Module 4: Hash Table

Jason Kremhelmer

Southern New Hampshire University

CS300: Analysis and Design

David Ostrowski

3/31/2024

# Module 4 HashTable

The code is broken down into functions/structs/classes:

* Class HashTable
  + Class definition of the HashTable objects
    - struct Node
      * Consists of a Bid struct
      * An integer key
      * Pointer to the next node
    - Hash, Constructors/destructors, Insert, PrintAll, Remove, and Search methods
    - Private members: vector node, tableSize
* HashTable()
  + Default constructor that sets the nodes vector’s size
* HashTable::hash(key)
  + The bidID is serving as the key
  + The key value is hashed using a modulo hash
* HashTable::Insert(Bid)
  + A Bid is passed
  + The function then adds the ‘passed’ Bid as a new Node in the hash table (nodes)
  + The new Node is placed at the end of the chained nodes if there is a collision
* HashTable::PrintAll()
  + Function loops through the HashTable starting at the hash table head and outputs, to the console, 4 values from the Bid struct
* HashTable::Remove(String)
  + Function starts at the hashed location and searches for the ‘passed’ String
  + Upon finding the String the Node containing it is freed from memory
  + The chain is updated accordingly if either is the Node containing String
* HashTable::Search(String)
  + Function starts at the head and searches for the ‘passed’ String
  + Upon finding the String the Node containing it is returned
* HashTable::Size()
  + A get method used to access the private member, size
* strToDouble
  + Used to convert the CSV file data into useable value
* Bid
  + Struct containing the data
  + Used with the vector that will be sorted
* displayBid
  + Used to send the values contained in the vector to the console
* loadBids
  + Function used to read in the csv data
  + Can read the csv path in from arguments or use a default path
  + Add parts of the data into the Bid structure and then adds that Bid to the unsorted HashTable
* main
  + *main* is the primary driver for the application
  + *main* has a menu to allow a user to enter a bid, load the data, view the data, and delete a node and then exit the application
  + *main* also reports the timing each algorithm takes to perform the sort using the *time.h* library

With the parser being supplied, the code was fairly straight forward. When adding nodes, I did face some challenges. Still attempting to decipher what happened. The IDE was rejecting the correct syntax until after restart. Once restarted, all worked.

# Pseudocode

**Main** Function()

**Read** cmd arguments

**Store** argument as CSV file path

**If** no cmd arguments load default CSV file path

**Loop** while choice is not equal to ‘9’

**Output** menu

**Get** user input; Store in choice

**Validate** user input

**If** choice is not 1-4 or 9 throw an error

**If** choice equals ‘1’

**Start** the clock and **store** in ticks

**Call** loadBids and store CSV data in HashTable *bidTable*

**Output** number of records in the CSV file

**Stop** the clock

**Output** the elapsed time needed to read in the CSV file

**If** choice equals ‘2’

**Call** PintAll() with *bidTable*

**If** choice equals ‘4’

**Start** the clock and store in ticks

**Call** Search() passing a *bidKey* to search for

**Stop** the clock

**Output** the elapsed time needed to find the *bidKey*

**If** choice equals ‘4’

**Call** Remove() passing *bidKey*

**If** Choice equals ‘9’

**Exit** the application

**Output** ‘Good bye’

**End**

**HashTable::hash(int)**

**Return** *key* modulo *tableSize*

**End**

**HashTable::Insert(*Bid*)**

**Call** hash() with *Bid* member *bidId* and **Store** in *tempKey*

**Check** if the hash location within nodes if the node is empty since start

**If** empty since start **Create** *newNode* with *Bid* and *tempKey*

**Set** hash location equal to *newNode*

**If** not empty since start

**Create** a *Node* pointer and set to the address of the hash table index

**Create** a *newNode* pointer with *Bid* and *tempKey*

**Loop** until the end of the linked chain is found and add *newNode*

**End**

**HashTable::PrintAll()**

**Create** a new Node pointer and **Set** to the address of the nodes beginning

**Loop** through the list; starting at the beginning

**If** key value at iteration is not equal to UINT\_MAX

**Output** to console: *bidId, title, amount, fund*

**End**

**HashTable::Search(*String*)**

**Create** a new *Node* pointer called *cursor*

**Set** *cursor* to the bucket at the hash location

**Loop** until *cursor* is NULL (end of the list)

**If** the Node at cursor contains a *bidId* equal to *String*

**Return** cursor

**Set** *cursor* equal to the next Node

**End**

**HashTable::Remove(String)**

**Create** a new *Node* pointer called *cursor*

**Set** *cursor* to the bucket at the hash location

**Create** a new *Node* pointer called *tempNode*

**Check** if *curor* is pointing at a chain of collisions or a single bucket

**If** a chain

**Check** if *cursor* *Bid* member, *bidId*, matches *String*

**If** match **Set** *tempNode* to the next node

**Set** *cursor* equal to *tempNode*

**Delete** *tempNode*

**If** not a match at start of chain **Scan** the list for a match

**If** a single bucket

**Set** the bucket *Bid* members equal to the default constructor

**End**