Augustus Mendy

CS-300-11373-M01

Southern New Hampshire University

June 15, 2024

**6-2 Submit Project One**

Pseudocode for Vector Data Structures CLASS Course

{

courseNumber, a variable of integer type course

Name is a string type variable.

Prerequisites for vector types with string data types}

Vector<Course> courses Function ReadFileData()

{

Initialize fData variable from class fstream Print "Enter a file name with extension “.txt” Initialize a string fileName

Take file name into fileName variable Set text variable to “.txt”

IF(strstr(Parameters : fileName.c\_str(),text.c\_str())) is not contain)

{

return

}

ELSE

{

Opened file fData using open(fileName) Initialize a string variable line1

WHILE(get the file data using fData variable)

{

CourseClass obj IF(line1 is valid or not)

{

Initialize stringstream function with splits with line1 Initialized variable string token

Initialize vector with string data Initialize vector with string dataPreq

While (splits data and assigns data into token)

{

Adds data token into vector data using pushback function

}

Sets courseName with data (data[0]) Sets courseNumber with data (data[1]) Initialize vector with string dataPreq FOR i =2 to length of vector data

Add data token into vectors dataPreq using push\_back function

IF (length of data is greater >=2)

{

obj.course\_Number=courseNumber obj.course\_Name=courseNumber

Copy vector(obj.course\_prerequisites ,dataPreq)

Add data obj into vector courseData using push\_back(obj)

}

ELSE

Print data is not valid

}

}

}

}

}

Function StoreFileData ()

{

Initialized fData object of fStream class fData.open("course.txt")

IF(fData is not exist)

{

print "File is not found" return zero

}

ELSE

{

Initialized fData string with line type

While(get file data using fData variable)

{

Course b

IF(line is valid or not)

{

stringstream splits line of the file Initialized variable string temp Initialized vector with string da

While (split assigns data into temp)

{

vectors da adds data using push\_back(temp)

}

Initialized cName,cNum Set cName with data[0] Set cNum with (data[1])

Initialized vector<string> preq FOR i=2 to length of vector da

vectors preq adds data using push\_back(da[i]) IF length of da is greater >=2

Set b.courseNumber with cNumber Set b.courseName with cName

vector courses data using push\_back(b) ELSE

Print data is not valid

}

}

}

}

Function SearchFileData ()

{

Print "Enter course number to search" Initialized variables courseNumber Initialized variables courseName

Take user input in course A bool variable T=false

FOR i =0 to length of courses vector

IF courses[i].courseNumber is equal to courseNumber then Print "course found "

T= true

IF T is not true:

Print "Course Not Found" Print "Course Details are "

FOR i =0 to length of courses vector

Display "Course Number is " courses[i].courseNumber Display "Course Name is " courses[i].courseName Display "Course prerequisites are "

FOR i =0 to length of courses[i].prerequisites vector Display courses[i].prerequisite[i]

}

Hash Table Data Structure Pseudocode CLASS Course

{

Integer type variable courseNumber String type variable courseName

Vector type with string data type prerequisites

}

Hashtable<Course> courses

Function ReadFileData()

{

Initialize fData variable from class fstream Print "Enter a file name with extension “.txt” Initialize a string fileName

Take file name into fileName variable Set text variable to “.txt”

IF(strstr(Parameters : fileName.c\_str(),text.c\_str())) is not contain)

{

return

}

ELSE

{

Opened file fData using open(fileName) Initialize a string variable line1

WHILE(get the file data using fData variable)

{

CourseClass obj IF(line1 is valid or not)

{

Initialize stringstream function with splits with line1 Initialized variable string token

Initialize hashtable with string data Initialize hashtyable with string dataPreq Initialize unassigned hash int key

While (splits data and assigns data into token)

{

Adds data token into vector data using pushback function

}

Sets courseName with data (data[0]) Sets courseNumber with data (data[1])

FOR i =2 to length of unassigned hash key

Add data token into data table dataPreq using push\_back function

IF (length of data is greater != key)

{

obj.course\_Number=courseNumber obj.course\_Name=courseNumber

Copy data table(obj.course\_prerequisites ,dataPreq)

Add data obj into data table courseData using push\_back(obj)

}

ELSE

Print data is not valid

}

}

}

}

}

Function StoreFileData ()

{

Initialized fData object of fStream class

fData.open("course.txt") IF(fData is not exist)

{

print "File is not found" return zero

}

ELSE

{

Initialized fData string with line type

While(get file data using fData variable)

{

Course b

IF(line is valid or not)

{

stringstream splits line of the file Initialized variable string temp Initialize hashtable with string data Initialize unassigned hash int key While (split assigns data into temp)

{

Hash table data adds data using data table

}

Initialized cName,cNum Set cName with data[0] Set cNum with (data[1])

Initialized hash<string> preq

FOR i=2 to length of unassigned hash key data vectors preq adds data using push\_back(da[i])

IF length of data is greater != key

Set b.courseNumber with cNumber Set b.courseName with cName

Data table courses data using push\_back(b) ELSE

Print data is not valid

}

}

}

}

Function SearchFileData ()

{

Print "Enter course number to search" Initialized variables courseNumber Initialized variables courseName Initialize unassigned hash int key Take user input in course

A bool variable T=false

FOR i =0 to length of courses unassigned hash key

IF courses[i].courseNumber is equal to courseNumber then Print "course found "

