Robert Lulashi

Southern New Hampshire University

CS 300 Project 1

4/9/2023

// \*\*\*\*\*\*Vector Pseudocode\*\*\*\*\*\*

// Opens file with courses outlined

// Read each line

// Parse out the course # title & prerequisites

// initialize mandatory set of prerequisites (CSI100 (introduction to computer science)  
MATH 201(discrete mathematics))\*\*(Just a sample of prerequisites, full list will be loaded from a CSV list)

// initialize courses   
(CSI101(introduction to progamming c++ prereq CSI100)  
CSI200(data structures prereq CSI101) CSI300(introduction to algorithms prereq CSI200, MATH 201)  
CSI301(advanced programming in c++ prereq CSI101)  
CSI350(operating systems prereq CS300)  
CSI400(large software development prereq CSI301, CSI350)\*\*\* full list will be imported from CSV file  
// Check for errors in file format & make sure formatting is proper

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

// If class has prerequisites cannot be registered else those prerequisites are met

//Else enroll if no prerequisites are required

// Create course objects and store them in the vector

// Search vector for course information and any prerequisites for a specific course

void printSampleSchedule(Vector<Course> courses) {

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 printCourseInformation(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

// Close

//\*\*\*\*\*\* Hash table pseudocode \*\*\*\*\*\*  
// Opens file with courses outlined

// Read each line

// Parse out the course # title & prerequisites   
// initialize mandatory set of prerequisites (CSI100 (introduction to computer science)  
MATH 201(discrete mathematics)) (Sample of prerequisite courses, all will be imported from CSV file ABCU\_Advising\_Program\_Input.csv)  
int numPrerequisiteCourses(Hashtable<Course> courses) { totalPrereq = hashtable

// add prereq to hashtable totalPrereq

// initialize courses   
(CSI101(introduction to progamming c++ prereq CSI100)  
CSI200(data structures prereq CSI101) CSI300(introduction to algorithms prereq CSI200, MATH 201)  
CSI301(advanced programming in c++ prereq CSI101)  
CSI350(operating systems prereq CS300)  
CSI400(large software development prereq CSI301, CSI350) …. (Not full list, will be imported from CSV file)

}

// Check for errors in file format & make sure formatting is proper

// If class has prerequisites cannot be registered else those prerequisites are met   
//Else enroll if no prerequisites are required   
  
// Create course objects and store them in the hashtable

//Ensure nullptr is set proper   
//Detect for improper collision of data   
// Search for course information and any prerequisites for a specific course

// void printSampleSchedule(Hashtable<Course> courses) {

for all key print key course

if prerequisites exist print all prerequisites

}

// void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

for all key print key course

if prerequisites exist print all prerequisites

// Close

// \*\*\*\*\*Binary search tree pseudocode \*\*\*\*\*\*  
// Opens file with courses outlined

// Read each line

// Parse out the course # title & prerequisites   
// initialize mandatory set of prerequisites (CSI100 (introduction to computer science)  
MATH 201(discrete mathematics)) (Sample of prerequisite courses, all will be imported from CSV file)  
// initialize courses

// int numPrerequisiteCourses(Tree<Course> courses) {node\* left == nullptr

Node\* right == nullptr

(CSI101(introduction to progamming c++ prereq CSI100)  
CSI200(data structures prereq CSI101) CSI300(introduction to algorithms prereq CSI200, MATH 201)  
CSI301(advanced programming in c++ prereq CSI101)  
CSI350(operating systems prereq CS300)  
CSI400(large software development prereq CSI301, CSI350) …. (Not full list, will be imported from CSV file)

}

// Check for errors in file format & make sure formatting is proper

// If class has prerequisites cannot be registered else those prerequisites are met   
//Else enroll if no prerequisites are required   
// Create course objects and store them in the binary search tree

//Ensure root == nullptr   
//Detect for improper collision of data   
// Search for course information and any prerequisites for a specific course

void printSampleSchedule(Tree<Course> courses) {

if prereq exist in left node print left node

if prereq exist in right node print right node

print all courses

}

void printCourseInformation(Tree<Course> courses, String courseNumber) {

if prereq exist in left node print left node

if prereq exist in right node print right node

print all courses

}  
// Close

| **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** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

Evaluation of each method

Vector:

Adding to a vector has a worst case run time of O(1). If the vector needs to be resized this time changes to O(n), adding these together changes our worst case run time to O(n^2).

Advantages

Adding to the end of the list is simple and fast. Also, elements stored in a vector are kept with their number so therefore they are quick to search for.

Disadvantages

Resizing a vector is not cost effective and changes made anywhere but the end of a vector are time consuming.

Hash Table:

Adding to a hash table has a worst case run time of O(1) if all keys are displaced evenly. If the hash table assigns all keys to the same value this changes the worst case run time to O(n).

Advantages

Adding and searching items are consistent in their time, compared to the other two methods, a faster time on average. Resizing is also quite fast and not as resource exhausting as a vector.

Disadvantages

Collisions can be troublesome but if good practices are in place, it balances it out. Organization is also lost because items are assigned to different value keys and can share a spot therefore losing a unique identity.

Binary Search Tree:

Adding to a binary search tree has a worst case run time of O(log n). If the tree becomes unbalanced, the worst case run time will become to O(n).

Advantages

Changes to the tree are faster than the other methods on average and the order is also preserved.

Disadvantages

Binary search trees require much more memory than the other two methods and the worst case run time really hurts if the tree becomes unbalanced.

I would recommend a hash table for ABCU to use. It is faster than a vector but more consistent than a binary search tree. The tree would be a great tool but if it becomes unbalanced it ties for a hash table and it is easier to correct for a hash table as it can grow as needed. Collisions are an issue but taking precautions should minimize this risk.