CS 300 – DSA: Analysis and Design — Kayla Macklin

Professor Salem Othman — 08/04/2025

6-2 Project One

Vector Pseudocode

START PROGRAM

OPEN file

IF no file is found THEN

    PRINT “Error: File does not exist”

    RETURN

END IF

CREATE vector <Course> courses

DECLARE variable “lineNumber”

    SET lineNumber = 1

FOR each line in file:

    ++ “lineNumber”

SET each line into a string of tokens

IF (length of tokens > 2):

    PRINT “lineNumber” << “Error: Need more parameters on line.”

ASSIGN first token to variable “courseNumber”

ASSIGN second token to variable “courseName”

IF (length of tokens = 3):

    ASSIGN third token to variable “prerequisite”

CHECK file for course with corresponding courseNumber in “prerequisite”

IF not in file:

    PRINT lineNumber << “Error: No prerequisite found.”

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

    totalPrerequisites = prerequisites of Course c

        for each prerequisite p in totalPrerequisites

            add prerequisites of p to totalPrerequisites

        print totalPrerequisites

}

Void printSampleSchedule (vector <Course> courses) {

}

FOR each course {

```
    IF course = courseNumber

        PRINT course information

        FOR each prerequisite {

            PRINT prerequisite information

}

END PROGRAM
```

Hash Table Data Structure

```
START PROGRAM

CREATE course structure with fields: courseNumber, courseName, coursePrerequisites

CREATE HashTable for course objects

FUNCTION readFile(fileName):

    OPEN fileName

    IF file does not open:

        PRINT "Error: File could not be opened"

        RETURN

    WHILE not at the end of file:

        READ line from file

        SPLIT line into tokens
```

```
        IF number of tokens < 2:
            PRINT "Error: Number of tokens must be greater than 2."
            CONTINUE
        CREATE course object
        SET courseNumber and courseName from tokens
        IF number of tokens > 2:
            FOR each token:
                IF token not courseName in fileName:
                    PRINT "Error: Course does not exist."
                ELSE:
                    ADD token to coursePrerequisites
            ADD course to HashTable
        CLOSE fileName
    FUNCTION printCourses(HashTable):
        FOR each course in HashTable:
            PRINT courseNumber, courseName, coursePrerequisites
    FUNCTION main():
        CALL readFile with data fileName
```

CALL printCourses with HashTable

END PROGRAM

Binary Tree Data Structure

START PROGRAM

Reading the File

USE fstream to OPEN file

CALL to OPEN file:

IF return value = -1 THEN PRINT “Error: file not found”

ELSE:

file can be found

OPEN file

WHILE not end of file

READ each line

IF < 2 values in line, RETURN “Error”

ELSE:

read parameters

IF there is a third or more parameter:

IF third or more parameter is in first parameter, ELSEWHERE continue

ELSE:

RETURN "Error"

CLOSE file

Creating course objects structure

INITIALIZE Course Structure struct Course

LOOP through file, WHILE not end of file

FOR each line in file

FOR first and second value

ADD courseNumber and courseName

IF a third value exists

THEN ADD prerequisites until newline found

Creating tree and add nodes

DEFINE Binary Tree Class

CREATE a root that points to null

INSERT method

IF root = null THEN current Course is Root

ELSE is courseNumber < Root, ADD left

IF left = null THEN ADD courseNumber

ELSE:

IF courseNumber < leaf, ADD left

IF courseNumber > leaf, ADD right

ELSE IF courseNumber > root, ADD right

IF right = null, ADD courseNumber

ELSE:

IF courseNumber < leaf, ADD left

IF courseNumber > leaf, ADD right

Search and print from Tree

ASK for input

Print method

IF root is not null, THEN:

traverse left, output if found

traverse right, output if found

END PROGRAM

Run Time Analysis

Vector

Code	Line Cost	# Time Executes	Total Cost
For all courses	1	n	n
If the course is the same as courseNumber	1	n	n
Print out the course information	1	1	1
For each prerequisite of the course	1	n	n
Print the prerequisite course information	1	n	n
		Total Cost	4n+1
		Runtime	O(n)

Hash Table

Code	Line Cost	# Times Executes	Total Cost
For all courses	1	n	n

If course is the same as courseNumber	1	n	n
Print out the course information	1	1	1
For each prerequisite of the Hashtable[course]	1	n	n
Print the prerequisite course information	1	n	n
		Total Cost	4n + 1
		Runtime	O(n)

