**Module 6-2: Project One Pseudocode and Analysis**

Ryan McFarland

Department of Computer Science, Southern New Hampshire University

CS 300: DSA: Analysis and Design

Professor Andujar

August 8, 2021

**Vector**

**//pseudocode for loading data into the vector data structure**

Ifstream myfile

Myfile.open(“name.csv”)

If file open fail

Print error

courseNames = second string of each line

While(myfile.good() and not end of file)

String line;

Getline(myfile, line, ‘\n’)

Course(line)

**//pseudocode for creating course objects and storing to vector**

Vector<Course> courses

Course(line)

String name

Course \*courseName = new Course

For each string

courseName-> num = first parsed string

courseName-> name = second parsed string

///////ensures at least two parameters on each line/////////////

If no num or name

Error

return

For each remaining string

/////////////ensures that any prerequisite exists as a course////////

If prerequisite name found in courseNames

Prerequisites = remaining parsed string

courseName->prerequisite = prerequisite

Prerequisites ++

Courses.push\_back(\*courseName)

**//pseudocode for searching and printing a specific course and printing course information out**

Cin << search term

For each course.at(i) with I less than course length

If course name = search term

Print course information

print prerequisites course information

**//Pseudocode for menu**

**//load data structure**

Ifstream myfile

Myfile.open(“name.csv”)

If file open fail

Print error

courseNames = second string of each line

While(myfile.good() and not end of file)

String line;

Getline(myfile, line, ‘\n’)

Course(line)

**//print course list, prints alphanumerically ordered list of all courses**

selectionSort(courses){

unsigned int min

for i<courses.size() ++i

min = i

for j=i+1 j<courses.size() ++j

if courses.at j < courses.at I

min = j

if min isn’t I then swap bids at I and min

} for i<courses.size() print all courses.at(i)

**//print course, prints course title and prerequisites for given course**

If course.at(i) == search term

Cout<< course->title << course-> prerequisites

**//exit**

return

**//pseudocode for print list in alphanumeric order**

//sort the courses by course number from low to high

selectionSort(courses)

//print the sorted list to display

Do Cout << sorted course.at(i) while i<courses.size()

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each string** | 1 | n | n |
| **courseName-> num = first parsed string** | 1 | 1 | 1 |
| **courseName-> name = second parsed string** | 1 | 1 | 1 |
| **If no num or name**  **Error**  **return** | 1 | 1 | 1 |
| **For each remaining string** | 1 | n | n |
| **If prerequisite name found in courseNames** | 1 | n | n |
| **Prerequisites = remaining parsed string** | 1 | n | n |
| **courseName->prerequisite = prerequisite** | 1 | n | n |
| **Prerequisites ++**  **Courses.push\_back(\*courseName)** | 1 | n | n |
| **Total Cost** | | | 6n + 3 |
| **Runtime** | | | O(n) |

**Hashtable**

**//pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors**

Ifstream myfile

Myfile.open(“name.csv”)

If file open fail

Print error

courseNames = second string of each line

While(myfile.good() and not end of file)

String line;

Getline(myfile, line, ‘\n’)

Course(line)

**//pseudocode to show how to create course objects and store them in the appropriate data structure**

Struct Course

Subject/number

Title

Class hashtable

Struct node

course

key

next

nodeconstructors

vector<Node> nodes(numberOfBuckets)

hash(key)

return key mod numberOfBuckets

Insert(course)

Key = hash(course sub/name)

Determine correct node position and place

**//pseudocode that will print out course information and prerequisites**

For I less than size of table

Look at each bucket that isn’t empty

If value isn’t default or null

Print the course information for each course in the container

**//Pseudocode for menu**

**//load data structure**

Ifstream myfile

Myfile.open(“name.csv”)

If file open fail

Print error

courseNames = second string of each line

While(myfile.good() and not end of file)

String line;

Getline(myfile, line, ‘\n’)

Course(line)

**//print course list, prints alphanumerically ordered list of all courses**

Vector<courses> course = new vector

For each hash key

Add all node entries to course vector

Sort vector by course id

Cout << vector.at(i) for all I’s less than vector.size()

**//print course, prints course title and prerequisites for given course**

Get course by hashing course id and looking in matching node for course

When course found return course and output all course info including prerequisites

