Milton Francisco   
CS-300 06MAY2024

**CS 300 Pseudocode Project One**

## Vector Pseudocode - Milestone 1

bool verifyFileFormat(string file){

//Could also consider using built-in ifstream& for file

**Create an inputFile pointer**

**Open the file**

**If the file does not open successfully**

**Output error message**

**Return**

**Create a vector<string> for course Numbers**

**Create a vector<string> for prereqs**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Use stringstream on line**

**Create a string “token” variable**

**Create a temp vector to hold the tokens**

**For each token in line**

**Push token to temp vector**

//Check for at least two parameters, after temp vector populated

**If temp vector size < 2**

**Output error message**

**Return false**

**Else**

**For each token in temp vector**

**Push first token to courseNumbers vectors**

**Push tokens after second token to prereqs vector**

**Close inputFile**

// Verify prereq exists as a course in the file (Separate in case courses are out of order in the file)

**Loop through prereqs vector**

**If prereq not in course Numbers vector**

**Output error message**

**Return false**

**Return true**

}

struct Course {

**string courseNumber**

**string title**

**vector<string> preReqs**

}

//File is already verified before using this function

vector<Course> loadCourses(string file) {

**Create an inputFile pointer**

**Open the file**

**Create a vector<Course> courses**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Create a Course object**

**Use stringstream on line**

**//Using getline on stringstream w/comm as delim**

**Set Course object courseNumber as first token**

**Set Course object title as second token**

**Loop through remaining tokens from line**

**Push token into preReqs vector**

**Push Course object to courses vector**

**Return courses vector**

}

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

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}  
void printCoursesInOrder(Vector<Course> courses) {

// Could sort here, or could sort separately  
 **For every course in courses  
 Output courseNumber  
 Output title**

**For each prerequisite**

**Output prerequisite**

}

## Hash Table Pseudocode- Milestone 2

//Global Variable:   
**unsigned int defaultSize = 0;**

//This is the same file parser used to verify file integrity as above

bool verifyFileFormat(string file){

//Could also consider using built-in ifstream& for file

**Create an inputFile pointer**

**Open the file**

**If the file does not open successfully**

**Output error message**

**Return**

**Create a vector<string> for course Numbers**

**Create a vector<string> for prereqs**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Use stringstream on line**

**Create a string “token” variable**

**Create a temp vector to hold the tokens**

**For each token in line**

**Push token to temp vector**

//Check for at least two parameters, after temp vector populated

**If temp vector size < 2**

**Output error message**

**Return false**

**Else**

**Push first token to courseNumbers vectors**

**For each token after 2nd token in temp vector**

**Push token to prereqs vector**

**Increment defaultSize** //amount of courses in file

**Close inputFile**

// Verify prereq exists as a course in the file (Separate in case courses are out of order in the file)

**Loop through prereqs vector**

**If prereq not in course Numbers vector**

**Output error message**

**Return false**

**Return true**

}

//structure to hold course information

struct Course {

**string courseNumber**

**string title**

**vector<string> preReqs**

}

// Linear Probing Hash Table

class HashTable {

private:

