INFO0027 : PROGRAMMING TECHNIQUES

Project 1 - Scrabble Puzzle Solver

LOWETTE Quentin
CHARRON Raphaël

April 24, 2020

1 General presentation

This project is based on the principle of Scrabble. The goal is to obtain the word or words that generate the highest score in a set of words at our disposal.

Indeed, we have at our disposal three files containing letters, words and scores. From these files we aim to return a word, or a combination of words that allows us to have a maximum score. To do this, we must first be able to link the file containing the scores to the file containing the letters and then apply this to the file containing the words.

The files containing the scores and the letters are linked in a rather obvious way but the file containing the words is more complex to link because we will probably have to browse several times the other two files to get the score of each word and repeat these operations for each word.

So the goal of this project is to create an algorithm that takes text these files entries to give us the word that has the highest score.

2 Beginning of work on the algorithms

When we started working on the project, we first set about understanding what part of the course it was based on. It seemed to us fairly quickly that the project was mainly based on the aspects of algorithmic sorting that we had seen in the different theoretical and practical courses.

Moreover, we wondered if the use of a tree would not be something judicious in order to be able to classify and have access quickly to the best word(s).

The use of a tree to quickly get the best was indeed the best solution we had considered. So we turned to this type of implementation to determine the word with the highest score.

3 Presentation of our solving method

We therefore thought of using the sorting methods we had seen during the theoretical course. Indeed, we first use a `Quick Sort` algorithm in our solving method.

Indeed, we first created with the `scoreLetter.c` program a table containing the letters and their respective scores. Then we sort this table. In this way we obtain a ranking of the letters according to their weight (i.e. according to their score). This will then be very useful in the further development of our solving method.

Following this, we have created a binary tree of type `Ternary Search Tree` in order to be able to place the words on branches. The nodes on our tree are composed in the classic way for this type of tree. Each node of the tree is composed of a letter constituting the word and a boolean that tells us if the word is finished or not. We also have the classical structure of this type of tree which is composed of 3 pointers which return to the following elements according to the given key value.

Either the key is lower than all the elements attached to the node and we move to the left, or the key is equal to the value of the node and we move to the middle or in the last case if the value is higher than the value of the node we move to the right of the tree. This way we are able to move through the tree in an optimal and fast way.

So once our tree has been set up, we then browse the table containing the letters that have been previously sorted and we search by taking the first letter of the table in our tree, then the second letter of the table and this until we get a word. We know that we get a word from the tree because the boolean indicates whether the word is finished or not.

This method of solving the problem therefore allows us to quickly access the word with the highest score. Because indeed, we simply move in a sorted table and in a tree which is therefore inexpensive from a complexity point of view. And finally, we quickly notice that the longest part of our algorithm is the fact of having to sort the table containing the letters and their scores because the larger the table, the longer it will take to sort.

Whereas the part that contains the unique tree is just a simple search on the word that has the highest score based on our sorted value table.

4 Complexity analysis

From the point of view of the complexity of our algorithm, we perform several types of operations in our method of solving the problem. The two main operations are the sorting of an array and the traversal of a tree.

First, we sort an array containing letters and their respective scores. To carry out this sorting, we use an algorithm of type `Quick sort`, we know from the theoretical course that the complexity of this sorting is in $\mathcal{O}(n\ log\ n)$ where n represents the number of elements of our table, that is to say in our case the number of letters that we have at our disposal and which are thus present in the table to be sorted.

In a second step, when we search for the word which has the maximum score, we browse our table and our tree. We know that in the case of a TST tree the complexity of the path is in $\mathcal{O}(\text{size of the key})$ as well as for the path of the sorted table the worst case for the complexity is in $\mathcal{O}(n)$. The complexity in this second part of the problem is less important because we simply have a reading of the data to perform, which therefore represents a linear complexity.

5 Difficulties encountered

Some aspects of the project topic were more difficult to achieve than others. In this section, we will present the different points that required more reflection on our part in order to resolve them.

First of all, when we first tried to make our algorithm work correctly in the basic case illustrated in the presentation of the project. Then we started

working on getting our algorithm to work correctly for the other cases described in the statement.

However, we encountered several problems when we tried to implement the cases with letters with negative scores and when the user wanted to have repeating words. Indeed, these parts were more difficult to achieve because we had to be able to make our algorithm understand that it was possible to go through our tree several times, or that the combination of several words could generate a maximum score.

6 Conclusion

To conclude this project was quite interesting because it allowed us to implement several methods that we had seen during the theoretical and practical courses. It also allowed us to confront complex algorithms. This project represented several hours of work in order to succeed in obtaining adequate results and also to present a clear and well-structured code.