**CS-300 Communication Hash Tables**

Zachary N. Locke

Southern New Hampshire University

Analysis and Design

Jack Lusby

October 15th, 2024

This project focused on implementing a hash table to manage bid information from a CSV file efficiently. The hash table structure enabled fast lookups, insertions, and deletions by using custom hash function and handling collisions with chaining. One of the main challenges was managing collisions in the hash table, which required properly linking bids with the same hash value in a linked list. Additionally, a key focus was parsing and loading large CSV files while ensuring efficient memory management and program performance. To address these challenges, I employed object-oriented design principles and careful debugging.

I optimized the program's performance by tracking the time taken for critical operations, such as loading bids and searching for specific entries. Error handling and testing were integral in ensuring that the program effectively handled edge cases like missing or duplicate bids. Overall, this project strengthened my understanding of hash tables, dynamic memory management, and file handling in C++. It also provided valuable experience in overcoming real-world coding challenges through careful problem-solving and code refinement.

Start HashTable.cpp

Define a structure named Bid:

Attributes:

- bidId: String

- title: String

- fund: String

- amount: Double

Create Function Declarations within the Bid structure:

- Bid(): Default Constructor — Initializes the attributes to default values.

- Bid(id, title, fund, amount): Parameterized Constructor — Accepts values for the attributes and initializes them.

- Bid(other): Copy Constructor — Creates a copy of another Bid object.

- operator=(other): Assignment Operator — Copies values from another Bid object to this object.

End

Start \\LinkedList.hpp Start HashTable.cpp

Define a structure named Bid:

Attributes:

bidId: String — Unique identifier for the bid.

title: String — Title of the bid.

fund: String — Fund associated with the bid.

amount: Double — Amount of the bid.

Function Declarations:

Bid(): Default constructor — Initializes the attributes to default values.

Create a class HashTable:

Attributes:

Node: A private structure that holds a Bid object and a pointer to the next node (for handling collisions).

bid: A Bid object.

key: An unsigned integer used as the hash key.

next: Pointer to the next node (in case of collisions).

vector<Node\*> nodes: A vector to hold pointers to nodes (each bucket).

unsigned int tableSize: Size of the hash table (default is 179 or user-defined).

Function Declarations:

HashTable(): Default constructor — Initializes nodes with a default size of 179.

HashTable(int size): Parameterized constructor — Initializes nodes with a custom size.

~HashTable(): Destructor — Deletes all dynamically allocated memory by traversing and deleting nodes.

unsigned int hash(string key): Hash function — Uses the djb2 hashing algorithm to generate a hash key.

void Insert(Bid bid): Inserts a bid into the hash table. Handles collisions by chaining.

void PrintAll(): Prints all bids stored in the hash table, traversing the linked lists in each bucket.

void Remove(string bidId): Removes a bid by its ID, adjusting pointers in case of collisions.

Bid Search(string bidId): Searches for a bid by its ID and returns the found Bid object. Returns an empty Bid if not found.

Start loadBids()

Function loadBids(string csvPath, HashTable\* hashTable):

Load bids from a CSV file into the hash table.

Open and parse the CSV file.

For each row:

Create a new Bid object.

Insert the bid into the hash table using hashTable->Insert(bid).

Measure and print the time taken to load all bids (clock ticks and seconds).

Start main.cpp

Create a HashTable object using a size specified by the user.

Prompt the user to input a hash table size.

If no size is specified, use the default size.

Set choice = 0 (for menu selection).

While choice != 9:

Display the menu with options:

Enter a Bid.

Load Bids from a CSV file.

Display All Bids.

Find a Bid by ID.

Remove a Bid by ID.

Exit.

Prompt the user to select a choice and read the input.

Switch on choice:

Case 1: Enter a Bid

Prompt user for Bid ID, Title, Fund, Amount.

Create a new Bid object with the entered values.

Call bidTable->Insert(newBid) to add the bid to the hash table.

Display "Bid entered and inserted successfully!".

Case 2: Load Bids from CSV

Prompt the user for the CSV file path.

Call loadBids(csvPath, bidTable) to load bids from the file into the hash table.

Display the number of bids loaded and the time taken.

Case 3: Display All Bids

Call bidTable->PrintAll() to print all bids currently in the hash table.

Case 4: Find Bid by ID

Prompt the user for the Bid ID.

Call bidTable->Search(bidId) to search for the bid.

If found:

Display the bid's details using displayBid(bid).

If not found:

Display "Bid not found".

Display the time taken for the search (clock ticks and seconds).

Case 5: Remove Bid by ID

Prompt the user for the Bid ID to remove.

Call bidTable->Remove(bidId) to remove the bid from the hash table.

Display "Bid removed successfully!" if the bid was found and removed.

If the bid was not found, display "Bid not found. Cannot remove.".

Case 9: Exit

Display "Goodbye!" and exit the program.

Default: Invalid choice

Display "Invalid choice, please try again".

After exiting the loop, clean up memory by deleting the hash table (delete bidTable).

End