**struct Node {** //Will probably change in implementation

**Course course;**

**unsigned int key;**

**bool isEmpty = true;**

**bool isEmptySinceStart = true;  
 }**

//Incremented in verifyFile() function

**unsigned int tableSize = DEFAULT\_SIZE;**

**vector<Node> courses;**

**unsigned int hash(string courseNumber){**

//str.erase is O(n) time complexity

**Remove first 4 letters and set as key**

**Return key modulo tableSize**

**}**

**public:**

**HashTable();**

**Insert(Course course);**

**Search(Course course);**

**inOrder();**

}

void HashTable::HashTable()**{**

**nodes.resize(tableSize)**

**}**

bool HashTable::Insert(Course course) **{**

**Find the key using hash(course.courseNumber)**

**Find the course Node from the key**

**Create a bucketsProbed variable**

**Loop while bucketsProbed < tableSize**

**If Node is empty**

**Set Node course to course**

**Set Node key to key**

**Set Node isEmpty to false**

**Set Node isEmptySinceStart to false**

**Return true**

**Increment bucket index using modulo tableSize**

**Increment number of buckets probed**

**Return false**

**}**

Course HashTable::Search(Course courseNumber) **{**

**Find the key using hash(courseNumber)**

**Create a bucketsProbed variable**

**Loop while bucket is not empty-since-start & bucketsProbed < tableSize**

**If bucket is not empty and courseNumber matches**

**Return course**

**Else Increment bucket index**

**Increment buckets probed**

**If not found**

**Output Not found message**

**}**void HashTable::inOrder(){ //Hash tables are inherently unsorted.

**//Extract courses vector of node type into a vector of courses  
 Create a vector of size tableSize**

**For every node in courses  
 Push only the Course object to the local vector  
  
 Call std::sort function with helper function to sort local vector**

**For every Course in local vector  
 Output courseNumber  
 Output title  
 For each prerequisite  
 Output prerequisite**

}

// if verifyFileFormat returns true, load courses

void loadCourses(string csvPath, HashTable\* courses) {

**Create an inputFile pointer**

**Open the file**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Create a Course object**

**Use stringstream on line**

**//Using getline on stringstream w/comm as delim**

**Set Course object courseNumber as first token**

**Set Course object title as second token**

**Loop through remaining tokens from line**

**Push token into preReqs vector**

**Insert Course object into hashtable using Insert(course)**

**If Insert(course) returns false**

**Output error message**

**Return**

}

void searchCourse(HashTable<Course> courses, String courseNumber) {

**Create a Course variable equal to courses.Search(courseNumber)**

**If it is an empty course  
 Output “course not found” message  
 Return**

**Output courseNumber**

**Output title  
 For each prerequisite  
 Output prerequisite string**

}

## Tree Pseudocode – Milestone 3

bool verifyFileFormat(string file){

//Could also consider using built-in ifstream& for file

**Create an inputFile pointer**

**Open the file**

**If the file does not open successfully**

**Output error message**

**Return**

**Create a vector<string> for course Numbers**

**Create a vector<string> for prereqs**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Use stringstream on line**

**Create a string “token” variable**

**Create a temp vector to hold the tokens**

**For each token in line**

**Push token to temp vector**

//Check for at least two parameters, after temp vector populated

**If temp vector size < 2**

**Output error message**

**Return false**

**Else**

**For each token in temp vector**

**Push first token to courseNumbers vectors**

**Push tokens after second token to prereqs vector**

**Close inputFile**

// Verify prereq exists as a course in the file

**Loop through prereqs vector**

**If prereq not in course Numbers vector**

**Output error message**

**Return false**

**Return true**

struct Course {

**string courseNumber**

**string title**

**vector<string> preReqs**

}

struct Node {

**Course course;**

**Node\* left;**

**Node\* right;**

**Node() {**

**left = nullptr;**

**right = nullptr;**

**}**

**Node(Course acourse):**

**Node() {**

**course = acourse;**

**}**

}

class Tree { //Could make a balancing approach (AVL/Red-black) later

private:

**Node\* root;**

**void addNode(Node\* node, Course course);**

**Node\* removeNode(Node\* node, string courseNumber);**

**void destroyTree(Node\* node);**

**void inOrder(Node\* node);**

public:

Tree() **{ root = nullptr; }**

virtual ~Tree() **{ destroyTree(root); }**

void InOrder**( return inOrder(root); )**

**void Insert(Course course);**

**Course Search(string courseNumber);**

}

void Tree::inOrder(Node\* node) { **If the node is null  
 Return  
 Call inOrder on left child  
 Output this node’s course information  
 Call inOrder on right child**  
}

void Tree::Insert(Course course) {  
 **If root is a nullptr  
 Create a new Node with course as argument  
 Set root as new Node  
 Else  
 Call addNode using root and course as arguments**   
}

void Tree::addNode(Node\* node, Course course) {

**Compare course courseNumber against node’s courseNumber**

**If comparison < 0 (course is smaller than node)  
 If the node’s left child is null  
 Set node’s left child as a new Node(course)  
 Else  
 Call addNode on node’s left child with course as arg  
 Else (course is larger than node)  
 If node’s right child is null  
 Set node’s right child as a new Node(course)  
 Else  
 Call addNode on node’s right child with course as arg**  
}

