Steven Gifford

CS-300-11010-M01 DSA: Analysis and Design

6-2 Submit Project One

August 10, 2025

Milestone 1 – Vector Implementation

Pseudocode – Open, Read, Parse File, and Validate Data

pgsql

CopyEdit

BEGIN

OPEN "courses.txt" FOR reading

IF file could not be opened THEN

PRINT "Error: Could not open file"

EXIT program

ENDIF

INITIALIZE empty list validCourseNumbers

INITIALIZE empty vector<Course> courses

FOR each line in file DO

SPLIT line by ',' into tokens

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

IF number of tokens < 2 THEN

PRINT "Error: Line format incorrect"

EXIT program

ENDIF

CREATE new Course object

SET Course.courseNumber = courseNumber

SET Course.title = courseTitle

INITIALIZE empty list prerequisites

FOR i = 2 to length of tokens - 1 DO

ADD tokens[i] to Course.prerequisites

ENDFOR

ADD courseNumber to validCourseNumbers

ADD Course object to courses vector

ENDFOR

CLOSE file

FOR each course in courses vector DO

FOR each prereq in course.prerequisites DO

IF prereq NOT IN validCourseNumbers THEN

PRINT "Error: Prerequisite", prereq, "not found for course", course.courseNumber

EXIT program

ENDIF

ENDFOR

ENDFOR

END

This pseudocode reads and parses the course data from a file into a vector while validating that all prerequisites exist. The vector stores all courses, and a separate list of valid course numbers ensures prerequisite accuracy.

Pseudocode – Course Struct

arduino

CopyEdit

DEFINE struct Course:

courseNumber : String

title : String

prerequisites : List<String>

INITIALIZE vector<Course> courses

The Course struct holds a course’s number, title, and prerequisites. All courses are stored in a vector for sequential access.

Pseudocode – Search and Print Course Info

pgsql

CopyEdit

FUNCTION searchCourse(courses : vector<Course>, searchCourseNumber : String)

FOR each course in courses DO

IF course.courseNumber == searchCourseNumber THEN

PRINT "Course Number: " + course.courseNumber

PRINT "Title: " + course.title

IF course.prerequisites is not empty THEN

PRINT "Prerequisites:"

FOR each prereq in course.prerequisites DO

PRINT " - " + prereq

ENDFOR

ELSE

PRINT "No prerequisites"

ENDIF

RETURN

ENDIF

ENDFOR

PRINT "Course not found"

END FUNCTION

This function searches the vector for a matching course number and prints its details when found.

Pseudocode – Print All Courses in Sorted Order (Vector)

pgsql

CopyEdit

FUNCTION printAllSortedVector(courses : vector<Course>)

SORT courses by courseNumber in ascending order

FOR each course in courses DO

PRINT course.courseNumber + ", " + course.title

ENDFOR

END FUNCTION

Sorting ensures the course list is in alphanumeric order before printing.

Milestone 2 – Hash Table Implementation

Pseudocode – Load Courses into Hash Table

sql

CopyEdit

FUNCTION LoadCourses(fileName, hashTable)

OPEN course data file with name fileName

IF file cannot be opened

PRINT "Error: Unable to open file."

EXIT program

FOR each line in the file

SPLIT the line into tokens separated by commas

IF number of tokens < 2

PRINT "Error: Missing course number or name."

CONTINUE to next line

SET courseNumber = first token

SET courseName = second token

CREATE empty list prerequisites

FOR each remaining token in the line

ADD token to prerequisites list

CREATE new Course object with:

courseNumber

courseName

prerequisites list

CALL hashTable.InsertCourse(Course)

END FOR

END FUNCTION

Courses are parsed from the file and stored in a hash table for fast lookups.

Pseudocode – Insert into Hash Table

pgsql

CopyEdit

FUNCTION InsertCourse(course)

COMPUTE hashKey = hash(course.courseNumber)

IF no course exists at hashTable[hashKey]

STORE course at hashTable[hashKey]

ELSE

ADD course to the linked list at hashTable[hashKey] // chaining

END FUNCTION

Uses separate chaining to handle collisions.

Pseudocode – Search and Print from Hash Table

pgsql

CopyEdit

FUNCTION SearchCourse(courseNumber)

COMPUTE hashKey = hash(courseNumber)

LOOK for course in hashTable[hashKey]

IF found

RETURN course

ELSE

RETURN "Course not found"

END FUNCTION

FUNCTION PrintCourseInfo(courseNumber)

CALL SearchCourse(courseNumber) → course

IF course = "Course not found"

PRINT "Course not found."

RETURN

PRINT course.courseNumber + ": " + course.courseName

IF course.prerequisites is empty

PRINT "No prerequisites"

ELSE

PRINT "Prerequisites:"

FOR each prereq in course.prerequisites

PRINT prereq

END FUNCTION

The hash table enables O(1) average-case retrieval.

