Pseudocode & Runtime Analysis

Name: Ivan Gonzalez

Purpose: Load course data and enable advisors to browse/query using Vector, Hash Table, and Binary Search Tree.

# 1) Shared Course Model

course := struct { number:string; title:string; prereqs:list<string> }

# 2) Vector Approach

Load + Validate File

open file at path  
if not open -> print error, return  
V := empty vector<course>; LINE := 0  
for rawLine in file:  
 LINE++  
 tokens := split rawLine by ','  
 if size(tokens) < 2 -> warn 'format error at LINE'; continue  
 c := course(); c.number := trim(tokens[0]); c.title := trim(tokens[1]); c.prereqs := []  
 for i from 2 to size(tokens)-1: c.prereqs.push\_back( trim(tokens[i]) )  
 V.push\_back(c)  
ALL := set of { c.number for c in V }  
for c in V: for p in c.prereqs: if p not in ALL -> warn missing p for c.number  
return V

Find + Print One Course

for c in V:  
 if c.number == query: print c.number+', '+c.title; print 'Prereqs: '+(join(c.prereqs, ', ') or 'none'); return  
print 'Course not found'

Print All (Alphanumeric)

W := copy(V); sort W by c.number asc  
for c in W: print c.number+', '+c.title

# 3) Hash Table Approach

Load + Validate File

open file; if fail -> return  
H := new HashTable(); order := []  
for line in file:  
 tokens := split line by ','; if size(tokens) < 2 -> warn; continue  
 c := course(tokens[0], tokens[1], tokens[2..]); H.insert(c.number, c); order.push\_back(c.number)  
for k in order: for p in H.get(k).prereqs: if not H.contains(p) -> warn missing p for k  
return H

Find + Print One Course

if H.contains(query): c := H.get(query); print c.number+', '+c.title; print 'Prereqs: '+(join(c.prereqs, ', ') or 'none')  
else: print 'Course not found'

Print All (Alphanumeric)

keys := H.keys(); sort keys asc  
for k in keys: c := H.get(k); print c.number+', '+c.title

# 4) Binary Search Tree Approach

Load + Validate File

open file; if fail -> return  
T := new BST(); inserted := []  
for line in file:  
 tokens := split line by ','; if size(tokens) < 2 -> warn; continue  
 c := course(tokens[0], tokens[1], tokens[2..]); T.insert(c.number, c); inserted.push\_back(c.number)  
for k in inserted: for p in T.find(k).prereqs: if T.find(p) is null -> warn missing p for k  
return T

Find + Print One Course

c := T.find(query)  
if c != null: print c.number+', '+c.title; print 'Prereqs: '+(join(c.prereqs, ', ') or 'none')  
else: print 'Course not found'

Print All (Alphanumeric)

inOrder(node):  
 if node == null -> return  
 inOrder(node.left); print node.course.number+', '+node.course.title; inOrder(node.right)  
call inOrder(T.root)

# 5) Menu (All Approaches)

repeat:  
 print menu (1=Load, 2=Print list, 3=Print course, 9=Exit)  
 read choice  
 if choice==1: DS := loadFromFile(path) # DS is V, H, or T  
 else if choice==2: if empty(DS) print 'Load first' else printAllSorted(DS)  
 else if choice==3: read code; findAndPrint(DS, code)  
until choice==9

# 6) Runtime Analysis (Worst-Case Big-O)

Assume n courses, p prerequisite tokens (often p = O(n)).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Structure | Load/Parse | Insert per Item | Prereq Check | Total Worst-Case |
| Vector | O(n) | O(1) amortized | O(n)+O(p) | O(n) |
| Hash Table | O(n) | Avg O(1), worst O(n) | O(p) avg | Avg O(n), worst O(n^2) |
| BST (unbalanced) | O(n) | O(h) ≤ O(n) | O(p·h) | Worst O(n^2); avg (balanced) ≈ O(n log n) |

Recommendation: Use a Hash Table for fast average-case lookups. For a sorted list (Option 2), sort the keys on demand (O(n log n)) or maintain a parallel BST/sorted vector after load.