# Hunter Brashears

11/17/2024

# CS 300 Pseudocode Document

**//Vector - Milestone 1**

Struct Course {

String courseNumber

String name

Vector<String> prerequisites

}

vector<Course> loadCourses(String filename) {

Create empty vector for course objects

Open file

If file can’t be opened

Print error message

Return

for each line in file

separate line into segments by comma

if segment is less than 2

Print error message

create course object

course.courseNumber is equal to first segment

course.courseName is equal to second segment

for each segment after 2

add the segment to course.prerequisites

add course object to the vector

for each course in the vector

for each prerequisite in course.prerequisites

if prerequisite does not equal any course

print error message

return courses

}

void searchCourse(Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

void printSortedCourses(vector<Course> courses) {

Sort courses by courseNumber(begin, end)

For each course in courses

Print course.courseNumber + course name

}

void main() {

While true

Print “Menu:”

Print “1. Load courses”

Print “2. Print sorted courses”

Print “3. Print course details”

Print “4. Exit”

Get user input as choice

If choice == 1

Courses = loadCourses(“courses.txt”)

Else if choice == 2

printSortedCourses(courses)

Else if choice == 3

print “Enter course number:”

get courseId

searchCourse(courses, courseId)

Else if choice == 9

break

else

print “Invalid selection”

}

**//Hash Table - Milestone 2**

struct Course {

String courseNumber

String name

Vector<String> prerequisites

}

hashTable<string, Course> loadcourses(String filename) {

create empty hash table for courses

Create and open file

If file can’t be opened

Print error message

Return empty hash table

for each line in file

separate line into segments by comma

if segment is less than 2

Print error message

create course object

course.courseNumber is equal to first segment

course.courseName is equal to second segment

for each segment after 2

add the segment to course.prerequisites

add course to hash table

for each course in the hash table

for each prerequisite in course.prerequisites

if prerequisite does not equal any course

print error message

return hash table

}

void searchCourse(HashTable<Course> courses, String courseNumber) {

if courseNumber is in the hash table

get course from the table

print the course information

for each prerequisite

print the prerequisite information

}

void printSortedCourses(HashTable<String, Course> courses) {

create empty vector<Course> sortedCourses

for each course in courses

add course to sortedCourses

sort() sortedCourses by courseNumber

for each course in sortedCourses

print courseNumber and course name

}

Void main() {

HashTable<String, Course> courses

While true

Print “Menu:”

Print “1. Load courses”

Print “2. Print sorted courses”

Print “3. Print course details”

Print “4. Exit”

Get user input as choice

If choice == 1

Courses = loadCourses(“courses.txt”)

Else if choice == 2

printSortedCourses(courses)

Else if choice == 3

print “enter course number:”

get courseId

searchCourse(courses, courseId)

Else if choice == 9

break

else

print “Invalid selection”

}

**//Binary Search Tree – Milestone 3**

struct Course {

String courseNumber

String name

Vector<String> prerequisites

}

struct Node {

Course course

Node left

Node right

}

struct BST {

Node root

}

Void insertCourse(tree, node) {

If tree->node == null

Tree->root = node

node left and right = null

else

current = root

while current != null

if node->key < current->key

if current->left is null

current->left = node

current = null

else

current = current->left

else

if current->right is null

current->right = node

current = null

else

current = current->right

node->left = null

node->right = null

}

BST loadCourses(String fileName) {

create an empty BST

create and open the file

if the file cannot be opened

print error message

return an empty BST

for each line in the file

split the line into segments by ","

if the number of segments is less than 2

print error message

continue

Create course object

set course.courseNumber to segments[0]

set course.name to segments[1]

for each segment after 2

add the segment to course.prerequisites

insertCourse(tree, course)

for each course in the BST

for each prerequisite in course.prerequisites

if prerequisite does not equal any course

print error message

return the BST

}

void searchCourse(Tree<Course> courses, String courseNumber) {

current = tree root

while current isn’t null

if courseNumber = current

return current

else if current < courseNumber

current = current->left

else

current = current->right

return null

}

void printCourse(Node root) {

if root is null

return

printSortedCourses(root.left)

print courseNumber + course name

printSortedCoures(root.right)

