Computer Science Project 4 Report

# Known Bugs and Problem

There are no known bugs or problems in the code. The program can successfully encode messages onto websites, and decode hidden messages from websites.

# Big-O Requirements

*All the methods satisfy the big-O requirements provided in the specification.*

* HashTable(unsigned int numBuckets, unsigned int capacity);

A for loop is present in the constructor to initialize all elements in the array (of pointers to nodes) to null pointers. Hence, the big-O of the function is O(B) where B is the number of elements/buckets in the array.

* ~HashTable();

The destructor destroys every node in each bucket. So the big-O is O(B+N).

* bool set(const KeyType& key, const ValueType& value, bool permanent = false);

The only loops present in this function are while loops that are used to traverse nodes in a particular bucket. So, given that the hash function effectively disperses data, the big-O of this function ultimately depends on the ratio of the capacity to the no. of buckets (as more buckets mean there are less chances of multiple nodes landing in one bucket). Hence the big-O is indeed O(C/B), where C is the capacity.

* bool get(const KeyType& key, ValueType& value) const;

Similar to the set function above, the while loops are only used to traverse nodes in a bucket. Hence the big-O again depends on the ratio of the capacity to the no. of buckets. So, the big-O is O(C/B).

* bool touch(const KeyType& key);

This method also has a big-O that depends on the ratio of the capacity to the no. of buckets. So, the big-O is O(C/B).

* bool discard(KeyType& key, ValueType& value);

Just like all three functions above, this function has also has a big-O of O(C/B).

* static void compress(const std::string& s, std::vector<unsigned short>& numbers);

This function has two for loops. One that runs 256 times, and the other that runs L times, where L is the length of the string being compressed. Hence the time function of this method is about L + 256 + a, where ‘a’ is some constant. Hence the big-O of this method is O(L).

* static bool decompress(const std::vector<unsigned short>& numbers, std::string& s);

This function also has two for loops. One that runs 256 times, and the other that runs V times, where V is the length of the vector which stores the compressed message. Hence the time function of this method is about V + 256 + a, where ‘a’ is some constant. Hence the big-O of this method is O(V).

# Pseudo codes:

## HashTable’s set() method

*if Hash Table is at maximum capacity*

*return false*

*compute the hash and determine the bucket number*

*if key already exists in the Hash Table*

*while the current node does NOT contain the key*

*move to the next node*

*set the value in the current node as the value passed in as the parameter*

*set this node as the most recent*

*else*

*while the next node in sequence is NOT a null pointer*

*move to the next node*

*if this data entry is NOT permanent*

*create a new node and let the current node point to this new node*

*adjust the more recent and less recent pointers in the new node*

*adjust the most recent and least recent pointers to nodes*

*else*

*create a new node and let the current node point to this new node*

*return true*

## HashTable’s touch() method

*compute the hash and determine the bucket number*

*if the node pointed to by the bucket is a null pointer*

*return false*

*while the current node does NOT contain the key*

*move to the next node*

*if this next ‘node’ is a null pointer*

*return false*

*If the current node is NOT a permanent entry*

*adjust the nodes before and after the current node so that they link to each other*

*adjust the least recent pointer if necessary*

*let the most recent pointer point to the current node*

*adjust the current node’s less recent and more recent pointers*

*return true*

*else*

*return false*

## HashTable’s discard() method

*if the least recent pointer is a null pointer*

*return false*

*temporary pointer = the pointer to least recent node*

*let the least recent pointer point to the last but one node (according to recency)*

*compute the hash and determine the bucket that consists the node to be discarded*

*while the next node does NOT point to the node to be discarded*

*move to the next node*

*adjust the nodes before and after the node to be deleted so that they link to each other*

*delete the node pointed to by the temporary pointer*

*return true*