**6-2 Project One**

Katrina S. Gilliland

Global Campus, Southern New Hampshire University

CS-300-H2986 DSA: Analysis and Design

Instructor Ricardo Scarello

# December 3, 2023

**6-2 Project One**

**Vector Data Structure Pseudocode**

**Opening file and checking for errors:**

#Include <fstream>

#Include <vector>

Use ifstream class to open text file;

IF file is not found {

OUTPUT “File cannot be found”;

CLOSE file;

}

WHILE file is found AND is not at the EOF {

READ line;

IF line holds fewer than two values OR next line does not start with courseNumber {

CLOSE file;

}

}

**Creating Course Objects and Storing Them in the Vector Data Structure:**

INITIALIZE vector as vector<int> courses;

ASSIGN courses.courseNumber to courseNumber;

ASSIGN courses.name to name;

ASSIGN courses.prerequisites to p and initialize to 0;

WHILE file is found and it is not at the EOF {

IF data is in file and was not caught by read file error {

READ line and use push back to add data to the vector;

}

}

**Program menu:**

**SET userChoice to zero;**

**WHILE userChoice is not equal to four {**

**SWITCH(userChoice) {**

**Case 1: LOAD the vector structure and LOAD data from text file into vector <int> courses; // Option to load data into structure**

**Case 2: OUTPUT course list (sort in alphanumeric order); // Option to print course list**

**Case 3: OUTPUT course title and prerequisites (sort in ascending order); // Option to print course title and prerequisites for any individual course**

**Case 4: CLOSE program; // Option to exit program;**

**GET userChoice; // Prompt user for input;**

**}**

**}**

**Print list of courses:**

**SORT course information by courseNumber (from lowest to highest);**

**OUTPUT sorted list;**

**Searching the data structure for a specific course and printing out the course information and prerequisites:**

void printCourseInformation(Vector<Course> courses, String courseNumber) {

**PROMPT user for input (specific course)**

**FOR all courses {**

**If course matches the user input {**

**OUTPUT course information;**

**}**

**FOR each course prerequisite {**

**OUTPUT prerequisite course information;**

**}**

**}**

**Hash Table Data Structure Psuedocode**

**Opening file and checking for errors:**

INITIALIZE empty set for storing courseNumbers;

FOR each line file {

SPLIT line into two tokens (token, token);

IF token length is less than 2 {

RETURN error;

}

}

GET courseNumber from tokens;

ADD courseNumber from tokens to empty set initialized above;

CHECK if prerequisite (last token) exists in file;

IF prerequisite does not exist in the file {

RETURN error;

}

RETURN file; // If all lines are processed successfully

**Creating course objects and storing them in the hash table data structure:**

INITIALIZE empty hash table for storing course objects;

FOR each line in file {

SPLIT line into tokens;

}

GET courseNumber, title, and prerequisites from tokens;

CREATE new course object using data from tokens;

ADD course object to hash table;

RETURN hash table with course objects;

**Program menu:**

**SET userChoice to zero;**

**WHILE userChoice is not equal to four {**

**SWITCH(userChoice) {**

**Case 1: LOAD the hash table structure and LOAD data from text file into the hash table; // Option to load data into structure**

**Case 2: OUTPUT course information stored in hash table (sort in alphanumeric order); // Option to print course list**

**Case 3: OUTPUT course key using courseNumber (sort in ascending order); // Option to print course title and prerequisites for any individual course**

**Case 4: CLOSE program; // Option to exit program;**

**GET userChoice; // Prompt user for input;**

**}**

**}**

**Print list of courses:**

**SORT course information by courseNumber (from lowest to highest);**

**OUTPUT sorted list;**

**Searching the data structure for a specific course and printing out the course information and prerequisites:**

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

FOR each courseNumber hash table { // Searching data structure

GET course object using key (courseNumber);

**OUTPUT prerequisite course information;**

**}**

**}**

**Tree Data Structure Pseudocode**

**Opening file and checking for errors:**

OPEN file using fstream;

FOR each line in open file {

SPLIT line into two tokens (“,”);

IF token length is less than two {

RETURN error since line does not contain at least two parameters;

GET next line;

}

IF token length is greater than two {

RETURN error since line cannot contain more than two parameters;

GET next line;

}

IF token length is equal to two {

ASSIGN first token/parameter to a variable named courseNumber;

ASSIGN second token/parameter to a variable named courseTitle;

ELSE {

IF token length is equal to three {

ASSIGN third token/parameter to a variable named prerequisite;

}

CLOSE file;

}

}

**Creating course objects and storing them in the vector data structure:**

CREATE empty vector named courses;

FOR each line in file {

SPLIT line into tokens (“,”);

CREATE course object named course;

ASSIGN first token in line to courseNumber variable in course;

ASSIGN second token in line to courseTitle variable in course;

IF the length of the tokens is equal to three {

ASSIGN third token in line to prerequisite variable in course;

}

PUSH course to the empty courses vector;

}

**Program menu:**

**SET userChoice to zero;**

**WHILE userChoice is not equal to four {**

**SWITCH(userChoice) {**

**Case 1: LOAD the tree structure and LOAD data from text file into the tree; // Option to load data into structure**

**Case 2: OUTPUT course information in tree (sort in alphanumeric order); // Option to print course list**

**Case 3: OUTPUT course key using courseNumber (sort in ascending order); // Option to print course title and prerequisites for any individual course**

**Case 4: CLOSE program; // Option to exit program;**

**GET userChoice; // Prompt user for input;**

**}**

**}**

**Print list of courses:**

**SORT course information by courseNumber (from lowest to highest);**

**OUTPUT sorted list;**

**Searching the data structure for a specific course and printing out the course information and prerequisites:**

CREATE empty tree structure;

