# CS 300

# Professor Lebel

# Michael Thomas

# 16 June 2024

# CS 300 Pseudocode Document

//Pseudocode

* CREATE course objects
  + courseName and courseNumber string variables from input file data stream
  + vector string for prerequisites from input file data stream
* READ data from file {
  + OPEN file
  + IF unable to open file, return value -1
    - OUTPUT “unable to open file”
  + ELSE
    - WHILE file still has data
      * PULL line to back of file stream
        + FOR each line in file

FOR first and second parameters

Use PUSHBACK to add value to file stream

IF a third parameter exists

Use PUSHBACK to add value until another line starts with the same course number

* + CLOSE file
  + RETURN file stream line
  + }

//Vector - Milestone 1

* 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
  + }

}

//Hash Table - Milestone 2

* STORE course objects in HashTable Class
  + CREATE node with default course node and integer key
  + SET integer size to default
  + INSERT course objects into HashTable
* INT prerequisite courses
  + CREATE key with hashed course number
  + RETRIEVE node using key, set to a new node
  + WHILE node is not nullptr
    - IF node in course equals courseNumber
      * SET total prerequisites to size of prerequisite node
    - FOR each prerequisite in total perquisites
      * PRINT total number of prerequisites
    - ELSE
      * SET node pointer to next node
* Void searchCourse (HashTable<Course> courses, String courseNumber) {
  + VOID HashTable PrintAll()
  + For all courses
    - IF the course is the same as courseNumber and node key does not equal UINT\_MAX
      * PRINT out the course information for index node
        + Node key, course ID
      * FOR each prerequisite of the course
        + PRINT prerequisite course information
      * WHILE next node is not equal to nullptr
        + PRINT node key, course ID
    - RETURN
  + }

}

//Binary Search Tree – Milestone 3

* STORE course objects in Tree Class
  + CREATE a root node that points to null
  + CREATE an insert method
    - IF default root is null, then current course is root
    - ELSE IF course number is less than root ADD to the left
      * IF left is null
        + ADD course number
      * ELSE
        + IF course number is less than leaf node

ADD left

* + - * + IF course number greater than leaf node

ADD right

* + - ELSE I course number is greater than root ADD to the right
      * IF right is null
        + ADD course number
      * ELSE
        + IF course number is less than leaf node

ADD left

* + - * + IF course number is greater than leaf node

ADD right

* + INSERT course objects into Tree
* void searchCourse(Tree<Course> courses, String courseNumber) {
  + SET current node equal to root
    - WHILE current node is not equal to null pointer
      * IF current courseID and courseID are both equal to 0
        + RETURN current courseID
    - IF courseID is smaller than current node, move left
      * SET current node equal to left
    - ELSE courseID is larger than current node, move right
      * SET current node equal to right
    - RETURN course
* DEFINE inOrder Node
  + IF node is not equal to null pointer
  + CALL inOrder node from the left
  + PRINT courseID, courseTitle, and prerequisites
  + CALL inOrder node from the right

//Menu

* SET integer for user input, default it to 0
* CREATE menu WHILE loop
  + WHILE choice is not equal to 9
    - Output menu choices
      * Choice 1: Load data from Course File
      * Choice 2: Print alphanumeric list of all courses in the Computer Science Department
      * Choice 3: Print the course title and prerequisites for any individual course
      * Choice 9: Exit the program
  + CREATE switch cases based on user input choice
    - CASE 1
      * LOAD Bids
      * BREAK
    - CASE 2
      * PRINT CourseList
      * BREAK
    - CASE 3
      * PRINT Course and Prerequisites
      * BREAK
    - CASE 9
      * PRINT “Goodbye”
      * END program
      * BREAK
    - Default:
      * PRINT “Error with user input, please submit numerical choice”
      * BREAK

## Runtime Analysis

Tables:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 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) |

