Mark Harper

December 8, 2024

Project 1

integer partition (vector, begin, end)

Define integer minimum equal to begin

Define integer maximum equal to end

Define integer pivot equal to begin plus end (end minus begin) divided by two

BOOL done equal false

WHILE not done

WHILE bids title is minimum compare to pivot title if less than 0

Increment minimum element

WHILE bids title is maximum

compare to pivot title if less than 0

Decrement maximum element

IF minimum is greater than or equal to maximum

Done equals true

ELSE

Swap minimum bids title with maximum bids title

Increment minimum

Decrement maximum

Return maximum

Define integer selection sort (vector)

Define integer minimum

Define integer maximum equal to bids size

Define integer place

FOR place equals 0, place is less than bids size, increment place

Minimum equals place

FOR variable j equals place plus 1, less than bids size, increment variable j

IF bids at variable j compare with bids at minimum, if less than 0

Minimum equals variable j

IF minimum not equal to place

Swap bids place for minimum

Define integer ticks equal to clock()

Call function selection sort (bids)

DISPLAY bids size(), bids read

Calculate time

Integer ticks equal to clock() minus ticks

DISPLAY time, ticks and clock ticks

DISPLAY time, ticks times 1.0 and clocks per second

BREAK

Define integer ticks equal to clock()

Call function quicksort (bids at 0, bids size minus 1)

DISPLAY bids size, bids read

DISPLAY time, ticks time 1.0 and clocks per second

BREAK

HashTable::HashTable()

Constructor

Initalize node structure by resizing tableSize

HashTable::HashTable(unsigned int size)

Constructor for tableSize

invoke local tableSize to size with (this->)

resize nodes size

HashTable::~HashTable()

Destructor

erase nodes beginning

unsigned int HashTable::hash(int key)

return key tableSize

void HashTable::Insert(Bid bid)

Create the given bid

retrieve node using key

if no entry found for the key

assign this node to the key position

else

if node is not used

assign old node key to UNIT\_MAX, set to key, set old node to bid and old node next to

null pointer

else

find the next open node

add new Node to end

Print all bids in HashTable

Iterate through each node in nodes vector

If key is not equal to UINT\_MAX

Output key, bidID, title, amount, and fund

Set node to the next iterator

While node is not equal to nullptr

Output key, bidID, title, amount, and fund

Set node to the next node

Remove bid from HashTable

Accept bid ID as input

Calculate hash value for bid ID

Retrieve the list of bids at the calculated hash index

If bid found at the first node

If no next node, set key to UINT\_MAX

Else replace current node with the next node

Else find and delete bid in the list

Search for bid in HashTable

Accept bid ID as input

Calculate hash value for bid ID

Retrieve the list of bids at the calculated hash index

If entry found for the key

Return the bid from the first node

If no entry found for the key

Return an empty bid

While node is not equal to nullptr

If the current node matches, return its bid

Set node to the next node

Return an empty bid

CREATE function for preOrder to pass root variable

CREATE function for inOrder to pass root variable

CREATE function for postOrder to pass root variable

CREATE root variable & set = nullptr

CREATE a left& right Node

CREATE insert bid function

IF (root == nullptr) {

SET root = new Node for bid;

ELSE;

add Node root & bid;

}

CREATE Remove function to take parameters’ root & bidId

CREATE Search function with parameter bidId

SET current node = root;

WHILE (current != nullptr) {

IF current bid is found {

return, current bid;

}

IF (bid < current node) {

transverse to left

}

CREATE addNode function w/ parameters: Node\*node, Bid bid

IF (node is greater) {

THEN add to the left;

}

IF (there is no left node) {

THEN node becomes left;

}

ELSE IF (there is no right node) {

THEN node becomes right;

}

ELSE;

recurse to the left;

}

}

CREATE inOrder function with parameters Node\*node

IF (node != nullptr) {

set inOrder to left;

}

print bidID, title,amt, and fund;

set inOrder to right;

CREATE preOrder function with parameters Node\*node

IF (node != nullptr) {

set preOrder to left;

}

print bidID, title,amt, and fund;

set preOrder to right;

CREATE postOrder function with parameters Node\*node

IF (node != nullptr) {

set postOrder to left;

}

print bidID, title,amt, and fund;

set postOrder to right;

Bid bid;

Return bid;

}

Start program

create two nodes variables for left and right;

create root variable set it to null;

create variables for course name, and an integer for course number;

open file

WHILE file is open

read data

parse each line

- check for course title

- check for course number

IF root != null

check if a prerequisite found

add prerequisite to right node

IF course parameters are less than two

add course to left node

display error msg

ELSE

add course name, num, and perquisite to right node

display Results

close file

**Create course objects and store them in the appropriate data structure.**

Start Program

create variables for course objects;

create root variable and set it to null;

create left and right node variables;

Open file

WHILE file is open

read file parse each line

IF root == null

SEARCH for course

IF course is found

create course object;

