Project Pseudocode

# Vector Implementation

FUNCTION loadCourses(Vector<Course> courses, String filename)  
 OPEN file with filename  
 IF file not opened:  
 PRINT "Error: Unable to open file"  
 RETURN  
 FOR each line in file:  
 SPLIT line by ',' INTO parts  
 IF parts.LENGTH < 2:  
 PRINT "Error: Invalid format: " + line  
 CONTINUE  
 SET courseNumber = parts[0]  
 SET courseTitle = parts[1]  
 SET prerequisites = EMPTY LIST  
 FOR i FROM 2 TO parts.LENGTH - 1:  
 ADD parts[i] TO prerequisites  
 CREATE course WITH courseNumber, courseTitle, prerequisites  
 ADD course TO courses  
 CLOSE file  
  
FUNCTION searchCourse(Vector<Course> courses, String courseNumber)  
 FOR each course IN courses:  
 IF course.courseNumber == courseNumber:  
 PRINT course.courseNumber  
 PRINT course.courseTitle  
 IF course.prerequisites IS EMPTY:  
 PRINT "Prerequisites: None"  
 ELSE:  
 FOR prereq IN course.prerequisites:  
 PRINT "Prerequisite: " + prereq  
 RETURN  
 PRINT "Course not found"  
  
FUNCTION printAllCourses(Vector<Course> courses)  
 SORT courses BY course.courseNumber  
 FOR each course IN courses:  
 PRINT course.courseNumber + ": " + course.courseTitle

# Hash Table Implementation

FUNCTION readCoursesFromFile(String fileName)  
 DECLARE hashTable AS HashTable<String, Course>  
 OPEN file with fileName  
 IF file not opened:  
 PRINT "Error: Unable to open file"  
 RETURN  
 WHILE NOT end of file:  
 READ line  
 IF line IS EMPTY:  
 CONTINUE  
 SPLIT line BY ',' INTO tokens  
 IF tokens.LENGTH < 2:  
 PRINT "Error: Invalid line format: " + line  
 CONTINUE  
 SET courseNumber = tokens[0]  
 SET courseTitle = tokens[1]  
 SET prerequisites = EMPTY LIST  
 FOR i FROM 2 TO tokens.LENGTH - 1:  
 ADD tokens[i] TO prerequisites  
 CREATE course WITH courseNumber, courseTitle, prerequisites  
 hashTable[courseNumber] = course  
 CLOSE file  
  
FUNCTION printCourseInformation(String courseNumber)  
 IF courseNumber NOT IN hashTable:  
 PRINT "Course not found: " + courseNumber  
 RETURN  
 course = hashTable[courseNumber]  
 PRINT "Course Number: " + course.courseNumber  
 PRINT "Course Title: " + course.courseTitle  
 IF course.prerequisites IS EMPTY:  
 PRINT "Prerequisites: None"  
 ELSE:  
 FOR prereq IN course.prerequisites:  
 PRINT "- " + prereq  
  
FUNCTION printAllCourses(HashTable<String, Course> hashTable)  
 keys = hashTable.KEYS()  
 SORT keys  
 FOR key IN keys:  
 PRINT key + ": " + hashTable[key].courseTitle

# Binary Search Tree Implementation

FUNCTION loadCoursesToBST(String fileName)  
 DECLARE bstRoot AS TreeNode = NULL  
 OPEN file  
 IF file not opened:  
 PRINT "Error: File not found"  
 RETURN  
 WHILE NOT end of file:  
 READ line  
 SPLIT line INTO tokens BY ','  
 IF tokens.LENGTH < 2:  
 PRINT "Error: Invalid format in line: " + line  
 CONTINUE  
 courseNumber = tokens[0]  
 courseTitle = tokens[1]  
 prerequisites = EMPTY LIST  
 FOR i FROM 2 TO tokens.LENGTH - 1:  
 ADD tokens[i] TO prerequisites  
 CREATE course WITH courseNumber, courseTitle, prerequisites  
 bstRoot = insertCourse(bstRoot, course)  
 CLOSE file  
  
FUNCTION insertCourse(TreeNode node, Course course)  
 IF node IS NULL:  
 RETURN NEW TreeNode(course)  
 IF course.courseNumber < node.course.courseNumber:  
 node.left = insertCourse(node.left, course)  
 ELSE:  
 node.right = insertCourse(node.right, course)  
 RETURN node  
  
FUNCTION searchCourse(TreeNode node, String courseNumber)  
 IF node IS NULL:  
 PRINT "Course not found"  
 RETURN  
 IF courseNumber == node.course.courseNumber:  
 PRINT node.course.courseNumber  
 PRINT node.course.courseTitle  
 IF node.course.prerequisites IS EMPTY:  
 PRINT "Prerequisites: None"  
 ELSE:  
 FOR prereq IN node.course.prerequisites:  
 PRINT "Prerequisite: " + prereq  
 RETURN  
 IF courseNumber < node.course.courseNumber:  
 CALL searchCourse(node.left, courseNumber)  
 ELSE:  
 CALL searchCourse(node.right, courseNumber)  
  
FUNCTION inOrderPrint(TreeNode node)  
 IF node IS NOT NULL:  
 CALL inOrderPrint(node.left)  
 PRINT node.course.courseNumber + ": " + node.course.courseTitle  
 CALL inOrderPrint(node.right)

# Menu Logic

REPEAT  
 PRINT "1. Load Data"  
 PRINT "2. Print Course List"  
 PRINT "3. Print Course Details"  
 PRINT "9. Exit"  
 INPUT choice  
 IF choice == 1:  
 courses = loadCourses("courses.txt")  
 ELSE IF choice == 2:  
 printAllCourses(courses)  
 ELSE IF choice == 3:  
 INPUT courseNumber  
 searchCourse(courses, courseNumber)  
 ELSE IF choice == 9:  
 EXIT  
 ELSE:  
 PRINT "Invalid option"  
UNTIL FALSE

Vector

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | #Times Executes | Total Cost |
| For all courses | 1 | n | n |
| If course is the same as courseNumber | 1 | n | n |
| For each prerequisite of the course | 1 | 1(once) | 1 |
| Print each prerequisite | 1 | p (prereqs) | p |
| Total Cost |  |  | 2n/p/1 |
| Runtime |  |  | O(n) |

Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | #Times Executes | Total Cost |
| Get course # | 1 | 1 | 1 |
| If course is null | 1 | 1 | 1 |
| For each prerequisite of the course | 1 | 1 | 1 |
| Print each prerequisite | 1 | p | p |
| Total Cost |  |  | p/3 |
| Runtime |  |  | O(1/p)(≈O(1)) |

Binary Search Tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | #Times Executes | Total Cost |
| Compare courseNumber to node | 1 | Log n | Log n |
| For each prerequisite | 1 | 1 | 1 |
| Print each prerequisite | 1 | p | p |
| Total Cost |  |  | Log n/p/1 |
| Runtime |  |  | O(log n/p) |

The ideal choice is a hash table due to faster lookup when searching by course number. Memory usage is moderate but reasonable since keys can be sorted in alphanumeric order. This is the best of the three options since vectors would be inefficient with a large number of courses. Search trees aren’t ideal due to lower search time and complex logic.