CS 300 Project One: Pseudocode and Runtime Analysis

Cassidy Reinitz   
Southern New Hampshire University

# Pseudocode - Vector

// Option 1: Load course data from file into vector  
open file "courses.txt"  
for each line in file:  
 split line by commas  
 if number of tokens < 2:  
 print "Invalid line format"  
 continue  
 create new Course object  
 set courseNumber = token[0]  
 set courseTitle = token[1]  
 set prerequisites = token[2...n]  
 add Course to vector  
  
// Option 2: Print all courses in alphanumeric order  
sort vector by courseNumber  
for each course in vector:  
 print courseNumber and courseTitle  
  
// Option 3: Print course info and prerequisites  
input: courseNumber  
for each course in vector:  
 if courseNumber == input:  
 print courseNumber and title  
 print prerequisites  
  
// Option 9: Exit  
exit program

# Pseudocode - Hash Table

// Option 1: Load course data into hash table  
open file "courses.txt"  
for each line in file:  
 split line by commas  
 if number of tokens < 2:  
 print "Invalid line format"  
 continue  
 create Course object  
 insert into hashTable with key = courseNumber  
  
// Option 2: Print all courses (requires collecting keys)  
create tempList = all keys from hashTable  
sort tempList  
for each courseNumber in tempList:  
 retrieve course and print courseNumber and title  
  
// Option 3: Print course info and prerequisites  
input: courseNumber  
if courseNumber in hashTable:  
 retrieve course  
 print course info  
 print prerequisites  
  
// Option 9: Exit  
exit program

# Pseudocode - Binary Search Tree

// Option 1: Load course data into BST  
open file "courses.txt"  
for each line in file:  
 split line by commas  
 if number of tokens < 2:  
 print "Invalid line format"  
 continue  
 create Course object  
 insert Course into BST using courseNumber as key  
  
// Option 2: Print all courses (in order traversal)  
perform inOrderTraversal(BST root)  
 if root is null: return  
 inOrderTraversal(root.left)  
 print courseNumber and title  
 inOrderTraversal(root.right)  
  
// Option 3: Print course info and prerequisites  
input: courseNumber  
search BST for courseNumber  
if found: print course info and prerequisites  
  
// Option 9: Exit  
exit program

# Runtime Analysis

Assume n = number of courses in the input file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Structure | Code Line | Line Cost | # Times Executed | Total Cost |
| Vector | for each line | 1 | n | n |
| Vector | search and insert | 1 | n | n |
| Hash Table | insert (average) | 1 | n | n |
| Tree | insert (avg) | log n | n | n log n |
| Tree | search | log n | 1 | log n |

# Advantages and Disadvantages

Vector:  
+ Easy to implement and iterate through  
+ Simple sorting using built-in functions  
- Inefficient search for specific courses (O(n))  
  
Hash Table:  
+ Very fast lookups (average O(1))  
+ Great for finding single courses by courseNumber  
- Not sorted; must extract and sort keys for ordering  
  
Binary Search Tree:  
+ Maintains sorted order naturally (O(log n) insert/search)  
+ Efficient in-order traversal for sorted print  
- Can become unbalanced (worst case O(n))

# Recommendation

I recommend using the Binary Search Tree for this project. It offers the best balance of efficient searching and natural sorted order for course listing. Although hash tables are faster for lookup, they require extra work to sort. Vectors are easiest to use but least efficient for searching. Given the requirement to list courses alphanumerically and support lookups, the BST is optimal.