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **CREATE vector** | 1 | 1 | 1 |
| **FOR each line in file stream** | 1 | n | n |
| **CREATE course object** | 1 | n | n |
| **INITIALIZE string token** | 1 | n | n |
| **INITIALIZE integer count** | 1 | n | n |
| **STORE course objects in data structure** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| **Hash Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **CREATE hash table** | 1 | 1 | 1 |
| **STORE course objects in hash table** | 1 | n | n |
| **CREATE node with default course node and integer key** | 1 | n | n |
| **SET integer size to default** | 1 | n | n |
| **INSERT course objects into hash table** | 1 | n | n |
| **SET integer for prerequisite course** | 1 | n | n |
| **CREATE key with hashed course number** | 1 | n | n |
| **RETRIEVE node using key, SET to a new node** | 1 | n | n |
| **WHILE node is not null pointer** | 1 | n | n |
| **IF node in course is equal to course number** | 1 | n | n |
| **SET total prerequisites to size of prerequisite node** | 1 | n | n |
| **FOR each prerequisite in total prerequisites** | 1 | n | n |
| **PRINT total number of prerequisites** | 1 | n | n |
| **ELSE; SET node pointer to next node** | 1 | n | n |
| **searchCourse, VOID Hash Table print all** | 1 | n | n |
| **FOR all courses** | 1 | n | n |
| **IF course is same as courseNumber and node key is not max** | 1 | n | n |
| **PRINT course information for index node** | 1 | n | n |
| **FOR each prerequisite of the course** | 1 | n | n |
| **PRINT prerequisite course information** | 1 | n | n |
| **WHILE next node is not equal to null pointer** | 1 | n | n |
| **PRINT node key and course ID** | 1 | n | n |
| **Total Cost** | | | 21n + 1 |
| **Runtime** | | | O(n) |

| **Binary Search Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| CREATE **root node pointing to null** | 1 | 1 | 1 |
| CREATE insert method | 1 | n | n |
| SET left and right equal to null pointer | 1 | n | n |
| IF node is less than root node, add to the left | 1 | n | n |
| IF no left node, add node to left | 1 | n | n |
| ELSE IF node is greater than left node, add node to right | 1 | n | n |
| ELSE if node is greater than root node, add to right | 1 | n | n |
| IF no left node, add node to left | 1 | n | n |
| ELSE if node is greater than left node, add node to right | 1 | n | n |
| INSERT course objects into TREE | 1 | n | n |
| SET current node equal to root | 1 | n | n |
| WHILE current node is not null | 1 | n | n |
| IF current course ID and set course ID are both 0 | 1 | n | n |
| RETURN current course ID | 1 | n | n |
| IF course ID is smaller than current node, move left | 1 | n | n |
| SET current node equal to the left | 1 | n | n |
| ELSE course ID is larger than current node, move right | 1 | n | n |
| SET current node equal to the right | 1 | n | n |
| RETURN course |  |  |  |
| **Total Cost** | | | 17n + 1 |
| **Runtime** | | | O(log n) |

Advantages and Disadvantages:

* Vector
  + Vector data structures allow for easy implementation and fast file parsing, outpacing both Hash Tables and Tree Data Structures. However, there are serious disadvantages with Vectors, as it comes with the slowest sorting performance and higher memory consumption.
* Hash Table
  + One of the best features of Hash Tables is the direct access to items, with the power to add and remove them in constant time. Proper implementation is key, as speed can be maximized if no search collisions occur. Unfortunately, Hash Tables do have high memory consumption and items are not kept in order, so items would need to be sorted before printing each time.
* Tree Data Structure
  + The biggest challenge with Tree Data Structures is ensuring that it is properly balanced, so it does go through a lengthy process in item insertion. Other than that, this data structure is efficient and quick, able to handle most processes in exponential time relative to the amount of data read in.

Recommendation:

* For this application, I believe the best approach would be the use of a Tree Data Structure. It is specifically requested to list courses in alphabetical order, and this order can be preserved by the natural specifications of the tree structure. The course data is also not lengthy, so the tree should be able to balance relatively easily. Prerequisites can also be children to course parent leaves, allowing for a leaf and its children to be printed when requesting a course and its prerequisites. The Tree Data Structure will help quickly complete searches and grant an organized insertion process while maintaining memory and performance.