Project One – Binary Search Tree

Pseudocode and Runtime Analyses

Chris Richards

Christopher.richards4@snhu.edu

bool OpenFile(filestream)

set file name to file name provided

Open the file

IF the file failed to open

return false # Indicating the file didn't open or load

ELSE

loadDataToBinaryTree(file)

IF the data failed to load to vector

return false # Indicating the file didn't open or load

End OpenFile

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| set file name to file name provided | 1 | | 1 | 1 |
| Open the file | 1 | | 1 | 1 |
| IF the file failed to open | 1 | | 1 | 1 |
| return false | 1 | | 1 | 1 |
| ELSE | 1 | | 1 | 1 |
| loadDataToBinaryTree (file) | 1 | | 1 | logn |
| IF the data failed to load to vector | 1 | | 1 | 1 |
| return false | 1 | | 1 | 1 |
| Total Cost | | | | logn + 7 |
| Runtime | | | | O(logn) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| filestream | | 1 | | |
| Total Input | | S(1) | | |
| Auxiliary Storage | |  | | |
| filename | | 1 | | |
| file | | 1 | | |
| Total Auxiliary | | S(2) | | |
| Total Space Complexity | | S(3) | | |

bool loadDataToBinaryTree(file)

initialize a new courseBinaryTree

WHILE there are lines to be read from file

read the current line in the file

initial string vector equal to splitLine(current line from file, ',')

IF the string vector does not contain at least two strings

return false

ELSE

create a new course node

set the course node id to the first string in the string vector

set the course node title to the second string in the vector

IF the string vector contains more than 2 strings # This is the prerequisite ids

FOR(the third item in the string vector to the End of the string vector)

add the prerequisite course id to the course node's prerequisite vector

End FOR

insert course node into courseBinaryTree

continue to the next line

return courseBinaryTree.verifyCourseData

End loadDataToCoursesBinaryTree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| initialize a new courseBinaryTree | 1 | | 1 | 1 |
| WHILE there are lines to be read from file | 1 | | n | n |
| read the current line in the file | 1 | | n | n |
| initial string vector equal to splitLine(current line from file, ',') | 1 | | n\*n | n2 |
| IF the string vector does not contain at least two strings | 1 | | n | n |
| return false | 1 | | 1 | 1 |
| ELSE | 1 | | n | n |
| create a new course node | 1 | | n | n |
| set the course node id to the first string in the string vector | 1 | | n | n |
| set the course node title to the second string in the vector | 1 | | n | n |
| IF the string vector contains more than 2 strings # This is the prerequisite ids | 1 | | n | n |
| FOR(the third item in the string vector to the End of the string vector) | 1 | | n\*n | n2 |
| add the prerequisite course id to the course node's prerequisite vector | 1 | | n\*n | n2 |
| insert course node into courseBinaryTree | 1 | | n\*logn | nlogn |
| courseBinaryTree.verifyCourseData | 1 | |  |  |
|  |  | |  |  |
| Total Cost | | | | logn + 7 |
| Runtime | | | | O(logn) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| file | | 1 | | |
| Total Input | | S(1) | | |
| Auxiliary Storage | |  | | |
| courseBinaryTree | | N | | |
| currentLine | | 1 | | |
| stringVector | | N | | |
| courseNode | | N | | |
| prerequisite | | 1 | | |
|  | |  | | |
| Total Auxiliary | | S(3N+2) | | |
| Total Space Complexity | | S(3N+3) | | |

vector<string> splitLine(string line, char delim)

set index begin to 0

set index End to 0

initialize a string vector

find delim character in line and set position to index End

IF(the delim character was not found)

set index End to the End of the line

End IF

WHILE(index begin is less than the length of the line - 1)

get the substring starting at index begin and Ending at index End # substring is by pos and length, so Ending is index End - index begin

appEnd substring to the string vector

set index begin to the position one after the position the delim character was found

set index End to the next position where the delim character is found.

IF(the delim character was not found)

set index End to the End of the line

End IF

End WHILE

return string vector

End splitLine

CourseBinaryTree

Constructor

set the root node to a null pointer

End Constructor

Deconstructor

delete the left node

delete the right node

End Deconstructor

Public void InOrder

call private inOrder(root)

End InOrder

Public void Insert(courseNode\*)

IF(the root is empty)

set the node's left to null

set the node's right to null

set the root to the new node

END IF

ELSE

call private addNode(root\*, course)

END ELSE

End Insert

Public void Remove(courseId)

call private removeNode(root\*, courseId)

END Remove

Public CourseNode\* Search(courseId)

call private search(root, courseId)

END Search

Public bool VerifyCourseData

return private verifyCourseData(root)

End VerifyCourseData

private void addNode(node\*, courseNode\*)

IF(courseNode's id is less than node's id)

IF(the left node is null)

set the node's left to the courseNode

END IF

ELSE

addNode(node->left, courseNode)

END ELSE

END IF

ELSE

IF(the right node is null)

set the node's right to the courseNode

END IF

ELSE

addNode(node->right, courseNode\*)

