INFO-0027: Programming Techniques

Project 1: Report

Goffart Maxime
180521

Joris Olivier
182113

Academic year 2020 - 2021

1 Introduction

Our implementations rely on 2 different data structures. The first implementation uses a hash table with chaining to solve the potential collisions. The second implementation uses a ternary search trie.

We decided to perform our performance study by testing our implementations with an increasing number of addresses in order to be able to analyse the differences between the implementations on graphics. For each number of addresses we were considering, we did 1000 different creations of the MAGIC A.D.T. and measured the mean time passed in the `MAGICindex`[1] and `MAGICreset` functions, and the mean amount of allocated memory.

For each of the 1000 tests, we used randomly generated 4 bytes addresses. The same batch of randomly generated addresses was used for the two different implementations.

2 Time performance

In this section, we will compare the time spent in the `MAGICindex` and `MAGICreset` functions depending on the implementation being used. We will also compare the total time spent.

2.1 MAGICindex function

For the `MAGICindex` function, we obtained the following chart:

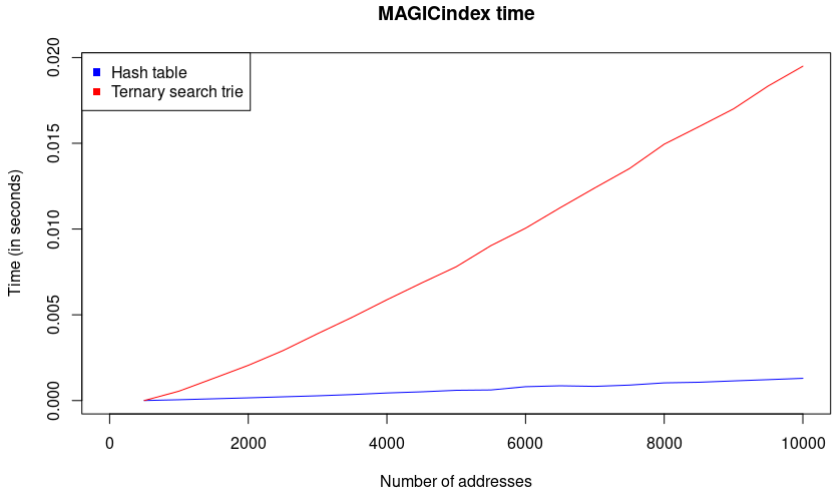

Figure 1: Mean time passed in the `MAGICindex` function depending on the number of addresses

We can easily observe that the implementation using a hash table is faster than the one using a ternary search trie. This is not an unexpected result because a search in a hash table has an average complexity of $\mathcal{O}(1)$ while a search inside a TST has an average complexity of $\mathcal{O}(log(n))$ where n is the number of elements being stored.

[1] We called MAGICindex on each address that was randomly generated.

2.2 MAGICreset function

For the `MAGICreset` function, we obtained the following chart:

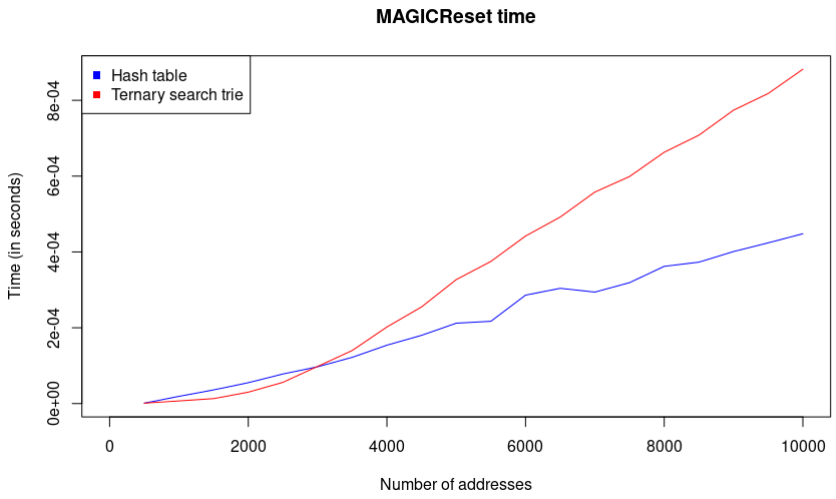

Figure 2: Mean time passed in the `MAGICreset` function depending on the number of addresses

We can observe that the TST implementation is faster when we store less than $\pm$ 3000 addresses. After that, the hash table implementation is faster. The slight advantage in time procured by the TST implementation for less than $\pm$ 3000 addresses, can be considered has insignificant due to how small the time differences are (a few tenth of a millisecond). In general, we can conclude that both implementations of the `MAGICreset` function are fast to execute because we are talking in tenths of a millisecond.

