Project One - Vector

Psuedocode and Runtime Analyses

Chris Richards

Christopher.richards4@snhu.edu

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

bool OpenFile(filestream, &coursesVector)

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

SET bool loaded to loadDataToVector(file, coursesVector)

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 | 1 | 1 | 1 |
| Open the file | 1 | 1 | 1 |
| If the file failed to open | 1 | 1 | 1 |
| Return false | 1 | 1 | 1 |
| loadDataToVector | 1 |  | n2logn |
| If the data failed to load | 1 | 1 | 1 |
| Return false | 1 | 1 | 1 |
| Total Cost | | | n2logn + 6 |
| Runtime | | | O(n2logn) |
|  | | |  |
| Runtime Analysis Space Complexity | | | |
| Inputs | | Cost | |
| Filestream | | 1 | |
| coursesVector | | N | |
| Total Input | | S(N+1) | |
| Auxiliary Storage | |  | |
| filename | | 1 | |
| File | | 1 | |
| Total Auxiliary | | S(1) | |
| Total Space Complexity | | S(N+3) | |

bool loadDataToCoursesVector(file, &coursesVector)

WHILE(there are lines to be read from file)

read the current line in the file

vector<string> lineVector = splitString(current line)

IF the lineVector does not contain at least two strings

RETURN false

ELSE

create a new course object

SET the course id to the first string in the string array

SET the course title to the second string in the array

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

FOR(the third item in the lineVector to the END of the lineVector)

ADD the prerequisite course id to the course object's prerequisite vector

ADD the course to the course vector

END WHILE

sortCourses(coursesVector, 0, coursesVector.size())

verifyCourses(coursesVector)

END loadDataToCoursesVector

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Runtime Analysis Time Complexity | | | |
| Code | Line Cost | # Times Executed | Total Cost |
| While (there are lines) | 1 | n | n |
| Read current line | 1 | n | n |
| vector lineVector = splitString | 1 | n\*n | n2 |
| if lineVector.size() < 2 | 1 | n | n |
| return false | 1 | 1 | 1 |
| else | 1 | n | n |
| for(third item to end of lineVector) | 1 | n\*n | n2 |
| ADD prerequisite to course | 1 | n\*n | n2 |
| ADD the course to courseVector | 1 | n | n |
| sortCourses | 1 | nlogn | nlogn |
| verifyCourses | 1 | n2\*logn | n2\*logn |
| Total Cost | | | n2logn + 3n2+5n+1 |
| Runtime | | | O(n2logn) |
|  | | |  |
| Runtime Analysis Space Complexity | | | |
| Inputs | | Cost | |
| File | | 1 | |
| coursesVector | | N | |
| Total Input | | S(N+1) | |
| Auxiliary Storage | |  | |
| currentLine | | N | |
| lineVector | | N | |
| Course | | 1 | |
|  | |  | |
| Total Auxiliary | | S(2N+1) | |
| Total Space Complexity | | S(3N+1) | |

## Sort the courses using quicksort ##

void sortCourses(&coursesVector, start, end)

IF(start is less than end)

partition = partitionCourses(coursesVector, start, end)

sortCourses(coursesVector, start, partition)

sortCourses(coursesVector, partition + 1, end)

END IF

END sortCourses

|  |  |  |  |
| --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | |
|  |  |  |  |
| Code | Line Cost | # Times Executed | Total Cost |
| IF(start is less than end) | 1 | 1 | 1 |
| partition = partitionCourses | 1 | nlogn | nlogn |
| sortCourses | 1 | n\*logn | nlogn |
| sortCourses | 1 | n\*logn | nlogn |
| Total Cost | | | 3nlogn + 1 |
| Runtime | | | O(nlogn) |
|  | | |  |
| Runtime Analysis Space Complexity | | | |
| Inputs | | Cost | |
| coursesVector | | N | |
| start | | 1 | |
| end | | 1 | |
| Total Input | | S(N+2) | |
| Auxiliary Storage | |  | |
| partition | | 1 | |
| Total Auxiliary | | S(1) | |
| Total Space Complexity | | S(N+2) | |

## Partition the courses for quicksort algorithm

void partitionCourses(&coursesVector, start, end)

SET low = start

SET high = end

SET Course pivot = coursesVector.at((high + low) / 2)

SET bool done to false

WHILE(done is not true)

WHILE(coursesVector.at(low).courseId is less than pivot.courseId)

INCREMENT low

END WHILE

WHILE( pivot.courseId < coursesVector.at(high).courseId)

DECREMENT high

END WHILE

IF(low is greater or equal to high)

SET done to true

END IF

ELSE

INITIALIZE tempCourse to coursesVector.at(low)

SET coursesVector.at(low) to coursesVector.at(high)

