Vector Pseudocode

// Opening program from the file

Vector <Course> (string courseCsv)

Initialize fstream to open file

Open file using file name

If file not found:

Print “Error, file not found.”

Return

Else file is found:

Read each line in file

Parses each line = Course number, course title, prerequisite

If items Less Than 2:

Print “Error, missing data.”

Return

Close the file

// Create course objects

Open file

Initialize course Vector vector<int> courseInfo()

Get course number, title, and prerequisite

Create new course object

Set course number, title, and prerequisite

Add course object to course list

Close file

Return

// Search data structures

Open file

Get input

Loop through vector

IF input == courseNumber

Output “Course Number: “ + courseNumber

Output “Course Title: “ + courseTitle

Output “Course Prerequisites: “ + coursePrerequisites

If prerequisites empty:

Output “No prerequisites needed.”

Else:

Output "Invalid course number.”

Close File

Hash Table Pseudocode

// Opening program from the file

Initialize fstream to open file

Open file using file name

If file not found:

Output “Error, file not found.”

Return

Else file is found:

Read each line in file

Parses each line = Course number, course title, prerequisite

If items Less Than 2:

Output “Error, missing data.”

Return

Close the file

// Load Data info a hash table

Create hashTable class

Initialize Course Vector vector<Node>

Create courseId int;

Create courseTitle string;

Create prerequisiteOne string;

Create prerequisiteTwo string;

Create prerequisiteCount int;

Initialize prerequisiteCount equal 0

Create Loop to load data

While not end of file

Loop through file

Unsigned int key

If key not found

Insert new key

Else if node pointer equal int max

Node pointer equal key

Node pointer next equal nullptr

Node pointer equal course

Else

While node pointer not equal to nullptr

Node equal node pointer next

New node (course, key)

// print data

For unsigned int equal 0, increment count

If node key is not equal int max

Print courseId, courseTitle, prerequisite1, prerequisite2

Iterate to next node key

While current key not equal to nullptr

Print courseId, courseTitle, prerequisite1, prerequisite2

Current key points to next

Return

End

Tree Pseudocode

// Opening program from the file

Initialize fstream to open file

Open file using file name

IF file not found:

Output “Error, file not found.”

Return

Else file is found:

Read each line in file

Parses each line = Course number, course title, prerequisite

If items Less Than 2:

Output “Error, missing data.”

Return

Close the file

// Define Binary tree attributes

Private

Node root

Void addNode

Void inOrder

Void preOrder

Void postOrder

Node removeNode

Public

BinarySearchTree

Void inOrder

Void preOrder

Void postOrder

Void Insert

Void Remove

Node Search

// Constructor

Set root to null

// Destructor

Recurse deleting every node

//Traverse Tree in order

This points to inOrder root

//Traverse tree in post-order

This points to postOrder root

// Traverse tree in preOrder

This points to preOrder root

// Insert a node into tree

If root is null

Insert new root

Else

Add node to root

// Remove a node

This points to remove root

// Search for a node

Current node is root

While current node is not null

If current node is equal to node

Return current node

If node is smaller than current node

Traverse node left

Else

Traverse node right

Return node

// Add a node

if node is bigger than current

if node points left is null

node equals new node

else

recurse down left

else

if node right is null

node equals new node

else

recurse down left

// pre order

If node is not null

In order node points left

Print Course number, course title, prerequisite

In order node points right

// post order

If node is not null

Post order node points left

Post order node points right

Print Course number, course title, prerequisite

// Pre order root

If node is not null

Print Course number, course title, prerequisite

Pre order node points left

Pre order node points right

Menu Pseudocode

// Create Menu class

Initialize choice equal to 0;

Print Menu choices

1. Load Course File
2. Print Course List
3. Print Selected Course
4. Exit

While choice is not equal 4

Switch statement (user input)

Case 1:

Loads course data file

Break

Case 2:

Print course list

Break

Case 3:

Get course selection

Print selected course

Break

Case 4:

Exit

Return

Print

Alphanumeric Pseudocode

// Opening program from the file

Initialize fstream to open file

Open file using file name

IF file not found:

Output “Error, file not found.”

