CS300 Project One Runtime analysis

6/12/2025

Ryan Erno

Vector Sort Runtime Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| File Read | Cost of line | | # of execution | Total cost | |
| Vector <Course> courseVector | 1 | | 1 | 1 | |
| Loop through ‘while’ contents are in the file(not end) { | 1 | | n | n | |
| String ‘line’ | 1 | | n | n | |
| List <String> ‘tokens’ = SPLIT(line, “,”) | 1 | | n | n | |
| ‘if’ len(tokens “,”) < 2 | 1 | | n | n | |
| Output << “Error: Lines must contain at least two parameters” (courseNumber, title) | 1 | | n | n | |
| Continue to next block | 1 | | n | n | |
|  |  | Big O notation  6n+1 = O(n) | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Creation of course object | Cost of line | # of execution | Total cost |
| Course ‘courseData’ | 1 | n | n |
| courseData.courseNumber = tokens[0] | 1 | n | n |
| courseData.name = tokens[1] | 1 | n | n |
| courseData.prerequisites = tokens[ ] (empty) | 1 | n | n |
|  |  | Big O notation  4n = O(n) | |

HashTable Runtime Table

|  |  |  |  |
| --- | --- | --- | --- |
| File Read | Cost of line | # of execution | Total cost |
| Hashtable <Course> courseTable | 1 | 1 | 1 |
| Loop through ‘while’ contents are in the file(not end) | 1 | n | n |
| String ‘line’ | 1 | n | n |
| List <String> ‘tokens’ = SPLIT(line, “,”) | 1 | n | n |
| ‘if’ len(tokens “,”) < 2 | 1 | n | n |
| Output << “Error Lines must contain at least two parameters (courseNumber, title)” | 1 | n | n |
| Continue to next block | 1 | n | n |
|  |  | Big O notation  6n + 1 = O(n) | |

|  |  |  |  |
| --- | --- | --- | --- |
| Creation of course object | Cost of line | # of execution | Total cost |
| Course ‘courseData’ | 1 | n | n |
| courseData.courseNumber = tokens[0] | 1 | n | n |
| courseData.title = tokens[1] | 1 | n | n |
| courseData.prerequisites = tokens [ ] (Empty) | 1 | n | n |
|  |  | Big O notation  4n = O(n) | |

Binary Tree Runtime Table

|  |  |  |  |
| --- | --- | --- | --- |
| File Read | Cost of line | # of execution | Total cost |
| readTreeData(filename, root) { | 1 | 1 | 1 |
| Node ‘root’ = null | 1 | 1 | 1 |
| Fstream readFile (file.csv) | 1 | 1 | 1 |
| Check ‘if’ (filename.is\_open()) { | 1 | 1 | 1 |
| Loop through ‘while’ contents are in the file(not end) { | 1 | n | n |
| List <String> ‘tokens’ = SPLIT(line, “,”) | 1 | n | n |
| ‘if’ len(tokens “,”) < 2 { | 1 | n | n |
| Output << “Error: Lines must contain at least two parameters” (courseNumber, title) | 1 | n | n |
| Continue to next block | 1 | n | n |
|  |  | Big O notation  5n + 4 = O(n) | |

|  |  |  |  |
| --- | --- | --- | --- |
| Creation of course object | Cost of line | # of execution | Total cost |
| Course courseData | 1 | n | n |
| courseData.courseNumber = tokens[0] | 1 | n | n |
| courseData.name = tokens[1] | 1 | n | n |
| courseData.prerequisites = tokens [] (empty) | 1 | n | n |
|  |  | Big O notation  4n = O(n) | |

Pros and cons to Vectors:

Vectors are a relatively simple data structure to implement compared to the others used here and have easy to use structure for inserting, they are also capable of dynamic sizing allowing for a change in size when a specific number of elements is not known, being the case for different files of lines. The drawback however in this context is that they are limited in data types often only allowing one type during use.

Pros and Cons to HashTables

Unlike vectors, hash tables can utilize various data types at once but like vectors they are also equally capable of having dynamic sizing and can be resized if needed. The main drawback in this context is that it is difficult in terms of logic to implement a proper traversal in.

Pros and Cons to Tree Structures

Trees are much more ordered than the other structures and can maintain an hierarchy like additional data tied to courses, the drawback much like hashtables are complexity and are not as easy to use as vectors, navigation requires careful understanding of its behavior.

Recommendation

From what I surmise from my code approach, all the structures seem to be similar in time complexity with some nuances, however using the hashtable seems like the best option, they are dynamic in sizing but are capable of handling various data types ruling out vectors. Hashtables also in general have a O(1) complexity when it comes to inserting or searching up contents if collisions do not occur.