Brianna Peoples

CS 300

Project One

Vector Data Structure Pseudocode:

```
Start program

Open the file "courses.txt"

If the file cannot be opened, print an error and stop program

Create an empty list called courseList.

For each line in the file:
   Split the line by commas
   If the line has less than 2 parts, print an error and skip it
   Set the first part as courseNumber
   Set the second part as courseTitle
   Store any remaining parts as prerequisites
   Create a course object with this data and add it to courseList

Close the file

Check that all prerequisites exist in courseList

If a prerequisite is missing, print an error

Show a menu:
1. Search for a course:
   Ask for a course number
   If found, print its title and prerequisites
   If not found, print an error
2. Print all courses:
   Sort courseList by courseNumber using quicksort or mergesort
   Print "Courses in alphanumeric order:"
   For each course in courseList:
      Print course.course_number + ": " + course.course_title
3. Exit

Repeat until the user chooses to exit
```

End the program

Hash Table Data Structure Pseudocode:

```
begin

    define course as
        course_number
        title
        prerequisites (list)

    define course_table as hash table

    function load_courses(file_name)
        open file file_name
        if file cannot open
            print "Error: Cannot open file"
            return

        while there are lines in file
            read line
            split line into list using commas

            if length of list < 2
                print "Error: Wrong format in line"
                continue

            course_number = list[0]
            title = list[1]
            prerequisites = list[2 to end]

            course = course(course_number, title, prerequisites)
            insert course into course_table using course_number as key

        close file

    function check_prerequisites()
        for each course in course_table
            for each prerequisite in course.prerequisites
                if prerequisite not in course_table
                    print "Error: Missing prerequisite " + prerequisite + " for " + course.course_number

    function print_all_courses()
        convert course_table to a list called course_list
```

```
    sort course_list by course_number using quicksort or mergesort
    print "Courses in alphanumeric order:"
    for each course in course_list
        print course.course_number + ": " + course.title
        if course.prerequisites is not empty
            print "Prerequisites: " + join course.prerequisites with ", "
        else
            print "No prerequisites"

function search_course(course_number)
    if course_number not in course_table
        print "Course not found"
        return

    course = course_table[course_number]
    print course.course_number + ": " + course.title
    if course.prerequisites is not empty
        print "Prerequisites: " + join course.prerequisites with ", "
    else
        print "No prerequisites"

function main()
    call load_courses("courses.txt")
    call check_prerequisites()
    call print_all_courses()

call main()

end
```

