**6-2 Project one:**

Reading file;

Use fstream to open file

Creating method void loadCourses (string, csv path, structure)

Open file, if the return value is “-1”, file not created

Else file is not found

While it is not EOF (end of file)

Read each line

If less than two values in a line, return Error

else read parameters

if there is a third and more parameters

if third or more parameters is in first parameter elsewhere,

continue

else return error

close file

hold course information:

create struct course { }

create identifiers: Course Id, Course Name, Prereqs

//vector //

Vector < course > load Course (string csv path)

For (int i = 0 ; i < file.rowCount ( ); i + +) {

create a data structure and add to the collection of courses

course # ;

course. courseID = file [ i ] [ 1 ] ;

course. name = file [ i ] [ 0 ] ;

while not end of line

course.prereq = file [ i ] [ 8 ] ;

course.push\_back (course) ;

//hash//

Create HastTable

create node struct

course course

unsigned int key

vector < node > nodes

define tableSize

unsigned int has (int key)

Create insert method void Hast Table : : Insert (course course)

Create the key for the given course, search for node with value of key

if no entry found for key

assign this node to the key position

else if node is used

assign old node key to Unit\_Max, set to key, set old node to

course and old node next to

null pointer

else find the next open node

add new newNode to end

void loadCourses (string csv path, HashTable \* hashTable)

loop to read rows of a csv file

for (unsigned int i = 0 ; i < file.rowCount ( ) ; i + +) {

Create a data structure and add to the collection of courses

course course ;

course. course Id = file [ i ] [ 1 ];

course.name = file [ i ] [ 0 ] ;

while not end of line

course.prereq = file [ i ] [ 8 ] ;

hash Table -> insert (course) ;

//tree //

Define a binary search tree to hold all courses

BinarySearchTree \* bst ;

bst = new BinarySearchTree ( ) ;

Course course ;

Create add node method void BinarySearchTree : : add Node (Node) \* node, course course)

if root ir null add root

if node is less than root than add to the left

if no left node

this node becomes left

if node is greater than root add right

if no right node

this node becomes right

void load courses (string csv path, BinarySearchTree \* bst)

loop to read row of a csv file

for (unsigned int i = 0 < file.rowCount ( ) ; i + +) {

create a data structure and add to the collection of courses

course course ;

course. Course Id = file [ i ] [ 1 ] ;

course.name = file [ i ] [ 0 ] ;

while not end of line

Course. Prereqs = file [ i ] [ 8 ] ;

bst -> insert (course)

Print Course info plus preeqs:

//vector //

Create method void print Course information ( vector < course > courses, string course id )

Get input for course Id

While vector is not empty

If the input is the same as course id

Outputcourse. Course Id < < output course name

while (prereq = true )

output course.prereq

//hash table //

Creating method void print Course Information (hash table < course > courses, string course Id)

Get input for course Id

assign key = course Id

assign node to the node .at (key)

if current node matches key

return course, display course (nodes [ key ]. course )

if node points to null, return null

else while the node is not null, check the key

if key matches the course Id, return course, display course ( nodes [ key ]. course )

point to the next nodes

//tree //

Create method void print course information (tree < course > courses, string course Id )

Get input for course Id

assign current node to root

while current is not null

if course.course Id matches current

return current, output course. Course Id < <

output course.name

while (prerep = true )

output course.prereq

if courselid is less than root

set current to left

else set current to right

Menu

Set choice to 0 ;

Create while loop for menu. While choice is not equal to 4

Output menu choice ( 1. load course files, 2. print course list 3. print Indvidual course 4. exit )

Create switch (choice)

Case 1: load courses ( course file, data structure ). using structures as data structure.chosen

Case 2: print sorted (courses) call function to print sorted class

Case 3 : print Course Information (course Id)

Case 4: Terminate program

Print sorted list:

//vector //

Create sorted print method print sorted (courses )

Create partian method int partition (vector < course > plus courses, int begin, int end,

set low index to first element, set high index to last element

set midpoint to low index + (high index – low index ) /2

set pivot to midpoint

decrement high index while pivot is less than high index

swap lower values to left of pivot, higher values to right of pivot

set temp value to low index

set low index to high index

set high index to temp

Create a quicksort method void quick sort (vector < course > + courses, int begin, int end )

set mid to 0, low index to begin, high index to end

if begin > = end, return

set low end index to partition (courses, low index, high index) ;

