Project One – Pseudocode

//Open File, read data, parse each line, check for format error

FUNCTION loadCourses

OPEN “CourseInformation.txt” file for reading

IF file not found

PRINT “File not found”

ELSE

CREATE an empty LIST to store course objects

CREATE a new VECTOR for courses

CREATE a new HASH TABLE for courses

CREATE a new TREE for courses

WHILE NOT end of file

READ line from file

IF line is EMPTY

PRINT “No data found”

ELSE

SPLIT line by comma parameter

IF less than 2 parameters

PRINT “Format error: Must have at least two parameters.”

ELSE

SET first parameter to courseNumber

SET second parameter to courseName

CREATE an empty LIST for prerequisites

FOR each prerequisite

IF NOT a courseNumber

PRINT “Format error: Prerequisite not found.”

ELSE

ADD prerequisite to the prerequisite list

CREATE courseObject

SET courseObject.courseNumber = courseNumber

SET courseObject.courseName = courseName

SET courseObject.coursePrerequistes = prerequisites

//Add to Vector

ADD courseObject to courses VECTOR

//Add to Hash Table

INSERT courseObject into courses HASH TABLE using courseNumber as key

//Add to Tree

INSERT courseObject into courses TREE based on courseNumber

CLOSE “CourseInformation.txt” file

PRINT “Data loaded successfully.”

//Create course object and store

FUNCTION courseObject (courseNumber, name, prerequistes)

CREATE newCourse

SET newCourse.courseNumber = courseNumber

SET newCourse.courseName = name

SET newCourse.coursePrerequistes = prerequisites

RETURN newCourse

//Search data for specific course, print out course information and prerequisites

FUNCTION searchCourse (courses, courseNumber)

IF courseNumber exists in courses HASH TABLE

courseObject = GET courseObject from courses HASH TABLE

PRINT “Course Number: “ + courseObject.courseNumber

PRINT “Course Name: “ + courseObject.courseName

PRINT “Prerequisites: “

FOR each prerequisite in courseObject.coursePrerequisites

PRINT prerequisite

ELSE

PRINT “Course not found.”

//Sort course information by alphanumeric

FUNCTION printAlphanumericList (courses, STRUCTURE)

//FIXME – depends on chosen data structure

IF structure is VECTOR

//Sort course objects in VECTOR by courseNumber

SORT courses VECTOR by courseObject.courseNumber

FOR each courseObject in courses VECTOR

PRINT courseObject.courseNumber + “ | “ + courseObject.courseName

IF structure is HASH TABLE

//Traverse HASH TABLE in ascending order of key

Keys = SORT keys in ascending order

FOR each key in keys

courseObject = GET courseObject from courses HASH TABLE using key

PRINT courseObject.courseNumber + “ | “ + courseObject.courseName

IF structure is TREE

//Perform an in-order traversal to print objects in TREE

FUNCTION inOrder(node)

IF node is not NULL

inOrder(node left)

PRINT node.courseNumber + “ | “ + node.courseName

inOrder(node right)

//Menu

SET choice = 0

WHILE choice is not 4

PRINT “Menu:”

PRINT “1. Load Data”

PRINT “2. Print Course List”

PRINT “3. Print Course and Prerequisites”

PRINT “4. Exit”

INPUT choice

SWITCH choice

CASE 1: //Load Data

loadCourses

CASE 2 – //Print Alphanumeric Course List

//FIXME – depends on chosen data structure

printAlphanumericList(courses, VECTOR)

printAlphanumericList(courses, HASH TABLE)

printAlphanumericList(courses, TREE)

CASE 3 – //Print course title and prerequisites

PRINT “Enter course number:”

INPUT courseKey

searchCourse(courses, courseKey)

CASE 4 – //End program

PRINT “Good bye.”

SET choice = 4

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create vector | 1 | 1 | 1 |
| ADD courseObject to courses VECTOR | 1 | N | N |
| FUNCTION printAlphanumericList (courses, VECTOR) | 1 | 1 | 1 |
| SORT courses VECTOR by courseObject.courseNumber | 1 | N | N |
| FOR each courseObject in courses VECTOR | 1 | N | N |
| PRINT courseObject.courseNumber + “ | “ + courseObject.courseName | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 3 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create hash table | 1 | 1 | 1 |
| INSERT courseObject into courses HASH TABLE using courseNumber as key | 1 | N | N |
| FUNCTION printAlphanumericList (courses, HASH TABLE) | 1 | 1 | 1 |
| Keys = SORT keys in ascending order | 1 | N | N |
| FOR each key in keys | 1 | N | N |
| courseObject = GET courseObject from courses HASH TABLE using key | 1 | 1 | 1 |
| PRINT courseObject.courseNumber + “ | “ + courseObject.courseName | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 4 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create tree | 1 | 1 | 1 |
| INSERT courseObject into courses TREE based on courseNumber | 1 | N | N |
| FUNCTION printAlphanumericList (courses, TREE) | 1 | 1 | 1 |
| FUNCTION inOrder(node) | 1 | N | N |
| IF node is not NULL | 1 | 1 | 1 |
| inOrder(node left) | 1 | N | N |
| PRINT node.courseNumber + “ | “ + node.courseName | 1 | 1 | 1 |
| inOrder(node right) | 1 | N | N |
| **Total Cost** | | | 4n + 4 |
| **Runtime** | | | O(n) |

//Vector

**Advantage**: Simple and memory efficient that also provides random access to its elements.

**Disadvantage**: Inefficient for inserting or removing elements in the middle which makes sorting slower compared to other structures.

//Hash Table

**Advantage**: Efficient for inserting and retrieving data with fast search times.

**Disadvantage**: Data is not sorted by default and needs to be sorted by keys separately.

//Tree

**Advantage**: Maintains sorted order and easy to retrieve data.

**Disadvantage**: Insertion and deletion can be slower if the tree is unbalanced.

//Recommendation

For this project, a Hash Table or Tree would be best suited to accommodate the needs of ABCU. A Tree would maintain a sorted order for printing out a full list organized alphanumerically. However, a Hash Table is more efficient for adding new courses and prerequisites while also being able to search quickly. For this project, I am going to recommend the use of a Hash Table for the code to prioritize search times and adaptability to any changes to Computer Science courses.