Tim Brady

October 5, 2022

CS300 – DSA: Analysis and Design 22EW1

Professor Atkison

**Module 6 Project One**

**Vector**

**Opening a file**

void readFile(File schedule, lines[])

CREATE vector for course number, course title and prereq

CREATE variable for current line and variable for stop set to false

WHILE not end of the file

CREATE courseInfo array, split by comma and append each line to vector

IF length of courseInfo < 2

SET stop variable equal true and break

SET courseNumber[i] equal courseInfo[0]

SET courseTitle[i] equal courseInfo[1]

INCREMENT i

IF length of courseInfo >= 2

FOR j equal 2 to length of courseInfo

Prereq[k] equal courseInfo[j]

INCREMENT j

IF stop equal false

FOR each value in prereq

IF value not in courseNumber

SET stop equal true and break

RETURN stop

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CREATE arrays for course number, course title and prereq | 1 | 1 | 1 |
| CREATE variable for current line and variable for stop set to false | 1 | 1 | 1 |
| WHILE not end of the file | 1 | n | n |
| CREATE courseInfo vector, split by comma and append each line to vector | 1 | 1 | 1 |
| IF length of courseInfo < 2 | 1 | n | n |
| SET stop variable equal true and break | 1 | 1 | 1 |
| SET courseNumber[i] equal courseInfo[0] | 1 | 1 | 1 |
| SET courseTitle[i] equal courseInfo[1] | 1 | 1 | 1 |
| INCREMENT i | 1 | 1 | 1 |
| IF length of courseInfo >= 2 | 1 | n | n |
| FOR j equal 2 to length of courseInfo | 1 | n | n |
| Prereq[k] equal courseInfo[j] | 1 | 1 | 1 |
| INCREMENT j | 1 | 1 | 1 |
| IF stop equal false | 1 | n | n |
| FOR each value in prereq | 1 | n | n |
| IF value not in courseNumber | 1 | n | n |
| SET stop equal true and break | 1 | 1 | 1 |
| RETURN stop | 1 | 1 | 1 |
| **Total Cost** | | | 7n+11 |
| **Runtime** | | | O(n) |

**Creating object**

CLASS course

CREATE variables for courseNumber and courseTitle

CREATE array for prereq

CREATE constructor

SPLIT the lines using comma and set courseNumber to 0 and courseTitle to 1

IF length of line >= 2

SET prereq equal to 2 to length of line

void createCourse(Courses <course>, File schedule)

IF readFile(schedule, lines) equals false

FOR each line in line array

APPEND new line to courses

ELSE print “file not read”

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CREATE variables for courseNumber and courseTitle | 1 | 1 | 1 |
| CREATE array for prereq | 1 | 1 | 1 |
| CREATE constructor | 1 | 1 | 1 |
| SPLIT the lines using comma and set courseNumber to 0 and courseTitle to 1 | 1 | n | n |
| IF length of line >= 2 | 1 | n | n |
| SET prereq equal to 2 to length of line | 1 | 1 | 1 |
| IF readFile(schedule, lines) equals false | 1 | n | n |
| FOR each line in line array | 1 | n | n |
| APPEND new line to courses | 1 | 1 | 1 |
| ELSE print “file not read” | 1 | 1 | 1 |
| **Total Cost** | | | 4n+6 |
| **Runtime** | | | O(n) |

**Printing Course Info**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| for all courses | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the course information | 1 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 1 | 1 | 1 |
| **Total Cost** | | | 3n+2 |
| **Runtime** | | | O(n) |

**Hash Table**

**Opening a File**

void readFile(File schedule, lines[])

