1. POTENTIAL BUGS:

The HashTable’s destructor function may not always free the newly allocated m\_buckets of the HashTable, causing the program to occasionally crash. I am not sure why this occurs… However, the rest of the program *should* work normally.

1. BIG-O QUESTIONS:

HashTable(..) constructor runs in O(B) time

~HashTable() runs in O(B+N) time

set(…) runs in O(C/B) time

touch(…) runs in O(C/B) time

discard(…) runs in O(C/B) time

1. PSEUDOCODE:

**HashTable::set(KeyType key, ValueType value, bool permanent)**

{

if HashTable is at capacity

return false;

Find bucket to put KeyType into, p.

If first node in bucket is empty

set p to a newly allocated node

set value of p to value

set key of p to key

set the next pointer of p to nullptr;

increase current size of HashTable by 1;

else if the first node already has the value of key

update p’s value to value

if p is a permanent association

set p to most recently used

return true;

else

repeatedly go through each node in bucket we hit the last node

if node already has the value of key

update p’s value to value

if p is a permanent association

set p to most recently used (sendToBack function)

return true

set p to the next node

set p to a newly allocated node

set p’s value to value

set p’s key to key

set p’s next pointer to nullptr

increase current size of HashTable

if this is a non-permanent association

create a new DLL (DLL is a doubly linked list that remembers which node was most recently used)

set p to remember this DLL

set p’s DLL’s value to value

update p’s DLL to be the most recently used (sendToBack function)

return true;

}

**HashTable::touch(KeyType key, ValueType value)**

{

Find the bucket that corresponds to the given key.

Set p to the first Node in the correct bucket

Repeatedly search through each node in bucket

if the current node has the same key and is not a permanent association

call sendToBack function on Node (defined below)

return true;

set current node to the next node

return false

}

**void HashTable::sendToBack(DLL\* curr, bool isNew)**

{

if curr is nullptr

return

if the list is empty

set curr to be m\_head and m\_tail

return

if curr is the tail

return

if curr is the head

set head to curr’s next

set head’s prev to nullptr

else if the curr being updated is not new to the list

link the two nodes surrounding curr to each other

link curr to the tail, making curr the new tail

return

}

**HashTable::discard(KeyType key, ValueType value)**

{

Set killMe to the tail

if the list is empty

return false

if killMe is also the head

set the head to nullptr

else

make killMe’s previous the new tail

set killMe’s key to key

delete killme

find the bucket where key is in

if the first node in bucket has the key (call this curr)

set value to curr’s value

link bucket to curr’s next

delete curr

decrease the current size of HashTable;

return true

repeatedly search through each node in bucket

if the curr node has the key

break;

set curr to curr’s next

if curr is not empty

Link curr’s previous to curr ‘s next

set value to curr’s value

delete curr

decrease the current size of HashTable

return true;

}