Project One

Brian P Kremer

SNHU

CS-300-10860-M01: Data Structures & Algorithms

Michael Rissover

June 22nd, 2024

Project One

Resubmission of pseudocode:

Vector:

- Create a vector to store course objects
- Open the file with the course data
- Read the lines within that file and create course objects for it
- Close the file
- Sort the course vectors in alphanumeric order by course number
- Print list of courses in the alphanumeric order

Hash Tables:

- Create a hash table to store the course objects
- Open the file with course data
- Read the lines within that file and create course objects for it
- Close the file
- Print list of courses in the alphanumeric order

Binary Search Tree:

- Create a binary search tree to store the course objects
- Open the file with the course data
- Read the lines inside the file and create course objects for it
- Close the file
- Print list of courses in the alphanumeric order

Pseudocode for the menu:

```
//set variables
1 = load data structure
2 = print course list
3 = print course
4 = exit
//Displaying menu’s options
Print “1. Load Data Structure”
Print “2. PRint Course List”
Print “3. Print Course”
Print “4. Exit”
Begin loop.
//Get user input then display options
If ‘1’ then
If data_structure isn’t loaded then
get filename from user
        Call loadDataStructure (filename)
        Set data_structure as return value
    Else
        print “Data is loaded. You may print course list or course details next. “
    EndIf

Else if ‘2’ then
```

```
    If data_structure is loaded then
        Call printCourseList(data_structure)
    Else
        Print “Data structure not loaded. Please select option 1. “
    EndIf

Else if ‘3’ then
    If  data_structure is loaded then
    Get courseName from user
    Call printCourse(data_structure, courseName)
    Else
    Print “data structure not loaded. Please select option 1. “
    EndIf

Else if ‘4’ then
    Print “Exiting program”
    Exit loop
Else
Print “Error. Try another option. “
End if
End loop
```

Pseudocode that prints courses in an alphanumeric ordered list using a vector:

```
//Vector that stores course objects
Courses = []
//Opening the file with course information
File = open (course.file)
//Read the files lines to create a course object
For line in file.read()
Course = course(line)
        Courses.append(course)
//Closing the file
File.close()
//sorting the corse vector in alphanumeric order by course number
Courses.sort(key=lambda, course: course.number)
//Outputting the course list in alphanumeric order
For course in courses:
Print(course.number, course.title)
```

Pseudocode for printing a list of courses in alphanumeric order using a hash table:

```
//Create a hash table to store the course objects
Course = {}
//Opening the course data file
File = open (course.file)
//Read the lines from the file and create a course onject for it
For line in file.readlines()
```

```
Course = Course(line)
Courses[course.number] = course
//Close the file
File.close()
//Output the list of courses in alphanumeric order
For course in sorted(courses.values()):
Print(course.number, course.title)
```

Pseudocode for printing a list of courses in alphanumeric order using a binary search tree

```
//Creating the binary tree to store the course objects in
Course = binarySearchTree()
//Open the file with the course information
File = open(course.file)



//Read each line of the file and create course objects for it
For line in file.readliness():
Course = Course(line)
Courses.insert(course)
//Close the file
File.close()
//output the list of courses in alphanumeric order
Courses.print_in_order()
```

Pseudocode for the menu:

```
//set variables
1 = load data structure
2 = print course list
3 = print course
4 = exit
//Displaying menu’s options
Print “1. Load Data Structure”
Print “2. Print Course List”
Print “3. Print Course”
Print “4. Exit”


Begin loop.
//Get user input then display options
If  ‘1’ then
If data_structure isn’t loaded then
get filename from user
        Call loadDataStructure (filename)
        Set data_structure as return value
    Else
```

```
        print “Data is loaded. You may print course list or course details next. “
    EndIf
Else if ‘2’ then
    If data_structure is loaded then
        Call printCourseList(data_structure)
    Else
        Print “Data structure not loaded. Please select option 1. “
    EndIf
Else if ‘3’ then
    If  data_structure is loaded then
    Get courseName from user
    Call printCourse(data_structure, course_name)
    Else
    Print “data structure not loaded. Please select option 1. “
    EndIf

Else if ‘4’ then
    Print “Exiting program”
    Exit loop
Else
Print “Error. Try another option. “
End if
End loop
```

Evaluation:

First, we'll look at our vectors.

Operations	**Cost Per Line**	**Number of times executed**	**Big O Value**
Opening and reading a file	1	O(n)	O(n)
Parsing each line and creating course objects	1	O(n)	O(n)

Hash Table:

Operations	**Cost Per Line**	**Number of times executed**	**Big O Value**
Opening and reading a file	1	O(n)	O(n)
Parsing each line and creating course objects	O(1)	O(n)	O(n)

Binary Search Tree:

Operations	**Cost Per Line**	**Number of times executed**	**Big O Value**
Opening and reading a file	1	O(n)	O(n)
Parsing each line and creating course objects	O(log n)	O(n)	O(n log n)

Overall advantages and disadvantages of the Vectors. Advantages would include the performance is better in sequential access to data elements. Vectors also have large memory storage. The disadvantages here are the deleted and inserted data elements in the middle are expensive, (O(n)). Overall advantages of the Hash table were swift search time and quick retrieval of data elements. A disadvantage would be the use of memory because of the complexity of implementing the table. Finally for the binary search tree, advantages include insertion and deletion operations perform efficiently. It also effectively sorts data elements. Similar to the hash table our disadvantage here is the amount of memory needed due to the structure.

Based off the Big O analysis we can conclude that a hash table would be our most suitable option for the scenario. The hash table provides swift search times for retrieval. This would give use quick access to the course information needed. The courses can have unique identifiers which are functionally effective for hash tables.