7-1 Final Project

Marcus Bullock

CS-260 Data Structures and Algorithms

Southern New Hampshire University

Contents

[Data Structures 3](#_Toc32821850)

[Vectors 3](#_Toc32821851)

[Code Sample 3](#_Toc32821852)

[Discussion 3](#_Toc32821853)

[Hash Tables 3](#_Toc32821854)

[Code Sample 3](#_Toc32821855)

[Discussion 4](#_Toc32821856)

[Tree 4](#_Toc32821857)

[Code Sample 4](#_Toc32821858)

[Discussion 5](#_Toc32821859)

[Algorithms 5](#_Toc32821860)

[Search 5](#_Toc32821861)

[Code Sample 5](#_Toc32821862)

[Discussion 5](#_Toc32821863)

[Sort 6](#_Toc32821864)

[Code Sample 6](#_Toc32821865)

[Discussion 6](#_Toc32821866)

[Hash/Chaining 6](#_Toc32821867)

[Code Sample 6](#_Toc32821868)

[Discussion 7](#_Toc32821869)

[Modularity 7](#_Toc32821870)

[Lessons Learned 7](#_Toc32821871)

# Data Structures

## Vectors

### Code Sample

vector <Bid> loadBids(string csvPath)

{

vector<Bid> myBids;

// initialize the CSV Parser using the given path

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

// loop to read rows of a CSV file

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

{

Bid bid;

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

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

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

myBids.push\_back(bid);

}

return myBids;

}

### Discussion

This snippet of code is taken from my lab2.cpp program. In this, I’m defining a vector with rows taken with declared variables from myBids. I’m then calling the vector in my for loop to assign data to each row from the csv file. Which was the focus point of this lab, to read contents of a csv file into a vector of bids. Which then will display all loaded bids.

## Hash Tables

### Code Sample

class HashTable

{

private:

struct Node

{

Bid bid;

unsigned key;

Node\* nextNodePtr;

//create node

Node()

{

key = UINT\_MAX;

nextNodePtr = nullptr;

}

// initialize node with bid

Node(Bid myBid) : Node()

{

bid = myBid;

}

Node(Bid myBid, unsigned newKey) : Node(myBid)

{

key = newKey;

}

};

vector<Node> myNode;

unsigned setSize = DEFAULT\_SIZE;

unsigned int hash(int key);

public:

HashTable();

virtual ~HashTable();

void Insert(Bid bid);

void PrintAll();

void Remove(string bidId);

Bid Search(string bidId);

};

### Discussion

This snippet was taken from lab5.cpp, in this I’m utilizing a hash table. For lab 5, we were to use a hash table and a hashing algorithm to create an efficient search algorithm for unique identifiers. My above hash table is storing my bids and identifiers to be called on via the hash function later.

## Tree

### Code Sample

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);

};

### Discussion

In my above code snippet taken from lab6.cpp, I’m creating a search tree, in doing so im creating a hierarchy for the data being stored. We were tasked with creating a requested feature of making an efficient searching algorithm that keeps data sorted for near instant searching. Tree’s and the code above is perfect for that, as it’s a tree data structures main function.

# Algorithms

## Search

### Code Sample

Bid LinkedList::Search(string bidId)

{

node\* temp = head;

node\* holder = new node;

holder->data.bidId = "";

while (temp != nullptr)

{

cout << temp->data.bidId << endl;

if (temp->data.bidId == bidId)

{

return temp->data;

}

cout << "test";

temp = temp->nextNode;

}

return holder->data;

}

### Discussion

In lab3.cpp, I ended up using the search algorithm when learning about linked lists. In the above code, I’m searching for a bidId and returning information once my selected bidId has been found. If it hasn’t, then it loops till my bidId is found.

## Sort

### Code Sample

void quickSort(vector<Bid>& bids, int begin, int end)

{

unsigned int middle = 0;

// stop algorithm running if there are 1 or 0 bids left

if (begin >= end)

{

return;

}

middle = partition(bids, begin, end);

quickSort(bids, begin, middle);

quickSort(bids, middle + 1, end);

}

### Discussion

In lab4.cpp, we used different methods of sorting, Selection sort and quick sort. I decided to choose the quick sort option just because it seemed the most optimal for me. Quick sort is a divide and conquer algorithm. Basically, it will take pivot elements from an array and partition them weather they or less than or greater than the pivot. Selection sorting is quite a bit slower as it repeatedly attempts to find the minimum element.

## Hash/Chaining

### Code Sample

void HashTable::PrintAll()

{

//print all bids to screen

for (unsigned int i = 0; i < myNode.size(); ++i)

{

displayBid(myNode[i].bid);

}

}

void HashTable::Remove(string bidId)

{

// Remove bid

unsigned key = hash(atoi(bidId.c\_str()));

myNode.erase(myNode.begin() + key);

}

### Discussion

Chaining is implemented by creating separate hash functions to implement what you want to do. Above I posted a few examples of the different functions that were created to handle different jobs. One for printing the bid information to the screen, and another for removing bids when it is no longer needed. This was for lab5.cpp when implementing the hash tables and hash algorithms.

# Modularity

My code can be considered modularized simply because you could take my algorithms and use them for other projects. The quicksort from my lab4.cpp that I displayed above would be a perfect example of this. There’s nothing too complex about the code, it’s easily readable and it’s a simple quick search that could be used for basically anything. The same went for the csv parser, we were able to use it for each lab project. Everything stayed the same, and even if we were to change the file it was as simple as changing the csv path. Though none of the labs required us to do that, the same would go for my quick sort. The variables listed may have to be changed a little, but the functionality would work.

# Lessons Learned

Everything in this class was new to me, I’m new to algorithms in general. I learned about several different data structures and how they are important. Different ones are more optimal to use in some circumstances, like hash tables when mapping specific key values, Or trees for a hierarchy of data. Though I did find the tree structures to be quite cumbersome. The algorithms we used was very interesting as well. It was very informative to see how different algorithms effected the speed of data retrieval and learning the most optimal data structure and algorithm to use to retrieve data the fastest. Being a Jr Software Engineer, this knowledge is going to be super beneficial to me. I’ve learned so much that I can apply to my day to day work and am more confident getting into different aspects of the code base. Especially ones that involve encryption, and I may even take a crack at AI in my free time.