Blake Kemp

December 4, 2022

CS 300

Southern New Hampshire University

**Project one Pseudocode and runtime analysis**

**Reading the File:**

Use fstream to be able to open the file

Make call to open file

IF return value is -1, File is not found

ELSE File is found

WHILE it is not the end of file

IF there are less than two values in a line, RETURN error

ELSE Read parameters

IF there is a third or more parameters

IF third or more is in the first parameter

CONTINUE

ELSE RETURN error

CLOSE FILE

**VECTOR:**

class Course {

int courseNumber

string courseTitle

vector<string> preReqs

}

courseSearch(vector<Course>courses, string courseNum) {

Create empty course

FOR(each course in courses) {

IF(current courseNumber == courseNum) {

Return course

Return empty

}

*//Open the file and store the information within the vector*

vector<string> openReadFile(string filename) {  
 INITIALIZE vector<string>

INTIALIZE string line to hole single line

INITIALIZE ifstream instream to get the contents of the file

OPEN file with instream using filename

IF(unable to open file)

OUTPUT “Unable to open file”

PULL line from the instream until all information is received

PUSH line to back

CLOSE file

RETURN information

}

*//Store the information from the file and create a course for each line to store in vector*

vector<Course\*>createSchedule(vector<string>contents) {  
 INITIALIZE vector<Course\*>

INITIALIZE stringstream lineStream

INITIALIZE string token to store each word from the line

INITIALIZE integer count to track the tokens per line

FOR(each new course)

SET count to 0

CREATE new Course\* course

FILL lineStream with the line contents

PULL token from lineStream till end of the line

IF(count equals 0)

course courseNumber equals token

INCREMENT count

ELSE IF (count equals 1)

courseTitle equals token

INCREMENT count

ELSE

IF(token equals courseNumber already in courses)

PUSH token to the back of course preReqs

ELSE

OUTPUT “Pre-reqs must be a previous course”

INCREMENT count

IF(count less than 2)

OUTPUT “Error with the file format, Check course number and Title”

EMPTY lineStream

PUSH course to back of courses

RETURN courses

}

**HASHTABLE:**

INITIALIZE course Vector vector<Node> nodes

CREATE HashTable Class

CREATE insert method to insert items into the table

LOOP through the file

WHILE not end of the file

FOR each line in file

FOR 1st and 2nd value

CREATE temp item to hold the values

IF a 3rd value exists

ADD to current value

CALL insert method for each value

**TREE:**

INITIALIZE Course Structure struct Course

LOOP through file

WHILE not End of file

FOR each line in file

FOR first and second value

ADD Course ID, and Course Name

IF third value exists

ADD prereq until newline is found

**Create Tree and add Nodes:**  
DEFINE Binary Tree Class

CREATE a root that points to NULL  
CREATE insert method

IF root is NULL

Current course equals root

ELSE IF course number is less than root

ADD to left

IF left equals NULL

ADD course number

ELSE

IF course number is less than leaf

ADD left

IF course number is greater than leaf

ADD right

ELSE IF course number is greater than root

ADD to right

IF right equals NULL

Add course number

ELSE

IF course number is less than leaf

ADD left

IF course number is greater than leaf

ADD right

**Print the course information and prerequisites:**

**VECTOR:**

CREATE method void printCourseInfo(vector<Course>courses, string courseId) {

Get input from the user

WHILE the vector is not empty

IF the input matches courseId

OUTPUT course.courseId << OUTPUT course.name

WHILE prereq is equal to true

OUTPUT course.prereq

**HASTHABLE:**

CREATE method void printCourseInfo(Hashtable<Course>courses, string courseId)

Get input from user

Assign input to a key

Assign node to node.at(key)

If the current node equals the key

OUTPUT course.courseId << output course.name

WHILE prereq is equal to true

OUTPUT course.prereq

**TREE:**

CREATE method void printCourseInfo(Tree<Course>courses, string courseId)

Ask user for input.