T= true

IF T is not true:

Print "Course Not Found" Print "Course Details are "

FOR i =0 to length of courses unassigned hash key Display "Course Number is " courses[i].courseNumber Display "Course Name is " courses[i].courseName Display "Course prerequisites are "

FOR i =0 to length of courses[i].prerequisites vector Display courses[i].prerequisite[i]

}

Binary Tree Search Data Structure Pseudocode CLASS Course

{

Integer type variable courseNumber String type variable courseName

Vector type with string data type prerequisites

}

BinaryTreeSearch<Course> courses Function ReadFileData()

{

Initialize fData variable from class fstream Print "Enter a file name with extension “.txt” Initialize a string fileName

Take file name into fileName variable Set text variable to “.txt”

IF(strstr(Parameters : fileName.c\_str(),text.c\_str())) is not contain)

{

return

}

ELSE

{

Opened file fData using open(fileName) Initialize a string variable line1

WHILE(get the file data using fData variable)

{

CourseClass obj

IF(line 1 key matches desired key or not)

{

Initialize stringstream function with splits with line1 Initialized variable string token

Initialize hashtable with string data Initialize hashtyable with string dataPreq Initialize unassigned hash int key

While (splits data and assigns data = null)

{

Adds data token into vector data using pushback function

}

Sets courseName with data (data[0]) Sets courseNumber with data (data[1])

FOR i =2 to length of unassigned current key

Add data token into data table dataPreq using push\_back function

IF (length of data is greater != current key)

{

obj.course\_Number=courseNumber obj.course\_Name=courseNumber

Copy data table(obj.course\_prerequisites ,dataPreq)

Add data obj into data table courseData using push\_back(obj)

}

ELSE

Print data is not valid

}

}

}

}

}

Function StoreFileData ()

{

Initialized fData object of fStream class fData.open("course.txt")

IF(fData is not exist)

{

print "File is not found" return zero

}

ELSE

{

Initialized fData string with line type

While(get file data using fData variable = current key)

{

Course b

IF(key is valid or not)

{

stringstream splits line of the file Initialized variable string temp Initialize tree with string data

Initialize unassigned key Initialize Node\* left

Initialize Node\* right

While (split assigns data into temp)

{

Tree data adds data using data table

}

Initialized cName,cNum Set cName with data[0] Set cNum with (data[1])

Initialized hash<string> preq

FOR i=2 to length of unassigned hash key data vectors preq adds data using push\_back(da[i])

IF length of data is greater != key

Set b.courseNumber with cNumber Root = current->left

Set b.courseName with cName Root = current->right

Data tree courses data using push\_back(b) ELSE

Print data is not valid

}

}

}

}

Function SearchFileData ()

{

Print "Enter course number to search" Initialized variables courseNumber Initialized variables courseName Initialize unassigned key

Take user input in course A bool variable T=false

FOR i =0 to length of courses unassigned hash key

IF courses[i].courseNumber is equal to courseNumber then Print "course found "

T= true

IF T is not true:

Print "Course Not Found"

Print "Course Details are "

FOR i =0 to length of courses current key

Display "Course Number is " courses[i].courseNumber Display "Course Name is " courses[i].courseName Display "Course prerequisites are "

FOR i =0 to length of courses[i].prerequisites vector Display courses[i].prerequisite[i]

}

With the menu I am going to follow the same lines as the previous assignments with setting the user input 1-3 and 9 to exit.

Menu Pseudocode

While choice does not equal 9

{

Print “1. Load Data”

Print “2. Print Course List” Print “3. Print Course” Print “9. Exit”

Print “ Please Make Selection” Switch User menu input

{

User inputs 1

Program loads course data Prints Please Make Selection

User input 2

Print course number and name Prints Please Make Selection

User input 3

Print “Enter course number to search” User input Course number

Print Course number, course name, prerequisites course numbers Prints Please Make Selection

User input 9

Print “Goodbye”

}

}”

Vector

|  |  |  |  |
| --- | --- | --- | --- |
| **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** **out** **the** **course**  **information** | 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** | | | 1(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** **out** **the** **course**  **information** | 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) |

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** **out** **the** **course**  **information** | 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) |

The benefits of a vector data structure are its simple implementation and the quickest way to add items and read files. It is possible to store multiple objects and remove vector elements. The fact that vectors require more memory because they are objects is a drawback.

Direct access to items is one of a hash table's benefits. Inserting and deleting data from hash tables can be done instantly. If used properly, hash tables can be the fastest data structures available. Hashing tables have two drawbacks: they use a lot of memory and element retrieval breaks order. Our software would not function if we had to extract, sort, and print each course's value since we would need to list the courses in alphabetical order.

The retrieval of things in order is one of the benefits of the binary search tree. O(logn) time allows for the insertion and deletion of objects. Compared to other data structures, the access speed is both faster and more efficient. The requirement for balance is one of a binary search tree's drawbacks. Since binary search trees are better suited for listing courses in alphanumerical order, that is the structure I would use. The courses are sorted using a tree traversal. It takes O(logn) time for the binary search tree to search, which is enough time to run the course lists program.

Recommendation

The presumption is that the data will be read into memory rarely, totally printed infrequently, and searched frequently; consequently, the Hash Table should be preferred. However, this means that the hash function and table size must be tuned to reduce collisions, allowing the code to run in O(1) rather than O(N).