**Pseudocode: Project One**

**Opening and reading data from file:**

OPEN file

IF file is found:

WHILE not at EOF (End of the File):

READ each line

IF line value is < 2:

RETURN ERROR

ELSE:

READ parameters

IF parameters ≥ 3:

IF third or more parameters in first parameter:

Continue READ

ELSE:

RETURN ERROR

END ELSEIF

END IF

END ELSEIF

END WHILE

ELSE:

RETURN File not found

END ELSEIF

CLOSE file

**Creating Course Objects (Vector):**

INITIALIZE Course Vector

WHILE not EOF (End of the File):

READ each line

FOR first and second value:

ADD value to Vector

END FOR

IF third value is present:

ADD value to new line

END IF

END WHILE

**Search Data Structure for Course and Print Information:**

INPUT from user

READ through Vector

IF the input is equal to courseNumber:

PRINT out course information

FOR each prerequisite to the course

PRINT prerequisite course information

ENDFOR

ENDIF

**Create Course Objects (HashTable):**

INITIALIZE Course Vector

CREATE HashTable Class

CREATE INSERT method for items to go into HashTable

READ through file

WHILE not EOF (End of the File)

READ each line

FOR first and second values:

CREATE TEMP item to store values

ENDFOR

IF a third value is present:

ADD to current value

END IF

CALL insert method

END WHILE

**Search and Print Course Information:**

INPUT from user

ASSIGN INPUT to key

SEARCH HashTable for key

IF key is found:

PRINT out course information

FOR prerequisites in courses:

PRINT out prerequisite course information

END FOR

END IF

**Create Course Objects (Tree):**

WHILE not at end of file

READ next line

FOR first and second value

ADD Course ID and Name

END FOR

IF third is present

ADD Prerequisite

IF another value is present

ADD Prerequisite

END IF

END IF

END WHILE

**Binary Tree Search Method:**

DEFINE Binary Tree Class

CREATE root

SET root = null pointer

CREATE Insert method

IF root = null pointer

SET root = current course

ELSE IF current course < root

TRAVERSE current course left

IF left = null

ADD course number

ELSE

IF course number < leaf

TRAVERSE LEFT

END IF

IF course number > leaf

TRAVERSE RIGHT

END IF

END ELSE IF

ELSE IF current course > root

TRAVERSE current course right

IF right = null

ADD course number

ELSE

IF course number < leaf

TRAVERSE LEFT

END IF

IF course number > leaf

TRAVERSE RIGHT

END IF

END ELSE IF

END ELSE IF

**Search and Print Course Information:**

INPUT from user

ASSIGN Input to bidId

SEARCH Binary Search Tree for bidId

IF bidId < node

TRAVERSE left

IF bidId < leaf

TRAVERSE left

ELSE

TRAVERSE right

END ELSE IF

ELSE

TRAVERSE right

END ELSE IF

IF bid Id = node

DISPLAY Course information

FOR prerequisites within course information

DISPLAY course prerequisites

END FOR

END IF

**Menu Pseudocode**

INITIALIZE Choice variable

SET choice = 0

DISPLAY Menu Options

1. Load Courses
2. Print Course List
3. Print Course
4. Exit

WHILE choice != 4

CREATE Switch statement with choice as parameter

CASE 1:

Load courses into data structure

Break;

CASE 2:

Print Entire Course List

Break;

CASE 3:

Print Course Information

Break;

CASE 4: Exit Program

Exit;

END SWTICH

END WHILE

**Print Alphanumerically**

**Vector:**

SORT data with selection sort with courses as variable

DISPLAY newly sorted courses

**Hash-Table:**

SORT data of bucket values

FOR each bucket

ADD values to vector

END FOR

SORT new vector

PRINT sorted vector

**Tree:**

SORT binary tree with inOrder Function

PRINT inOrder Left

PRINT current node

PRINT inorder Right

**Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector** | 1 | 1 | 1 |
| **FOR each next line in file** | 1 | n | n |
| **print out the course information** | 1 | n | n |
| **FOR each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| **HashTable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create HashTable Class | 1 | 1 | 1 |
| Create Insert Method | n/a | n/a | n/a |
| Create Hash Key for courses | 1 | n | n |
| IF no entry for key is found | 1 | n | n |
| Assign key to bucket | 1 | n | n |
| ELSE | 1 | n | n |
| Find next open bucket | 1 | n | n |
| Assign key to bucket | 1 | n | n |
| FOR each next line in file | 1 | n | n |
| for all courses | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the course information | 1 | n | n |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 1 | n | n |
| **Total Cost** | | | 12n + 1 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Add node method** | n/a | n/a | n/a |
| **CREATE Root equal to zero** | 1 | 1 | 1 |
| **FOR next node** | 1 | n | n |
| **IF node is less than root** | 1 | n | n |
| **TRAVERSE node left** | 1 | n | n |
| **IF node is null** | 1 | n | n |
| **Create and assign node** | 1 | 1 | 1 |
| **IF node is greater than root** | 1 | n | n |
| **TRAVERSE node right** | 1 | n | n |
| **IF node is null** | 1 | n | n |
| **Create and assign node** | 1 | 1 | 1 |
| **FOR all courses** | 1 | n | n |
| **IF the course is the same as courseNumber** | 1 | n | n |
| **Print out the course information** | 1 | 1 | 1 |
| **FOR each prerequisite of the course** | 1 | n | n |
| **Print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 11n + 4 |
| **Runtime** | | | O(n) |

**Advantages/Disadvantages**

**Vector:**

Vectors are a singular dimensional array that store data elements within them and are able to have it’s size adjusted higher or lower than what it needs to be. When scanning through a vector for information it will scan every element until the desired element is found, which can be sort of a disadvantage depending on the size of the information you’re attempting to sort through. Even though that would seem like a not so efficient way to sort and read through data, we can see that it has a runtime of this method is 4n+1 according to our runtime chart.

**Hash-Table:**

Hash-tables are a style of data management that involves storing data with a key and bucket system. Each set of data in the structure is given a value and then stored into its respective slot, or “bucket” as mentioned earlier. In order to find this data, the proper key must be used since each bucket has a unique key allocated to that set of data. Although this sounds like a very appealing method to use, it wouldn’t be the most effective since it doesn’t store the data in an ordered fashion. This would mean that in order to print out every course in alphanumeric order, each course would have to be removed, then sorted, then printed. With a cost of 12n+1, I would have to say that this is not the most effective method for the program.

**Tree:**

A binary search tree is a node based structure in which the top node (or the root) is allowed to have a maximum of two children. This rule also applies to the other children nodes throughout the tree. Nodes that are less than the root node will be stored to the left and nodes that are greater in value will be stored to the right. Although this search method normally has incredible runtime with each of the searching, inserting, and deleting of elements, one of the downsides that comes with this is the runtime is also impacted by the height of the tree. This method also has the largest cost time with it being estimated to be 11n+4, making it another reason to not be our go-to method for this project.

**Evaluation:**

After having looked three data structures, I believe that using a vector would be the best data structure for this job. My reasoning being is that I feel that making a sorted list for the courses would not only be easier for printing the data alphanumerically, but it would also be easier to print the information in the necessary order for the courses to be completed in. In other words, it would be easier to print out a schedule while doing the data as a vector that’s in a sorted list as opposed to a perfect tree or utilized hash-table. Another great aspect to using vectors is that the data can easily be added or removed as needed. The runtime analysis (which I feel I’ve done incorrectly) for the vector is of the least value when compared to the others, which is another reason I believe a vector would be the best choice. However, I think I would like to work with a BTS simply to improve my programming skills and fully understand the data structure.