Middlesex University

CST2550 Software Engineering

Management and Development

**COURSEWORK 2 REPORT**

Student ID: M00848952

Year: 2

2023

## INTRODUCTION

These projects aim to design and implement an appropriate data structure for an audio application. This data structure will give functionality to add tracks from a file, remove tracks, search and save tracks to a file.

In this report, we will discuss about the selected and designed algorithm and what are the reasons for his choice, why we chose Hash Tables with Separate Channing Collision Handling, their advantages and disadvantages, as well as the time complexity calculations of the most important functions like put, remove, and search. As part of the second section of this report, we will discuss the testing approach used for application and component testing, and you will be provided with a Testing Table that shows what was tested, the actions taken, the inputs, and the expected outputs.

As a final step, conclusions will be drawn regarding the project in general and the Designed Data Structure in particular, what are the limitations and critical reflections, and how to change this approach on similar tasks in the future.

**Report layout**

**Introduction –** Project overview…………………………………………………………………………………………….……..1

**Data Structure Design -** Description of the designed data structure……………………………………….…….2

**Time complexity** – providing time complexity for important functions……………………………….…..……3

* Inserting
* Removing
* Searching

**Testing** – used testing approach and tests table………………………………………………….……………………..…5

**Conclusion –** Conclusions about the project……….………………….……………………………………………………..6

**References**…………………………………………………………………………….………………………………………………………7

## DATA STRUCTURE DESIGN

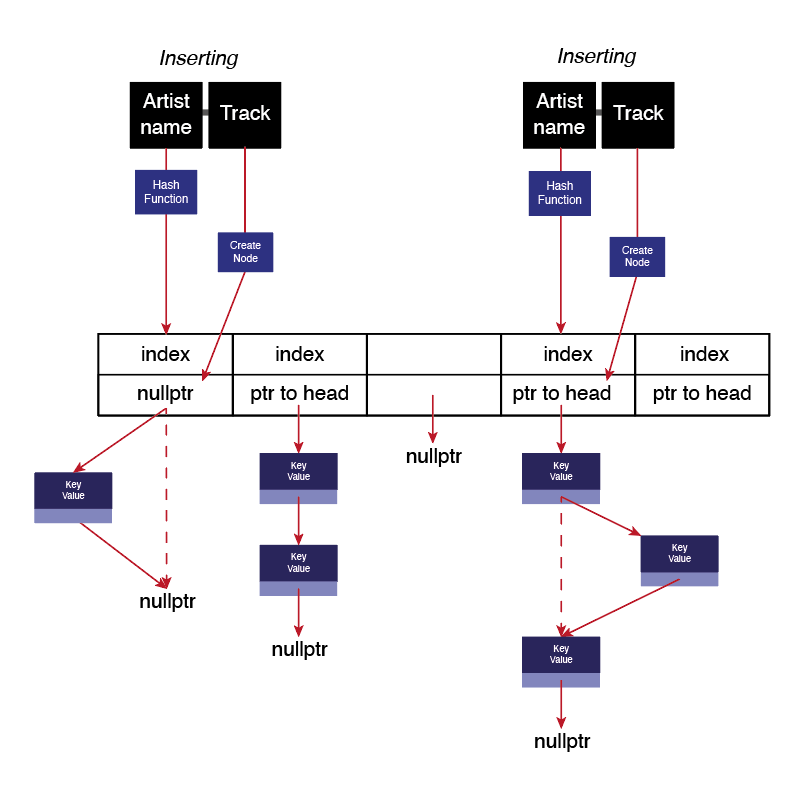
The most commonly used and important feature of an audio streaming application is searching, and in a world with billions of tracks, an appropriate data structure and algorithm are essential. For the current project, the hash table was selected due to its greatest advantage over other data structures, which is its speed to perform the most important operation in our case. Hash Table can do them all in constant time (khalilstemmler.com, 2022).

