtime of this function is O (1).

The bool operator == (BoardTile& l, BoardTile& r) is a assignment operator function that overloads the (==) operators. This is used to compare the strings in two different BoardTile objects to see if they are equalling each other. This function is specifically uses as a stopping condition in the Solve Puzzle implementation. The runtime of this function is O (1).

Class Sliding Solver Implementation:

The Sliding Slover class has three private member variables. The first private variable is a BoardTile object called Ending. This variable purpose to hold the end configuration of the BoardTile so it can be used to solve the puzzle in other functions. The second member variable is a vector<BoardTile> previousPlaces. This variable is used to hold previous BoardTile objects to ensure we don't have any duplication. The last variable is a priority queue <BoardTile, vector<BoardTile>, Compare> tile Queue which is a representation of a minheap of BoardTile objects.

The Sliding Slover classes three private functions.

The first private function is a void makeSorted () which uses the sort algorithm to sort the member variable is a vector<BoardTile> previousPlaces to help ensure we don't have any duplicate objects. The runtime of this function is O (N) where N is the number of elements in the vector.

The second private function is int hasBeen (vector<BoardTile>, BoardTile) which uses binary search to check if the BoardTile object is a duplicate of an element in the member variable is a vector<BoardTile> previousPlaces. The runtime of this function is O (log n) where N is the number of elements in the vector.

The last private function is void SolvePuzzle (BoardTile) which continues until the Ending member variable equals the BoardTile Parameter. We initialize a vector which hold the next configurations where it gets the next configs from the best configuration, we found so far according to the distance it taken so far. If its not a duplicate it pushed into the queue and throw the previous BoardTile into the PreviousPlaces vector and sort the previousPlaces vector and we take the next config off the queue, and this continues until the ending == Answer. Then we set the ending to equal the answer. The runtime of this function is O (n log n) where N is the elements in the vector.

The member function bool small (BoardTile &a, BoardTile &b) is used to check if the boardTile object a is smaller then the BoardTile object B. if this is true then it will return true otherwise it will return false. The runtime of this function is O (1).

The member constructor SlidingSolver (string startConfig, string endConfig) initializes two BoardTile objects a & b to represent the startConfig and the endConfig and initializes the ending to also equal BoardTile b. Then we call the function Solve Puzzle using the BoardTile a as a parameter and solve the problem. The runtime of this function is O (1).

The member function BoardTile getGud () is used to return the Ending member variable. The runtime of this function is O (1).

Class Compare Implementation:

This class has one function called bool operator () (BoardTile l, BoardTile r) which compares the distance between two different board tiles to see if the BoardTile l is greater than BoardTile r. This has the runtime of O (1).

Improvements to my current solution

The improvements that I would make to my current solution is that I would create template classes so that we could solve the puzzles of any type. I would also like to get rid of the comparator class and add a comparator function within the sliding solver class so we can get rid of the whole Compare class. I would also like to add another function that prints the steps to reach the solution. This would lead the user to understand the program more and creates a better interface for the user. I would also like to make another function that checks to see if there are multiple solutions and shows the user that there are more than one solution and prints all of them.