Alexander Ray

CS 300

Prof. Alim

October 13, 2024

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **Print course info** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 2 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |

Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **Print course info** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 2 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

Binary Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **Print course info** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

Vector

**Program opens file**

While file still running

Program reads lines

For all lines read:

If no format errors found

Parses lines

Else:

Output (“Format error detected.”)

Split line tokens

If tokens <= 2

Return false

Output (“Error.”)

Else: return true

Program reads lines

Course object()

course= new Course

Title= course.title

Number= course.number

prerequisites= course.prerequisites

return Course

end

Append()

vector= new node

end

Search data structure

If courseNumberSearch= courseNumber

Output (Course information and prerequisites)

Else:

Output(Error not in sync.)

PrintAll()

for nodes from beginning

if current node != Max

Output( objectinfo)

node= next node

loop while node != nullptr

output(object info)

node= next node

Hash Table

While file still running

Program reads lines

For all lines read:

If no format errors found

Parses lines

Else:

Output (“Format error detected.”)

Split line tokens

If tokens <= 2

Return false

Output (“Error.”)

Else: return true.

Determine hash table dimensions

Determine hash table dimensions

Remove(nodes.beginning)

return key table size

int key= course.couseId str()

node\*node= node[key]

if node->key== Max

node->course= course

node->key= key

else if node->key==Max

node->key=Max

node->next=new Node(course, key)

else

node\*currentNode= node

loop while currentNode->next != nullptr

currentNode = currentNode->next

currentNode->next= new Node(course, key)

PrintAll()

for nodes from beginning

if current node != Max

Output( objectinfo)

node= next node

loop while node != nullptr

output(object info)

node= next node

Binary Tree

Program opens file

While file still running

Program reads lines

For all lines read:

If no format errors found

Parses lines

Else:

Output (“Format error detected.”)

Split line tokens

If tokens <= 2

Return false

Output (“Error.”)

Else: return true

BinarySearchTree()

root =nullptr

loop while root != nullptr

Remove root->course

addNode()

if node-> course compare(courseId)>0

if node->left== nullptr

node->left= new node

else

addNode(node->left, course )

else

if node->right==nullptr

node->right = new node(course)

else

addNode(node->right, courses)

PrintAll()

for nodes from beginning

if current node != Max

Output( objectinfo)

node= next node

loop while node != nullptr

output(object info)

node= next node

Assessment

The vector sorting method is the most efficient overall according to the parameters set, even though it is the most meticulous in having to check items individually and returning a positive or negative assessment for the presence of the desired variable instead of the other methods’ “dividing and conquering ” of specified group parameters. Initially one might think this would make the method take longer and be less efficient overall, but when you look at the fact that creating and searching specified groups adds more steps to the overall process it makes sense that vector sorting’s more simplified and straightforward approach works faster.

Hash Tables were my initial guess for which method would be most efficient since the keys assigned to the values seem like they would make returning the desired course simple, easy, and efficient. While it is true that the method is faster and more efficient once the list and keys are implemented, it is the method through which the program must go about assigning keys and creating the list where the inefficiencies begin to show. Creating a space for each individual course can be time consuming as well as the fact that specified organizational capabilities of hash tables is limited means that the upfront inefficiencies outweigh the eventual payoff.

Binary trees are efficient in the upfront sorting and assigning that hash tables must take extra steps to execute, however in the extra steps it takes to create and navigate the branching pathways the vector sorting method can simply go from node to node comparing them directly instead of needing to create and navigate branches checking for the nullptr. The need to consistently check the left and right child branches and select the desired pathway adds an extra layer of complexity where everything must be properly grouped, assessed and connected.

In conclusion since the vector sorting method is the most efficient overall with its straightforward approach, it would be the best choice for our program at ABCU.