Course Tree::Search(string courseNumber) {

**Create a current pointer pointing to root  
   
 Loop while current is not null  
 Compare courseNumber param to current node’s courseNumber  
  
 If comparison equals 0 (match)  
 Return current node’s course  
  
 Else if comparison shows courseNumber smaller ( < 0)  
 Set current pointer equal to current’s left child  
  
 Else (comparison is > 0)  
 Set current pointer equal to current’s right child  
  
 //If this is reached, the course was not found  
 Create an empty course  
 Return empty course**}

void loadCourses(string csvPath, Tree\* bst) {

**Create an inputFile pointer**

**Open the file**

**Create a string “line” to hold line from file**

**Loop while there are lines in inputFile**

**Create a Course object**

**Use stringstream on line**

**//Using getline on stringstream w/comm as delim**

**Set Course object courseNumber as first token**

**Set Course object title as second token**

**Loop through remaining tokens from line**

**Push token into preReqs vector**

**Insert Course object into Tree using bst->Insert(course)**

**Return**

}

void searchCourse(Tree<Course> courses, String courseNumber) {

//This could be implemented differently and should change later

**Create a Course variable equal to courses.Search(courseNumber)**

**If it is an empty course  
 Output “course not found” message  
 Return**

**Output courseNumber**

**Output title  
 For each prerequisite  
 Output prerequisite string**

}

## Main Menu

//Implement data structure initialization before this  
**Create a userChoice variable equal to 0.**  
**Loop while userChoice does not equal 9  
 Output menu title  
 Output Option 1 - Load courses into structure  
 Output Option 2 – Print ordered list of courses  
 Output Option 3 – Print Course title and prerequisites  
 Output Option 9 – Exit Program  
 Output prompt for user choice entry  
 Obtain user choice in userChoice variable (using cin)  
 switch (userChoice)  
 case 1:** //If we would like to show how long functions take we can have a clock() here **Call loadCourses function passing in filePath and specific data structure** // We could then calculate how long the function took here **break (end case)  
 case 2:  
 Call data structure’s specific Print inOrder function  
 break (end case)  
 case 3:  
 Create a courseChoice string variable  
   
 Loop while courseChoice is empty string  
 Prompt user for course to search for   
 Obtain courseChoice from user input** (cin) **If courseChoice is proper format  
 Call data structure’s specific Search function with courseChoice  
 Set result equal to a tempCourse variable   
 Output courseNumber  
 Output title  
 For each prerequisite   
 Output prerequisite  
 Else  
 Output incorrect usage message  
 Set courseChoice to emptyString  
 break (end case)  
 default:** //Handles if user input is not a number and prevents infinite loop **Clear instream  
 Ignore previous user input up to the newline  
 break (end case)** //User choice equals 9 so this is now reached  
**Output Goodbye**  
**Return proper exit code**

## Vector Runtime Analysis

| **verifyFileFormat() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create an inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Check if file opened successfully** | 1 | 1 | 1 |
| **Create a courseNumbers vector** | 1 | 1 | 1 |
| **Create a preReqs vector** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **For all lines in input file** | 1 | n | n |
| **Create a stringstream for the line** | 1 | n | n |
| **Create a token variable** | 1 | n | n |
| **Create a temp vector** | 1 | n | n |
| **For each token in line** | 1 | p | n \* p |
| **Push token to temp vector** | 1 | p | n \* p |
| **If temp vector size < 2** | 1 | n | n |
| **Push token to courseNumbers vector** | 1 | n | n |
| **For each token in temp after second** | 1 | p | n \* p |
| **Push token to preReqs vector** | 1 | p | n \* p |
| **Close input file** | 1 | 1 | 1 |
| **For each prereq in preReqs** | 1 | q | q |
| **Check if prereq in courseNumbers** | 1 | q | q |
| **If it is not throw error** | 1 | q | q |
|  |  |  |  |
| **Total Cost** | | 4(n\*p) + 3q + 6n + 7 | |
| **Runtime** | | O(n \* p) | |

