Project One

Filip Arghir

Southern New Hampshire University

CS300

Saba Jamalian

10/9/2025

```
Vector Pseudocode -
Data Structure Definition:
Structure Course
String courseNumber
String courseTitle
Vector<string>  prerequisites
End Structure

Load Data From File
Function loadCourses(fileName: String) returns Vector<Course>
Declare Vector<Course> courses

Open file with fileName
If file cannot be opened
                Print ‘Error: Could not open file”
                Return empty courses
        End If

For each line in file
Split line by comma into tokens

        If number of tokens < 2
                Print “Error: Invalid line format”
                Continue to next line
        End If

        Create new Course newCourse
        newCourse.courseNumber = tokens[0]
        newCourse.courseTitle = tokens[1]

        For each token from tokens[2] to end
                Add token to newCourse.prerequisites
        End For

        Add newCourse to courses
End For

Close File
//Validate prerequisites
For each course in courses
For each rereq in course.prerequisites
If prereq does not exist in courses
Print ”Warning: prerequisite ” + prereq + “ not found”
End If
End For
End For
```

```
        Return Courses
End Function
```

Create Course Objects and Store in Vector:

```
Procedure addCourses(courses : Vector<Course>, course : course)
Append course to courses
End Procedure
```

Search and Print Course Information

```
Function searchCourse(courses : Vector<Course>, courseNumber : String)
For each course in courses
If course.courseNumber equals courseNumber
Print course.courseNumber+  ": " + course.courseTitle
If course.prerequisites is empty
Print "Prerequisites: None"
Else
Print " Prerequisites:"
For each prereq in course.prerequisites
Print prereq
                End For
            End If
            Return
        End If
    End For
    Print "Course " + courseNumber + " not found"
    End Function
```

Print All Courses:

```
Procedure printAllCourses(course : Vector<Course>)
    For each course in courses
    Print course.courseNumber + ", " + course.courseTitle
    End For
End Procedure
```

Run Time Analysis – Vector Data structure

Loading Data and creating course objects

Code	Line Cost	# Times Executes	Total Cost
Open file	1	1	1
For each line in file	1	n	n
Split line by comma	1	n	n
Check if tokens < 2	1	n	n
Create new Course	1	n	n

Assign courseNumber and title	1	n	n
Add prerequisites to course	1	n	n
Add course to vector	1	n	n
Close file	1	1	1
Total Cost			**7n + 2**
Runtime			**O(n)**

Print All Courses Sorted (Option 2)

Code	Line Cost	# Times Executes	Total Cost
Sort vector	n log n	1	n log n
For each course in vector	1	n	n
Print course information	1	n	n
Total Cost			**n log n + 2n**
Runtime			**O(n log n)**

Search and Print Course(Option 3)

Code	Line Cost	# Times Executes	Total Cost
For each course in vector	1	n	n
Compare courseNumber	1	n	n
Print course information	1	1	1
Print prerequisites	1	1	1
Total Cost			**2n + 2**
Runtime			**O(n)**

Hash Table Pseudocode -

File Reading and Validation

```
void loadDataStructure(HashTable courses, String fileName) {
** open file **
** if file cannot be opened **
        ** display error message **
** return **
**while not end of file**
        **read line from file**
        **split line by comma**
```

```
                **if tokens size < 2**
                        **display format error**
                        **continue**
                        **add course data to temporary list**
        **close file**
**for each course in temporary list**
   **for each prerequisite**
      **if prerequisite not found in course list**
        **display error and return**
        }
```

Course Object Creation and Hash Table Storage

```
        Structure Course
                courseNumber, title, prerequisites
         End Structure

        Structure Node
                course, key, next
        End Structure
        void Insert(HashTable courses, Course course) {
        ** calculate hash key from course number **
                ** if bucket is empty **
                        ** create new node **
                ** else **
                        ** add to front of chain **
        }
```

Course Information and Prerequisites Printing

```
//Hash Table - Milestone 2

void searchCourse(HashTable<Course> courses, String courseNumber) {
        **calculate hash key from course number**
        **traverse chain at bucket**
        **if course is found**
                **print out the course information**
                **for each prerequisite of the course**
                **search hash table for prerequisite**
                **print the prerequisite course information**
 }

void Remove(HashTable<Course> courses, String courseNumber) {
        **calculate hash key from course number**
        **find course in chain**
        **remove node from chain**
 }
```

```
void PrintAll(HashTable<Course> courses) {
        **for all buckets**
        **traverse each chain**
        **print course information**
}
```

Main Program Structure

Main Program

Initialize courses = new HashTable

While user has not chosen Exit
    Display menu options
    Read user choice

    If choice == 1 → **load courses into hash table**
    If choice == 2 → **print all courses**
    If choice == 3 → **search for courseNumber**
    If choice == 9 → exit program

    End While
    Print "Thank you for using the course planner!"
End Program

Hash Table Data Structure -

Code	Line Cost	# Times Executes	Total Cost
Open file	1	1	1
For each line in file	1	n	n
Split line by comma	1	n	n
Check if tokens < 2	1	n	n
Create course object	1	n	n
Calculate hash key	1	n	n
Insert into hash table	1	n	n
Close file	1	1	1
Total Cost			**6n + 2**
Runtime			**O(n)**

