Hermmy Wang

**Project 4 Report**

1. **Bugs**

As far as I know, my program does not have any known bugs and has successfully passed a series of test cases.

1. **Description** of what data structures and algorithms for each of your classes’ non-trivial methods and data structures.
2. **MyHash**
   1. **Data structure:** I used a dynamic array of pointers, with each bucket pointing to a linked list of nodes. Each node consists of a key and a value.
   2. **void reset():** First I iterate through the dynamically allocated array of pointers. For each bucket with a linked list, I visit and delete every node. And then I delete the array itself. Second, I dynamically allocate a new array of pointers for the original array pointer, and initialize it with null pointers. And lastly I reset the number of items and the capacity.
   3. **void****associate(const KeyType& key, const ValueType& value)**

Find the bucket number for the new key

Look at that bucket in the array, if it is a null pointer

Initialize the bucket with the key/value pairs

Increment the number of items

If it is not a null pointer

Iterate through the linked list

If the key is already in the list, update the value

If the key is not found in the list

Initialize a new node and add to the linked list

Increment the number of items

If the load factor exceeds the maximum

Resize the array

* 1. **const ValueType\* find(const KeyType& key) const**: First I find out where the target bucket is. And then I iterate through the linked list and return the key’s value if it is found. Otherwise, return a nullpointer.
  2. **void resizeArray()**

Double the size

Dynamically allocate a bigger array and initialize it with null pointers

For each valid linked list in the original array

For every node in this linked list

Find the new bucket number of the node’s key

If this bucket in the new array has never been used

Initialize it with the node’s key and value

Else

Initialize a new Node with the node’s key and value

Link the new node to the head of the linked list

Delete the node in the original array

Delete the array

Set the original array to the new array.

1. **TokenizerImpl**
   1. **vector<string> tokenize(const string& s) const**:

For every character in the given string,

If it is not a separator

concatenate the character to the word string

Else

Push back the word string for return if it is not empty

Reset the word string

Push back the last word if it is not empty

Return the vector of word strings

* 1. **Data structure:** I used a vector of characters to store the separators in order to use the find function in <algorithm>

1. **WordListImpl**
   1. **vector<string> findCandidates(string cipherWord, string currTranslation)**

Check the length consistency of the cipher word and the current translation

Check if the cipher word or the current translation contain valid characters

Check character consistency of the cipher word with the current translation

Set both string arguments to lower case

Find the vector with the pattern of the cipher word

Store all the words with that pattern into a vector of potential candidates

For each potential candidate

Check length and letter consistency of the potential candidate with the current translation

Push back the real candidate for return

Return the candidates

* 1. **string pattern(const string& s):** Initialize a pattern string of the same length as s with all ‘0’s. Look at each character in s, for every character after s, if they are the same, set the corresponding index of pattern to a letter. Move on to the next letter pattern.
  2. **bool contains(string word) const**: I use the MyHash with all the words as keys to call find() to find the target.
  3. **Data structure:** I used a MyHash for the wordlist. The key is the letter patterns; the value is the vector of words with that letter pattern. I used another MyHash to store all words as keys and their patterns as values (for a faster search).

1. **TranslatorImpl**
   1. **bool pushMapping(string cipherText, string plainText):** Check length consistency. Check validity of every character. Check conflicts in the current mapping table and the new mapping. Push back the current mapping. Update the current mapping.
   2. **bool popMapping():** If the stack has nothing to pop, return false. Restore the top mapping to the current mapping. Pop the top mapping.
   3. **string getTranslation(const string& cipherText) const:** Initialize a result string as the same length of ciphertext with all ‘?’s. For every character in the ciphertext, find the corresponding plain letter from the current mapping table (keeping the case). If the character is not a letter, leave it unchanged.
   4. **Data structure:** I implement a stack using a singly linked list. Each node in the linked list contains a mapping table (2D char array) and a node pointer pointing to the next node. I use a 2D char array to store the current mapping table.
2. **DecrypterImpl**
   1. **Data structure:** I used a WordList pointer to store the wordlist.
   2. I used the algorithm described in the spec for the rest of the class.
3. **Big-O**: Each method satisfies the big-O requirements as far as I know.
4. **MyHash**
   1. MyHash(double maxLoadFactor) and ~MyHash(): O(B)
   2. void reset(): O(B)
   3. void associate(const KeyType& key, const ValueType& value): O(1) or O(B) with resizing
   4. const ValueType\* find(const KeyType& key) const: O(1)
5. **TokenizerImpl**
   1. TokenizerImpl(string separators): O(P)
   2. vector<string> tokenize(const string& s) const: O(SP)
6. **WordListImpl**
   1. bool loadWordList (string filename): O(W)
   2. bool contains(string word) const: O(1)
   3. vector<string> findCandidates(string cipherWord, string plainWord): O(Q) (Q: number of words with that pattern)
7. **TranslatorImpl**
   1. bool pushMapping(string cipherText, string plainText): O(N + 26)
   2. bool popMapping(): O(26)
   3. string getTranslation(const string& cipherText) const: O(N)
8. **DecrypterImpl**
   1. bool load(string filename): O(W)