END ELSE

END ELSE

END addNode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| IF(courseNode's id is less than node's id) | 1 | | 1 | 1 |
| IF(the left node is null) | 1 | | 1 | 1 |
| set the node's left to the courseNode | 1 | | 1 | 1 |
| ELSE | 1 | | 1 | 1 |
| addNode(node->left, courseNode) | 1 | | logn | logn |
| ELSE | 1 | | 1 | 1 |
| IF(the right node is null) | 1 | | 1 | 1 |
| set the node's right to the courseNode | 1 | | 1 | 1 |
| ELSE | 1 | | 1 | 1 |
| addNode(node->right, courseNode\*) | 1 | | logn | logn |
| Total Cost | | | | 2logn + 8 |
| Runtime | | | | O(logn) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| node\* | | N | | |
| courseNode\* | | N | | |
| Total Input | | S(2N) | | |
| Auxiliary Storage | |  | | |
| Total Auxiliary | | S(0) | | |
| Total Space Complexity | | S(2N) | | |

private void inOrder(node\*)

IF(the node is null)

return

END IF

inOrder(node->left)

print node

inOrder(node->right)

END inOrder

prive CourseNode\* removeNode(node\*, courseId)

IF(the node is null)

return node

END IF

IF(the courseId is less than the node's courseId)

set the nodes left to removeNode(node->left, courseId)

END IF

ELSE IF(the courseId is greater than the node's courseId)

set the nodes right to removeNode(node->right, courseId)

END ELSE IF

ELSE

IF(the nodes left and right are null)

delete the node

return null

END IF

ELSE IF(the node's left is null)

set a temp node to the node's right

set the node's left and right to null

delete the node

return the temp node

END ELSE IF

ELSE IF(the node's right is null)

set a temp node to the node's left

set the node's left and right to null

delete the node

return the temp node

END ELSE IF

set a successor node to findSuccessor(node->right)

set the successor node's right to the node's right

set the node's left and right to null

delete the node

return the successor node

END ELSE

return the node

END removeNode

private CourseNode\* search(node\*, courseId)

IF(the node is null)

return null

END IF

IF(the courseId equals the node's courseId)

return the node

END IF

IF(the courseId is less than the node's courseId)

return search(node->left, courseId)

END IF

return search(node->right, courseId)

End search

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| IF(the node is null) | 1 | | 1 | 1 |
| return null | 1 | | 1 | 1 |
| IF(the courseId equals the node's courseId) | 1 | | 1 | 1 |
| return the node | 1 | | 1 | 1 |
| IF(the courseId is less than the node's courseId) | 1 | | 1 | 1 |
| return search(node->left, courseId) | 1 | | logn | logn |
| return search(node->right, courseId) | 1 | | logn | logn |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
| Total Cost | | | | 2logn + 5 |
| Runtime | | | | O(logn) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| node\* | | N | | |
| coursed | | 1 | | |
| Total Input | | S(N+1) | | |
| Auxiliary Storage | |  | | |
| Total Auxiliary | | S(0) | | |
| Total Space Complexity | | S(N+1) | | |

private CourseNode\* findSuccessor(node\*)

IF(the node's left is null)

return the node

END IF

findSuccessor(node->left)

End findSuccessor

private bool verifyCourseData(node\*)

bool valid = true

FOR EACH(prerequisite in node)

IF(search(root, prerequisite->courseId) returns nothing)

return false

END IF

END FOR EACH

valid = verifyCourseData(node->left)

valid = verifyCourseData(node->right)

return valid

end verifyCourseData

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| bool valid = true | 1 | | 1 | 1 |
| FOR EACH(prerequisite in node) | 1 | | n | n |
| IF(search(root, prerequisite->courseId) returns nothing) | 1 | | n\*logn | nlogn |
| return false | 1 | | 1 | 1 |
| valid = verifyCourseData(node->left) | 1 | | n\*n\*logn | n2logn |
| valid = verifyCourseData(node->right) | 1 | | n\*n\*logn | n2logn |
| Total Cost | | | | n2logn + n + 2 |
| Runtime | | | | O(n2logn) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| node\* | | N | | |
| Total Input | | S(N+1) | | |
| Auxiliary Storage | |  | | |
| Total Auxiliary | | S(0) | | |
| Total Space Complexity | | S(N) | | |

End CourseBinaryTree

struct CourseNode

Constructor

set left to null

set right to null

End Constructor

Deconstructor

delete left

delete right

End Deconstructor

string courseId

string courseTitle

vector<string> prerequisites

CourseNode\* left

CourseNode\* right

End Course

void printMenu

PRINT "1. Load course data[ENDL]"

PRINT "2. Print all course data[ENDL]"

PRINT "3. Find a course[ENDL]"

PRINT "0. EXIT[ENDL]"

END printMenu

void printCourse(course)

PRINT "[course.title] [ENDL]"

PRINT "Prerequisites:"

FOR EACH prerequisite in course

PRINT " [prerequisite]"

END FOR EACH

END printCourse

void Main(args)

INITIALIZE int userInput = -99

INITIALIZE courseBinaryTree courses

WHILE (userInput is not 0)

printMenu

GET userInput

IF(userInput is 1)

set fileStream to new FileStream(file from args)

OpenFile(fileStream, coursesVector)

END IF

IF(userInput is 2)

IF(courses.root is null)

PRINT "Courses have not yet been loaded"

END IF

ELSE

courses.InOrder()

END ELSE

END IF

IF(userInput is 3)

SET course = courses.Search(courseId from args)

IF(course is an empty course)

PRINT "Course not found"

END IF

ELSE

printCourse(course)

END ELSE

END IF

END WHILE

END Main