Lukas Pentowski

CS300 DSA: Analysis and Design

Project One

**Vector**

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**fin OPENS file**

**IF file IS OPEN**

**Create variables to read line, courseNum, courseTitle, coursePreq**

**WHILE not at EOF**

**Line EQUALS next line**

**Break down line into different params using the separator char**

**IF no info for coursePreq**

**Set it to be empty**

**RETURN variables of the line**

**CLOSE file**

**ELSE**

**Message saying File could not be open**

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

**Vector(file info)**

**Create a new vector**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**Store output from openFile into the Vector**

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites**.

searchCourses(courseNum param)

Create a vector of the course items, ensuring all three areas have some form of data even if empty (Prereq may be empty)

WHILE not at the end of the vector

IF current vector item matches the courseNum prompt

RETURN course information

ELSE

Current item EQUALS next item

RETURN empty item or Course Cannot be found statement

**Hash Table**

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**fin OPENS file**

**IF file IS OPEN**

**Create variables to read line, courseNum, courseTitle, coursePreq**

**WHILE not at EOF**

**Line EQUALS next line**

**Break down line into different params using the separator char**

**Verify at least info for courseNum and courseTitle exist**

**IF there is info for coursePreq**

**Loop through coursePreq values**

**Search each line’s courseNum info for a matching value**

**IF no info for coursePreq**

**Set it to be empty**

**RETURN each value**

**CLOSE file**

**ELSE**

**Message saying File could not be open**

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

**Create a Vector for nodes**

**Create a HashTable**

**Create a Hash function to create keys from courseNum**

**insertCourse(courseNum)**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**Call Hash(courseNum)**

**Store output from openFile into the buckets using the vector nodes**

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites**.

searchCourses(courseNum param)

Call Hash(courseNum) for key value

WHILE not at the end of the table

IF current node key matches the key value and the courseNum value matches

RETURN course information

ELSE

Current item EQUALS next item

RETURN empty item

**Binary Search Tree**

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**fin OPENS file**

**IF file IS OPEN**

**Create variables to read line, courseNum, courseTitle, coursePreq**

**WHILE not at EOF**

**Line EQUALS next line**

**Break down line into different params using the separator char**

**Verify at least info for courseNum and courseTitle exist**

**IF there is info for coursePreq**

**Loop through coursePreq values**

**Search each line’s courseNum info for a matching value**

**IF no info for coursePreq**

**Set it to be empty**

**RETURN each value**

**CLOSE file**

**ELSE**

**Message saying File could not be open**

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

Establish a Binary Search Tree structure

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

IF the BST has no current nodes

First line that is not a header becomes the root node

IF the course value is LESS than the current node

Course becomes left child

IF the course value is GREATER than the current node

Course becomes right child

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites**.

searchCourses(courseNum param)

Start at the root node

WHILE current node is not null

If param is EQUAL to current node

RETURN node info

ELSE IF param is LESS than current node

Current node EQUALS left child node

ELSE IF param is GREATER than

Current node EQUALS right child node

**Menu**

DISPLAY main menu:

1. Load Course Information
2. Print Course List
3. Print Course
4. Exit

INPUT choice

WHEN choice EQUALS 1

LOAD Course Information CSV to proper storage structure

BREAK

WHEN choice EQUALS 2

Iterate through storage structure

OUTPUT each item

BREAK

WHEN choice EQUALS 3

PROMPT for Course to search for

Call searchCourses using the INPUT from prompt

IF searchCourses RETURNS a value that IS NOT null

OUTPUT the Course Information

ELSE

OUTPUT course not found message

BREAK

WHEN choice EQUALS 9:

Print “Good bye.”

RETURN

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

**Sort the course information by alphanumeric course number from lowest to highest.**

**Print the sorted list to a display.**

**Vector**

**Vector(file info)**

**Create a new vector**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**Store output from openFile into the Vector**