Print All Course Sorted -

Code	Line Cost	# Times Executes	Total Cost
For each bucket	1	n	n
Copy course to temp vector	1	n	n
Sort temporary vector	n log n	1	n log n

For each course in sorted vector	1	n	n
Print course information	1	n	n
Total Cost			**n log n + 4n**
Runtime			**O(n log n)**

Search and Print Course

Code	Line Cost	# Times Executes	Total Cost
Calculate Hash key	1	1	1
Search bucket chain	1	1	1
Print course information	1	1	1
Print prerequisites	1	1	1
Total Cost			**4**
Runtime			**O(1)**

Binary Search Tree Pseudocode - – Milestone
Structure Course
    courseNumber, title, prerequisites
End Structure

Structure Node
    course, left, right End
Structure

File Reading and Validation
void loadDataStructure(BinarySearchTree courses, String fileName) {
** open file **
** if file cannot be opened **
    ** display error message **
    ** return **

while not end of file
    read line from file
    split line by comma

    if tokens size < 2
    display format error
    continue

add course data to temporary list

```
**close file**

    **for each course in temporary list**
        **for each prerequisite**
            **if prerequisite not found in course list**
                **display error and return**

    }
```

Course Object Creation and Binary Search Tree Storage

```
Structure Course
   courseNumber, title, prerequisites
End Structure

Structure Node
   course, left, right
End Structure
void Insert(BinarySearchTree courses, Course course) {
** if root is null **
    ** create new node as root **
** else **
    ** find correct position in tree **
    ** compare course numbers **
    ** less than current node **
        ** go left **
    ** else **
        ** go right **
    ** insert as new leaf node **
}
```

Course Information and Prerequisites Printing

```
void PrintCourseList(BinarySearchTree courses) {
** perform in-order traversal **
    ** visit left subtree **
    ** print course number and title **
    ** visit right subtree **
}
void PrintCourse(BinarySearchTree courses, String courseNumber) {
** search tree for course **
** if course found **
    ** print course number **
    ** print course title **
    ** print prerequisites **
** else **
    ** display course not found **
}
```

Main Program
Initialize courseTree
While user has not chosen Exit
Display menu
If choice == 1 → load courses
If choice == 2 → print course list
If choice == 3 → print course details
End While
Print "Thank you for using the course planner!"
End Program
Binary Search Tree Data Structure -

Code	Line Cost	# Times Executes	Total Cost
Open file	1	1	1
For each line in file	1	n	n
Split line by comma	1	n	n
Check if tokens < 2	1	n	n
Create course object	1	n	n
Insert into BST	log n	n	n log n
Close file	1	1	1
Total Cost			**n log n + 4n + 2**
Runtime			**O(n log n)**

Print All Courses

Code	Line Cost	# Times Executes	Total Cost
In-Order traversal	1	n	n
Print Course information	1	n	n
Total Cost			**2n**
Runtime			**O(n)**

Search Print Courses

Code	Line Cost	# Times Executes	Total Cost
Search BST for course	log n	1	log n
Print course information	1	1	1
Print prerequisites	1	1	1
Total Cost			**log n + 2**

Runtime			**O(log n)**

Summary Comparison

Operation	Vector	Hash Table	Binary Search Tree
Load Data	O(n)	O(n)	O(n log n)
Print Sorted List	O(n log n)	O(n log n)	**O(n)**
Search Course	O(n)	O(1)	O(log n)

Advantages and Disadvantages

Vector

- Advantage: Easy to implement and debug, great memory layout and fast iteration
- Disadvantages: Search aand insertion operations are O(n) and sorting must be performed every time data ais needed meaning its not efficient for constant lookups
- Use Case: ideal for smaller data sets but scales poorly as the data set grows

Hash Table

- Advantages: O(1) average lookup and insertion and ideal for quick search and retrieval by course number
- Disadvantages: Requires additional data structure or sorting for alphanumeric listings and higher memory usage due to hashing also potential for collisions and uneven bucket distribution
- Use Case: Perfect for direct lookups but sorting all courses by course number is inefficient compared to a tree

Binary Search Tree

- Advantages: Maintains sorted order automatically; O(log N) search and insertion for trees, is efficient for ordered printing and individual lookups
- Disadvantages: More complex for implementation and has potential for O(n) if not balanced properly, will use more memory per node due to pointers

- Use Case: best for this exact advising system as its efficient for both sorted traversal and logarithmic time searching for a course number

After evaluating and implementing all three data structures, I advise the binary search tree for ABCU's course advising system. This is due to the frequent print and alphanumeric ordered list of courses as well as retrieval of specific courses. A binary search tree or BST helps preserve alphanumeric order without additional sorting and while a hash table offers constant time searches it doesn't maintain an order properly meaning that efficiency can be O(n log n). Sorting is required whenever a course must be listed and the vector data structure while simple performs searches and insertions in linear time but for large data sets struggles. Overall the BST is optimal due to its balance between time complexity, memory efficiency, functional requirements, O(log n) for search and insertion and O(n) for ordered printing.