Jason Barry

Professor Griffith

CS-300-R3292 DSA: Analysis and Design 24EW3

14 February 2024

Project One –Runtime Analysis and Evaluation

**Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Runtime Analysis** | | | |
| **Vector Step** | **Cost per Line** | **Number of Times Executed** | **Total Cost** |
| Creating the Vector | 1 | 1 | 1 |
| For each line in the File | 1 | N | N |
| Creating Vector Course Item (ID, Name, and PreReq) | 3 | N | N |
|  |  | **Total Cost** | 4N + 1 |
|  |  | **Run Time** | O(N) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Table Runtime Analysis** | | | |
| **Hash Table Step** | **Cost per Line** | **Number of Times Executed** | **Total Cost** |
| Creating the Hash Table | 1 | 1 | 1 |
| Create the Key | 1 | 1 | 1 |
| For each line in File | 1 | N | N |
| If no entry in node | 1 | N | N |
| Assign node to Key | N | N | N |
| Else | 1 | N | N |
| assign old node key to UINT\_MAX | 1 | N | N |
| Set Old Course and PreReq to Null Pointers | 2 | N | N |
| Find Next Open Node | N | N | N |
| Fill Open Node | 1 | N | N |
| Add New Node to End | 1 | N | N |
|  |  | **Total Cost** | 9N + 2 |
|  |  | **Runtime** | O(N) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Search Tree (BST) Runtime Analysis** | | | |
| **BST Step** | **Cost per Line** | **Number of Times Executed** | **Total Cost** |
| Create the Tree Method | 1 | 1 | 1 |
| If root is Null | 1 | 1 | 1 |
| Add Root | 1 | 1 | 1 |
| If Node is less than root, go left | 1 | N | N |
| If there is no node to the left | 1 | N | N |
| Add the node to the left | 1 | N | N |
| If Node is greater than root, go right | 1 | N | N |
| If there is no node to the right | 1 | N | N |
| Add a node to the right | 1 | N | N |
| For each line in File | 1 | N | N |
| Create Vector Item (ID, Name, and PreReq) | 1 | N | N |
|  |  | **Total Cost** | 8N + 3 |
|  |  | **Runtime** | O(N) |

**Evaluation:**

The academic advisors in the Computer Science department at ABCU want to determine which data structure (vectors, hash tables, and trees) have the most optimum runtime for loading a list of courses, sorting these courses, and searching for a specific course.

One of the possible data structures being examined are vectors. The primary advantage of vectors is their ability to add and/or remove objects from a list of objects which has an average runtime complexity of O(1). Additionally, vectors are far easier to implement than other data structures. Unfortunately, this ease of use is complicated by an increased runtime complexity when searching for a specific object. After all, if an object is at the end of the vector, each individual element must be searched. This creates a worst-case runtime of O(N) which is inefficient.

Another possible data structure are hash tables. Unlike vectors, this form of data structure excels in searching for a specific value. This is because hash tables (essentially) eliminate half of all items in the list after each iteration through the loop (until the item is found). This means the average runtime for a search function on a hash table is O(1). Unfortunately, hash tables are inefficient for sorting a list of values because the sort function has a worst-case runtime of O(N). Furthermore, hash tables are more difficult to implement than their vector counterpart. This is especially because the programmer needs to know the size of the hash table ahead of time.

The final data structure being examined are known as trees. Overall, trees are the most optimum data structure because of its ability to automatically sort new entries whenever they are added to the tree. Because this process is automatic, the average run-time of adding and sorting a binary tree is O(1). Furthermore, like hash tables, trees are extremely efficient whenever a specific value is search. This is because every iteration through the loop (until an item is found) essentially eliminates half of the items in the tree.

In conclusion, search trees provide the most optimal data structure due to their increased speed when adding, searching for, and sorting a list of items.