Jacob Bennett

SNHU

CS300

Milestone 1

1. We will be utilizing ifsteam

Open file – will return -1 if no file is found

While (not end of file)

Read each line

Check if two parameters on each line – throw exception or error

Read parameters

Check if more than three parameters – if more than 3 then continue

Final else that returns error if above two options do not work

Close file.

1. Initialize course object data structure (vector)

Loop through file

While loop

Add values to vectors

If 3rd value present add to vector until we hit the next line

1. Search the data structure

Ask for input(validate user input and throw exception if necessary

Iterate through vector

If input == course

Print out course info

Print out prerequisites

Return

Milestone Two

1. Utilize Ifstream

Initialize call to open file

If( file found)

Read line

If( < 2 values)

Return exception error

Else if <3 values continue with courseNumber (prereq)

Else continue to reach file

Close file

Print courses, prereq’s course number at end of line

1. Create the HashTable file

Initialize course nodes

Initialize ID, title, prereq1, prereq2, prereqCount variables and set them to 0

Loop through file while not end of file

if node at key not found

insert new node at key

else if node pointer->key equals UINT\_MAX

node pointer->key = key

node pointer->next = nullptr

else

while node next pointer is not equal to nullptr

node = node->next

initialize new node and pass vars

1. Create Hashtable print function

For loop where counter is less than tableSize

If node key is not equal to UNINT\_Max

Print

While next pointer is not nullptr

Node is equal to node->next

Print

Milestone three

1. Utilize Ifstream

Initialize call to open file

If( file found)

Read line

If( < 2 values)

Return exception error

Else if <3 values continue with courseNumber (prereq)

Else continue to reach file

Close file

Print courses, prereq’s course number at end of line

1. Initialize struct

Loop through file

For value 1,2

Add ID, and name

For 3rd

Add prereq’s

Create root

Use insert method

If root is null then the current node is root

Else

Course number < root then add to left

Else

If course number < leaf add left

If course number > leaf add right

Course number > root then add to right

Else

If course number < leaf add left

If course number > leaf add right

1. Call print function

While root != nullptr

Traverse left and output

Once reach bottom go back to root

Traverse right and output

Menu Pseudocode

While ( user input choice is not 9)

Print choices

Print 1 load data

Print 2 print course list

Print 4 print individual course

Print 9 exit

Switch case to make selection

Pseudocode to print of list of courses in computer science program in alphanumeric order

Sort through vector

Select pivot to be mid-point

Partition the vector

Recursive algorithm

Base case

recursion

print

for loop while size of courses is less than counter

print courses

**Runtime Analysis**

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector Data Structure** | 1 | 1 | 1 |
| **Reading line in file** | 1 | n | n |
| **Initialize vector object** | 1 | n | n |
| **Append data to vector object** | 1 | n | n |
| **Check and append 3rd prereq** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Hash table Data Structure** | 1 | 1 | 1 |
| **Initialize course nodes** | 1 | n | n |
| **Initialize temp vars** | 1 | 1 | 1 |
| **Loop through file – if node at key not found then insert new node at key** | 1 | n | n |
| **Loop through file – else if node pointer == UINT\_MAX** | 2 | n | 2n |
| **Loop through file – else while next pointer is not equal to null** | 1 | 1 | 1 |
| **Node= node->next** | 1 | n | n |
| **Initialize new node and pass vars** | 1 | N | n |
| **Total Cost** | | | 6n + 3 |
| **Runtime** | | | O(n) |

Binary Search Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Binary Search Tree Data Structure** | 1 | 1 | 1 |
| **Loop through file** | 1 | n | n |
| **Add values 1,2 and prepreqs for 3rd value** | 1 | n | n |
| **Create root** | 1 | 1 | 1 |
| **Insert method** | 1 | n | n |
| **If root is null then cur node is root** | 1 | 1 | 1 |
| **Else – course # less than root then add to left** | 1 | n | n |
| **Else – course # less than leaf then add left** | 1 | n | n |
| **If course # greater than leaf then add right** | 1 | n | n |
| **Course # greater than root then add to right** | 1 | n | n |
| **Else – course # less than leaf add left** | 1 | n | n |
| **course # greater than lead then add right** | 1 | n | n |
| **Total Cost** | | | 9n + 3 |
| **Runtime** | | | O(n) |

Evaluation – Advantages/Disadvantages – Recommendations

The greatest advantage of a vector is the ease of use and quick setup. They also take very little memory, and they can be accessed in constant time. The largest disadvantage to a vector is that when you resize you eat up compute cycles. Hash tables can insert, delete and search in constant time on average. The largest drawback would be collisions, where you have multiple keys mapping to the same memory location. Another disadvantage is that you cannot sort a hash table. Binary search trees have one large advantage over the other two data structures. Sorting complexity is very fast. Binary search trees can also dynamically change size without resizing. One of the largest drawbacks to a binary search tree is the ability to have unbalanced trees, which could degrade performance to O(n).

My recommendation for this specific program would be to use a vector. It has the smallest total cost in the analysis section. It is also the most memory efficient. This recommendation is based solely on the requirements. If we had more data that needed to be searched through transitioning to a binary search tree would be prudent!