Antonio Colantuono

Evaluation for final

**BST**

To start off let’s set some parameters let n = the number of courses in the file, let p = the number of prereqs per course. For BST the cost per line is mostly efficient for the worst case scenarios it is the following: for me to open the file we are at O(1), for the system to read the file line by line we are at O(n), to transpose each line of the file into a list it is O(n) to validate our prerequisites O(p\*logn\*n) and to insert the code into our BST we are at O(n log n). Opening a file is O(1) because we open the file one time not multiple times so it only needs to execute once. For reading the file it is O(n) because the code is reading the file line by line for n times where n is equal to the number of courses in the file. Since the user is not aware of how many courses are in the current file that is why it is O(n). To transpose the file into a list it is also O(n) since we are going line by line and the user does not know how many lines are in the original file. Our most expensive area is validating our prereqs since we have to search the tree for our prereqs and then check which courses the prereqs apply to and make sure all courses have their prereqs listed. Since this is most likely going to be a while loop and then a sorting mechanism inside the while loop that is where the cost is the most expensive. Inserting the courses into our tree are also expensive since it is going to be a while loop that loops over the data multiple times until it is all completed.

**Hash Table**

To start off let’s set the same parameters let n = the number of courses in the file, let p = the number of prereqs per course. Our worst-case scenarios are mostly the same except for the sorting alpha numerically function. For the insert course function, we are running through the data n number of times depending on how many courses we need to insert over a given period so that one is O(n). For searching for a course, we do not know how many courses they are looking for in the scenario so we would have that function be O(n) as well. Same goes for remove a course and print all courses. Our most expensive section is the sorting alphanumerically section since we have to use a while loop and compare items on a list to one another to then print them out in alpha numerical order for the user. The remove insert search and delete could end up being the most time consuming depending on how long of a list we end up having eventually because if the list is five courses long for example it will have a quick turn around time but if the list is 1,500 courses long that could eat into some of our cost.

**Linked List**

For a linked list our slowest areas are the search function, remove function and the sort list function. Since a linked list does not have an automatic ordering feature the ordering of the list could take awhile doing this method causing us to have long wait times and high costs. Searching in linked lists is also the slowest of the three methods so that could eat up a lot of the costs as well and the longer the list the longer the search takes of the data structures. This structure works well for smaller lists that do not need to be sorted in specific methods like alpha numerical order.

**Conclusion**

Now that we went over the worst cases for each structure of data, we can decide which one we are going to use for our project. We are not going to use a linked list because that would not make sense for this project. We want our data to be easily accessible, easy to sort in any specific order and to be the least cost effective and unfortunately for what we are trying to do linked lists is seeming to be the most expensive. Hash tables would be the next most expensive since it does not have a natural sorting storage method and instead everything is ordered in hashes. So for us to organize the data in alphanumerical order, we would have to parse all the hashes for all the course codes and then organize the hash variables that are holding the course codes into a list from smallest to largest and then print the data which can take a lot of time depending on the overall list of courses. For this project a BST works best because we can just call upon an in order traversal method to sort by string course number and then order the data that way without any additional steps. It is also the fastest method for searching and inserting and removing since we are sorting the data in tree format vs a linked list or hashes. BST has a approach where if we are looking for a larger value than our current one we can eliminate half the data set immediately since the left side of a BST is where all the smaller values go and the right side of a BST is where all the larger values go resulting in a quicker turn around time for just about anything when compared to the other two ways of searching for data.

**Runtime Chart**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation | Average | Linked List | BST | Hash Table |
| Insertion | O(n) | O(n) | O(n) | O(n) |
| Deletion | O(n) | O(n) | O(n) | O(n) |
| Search | O(n) | O(n) | O(n) | O(n) |
| Alphanumerical sorting | O(n^2) | O(n^2) | O(n^2) | O(n log n) |
| Reading the file | O(n) | O(n) | O(n) | O(n) |