IF file is efile

DECLARE line variable

READ input file

IF file is open

WHILE file has no errors

GET line from file

IF line is not blank

APPEND line to hash table

CLOSE file

IF file size = 0

PRINT error

PARSE header

PARSE content

ELSE

PRINT error

ELSE

IMPLEMENT istringstream

WHILE getting lines

IF line is not blank

APPEND to hash

IF file size = 0

PRINT error

PARSE header

PARSE content

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| IF file is efile | 1 | 1 | 1 |
| DECLARE line variable | 1 | 1 | 1 |
| READ input file | 1 | 1 | 1 |
| IF file is open | 1 | 1 | 1 |
| WHILE file has no errors | 1 | n | n |
| GET line from file | 1 | 1 | 1 |
| IF line is not blank | 1 | n | n |
| APPEND line to hash table | 1 | 1 | 1 |
| CLOSE file | 1 | 1 | 1 |
| IF file size = 0 | 1 | 1 | 1 |
| PRINT error | 1 | 1 | 1 |
| PARSE header | 1 | 1 | 1 |
| PARSE content | 1 | n | n |
| ELSE PRINT error | 1 | 1 | 1 |
| ELSE IMPLEMENT istringstream | 1 | 1 | 1 |
| WHILE getting lines | 1 | n | n |
| IF line is not blank | 1 | n | n |
| APPEND to hash | 1 | 1 | 1 |
| IF file size = 0 | 1 | 1 | 1 |
| PRINT error | 1 | 1 | 1 |
| PARSE header | 1 | 1 | 1 |
| PARSE content | 1 | n | n |
| **Total Cost** | | | 6n+16 |
| **Runtime** | | | O(n) |

**Create Object**

CREATE key for bid

RETRIEVE node using key

IF no entry found

ASSIGN node key position

INSERT into vector

ELSE

IF old node key equals UINT\_MAX

SET key

SET old node to bid

SET old node next to null

ELSE

WHILE next is not null

SET node equal next

ASSIGN with bid and key

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CREATE key for bid | 1 | 1 | 1 |
| RETRIEVE node using key | 1 | 1 | 1 |
| IF no entry found | 1 | 1 | 1 |
| ASSIGN node key position | 1 | 1 | 1 |
| INSERT into vector | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF old node key equals UINT\_MAX | 1 | 1 | 1 |
| SET key | 1 | 1 | 1 |
| SET old node to bid | 1 | 1 | 1 |
| SET old node next to null | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| WHILE next is not null | 1 | n | n |
| SET node equal next | 1 | 1 | 1 |
| ASSIGN with bid and key | 1 | 1 | 1 |
| **Total Cost** | | | n+13 |
| **Runtime** | | | O(n) |

**Print Course Info**

FOR node begin to end

IF key not equal to UINT\_MAX

PRINT course info

SET node equal next

WHILE node not equal to null

PRINT course info

SET node = next

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| FOR node begin to end | 1 | n | n |
| IF key not equal to UINT\_MAX | 1 | 1 | 1 |
| PRINT course info | 1 | 1 | 1 |
| SET node equal next | 1 | 1 | 1 |
| WHILE node not equal to null | 1 | n | n |
| PRINT course info | 1 | 1 | 1 |
| SET node = next | 1 | 1 | 1 |
| **Total Cost** | | | 2n+5 |
| **Runtime** | | | O(n) |

**Binary Tree**

**Opening a File**

void readFile(File schedule, lines[])

CREATE arrays for course number, course title and prereq

CREATE variable for current line and variable for stop set to false

WHILE not end of the file

CREATE courseInfo array, split by comma and append each line to array

IF length of courseInfo < 2

SET stop variable equal true and break

SET courseNumber[i] equal courseInfo[0]

SET courseTitle[i] equal courseInfo[1]

INCREMENT i

IF length of courseInfo >= 2

FOR j equal 2 to length of courseInfo

Prereq[k] equal courseInfo[j]

