# CS 300 Pseudocode Document – John Punger 11 June 2023

**// Vector pseudocode**

// import to allow for use of a file reader

// create default vector constructor

// create vector constructor for course objects

// Set variables for vector objects:

// String course number

// String course name

// String course prerequisites

//Create structure of a “course” for vector use

Course Structure {

courseNumber

courseName

coursePreReq1

coursePreReq2

}

//Create a vector of course objects

//loadCoursesToVector

//Create a while loop to input data to vector objects (continue until no more lines)

//(verify line starts with CSCI or MATH for computer science or math class)

//intake an entire line of the file at a time, ensuring there is a comma found on the line

//create second while loop to place portions of the line, separated by commas, into vector variables

Item 1: course number

Item 2: course name

Initialize integer n = 1

While loop till line complete

If not finished next is coursePreReq(n)

Check if coursePreReq(n) matches a loaded courseNumber

Flag for potential error if no match

n++

//from case 1: generate course listing

void printCourseInformation(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**

}

//Report list of prerequisites for a course

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 number of totalPrerequisites

}

**// add course manually**

**Create course object to intake information**

**Ask for courseNumber**

**Input courseNumber**

**Ask for courseName**

**Input courseName**

**Initialize temporary variable n to 1**

**Perform loop until user states ‘0’ to exit**

**Ask for coursePreReq(n)**

**Input coursePreReq(n)**

**n++**

**return course**

**//Vector Menu**

**Start main**

**Establish entry variable**

**Start while loop till user says ‘9’**

**Output the menu to screen**

**Output option 1: Load Courses //case 1**

**Output option 2: Print all Courses //case 2**

**Output option 3: Print Single Course by courseNumber //case 3**

**Output option 4: Print Single Course by courseName //case 4**

**Output option 9: Exit //case not needed, exits automatically**

**Output “Enter choice”**

**Input choice number from user**

**Create switch case for each option**

**Case 1**

**Start clock**

**Load courses from file with loadCoursesToVector method above**

**Output number of courses read from file**

**Output time it took to read all files**

**Break**

**Case 2**

**Use for Loop based on number of courses in vector**

**Loop through and print each course alphanumerically //with printCourseInformation method**

**Break**

**Case 3**

**Ask user for courseNumber**

**Search vector by courseNumber**

**Output all course information for that course //with printCourseInformation method**

**Break**

**Case 4**

**Ask user for courseName**

**Search vector by courseName**

**Output all course information for that course //with printCourseInformation method**

**Break**

**End Main**

## Vector Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector** | 1 | 1 | 1 |
| **For each line** | 1 | n | n |
| **Increase vector size** | 1 | n | n |
| **Add item 1 as courseNumber** | 1 | n | n |
| **Add item 2 as courseName** | 1 | n | n |
| **Add each preReq** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

**Hashtable pseudocode**

**Include necessary imports for use with a file reader**

**Establish default ‘key’**

**Establish default ‘hash function’**

**Create default HashTable constructor**

**Create different levels of HashTable Constructors for course objects**

* **Utilize already designed vector pseudocode to convert vector data into node**
* **Create Node Structure to include:**
  + **Course Number**
  + **Course Name**
  + **Course Prerequisites**

**Create Vector of Nodes for course objects to be added into**

**//Task1: File Input**

**//Task2: Create Course Object(s)**

**//Task3: Print Course Info**

* **Create while loop that reads data file.**
  + **For each line of data file intake as a string**
  + **Count number of commas in 1 string**
  + **Assign part 1 (till comma) as unique course number (string)**
    - **Convert string to integer (using “stoi/atoi”)**
    - **Assign value to a key**
  + **Assign part 2 as course name**
  + **Assign part 3+ in another while loop**
    - **While still more items, name them PreReq 1, 2..n**
  + **Create node from Course Name, ‘key’, and applicable PreReqs**
  + **Add node to vector of nodes**
  + **Repeat for all input lines**
* **Once data file finished inputting:**
* **scan for error/miswritten PreReqs by comparing to ‘key’s for a match.**
* **Remove if no match**

**To print single course**

* **Search(by ‘key’)**
  + **If found in vector:**
    - **Return course(name, number, and PreReqs)**
  + **Else** 
    - **return “not found”**

**To print all courses**

* **create a for loop**
* **start with index 0, tracked by an incremental variable and continue until size of hashTable**
  + **Use if statement to see if node at index (variable) is empty**
    - **If empty, skip**
    - **If not empty, return course (name, number, and PreReqs)**
    - **//alter this one line for case 3 and 4 from main function (PreReqs)**

**To print course information, choose from cases:**

1. **File Input**
2. **Search / Print Single Course w/ PreReqs**
3. **Print all Courses w/o PreReqs**
4. **Print all Courses w/ PreReqs**

int numPrerequisiteCourses(Hashtable<Course> courses) {

}

void printSampleSchedule(Hashtable<Course> courses) {

}

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

}

**//**

**//HashTable Menu**

**Start main**

**Establish entry variable**

**Start while loop till user says ‘4’**

**Output the menu to screen**

**Output option 1: Load Courses**

**Output option 2: Print all Courses**

**Output option 3: Print Single Course by courseNumber**

**Output option 4: Print Single Course by courseName**

**Output option 9: Exit**

**Output “Enter choice”**

**Input choice number from user**

**Create switch case for each option**

**Case 1**

**Start clock**

**Load courses from file with a method above**

**Output number of courses read from file**

**Output time it took to read all files**

**Break**

**Case 2**

**Use for Loop based on number of courses in vector**

**Loop through and print each course alphanumerically**

**Break**

**Case 3**

**Ask user for courseNumber**

**Search vector by courseNumber**

**Output all course information for that course**

**Break**

**Case 4**

**Ask user for courseName**

**Search vector by courseName**

**Output all course information for that course**

**Break**

**End Main**

## HashTable Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hashtable** | 1 | 1 | 1 |
| **For each line** | 1 | n | n |
| **Create node** | 1 | n | n |
| **Add item 1 as courseNumber** | 1 | n | n |
| **Add item 2 as courseName** | 1 | n | n |
| **Add each preReq** | 1 | n | n |
| **Locate spot to place node based on courseNumber** | 1 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |

**Tree pseudocode**

//int numPrerequisiteCourses(Tree<Course> courses) {}

//

//void printSampleSchedule(Tree<Course> courses) {}

//void printCourseInformation(Tree<Course> courses, String //courseNumber) {}

**//Task1: File Input**

**//Task2: Create Course Object(s)**

**//Task3: Print Course Info**

Include imports for file reader

Establish constructor for tree structure

Initialize root to null

Establish node structure

Establish bid structure

Establish variable for node

Establish variable for parent node

Establish variable for predecessor node

Establish variable for successor node

Establish variable for current node

Establish variable for bid id

Establish variable for bid title

Establish variable for bid preReqs

Create while loop that reads data file

For each line of data intake as a string

Count number of commas in 1 string (1 line)

Convert part 1 from string to int

Create bid

Assign part 1 as a bid Id

Assign part 2 as bid title

Assign parts 3+ as bid preReq 1..n

Create a new node

Insert node into tree based on bidId

Loop until nullpointer found

If value lower than current node, move to left pointer

Else move to right pointer

Place node

//Cross reference bidId’s for error/typos/invalid ids.

Create a for loop for all bids

Compare bidIds

Ensure at least 1 match is made for all bidIds

++matchCount

Remove if matchCount = 0 after loop

**To print single course**

* **Search(by ‘bidId’)**
* **Set currentNode = root node**
* **Loop until bidId found**
  + If value lower than current node, move to left pointer
  + Else move to right pointer
* **Return course(name, number, and PreReqs)**
* **Else** 
  + **return “not found”**

**To print all courses**

**//print in order of bidId**

* **Print courses in order of left, center, right node pointer values**
* **//alter this one line for case 3 and 4 from main function (PreReqs)**
* **print course(name, number, and PreReqs)**

**To print course information, choose from cases:**

1. **File Input**
2. **Search / Print Single Course w/ PreReqs**
3. **Print all Courses w/o PreReqs**
4. **Print all Courses w/ PreReqs**

**//Tree Menu**

**Start main**

**Establish entry variable**

**Start while loop till user says ‘4’**

**Output the menu to screen**

**Output option 1: Load Courses**

**Output option 2: Print all Courses**

**Output option 3: Print Single Course by courseNumber**

**Output option 4: Print Single Course by courseName**

**Output option 9: Exit**

**Output “Enter choice”**

**Input choice number from user**

**Create switch case for each option**

**Case 1**

**Start clock**

**Load courses from file with a method above**

**Output number of courses read from file**

**Output time it took to read all files**

**Break**

**Case 2**

**Use for Loop based on number of courses in vector**

**Loop through and print each course alphanumerically**

**Break**

**Case 3**

**Ask user for courseNumber**

**Search vector by courseNumber**

**Output all course information for that course**

**Break**

**Case 4**

**Ask user for courseName**

**Search vector by courseName**

**Output all course information for that course**

**Break**

**End Main**

## Binary Tree Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Tree** | 1 | 1 | 1 |
| **For each line** | 1 | n | n |
| **Create Node** | 1 | n | n |
| **Add item 1 as courseNumber** | 1 | n | n |
| **Add item 2 as courseName** | 1 | n | n |
| **Add each preReq** | 1 | n | n |
| **Locate spot to save Node** | 1 | n/2 | n/2 |
| **Total Cost** | | | 6n/2 + 1 |
| **Runtime** | | | O(n) |

Advantages and Disadvantages:

The advantage of a vector is that they are very simple to initiate and add values to. Each new course in the vector would add another object onto the end of the vector. However, some disadvantages, if a course is desired to be added to the beginning or middle of the vector, a lot of shifting would need to occur to make room, causing excess energy. Furthermore, vector searching for unknown indexes is limited to linear searches.

The advantage of a hashtable is that it can prevent the addition of multiple courses that have the same values. Once a “bucket” for a courseNumber is filled, an error will occur if attempting to load another course into that “bucket.” However, these errors, or collisions, will need to be handled. Additionally, unless you plan to have a lot of blank “buckets”, such as have 1-999 bucket empty to begin course values at 1000-1999, you will have a lot of wasted memory. With having less memory use, as is normal, your hashtable will be sorted in an ordered fashion which makes recalling or searching for values also random. Random in this case means an expected value of average or n/2. Thus, while random, it still is better than a linear search such as vectors for unknown values or unordered data.

Binary Trees allow more efficient sorting, placing, and removing of elements, but are not as memory efficient such as a vector (which uses 1 slot for 1 object and does not need to maintain empty memory). Additionally, if a value such as a courseNumber is known, searching in a vector or hashtable would be faster than transversing through a tree.

Recommendation:

My recommendation for this scenario is to go with a Binary Tree since it can be easier placed, sorted and altered when adding or removing courses with a minimum time of O(log n) and a max time of O(n). Add or removing a course in a vector would cause lots of shifting which takes up valuable time. A binary tree can easily be sorting from the start to be in order and thus would be faster to meet the specified result of printing courses in alphanumberic order. As stated, a vector could be initiated in order, but if the data loaded in is not in order, it would take up to O(n^2) to quicksort it. A hashtable can be established in order if using excess blank memory, or otherwise would use O(n) steps to locate information from bins to sort to print.