Project One: ABCU Course Planner

Pseudocode and Big-O Analysis

Emily Domoracki | February 2023

Professor Cory Thoma | CS-300 23EW3 SNHU

Below, I’ve put together pseudocode for the ABCU Course Planner application, including three different data structures that could be used to store course data. The advantages of using a vector are that it is simple to implement and very quick to load the data into the structure. However, printing an alphanumerically sorted course list or searching for a course is much less efficient than a binary search tree. A binary search tree has the advantage of quickly producing an alphanumerically sorted list or search results. A drawback to the binary tree is it’s slow to initially load the data, but it’s not as complex to implement as a hash table. If producing an ordered list was not a priority for this application, I might suggest using a hash table. A hash table will return a searched element with optimal speed. However, it is more complex than either of the data structures to implement and it does not store an ordered list. Based on the needs of ABCU’s advisor, I my recommendation is to implement a binary search tree data structure. Loading the course list will take longer than the other two structures, but the functionality of it thereafter will exceed both a vector and a hash table.

**//MAIN METHOD**

Main()

//initialize local objects for vector data structure

Vector <Course> courses

//initialize local objects for hash table data structure

HashTable\* courseTable = new HashTable()

//initialize local objects for bst data structure

BinarySearchTree\* bst = new BinarySearchTree()

Course course

String filePath, courseNumber

Int choice = 0

While (choice does not equal 9)

print “Main Menu”

print “1. Load Course List”

print “2. Print Course List”

print “3. Find Course by Course Number”

print “9. Exit”

user input = choice

Switch (choice)

case 1:

if (string filePath is empty)

print “Please enter course list file path”

user input = filePath

//for vector data structure

LoadFile(string filePath)

//for hash table data structure

LoadFile(string filePath, HashTable\* hashtable)

//for bst data structure

LoadFile(string FilePath, BinarySearchTree\* bst)

break

case 2:

//for vector data structure

selectionSort(vector <Course> courses)

For (integer i starting index at zero, while i < the size of vector courses,

increase index by 1)

Print courseNumber, courseName

//for hash table data structure

Create vector <Course> courses

For (integer i starting index at zero, while i < the size of courseTable,

increase index by 1)

Create Node node = reference course at i;

if (node key is not null or initialized with default int)

Create tempCourse = node’s course

Pushback tempCourse in courses vector

SelectionSort(vector <Course> courses)

//for bst data structure

Call inOrder() function for local bst

Break

Case 3:

print “Please enter the course number: ”

user input = courseNumber

convert course number to uppercase

//for vector data structure

FindCourse (vector <Course> courses, string courseNumber)

//for hash table data structure

course = local hashTable points to FindCourse(string courseNumber)

if (course’s courseNumber is not empty)

print courseNumber, courseName

else

print “course not found

//for bst data structure

course = local bst points to FindCourse(string courseNumber)

if (course’s courseNumber is not empty)

print courseNumber, courseName

else

print “course not found”

break

print “Goodbye”

exit program: return 0

**//LOAD AND PARSE FUNCTION**

void LoadFile(string filePath)

Create instance of fstream: fstream fin

Open file: fin.open( filePath)

Create string object to hold lineRead and lineParsed

Create vector to hold lineParsed: vector <string> parsedObjects

Create vector to hold row as single element: vector <vector<string>> fileParsed

If (file is open)

While (file has no errors)

while (reading current line to lineRead)

clear objectsParsed vector

create stringstream: ss(lineRead)

while (parsing lineRead by char ‘ , ‘ and storing in lineParsed)

add lineParsed to objectsParsed vector

if (objectsParsed vector size >= 2)

add row vector to fileParsed vector

else

print “File format error”

//for vector data structure only

Create vector to hold Course objects: vector <Course> courses

For (integer i starting index at zero, while i < the size of fileParsed vector,

increase index by 1)

create course object: Course course

courseNumber = first element in row i of fileParsed

courseName = second element in row i of fileParsed

if (the size of vector fileParsed == 3)

coursePrerequisiteOne = third element in row i of fileParsed

if (the size of the vector fileParsed == 4)

coursePrerequisiteOne = third element in row i of fileParsed

coursePrerequisiteTwo = fourth element in row i of fileParsed

//for vector data structure

add course to vector courses

//for hash table data structure

Hash table points to insert(course)