Tree Data Structure Pseudocode:

```
struct course
    course_number : string
    course_title : string
    prerequisites : list of string
end struct

struct node
    course : course
    left : node
    right : node
end struct

class binarysearchtree
    root : node

    method insert(course_item : course)
        if root is null then
            root = create_new_node(course_item)
        else
            insert_recursive(root, course_item)
        end if
    end method

    method insert_recursive(current_node : node, course_item : course)
        if course_item.course_number < current_node.course.course_number then
            if current_node.left is null then
                current_node.left = create_new_node(course_item)
            else
                insert_recursive(current_node.left, course_item)
            end if
        else
            if current_node.right is null then
                current_node.right = create_new_node(course_item)
            else
                insert_recursive(current_node.right, course_item)
            end if
        end if
    end method

    method print_sorted_courses()
        print "Courses in alphanumeric order:"
        in_order_traversal(root)
```

```
    end method

    method in_order_traversal(current_node : node)
        if current_node is not null then
            in_order_traversal(current_node.left)
            print_course_info(current_node.course)
            in_order_traversal(current_node.right)
        end if
    end method

    method print_course_info(course_item : course)
        print "course number: " + course_item.course_number
        print "course title: " + course_item.course_title
        if course_item.prerequisites is empty then
            print "prerequisites: none"
        else
            print "prerequisites: " + (join course_item.prerequisites with ", ")
        end if
        print "-------------------------------------"
    end method
end class

function load_courses_from_file(filename : string) returns binarysearchtree
    declare bst : binarysearchtree
    declare course_map : map of string to course
    declare file : file
    declare line : string
    declare line_number: integer = 0

    bst = create_new_bst()
    file = open_file(filename)

    if file cannot be opened then
        print "error: cannot open file " + filename
        return bst
    end if

    while not eof(file)
        line_number = line_number + 1
        line = read_line(file)

        if line is empty then
            continue
        end if
```

```
        tokens = split(line, ',')

        if length(tokens) < 2 then
            print "error on line " + convert_to_string(line_number) + ": not enough data."
            continue
        end if

        course_number = trim(tokens[0])
        course_title  = trim(tokens[1])

        declare prereq_list : list of string
        for i from 2 to (length(tokens) - 1)
            add trim(tokens[i]) to prereq_list
        end for

        declare new_course : course
        new_course.course_number  = course_number
        new_course.course_title   = course_title
        new_course.prerequisites  = prereq_list

        course_map[course_number] = new_course
    end while

    close_file(file)

    for each key in course_map
        current_course = course_map[key]
        for each prereq in current_course.prerequisites
            if prereq not in course_map then
                print "warning: prerequisite " + prereq + " for course " + key + " not found."
            end if
        end for
    end for

    for each key in course_map
        bst.insert(course_map[key])
    end for

    return bst
end function

function main()
    declare bst : binarysearchtree
```

```
    declare filename : string = "abcuniversity_coursedata.txt"

    bst = load_courses_from_file(filename)

    print "=== all courses (sorted) ==="
    bst.print_sorted_courses()
end function
```

Menu Pseudocode:

```
function menu()
    Repeat until user chooses to exit:
        Print "1. Load Courses"
        Print "2. Print All Courses (Sorted)"
        Print "3. Search for a Course"
        Print "4. Exit"
        Get user input

        If input is 1:
            Call load_courses("courses.txt")
            Print "Courses loaded successfully."

        If input is 2:
            Call sort_courses()
            Call print_all_courses()

        If input is 3:
            Print "Enter course number: "
            Get course_number
            Call search_course(course_number)

        If input is 4:
            Print "Exiting program..."
            Exit loop
```

Vector Data Structure Runtime Analysis:

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`	2	1	1
`for each prerequisite of the course`	1	n	n
`print the prerequisite course information`	2	n	n
		Total Cost	6n + 1
		Runtime	1(n)

Hash Table Data Structure Runtime Analysis:

Code	Line Cost	# Times Executes	Total Cost
`for all courses`	2	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`	2	n	n
`print the prerequisite course information`	4	n	n
		Total Cost	9n + 1
		Runtime	O(n)

Tree Data Structure Runtime Analysis:

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`	2	1	1
`for each prerequisite of the course`	1	n	n
`print the prerequisite course information`	4	n	n
Total Cost			8n + 1
Runtime			O(n)

Analysis of the Advantages and Disadvantages of Each Data Structure

Each data structure has strengths and weaknesses for handling course data. A vector is simple and memory-efficient. It allows easy access but requires scanning the entire list to find a course, making searches slower. Since vectors do not maintain order, sorting is needed before printing courses in alphanumeric order.

A hash table provides the fastest search time since it directly looks up courses using keys. It is also efficient for inserting and deleting courses. However, it does not store data in order, so extra steps are needed to sort before printing. It also requires more memory and can slow down if too many items share the same key.

A binary search tree keeps courses sorted naturally. Searching is faster than in a vector, and printing courses in order is simple. However, if the tree becomes unbalanced, it can slow down operations. Binary search trees also use more memory since each node stores extra links.

Since printing courses in alphanumeric order is required, a binary search tree is the best choice. It keeps courses sorted and allows efficient searching and printing. However, if ease of use is the priority, a vector is a simpler option, even though sorting is needed before printing.