Pseudocode – Print All Courses in Sorted Order (Hash Table)

pgsql

CopyEdit

FUNCTION printAllSortedHashTable(hashTable)

INITIALIZE empty list allCourses

FOR each bucket in hashTable

FOR each course in bucket

APPEND course to allCourses

ENDFOR

SORT allCourses by courseNumber in ascending order

FOR each course in allCourses

PRINT course.courseNumber + ", " + course.title

ENDFOR

END FUNCTION

Gathers all courses from the hash table, sorts them, and prints in alphanumeric order.

Milestone 3 – Binary Search Tree Implementation

Pseudocode – Load Courses into BST

pgsql

CopyEdit

STRUCT Course

STRING courseNumber

STRING name

LIST<STRING> prereqs

END STRUCT

STRUCT Node

Course data

Node\* left

Node\* right

END STRUCT

CLASS CourseBST

Node\* root = null

METHOD insert(Course c)

root = insertRec(root, c)

END METHOD

FUNCTION insertRec(Node\* n, Course c) -> Node\*

IF n == null THEN

RETURN new Node(c, null, null)

END IF

IF c.courseNumber < n.data.courseNumber THEN

n.left = insertRec(n.left, c)

ELSE IF c.courseNumber > n.data.courseNumber THEN

n.right = insertRec(n.right, c)

ELSE

n.data = c

END IF

RETURN n

END FUNCTION

FUNCTION find(String courseNumber) -> Course or EMPTY

Node\* cur = root

WHILE cur != null

IF courseNumber == cur.data.courseNumber THEN

RETURN cur.data

ELSE IF courseNumber < cur.data.courseNumber THEN

cur = cur.left

ELSE

cur = cur.right

END IF

END WHILE

RETURN EMPTY Course

END FUNCTION

END CLASS

Stores courses in sorted order by course number.

Pseudocode – Build Tree from File

sql

CopyEdit

FUNCTION buildTreeFromFile(filePath) -> CourseBST or ERROR

result = loadCourses(filePath)

IF result is ERROR THEN RETURN result END IF

DECLARE bst = new CourseBST

FOR each c IN result

bst.insert(c)

END FOR

RETURN bst

END FUNCTION

Loads and inserts all courses into the BST.

Pseudocode – Print Course Info from BST

pgsql

CopyEdit

FUNCTION printCourseInfo(tree, targetNumber)

c = tree.find(targetNumber)

IF c.courseNumber is empty THEN

PRINT "Course " + targetNumber + " not found."

RETURN

END IF

PRINT c.courseNumber + ", " + c.name

IF c.prereqs is empty THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites: " + join(c.prereqs, ", ")

END IF

END FUNCTION

Retrieves and prints a specific course’s information.

Pseudocode – Print All Courses in Sorted Order (BST)

pgsql

CopyEdit

FUNCTION printAllSortedBST(node)

IF node == null THEN RETURN

printAllSortedBST(node.left)

PRINT node.data.courseNumber + ", " + node.data.name

printAllSortedBST(node.right)

END FUNCTION

An in-order traversal naturally prints the courses in sorted order.

Pseudocode – Menu (All Implementations)

pgsql

CopyEdit

FUNCTION menu()

WHILE TRUE

PRINT "1. Load Courses"

PRINT "2. Print Course List (sorted)"

PRINT "3. Print Course Information"

PRINT "9. Exit"

INPUT choice

IF choice == 1

CALL loadDataStructure()

ELSE IF choice == 2

CALL printAllSorted()

ELSE IF choice == 3

INPUT courseNumber

CALL printCourseInfo(courseNumber)

ELSE IF choice == 9

EXIT

ELSE

PRINT "Invalid choice."

ENDWHILE

END FUNCTION

Provides the core user interface to interact with the chosen data structure.

Advantages and Disadvantages

The vector implementation is simple, intuitive, and preserves the file’s original order, making it easy to traverse and display. However, it is inefficient for searching, requiring O(n) time in the worst case, and insertions or deletions in the middle of the list can be costly. The hash table offers extremely fast average-case search and insertion times, often O(1), making it ideal for quick lookups, but it does not maintain order, requires collision handling, and may need resizing. The binary search tree maintains sorted order and offers O(log n) average-case search and insertion times, but without balancing, its performance can degrade to O(n) in the worst case, and it is more complex to implement.

Recommendation

If speed is the top priority, the hash table is the best choice for most lookups. If sorted order is required without extra sorting, the BST is preferable. The vector is the simplest option but is best suited for smaller datasets due to its slower search times.

Runtime Analysis

| Operation | Vector | Hash Table | BST |
| --- | --- | --- | --- |
| Search | O(n) | O(1) avg, O(n) worst | O(log n) avg, O(n) worst |
| Insert | O(1) amortized | O(1) avg, O(n) worst | O(log n) avg, O(n) worst |
| Delete | O(n) | O(1) avg, O(n) worst | O(log n) avg, O(n) worst |
| Traversal | O(n) | O(n) | O(n) |

Memory usage varies: vectors use contiguous memory and minimal overhead, hash tables require extra space for buckets and linked lists, and BSTs need additional memory for node pointers.