//for bst data structure

Bst points to insert(course)

Else

print “failed to open file”

Close file

**//HASH TABLE FUNCTIONS**

Void Insert(course)

Create key for course:

key = call hash function(convert to integer(courseNumber converted to c-string))

check key for existing node: Node\* oldNode = reference node at key

If (oldNode == null)

Create new Node\* newNode initialized with (course, key)

Else if (oldNode exists but is empty)

Assign oldNode key with key

Assign oldNode course with course

Point oldNode to null

Else

While (if the next node is not null)

oldNode = next node

Add newNode initialized with (course, key) to end of list

Int hash(int key)

Return key mod hash table size

HashTable FindCourse (string courseNumber)

Create instance of Course course

Create key for course:

key = call hash function(convert to integer(courseNumber converted to c-string))

check key for existing node: Node\* node = reference node at key

If ( node is null or empty)

Return empty course

If (node is not null or default initialized int

and node courseNumber matches passed courseNumber)

Return this node’s course

While (node does not equal null)

If (node key is not default initialized int and

and node courseNumber matches passed courseNumber)

Return this node’s course

Node = next node

**//BST FUNCTIONS**

Void Insert (Course course)

If (root is null)

root = new node initialized with course

Else

call function addNode with this root and course as param

Void FindCourse (string courseNumber)

Current node = root

While (current is not null)

if (current node’s courseNumber is equal to key courseNumber)

return current node’s course

if (key courseNumber < current node’s courseNumber)

current = node to the left of current

else

current = node to the right of current

Create empty course

Return empty course if courseNumber not found

**//VECTOR FUNCTIONS**

void FindCourse (vector <Course> courses, string courseNumber)

string keyCourseNumber = passed courseNumber

For (integer i starting index at zero, while i < the size of vector courses,

increase index by 1)

If (courseNumber in course at index i == keyCourseNumber)

print courseNumber, courseName

if (coursePrerequisiteOne is not empty)

print coursePrerequisiteOne

if (coursePrerequisiteTwo is not empty)

print coursePrerequisiteTwo

Return

Print “course not found”

Return

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Create Vector Data Structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Vector <Course> courses** | 1 | 1 | 1 |
| **For (int I=0; I < fileParsed.size(); ++I)** | 1 | n | n |
| **Course course** | 1 | 1 | 1 |
| **Course.courseNumber = fileParsed[I][0]** | 1 | n | n |
| **Course.courseName = fileParsed[I][1]** | 1 | n | n |
| **If (fileParsed.size() == 3)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **If (fileParsed.size() == 4)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **Course. CoursePrerequisiteTwo = fileParsed[I][3]** | 1 | n | n |
| **Add course to vector courses** | 1 | n | n |
| **Total Cost** | | | 9n + 2 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Create Hash Table Data Structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **For (int I=0; I < fileParsed.size(); ++I)** | 1 | n | n |
| **Course course** | 1 | 1 | 1 |
| **Course.courseNumber = fileParsed[I][0]** | 1 | n | n |
| **Course.courseName = fileParsed[I][1]** | 1 | n | n |
| **If (fileParsed.size() == 3)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **If (fileParsed.size() == 4)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **Course. CoursePrerequisiteTwo = fileParsed[I][3]** | 1 | n | n |
| **Point insert(course) to hash table** | O(n) | n | n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Create BST Data Structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **For (int I=0; I < fileParsed.size(); ++I)** | 1 | n | n |
| **Course course** | 1 | 1 | 1 |
| **Course.courseNumber = fileParsed[I][0]** | 1 | n | n |
| **Course.courseName = fileParsed[I][1]** | 1 | n | n |
| **If (fileParsed.size() == 3)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **If (fileParsed.size() == 4)** | 1 | n | n |
| **Course. CoursePrerequisiteOne = fileParsed[I][2]** | 1 | n | n |
| **Course. CoursePrerequisiteTwo = fileParsed[I][3]** | 1 | n | n |
| **Point insert(course) to bst** | O(log n) | n | n |
| **Total Cost** | | | O(log (n)) + 8n + 1 |
| **Runtime** | | | O(log (n)) |