Joshua D Rodriguez

Southern New Hampshire University

Professor Saba Jamaliannasrabadi

CS 300 Project One

1. ***Pseudocode From Milestones***
2. ***Opening and Reading the File***
3. ***Vector Based:***

FUNCTION LOAD\_COURSE\_DATA (file\_name)

OPEN file\_name

IF file cannot be opened

PRINT "Error: File not found or cannot be opened."

RETURN

CREATE an empty vector named courses

WHILE there are lines in the file

READ current line

SPLIT line into tokens using a comma as a delimiter

IF number of tokens < 2

PRINT "Error: Invalid format, skipping line."

CONTINUE

CREATE Course object

SET Course.courseNumber = first token

SET Course.title = second token

FOR each remaining token in the line (PREREQUISITES)

ADD prerequisite to Course.prerequisites

ADD Course object to courses vector

CLOSE file

RETURN courses

1. ***Hash Table Based:***

FUNCTION LOAD\_COURSE\_DATA (file\_name)

OPEN file at file\_name

IF file cannot be opened

PRINT "Error: Unable to open file."

RETURN

WHILE not end of file

READ line from file

SPLIT line into COURSE DATA LIST

COURSE\_NUMBER <- COURSE DATA LIST [0]

COURSE\_NAME <- COURSE DATA LIST [1]

PREREQUISITES <- COURSE DATA LIST [2 to end] (IF ANY)

CALL CREATE\_COURSE\_OBJECT(COURSE\_NUMBER, COURSE\_NAME, PREREQUISITES)

CLOSE file

1. ***Binary Search Tree Based:***

FUNCTION LOAD\_COURSE\_DATA(file path, BST)

PRINT “Loading CSV file”

OPEN file at file path

IF file cannot be opened

PRINT “Error: cannot open file.”

RETURN

READ first line as header

FOR each line in file

CREATE new Course object

ASSIGN courseNumber = column[0]

ASSIGN title = column[1]

ASSIGN prerequisites = column[2 to end]

INSERT course into BST

CLOSE file

PRINT “File loaded successfully”

END FUNCTION

1. ***Creating and Storing Course Objects***
2. ***Vector Based:***

STRUCT COURSE

STRING courseNumber

STRING title

VECTOR of STRING prerequisites

DECLARE vector courses to store Course objects

1. ***Hash Table Based:***

FUNCTION CREATE\_COURSE\_OBJECT(COURSE\_NUMBER, COURSE\_NAME, PREREQUISITES)

INITIALIZE NEW COURSE as Course object

NEW COURSE.courseNumber <- COURSE\_NUMBER

NEW COURSE.title <- COURSE\_NAME

NEW COURSE.prerequisites <- PREREQUISITES

CALL INSERT\_INTO\_HASH\_TABLE(NEW COURSE)

END FUNCTION

1. ***Binary Search Tree Based:***

FUNCTION INSERT(BST, COURSE)

IF BST is NULL

CREATE new node for COURSE

RETURN

IF COURSE.courseNumber < BST.courseNumber

INSERT(BST.left, COURSE)

ELSE

INSERT(BST.right, COURSE)

1. ***Printing Course Information***

FUNCTION FIND\_COURSE (data\_structure, courseNumber)

SEARCH data\_structure for courseNumber

IF found

PRINT course details and prerequisites

ELSE

PRINT "Course not found."

1. ***Menu System***

FUNCTION DISPLAY\_MENU()

PRINT "1. Load Course Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Information"

PRINT "9. Exit"

INPUT user selection

RETURN user selection

FUNCTION MAIN()

DECLARE data structure (vector, hash table, or BST)

DO

CALL DISPLAY\_MENU()

SWITCH user selection

CASE 1:

CALL LOAD\_COURSE\_DATA(file\_name)

CASE 2:

CALL PRINT\_ALL\_COURSES(data structure)

CASE 3:

INPUT course number

CALL FIND\_COURSE(data structure, course number)

CASE 9:

PRINT "Exiting program."

EXIT

DEFAULT:

PRINT "Invalid selection. Try again."

WHILE user selection != 9

1. ***Printing All Courses in Alphabetical Order***

FUNCTION PRINT\_ALL\_COURSES (data\_structure)

IF data\_structure is a vector

SORT vector alphabetically

ELSE IF data\_structure is a BST

PERFORM in-order traversal

FOR each Course in data\_structure

PRINT courseNumber and title

1. ***Evaluating Runtime and Memory Usage***

Each data structure worked about how I expected, with some standing out for speed and others for organization. The vector was easy to use but not great for searching since it required scanning everything (O(n)), which slowed things down as the dataset grew. The hash table was by far the fastest for lookups, averaging O(1) time, but it didn’t keep things in order, so extra work was needed if I wanted to print courses alphabetically. The binary search tree (BST) did a decent job balancing speed and structure, maintaining O(log n) search times, unless it became unbalanced, in which case it could slow to O(n).

For cost analysis, I assumed each line has a base cost of 1, unless it’s calling a function, in which case the cost depends on the function’s runtime. Insertion in a vector is O(1) if adding to the end but O(n) if inserting in order. Searching also takes O(n). A hash table provides O(1) insertions and lookups, but if there are too many collisions, it can drop to O(n). The BST keeps insertions and searches at O(log n) when balanced, but it can degrade to O(n) if not structured properly.

For printing, a vector requires O(n log n) since sorting is needed before listing courses. A BST can print in O(n) using in-order traversal, while a hash table also needs O(n log n) sorting if the output needs to be ordered.

|  |  |  |  |
| --- | --- | --- | --- |
| OPERATION | VECTOR | HASH TABLE | BINARY SEARCH TREE |
| INSERTION | O(1) (unsorted) / O(n) (sorted) | O(1) (avg) / O(n) (worst) | O(log n) (balanced) |
| SEARCH | O(n) | O(1) (avg) / O(n) (worst) | O(log n) (balanced) |
| SORTING | O(n log n) | N/A | O(n) (in-order traversal) |
| PRINT COURSES | O(n log n) | O(n log n) | O(n) |

1. ***Strengths and Weaknesses of Each Structure***

Each structure has its place depending on the situation. The vector is simple and easy to work with, but it’s inefficient for searching and sorting, making it a bad choice for large datasets. The hash table is the best option for fast lookups, thanks to its O(1) retrieval time in most cases. It makes searching a breeze, but it doesn’t store data in any specific order, which can be a problem if sorted output is needed. The BST is a solid middle-ground solution. It naturally sorts data while keeping searches relatively fast (O(log n)). However, if left unbalanced, performance can degrade, making it harder to justify compared to the other two structures.

1. ***Final Recommendation***

After testing everything, the hash table is the best choice for this project. It delivers the fastest lookup times, making it ideal for retrieving course data quickly. The downside is that it doesn’t maintain order, but sorting can be applied when necessary. The trade-off is worth it for the speed benefits. If keeping courses sorted always was a must, a hybrid approach would make the most sense using a hash table for fast lookups and a BST for structured order. The vector, while straightforward, isn’t a great option here due to its slow search times (O(n)) and sorting overhead. Overall, the hash table provides the best balance between efficiency and usability, making it the ideal choice for managing course data.