**Benchmarking report**

**Introduction**

This benchmarking report aims to analyse and compare two data structures that can be implemented in the “Morse code interpreter app”. Performance testing will be conducted, by comparing the outcome of each option and analyse the factors that will affect each option’s performance, the optimal data structure will be suggested.

**Performance test**

The performance testing uses two node.js programs for each data structure, they are very similar in the way they work, but some parts are different due to the use of different data structures.

Both programs have a “morse table” along with the “random number”, and “check letter” function. The “random number” function can generate a four digit binary number, which means each digit can only be 0 or 1. Binary number is used because morse code only has dot/dit and dash/dah, so the binary digit can represent them respectively.

The “check letter” function, this is the vital function for the test. To use this function, we need to introduce the “morse table” first. The “morse table” is a table structure, each element contains a letter and a code. However, for each data structure, the table’s code part is different. The code is made up by four binary digits same as what is generated in the “random number” function. A full morse code table will have 36 elements, but for this benchmarking test, using 5 elements are enough. One of the morse tables has the code in a string structure which is the four digits together as one string. The other table has the code in an array structure, so the code is in a list and there are four elements in the list, each element is a binary digit. Based on the difference between the two types of morse tables, the “check letter” function has a unique way of checking for each implementation. The “check letter” function aims to check if the input matches a code corresponding to a letter in the morse table. If the input matches a letter, that letter will be returned, otherwise ‘null’ will be returned. Checking the string structure table is easier, as node.js allows comparing two string, the check is simply comparing the input string against a letter’s code in morse table and if it matches one, return that letter otherwise just return ‘null’. On the other hand, checking the array structure requires comparing one element at a time. There is a boolean variable initialized as false, then the function gets a code from the morse table and compares each element in the input string against the corresponding element in the code array’s element. Once a pair of elements do not match, the boolean variable will be set to true. After comparing all elements, if the boolean variable is left false, that code’s corresponding letter is returned otherwise ‘null’ is returned.

Using the above functions and the morse table, the performance tests can be conducted and this part is same for both implementation. Firstly, a random number list is generated using the “random number” function, this list will become the input in the test, where each element in the list represents a ‘motion data’ in the “Morse code interpreter app”. The size of the list can be changed, so different input size can be used in the test to make the test result more accurate. Using this random number list, the test loops through each element in it and use the “check letter” function to check if that element matches a letter in the morse code, if there is a match, that letter will be printed in the terminal, otherwise ‘null’ is printed. The duration of this whole process will be calculated by recording the start time before it starts and the end time when all results are printed, using the “date” function in node.js, times are all recorded in milliseconds.

By acquiring the perform times of each implement, they can be compared and the better implementation can be decided.

**Benchmarking result**

This section shows the test results using the tests in the previous section.

\*: Horizontal index shows implementation, vertical index shows input size.

\*: Main contents are the average result times in milliseconds.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **String** | **Array** | **Difference** |
| **10000** | 530 | 591 | 61 |
| **50000** | 2218 | 2584 | 366 |
| **100000** | 4435 | 5052 | 617 |
| **1000000** | 40558 | 42055 | 1497 |

The results in milliseconds shown in the table are the average results from 3 tests. Acquiring the average results is not too important in this benchmarking test, because since the inputs are generated randomly, the performance of each data structure should not have a big difference. However, to avoid the chances of getting special cases, we still use the average result, and a sample size of 3 is enough to show the normal performance of each data type.

This table of results is acquired by a test that gives both data structures the same input. Before this test the two data structures were in two separate tests where they would have the same input size but not exactly the same input, the results are close to this one, but we are presenting this table as it shows a more accurate result of each data structure’s performance.

**Conclusion**

Based on the benchmarking tests we conducted, we found that the implementation of both data structures are simple, and the group members are all familiar with them as they have been using them in previous projects.

The test results show the performance and efficiency of each data structure. In comparison, the string implementation is faster than the array implementation in all the input sizes. In addition, as the input size increases, the time difference between the two implementations grows larger and larger. Therefore, we can make the assumption that the larger the input size is, the more time we can save.

In conclusion, a string data structure is faster at performing decoding morse code than an array structure. Since the team members have no difficulties with using either data structure, the team will use a string data structure for the “Morse code interpreter app”.