CS-300 Module 6 Pseudocode by Justin Leger

|  |  |  |  |
| --- | --- | --- | --- |
|  | Use Cases | To Implement | Insert/Delete |
| Binary Tree | Searching, Sorting | Medium complexity | O(logN) |
| Hash Table | Fast lookups | High complexity | O(1) |
| Vector | Random access | Low complexity | O(1) |

Vectors are best for frequently appending elements and fast access by index. Binary trees are best when you need to maintain a sorted order and perform frequent insertions and deletions. Hash tables are best for fast lookups, insertions, and deletions.

Due to hash tables efficiency at performing insertions, deletions, and lookups I’d recommend hash table, despite the complexity to implement.

PSEUDOCODE:

Begin program.

Include libraries and headers

Global definitions visible to all classes and methods

Structure to hold Courses{

courseID

courseName

preReqList

}

Set up constructors

Set up destructors

Main function is menu loop:

Load CSV file path

While choice is not 9 for “Exit”, display:

Menu:

1. Load Courses
2. Display All
3. Display One

9. Exit program

Get choice using switch structure

If choice is 1, load all courses from CSV file into each data structure

If choice is 2, display all courses

If choice is 3, display one specified course

If choice is 9, exit program.

int vectorPartition(vector<Course>& courses, int begin, int end) {

set low and high equal to begin and end

calculate middle element as middlePoint

while not done

increment low while it’s less than pivot

decrement high while it’s less than pivot

where low meets high

else

swap low and high bids

move end points closer

return high

}

//sort from lowest to highest

void vectorQuickSort(vector<Course>& courses, int begin, int end) {

set mid equal to 0

if there are 1 or zero bids to sort, partition already sorted then return

partition bids into low and high

recursively sort low partition

recursively sort high partition

}

vector<Bid> loadCourses(string csvPath) {

define vector structure to hold courses

initialize CSV parser using the given path

try

loop to read rows of CSV file

create data structure and add to collection of Courses

push this course to the end

catch error

return courses

}

Void vectorPrint(vector<Course>& courses){

Loop and display the courses to read

}

void printTree(Node\* node) {

if node is not equal to nullptr

InOrder left

Output courseID, courseName, preReqList

InOrder right

}

Void printTable(){

For node begin to end iterate

If key not equal to UINT\_MAX

Output key, courseID, courseName, preReqList

Node is equal to next iter

While node not equal to nullptr

Output key, courseID, courseName, preReqList

Node is equal to next node

}

//print one course

void treePrintOne(string courseId){

Set current node equal to root

Keep looping downwards until bottom reached or matching courseId found

If match found, print current course

If course is smaller than current node then traverse left

Else larger so traverse right

}