Assign current node to root

WHILE current does not equal null

IF course.courseId matches the current

RETURN current and OUTPUT course.courseId << OUTPUT course.name

WHILE prereq is equal to true

OUTPUT course.prereq

IF coursed is less than the root

SET current to the left

ELSE set the current to the right

**MENU:**

SET choice to 0

CREATE while loop for menu

WHILE choice is not equal to 0

OUTPUT menu choices

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

CREATE switch(case)

Case 1: loadCourses(CourseFile, dataStructure) FIXME: Use the data structure chosen

Case 2: printSorted(courses) call the function to print sorted class list

Case 3: printCourseInfo(CourseTitle)

Case 4: Close the program

**Print the sorted list:**

**VECTOR:**

CREATE a sorted print method named printSorted(courses)

CREATE partition method int partition(vector<Courses>& couses, int begin, int end)

SET lowIndex to the first element and the highIndex to the last element

SET the midpoint to lowIndex added to (highIndex – lowIndex) / 2

SET pivot to the midpoint

DECREMENT highIndex while the pivot is less than highIndex

SWAP the lower values to the left of pivot and higher values to the right

SET tempVal equal to lowIndex

SET lowIndex equal to highIndex

SET highIndex equal to tempVal

CREATE quickSort method void quickSort(vector<Course>& courses, int begin, int end)

SET mid equal to 0

SET lowIndex equal to begin

SET highIndex equal to end

IF begin is greater than or equal to end then RETURN

SET lowEndIndex to partition(courses, lowIndex, highIndex)

Make a recursive call to quickSort

quickSort(courses, lowIndex, lowEndIndex)

quickSort(courses, lowEndIndex + 1, highIndex)

CREATE display course method void displayCourses(Course course)

OUTPUT course.courseId, course.name, course.prereq

LOOP through the vector to display the courses

CREATE inOrder method void BinarySearchTree::inOrder(Node\* node)

IF node does not equal null

Look to the most left side first

inOrder node to left

OUTPUT course.courseId, course.name, course.prereq

Look at the next right leaf

OUTPUT course.courseId, course.name, course.prereq­­

**Runtime analysis for reading the file and creating course objects:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **Times executed** | **Total** |
| Create Vector | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **Times Executed** | **Total** |
| Create vector | 1 | 1 | 1 |
| While prereq exists | 1 | n |  |
| Append the prereq | 1 | n | n |
| Pushback course item | 1 | n | n |
| TOTAL COST | | | 5n + 1 |
| RUNTIME | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **HASHTABLE** | **Line Cost** | **Times Executed** | **Total** |
| Create Hashtable | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| Create key for courses | 1 | n | n |
| IF no entry is found for key | 1 | n | n |
| Assign node to key | 1 | n | n |
| ELSE | 1 | n | n |
| Assign the old node to key UNIT\_MAX, set that to key, set old node to course and old node next to nullptr | 4 | n | 4n |
| Else | 1 | n | n |
| Find the next open node | 1 | n | n |
| Add the new node to the end | 1 | n | n |
| FOR each line in the file | 1 | n | n |
| Create vector course item | 1 | n | n |
| WHILE prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert Course item | 1 | n | n |
| TOTAL COST | | | 16n + 1 |
| RUNTIME | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **TREE** | **LINE COST** | **Times Executed** | **TOTAL** |
| Create tree | 1 | 1 | 1 |
| Add node method | 0 | 0 | 0 |
| IF root equals null add root | 1 | 1 | 1 |
| IF node is less than root add to left | 1 | n | n |
| IF there is no left node | 1 | n | n |
| This node turns into left | 1 | n | n |
| IF node is greater than root add to right | 1 | n | n |
| IF there is no right node | 1 | n | n |
| This node turns into right | 1 | n | n |
| FOR each line in the file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert Course Item | 1 | n | n |
| TOTAL COST | | | 11n + 2 |
| RUNTIME | | | O(n) |

Each data structure offers benefits and drawbacks depending on the program's needs. The vector method has the benefit of reading the file and inserting the course objects more quickly than any other method. It is a very simple method where each item is just added to the end of a vector when the file is parsed. Despite having the same O(n) notation, the runtime of the three methods was the quickest at 5n+1. The inability to search the list for a particular course is a drawback of utilizing a vector. Each element in the vector must be checked by the software until a match is made.

Hash tables provide the benefit of having speedy list searches. The locations of a certain course can be identified by generating a key, which makes it simple to search for and print. When building the initial list, it takes longer because a key must be made for each item and a place sought to insert each course. Hash tables are also not well suited for sorting. The table cannot be sorted by itself. Each value must be extracted, sorted, and printed before an alphanumeric list of all courses can be printed. Thus, it is probably not the optimal data format for this program when it comes to portability.

The benefit of binary trees is that they can be searched more quickly than vectors. Knowing the search path makes it very simple to descend the tree until the value is discovered. It is speedier than a vector but not as simple as a hash table. In the worst situation, the tree would have to look through every component if it had no more leaves. In this case, the search time would be O(h), where h is the tree's height.

Finally, for this job, I would advise a vector sort. I believe that the client will value the ability to quickly sort and print the whole catalog. Additionally, the time lost during the search is not as awful as it could be given the benefit of the sort. The vector, in my opinion, is the finest choice overall.