| **loadCourses() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Create a courses Vector** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **Loop while through lines in inputFile** | 1 | n | n |
| **Create a Course Object** | 1 | n | n |
| **Create a stringstream on line** | 1 | n | n |
| **Set Course object courseNumber** | 1 | n | n |
| **Set Course object title** | 1 | n | n |
| **For each remaining token** | 1 | p | n \* p |
| **Push token to Course prereq vector** | 1 | p | n \* p |
| **Push Course Object to courses Vector** | 1 | n | n |
| **Return courses vector** | 1 | 1 | 1 |
| **Total Cost** | | 6n + 2np + 5 | |
| **Runtime** | | O (n \* p) | |

## HashTable RunTime Analysis

| **verifyFileFormat() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create an inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Check if file opened successfully** | 1 | 1 | 1 |
| **Create a courseNumbers vector** | 1 | 1 | 1 |
| **Create a preReqs vector** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **For all lines in input file** | 1 | n | n |
| **Create a stringstream for the line** | 1 | n | n |
| **Create a token variable** | 1 | n | n |
| **Create a temp vector** | 1 | n | n |
| **For each token in line** | 1 | p | n \* p |
| **Push token to temp vector** | 1 | p | n \* p |
| **If temp vector size < 2** | 1 | n | n |
| **Push token to courseNumbers vector** | 1 | n | n |
| **For each token in temp after second** | 1 | p | n \* p |
| **Push token to preReqs vector** | 1 | p | n \* p |
| **Increment default size variable** | 1 | n | n |
| **Close input file** | 1 | 1 | 1 |
| **For each prereq in preReqs** | 1 | q | q |
| **Check if prereq in courseNumbers** | 1 | q | q |
| **If it is not throw error** | 1 | q | q |
| **Total Cost** | | 4(n\*p) + 3q + 7n + 7 | |
| **Runtime** | | O(n \* p) | |

| **loadCourses() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Create a courses Vector** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **Loop through lines in inputFile** | 1 | n | n |
| **Create a Course Object** | 1 | n | n |
| **Create a stringstream on line** | 1 | n | n |
| **Set Course object courseNumber** | 1 | n | n |
| **Set Course object title** | 1 | n | n |
| **For each remaining token** | 1 | p | n \* p |
| **Push token to Course prereq vector** | 1 | p | n \* p |
| **Call Insert(course) for hashtable** | O(n) | n | n2 |
| **Total Cost** | | n2 \* 6n + 2np + 5 | |
| **Runtime** | | O (n2) | |

| **HashTable::Insert() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Find the key using hash(courseNumber)** | O(n) | 1 | O(n) |
| **Find the course Node from key** | 1 | 1 | 1 |
| **Create a bucketsProbed variable** | 1 | 1 | 1 |
| **Loop while bucketsProbed < tableSize** | 1 | n | n |
| **If node empty** | 1 | n | n |
| **Set node course to course** | 1 | 1 | 1 |
| **Set node key to key** | 1 | 1 | 1 |
| **Set isEmpty to false** | 1 | 1 | 1 |
| **Set isEmptySinceStart to false** | 1 | 1 | 1 |
| **Return true** | 1 | 1 | 1 |
| **Increment bucket index** | 1 | n | n |
| **Increment bucketsProbed** | 1 | n | n |
| **Return false** | 1 | 1 | 1 |
| **Total Cost** | | O(n) + 4n + 7 | |
| **Runtime** | | O(n) | |

## Tree Runtime Analysis

| **verifyFileFormat() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create an inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Check if file opened successfully** | 1 | 1 | 1 |
| **Create a courseNumbers vector** | 1 | 1 | 1 |
| **Create a preReqs vector** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **For all lines in input file** | 1 | n | n |
| **Create a stringstream for the line** | 1 | n | n |
| **Create a token variable** | 1 | n | n |
| **Create a temp vector** | 1 | n | n |
| **For each token in line** | 1 | p | n \* p |
| **Push token to temp vector** | 1 | p | n \* p |
| **If temp vector size < 2** | 1 | n | n |
| **Push token to courseNumbers vector** | 1 | n | n |
| **For each token in temp after second** | 1 | p | n \* p |
| **Push token to preReqs vector** | 1 | p | n \* p |
| **Close input file** | 1 | 1 | 1 |
| **For each prereq in preReqs** | 1 | q | q |
| **Check if prereq in courseNumbers** | 1 | q | q |
| **If it is not throw error** | 1 | q | q |
| **Total Cost** | | 4(n\*p) + 3q + 6n + 7 | |
| **Runtime** | | O(n \* p) | |

