RUSH HOUR SOLVER

Documentation

Hamza Benhamida
301418508

Introduction:

The goal of this project is to create an algorithm which will solve rush hour puzzles. I will implement this algorithm into a method called “solveFromFile(String input, String output)”, input string represents the file of the board as a text file and output string represents the file the method will write the moves needed to finish the board.

1st Implementation:

Characteristics:

- Using BFS for graph traversals
- Using class RushHour to represent one state of the board. Using the car class to store the x, y direction,length and name into separate fields.
- Generating children of a state: for each car it will get all the possible different moves within a distance of 1.
- Calculating the hashcode during the creation of the RushHour object

Data Structures & Algorithms:

When creating the data structure of the board, first I found which methods will be needed for the board then I decided which properties will be necessary and their structure.

For the RushHour class: I used a 2d char array to represent a board, also used a string to store the move, an int to store the hashcode and a hashmap to store the cars and their corresponding names. I also used a car class which implemented the x,y,direction and length fields. I also implemented a method called children() which would return the possible neighbours of the current state.

For the Solver class: I used a BFS algorithm to traverse the graph and find the finished state. I also built a method which would recover the path from the initial state. Addiotionally, I decided to write helper methods to get moves into a string format and one method to write the solution into the file.

Outcomes/Findings:

After testing the program it seems to not work for most cases and it is taking a long time. We can also notice that it lacks memory for most of the boards.

2nd Implementation:

Characteristics:

- Using class RushHour to represent one state of the board. Using the car class to store the x, y direction,length and name one single int variable.

- Generating children of a state: for each car it will get all the possible different moves within all possible distances.
- Improving the HashCode.

Data Structures & Algorithms:

I kept the overall structure of the first implementation, however I made some adjustments in the children() method, as well as the HashCode considering all the cars and their position in the board to create the hash. Furthermore, I changed the car class structure by using only one variable to store x,y,direction and length to use less memory.

Outcomes/Findings:

After testing the program it seems to work for 1 case only and it is still taking quite some time.

3nd Implementation:

Characteristics:
- Using DFS instead of BFS

Data Structures & Algorithms:

Decided to change approach and use the DFS algorithm to traverse the graph and find the solution.

Outcomes/Findings:

It works really fast now!!!

Conclusion:

It seems that I could improve the class RushHour and represent the car class as byte to make the algorithm faster. Moreover, I could use aStar algorithm to optimize it even more.