Brandon Gallagher

04//06/2023

CS 300 – Project One

Binary Search Tree Psuedocode

INITILIAZE root variable to equal NULL

CALL inOrder function and pass root

POST order the root using BST

PRE order the root using BST

INSERT a bid

IF root is equal to null pointer

Root becomes a new node

FUNCTION to remove a bid

BSTRemove (node, node)

FUNCTION to search for a bid

SET currentNode equal to root

WHILE loop to track if the search hits the bottom or when it finds the matching bid

IF currentBid is equal to bid id

CurrentBid equals bid id

Return currentBid

IF bid is smaller than current bid

TRAVERSE left

ELSE

TRAVERSE right

RETURN bid

INSERT bid function

IF node is larger than current node

ADD node to the left

IF the left node is null

CurrentNode becomes left node

ELSE

Recurse down the left

ELSE

IF right node equals null

Current node equals right

ELSE

Recurse down the right

PRE order function for tree root

POST order function for tree root

Vector pseudocode

SET head node to NULL

SET tail node to NULL

SET newNode equal to new node

IF head is equal to NULL

SET newNode equal to the head and tail

ELSE tail points to the new node

SET tail equal to the new node

INCREMENT size of bid list

SET newNode equal to a new node

IF head is not NULL

SET newNode to point to the current head as its next node

SET head equal to newNode

INCREMENT size

START at the head of printList

WHILE current does not equal NULL

PRINT current bid information

SET current equal to next node to continue printing through list

IF statement when current node is equal to head node

SET head node to point to the next node

DECREMENT size

IF statement when current node is equal to head node

SET head node to point to the next node

DECREMENT size

RETURN the search bidding Id

START at the head of printList

WHILE loop to search through the entire printList

IF statement for when the current node matches the next node

RETURN the next node

ELSE

SET current node equal to next

RETURN the bid

Data Pseudocode

IMPORT data lists to enable loading courses through different forms of lists (vector, hash, and tree)

LOAD course list into desired data structure

PRINT course list alphanumerically

WHILE loop course list until you reach the end of the list

IF previous node < current node

Current node becomes previous node

ELSE current node becomes next node

PRINT course prerequisites and the title of said course

EXIT function to exit the program

Alphanumeric list pseudocode

SORT course list by alphanumeric

WHILE loop to sort list

IF current node is greater than next node

Current node becomes previous node

Next node becomes current node

ELSE

Next node becomes previous node

PRINT new alphanumeric course list to desired data structure EEEE

Data Structure Comparison

Hash Tables -

Hash tables store data using arrays and create key pairs as data. This produces two values, a key and an index. Hash tables are incredibly fast at searching and removing data stored within but run into the issue of duplicate keys and collisions.

Binary Search Trees –

BSTs store data in a tree starting from a root node and branching down. This is the simplest of data structures and is visually pleasing. BSTs are also very fast at searching through the data. Unfortunately removing and changing nodes can be very time consuming and depend on the height of the tree making BST’s good for certain situations.

Vectors -

Vectors are simplistic, like BST’s, and this isn’t a bad thing. Data is stored using pointers linking each node to one another. There are multiple variations of vectors making it very diverse and it is easily accessible in many languages. The only downside vectors have lies in its inability to quickly search for data. Searching has to go through each node and depending on how long the vector is can take some time.

|  |  |  |
| --- | --- | --- |
| Data Structure | Pros | Cons |
| Hash Tables | Fast, easily change and search | Collisions can be an issue if they aren’t thought of |
| BST | Simplistic and quick | Changing any of the nodes and data is very time consuming |
| Vectors | Simplistic and easily usable across multiple languages | Requires running through each node to find data |

Recommendation

I will be using Hash tables as my data structure for my code. Hash tables are still my weakest data structure, but I’m sure given the time hash tables will aid the runtime of my code. Hash tables are easy to use and quick at searching and changing the data stored within. They do have multiple commands required to make the structure work, searching, insertion, and deletion, but this aids in the ease of use of the data structure.