Amanda Nelson

CS-300

12/4/24

**Project One Pseudocode**

**Vector**

void readFileAndParse(filename) {

//open file as input

//create empty list to store the data

//while loop to read each line

//split line into course and prerequisites by using delimiter = ”,”

//if statement to check for at least two parameters

//parse each line by extracting data

//validate data

//create vector from parsed lines

//if statement for passing valididation, create item, and add to list

//close file

//ENDFOR

//return data

//ENDIF

//ENDWHILE

}

//define class for course:

//attribute courseNumber: string

//attribute name: string

//attribute prerequisite: array of strings

//create empty vector to store data from courses

//for each loop to extract data

//courseCode

//name

//prerequisite

//create new course object (code, name, prerequisite)

//add new course object to vector

//ENDFOR

}

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

//ENDFOR

//ENDFOR

}

**For a given course, print out its title and prerequisites.**

//Function printCourseinfo(Vector<Course> courses, String courseCode)

//input for courseID

//while loop

//vector is not empty

//if input is same as courseCode

//print courseCode and name

//while prerequisite = true

//print prerequisite

//ENDwhile

//ENDif

//ENDwhile

**Hash Table**

void readFileAndParse(filename) {

//open file as input

//create empty list to store the data

//while loop to read each line

//split line into course and prerequisites by using delimiter = ”,”

//if statement to check for at least two parameters

//parse each line by extracting data

//validate data

//create hash table from parsed lines

//if statement for passing valididation, create item, and add to list

//close file

//ENDFOR

//return data

//ENDIF

//ENDWHILE

}

//define class for course:

//attribute courseNumber: string

//attribute name: string

//attribute prerequisite: array of strings

//create empty hashTable to store data from courses

//for each loop to extract data

//courseCode

//name

//prerequisite

//create new course object (code, name, prerequisite)

//create hash key from courseCode % 100

//append course data into hash table

//close file

//ENDFOR

//print course data

}

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

//for all courses % 100

//calculate the index for the key

//check if key exists

//if hash table is not empty

//iterate through chain

//return if found

//return null if not found

//ENDif

//ENDFOR

}

**For a given course, print out its title and prerequisites.**

//Function printCourseinfo(hashTable<Course> courses, String courseCode)

//input for courseCode

//assign key to courseCode

//assign node to key

//if currNode = key

//return course node key

//ENDif

//if node is empty

//return null

//ENDif

//else node does not = null compare to key

//if key is a match print

//next node

//ENDif

//ENDif

**Binary Search Tree**

void readFileAndParse(filename) {

//open file as input

//create empty list to store the data

//while loop to read each line

//split line into course and prerequisites by using delimiter = ”,”

//if statement to check for at least two parameters

//parse each line by extracting data

//validate data

//if statement for passing valididation, create item, and add to list

//close file

//ENDFOR

//return data

//ENDIF

//ENDWHILE

}

//define class for course:

//attribute courseNumber: string

//attribute name: string

//attribute prerequisite: array of strings

//function createCourse

//for all courses

//courseCode

//name

//prerequisite

//create new course object (code, name, prerequisite)

//return new course

//Binary tree struct

//set attribute data, left, and right

//function to insert

//If/else statement in insert course in binary tree either left or right using nodes

//ENDIF

//return

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

//for all courses

//if the root is null

//return null

//if root equals key

//return root

//else if key is less than root

//return root left

//else if key is more than root

//return root right

//ENDif

//ENDFOR

}

**For a given course, print out its title and prerequisites.**

//Function printCourseinfo(Tree<Course> courses, String courseCode)

//input for courseCode

//assign currnode to root

//while currnode is not null

//if courseCode = currnode

//return currnode

//while prerequisite is true

//print prerequisite

//ENDwhile

//if courseCode < root

// set currnode to left

//else

//currnode is to right

//END if

//END while

**Menu**

//set choice to 0

//while loop for menu choices.