2.3 Total time

For the total time spent, we obtained the following chart:

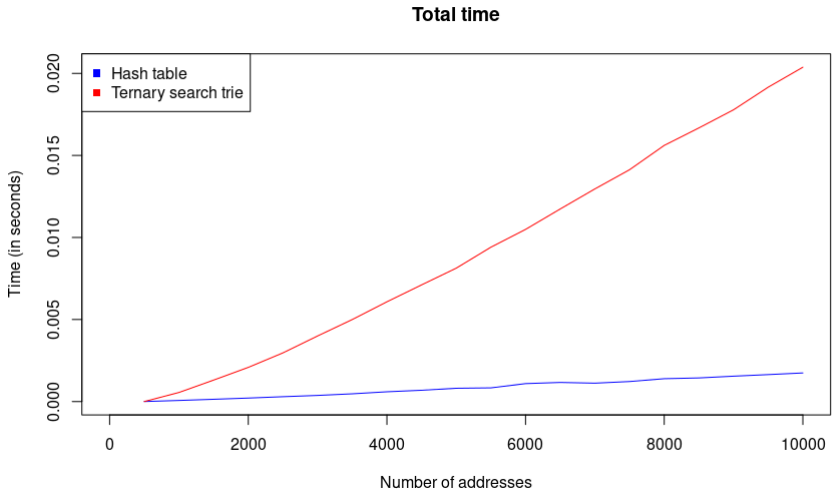

Figure 3: Mean time passed in the A.D.T. functions depending on the number of addresses

Since the time spent in the `MAGICindex` and `MAGICreset` functions was smaller for the implementation using a hash table, it is logical that the total time is smaller for the implementation using a hash table.

3 Memory perfomance

For the amount of memory being allocated, we obtained the following chart:

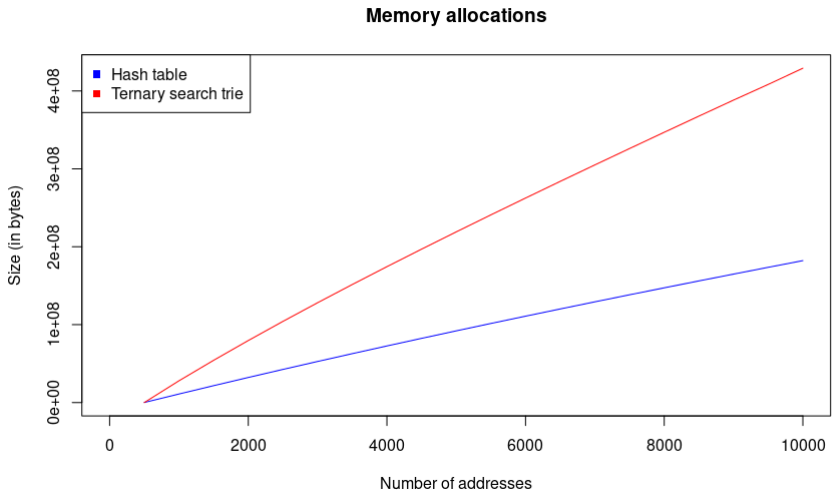

Figure 4: Mean amount of memory allocated depending on the number of addresses

We can observe that the implementation using a hash table uses less memory than the one using a TST. This can be explained by the fact that each node in the TST stores 3 pointers, one digit, and an Item. While the hash table uses only an array of linked list (the linked list being used to solve the collisions by chaining). If there is no collision at all, it uses a bit more memory than an array of pointer because we need to store the index in the auxiliary array.

4 Conclusion

In conclusion, we observed that our first implementation, which uses a hash table, has better performances than our second one, which uses a ternary search trie, in terms of time and memory.