Additionally, for the current project we will use Separate Channing Collision Handling because:

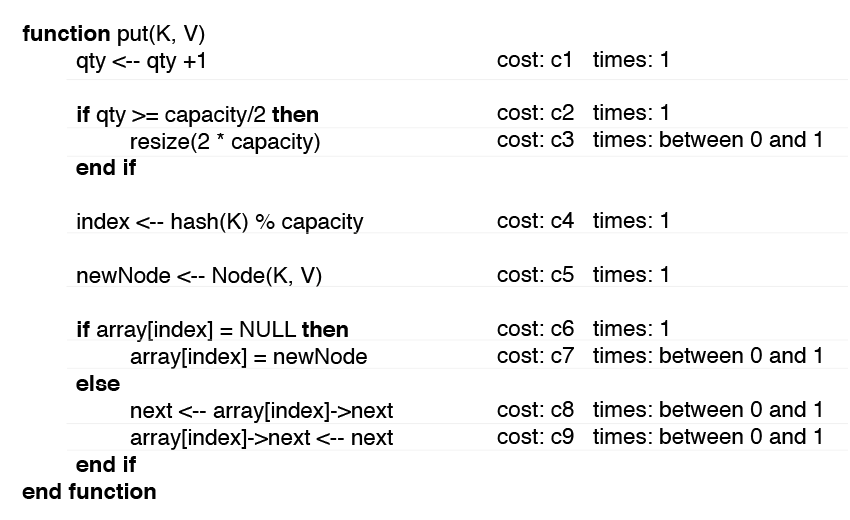
* One artist could have one ore many tracks, and we don’t know initially how many they are.
* Searching by artis/band name must give all the tracks, that will be much efficient with Separate Channing than Linear Probing. In the case of Linear Probing handling, a big number of data will create big clusters, and because we will have many collisions, it will be expensive to find all the artists' tracks.
* Searching time complexity will have smaller variations. Because all the artist tracks will be stored in the same linked list, the worst case is O(n) where n is the number of Nodes in the linked list.
* It will minimise the problem of a significant performance degradation of the table.

For track saving, a Node is created that contains a key (artist/band name), value (an object with all track information), and a pointer to the next Node.

Because the search will be performed by artist/band name, the designed hash table will take artist name as a key and converted to a hash table index using a hash function. On the index position value, if nullptr is found, it will be simply changed and pointed to the newly inserted node. Otherwise, the new node will be inserted after the first one.

Data structure design

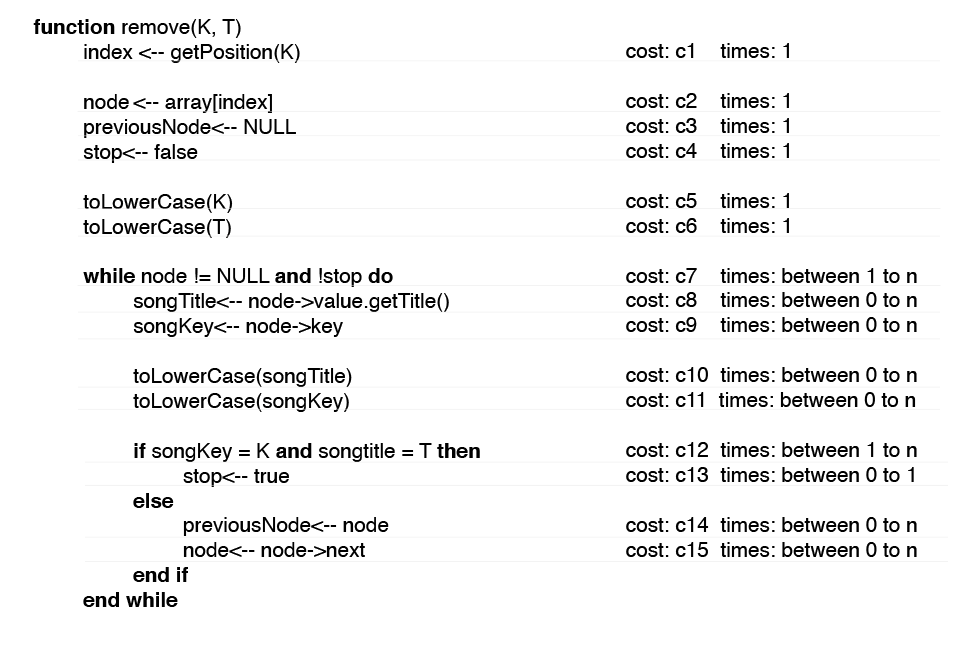
## **INSERTING**

To insert the newly created data (Track) into the Hash Table, the designed algorithm will transform the key (artist/band name) into an index, and check if a linked list exists at the selected position, or not. If the linked list doesn’t exist, it will add the node to the index position, otherwise it will add the new node after the first node.

