Prison Management System

Muhammad Miqdad Ahmad
BSCE22001
Department of Computer and Software Engineering
Information Technology University
Lahore, Pakistan
bsce22001@itu.edu.pk

Ahmad Waleed Akhtar
BSCE22003
Department of Computer and Software Engineering
Information Technology University
Lahore, Pakistan
bsce22003@itu.edu.pk

Muhammad Arham
BSCE22007
Department of Computer and Software Engineering
Information Technology University
Lahore, Pakistan
bsce22007@itu.edu.pk

***Abstract*— A project called "Police Management System" intends to make managing prison records easier. This system will offer a user-friendly interface for Admin, prisoner, and normal users while effectively storing and retrieving prison records using a binary tree data structure. Admin will have more authority over the system than a normal user. The system will also take complaints from all types of users and store them in a database.**

I. Introduction

IN THIS PROJECT WE WILL MAKE A PRISON MANAGEMENT SYSTEM. IT WILL STORE INFORMATION ABOUT THE PRISONERS, THEIR PERSONAL INFORMATION, THE CRIMES THEY COMMITTED, THEIR PUNISHMENT, ETC. THIS SYSTEM WILL MAINLY BE FOR THE ADMINISTRATION OF THE PRISON, FOR THEM TO RECORD INFORMATION BUT THERE WILL ALSO BE AN OPTION FOR ORDINARY PEOPLE (RELATIVES OR ACQUAINTANCES OF THE PRISONERS) TO VIEW SOME OF THE INFORMATION OF THE PRISONER WHICH THEY WILL SEARCH FOR BY NAME OR ID.

THE SYSTEM WILL BE PASSWORD PROTECTED TO SAFEGUARD THE INFORMATION FROM GETTING INTO UNAUTHORIZED HANDS. ALL THE MODIFICATIONS WOULD ONLY BE POSSIBLE IF THE USER KNOWS THE PASSWORD TO GET INTO THE SYSTEM.

THE PROGRAM WILL BE MENU DRIVEN, IT WILL PROMPT THE USER TO SELECT WHAT TO DO FROM VARIOUS OPTIONS, E.G., WHETHER HE WANTS TO MODIFY SOME DATA OR ONLY VIEW THE DATA.

II. Objectives

CREATING A USER-FRIENDLY INTERFACE FOR USERS, PRISONERS, AND ADMIN.

EFFICIENTLY MANAGING DATA THROUGH BINARY TREES.

EFFICIENTLY UPDATING, ADDING, AND REMOVING DATA.

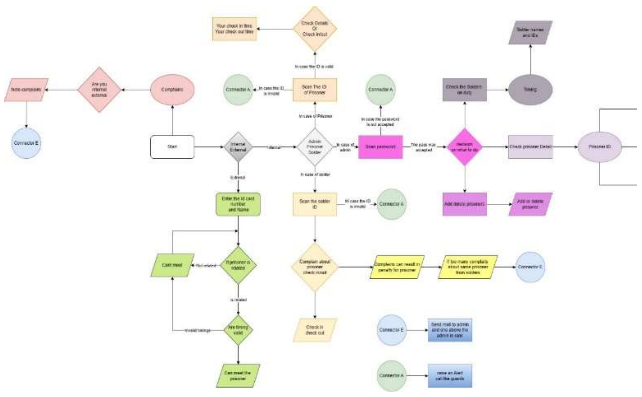
EFFICIENTLY ALLOWING USERS TO SEARCH PUBLIC RECORDS.

LET THE PRISONER SEE HIS RECORD.

USING A USER INTERFACE FOR ADMIN AND THE USER TO INTERACT WITH THE SYSTEM.

III. Methodology

We are using Binary search trees in our code to manage the Data Base. The data is stored in multiple binary trees. First there is a binary tree that divides our database into seven sections depending on the grade (A to G). That way each node represents a single grade.
Within each node on that tree, another tree is formed that stores the data of the prisoners that belong to the respective grade.
Using such Binary trees in such a hierarchical manner reduces the search time even further.
To store data, we could have used linked lists but then the search would have been O(n). In the case of the BST its log(n). In the case of an unbalanced search tree, the complexity is O(n) but we have a function that always converts the BST into a balanced tree so that the complexity is always O(log n). The complexity of the balancing function is O(n).
After balancing, we rewrite the file using the Breath first traversal technique. It writes the file in such a format that when we read the file again to make a tree, the tree formed is fully balanced by default. This further allowed us to improve the program. Now the function to balance the tree is called only when a new node is added or the old one is deleted.
Workflow diagram



IV. Research (Merge Sort vs Quick Sort)

We have written a function to sort the arrays when we add or remove a prisoner. For this, we have used quick sort. The reason for using quick sort is that it is more efficient than the merge sort in case of arrays that are already partially sorted. Merge sort does not adapt and will perform its functionality even if the array is partially sorted. So, it has the same best, worst, and average complexities. But it's different for quick sort. It is more adaptable and has different complexities if the arrays are partially sorted.

A. *Merge sort complexity*

Worst Case: O(n log n)
Average Case: O(n log n)
Best Case: O(n log n)

Requires additional space proportional to the size of the input array. This makes it less memory-efficient, especially for large datasets.
The space complexity is O(n).

B. *Quick sort complexity*

Worst Case: O(n^2)
Average Case: O(n log n)
Best Case: O(n log n)

In general, quick sort has better average-case time complexity than merge sort, but its worst-case time complexity can be worse.

The quick sort version has a space complexity of O(log n) due to the recursive nature of the algorithm.
Quicksort is often more memory-efficient than merge sort because it can be implemented in an in-place manner.

Initially we were using only the Depth first traversal method to move through the tree. We also used this method to write the data from the tree to the file. It was inefficient because we had to balance the tree every time when ever we read from the file.
Then we researched and found out the Breath first method of traversal. This allowed us to write the tree as is in the file. So, when it was read, we got a full balanced tree.

C. *Conclusion (Comparison of merge and quick sort)*

In summary, while both merge sort and quicksort have average-case time complexities of O(n log n), merge sort is more consistent in its performance and is a stable sorting algorithm, but it requires additional space. Quicksort, on the other hand, is usually faster in practice and can be more memory-efficient when implemented in-place, but its worst-case time complexity is higher. The choice between them depends on the specific requirements and characteristics of the data being sorted.
We have used the quick sort algorithm because we want to design a system that will hold the data of a lot of prisoners. So, memory usage will be a problem. The quick sort will never achieve the worst complexity because all our files are partially sorted, and the only sorting needed is when we add or remove the data of a prisoner.

Final Results/Outcomes

The program runs perfectly fine. All the functionalities work efficiently taking time and space complexities into consideration while effectively using data structures to produce the most suitable outcomes.

Moreover, by the use of The Hierarchical tree, we can keep the data in separate files as we make the trees separate from each other. This also makes out program modular.

Also, by the use of the Hierarchical tree we decrease the search time from log(n) to log (k). (Log n is the total number of prisoners in the database and k is the number of nodes in the inner or outer tree. Depending on which one is bigger.)