Project One

Justin J Perez

SNHU

CS 300

Nathan Lebel

8/5/2025

Introduction

Sorting algorithms is something often not known about beginner level programmers or clients when building a program. However, it is probably the biggest thing that isn't the code itself that can majorly affect your code especially when working on big projects you plan to use for multiple years. If you select a wrong algorithm or even worse no algorithm, then you're setting yourself up to have a mandatory refactor down the line if it even launches.

For ABCU they are Requesting a program that will print all the computer science courses in alphanumerical order and print out its title and prerequisites. Going Forward we will look at each structure with this lens we will also use the code from past assignments they were pleased with.

Vector Data Structure Pseudocode

Structure called course that checks for

- CourseID
- CourseName
- Prerequisites

Function ISCourseValid(CourseID)

- If course ID is longer than 7 characters
  - Invalid
- If the first 4 characters aren't letters
  - Invalid
- If the last 3 are not digits
  - Invalid
- Else
  - Valid

Function loadCoursesfromfile(Filename)

Create an empty list of Lines

Create an empty list of course IDS

For each line in the file

    Split the line by Commas

    If fewer then 2 items after splitting the line

        “Invalid format!”

Check if CourseID is valid

If CourseID is valid continue

    Else “Invalid course ID on XX”

Add coursed to CourseIDs

Add the full line with courseIDs

return the list of CourseIDs

Function Build Courses(Lines, CourseIDs)

    Create an empty list of course objects

For each line in lines

    Split by commas

    First item is coursed

    Second item is CourseName

    All other items on line are prerequisites

For each prerequisite check if CourseID is valid

- if not, return “invalid prerequisites”

Create couse object with CourseID, Name, and Prerequisites

Add it to the list of course objects

Return the list of course objects

Function Search Course(Course list, CourseIDSearch)

For each course in Courselist do the following

- If Searched course matches what we are looking for
  - Print Course ID and Name
  - If prerequisites
    - Print Prerequisites
  - Else
    - “No prerequisites
- Else
  - “No Course of that name!”

Main Program:

- Load File
- Load courses, lines, and course IDS.
- Build a course list from the File
- Ask user for input on what to do
  - Show all the courses
    - Print all courses
  - Search course

        - Start Search Function
    - Add Course
        - Add new line to file and ensure input is valid
    - Exit
        - Exit Program.

Hash Table Data Structure Pseudocode

Reading the file():

- Use Fstream to open the file
- Make call to open file
- If file isn't found return 1
    - Else while it's not the EOF
        - Read each line
            - If there is less then 2 values on 1 line then give an error
            - Else read parameters
            - If 3 or more keep writing until finished.
            - else finish
- Close File

Create Hash table():

- Create a vector<Node> called nodes
- Create a HashTable class
- Define an insert method inside the HashTable class to insert items

- Open the input file
- While not at end of file:
    - Read each line from the file
    - Split the line by commas into parts

    - Create a temporary Course object
    - Set the course number and course title using the first two parts

    - If a third value (or more) exists:
        - Add all additional values as prerequisites to the Course object

- Call the insert method to add the Course object to the hash table

Search and print from Hash Table () :

- Ask the user for input (course number)
- Assign input to a variable called key
- If the key is found on the hash table:
  - Print the course number and title
  - If the course has prerequisites:
    - For each prerequisite on the list:
      - Print the prerequisite
  - Else:
    - Print “Prerequisites: None”
- Else:
  - Print “Course not found”

Tree Data Structure Pseudocode

Reading the file():

- Use Fstream to open the file
- Make call to open file
- If file isn’t found return 1
  - Else while it’s not the EOF
    - Read each line
      - If there is less then 2 values on 1 line then give an error
      - Else read parameters
      - If 3 or more keep writing until finished.
      - else finish
- Close File

Create Binary Search Tree():

