Jason O’Connell

8/7/2022

Pseudocode Analysis Project Module 6

[Prompt 1] Opening and reading the text file (CSV format) to check for errors

void checkFileIntegrity FUNCTION {

IF file is empty {

THROW file empty exception

}

PRE-REQUISITE vector set to empty

COURSE-CODE vector set to empty

FOR each line in file {

COUNT the arguments in that line

IF number of arguments LESS THAN 2 {

THROW invalid file format exception

}

ADD first argument to COURSE-CODE vector

IF number of arguments greater than 2 {

FOR argument in remaining arguments {

IF argument not in PRE-REQUISITE vector {

ADD argument to PRE-REQUISITE vector

}

}

}

}

FOR course in PRE-REQUISITE vector {

IF course not in COURSE-CODE vector {

THROW missing course exception

}

}

}

// [Prompt 2] Course class for building course objects with hash table capabilities

CLASS Course {

PRIVATE:

String course code

String course name

Vector of pre-requisites

PUBLIC:

Course(String code, String name, Vector preReqs) {

CREATE a Course object from passed parameters

}

}

// [Prompt] Vector data structure for storing course objects

// [Prompt] Hash table class for storing course objects

CLASS BinarySearchTree {

PRIVATE:

Node object/struct that will contain a course object, a left pointer, and a right pointer

Pointer to the root node

PUBLIC:

FUNCTION to print out all courses and preReqs

}

// [Prompt] Menu function

While TRUE {

IF input equals 1 {

LOAD data into the data structure

}

ELSE IF input equals 2 {

PRINT the ordered list

}

ELSE IF input equals 3 {

PRINT the course title and prerequisites for that course

}

ELSE IF input equals 4 {

BREAK

}

}

// [Prompt 2] Loading data into the tree

LoadData FUNCTION{

FOR each line in file{

Set course code to first argument in current line

Set name to second argument in current line

Set prerequisites vector to empty vector

FOR each remaining argument in current line{

ADD prerequisite to prerequisites vector

}

CREATE Course class via constructor(

Parameter code

Parameter name

Parameter prerequisites

)

CREATE new node whilst add the course to that node

If the new node is larger{

If left node exists{

Recurse down the left node

}

Else{

New node becomes left node

}

}

Else{

If right node exists{

Recurse down the right node

}

Else{

New node becomes right node

}

}

}

}

// [Prompt 3] Print out function for tree

printCourse FUNCTION(Course c){

Recursively LOOP through the nodes{

IF current node is not empty{

IF Course in the current node equals Course c{

PRINT out the course and its prerequisites

}

}

}

}

Big O analysis for worst case search tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for each line in file** | 1 | n | n |
| **Set course code to first line in file** | 1 | 1 | 1 |
| **Set name to second argument in current line** | 1 | 1 | 1 |
| **Set prereqs vector to empty vector** | 1 | 1 | 1 |
| **For each remaining argument in current line** | 1 | n | n |
| **Add prereq to prereqs vector** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Create a new node** | 1 | n | n |
| **If/Else the new node is larger** | 1 | n | n |
| **If left/right node exists** | 1 | n | n |
| **Recurse down the left/right side** | 1 | n | nLog(n) |
| **Total Cost** | | | nLog(n) +7n + 3 |
| **Runtime** | | | O(nLog(n)) |

// [Prompt 2] Loading data into the Hash table

LoadData FUNCTION{

FOR each line in file{

Set code to first argument in current line

Set name to second argument in current line

Set prerequisites vector to empty vector

FOR each remaining argument in current line{

ADD prerequisite to prerequisites vector

}

CREATE Course object via constructor(

Parameter code

Parameter name

Parameter prerequisites

)

CALCULATE a key for the given course

SET currentNode to the node at the key

IF currentNode does not exist{

Assign currentNode to the key position

Insert the course into this node

}

ELSE IF currentNode is not used{

Assign the next node to null

Insert the course into this node

}

ELSE {

WHILE old node is not empty{

Go to the next node

}

Add new node to end

Insert course into this node

}

}

}

// [Prompt 3] Print out function for hash table

printCourse FUNCTION(Course c){

CREATE the key for the given course c

SET currentNode to the node at the key

IF currentNode is not empty{

Print out course information

Print out course prerequisites

}

}

Big O analysis for worst case hash table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for each line in file** | 1 | n | n |
| **Set course code to first line in file** | 1 | 1 | 1 |
| **Set name to second argument in current line** | 1 | 1 | 1 |
| **Set prereqs vector to empty vector** | 1 | 1 | 1 |
| **For each remaining argument in current line** | 1 | n | n |
| **Add prereq to prereqs vector** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Calculate a key for the given course name** | 1 | n | n |
| **Set current node to the node at the key** | 1 | n | n |
| **If current node does not exist** | 1 | n | n |
| **Assign current node to the key position** | 1 | n | n |
| **Insert the course into this position** | 1 | n | n |
| **Else if current node is not used** | 1 | n | n |
| **Assign the next node to null** | 1 | n | n |
| **Insert the course into this node** | 1 | n | n |
| **Else** | 1 | n | n |
| **While old node is not empty** | 1 | n^2 | n^2 |
| **Go to the next node** | 1 | n^2 | n^2 |
| **Add new node to end** | 1 | n | n |
| **Insert course into this node** | 1 | n | n |
| **Total Cost** | | | 3n^2 + 15n + 3 |
| **Runtime** | | | O(n^2) |

