Southern
New Hampshire
University

Chris Trimmer

CS-300-T1159 DSA: Analysis and Design

Milestone 5-3: Tree Data Structure Pseudocode

22EW1 – 09/29/2022

The purpose of this document is to provide a pseudocode and algorithm analysis of code for a course planner that we are going to design for ABC University (ABCU). The pseudocode will consist of functions that pertain to file parsing, creating objects and storing them in a binary search tree data structure, searching the data structure, and printing the data. Where applicable, a runtime analysis we be documented for a function.

The main object used in the assignment is a Course class. Each course will contain a string value for its id, a string value for its title, and a list will store prerequisites courses for the given Course object. I use a vector to store prerequisites because a course can have more than one prerequisite and a vector is suitable for dynamic insertion. The binary search tree will represent the schedule of class objects. At a high-level representation, a schedule will contain multiple courses, and each course in the schedule can have zero or more prerequisite courses.

File Parsing

```
Void LoadCourses(string filepath, BinarySearchTree* bst) {

  Create vector to store the master course list
  Call to create the master course list using filepath

  Create infile ifstream object
  Use infile object to open the filepath

  If the infile object returns null
    return to caller immediately

  Create string object, line, to hold a line read from the file
  Create a char object, delim, which is a "," (comma), to use as delimiter when reading each line
  Create a string object, word, to hold each word in the line
  Create a vector of strings, courseLine, that will hold each word

  Loop through infile object, and store each line in the line object
    Pass each line as an object to stringstream object, fullLine and parse using delim

    Loop through fullLine, and push back each word in the line to courseLine vector

    // check file format
    If the courseLine vector contains less than two objects, then this is an incomplete record
      Print output to user regarding invalid file format
      Return to caller

    // verify if prerequisite classes are valid
    For each word in courseLine vector
```

```
      Remove whitespace from each word

      If the word is not a valid course
         Then skip the word

   // now that we have a valid courseLine and valid prerequisites, perform object creation
   Call AddCourse

   (… outer loop continues after completion of creating the course object, and inserting into the
hash table)

   Clear the courseLine vector before starting the next loop

} // end LoadSchedule
```

Code	Line Cost	Execution Times	Total Cost
Create master course vector	1	n	n
Create ifstream object	1	1	1
Open filepath	1	1	1
if infile is null, return	2	1	2
Create 4 local variables	4	1	4
while file has lines to read	1	n	n
create sstream object	1	1	1
while there are words in the stream object	1	n	n
trim whitespace from each wordfor each sstream object	4	n	4n
push each word to vector	1	n	n
if courseLine less than 2	2	1	2
return to caller	1	1	1
Create course object	1	1	1
if size of courseLine > 2	2	1	2
for each prereq	1	n	n
compare prereq to master course list	1	n	n
if the prereq is found	2	1	2
insert prereq to course	1	1	1
Search tree to verify if course has already been added	1	logN+1	logN+1
if the course doesn't exist	2	1	2

then call Add Course	1	2logN + 9	2logN + 9
clear the courseLine vector	1	1	1
close the file	1	1	1
		Total Cost:	3logN + 10n + 32
		Runtime	O(NlogN)

Creating and Storing Objects

```
Void AddCourse(String courseLine) {

  Instantiate Course object

  For each word in line
    Set course id to line[0]
    Set course title to line[1]

  For each additional word in line
    Push back word into the prereq list stored in the Course object

  // the course now has all its data
  Call insert function of binary search tree class passing the Course object

}
```

Code	Line Cost	Execution Times	Total Cost
Instantiate blank course object	1	1	1
for each word in lines	1	n	n
set course id to line[0]	1	1	1
set course title to line[1]	1	1	1
for each additional word in lines	1	n	n
push back word to prereq vector	1	1	1
call insert function to pass course object	1	1	2logN + 10
		Total Cost:	2logN + 2n + 14
		Runtime	O(NlogN)

Inserting the Courses in the tree (Recursive)

```
Void InsertRecursive(Node* node, Course course) {
  If course.id is less than the node.id
    If node->left is null
      Create new node with course object
      Assign new node to node->left
    Else
      Recursively call InsertRecursive with node->left and the course
  Else
    If node->right is null
      Create new node with course object
      Assign new node to node->right
    Else
      Recursively call InsertRecursive with node->right and the course
}
```

Code	Line Cost	Execution Times	Total Cost
if id is less than node Id	2	1	2
if node->left is null	2	1	2
create new node with object	1	1	1
assign new node to node->left	1	1	1
else, recursively call insert w/left	1	logN+1	logN+1
else if node->right is null	1	1	1
create new node with object	1	1	1
assign new node to node->right	1	1	1
else, recursively call insert w/right	1	logN+1	logN+1
		Total Cost:	2logN + 9
		Runtime:	O(NlogN)

Get number of prerequisites

```
Int GetNumberOfPrereqs(string key) {

  For each letter in key
    Transform letter to uppercase

  Initialize sum variable to 0
  For each course
    If the key matches the course->id
      For each prereq
        Increment sum

  Return sum

}
```

Code	Line Cost	Execution Times	Total Cost
For each letter in key	1	n	1
transform letter to uppercase	1	1	1
Initialize sum variable to 0	1	1	1
for each course	1	n	n
if key matches courseId	1	1	1
for each prereq	1	n	n
increment sum	1	1	1
return sum	1	1	1
		Total Cost:	2n + 6
		Runtime	O(n)

Search for a Course (Recursive)

```
Course SearchRecursive(Node* node, string key) {
  If node is not null
    If key is equal to node->id
      Return the node->course
    Else if the key is less than the node->id
      Recursively call SearchRecursive with node->left and key
    Else
      Recursively call SearchRecursive with node->right and key
}
```

Code	Line Cost	Execution Times	Total Cost
for each letter in key	1	n	n
transform letter to upper	1	1	1
if node not equal null	1	1	1
if key equal to node->id	1	1	1

return node	1	1	1
else if key less than node->id	1	1	1
recursively call node->left	1	logN+1	logN+1
else recursively call node->right	1	logN+1	logN+1
		Total Cost:	2logN + N + 5
		Runtime	O(NlogN)

Print Courses – InOrder Traversal

```
Void inOrder(Node* node) {
  if node is not null
    recursively call inOrder with left side
    print the node info
    recursively call inOrder with right side
}
```

Code	Line Cost	Execution Times	Total Cost
if Node* not equal null	1	1	1
recursively call node->left	1	logN+1	logN+1
PrintCourseInfo	1	1	1
recursively call node->right	1	logN+1	logN+1
		Total Cost:	2logN + 4
		Runtime	O(NlogN)