Reese Thurman  
CS 300  
Project One  
10/13/2024  
  
**Milestone 1 Pseudocode**

Define Course Structure

String courseTitle

String courseNum

Vector<String> prereqs

Declare Variables

Vector<Course> courses

Vector<String> getPrereqs(String fileLoc) {

Vector<String> courseList

Open file at fileLoc

While there are lines in file {

Read line

Parse line by “,”

Add course title to courseList

}

Close file

Return courseList

void getCourses(String fileLoc) {

Bool isError

Vector<String> courseList

Vector<String> prerequisites

String courseNum

String courseTitle

courseList = getPrereqs(fileLoc);

Open file at fileLoc

While there are lines in file {

isError = false

Read line

Parse line by “,”

If line < 2

Print “Error – Each course must have a course title and course number”

isError = true

Else

Set courseTitle

Set courseNum

For each element in prerequisites

If prerequisite not in courseList

Print “Error – Prerequisite does not exist.”

isError = true

if isError == false

Create new Course object using courseTitle, courseNum, and prerequisites

Add Course object to courses vector

}

Close file

}

void searchCourse(Vector<Course> courses, String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

void printCourses(Vector<Course> courses) {

Set curNode to head

While curNode is not null:

PRINT curNode's course information

SET curNode to curNode's next node  
}

int main() {

Declare Variables

String fileLoc

String userInput

While user does not select Exit {

PRINT “Enter ‘1’ to load courses”

PRINT “Enter ‘2’ to print all courses in order”

PRINT “Enter ‘3’ to print individual course information”

PRINT “Enter ‘9’ to exit”

Set userInput = user input

If userInput == 1

CALL getCourses(courses, fileLoc)

Break

If userInput == 2

CALL printCourses(courses)

Break

If userInput == 3

PRINT “Enter course number”

Set userInput = user input

CALL searchCourse(courses, userInput)

Break

If userInput == 9

Print “Goodbye”

Break

}

}

**Milestone 2 Pseudocode**

Define Course Structure:

String courseTitle

String courseNum

Vector<String> prereqs

Define Constants:

DEFAULT\_SIZE set to 50

Define a HashTable Class:

Private:

Node structure

Vector<Node> nodes

unsigned int tableSize

unsigned int hash(int key)

Public:

Constructor

Destructor

void Insert(Course course)

void searchCourse(HashTable<Course> courses, String courseNumber)

**HashTable Methods**

Constructor()

* Set tableSize to DEFAULT\_SIZE
* Set nodes to size of tableSize

Destructor()

- Traverse every linked list in nodes.

- Delete each node.

- Clear the nodes vector

unsigned int hash(int key)

- Calculate the has using the formula key % tableSize

- Return hash

void Insert(Course course)

- Create new node

- Use courseNum to calculate key using hash()

- If key location is empty

- Insert course at key location

- If key location is not empty:

- Traverse linked list and add course at the end of list

void searchCourse(HashTable<Course> courses, String courseNumber) {

- Use courseNumber to calculate key using hash()

- If there is a key match in the hash table:

- If the head is a match, PRINT the matching course information

- Else If there is a linked list, traverse the list and search by courseNumber

- If a match is found, PRINT the course information

}

**Other Methods**

Vector<String> getPrereqs(String fileLoc) {

Vector<String> courseList

Open file at fileLoc

While there are lines in file {

Read line

Parse line by “,”

Add course title to courseList

}

Close file

Return courseList

void getCourses(HashTable<Course> courses, String fileLoc) {

Bool isError

Vector<String> courseList

Vector<String> prerequisites

String courseNum

String courseTitle

courseList = getPrereqs(fileLoc);

Open file at fileLoc

While there are lines in file {

isError = false

Read line

Parse line by “,”

If line < 2

Print “Error – Each course must have a course title and course number”

isError = true

Else

Set courseTitle

Set courseNum

For each element in prerequisites

If prerequisite not in courseList

Print “Error – Prerequisite does not exist.”

isError = true

if isError == false

Create new Course object using courseTitle, courseNum, and prerequisites

Insert Course object into courses HashTable

}

Close file

}

void printCourses(HashTable<Course> courses) {

For i from 0 to size of nodes - 1

If nodes[i].key is not equal to UINT\_MAX

PRINT course information

SET curNode to nodes[i]

WHILE curNode.next is not null:

SET curNode to curNode.next

PRINT course information

}

int main() {

Declare Variables

String fileLoc

String userInput

HashTable courses

While user does not select Exit {

PRINT “Enter ‘1’ to load courses”

PRINT “Enter ‘2’ to print all courses in order”

PRINT “Enter ‘3’ to print individual course information”

PRINT “Enter ‘9’ to exit”

Set userInput = user input

If userInput == 1

CALL getCourses(courses, fileLoc)

Break

If userInput == 2

CALL printCourses(courses)

Break

If userInput == 3

PRINT “Enter course number”

Set userInput = user input

CALL searchCourse(courses, userInput)

Break

If userInput == 9

Print “Goodbye”

Break

}

}

**Milestone 3 Pseudocode**

Define Course Structure:

String courseTitle

String courseNum

Vector<String> prereqs

Define a node structure (Node):

Course course

Node left

Node right

Define a BinarySearchTree Class:

Private:

Node root

void addNode(Node node, Course course)

void inOrder(Node node)

Public:

Constructor

Destructor

void InOrder()

void Insert(Course course)

void searchCourse(Tree<Course> courses, String courseNumber)

**BinarySearchTree Methods**

Constructor()

* Set tableSize to DEFAULT\_SIZE
* Set nodes to size of tableSize

Destructor()

- WHILE root is not nullptr

- CALL removeNode(root, courseNumber)

void inOrder()

* CALL inOrder(root)

void Insert(Course course)

* IF root is nullptr
  + root equals new Node
* ELSE
  + CALL addNode(root, course)

void addNode(Node node, Course course)

* IF courseNumber less than node->courseNumber
  + IF node->left equals nullptr
    - node->left is set as new Node
  + ELSE
    - RECURSE addNode(node->left, course)
* ELSE
  + IF node->right equals nullptr
    - node->right equals new Node
* ELSE
  + RECURSE addNode(node->right, course)

void inOrder(Node node)

* IF node is not nullptr
  + - * CALL inOrder(node->left)
      * PRINT course information
      * CALL inOrder(node->right)

void searchCourse(BinarySearchTree<Course> courses, String courseNumber) {

* current equals root
* isFound equals FALSE
* WHILE current is not nullptr
  + IF current->courseNumber equals courseNumber
    - PRINT current->course
    - isFound equals TRUE
    - BREAK
  + ELSE IF current->courseNumber less than current courseNumber
    - current equals current->left
  + ELSE
    - current equals current->right
* IF isFound equals FALSE
  + PRINT “Course not found!”

}

**Other Methods**

Vector<String> getPrereqs(String fileLoc) {

Vector<String> courseList

Open file at fileLoc

While there are lines in file {

Read line

Parse line by “,”

Add course title to courseList

}

Close file

Return courseList

void getCourses(BinarySearchTree<Course> courses, String fileLoc) {

Bool isError

Vector<String> courseList

Vector<String> prerequisites

String courseNum

String courseTitle

courseList = getPrereqs(fileLoc);

Open file at fileLoc

While there are lines in file {

isError = false

Read line

Parse line by “,”

If line < 2

Print “Error – Each course must have a course title and course number”

isError = true

Else

Set courseTitle

Set courseNum

For each element in prerequisites

If prerequisite not in courseList

Print “Error – Prerequisite does not exist.”

isError = true

if isError == false

Create new Course object using courseTitle, courseNum, and prerequisites

Insert Course object into courses BinarySearchTree

}

Close file

}

void printCourses(BinarySearchTree<Course> courses) {

}

int main() {

Declare Variables

String fileLoc

String userInput

BinarSearchTree courses

While user does not select Exit {

PRINT “Enter ‘1’ to load courses”

PRINT “Enter ‘2’ to print all courses in order”

PRINT “Enter ‘3’ to print individual course information”

PRINT “Enter ‘9’ to exit”

Set userInput = user input

If userInput == 1

CALL getCourses(courses, fileLoc)

Break

If userInput == 2

CALL courses->inOrder()

Break

If userInput == 3

PRINT “Enter course number”

Set userInput = user input

CALL searchCourse(courses, userInput)

Break

If userInput == 9

Print “Goodbye”

Break

}

}

**Runtime Analysis Tables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Structure** | **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Vector | Open file at fileLoc | 1 | 1 | 1 |
| While there are lines in file (read line) | 1 | n | n |
| Parse line | 1 | n | n |
| Add course to vector | 1 | n | n |
| **Total Cost** | | | | 3n + 1 |
| **Runtime** | | | | O(n) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Structure** | **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Hash Table | Open file at fileLoc | 1 | 1 | 1 |
| While there are lines in file (read line) | 1 | n | n |
| Parse line | 1 | n | n |
| Calculate hash | 1 | n | n |
| Insert into hash table | 1 | n | n |
| **Total Cost** | | | | 4n + 1 |
| **Runtime** | | | | O(n) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Structure** | **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Binary Search Tree | Open file at fileLoc | 1 | 1 | 1 |
| While there are lines in file (read line) | 1 | n | n |
| Parse line | 1 | n | n |
| Insert into binary search tree | 1 | n | n |
| **Total Cost** | | | | 3n + 1 |
| **Runtime** | | | | O(n) |

**Analysis and Recommendations**

Three distinct data structures were considered for the software project in support of the Computer Science department at ABCU. The reviewed data structures included the vector, hash table, and binary search tree. After careful consideration of the advantages and disadvantages of each structure, I recommend the use of vectors for the software project.

The vector data structure has several advantages, including fast access to elements by index relative to the other data structures. Vectors are also easy to implement in comparison to hash tables and binary search trees. In contrast, hash tables are subject to collisions, which can degrade performance. Binary search trees, on the other hand, are not subject to collisions, but trees can become unbalanced, which leads to degraded performance. Binary search trees are also more complex to implement and maintain compared to vectors.

Overall, the worse-case scenario runtime performance for each data structure in this project would be very similar (O(n)). However, vectors are recommended due to their ease of use and performance advantage, particularly in relation to fast access to data elements. The relative simplicity of vectors will also shorten the overall development timeline of the software project.