//[Prompt 2] Loading data into the vector

Vector loadData FUNCTION {

Vector courses set to empty vector

FOR each line in file {

Set code to first argument in current line

Set name to second argument in current line

Set prerequisites vector to empty vector

FOR each remaining argument in current line{

ADD prerequisite to prerequisites vector

}

CREATE Course object via constructor(

Parameter code

Parameter name

Parameter prerequisites

)

ADD newly created Course to courses vector

}

}

//[prompt 3] Searching the vector for a specific course

Void printCourse FUNCTION (Vector courses, Course c ) {

FOR each course in courses {

IF course is equal to c {

Print course information

Print course prerequisites

}

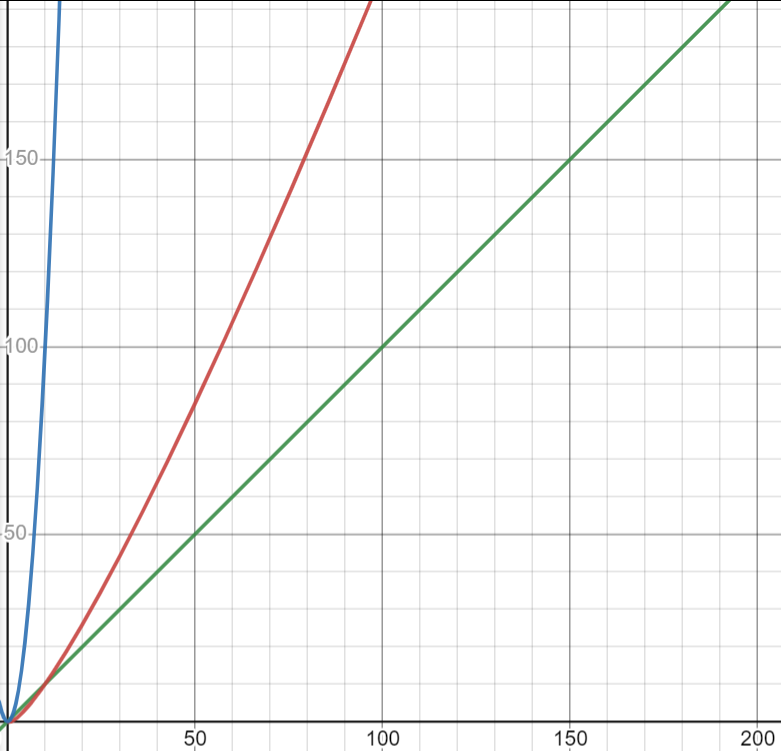
}

}

Big O analysis for worst case vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create an empty vector** | 1 | 1 | 1 |
| **for each line in file** | 1 | n | n |
| **Set course code to first line in file** | 1 | 1 | 1 |
| **Set name to second argument in current line** | 1 | 1 | 1 |
| **Set prereqs vector to empty vector** | 1 | 1 | 1 |
| **For each remaining argument in current line** | 1 | n | n |
| **Add prereq to prereqs vector** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Add new course object to vector** | 1 | n | n |
| **Total Cost** | | | 5n + 4 |
| **Runtime** | | | O(n) |

Overall Analysis



Legend: green: Vector [O(n)], red: Tree [O(nlog(n))], blue: Hash table [O(n^2)]

In the above graph of the time complexities for each, we can see that a vector is clearly the most efficient with an increase in n.

An argument can be made that, in lower sizes of n, particularly less than about 15, the hash table is actually the more efficient option, but not by much. As the size of n increases, we can see that the efficiency starts to diverge from the vector because of the amount of recursion that has to be done as the tree expands. This chart tells us that a small tree is quite efficient. But as n grows, tree size is going to grow, and the algorithm becomes less efficient than other methods.

Hash table efficiency is largely determined by implementation. The has table uses a has function to generate a key for every entry. Hash functions can be very simple like the modulus function used in the hash table assignment. But unfortunately, simple hash functions are going to lead to collisions as you increase n. You can work around collision by having overflow buckets and such but that leads to nested for loops, hence the n^2. If you were to have a more complex hashing function up front, that’s going to cost you in executions. Pick your poison.

On a final note, it’s worth mentioning that a vector is quite slow at inserting, removing, and accessing data compared to the other methods because it has to shift all of the data to maintain the in-order vector concept in memory.

The data file indicated that size of n would probably be somewhere around 20 to 50. In this scenario, I would recommend using the vector as the data structure because of its effieicency compared to the other structures at in that range of n