ABCU Final

SNHU

Jazzlynn Gonzalez

August 4, 2025

Vector Pseudocode

Function LoadCourses(filename):

Open file with name ‘filename’

If file cannot be opened:

Print “Error: Could not open file.”

Return

For each line in the file:

Split line by comma into tokens

If number of tokens < 2:

Print “Error: Invalid line format – less than 2 items.”

Continue to next line

Set courseNum = tokens[0]

Set courseTitle = tokens[1]

Create new Course object newCourse

Set newCourse.courseNumber = courseNum

Set newCourse.courseTitle = courseTitle

For each token from index 2 to end:

Add token to newCourse.prerequisites

Add courseNum to allCourseNumbers

Add newCourse to courseList

Function ValidatePrerequisties():

For each course in courseList:

For each prereq in course.prerequisties:

If prereq not in allCourseNumbers:

Print “Error: Prerequisite ‘” + prereq + ‘”for course” + course.courseNumber + “not found.”

Function FindAndPrintCourse(courseID):

Set found = false

For each course in courseList:

If course.courseNumber == courseID:

Print “Course Number: “ + course.courseNumeber

Print “Course Title: “ + course.courseTitle

If course.prerequisites is empty:

Print “Prerequisites: None”

Else:

Print “Prerequisites:”

For each prereq in course.prerequisites:

Print “ – “ + prereq

Set found = true

Break

If found is false:

Print “Course not found.”

Hash Table Pseudocode

Function loadCoursesFromFile(filename):

Create empty HashTable<Course> courseTable

Create list validCourseNumbers

Open file with name = filename

While file has next line:

Read and split line by comma into tokens

If number of tokens < 2:

Print “Invalid line format: “ + line

Continue

Add tokens[0] to validCourseNumbers

Rewind file to beginning

While file had next line:

Read and split line by comma into tokens

Set courseNumber = tokens[0]

Set name = tokens[1]

Create empty list prerequisites

For each token from index 2 to end:

If token not in validCourseNumbers:

Print “Invalid prerequisites: “ + token

Add token to prerequisites

Create new Course newCourse

Set newCourse.courseNumber = courseNumber

Set newCourse.name = name

Set newCourse.prerequisites = prerequisites

Insert newCourse into courseTable with key = courseNumber

Function searchCourse(courseTable, courseNumber)

Set course = courseTable.get(courseNumber)

If course is null:

Print “Course not found: “ + courseNumber

Return

Print course.courseNumber + “,” + course.name

If course.prerequisites is not empty:

Print “Prerequisties:”

For each prereq in course.prerequisites:

Set prereqCourse = courseTable.get(prereq)

If prereqCourse is not null:

Print prereqCourse.courseNumber + “,” + prereqCourse.name

Else:

Print prereq + “ (Course info not found)”

Binary Search Tree Pseudocode

Function loadCourses(filename):

Open the file

If file cannot be opened:

Print “Error: Could no open file.”

Return

For each line in the file:

Split line by comma into tokens

If number of tokens < 2:

Print “Error: Invalid line format.”

Continue

Create Course newCourse with tokens[0] and tokens[1]

For each token from index 2 to end:

Add token to newCourse.prerequisties

Add courseNumber to allCourseNumbers

Add newCourse to tempCourseList

For each course in tempCourseList:

For each prereq in course.prerequisites:

If prereq not in allCourseNumbers:

Print “Error: Prerequisite “ + prereq + “ not found.”

rootNode = insertCourse(rootNode, course)

Function insertCourse(treeNode, course):

If treeNode is null:

Create new node with course

Return

If course.courseNumber < treeNode.course.courseNumber:

treeNode.left = insertCourse(treeNode.left, course)

Else:

treeNode.right = insertCourse(treeNode.right, course)

Return treeNode

Function findCourse(treeNode, Course):

If treeNode is null:

Create new node with course

Return new node

If course.courseNumber < treeNode.course.courseNumber:

treeNode.left = insertCourse(treeNode.left, course)

Else:

treeNode.right = insertCourse(treeNode.right, course)

Return treeNode

Function searchCourse(rootNode, courseNumber):

Set result = findCourse(rootNode, courseNumber)

If result is not null:

Print “Course Number: “ + result.courseNumber

Print “Course Title: “ + result.courseTitle

If result.prerequisites is empty:

Print “Prerequisites: None:

Else:

Print “Prerequisites:”

For each prereq in result.prerequisites:

Print “-“ + prereq

Else:

Print “Course not found.”

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Line/Action | Cost per Line | # of Executions | Total Cost |
| Open file | 1 | 1 | 1 |
| Read each line | 1 | N | N |
| Split and parse line | 1 | N | N |
| Create Course Object | 1 | N | N |
| Store in data structure | 1 | N | N |
| Validate prerequisites (nested loop) | 1 | N x m | N2 |

Total worst case runtime: O(n2)

This is because the nested loop checks each prerequisite for each course.

Evaluation

The vector is a simple data structure that works like a list. It is easy to understand, use, and adding a new course to the end is very fast. It also stores data close together in memory which makes it efficient. However, finding a specific course can be slow because the program has to look through each course one at a time. The courses are not stored in order, so they need to be printed in order, the while list has to be sorted.

The hash table is very fast for searching, adding, and deleting courses. It can find a course almost instantly in most cases. Which could be good for quick lookups. However, it does not keep the courses in any order, so the data must be taken out and sorted before printing. Hash tables also use more memory than a vector and if too many courses end up in the same memory spot, searching can be slow.

The binary search tree is a good option because it keeps the courses in order as they are added. This makes it easy to print the list in order without having to sort it later. When the tree is balanced, searching and adding courses is fast. It usually uses less memory than a hash table, but more than a vector. Unfortunately, if the tree becomes unbalanced, searching and adding courses can be slow. A tree is also more complicated to build compared to a vector or a hash table.