}

void main() {

BST tree = null

while true Print "Menu:"

Print "1. Load courses"

Print "2. Print all courses"

Print "3. Print course details"

Print "9. Exit"

Get user input as choice

if choice == 1

tree = loadCourses("courses.txt")

else if choice == 2

if tree.root is null

Print "Error: No courses loaded."

else printSortedCourses(tree.root)

else if choice == 3

if tree.root is null

Print error message

else

Print "Enter course number:"

Get courseNumber

result = searchCourse(root, courseNumber)

Print result.courseNumber + course name

if result.course prerequisites is not empty

Print prerequisites

else if choice == 9

break

else

Print "Invalid selection."

}

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Create an empty BST | 1 | 1 | 1 |
| Open the file | 1 | 1 | 1 |
| For each line in the file | 1 | N | N |
| Split line into segments separated by “,” | 1 | N | N |
| If the number of segments is less than 2 | 1 | N | N |
| Continue | 1 | N | N |
| Create course object | 1 | N | N |
| Set course.courseNumber to segment[0] | 1 | N | N |
| Set course.name to segment[1] | 1 | N | N |
| For each segment after 2 | 1 | N | N |
| Add segment to prerequisites | 1 | N | N |
| insertCourse(tree,course) | 1 | N | N |
| for each course in the BST | 1 | N | N |
| for each prerequisite in course.prerequisites | 1 | N\*M | NM |
| if prerequisite does not equal any course | 1 | N | N |
| Print error message | 1 | N | N |
| Return BST | 1 | N | N |
| **Total Cost** | | | 14N+NMM+2 |
| **Runtime** | | | O(NM) |

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Create empty Vector | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| For each line in the file | 1 | N | N |
| Split line into segments separated by “,” | 1 | N | N |
| If the number of segments is less than 2 | 1 | N | N |
| Continue | 1 | N | N |
| Create course object | 1 | N | N |
| Set course.courseNumber to segment[0] | 1 | N | N |
| Set course.name to segment[1] | 1 | N | N |
| For each prerequisite |  | N | N |
| Add to prerequisites | 1 | N | N |
| Add course to to Vector | 1 | N | N |
| For each course in vector | 1 | N | N |
| For each prerequisite in course | 1 | M\*N | NM |
| Check if prerequisite exists | 1 | M\*N | NM^2 |
| Return courses | 1 | 1 | 1 |
| **Total Cost** | | | 11N+NM^2+NM+3 |
| **Runtime** | | | O(NM^2) |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Create empty Hash Table | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| For each line in the file | 1 | N | N |
| Split line into segments separated by “,” | 1 | N | N |
| If the number of segments is less than 2 | 1 | N | N |
| Continue | 1 | N | N |
| Create course object | 1 | N | N |
| Set course.courseNumber to segment[0] | 1 | N | N |
| Set course.name to segment[1] | 1 | N | N |
| For each prerequisite |  | N | N |
| Add to prerequisites | 1 | N | N |
| Add course to Hash Table | 1 | N | N |
| For each course in Hash Table | 1 | N | N |
| For each prerequisite in course | 1 | M\*N | NM |
| Check if prerequisite exists | 1 | M\*N | NM |
| Return courses | 1 | 1 | 1 |
| **Total Cost** | | | 11N+2NM+3 |
| **Runtime** | | | O(MN) |

**Evaluation**

Vectors are easy to implement compared to a hash table or a binary search tree. This simplicity does make it a little slower for searches though since it must be a linear search. The hash table is more complicated to implement but does a really good job when searching for elements and does not require to be sorted when searching. A hash table needs to have measures in place to avoid multiple collisions, making sure it remains balanced so it can maintain its effectiveness. Binary search trees maintain a sorted order, which can be useful for certain scenarios, but like the hash table takes more to set up. Binary search trees are typically faster at searching than vectors, but for large data sets may not be as applicable because of the memory required for the nodes.

For this program I would recommend using a binary search tree. For course searches and prerequisite validation a binary search tree would be an excellent choice, as it handles the data well. A BST would be more effective for printing the courses in sorted order compared to the hash table, which was a close second for this program.