7-1 Final Project: Portfolio of Work and Reflection

Christian Rojas

CS-260-T5431

06/23/2021

1. Data Structures:

Definition: A data structure is a format that organizes, stores, and manages a data set. Data structures allow for a system to efficiently retrieve and modify the data within the set. The data structure is comprised of the data values, their functions and relationships.

* Vectors:
  + A vector is the same as arrays – in the sense that they use an allocated memory system to store elements. Vectors are ordered lists of items comprised of a specific data type. Each item in a vector is called an ‘element’. To access a vector, they are placed in contiguous storage to be accessed with iteration.
  + We practiced vectors (and their importance) a lot in lab 2-1. In lab 2-1, we defined a vector data structure and had it hold a set of bids:

written as vector<Bid>bids;

After initializing the vector, I could access the ith element in the list by using the vector as an array.

Example code:

bid.title = file[i][0]

bid.fund=file[i][2]

* Hash Tables:
  + The definition of a hash table is a data structure that stores information that is not yet organized into an array, keeping track of where the data is stored. In a hash table, a ‘key’ is a value that is used to map the index. The ‘key’ is also where pieces of information is stored. Storing the data in a key is very efficient in terms of memory.

For practice, Lab 5-2 focused on hash tables. The programming activity had me implement a hash table. I think a good example would be the code that I implemented to define structures to hold bids:

HashTable\* bidTable;

struct Node {

Bid bid;

unsigned key;

Node\* next;

Node() {

key = UINT\_MAX;

next = nullptr;

}

…

public:

HashTable();

HashTable(unsigned size);

virtual ~HashTable();

void Insert(Bid bid);

void PrintAll();

void Remove(string bidId);

Bid Search(string bidId);

};

* Tree Structures:

The definition of a tree is a data structure that consists of a root, parent, internal node, and a leaf. The root being a tree with no parent. The parent is the node with a child. The internal node contains at least one child, and the leaf is a tree node with no children. Tree structures are truly fantastic, and while they do take some time to implement properly, as long as your machine is able to handle the load on the memory – or better yet, if the trees use cloud storage/high speed SSDs.

In lab 6-2 I worked on implementing tree structures into my program and it gave me a really good look at what these structures are all about. The following example shows the binary search tree tool.

class BinarySearchTree {

private:

Node\* root;

void addNode(Node\* node, Bid bid);

void inOrder(Node\* node);

Node\* removeNode(Node\* node, string bidId);

public:

BinarySearchTree();

virtual ~BinarySearchTree();

void InOrder();

void Insert(Bid bid);

void Remove(string bidId);

Bid Search(string bidId);

};

1. Algorithms

Algorithms are a dream come true. Programs that are able to be given a set of instructions and then follow those instructions autonomously to get from point A to B. Algorithms in simple terms, are sequences to accomplish a task. The following are three algorithms that we went over in class.

* Search

This specific algorithm is used to search for things. Search runs through lists of data to retrieve relevant information. Linear and binary search are the two sub-types that can be used when implementing the search algorithm. The search algorithm can been seen in lab 2-1.

* Sort

Sort is another algorithm that we practiced.