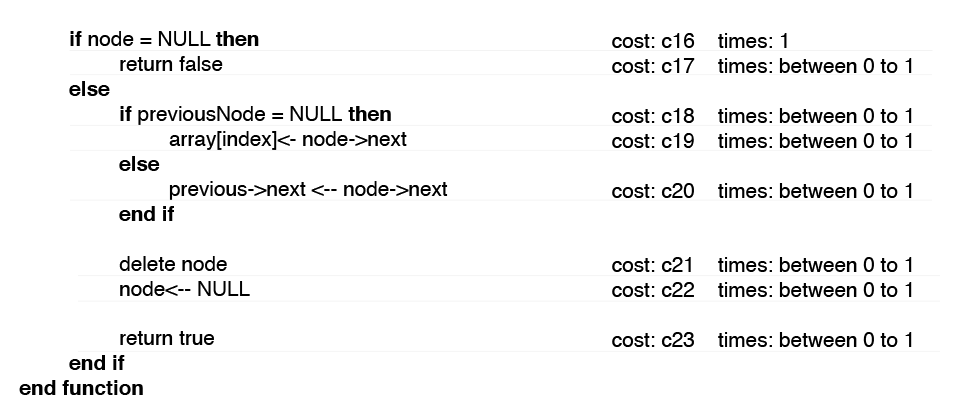
The overall Time complexity of the INSERTING function will be O(1), because it will not check the linked list for duplications.

Additionally, the function will check if it is necessary to resize the Hash Table. In case of RESIZING, his cost will be added to the INSERTION function cost.