INCREMENT j

IF stop equal false

FOR each value in prereq

IF value not in courseNumber

SET stop equal true and break

RETURN stop

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CREATE arrays for course number, course title and prereq | 1 | 1 | 1 |
| CREATE variable for current line and variable for stop set to false | 1 | 1 | 1 |
| WHILE not end of the file | 1 | n | n |
| CREATE courseInfo array, split by comma and append each line to array | 1 | 1 | 1 |
| IF length of courseInfo < 2 | 1 | n | n |
| SET stop variable equal true and break | 1 | 1 | 1 |
| SET courseNumber[i] equal courseInfo[0] | 1 | 1 | 1 |
| SET courseTitle[i] equal courseInfo[1] | 1 | 1 | 1 |
| INCREMENT i | 1 | 1 | 1 |
| IF length of courseInfo >= 2 | 1 | n | n |
| FOR j equal 2 to length of courseInfo | 1 | n | n |
| Prereq[k] equal courseInfo[j] | 1 | 1 | 1 |
| INCREMENT j | 1 | 1 | 1 |
| IF stop equal false | 1 | n | n |
| FOR each value in prereq | 1 | n | n |
| IF value not in courseNumber | 1 | 1 | 1 |
| SET stop equal true and break | 1 | 1 | 1 |
| RETURN stop | 1 | 1 | 1 |
| **Total Cost** | | | 6n+12 |
| **Runtime** | | | O(n) |

**Create Object**

INITIALIZE Course Structure struct Course  
WHILE not end of the file  
 FOR every line in the file  
 SET course ID, and Course Name  
 WHILE not new line

SET prerequisites

DEFINE binary tree  
SET root = null  
DEFINE Insert method  
IF root is null

Current = root   
ELSE

IF course smaller than root

IF left is null

ADD to left  
ELSE  
 IF course is smaller

ADD to left  
ELSE IF course number is larger

ADD to right

ELSE

IF left is null

ADD to right  
ELSE  
 IF course is smaller

ADD to left  
ELSE IF course number is larger

ADD to right

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| INITIALIZE Course Structure struct Course | 1 | 1 | 1 |
| WHILE not end of the file | 1 | n | n |
| FOR every line in the file | 1 | n | n |
| SET course ID, and Course Name | 1 | 1 | 1 |
| WHILE not new line | 1 | n | n |
| SET prerequisites | 1 | 1 | 1 |
| DEFINE binary tree | 1 | 1 | 1 |
| SET root = null | 1 | 1 | 1 |
| DEFINE Insert method | 1 | 1 | 1 |
| IF root is null | 1 | 1 | 1 |
| Current = root | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF course smaller than root | 1 | 1 | 1 |
| IF left is null | 1 | 1 | 1 |
| ADD to left | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF course is smaller | 1 | 1 | 1 |
| ADD to left | 1 | 1 | 1 |
| ELSE IF course number is larger | 1 | 1 | 1 |
| ADD to right | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF left is null | 1 | 1 | 1 |
| ADD to right | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF course is smaller | 1 | 1 | 1 |
| ADD to left | 1 | 1 | 1 |
| ELSE IF course number is larger | 1 | 1 | 1 |
| ADD to right | 1 | 1 | 1 |
| **Total Cost** | | | 3n+25 |
| **Runtime** | | | O(log(n)) |

**Print Course Info**

SET current = root

WHILE current not null

IF course = current

RETURN current prerequisites

ELSE IF course smaller than current

CURRENT = current left

ELSE

CURRENT = current right

RETURN null

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| SET current = root | 1 | 1 | 1 |
| WHILE current not null | 1 | n | n |
| IF course = current | 1 | 1 | 1 |
| RETURN current prerequisites | 1 | 1 | 1 |
| ELSE IF course smaller than current  CURRENT = current left | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| CURRENT = current right | 1 | 1 | 1 |
| RETURN null | 1 | 1 | 1 |
| **Total Cost** | | | n+8 |
| **Runtime** | | | O(n) |

There are advantages and disadvantages for each of these three data structures. Vectors are straight forward and make adding or removing items in the list very easy when an item is at the end of the list. However, the list must be sorted and dealing with items in the middle of the list is not so efficient. Each item would need to be moved when doing this which makes dealing with large vectors more difficult.

Hash tables make accessing each item easy and fast as items are sorted by their keys. Disadvantages include needing to have an estimate of the required size and possible degradation of the database as more collisions occur. The larger the data set, the more likely collisions will happen.

The binary search tree with a runtime of O(log(n)) has fast insertion. Each iteration eliminates half of the list and greatly reduces time needed. However, with an unbalanced tree, the performance will take a hit. With this is mind, I would choose a binary search tree for the project. The runtime is slightly better with this option and doing things like sorting or inserting anywhere in the list is going to be easier.