//choice not equal to 9

//output choices for user to pick

//switch case

//option 1: load file data

//option 2: print alphanumeric order of course

//option 3: print course title and prerequisites for courses

//option 9: exit

//ENDwhile

**Print in alphanumeric order**

//function printSortedCourses

//sort courses in ascending alphanumeric order

//print each course in sorted list

**Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open file as input | 1 | n | n |
| create empty list to store the data | 1 | n | n |
| while loop to read each line | 1 | n | n |
| split line into course and prerequisites by using delimiter = ”,” | 1 | n | n |
| if statement to check for at least two parameters | 1 | n | n |
| parse each line by extracting data | 1 | n | n |
| validate data | 1 | n | n |
| if statement for passing valididation, create item, and add to list | 1 | n | n |
| Close file | 1 | n | n |
| Return data | 1 | n | n |
| Create class course | 1 | n | n |
| Create vector | 1 | 1 | 1 |
| For each loop to get data | 1 | 1 | 1 |
| Create new course object (code, name, prerequisite) | 1 | n | n |
| Add new course object to vector | 1 | n | n |
| For all courses | 1 | n | n |
| If course is same courseNumber | 1 | n | n |
| Print out the course info | 2 | 1 | 1 |
| For each prerequisite of course | 1 | n | n |
| Print prerequisite course info | 1 | n | n |
| **Total Cost** | | | 20n + 3 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open file as input | 1 | n | n |
| create empty list to store the data | 1 | n | n |
| while loop to read each line | 1 | 1 | 1 |
| split line into course and prerequisites by using delimiter = ”,” | 1 | n | n |
| if statement to check for at least two parameters | 1 | n | n |
| parse each line by extracting data | 1 | n | n |
| validate data | 1 | n | n |
| if statement for passing valididation, create item, and add to list | 1 | n | n |
| Close file | 1 | n | n |
| Return data | 1 | n | n |
| Create class course | 1 | n | n |
| Create hash table | 1 | 1 | 1 |
| Create new course object (code, name, prerequisite) | 1 | n | n |
| Create hash key from courseCode | 1 | n | n |
| Append course data into hash table | 1 | n | n |
| Close file | 1 | n | n |
| Print course data | 1 | n | n |
| For all courses | 1 | n | n |
| calculate the index for the key | 1 | n | n |
| check if key exists | 1 | 1 | 1 |
| if hash table is not empty | 1 | n | n |
| iterate through chain | 1 | n | n |
| return if found | 1 | n | n |
| return null if not found | 1 | n | n |
| **Total Cost** | | | 24n + 3 |
| **Runtime** | | | O(n) |

| **Binary Search Tree** | | **Line Cost** | **# Times Executes** | **Total Cost** | |
| --- | --- | --- | --- | --- | --- |
| open file as input | | 1 | n | n | |
| create empty list to store the data | | 1 | n | n | |
| while loop to read each line | | 1 | 1 | 1 | |
| split line into course and prerequisites by using delimiter = ”,” | | 1 | n | n | |
| if statement to check for at least two parameters | | 1 | n | n | |
| parse each line by extracting data | | 1 | n | n | |
| validate data | | 1 | n | n | |
| if statement for passing valididation, create item, and add to list | | 1 | n | n | |
| Close file | | 1 | n | n | |
| Return data | | 1 | n | n | |
| define class for course | | 1 | n | n | |
| Create class course | | 1 | n | n | |
| for all courses | | 2 | n | n | |
| courseCode | | 1 | n | n | |
| name | | 1 | n | n | |
| prerequisite | | 1 | n | n | |
| create new course object | | 1 | n | n | |
| return new course | | 1 | n | n | |
| Binary tree struct | | 1 | n | n | |
| set attribute data, left, and right | | 1 | n | n | |
| function to insert | | 1 | n | n | |
| If/else statement in insert course in binary tree either left or right using nodes | | 1 | n | n | |
| return | | 1 | n | n | |
| for all courses | | 1 | n | n | |
| if the root is null | | 1 | n | n | |
| return null | | 1 | n | n | |
| if root equals key | | 1 | n | n | |
| return root | | 1 | n | n | |
| else if key is less than root | | 1 | n | n | |
| return root left | | 1 | n | n | |
| else if key is more than root | | 1 | n | n | |
| return root right | | 1 | n | n | |
| **Total Cost** | | | | 32n + 1 | |
| **Runtime** | | | | O(n) | |

**Recommendation**

After my analysis of the worst-case runtimes, all of the above options are equal with a O(n) runtime. Any of these structures would work for this application. Looking at the information about what this program will do most, it is my recommendation to use a hash table. Searching for a course and printing the courses will be done numerous times. While loading the courses, adding, and deleting a course will be done rarely. I believe by using a % 100 within the hash table this will make it easier for the user to search for a course. I will also be implementing chaining to deal with collisions.