**//exit**

return

**//pseudocode for print list in alphanumeric order**

//sort the courses by course number from low to high

For each bucket

Add courses in node to a vector or list

Sort list or vector

**//print the sorted list to display**

Cout << vector.at(i) until size reached

| **Hashtable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **vector<Node> nodes(numberOfBuckets)** | 1 | n | n |
| **hash(key)** | 1 | n | n |
| **return key mod** | 1 | n | n |
| **numberOfBuckets** | 1 | n | n |
| **Insert(course)** | 1 | n | n |
| **Key = hash(course sub/name)** | 1 | n | n |
| **Determine correct node position and place** | 1 | n | n |
| **Total Cost** | | | 7n |
| **Runtime** | | | O(n) |

**Tree**

//read file, each line is a course

fileIn{

getline = new course

classsubj/num = first string

courseName = next string

if either is null then throw error

}

Struct course{

String classsubj/num

String courseName

PrereqList{strings after first two go here}

}

Struct node{

root

left

right

}

Class bst{

Add

Remove

Search

inorder

}

void printSampleSchedule( node, course) {

for each node in schedule print course

}

void printCourseInformation(node, courseName) {

start at root node

check if match

if no match and this->courseName>courseName

look on right for value

if course found return course

if no match and this->courseName<courseName

look on left for value if found return

}

**//Pseudocode for menu**

**//load data structure**

fileIn{

getline = new course

classsubj/num = first string

courseName = next string

if either is null then throw error

}

**//print course list, prints alphanumerically ordered list of all courses**

inOrder(Node\* node){

if node not nullptr

call inorder recursively with the node.left

cout << course information

inOrder recursively with node.right

}

**//print course, prints course title and prerequisites for given course**

search(string courseId){

Node\* current = root

While current not nullptr

If current matches return current

If course is smaller than current set current to left

Else course is larger set current to right node and keep looking

}

**//exit**

close

**//pseudocode for print list in alphanumeric order**

**//sort the courses by course number from low to high**

Bst->inOrder();//this calls inOrder, defined above

**//print the sorted list to display**

inOrder prints as written

`

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Getline** | 1 | n | n |
| **Course id = string1** | 1 | n | n |
| **Course name = string2** | 1 | n | n |
| **course prerequ = strings after** | 1 | n | n |
| **Bst insert(Course course)** | 1 | n | n |
| **Insert(course)** | 1 | n | n |
| **If root is nullptr** | 1 | n | n |
| **Root = new node(class)** | 1 | N | n |
| **Else addNode(root,class)** | 1 | n | n |
| **Total Cost** | | | 9n |
| **Runtime** | | | O(n) |

**Advantages and disadvantages of each structure:**

**According to the big o cheatsheet from bigocheatsheet.com, a binary search tree, hash table, and vector all have space complexities that will depend on n, O(n), with n = the number of data values that are being processed. A hash table is not able to provide an ordered list of data values, with the exception of the random ordering that occurs as a result of the hash function. This means that a hash table may be a poor choice for this assignment since new memory will be used to house the subsequent data type created in the process of sorting a hash table linearly. A doubly linked list performs well in some cases for inserting and deleting items, with a time complexity of O(1), but the structure is better used to further organize other data structures. A binary search tree has an average access, search, insertion, and deletion time of O(log(n)) and worst case times of O(n). This means that a binary search tree may see increases in performance times versus a doubly linked list or vector for certain operations.**

**Hash tables seem to be a good method of speeding up the time to find an item while not having to worry too much with how the data is separated in the table. The hash table is mainly used to store and retrieve items in a short amount of time but is not a good choice for items that need to be sorted more thoroughly. Vectors do not necessarily perform as fast as the other choices for access in some cases however the orderliness of the vector construction process and the ability to easily sort the values make vector a contender for this assignment. A binary search tree is fairly easy to access values from smallest to largest with widely known recursive calls of different order depending on the desired sort. This makes printing ascending values easier than if using a vector data structure because to sort a vector additional functions to sort will be required that are possibly more confusing to write. A binary search tree can store custom classes with multiple data fields or just a single variable. This means after creating the custom objects it is relatively easy to populate a binary search tree. With each data structure having similar space and time complexities, hash table requiring a separate structure to sort properly, and vector requiring a somewhat complicated sorting algorithm, *I elect to implement the requested project using a binary search tree.***