Project One – Hash Table

Psuedocode and Runtime Analyses

Chris Richards

Christopher.richards4@snhu.edu

##### BEGIN OPEN AND LOAD FILE DATA #####

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

loadDataToHashTable (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 |
| loadDataToHashTable (file) | 1 | | 1 | n2 |
| IF the data failed to load to vector | 1 | | 1 | 1 |
| return false | 1 | | 1 | 1 |
| Total Cost | | | | n2+7 |
| Runtime | | | | O(n2) |
| 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 loadDataToHashTable(file)

initialize a new courseHashTable

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 courseHashTable

continue to the next line

return courseHashTable.verifyCourseData

end loadDataToCoursesHashTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| initialize a new courseHashTable | 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 courseHashTable | 1 | | n | n |
| return courseHashTable.verifyCourseData | 1 | | 1 | 1 |
| Total Cost | | | | 3n2+9n+3 |
| Runtime | | | | O(n2) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| file | | 1 | | |
| Total Input | | S(1) | | |
| Auxiliary Storage | |  | | |
| courseHashTable | | N2 | | |
| currentFile | | 1 | | |
| stringVector | | N | | |
| courseNode | | N | | |
| Total Auxiliary | | S(N2+2n+1) | | |
| Total Space Complexity | | S(N2+2n+2) | | |

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

##### END OPEN AND LOAD FILE DATA #####

CourseHashTable

constructor()

set table size to default value

resize course node vector to table size

end constructor

constructor(int tableSize)

set table size to tableSize

resize course node vector to table size

end constructor

int hash(string key)

int convertedKey = ascii integer total of string

return hashed version of key

end hash

void insert(course node)

set key variable to hash(course node's course id)

end insert

CourseNode search(string courseId)

set int key to hash(courseId)

get currentNode located at bucket key in course node vector

IF(current node's courseId equals the courseId to be found)

return the courseNode

END IF

ELSE

set currentNode to currentNode's next Node

WHILE(the currentNode is not null)

IF(currentNode's courseId equals the courseId to be found)

return the currentNode

END IF

END WHILE

END ELSE

return an empty courseNode # we didn't find the course node

end search

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
|  |  | |  |  |
| Code | Line Cost | | # Times Executed | Total Cost |
| set int key to hash(courseId) | 1 | | 1 | 1 |
| get currentNode located at bucket key in course node vector | 1 | | 1 | 1 |
| IF(current node's courseId equals the courseId to be found) | 1 | | 1 | 1 |
| return the courseNode | 1 | | 1 | 1 |
| ELSE | 1 | | 1 | 1 |
| set currentNode to currentNode's next Node | 1 | | 1 | 1 |
| WHILE(the currentNode is not null) | 1 | | n | n |
| IF(currentNode's courseId equals the courseId to be found) | 1 | | n | n |
| return the currentNode | 1 | | 1 | 1 |
| return an empty courseNode | 1 | | 1 | 1 |
| Total Cost | | | | 2n + 8 |
| Runtime | | | | O(n) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| coursed | | 1 | | |
| Total Input | | S(1) | | |
| Auxiliary Storage | |  | | |
| key | | 1 | | |
| currentNode | | N | | |
| Total Auxiliary | | S(N+1) | | |
| Total Space Complexity | | S(N+2) | | |

void printCourses

FOR EACH(bucket in the courseNodes vector)

FOR EACH(node in the currentBucket's list)

print the node's course id

print the node's course title

FOR EACH(item in the node's prerequisite vector)

print the current prerequisite

END FOR EACH

END FOR EACH

END FOR EACH

end printCourses

bool verifyCourseData

FOR EACH(node in courseNodes vector)

FOR EACH(node in the currentBucket's list)

FOR EACH(prerequisite in node's prerequisite vector)

IF(courseHashTable.search(prerequisite) returns nothing)

return false

END IF

END FOR EACH

END FOR EACH

END FOR EACH

return true # All prerequisites found

end verifyCourseData

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | |
| Code | Line Cost | | # Times Executed | Total Cost |
| FOR EACH(node in courseNodes vector) | 1 | | n | n |
| FOR EACH(node in the currentBucket's list) | 1 | | n\*n | n2 |
| FOR EACH(prerequisite in node's prerequisite vector) | 1 | | n\*n\*n | n3 |
| return false | 1 | | 1 | 1 |
| return true | 1 | | 1 | 1 |
| Total Cost | | | | n3+n2+n+2 |
| Runtime | | | | O(n3) |
| Runtime Analysis Space Complexity | | | | |
| Inputs | | Cost | | |
| Total Input | | S(0) | | |
| Auxiliary Storage | |  | | |
| courseNodesVector | | N2 | | |
| courseNode | | N | | |
| courseHashTable | | N2 | | |
| Total Auxiliary | | S(2N2+N) | | |
| Total Space Complexity | | S(2N2+N) | | |

end CourseHashTable

struct CourseNode

string courseId

string courseTitle

vector<string> prerequisites

CourseNode\* next

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 courseHashTable 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.tableSize < 1)

PRINT "Courses have not yet been loaded"

END IF

ELSE

courses.printCourses

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