Patrick Valencia

DSA: Analysis and Design

Project 1

//Vector Pseudo Code

//Loads the file

loadBids(string filePath, list) {

cout << "Loading file " << filePath << endl;

// initialize the Parser

Std:: Parser file

// loop to read rows of a CSV file

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

//Print bid information to display in proper format in order

cout << bid.bidId << ": " << bid.title << " | " << bid.fund << " | " << bid.amount << endl;

Catch function to reloop if error is found in parser

//Hashtable Pseudocode

//Load bids from csv

void loadBids(string csvPath, HashTable\* hashTable) {

cout << "Loading CSV file " << csvPath << endl;

// initialize the CSV Parser using the given path

csv::Parser file = csv::Parser(csvPath);

// Create a data structure and add to the collection of bids

Bid bid;

bid.bidId = file[i][1];

bid.title = file[i][0];

bid.fund = file[i][8];

bid.amount = strToDouble(file[i][4], '$');

Catch function to reloop if error is found in parser

Print Course in order within a loop

// Tree PseudoCode

//Load bids from csv

void loadBids(string csvPath, BinarySearchTree\* bst) {

cout << "Loading CSV file " << csvPath << endl;

// initialize the CSV Parser using the given path

csv::Parser file = csv::Parser(csvPath);

// Create a data structure and add to the collection of bids

Bid bid;

bid.bidId = file[i][1];

bid.title = file[i][0];

bid.fund = file[i][8];

bid.amount = strToDouble(file[i][4], '$');

Print Course in order within a loop

Evaluation

When taking a look at the run-time memory there are clear advantages and disadvantages within using these different data structures. If we take into account a line cost of 1 per each line and equate that to time spent executing as some variable for example n, would break down to vectors: 6n, hashtable: 10n, binary-tree: 9n. With this in mind it would appear that overall the hashtable data structure would take more time in executing certain lines of code when compared to either vector or binary-tree data structures. This does make some sense due to how a hash table is created and structured. While it would take time to execute certain commands, hash tables in general have a better caching system in which it requires less memory reads when compared to either vectors or binary-trees. But one downside to this with the caching system in place, they can sometimes reserve too much unnecessary data and in some instances gobble up too much of a system's memory. This is where binary-trees come into play. Binary search trees are incredibly memory efficient due to how they only reserve the memory that they need. When compared to hashtables, binary tree search would only use and store as much information as the user needed without keeping any excess limiting memory usage.

When it comes to vectors this is incredibly important. Since a vector is a more general hash table, it has more general distinct functions when compared. Vectors are great in implementing different types of data structures like queues, stacks, or also hashtables, but its biggest flaw is the amount of time for element retrieval. A vector takes O(n) time when performing insertions because adding an element in the middle of an array requires you to shift all subsequent elements over by one. This is not the case with a hash table as it allows for O(1) insertions time due to the hashtable giving a key to the insertion and then jumping to said key.

Recommendation

Each data structure used within this project has both positive and negative aspects. The only way to fully understand which one is the best for a person is to know what they want to accomplish with their code and then compare to see which one fits their requirements. As in this case, my recommendation would be to use the hashtable data structure. Within all these listings and the way each file is parsed it makes the most sense to use. It has an incredible cache system and adding new structures with its key based design make it incredibly fast and simple. In addition, due to each information having an associated key, data retrieval is incredibly reliable and blazingly fast which would be incredibly beneficial when trying to locate specific information.