CS 214 Assignment 1: Tokenizer

Kevin Sung and Joyce Wang

September 19, 2013

1 Implementation

1.1 struct *TokenizerT_*

The struct contains 5 members:

- *separators*: the string of separators
- *tokens*: the string of tokens
- *firstIndex* and *secondIndex*: indices used to parse tokens
- *arr*: bit array of separator chars

1.2 TokenizerT **TKCreate*

When a *TokenizerT* is created, the input "separator" and "token" strings are copied into *separators* and *respectively*, and *firstIndex* and *secondIndex* are both set to 0. *arr*is set to an array of 256 short ints, where $arr[c] = 1$ if c is a separator character, 0 otherwise. This bit array allows us to check whether a given character is a separator character in constant time.

1.3 char **TKGetNextToken*

First, the indices *first* and *second* are advanced to the next non-separator character. Then, *second* is advanced to the following separator. The size of the token is equal to the number of characters between *first* and *second*, including the character at *first* but not *second.* A string of the correct size is created, and *first* is advanced to *second* while the characters that *first* passes by are added to the string. Once *first* has reached *second*, the string is terminated and returned.

1.4 void *TKDestroy*

This method just frees the memory allocated to each member of the tokenizer, and then the tokenizer itself.

2 Time Complexity

I will analyze the method *TKGetNextToken* and count the number of character comparisons. Each character in the input ”token” string is traversed twice, once by *first* and once by *second.* Each time a character is encountered, it is checked to see whether it is a separator or not. This check is done in constant time due to the bit array of separator characters. In the worst case, the character is a special character and must be resolved, but this is done with no more than a constant number of comparisons (with the list of special characters). Therefore, for each character in token, no more than a constant number of comparisons are performed. The running time does not depend on the length of the ”separator” string due to the use of the bit array, which acts as a hash table. Therefore, the method runs in *O(n)* time, where n is the number of characters in the ”token” string.