4-2 Assignment: Hash Tables

David Vega

Southern New Hampshire University

CS-300 DSA: Analysis and Design

Professor David Ostrowski

July 24, 2022

# Code Reflection

The purpose of this program is to load a CSV file that contains bid information that can be displayed and manipulated by the user. Users can search, remove, and insert bids. The program utilizes a hash table to store bids. This assignment was a little bit different than previous assignments. A hash table stores key/value pairs. In this case, the bid information is the value and the bidID is the key that will be mapped to a list location or “index”. You can then perform various functions using the key to access the information stored in the hash table. The key is “hashed” or “mapped” to an index location using the modulo operator ‘’%” and the table size. The reason for this is the remainder of the calculation is used as the index location to store the bid information. The bidID is also a unique number that will produce a unique index for each element. However, sometimes there can be collisions because the remainder for different hashed keys can be the same. Chaining is a method implemented to reduce collisions because of key/value pairs being assigned to the same location.

START PROGRAM

WHILE choice IS NOT equal to 9

EXECUTE

PRINT user menu:

1. Load Bids
2. Display All Bids
3. Find Bid
4. Remove Bid
5. Insert Bid
6. Exit

READ IN user input and store to variable choice for menu selection.

CHOICE 1 - (Load Bids)

SET variable ticks equal to clock() method. Stores starting clock tick.

INITIALIZE csv file

PARSE csv file and stores data into bid object

DEFINE vector to store collection of bids

RETURN bids

PRINT size of bids

CALCULATE elapsed time

Clock() – ticks

PRINT elapsed time

CHOICE 2 (Display All Bids)

INVOKE PrintAll() // Loops through hash table and prints all elements

BREAK;

CHOICE 3 - (Find Bid)

SET variable ticks equal to clock() method. Stores starting clock tick.

SET bid equal to Search() method and pass in bidkey as a parameter.

INVOKE Search()

Search() will do the following:

* + SET key equal to hashed bidID
  + SET new pointer called “node” equal to reference of nodes element at keyth position.

IF node IS NOT equal to null pointer AND node key IS NOT equal to UINT\_MAX AND node bidID compared to passed in bidID is a match

{

RETURN node bidID;

}

IF node IS EQUAL to null pointer OR node key IS equal to UINT\_MAX

{

RETURN bid;

}

WHILE node IS NOT equal to null pointer

{

IF node key IS NOT equal to UINT\_MAX AND node bidID compare bidID IS a match

{

RETURN node bidID;

}

SET node equal to next node;

}

RETURN bid;

}

CALCULATE elapsed time

IF bidId IS NOT empty

{

INVOKE displayBid() method and pass in bid;

}

ELSE

{

PRINT not found message;

}

PRINT clock ticks

PRINT clock ticks per second

BREAK;

CHOICE 4 - (Remove Bid)

INVOKE Remove() method with passed in bidId;

Remove() method does the following:

SET key equal to hashed bidID

ERASE node starting at the beginning with the hashed key

RESIZE nodes table

CHOICE 5 - (Insert Bid)

CREATE getBid() method as global method

GetBid() method obtains input from user.

SET bid equal to getBid() method

INVOKE Insert() method and pass in bid

{

Insert method does the following:

SET key to hashed bidID

IF the nodes keys at the keyth position IS equal to UINT\_MAX

{

CREATE new node

SET new node equal to Node with passed in bid and key

SET nodes at keyth position equal to new node

}

ELSE

{

CREATE new pointer currNode // to store current node

SET currNode equal to reference nodes at keyth position

CREATE new pointer newNode // to store NEW node

SET newNode equal to a new Node with passed in bid and key

WHILE next currNode IS NOT equal to NULL

{

POINT currNode to the next currNode

}

SET currNode next equal to the newNode

RESIZE nodes table

}

}

INVOKE displayBid() and pass in bid

BREAK;

}

}

ELSE

PRINT “Goodbye”

RETURN 0

}

END PROGRAM