Return

Else file is found:

Read each line in file

Parses each line = Course number, course title, prerequisite

If items Less Than 2:

Output “Error, missing data.”

Return

Close the file

// Create sorted course partition vector to divide courses to low and high

Partition sort vector courses

Int low is beginning

Int high is end

Int pivot = begin + (end – begin) / 2

Bool done = false

// While loop for not done

While not done

While course low continue incrementing

While course high continue decrementing

If low is greater than or equal to high

Done

Else

swap low and high course

Return high

// Create quicksort vector

Selection sort vector courses

Unsigned int min

Unsigned int max is equal to course size

For position 0, position less than max, increment position

Set min equal to position

For unsigned i equal position + 1, i less than max, increment position

If course I compares to less than min

Min is equal to i

If min is not equal to position

Swap position with min

Print Courses from min, increment

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | Time executed | Overall Cost |
| Initialize vector | 1 | 1 | 1 |
| Get course number | 1 | n | n |
| Get course title | 1 | n | n |
| Get course prereq | 1 | n | n |
| Set course number | 1 | n | n |
| Set Course title | 1 | n | n |
| Set Course Prereq | 1 | n | n |
| Add course to list | 1 | n | n |
| Return | 1 | n | n |
| Get input | 1 | n | n |
| If loop | 6 | n | n |
| Else | 2 | n | n |
|  | Line Cost 18 |  | Runtime O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table | Line Cost | Time Executed | Overall Cost |
| Create Class | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| Create attributes | 5 | n | n |
| Initialize PrereqCount | 1 | n | n |
| Load Data Loop | 14 | n | n |
| Unsigned int = 0 | 1 | n | n |
| If statement | 1 | n | n |
| Print | 1 | n | n |
| Iterate to next node | 1 | n | n |
| While loop | 2 | n | n |
| Current points next | 1 | n | n |
| Return | 1 | n | n |
|  | Line cost 30 |  | Runtime O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| BST | Line Cost | Time executed | Overall Cost |
| Private Attributes | 6 | 1 | 1 |
| Public Attributes | 7 | 1 | 1 |
| Set root null | 1 | n | N |
| Recurse delete | 1 | n | N |
| Traverse in order | 1 | n | N |
| Traverse post order | 1 | n | N |
| Traverse pre order | 1 | n | n |
| If root null | 2 | n | n |
| Else add node | 2 | n | n |
| Remove node | 2 | n | n |
| Current node = root | 1 | n | n |
| While loop | 3 | n | n |
| If statement | 2 | n | n |
| Else statement | 2 | n | n |
| Return node | 1 | n | n |
| If node bigger than | 5 | n | n |
| Else | 5 | n | n |
| Preorder If not null | 4 | n | n |
| Postorder if not null | 4 | n | n |
| Preorder root not null | 4 | n | n |
|  | Line cost 55 |  | O(n) |

Evaluation

The vector has a worst case runtime of O(2) and a line cost of 18. A vector is great for simple and insignificant amounts of data. The data setup is straightforward and leaves a small margin for error. It is easy to add additional entries into a vector. Vectors are not ideal for large sums of data, as they search data 1 by 1. Given the simplicity of the data structure, a vector is not the best suited for complex functions such as searching and printing.

The hash table has the worst case runtime of O(2) and a line cost of 30. A hash tables divide up the sorting process, which makes for a quicker sort. Searching for single items will be a faster process than a vector with the hash tables divide and conquer strategy. A hash table setup is more complex than a vector setup. This method is also not best suited for searching and printing functions.

The binary tree has the worst case runtime of O(2) and a line cost of 55. This method has the same complexity as a hash table in regards of setup. The nature of the binary tree method sort data as it is entered into the database. This allows for the quickest sorting abilities and is ideal for large amounts of data. Because the data is already sorted to begin with, searching and printing functions will run easily. The only disadvantage for the tree is the use of more resources to run it.

Recommendation

Due to the advantages of a binary sort tree and its ability to traverse directions between entries, I believe this is the best data structure to suit the needs of the course catalog. The biggest drawback is the complexity of setting up, but after initial set up it will be best for searching and printing course data.