Oliver Kielian

Bobby Scott

CS 280

23 April 2023

Data Structures Final Project

Introduction: The Problem and how we are addressing it.

We have decided to create a website that can take input of movie names and ratings from a user and then hold that data so it may be retrieved later on. To solve this problem, we will be using these data structures: a red-black tree (RB tree), a binary search tree (BST), an array, a list, and a stack. These data structures will be the core of our solution as they will hold the actual data such as movie names, ratings, etc. We want users to be able to easily interact with our data structure and have the structure return data in a prompt and efficient manner. We also desire that on the server side the data structure does not demand massive amounts of memory. To evaluate the effectiveness of our data structures, we will be conducting a theoretically based analysis. Theoretical analysis will involve evaluating each data structure's time and space complexity for various operations such as search, insertion, and deletion. We will also consider the practical implementation of each data structure in our website, evaluating their response time and memory usage performance. Furthermore, we will use benchmarking and profiling techniques to compare our data structures' performance and identify any bottlenecks. Through this approach, we aim to select the most efficient and practical data structure for our website, while ensuring optimal performance and minimal resource usage.

Data structures: The features of our data structures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Search | Insertion | Delete | Space Comp |
| RB Tree | O(n log n) | O(h) | O(h) | O(n) |
| BST | O(n log n) | O(h) | O(h) | O(n) |
| Array | O(n) | O(1) | O(n) | depends |
| List | O(n) | O(1) | O(n) | O(n) |
| Stack | O(n) | O(1) | O(n) | O(n) |

When it comes to storage the RB tree and BST will use a method that compares titles and sorts then alphabetically. For the array, list and stack, the initial input does not matter as much but there will be a sort function that sorts the titles based on their reviews and will sort them from greatest to least. The runtime for the sort function will likely be O(n log n) as a merge sort function will be used. The “depends” in the space comp column of the table is referring to a dynamic array. A dynamic array will be used in the analysis as there is no knowing how many movies will be inserted into the structure. So, for space complexity, a dynamic array must store the titles already in the array but also any possible extra space in order to account for growth.

Analysis: Our findings

There are several different ways that this problem can be solved. Two of these ways are a red black (RB) tree or a basic binary search tree (BST). Both options offer quick search, insertion, and deletion. These two structures are the best overall in terms of time complexity if they are balanced. RB trees will always be balanced but there is a chance that a BST will not be balanced which will cause the runtimes to be the same as the rest of the structures. There could be a sort function implemented for the BST which would resolve the problem of it possibly being unbalanced, this will be discussed more in depth in the discussion section. Another downfall to these data structures is that they are a little difficult to implement. Luckily, there are libraries that make the implementation of these data structures much easier. The organization of these structures can also be based on several things such as the rating itself or the title of the movie (alphabetical order in a sense). A regular BST may require a sorting function to make sure it stays balanced, but a RB tree has no need for that as it stays balanced at all times.

A linked list or an array are also two structures that can be used to solve this problem. Runtimes for searching and deleting are slower than a BST (RB tree is also included in this as it is a BST) but insertion is faster at O(1) runtime as all inserting just requires putting the title on the end of the list or array. Arrays and linked lists are fairly easy to implement however, arrays can be more complicated if they are required to be dynamic as they would be in this case. Having an array be dynamic also influences it is space complexity as space is required for what is already in the array plus space needed expand the array if needed. Linked lists have the same space complexity (O(n) where n is the number of elements in the list) as a BST which sets them as equal in that requirement. The same holds true for RB trees and the final structure that was consider, a stack.

Stacks have the same time complexities as arrays and linked list as said above. They are also easy to implement and have a space complexity of O(n). The major download with a stack is two stacks are needed for each function. When searching through stacks, the last one must pop off and be stored in the second stack until the title being searched for is found. Implementing the sorting algorithm may prove to be a bit more difficult than an array or a linked list as well. The space complexity is the same as all the other structures have been thus far (excluding the array). In next section, all that was discussed in this section regarding the structures will be compared against each other. Each structure has its strengths and weaknesses but which of the structures strengths will be the most clear and useful for purposes of solving our problem is the question.

Discussion:

Since the problem focuses on searching, organization, and space those will be the elements that are the most considered. Since most of the data structures use the same amount of storage (an array will use a little more) searching and organization will be what we really consider when choosing a structure. Based on these qualifications a linked list or a RB tree will be the final two data structures considered. First, why the other’s did not qualify. An array will use the most space out of all of these structures and while it is not as much as the other structures, since one of the requirements is the least space, array is out of the picture. A BST is also out because the runtimes all depend on the tree being balanced so an extra function for organization would have to be implemented and this is not the case for a RB tree. While there could be a sort function implemented for the BST, that function is already present for a RB tree that is inserted into both the insertion and deletion functions. It would be unnecessary to implement a sort function for a BST when a RB tree already has one built in. A stack would not be a desirable choice because while it is easy to implement the runtimes, searching, sorting, etc. are not as simple as a linked list or a RB tree. So, a stack is just more hassle than it is worth especially if there are more or equally complicated structures that can be used in place that have better runtimes.

The reason a RB tree and a linked list could be used is because the linked list offers the best organization, and a RB tree offers the best runtime for searching. In this case, searching may be the best one to put first. Since our problem has us creating a website that stores movie titles, there could be hundreds of movies stored and searching is what people who use the site are going to be doing. The difference between searching through hundreds of movies with a runtime of O(n) compared to a runtime of O(n log n) really makes a difference. If the website were to hold a small number of titles (small enough that the difference between a searching time of O(n) and O(n log n) is negligible) then perhaps a linked list would work just as well as a RB tree in storing the titles. However, since the website has a chance of growing, implementing a RB tree for the beginning is essential to avoiding copying all the titles from the list to the tree as, at that point, the search runtime will become more important that clear organization. RB trees do offer some form of organization, but it is a little complicated compared to a linked list. However, since the users of the website are unlikely to see the structure itself, how exactly the titles are organized are less important that search runtime.

Conclusion:

In conclusion, a RB tree is the best data structure for our purposes. It offers quick runtimes no matter the number of movies stored in the tree and offers self-balancing properties that allow those runtimes to stay fast. While the organization is not the most ideal, it does offer organization and users will likely not interact with the organization without searching or asking the tree to be printed. The data structure is also a little difficult to implement but with the help of libraries, that task becomes much easier. Overall, based on the requirements our problem faced, a RB will meet them the most efficiently.