Brian Engel

Hash Table Code Reflection and Pseudocode

The code in this project is designed to create a hash table from an imported list of bids. The collisions in the vector are handled with the chaining technique using a linked list. There are also methods for searching the hash table, printing the hash table, and removing a bid from the hash table. Everything was pretty straight forward in this project, except it took me a while to get the pointers and references in the right place.

HashTable() {

Resize the hashtable to tablesize

For (I = 0; i < tablesize; ++I) {

initialize hashtable[I] to new node

}

}

HashTable(unsigned int size) {

Change this->tablesize to size

Resize the hashtable to tablesize

}  
virtual ~HashTable() {

For (I = 0; i < tablesize; ++I) {

set a pointer to this nodes next pointer

while the pointer is not null {

set temp pointer to pointer

set pointer to pointer->next;

delete temp;

}

}

}

unsigned int hash(int key) {

return key % tableSize;

}  
void Insert(Bid bid) {

Set Key to the Hash of the bidID

Set Oldnode to hashtable[key]

If oldnode is UINT\_MAX (unused) {

hashtable[key] = new node(bid, key)

}

Else {

Newnode = new node(bid, key)

while (oldNode->next != nullptr) {

oldNode = oldNode->next;

}

oldNode->next = newNode;

}

}  
void PrintAll() {

For (I = 0; i < tablesize; ++I) {

if hashtable[i].key != UINT\_MAX (a used node) {

Output all necessary information

current = hashtable[i].next

while (current != nullptr) (checking to see if the node is chained) {

Output all necessary information

current = current->next;

}

}

}

}  
void Remove(string bidId) {

Set Key to the Hash of the bidID

Current = hashtable[key]

Temp = null

If current = null or current->key = UINT\_MAX {

return (the node is not in list)

}

if (current->next == null and current->bid.bidId == bidId) {

current->key = UINT\_MAX

Return (node matches and is the only node in location)

}

while current != null and current->bid.bidId != bidId { (move through list until bidId is found or end of list)

temp = current;

current = current->next;

}

if (current == nullptr) {

return (no match for node)

}

if (temp == nullptr) { (node is at head of list)

hashtable[key] = current->next

}

else { (node is in the middle or end of list)

temp->next = current->next;

}

}

Bid Search(string bidId) {

Set Key to the Hash of the bidID

Node = hashtable[key]

if node == null or node->key == UINT\_MAX { (node not in hashtable)

return Bid(); (empty bid)

}

while node != null {

if node->bid.bidId == bidId {

return node->bid;

}

node = node->next;

}

return Bid();

}