**sortVector(Vector name, low, high)**

**Select pivot to be the middle element**

**WHILE NOT done**

**WHILE Course Number is GREATER THAN pivot**

**INCREASE the low**

**WHILE Course Number is GREATER THAN pivot**

**DECREASE the high**

**IF low IS GREATER THAN OR EQUAL TO high**

**DONE**

**ELSE**

**SWAP low and high values**

**INCREASE low**

**DECREASE high**

**Iterate through Vector**

**OUTPUT Course Information**

**HASH Table**

**Create a Vector for nodes**

**Create a HashTable**

**Create a Hash function to create keys from courseNum**

**insertCourse(courseNum)**

**openFile(courseNum param, courseTitle param, coursePreq param, separator char)**

**Call Hash(courseNum)**

**Store output from openFile into the buckets using the vector nodes**

**Enumerate through the keys turning the HashTable into a sorted map**

**Use Map begin to OUTPUT class information**

**Binary Search Tree**

**Create vector elements of each course**

**Create an array of the vector elements**

**Use the array elements of vectors to create the BST**

**Use In Order Traversal to output from smallest Course Number to highest.**

| **Reading File** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **openFile(courseNum param, courseTitle param, coursePreq param, separator char)** |  |  |  |
| **fin OPENS file** | 1 | 1 | 1 |
| **IF file IS OPEN** | 1 | 1 | 1 |
| **Create variables to read line, courseNum, courseTitle, coursePreq** | 4 | 1 | 4 |
| **WHILE not at EOF** | 1 | n | n |
| **Line EQUALS next line** | 1 | n | n |
| **Break down line into different params using the separator char** | 1 | n | n |
| **IF no info for coursePreq** | 1 | n | n |
| **Set it to be empty** | 1 | n | n |
| **RETURN variables of the line** | 1 | n | n |
| **ELSE** |  |  |  |
| **Message saying file could not be open** | 1 | 1 | 1 |
| **Total Cost** | | | 6n + 7 |
| **Runtime** | | | O(n) |

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Vector(file info)** |  |  |  |
| **Create new vector** | 1 | 1 | 1 |
| **openFile(courseNum param, courseTitle param,coursePreq param, separator char)** |  |  | 6n+7 |
| **Store output from openFile into the Vector** | 1 | n | n |
| **Total Cost** | | | 7n + 8 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create a vector for nodes** | 1 | 1 | 1 |
| **Create a HashTable** | 1 | 1 | 1 |
| **Create a Hash function to create keys from courseNum** | 1 | 1 | 1 |
| **insertCourse(courseNum)** |  |  |  |
| **openFile(courseNum param, courseTitle param, coursePreq param, separator char)** |  |  | 6n+7 |
| **Call Hash(courseNum)** | 1 | n | n |
| **Store output from openFile into the buckets using the vector nodes** | 1 | n | n |
| **Total Cost** | | | 8n + 10 |
| **Runtime** | | | O(n) |

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Establish a Binary Search Tree structure** | 1 | 1 | 1 |
| **openFile(courseNum param, courseTitle param, coursePreq param, separator char)** |  |  | 6n+7 |
| **IF the BEST has no current nodes** | 1 | 1 | 1 |
| **First line that is not a header becomes the root node** | 1 | 1 | 1 |
| **IF the course value is LESS than the current** | 1 | n | n |
| **Course becomes left child** | 1 | n | n |
| **IF the course value is GREATER than the current node** | 1 | n | n |
| **Course becomes right child** | 1 | n | n |
| **Total Cost** | | | 10n + 10 |
| **Runtime** | | | O(n) |

Out of the three data structures analyzed the Vector Structure was the fastest to read and insert course elements to it, with cost at 7n+8. Opening and reading the file would be the same total cost of 6n+7. From there each different structure would require the lines necessary to establish the structure and any lines for evaluation and inserting elements.

Each structure has its own advantages and disadvantages to using it. For vectors, some of the advantages are ease of creating and storing elements to the structure, as well as sorting the vector. A disadvantage would be the amount of time required to search the vector. Each element would need to be compared in order to find the right one. If the vector does not contain the corresponding element, the search would only be complete after going through all X elements. An advantage to using a Hash Table would be the speed of searching for an element. Since each element is stored in buckets based on a corresponding key value the length of time required to search a table is shorter in comparison to a vector. The bucket that matches the key value would be the scope of the search. This leads to one of the disadvantages of using a Hash Table though. Duplicate values would need to be stored in separate buckets to prevent confusion. This requires additional time to insert elements to the table. Lastly, an advantage to using a Binary Search Tree is the search speed compared to a vector. At most only half the total number of elements will be used to find the requested element, thanks to the less than or greater than the current node values of the child nodes. If the value is greater than the current, the search only needs to go to the right child and ignore the child on the left. A disadvantage to using a Binary Search Tree is the complexity of adding or removing a new node. Depending on the values, there may be several nodes shifting to accommodate the new node.

Though a vector had the lowest total cost for runtime analysis, I would recommend using a Hash Table for storing the information required by the school. The Hash Table was right behind the vector with a total cost of 8n+10. I recommend the Hash Table due to the search requirements of the school, as well as ease of adding information to the table. There should be no issues with duplicate elements due to the unique course number values and class titles.