Bradley Mont

Com Sci 32

Professor Smallberg

Lecture 2, Discussion 2C

UID: 804-993-030

Project 4: Cracked

1. My DecoderImpl::crack() function has bugs. It works for certain (usually smaller ciphertexts), but it usually does not work for larger cipher texts. It always works for single word ciphertexts, but as the ciphertext gets more words, the more likely it is that it won’t work. However, it has detailed comments throughout the function that will allow one to see that I implemented a very large portion of the function. I just failed to fully figure it out.

2.

MyHash

* I chose a dynamic array of Bucket pointers for my hash table. The dynamic array allowed me to access any bucket in O(1) time, and dynamic arrays are good for resizing.
* For associate(), I checked if the key was already in the table, and inserted or updated accordingly. For resizing, I created a temporary hash table twice the size of the current one, copied all the nodes into that one, deleted my old hash table, copied the temp hash table into the real hash table, and then deleted the temp hash table.

Tokenizer

* I used a vector of chars to keep track of the separators. That way, I could easily iterate through the vector to see if a character is a separator.
* For tokenize(), I looped through the string and handled each of 4 situations how they trivially should be handled: valid character at the beginning of a word, valid character in the middle of a word, separator ending a word, and separator following a separator.

Wordlist

* I used a MyHash object that mapped a string (letter pattern) to a vector of strings (all words that have that letter pattern). This allowed me to find a letter pattern in O(1) time and then see all the words with that pattern.
* For findCandidates(), I first found all possible words with the given letter pattern using MyHash. I then iterated through all the words. If the translation had a letter, I checked to see that the letters matched. If the translation had a ‘?’, I checked to see that the current word was a letter. Only if it passed all those for each character did I pass it as a valid translation.

Translator

* I used a MyHash from char to char to represent my mapping table. This allowed me to find the mapping for any letter in O(1) time.
* I also used a vector of maps (mapping chars to chars) to represent the history of pushed mappings. This allowed me to call popBack() whenever I wanted to pop a mapping.
* For pushMapping(), I created a temp mapping to add to just in case I found an inconsistency later in the function. For each character, I checked to see if it already had a conflicted mapping or if multiple letters had that same mapping. In that case, I returned false without making any changes. Plus, in a very unorthodox manner, I used a vector of strings to represent the temp mappings. This ended up being very convenient because I made the 0 position of the string the letter, and then the 1 position of the string what it maps to. I then copied the temp mapping into an actual mapping and pushed that mapping to history.
* For popMapping, I simply got the last value of history, popped it, and then copied that into the current mapping.

Decrypter

* I used a vector of bools initialized to all false to keep track of which words I’ve already chosen.
* I also used a vector of strings to represent my final decoded message(s). This way, I updated it throughout the crack() function, and then returned it at the end.
* Non-trivial parts of crack():
  + For finding out the next word to choose, I chose the one with the most ‘?’s in its translation because that was the most unknown.
  + If the candidates vector was empty, I popped the current mapping and tried another word.
  + For checking for a correct message, I checked to see that every translated word had no ‘?’s and was a valid word.
  + I recursively called the function if I had a mapping that was incomplete, but also had no inconsistencies or errors yet.

3. Big-O Requirements:

MyHash

* MyHash(double maxLoadFactor) – yes
* ~MyHash() - yes
* reset() – yes
* void associate(const KeyType &key, const ValueType &value) - yes
* ValueType\* find(const KeyType& key) - yes
* getNumItems() – yes
* getLoadFactor() - yes

Tokenizer

* Tokenizer(std::string separators) - yes
* tokenize(const std::string &s) - yes

WordList

* loadWordList(std::string filename) - yes
* contains(std::string word) - yes
* findCandidates(std::string cipherWord, std::string currTranslation) - yes

Translator

* pushMapping(std::string ciphertext, std::string plaintext) – yes
* popMapping() - yes
* getTranslation(const std::string& ciphertext) – none given

Decrypter

* load(std::string filename) - yes