ADD course to right node

IF course is not found

print error msg

close file

stop program

print objects

end program

**Print out course information and prerequisites.**

create root variable and set it to null;

create left and right node variables;

open file

WHILE file is open

read file

parse each line

IF root == null

check for course title

check for course number

IF root == null

check if a prerequisite found

add prerequisite to right node

IF course parameters are < two

add course to left node

display error message

ELSE

add course name, num, and perquisite to right node

display msg

close file

end program

**Pseudocode for a Menu**

create an integer for switch statement, set it to 0 name it uInput;

Create a Bid variable to access the

WHILE input does not equal 4;

PRINT 1. Load Data Structure

PRINT 2. Course List(Alphabetical Order)

PRINT 3. Course Title

PRINT 4. Exit

Option 1:

loadBids(bid);

break;

Option 2:

Print: Course List;

break;

Option 3:

Print Course;

break;

Option 9

Print Exit

end program

break;

Default:

PRINT: no input found from user;

break;

Design pseudocode that will print out the list of the courses in the Computer Science program in

alphanumeric order.

Start

Define the list of courses.

Sort the lists of courses in alphanumeric order from lowest to highest

Print the sorted list of courses.

End

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Define integer minimum equal to begin | 1 | 1 | 1 |
| Define integer maximum equal to end | 1 | n | n |
| Define integer pivot equal to begin plus end (end minus begin) divided by two | 1 | n | n |
| BOOL done equal false | 1 | 1 | 1 |
| WHILE not done | 1 | n | n |
| WHILE bids title is minimum compare to pivot title if less than 0 | 1 | n | n |
| Increment minimum  element | 1 | n | n |
|  |  | Total Cost | 5n+2 |
|  |  | Run Time | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Table** | **Line Cost** | **# Time Executes** | **Total Cost** |
| Initalize node structure by resizing tableSize | 1 | 1 | 1 |
| invoke local tableSize to size with (this->) | 0 | 0 | 0 |
| resize nodes size | 1 | n | n |
| erase nodes beginning | 1 | n | n |
| Assign node to key | 1 | n | n |
| Else | 1 | n | n |
| assign old node key to UNIT\_MAX, set to key, set old node to course and old node next to null pointer | 4 | n | 4n |
| Else | 1 | n | n |
| find the next open node | 1 | n | n |
| add new newNode to end | 1 | n | n |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
|  |  | Total Cost | 9n+2 |
|  |  | Run Time | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Tree** | **Line Cost** | **#Time Executes** | **Total Cost** |
| CREATE function for preOrder to pass root variable | **1** | **1** | **1** |
| CREATE function for inOrder to pass root variable | 1 | 1 | 1 |
| CREATE function for postOrder to pass root variable | 0 | 0 | 0 |
| CREATE root variable & set = nullptr | 1 | 1 | 1 |
| CREATE a left& right Node | 1 | n | n |
| CREATE insert bid function | 1 | n | n |
| IF (root == nullptr) { | 1 | n | n |
| ELSE; | 1 | n | n |
| add Node root & bid;  CREATE Remove function to take parameters’ root & bidId | 1 | n | n |
| CREATE Search function with parameter bidId | 1 | n | n |
| SET current node = root; | 1 | n | n |
| WHILE (current != nullptr) { | 1 | n | n |
| IF current bid is found { | 1 | n | n |
| return, current bid; | 1 | n | n |
| IF (bid < current node) | 1 | n | n |
| transverse to left | **1** | **n** | **n** |
|  |  | **Total Cost** | **11n+1** |
|  |  | **Run Time** | **O(n)** |

**Explain the advantages and disadvantages of each structure in your evaluation.**

To calculate the cost per line of code and the number of times each line executes in an algorithm, you need to analyze the code’s time complexity and how operations interact with the data structure containing n courses. I must determine the operation for each line of code. Use the time complexity of the chosen data structure to compute the cost of that operation. Then you have to estimate the number of times a line of code will run based on loops, recursion, or other control structures.

The advantages of a vector data structure are easy implementation with the fastest method of reading a file and adding objects. A vector can store multiple objects and elements can be deleted by a vector. A disadvantage of vectors is that since they are an object, there is more memory consumption. The advantages of a hash table are there is direct access to items. Hash tables can insert and delete in constant time. Hash tables can be the best data structure for speed if implemented correctly. Disadvantages of hash tables are memory consumption; element retrieval does not preserve order. Since we need to list the courses in alphanumerical order, all courses each value must be extracted, sorted, and then printed which would not work for our program.

The advantages of binary search tree are the retrieval of items are in order. Objects can be inserted and deleted in O(logn) time. The access speed is faster and more efficient than the other data structures. A disadvantage of a binary search tree is that it must be balanced. The structure I would use is a binary search tree as binary search tree is more suited to listing course in alphanumerical order. The tree traversal does the sorting for the courses. The binary search tree for searching takes O(logn) time which is sufficient to run the program for the course lists.