| **loadCourses() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create inputFile pointer** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| **Create a string variable** | 1 | 1 | 1 |
| **Loop through lines in inputFile** | 1 | n | n |
| **Create a Course Object** | 1 | n | n |
| **Create a stringstream on line** | 1 | n | n |
| **Set Course object courseNumber** | 1 | n | n |
| **Set Course object title** | 1 | n | n |
| **For each remaining token** | 1 | p | n \* p |
| **Push token to Course prereq vector** | 1 | p | n \* p |
| **Call Insert(course) for the tree** | O(n) | n | n2 |
| **Total Cost** | | n2 + 2np + 3 | |
| **Runtime** | | O(n2) | |

| **Tree::Insert() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If root is nullptr** | 1 | 1 | 1 |
| **Call addNode w/root and course as args** | O(n) | 1 | O(n) |
| **Total Cost** | | O(n) + 1 | |
| **Runtime** | | O(n) | |

| **Tree::addNode() Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Compare course number against node** | 1 | 1 | 1 |
| **If comparison < 0** | 1 | 1 | 1 |
| **Check if node’s left child is null** | 1 | 1 | 1 |
| **Else Call addNode on left child** | 1 | n | n |
| **//Even in worst case only the if or else block will execute and they have same time complexity** |  |  |  |
| **Total Cost** | | O(n) + 3 | |
| **Runtime** | | O(n) | |

## ****Advantages and Disadvantages****

The data structures being considered for use are a vector, hash table, and tree. If we inspect the runtime of the three data structures, we can see that the vector has the fastest runtime for loading courses. Both the hash table and tree structures have a worst-case runtime of O(n2), while the vector is O(n \* p) where n is the number of lines in the file, and p is the class with the largest number of prereqs. While unlikely, there is a potential for a class that has n – 1 prerequisites. This would make the vector’s runtime closer to O(n2) than O(n). In the average case, it is closer to O(n). All of this said, these runtimes are only for loading a file into a structure.

The advisors wish to have a program that prints a list of all courses in alphanumeric order, as well as print out a given course. These runtimes were not desired in this document, but will greatly affect the runtime of this program. For the vector, using std::sort from the C++ library, the runtime for sorting is O(NlogN). Using this will display the alphanumeric list in O(n), and finding a specific course will be worst-case O(n) as well.

The hash table has a larger load time and is inherently unsorted. To sort this, the values must be extracted and then sorted, which coud cause a longer runtime. This data structure is not the best for displaying values in order. The hashtable will usually excel in searching for a specific course, but if all the courses have the same hash key, it could have a similar time complexity to the vector. On average, the runtime will be closer to O(1) when searching for a specific course.

Moving on to the tree data structure, while the load time is more than the vector, it excels in searching for a course. When searching for a specific course, if the tree is properly balanced, the runtime could be O(logN), which is faster than the vector. It should be noted that properly balancing a tree could introduce a larger load time. Displaying all courses in order has a runtime of O(NlogN), in a perfectly balance, fully-occupied tree.

For all of these data structures, as the file size increases, the load, search, and display time also increases. With that, the tree or hash table would be beneficial for larger files. The tree would be especially beneficial for larger files, due to its excellent search time.

## Recommendation

For the current requirements, I recommend using a vector, due to the worst-case and average case runtimes of the three main functions utilized in this. For a larger file, arguments could be made for the tree structure, and if the data was not required to be displayed in order, the hash table could enter the conversation. Overall, the vector has the best average cases for loading the courses, searching for a course, and displaying all courses in order. While I contemplated the tree for most of this course, the vector is a simple implementation, and will provide the best overall results for this program. The best data structure can be different for each task, and a vector will be ideal in this scenario.