Make recursive call to quicksort

quicksort, sort (courses, low index, low End Index) ;

quicksort ( courses, low End Index + 1, high Index )

create display course method void display Course (course course ) {

cout < < course Id < < “ : “ < < course name < < “ l “ < < course prereq < < end1 :

loop through vector to display courses

for ( int i = 0 ; < course .size ( ) ; + + i )

display course (course [ i ] //tree//

Create in order method void BinarySearchTree : : in order (node \* node)

if node = null ;

check most left side first

in order (node -> left )

cout < < course, course Id < < “ : “ < < course.name < < “ l “ < < course.prereq < < end1;

check next right leaf

in order (node –> right)

cout < < course.course Id < < “ : “ < < course.name < < “ l “ < < course.prereq < < end1;

**Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **vector** | **Line cost** | **# times executed** | **Total cost:** |
| **Create vector** | **1** | **1** | **1** |
| **For all line in file** | **1** | **n** | **n** |
| **Create vector course item** | **1** | **n** | **n** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| vector | Line cost | #times executes | Total cost |  |
| Create vector | 1 | 1 | 1 |  |
| While prereqs | 1 | n | n |  |
| Append prereq | 1 | n | n |  |
| Pushback course item | 1 | n | n | Total cost 5n + 1 Runtime. O (n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash table | Line cost | #times executes | Total cost |
| Creating hash table | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| Create key for course | 1 | n | n |
| If no entry found for key | 1 | n | n |
| Assign node to key | 1 | 1 | n |
| else | 1 | n | n |
| Assign old node key to Uni Max | 4 | n | 4n |
| Set to key, set old node to course and old node next to null pointer | 1 | n | n |
| else | 1 | n | n |
| Find the next open node | 1 | n | n |
| Add new Node to end | 1 | n | n |
| For each line in file | 1 | 1 | n |
| Create vector course item | 1 | n | n |
| While prereqs exist | 1 | n | n |
| Append prereqs | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  |  | Total cost: 17n + 1  Runtime: O (n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Line Cost | #Times executes | Total cost |
| Create tree | 1 | 1 | 1 |
| Add node method | 0 | 0 | 0 |
| If root is null, add root | 1 | 1 | 0 |
| If node is less than root then add to left | 1 | n | n |
| If no left node | 1 | n | n |
| this node becomes left | 1 | n | n |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | LIne cost | #Times executes | Total cost: |
| if node is greater than root add right. | 1 | n | n |
| If no right node | 1 | n | n |
| This node becomes right | 1 | 1 | n |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereqs | 1 | n | n |
| Append prereqs | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  | Total cost: 8n + 2  Runtime: O (n) |  |

All data structure has disadvantages and many advantages for all requirements of the given program. The advantage of the vector method is that it is the faster method for reading the file and also adding course objects. This file is parsed and each individual item is appended to the end of a vector. The runtime was the smallest at 5n + 1 through all three methods but they all share the same O (n) notation. A big disanvantage in using the vector method would be searching for a list of the specific course. The specific program needs to check all items in the vector before a match is found.

The main advantages of hash tables would be that they are able to search a list faster. When we create a key locations for a specific course will be known, easilhy search and printed. This would be a slower implemenntation as we create the initial list, and for each item a key has to be created. The table cannot be sorted. In order to print alphanumeric list of each individual course each value must be extracted, printed, and sorted. In conclusion this woud not be the best data structure for the program.

The advantages of Binary trees are they are faster in searching than the vector. If we know the course being selected and search it would be easier to run down the tree until a value is found. It would not be easy like a hash table, but definitely faster than a vector. A disavantage would be the tree would need to look for every element if the tree would end up with left leaves. In conclusion this would allow the search time at O (n) into where h would be the height of the tree. I would strongly advice a vector for this given project. By sorting faster to print the entire catalog would be extremely valuable to the client. The loss of time for searching would not be as bad in regards to utility of the sort.