Brent Longstreet

Professor Rissor

August 4, 2023

Project One

**Opening and reading file**

open file

if file is found

    while current line is not end of file

        parse data

        if number of parameters is < 2

            throw error

        if last parameter IS NOT the first parameter in any line of the file

            throw error

else

    throw error

close file

**Vector**

Create struct Course{} with courseID, courseName, Prequisite

vector<Course> loadCourses(string csvPath)

for loop through file

    Course course

    set course.courseId

    set course.name

        while not end of current line

            append course.prerequsite

    pushback course item

**Binary Search Tree (BST)**

Define bst

bst = new BinarySearchTree

Course course

If root node is null

    set root equal to new node

else

    call addNode method passing in the root course

if passed-in courseId > current nodes value

    if current node's left child is null

        set current node's left to passed-in value

    else

        recurse down left node

else

    if current node's right child is null

        set current node's left to passed-in value

    else

        recurse down right node

for loop until end of line

    set course.courseId

    set course.name

    while not end of current line

        append course.prerequsite

    insert course into bst

**HashTable**

initialize course Vector vector<Node> nodes

create table class

create insert method & insert course data into hash table

Create key by hashing course.courseId

Retrieve node using key

if no results are found for given key

    create new node passing in id and key

    insert node

else

    if node is not used

        set previous node's key to key

        set previous node's Id to Id

        set previous node's next to null

    else (node is used)

        while previous node's next is not empty

            set previous node to previos node's next

        add new node to the end

for loop until end of line in file

    set course.courseId

    set course.name

    while not end of current line

        append course.prerequsite

    insert course into hashtable

**Print course information and prerequisites**

**Vector**

create print course info method

retreive input for course.courseId

while loop until vector is empty

    if course.courseId matches input

        print course.courseId and course.name

    while there is still prerequsites

        print course.prerequsite

**HashTable**

create print course info method

retreive input for course.courseId

create key based on given courseId

if current node is null

    return null

if current node matches key

    print course information

else

    while node is not null

        if key matches courseId

            print course information

        set current node to next node

**Binary Search Tree (BST)**

create print course info method

retreive input for course.courseId

set current node to root

while current is not null

    if current node equals course.courseId

        print course information

        while prerequsite is not null

            print course.prerequsite

    if current node is less than course.courseId

        update current to right

    else

        update current to left

**Menu**

set choice to 0

while choice is not 4

    print menu options

    swtich statement

    case 1

        call loadCourses() passing in parameters

    case 2

        call printSorted() passing in parameters

    case 3

        call printCourseInformation() passing in parameters

    case 4

        print goodbye

        break

**Print list of course in alphanumeric order**

create printSorted() method

create partition method

    set lowIdx to 1st element

    set highIdx to last element

    set midPoint to (lowIdx + highIdx) / 2

    set pivot to midPoint

    while pivot is < highIdx

        decrement highIdx

    call swap() method to swap low values to the left of pivot and high ones to the right

        create temp and set it equal to high midIdx

        lowIdx = highIdx

        highIdx = temp

create quicksort(start, end) method

    set lowIdx = start

    set midIdx = 0

    set highIdx = end

    if start >= end

        return

    create lowEndIdx equal to partition(lowIdx, highIdx)

    recursive calls to quicksort passing in the new start and end

create printCourseInformation() method

 for loop until end of vector

    display course[i] informations

create inOrderTraversal() method

    if node is not null

        call inOrderTraversal(node->left)

        print course information

        call inOrderTraversal(node->right)

**Runtime**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | n | 1 |
| **For loop until end of file** | 1 | n | n |
| **Set course values** | 1 | n | n |
| **While loop until prereq is empty** | 1 | n | n |
| **Append prereq** | 1 | n | n |
| **Pushback course item** | 1 | n | n |
|  |  |  |  |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |
|  | | |  |

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define BST** | 1 | 1 | 1 |
| **If root node is null, set root to new node** | 2 | 1 | 1 |
| **Call addNode method** | 1 | n | n |
| **If courseId is less than node, add node to left** | 1 | n | n |
| **If childs not null recurse down left side** | 2 | n | n |
| **If courseId is greater than node, add node to right** | 1 | n | n |
| **If childs not null recurse down right side** | 2 | n | n |
| **For loop until end of file** | 1 | n | n |
| **Create vector course item** | 1 | n | n |
| **While loop until prereq is empty** | 1 | n | n |
| **Append prereq** | 1 | n | n |
| **Insert course item into BST** | 1 | n | n |
| **Total Cost** | | | 10n + 1 |
| **Runtime** | | | O(n) |

| **HashTable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hash table** | 1 | 1 | 1 |
| **Create key** | 1 | 1 | 1 |
| **If no entry, assign node to key** | 2 | n | n |
| **else** | 1 | n | n |
| **If node is not used** | 1 | n | n |
| **Update previous nodes key, bid, and next to null** | 3 | n | n |
| **else** | 1 | n | n |
| **While prev nodes next is not null** | 1 | n | n |
| **Set prev node to previous nodes next** | 1 | n | n |
| **Add new node to end** | 1 | n | n |
| **For loop until end of file** | 1 | n | n |
| **Set course id, name** | 1 | n | n |
| **While loop until prereq is empty** | 1 | n | n |
| **Append prereqs** | 1 | n | n |
| **Insert course into hashtable** | 1 | n | n |
| **Total Cost** | | | 14n + 2 |
| **Runtime** | | | O(n) |

The three data structures I’ve analyzed possess both advantages and disadvantages for the given scenario. The vector had the shortest run time out of all the data structures. Adding the course objects was straightforward and required little code. The disadvantage of the vector comes from its ability to search for a specific course. Trying to search for a specific course in a vector requires a check on every item until the match is found. The binary search tree had the second shortest run time. Binary search trees offer quick search times due to splitting the dataset in half and searching based on greater or less than values. A disadvantage to using a binary search tree is with an uneven tree it doesn’t result in quick search times and has double the amount of runtime as a vector. An advantage to using a hash table is its ability to access an element quickly. This is due to hashing a key and accessing the element without the need to search the entire vector. A disadvantage to using a hash table is the complexity and the slow runtime.

Now that I have analyzed all three data structures, I would recommend the use of a vector in this scenario. Based on the Big O analysis results, the vector offered the lowest runtime. Compared to the other data structures, the vector was twice as fast. Additionally, vectors are dynamic and don’t need a prespecified size like the hash table. The vectors quick and simple implementation outweigh its disadvantages and will be the best choice for the given scenario.