Tree Data Structure

Code	Line Cost	# Times Executes	Total Cost
For all Nodes	1	n	n
If the course is the same as courseNumber	1	n	n

Print out the node's information	1	1	1
If course has left node	1	n/2	n/2
Print left node as prerequisite course information	1	n	n
If course has right node	1	n/2	n/2
Print right node as prerequisite course information	1	1	1
		Total Cost	2(n/2) + 3n + 2
		Runtime	O(n)

Advantages and Disadvantages

Vector Data Structure

Advantages	**Disadvantages**
Vectors automatically adjust their size when elements are added or taken away, which makes them flexible in terms of sizing	Adding or removing elements from anywhere except the end means you have to move other elements around

Elements are kept in one block of memory, enabling quick access and improved cache performance	Vectors need a big chunk of continuous memory. When memory is fragmented in systems, it can cause allocation failures
Vectors manage memory automatically. They take care of allocating and freeing up memory on their own, which helps reduce the chances of memory leaks	There is no built-in feature for quick searching. Finding an element takes O(n) time unless the vector is sorted
Vectors are versatile and can hold any kind of data type	Vectors increase by a set amount, which can result in wasted memory if the growth exceeds what is actually required

Hash Table Data Structure

Advantages	**Disadvantages**
Hash tables allow for constant time lookups. When you access, insert, or delete elements, it usually takes O(1) time. This is really beneficial in situations where performance is crucial, such as in real-time systems	Hash tables do not keep the order of elements, so if you require sorted data, this data structure is not the best choice
Hash tables don't need the elements to be sorted	When multiple keys hash to the same index, it leads to a collision that can create extra work and reduce performance

Keys in hash tables can include strings, numbers, and sometimes even custom objects. This flexibility makes hash tables perfect for organizing complex data	Hash tables usually use more memory than needed to minimize collisions, which can be a problem in situations where memory is limited
Hash tables are great for managing large datasets	If hash functions are not well designed, it can cause clustering and performance problems

Binary Tree Data Structure

Advantages	**Disadvantages**
Tree data structures are great for modeling hierarchical relationships, which makes it simple to show parent-child structures	Trees need complex logic to keep their structure intact. This complexity makes it challenging to carry out insertion, deletion, and balancing operations, as well as to troubleshoot them
This data structure allows for flexible data management and can expand or contract without the need to reallocate big chunks of memory	Hash tables are quicker for searching, adding, and removing elements compared to tree data structures
A balanced tree provides O(log n) time for searching, inserting, and deleting, which makes searching more efficient	Every node keeps several pointers, which increases memory usage. When you compare them to hash tables, trees are not as efficient in terms of memory

When you do an in-order traversal of a binary search tree, it gives you the elements sorted. This is perfect for situations where you need sorted data or when you're setting up priorities	Traversing a tree requires O(n) time, which can be costly for larger tree data structures.

After looking at the three data structures – vectors, hash tables, and trees, I think the best option for coding is the Hash Table. This choice is based on its excellent performance regarding time complexity for key operations. Hash tables provide constant-time complexity, O(1), for accessing, inserting, and deleting when using keys, which makes them perfect for applications that need quick lookups and updates.

On the other hand, vectors allow O(1) access by index but have O(n) search and deletion times, especially when you need to change elements in the middle of the structure. Trees, especially balanced binary search trees, have O(log n) performance for most operations and are useful when you need ordered traversal or range queries. However, they are usually slower than hash tables for simple key-based access.

Hash tables work really well when the order doesn't matter, and the dataset is large or changing. They are great for things like caching, indexing, and dictionary-style mappings, where fast access and updates are super important. While trees are better for keeping sorted data or doing hierarchical queries, and vectors are handy for indexed access and sequential storage, hash tables offer the best mix of speed and simplicity for most general programming tasks. Overall, I believe that a hash table is the most efficient and practical data structure to go with.

References

Team. (2024, October 12). *A comprehensive Big O notation guide for beginners*. AlgoCademy Blog. https://algocademy.com/blog/a-comprehensive-big-o-notation-guide-for-beginners/

Big o notation cheat sheets. (n.d.). https://www.big-o.academy/cheatsheet/