SET coursesVector.at(high) to tempCourse

INCREMENT low

DECREMENT high

END ELSE

END WHILE

return high

END partitionCourses

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | | |
|  |  | |  | |  |
| Code | Line Cost | | # Times Executed | | Total Cost |
| SET high | 1 | | 1 | | 1 |
| SET low | 1 | | 1 | | 1 |
| Set course pivot = coursesVector.at(low + high) / 2) | 1 | | 1 | | 1 |
| SET bool done = false | 1 | | 1 | | 1 |
| WHILE(done not true) | 1 | | n | | n |
| WHILE(coursesVector.at(low).courseId is less than pivot.courseId) | 1 | | n\*logn | | nlogn |
| INCREMENT low | 1 | | n\*logn | | nlogn |
| WHILE( pivot.courseId < coursesVector.at(high).courseId) | 1 | | n | | n |
| DECREMENT high | 1 | | n\*logn | | nlogn |
| IF(low is greater or equal to high) | 1 | | n | | n |
| SET done to true | 1 | | 1 | | 1 |
| ELSE | 1 | | n | | n |
| INITIALIZE tempCourse to coursesVector.at(low) | 1 | | n | | n |
| SET coursesVector.at(low) to coursesVector.at(high) | 1 | | n | | n |
| SET coursesVector.at(high) to tempCourse | 1 | | n | | n |
| INCREMENT low | 1 | | n | | n |
| DECREMENT high | 1 | | n | | n |
| return high | 1 | | 1 | | 1 |
| Total Cost | | | | | 3nlogn +8n+5 |
| Runtime | | | | | O(nlogn) |
|  | | | |  | |
| Runtime Analysis Space Complexity | | | | | |
| Inputs | | Cost | | | |
| coursesVector | | N | | | |
| start | | 1 | | | |
| end | | 1 | | | |
| Total Input | | S(N+2) | | | |
| Auxiliary Storage | |  | | | |
| high | | 1 | | | |
| low | | 1 | | | |
| pivot | | 1 | | | |
| done | | 1 | | | |
| tempCourse | | 1 | | | |
| Total Auxiliary | | S(5) | | | |
| Total Space Complexity | | S(N+7) | | | |

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

## Helper function to split string into vector<string> via delim character ##

vector<string> splitString(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 splitString

## Verify that each prerequisite in a course is an actual course ##

void verifyCourses(&coursesVector)

FOR EACH course in coursesVector

FOR EACH prerequisite in course.prerequisites

IF(findCourse(prerequisite) returns an empty course)

REMOVE course from coursesVector

BREAK loop

END IF

END FOR EACH

END FOR EACH

END verifyCourses

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | | |
|  |  | |  | |  |
| Code | Line Cost | | # Times Executed | | Total Cost |
| FOR EACH course in coursesVector | 1 | | n | | n |
| FOR EACH prerequisite in course.prerequisites | 1 | | n\*n | | n2 |
| IF(findCourse(prerequisite) returns an empty course) | 1 | | n\*n\*logn | | n2\*logn |
| REMOVE course from coursesVector | 1 | | 1 | | 1 |
| Total Cost | | | | | n2\*logn+3nlogn +8n+5 |
| Runtime | | | | | O(n2\*logn) |
|  | | | |  | |
| Runtime Analysis Space Complexity | | | | | |
| Inputs | | Cost | | | |
| coursesVector | | N | | | |
| Total Input | | S(N) | | | |
| Auxiliary Storage | |  | | | |
| course | | 1 | | | |
| prerequisite | | 1 | | | |
| Total Auxiliary | | S(2) | | | |
| Total Space Complexity | | S(N+2) | | | |

## Print all courses in the provided vector<Course>

void printCourses(coursesVector)

FOR EACH course in the coursesVector

printCourse(course)

END FOR EACH

END printCourses

## Find a course by id within the provided vector<Course>

Course findCourse(coursesVector, id)

return findCouse(coursesVetor, id, 0, coursesVector.size() - 1)

END findCourse

## Overloaded function to include

Course findCourse(coursesVector, id, start, end)

IF(start < end)

SET mid to (start + end) / 2

IF(coursesVector.at(mid).courseId equals id)

RETURN coursesVector.at(mid)

END IF

IF(id is less than coursesVector.at(mid))

return findCourse(coursesVector, id, start, mid - 1)

END IF

IF(id is greater than coursesVector.at(mid)

return findCourse(coursesVector, id, mid + 1, end)

END IF

END IF

return empty Course

END findCourse

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Runtime Analysis Time Complexity | | | | | |
|  |  | |  | |  |
| Code | Line Cost | | # Times Executed | | Total Cost |
| IF(start < end) | 1 | | 1 | | 1 |
| SET mid to (start + end) / 2 | 1 | | 1 | | 1 |
| IF(coursesVector.at(mid).courseId equals id) | 1 | | 1 | | 1 |
| RETURN coursesVector.at(mid) | 1 | | 1 | | 1 |
| IF(id is less than coursesVector.at(mid)) | 1 | | 1 | | 1 |
| return findCourse(coursesVector, id, start, mid - 1) | 1 | | logn | | logn |
| IF(id is greater than coursesVector.at(mid) | 1 | | 1 | | 1 |
| return findCourse(coursesVector, id, mid + 1, end) | 1 | | logn | | logn |
| return empty course | 1 | | 1 | | 1 |
| Total Cost | | | | | 2logn + 7 |
| Runtime | | | | | O(logn) |
|  | | | |  | |
| Runtime Analysis Space Complexity | | | | | |
| Inputs | | Cost | | | |
| coursesVector | | N | | | |
| id | | 1 | | | |
| start | | 1 | | | |
| end | | 1 | | | |
| Total Input | | S(N+3) | | | |
| Auxiliary Storage | |  | | | |
| mid | | 1 | | | |
| Total Auxiliary | | S(1) | | | |
| Total Space Complexity | | S(N+4) | | | |

struct Course

string courseId

string courseTitle

vector<string> prerequisites

END Course

vector<string> splitString(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 splitString

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 vector<course> coursesVector

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(coursesVector.size() < 1)

PRINT "Courses have not yet been loaded"

END IF

ELSE

printCourses(coursesVector)

END ELSE

END IF

IF(userInput is 3)

SET course = findCourse(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