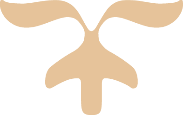


INTELLIGENT SYSTEMS: SUBTASK 2

Comparison of Implemented Data Structures





Index:

1. Requirements of the Application. page 3
2. Object Oriented Structure. page 3
3. Libraries and Used Data Structures. page 4
4. Solution of Problems. page 4
5. User Manual. page 5

**1. Requirements of the Application**

With the result of the previous task, we will get a M x N-1 matrix, with the black one identificated. The objective is to explore the different possibilities with a search tree. The following tree is made of nodes, that represent each of the states that the puzzle can take. The information given by the node is parent, state, cost, action, depth and value (a random integer).

As a frontier, the elements must be represented in an ordered list, based on the value that correspond to each one.

Finally, it is necessary to compare two data structures, using both of them in the creation and store of them. The most important scopes to compare are the times needed to insert in the frontier, and the maximum number of nodes that can be saved.

**2. Object Oriented Structure**

In this subtask, we started to introduce Object Oriented Programming in order to have reusability, fiability and a code easier to modify and understand.

These are the following Object Classes that exist in the Subtask 2:

* State: Has all the attributes that defines any state of the puzzle. It is used in the main method every time a new move is made.

* nodeTree: This class can be created with the object State. Another way to create one of them is with a normal constructor (filling each of its attributes).

* ImgProcessor: Contains the creation and comparison of the images of subtask 1, so we can use it in an Object Oriented way in other classes.

**3. Libraries and Used Data Structures**

In addition to every library used in the last subtask,

* java.util.Random: The random generator to create the value at each node (an integer between 100 and 1000.
* Priority Queue: This structure can take the nodes as they are created, ordering them, depending on their value, after the iteration in which the come inside the structure.
* Linked List: This list saves the nodes in the order when they are created. To order them by value, it is necessary to go across the list in each iteration, comparing their values to order them in a correct way.

**4. Solution of Problems**

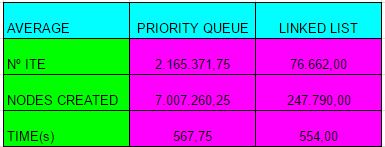
* Selecting two data structures: We decided to compare between a priority queue and a linked list. The current solution has two classes with the same implementation, the only change is how the frontier is implemented and which data structure is being used. That allows us to run them without changing any code.

* Creation of correct values: With the creation of random values at the constructor of node Tree, every sibling with the same parent had the same value. In order to resolve this problem, we create a random value at every iteration where a node tree object is created.

* The method to know the position of the black space: Before, we had a method that is called after each move, to know the new position of the black point. In order to change that, we defined two integer into State class. Those integers are changed as the black point is moved, so we already know the new position.

**5. Comparison between structures**

This is the comparison between both data structures. We take into account the number of iterations and nodes created at a certain time, when Eclipse tool can’t afford more memory for the structure. The Priority Queue couldn’t afford more memory after 567 seconds, while the Linked List kept running in the program. To compare them, we have established a similar time, comparing their other characteristics.



These are the conclusions that we have taken from the study of both structures and the comparison that we did.

* By the number of iterations of each structure, we can define that Priority Queue works better in the same conditions and time than a Linked List, almost 3,5 times better.
* The number of nodes created by the Priority Queue is much greater than in the Linked List. This is due to in a similar time, the Priority Queue create them much faster and easier, as the Linked List has to order all the nodes each time a new one is created.
* In conclusion, the Priority Queue is much easier to apply, can work longer and creating more nodes, so a Linked List looks out-dated in comparison.