- Define A struct called Course with:
    - CourseID
    - courseTitle
    - Prerequisites
- Define a tree node class with:
    - Course object
    - Left (TreeNode)
    - Right (TreeNode)
- Define a BinarySearchTree class with:
    - Root(treenode)
    - Insert method to add course into BST based on the coursed
- Open input file with Fstream
- While not at end of file:
    - Read each line from the file
    - Split the line by commas into parts
    - Create a temporary Course object
    - Set CourseID and courseTitle using the first two parts
    - If a third value (Or more):
        - Add all extra values as prerequisite to the course Object.
    - Call insert method to add the course object to bst

Search and print from Binary Search Tree() :

- Ask the user for input (course number)
- Assign input to a variable called key
- If the key is found on the BST:
    - Print the course number and title
    - If the course has prerequisites:
        - For each prerequisite on the list:
            - Print the prerequisite
    - Else:
        - Print “Prerequisites: None”
- Else:
    - Print “Course not found”

Evaluation

BIG O Notion

The Big O notation is a way to describe how efficient an algorithm is by measuring its performance. It helps understand the worst possible case and the average case. A runtime analysis table is in this document to explain it further.

Runtime Analysis table

Data Structures	Load Data (insertion)	Print All courses	Print Specific Course	Advantages	Disadvantages
Vector	O(n)	O(n log n)	O(n) (Linear search)	Easy to implement Fast Insertion toward the end	Slow search Not effective for large databases
Hash Table	O(1) (average) / O(n) (worst case)	O(n)	O(1) average / O(n) worst case	Fast Average Insertion and search Good for direct lookups	Unordered data (No Natural sorting)
Binary Search Tree	O(log n) (average) / O(n) (Worst case)	O(n)	O(log n) average / O(n) worst case	Keep Data sorted and efficiently travel through data Decent Search Speed	Can become unbalanced leading to the worst-case O(n) Complex to implementation

Advantages & Disadvantages

It's important to know before going into this every algorithm does have it advantages but some are better for ABCU purpose.

- Vector
    - Advantages
        - Simple and fast development
        - Great for storing items in an order
        - Good for Small and fixed sized Data
    - Disadvantages
        - Slow searching
        - Inserting in the middle requires a huge shift of elements
        - Not effective for large datasets, it’s extremely slow the bigger the data is
- Hash table
    - Advantages
        - Decent search, insert, and delete speeds
        - Great for quick lookups using Keys
        - Doesn’t require sorted data
        - Flexible to any size
        - Readability for coders
    - Disadvantages
        - No natural order. Will have a unsorted output
        - Worst one if a collision happens
        - Can be complex to implement
        - Use a lot more memory than other structures
- Binary Search Tree (BST)
    - Advantages

        - Maintain Data in sorted Order
        - In order Traversal of nodes
        - Fast searches insert and delete
    - Disadvantages
        - Dangerous if becoming unbalanced makes it extremely slow
        - Complex to implement
        - Performance depends on tree structure.
        - Sometimes you must unbalance it

Recommendation

After looking critically at each algorithm, the one I believe would be the best and the easiest to implement would be a hash table. While it doesn't provide the fastest speeds for inserting and searching data it by far has the most practicality of the use as ABCU wants a program to search up data at a whim and pull it to view. It also doesn't have any of the horrible disadvantages like binary search tree the BST has a huge disadvantage of if it becomes unbalanced the consequences could be huge and be forced to have the worst times load times. This could be common due to how binary nodes are set up where data must be related. If the school has only 5 art classes compared to 10 computer classes, this could immediately make the data unbalanced and cause a big slowdown in performance. If we were unable to use a hash table, the BST would be the best option though we would have to tread very carefully making it. The only algorithm I would always stay away from is the vector as the data set is going to be too large for it to handle it effectively.

References

- Comment, info, M., SSoumyadeepDebnath, SSoumyadeepDebnath, SSoumyadeepDebnath, S, SoumyadeepDebnath, & Follow. (2025, April 21). *Big O notation tutorial - A guide to big O analysis*.

- GeeksforGeeks. https://www.geeksforgeeks.org/dsa/analysis-algorithms-big-o-analysis/*Know thy complexities!*. Big. (n.d.). https://www.bigocheatsheet.com/