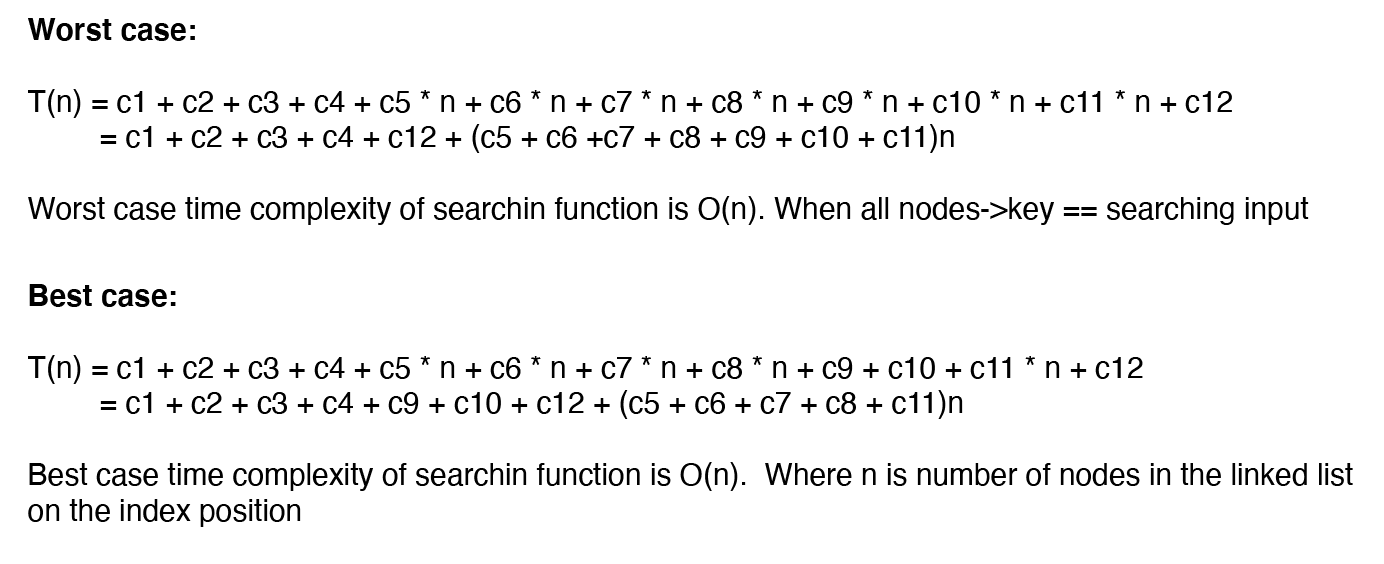
## **REMOVING**

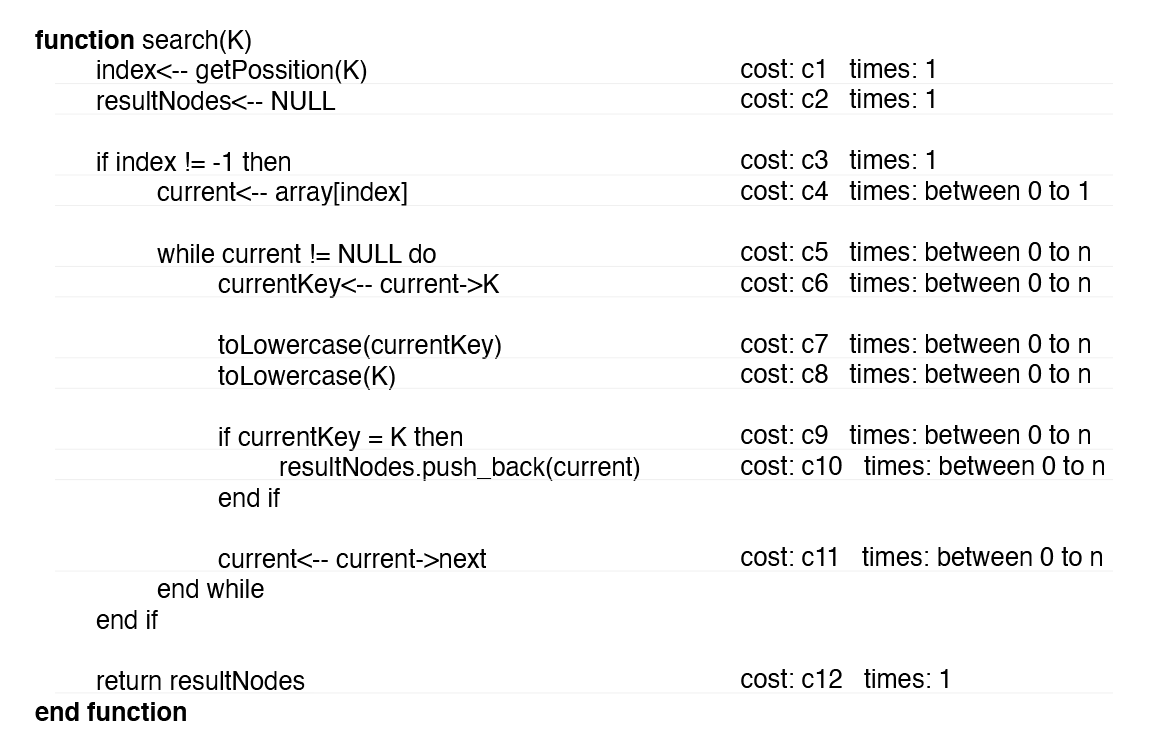


The time complexity of Removing function is O(n), where n is the number of nodes in the linked list. Like in the case of Searching, we must find all nodes where the key is equal to input.

In our application there are two removing functions, one for removing all the nodes with selected key, and the second, for removing one node with selected key and value.

## **SEARCHING**





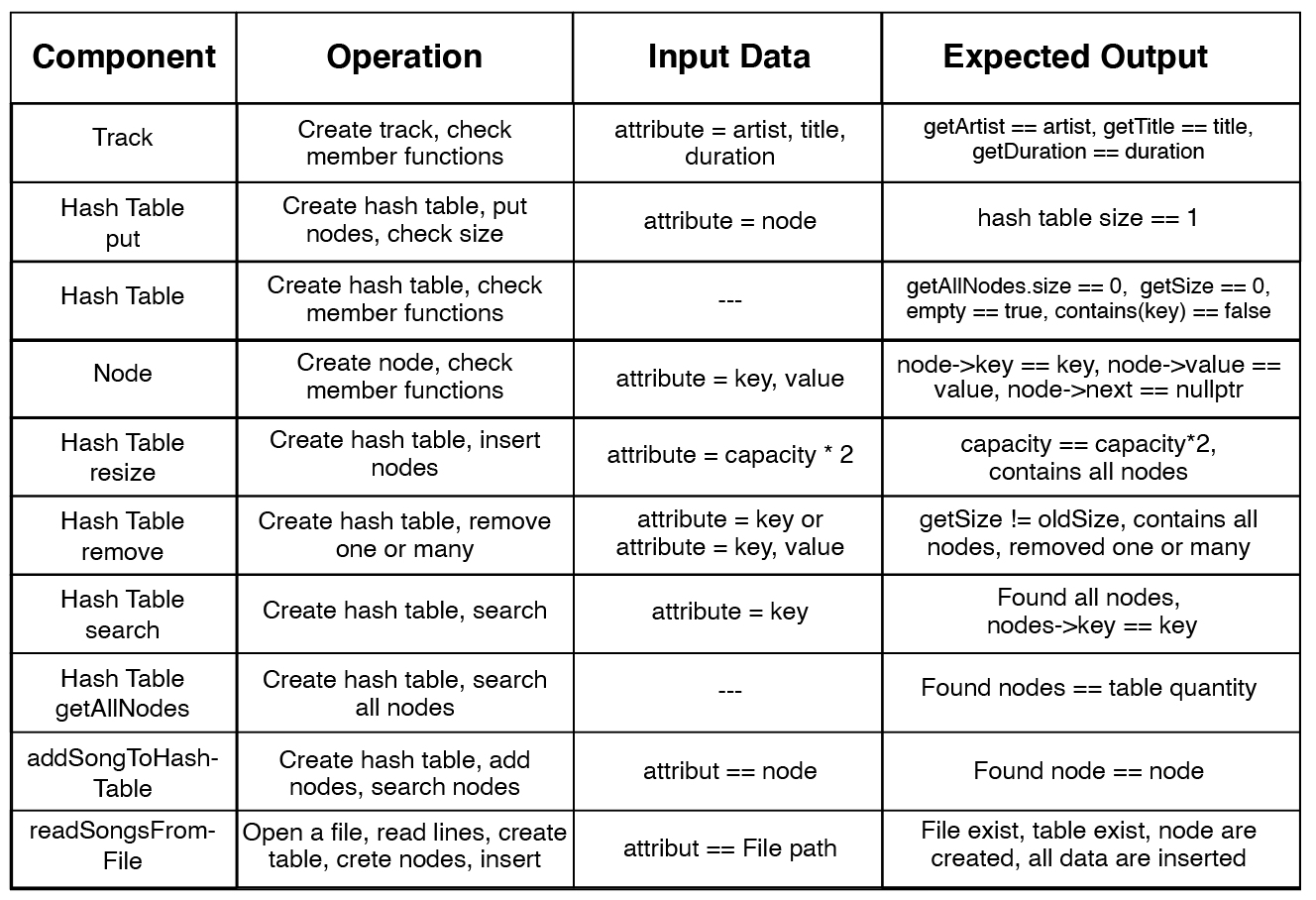
Searching by artist/band name has the average time complexity of O(n) where n is the length of the linked list at the key position in the hash table. Because the output could be one to many (one artist, many tracks), we must check all the nodes in the linked list (Fulber-Garcia, 2023).