FOR each course in the courses vector { // Searching vector structure

USE courseNumber as key and ADD course to empty tree;

**}**

**FOR each key in the tree {**

**OUTPUT the key; // Outputting courseNumber**

**OUTPUT the courseTitle;**

**IF course has a prerequisite {**

**OUTPUT prerequisite and prerequisite courseNumber;**

**}**

**}**

**Vector Data Structure Run-Time**

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Use ifstream class to open text file; | 1 | 1 | 1 |
| IF file is not found { | 1 | 1 | 1 |
| OUTPUT “File cannot be found”; | 1 | 1 | 1 |
| CLOSE file; } | 1 | 1 | n |
| WHILE file is found AND is not at the EOF { | 1 | n | n |
| READ line; | 1 | n | n |
| IF line holds fewer than two values OR next line does not start with courseNumber { | 1 | n | n |
| CLOSE file; | 1 | 1 | 1 |
| INITIALIZE vector as vector<int> courses; | 1 | 1 | 1 |
| ASSIGN courses.courseNumber to courseNumber; | 1 | n | n |
| ASSIGN courses.name to name; | 1 | n | n |
| ASSIGN courses.prerequisites to p and initialize to 0; | 1 | n | n |
| WHILE file is found and it is not at the EOF { | 1 | n | n |
| IF data is in file and was not caught by read file error { | 1 | 1 | 1 |
| READ line and use push back to add data to the vector; | 1 | n | n |
| **Total Cost** | | | 9n+6 |
| **Runtime** | | | O(n) |

**Hash Table Data Structure Run-Time**

| **Hash Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| INITIALIZE empty set for storing courseNumbers; | 1 | 1 | 1 |
| FOR each line file { | 1 | n | n |
| SPLIT line into two tokens (token, token); | 1 | n | n |
| IF token length is less than 2 { | 1 | n | n |
| RETURN error; | 1 | n | n |
| GET courseNumber from tokens; | 1 | n | n |
| ADD courseNumber from tokens to empty set initialized above; | 1 | n | n |
| CHECK if prerequisite (last token) exists in file; | 1 | n | n |
| IF prerequisite does not exist in the file { | 1 | n | n |
| RETURN error; | 1 | 1 | 1 |
| RETURN file; | 1 | n | n |
| INITIALIZE empty hash table for storing course objects; | 1 | 1 | 1 |
| FOR each line in file { | 1 | n | n |
| SPLIT line into tokens; | 1 | n | n |
| GET courseNumber, title, and prerequisites from tokens; | 1 | n | n |
| CREATE new course object using data from tokens; | 1 | n | n |
| ADD course object to hash table; | 1 | n | n |
| RETURN hash table with course objects; | 1 | 1 | 1 |
| **Total Cost** | | | 14n + 4 |
| **Runtime** | | | O(n) |

**Tree Data Structure Run-Time**

| **Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OPEN file using fstream; | 1 | 1 | 1 |
| FOR each line in open file { | 1 | n | n |
| SPLIT line into two tokens (“,”); | 1 | n | n |
| IF token length is less than two { | 1 | n | n |
| RETURN error since line does not contain at least two parameters; | 1 | 1 | 1 |
| GET next line; | 1 | n | n |
| IF token length is greater than two { | 1 | n | n |
| RETURN error since line cannot contain more than two parameters; | 1 | 1 | 1 |
| GET next line; | 1 | n | n |
| IF token length is equal to two { | 1 | n | n |
| ASSIGN first token/parameter to a variable named courseNumber; | 1 | n | n |
| ASSIGN second token in line to courseTitle variable in course; | 1 | n | n |
| ELSE { | 1 | n | n |
| IF token length is equal to three { | 1 | n | n |
| PUSH course to the empty courses vector; | 1 | n | n |
| **Total Cost** | | | 12n+3 |
| **Runtime** | | | O(n) |

**Evaluation of Vector Data Structure:**

The vector data structure is super dynamic. Vectors make it simple to insert and remove elements by automatically adjusting the vector size. Using iterators, vectors make it easy to navigate through and retrieve a program’s items. While there are many advantages to using a vector data structure, there are also some disadvantages; for example, vectors may consume more memory than other structures because they allocate more memory than needed to make room for changes in the vector and vector growth.

**Evaluation of Hash Table Data Structure:**

The benefits of using a hash table data structure are that they offer a quick and efficient way to search, create, and remove stored data. Regardless of the size of the data, hash tables typically have a constant time complexity for conducting these types of operations; this differs from data structures like search trees, which utilize a linear time complexity. A disadvantage of using hash tables is that they can be met with many collisions and the chance that a collision will occur only increases as the amount of data grows.

**Evaluation of Tree Data Structure:**

Tree data structures organize data in a hierarchial manner, which is useful for defining the relationships and connections between the various elements in the data. Because tree structures define these relationships and connections so well, it is easy to search for and retrieve data, however, a disadvantage of using a tree data structure is that it can be temperamental; any change in the data, no matter the size, can cause a large change in the structure of the tree, thus potentially causing instability in the program.

**Recommendation for Program’s Data Structure:**

Based on the Big O analysis results and my evaluation of the three data structures, my recommendation would be for the program to be built using a vector data structure. Observing my Big O analysis tables, the lines of code written for the vector data structure will cost less than the hash table and tree structures. In addition to a less costly run-time and as noted in my evaluation, the vector data structure is extremely flexible, making it easier to modify and fit to the requirements of ABCU’s academic advisors.