Jeffrey Karl

CS-300

20 February 2024

Project One: Pseudocode and Runtime Analysis

**Read data from file into vector:**

Use fstream to open file

Open file, if return value = -1, file not found

While not End Of File

Read line

If there are less than two values in a line, throw error

If prerequisite does not exist throw error

Close file

**Vector Create and Store Course Objects:**

Struct course

String courseNumber

String title

String prerequisite1

String prerequisite2

Create vector<course> to store course information

Create new course variable

Open file using fstream

While not EOF

Read line

Parese line into course.courseNumber, course.title, course.prerequisite1, and course.prerequisite2

Push course to vector<course>

Close file

**Hash Table Create and Store Course Objects:**

Struct course

String courseNumber

String title

String prerequisite1

String prerequisite2

Create vector<nodes> nodes to store course information

Create a hash table class

Add an insert function to add items to hash table

Create new course variable

Open file using fstream

While not EOF

Read line

Parese line into course.courseNumber, course.title, course.prerequisite1, and course.prerequisite2

Push course to hashtable using insert function

Close file

**Binary Search Tree Create and Store Course Objects:**

Struct course

String courseNumber

String title

String prerequisite1

String prerequisite2

Create new course variable

Open file using fstream

While not EOF

Read line

Parese line into course.courseNumber, course.title, course.prerequisite1, and course.prerequisite2

Insert course into linked list

Close file

**Menu:**

While user does not input 4

Switch user input

Case 1:

Load data structure

Case 2:

Print course list

Case 3:

Print course

Case 4:

Exit program

**Print Course List:**

Start at the root node and check course name value for alphabetical order.

Travers left and right down the search tree

Print the course information out

**Print course:**

Search tree for specific course.

If the node is found

Print course to screen.

**Exit Program:**

If the user enters option to end program

Print goodbye to screen.

Return zero to kill the program.

**Sort alphabetical order:**

Start at root node

Use temp node

If temp node is less then left not

Switch node with left node

Recall function

Else if temp node is greater then right node

Switch node with right node

Recall function

**Print sorted list to display:**

Call the print course list function

**Evaluation**

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create new course variable** | **1** | **1** | **1** |
| **Open file using fstream** | **1** | **1** | **1** |
| **While not EOF** | 1 | n | n |
| **Read line** | 1 | n | n |
| **Parse line into course struct** | 1 | n | n |
| **Append to vector** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 3 |
| **Runtime** | | | O(n) |

Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create new course variable** | **1** | **1** | **1** |
| **Open file using fstream** | **1** | **1** | **1** |
| **While not EOF** | 1 | n | n |
| **Read line** | 1 | n | n |
| **Parse line into course struct** | 1 | n | n |
| **Append to hash table** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 3 |
| **Runtime** | | | O(n) |

Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create new course variable** | **1** | **1** | **1** |
| **Open file using fstream** | **1** | **1** | **1** |
| **While not EOF** | 1 | n | n |
| **Read line** | 1 | n | n |
| **Parse line into course struct** | 1 | n | n |
| **Append to tree** | 4 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 7n + 3 |
| **Runtime** | | | O(n) |

The advantages of vectors are that they are easy to append to the end and iterate over, but they have poor performance when trying to append to the front or middle of the array. Hash tables support fast operations such as insertion, deletion, and retrieval, but they typically take up more space and they are also unordered. Binary search trees can be traversed in a sorted order, they have a flexible size and fast operations, but they are a complex data structure and harder to use correctly.

Since ordering, insertion, deletion, and retrieval are all important operations to this program, using a binary search tree would provide the best performance for these operations. While it might be cheap to append to the end a vector, having to sort the vector later is going to be an expensive operation. Hash tables are unordered, so they aren’t going to be a good choice for a data structure that needs to be sorted. So, for the use case of this software, I would recommend using a binary search tree.