In the worst case, time complexity could be O(n), where n is equal to the hash table quantity, because all data are on the same hash table index (in the same linked list) (Fulber-Garcia, 2023).

## **TESTING**

For the test implementation of the presented project, we used the proactive testing approach, in which testing is started as early as possible to find all the problems before increasing the project size (tutorialspoint.com, u.d.).

In the application development process, all the main components were tested, including hash table member functions, put, remove, search, node member functions, reading from a file, creating nodes, searching by user input, removing one or man, etc. Moreover, the components on which changes have been made were tested after each major change or after a certain period of time.



## **CONCLUSION**

**SUMMARY OF WORK DONE**

An appropriate data structure design for an audio streaming application has been presented in this report, which discusses the reasons for choosing the Hash Table with Separate Channing collision handling and why it is more appropriate for our project than linear probing collision handling. Additionally, pseudo code for the most important functions such as Put, Remove and Search has been provided with time complexity calculations and discussion about worst cases and average execution time.

Finally, we covered the testing approach used for application and data structure testing in the process of developing and before project submission. We presented a Tests table, where are shown the tested elements, actions, input and expected output.

**LIMITATIONS AND CRITICAL REFLEXION**

The first limitation of the current project is using application data structure for storing large amount of data instead of storing them in an external Database, which is more efficient for data storing, and searching. The second limitation could be avoiding the problem of memory complexity and the problems that could appear with increasing the data number or the project size.

Additionally, we didn’t cover and analyse properly other data structures or linear probing collision handling. This could have a better time complexity than Separate changing, if for example will be use double hashing for minimising the clusters size or another appropriate solution.

**HOW TO CHANGE APPROACH ON SIMILAR TASK IN THE FUTURE**

In the future, on similar tasks, we could use a different approach and remove all the limitations, using appropriate solutions for every task or part of the project. Starting with more detailed analysis, application design and choosing the most suitable instrument for data storage, like a SQL database, designed specifically for managing a very large amount of data.

It is very important to consider all aspects of the task, as well as the possibility of increasing the application complexity without losing efficiency.

# Bibliography

Fulber-Garcia, V., 2023. *baeldung.com.* [Online]   
Available at: https://www.baeldung.com/cs/hash-tables  
[Accessed 26 03 2023].

tutorialspoint.com, n.d. *tutorialspoint.com.* [Online]   
Available at: https://www.tutorialspoint.com/software\_testing\_dictionary/test\_approach.htm  
[Accessed 26 03 2023].