Albert Waterman

CS300 project 1

7 August 2025

**Pseudocode Vector**

Start

Menu Function:  
  
DO  
 Display:  
 1. Load data  
 2. Print course list  
 3. Print course details  
 9. Exit  
 Get user input  
  
 IF input = 1  
 Call LoadCourseFile()  
  
 ELSE IF input = 2  
 Call PrintAllCourses()  
  
 ELSE IF input = 3  
 Prompt for course number  
 Call searchCourse()  
  
WHILE input != 9

Main Function:

Initialize the courseList vector

Call LoadCourseFile and load data from file into courseList

User input for course number to search

Call FindAndPrintCourse with courseList and course number

LoadCourseFile Function;

Open file

If file fails to open prompt error and exit

WHILE not the end of file

Read line from file

Split line with “, “ into list of values

If less than 2 values

Prompt error with bad file format and skip that line

SET courseNumber = first value

SET courseTitle = Second value

SET preReq = remaining values

Create Course object with courseNumber, courseTitle, preReq

Add Course object to courseList

FOR each course in courseList

For each preReq in course

Check if preReq exist in courseList

IF not

Display warning “missing prerequisite”

Return courseList

PrintAllCourses Function:

Sort courseList by courseNumber (use quick sort)

FOR each course in sorted courseList

Print courseNumber and courseTitle

searchCourse function:

For each course in courseList

IF courseNumber matches users input

Print course number, title

IF course has no preReq

Print “Prerequisites, none”

Else

For each preReq in course

Print “preReq course number”

END

If match not found

Print “No course found”

END

**Pseudocode Hash Table**

Start

Menu Function:  
  
DO  
 Display:  
 1. Load data  
 2. Print course list  
 3. Print course details  
 9. Exit  
 Get user input  
  
 IF input = 1  
 Call LoadCourseFile()  
  
 ELSE IF input = 2  
 Call PrintAllCourses()  
  
 ELSE IF input = 3  
 Prompt for course number  
 Call searchCourse()  
  
WHILE input != 9

Main Function:

Initialize the courseTable (hash table)

Call LoadCourseFile and load data from file into courseTable

User input for course number to search

Call searchCourse with courseTable and user input

LoadCourseFile Function;

Open file

If file fails to open prompt error and exit

WHILE not the end of file

Read line from file

Split line with “, “ into list of values

If less than 2 values

Prompt error with bad file format and skip that line

SET courseNumber = first value

SET courseTitle = Second value

SET preReq = remaining values

Create Course object with courseNumber, courseTitle, preReq

Insert Course object into courseTable using courseNumber as key

FOR each course in courseTable

For each preReq in course

Check if preReq exists in courseTable

IF not

Display warning “missing prerequisite”

Return courseTable

PrintAllCourses Function:

Create a tempList vector

FOR each course in courseTable

Add course to tempList

Sort tempList by courseNumber

FOR each course in sorted tempList

Print courseNumber and courseTitle

searchCourse function:

Search courseTable using courseNumber input

IF course not found

Print “no course found”

IF course has no preReq

Print “Prerequisites, none”

Else

For each preReq in course

Search courseTable for prereq course

IF found

Print “preReq course number and title”

ELSE

Print prereq course number + “not found”

END

**Pseudocode Binary Search Tree**

Start

Menu Function:  
  
DO  
 Display:  
 1. Load data  
 2. Print course list  
 3. Print course details  
 9. Exit  
 Get user input  
  
 IF input = 1  
 Call LoadCourseFile()  
  
 ELSE IF input = 2  
 Call PrintAllCourses()  
  
 ELSE IF input = 3  
 Prompt for course number  
 Call searchCourse()  
  
WHILE input != 9

Main Function:

Initialize the courseTree (binary search tree)

Call LoadCourseFile and load data from file into courseTree

User input for course number to search

Call searchCourse with courseTree and user input

LoadCourseFile Function;

Open file

If file fails to open prompt error and exit

WHILE not the end of file

Read line from file

Split line with “, “ into list of values

If less than 2 values

Prompt error with bad file format and skip that line

SET courseNumber = First value

SET courseTitle = Second value

SET preReq = remaining values

Create Course object with courseNumber, courseTitle, preReq

Insert Course object into courseTree using courseNumber as key

FOR each course in courseTree

For each preReq in course

Check if preReq exists in courseTree

IF not

Display warning “missing prerequisite”

Return courseTree

PrintAllCourses Function:

Call inOrderTraversal(courseTree)

InOrderTraversal Function (node):

IF node is not null

InOrderTraversal node->left

Print node courseNumber and courseTitle

InOrderTraversal node->right

searchCourse function:

Search courseTree using courseNumber input

IF course not found

Print “no course found”

IF course has no preReq

Print “Prerequisites: none”

Else

For each preReq in course

Search courseTree for prereq course

IF found

Print “preReq course number and title”

ELSE

Print prereq course number + “not found”

END

**Run Time Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Vector** | **Hash Table** | **BST** |
| **Load and insert** | O(N) | O(N) fastest and O(N^2) if there is collisions occurring | O(log N) best and O(N) worst depending on balance |
| **Searching** | O(N) | O(1) fastest and O(N) if there is collisions occurring | O(log N) and O(N) depend on trees balance. |
| **Sorting and Printing** | O(N log N)  This is using quicksort method | O(N log N) uses a temp list and sorts | O(N) tree is in traversal order. |

When looking at performance, each data structure has its own strengths and weaknesses. With a vector, inserting at the end is fast at O(1), but with searching means checking every course one by one, which is O(N). Printing a sorted list also takes a lot of work since it requires running a sorting algorithm like quicksort at O(N log N). Hash tables are great for fast insert and search in the best case at O(1), but collisions can slow them down to O(N). They are also unordered, so printing in alphanumeric order means extracting all the items into a temporary list and sorting them, which can add a lot of extra time. BST works differently. In a balanced tree, insert and search are O(log N), and one of the biggest advantages is that in-order traversal automatically makes a sorted list in O(N) without needing additional sorting steps. But, if the tree becomes unbalanced, performance can drop to O(N).

I think BST is the best choice for this. The built-in sorted order from in-order traversal and the course list can be printed exactly as advisors want without extra processing. Searching for a course is great, and the structure will scale well as more courses are added. Hash tables are fast for searches but don’t maintain sorted order, which requires extra processing, and vectors are less efficient with larger datasets, so as this grows vectors get worse. A BST balances speed and order, making it the best option for ABCU.