University of Waterloo
CS240 Fall 2017
Assignment 5

Written Questions Due Date: Wednesday, November 29, at 5:00pm
Programming Question Due Date: Monday, December 4, at 5:00pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f17/guidelines.pdf` for guidelines on submission. This assignment contains written and programming problems. Submit your written solutions electronically as a PDF with file name a05wp.pdf using MarkUs. We will also accept individual question files named a05q1w.pdf, a05q2w.pdf, a05q3w.pdf, a05q4w.pdf if you wish to submit questions as you complete them.

Problem 5 contains a programming question; submit your solution electronically as a file named `lzcount.cpp`.

Problem 1 Boyer-Moore [4+4+4+4+4=20 marks]

a) Construct the last occurrence function L and suffix skip array S for pattern $P = adobodoa$ where $\Sigma = a, b, c, d, o, t$.

b) Trace the search for P in $T = dotadotadotdotadobodoadot$ using the Boyer-Moore algorithm.

c) For any $m \geq 1$ and any $n \geq m$, give a pattern P and a text T such that the Boyer-Moore algorithm looks at exactly $\lfloor n/m \rfloor$ characters. Justify your answer.

d) For any $m \geq 1$ and any $n \geq m$ that is a multiple of m, give a pattern P and a text T such that the Boyer-Moore algorithm looks at all characters of the text at least once and returns with failure. Justify your answer.

e) A number of heuristics can be used with Boyer-Moore to reduce the number of comparisons performed between P and T. Suppose we use Boyer-Moore with only the Peek heuristic. The Peek heuristic states that if $P[j] \neq T[i]$ and $P[j-1] \neq T[i-1]$ then the next location to search for P at is $T[i+m-1]$. Show that the Peek heuristic may fail to find P in T, i.e., find a pattern P, and a text T containing P, such that Peek fails to find P in T.

Problem 2 Karp-Rabin [4+4=8 marks]

a) Trace the Karp-Rabin pattern matching algorithm when looking for the pattern `123` in `792365740165241069317830123`, where the signature is $h(w) = h(w_1w_2w_3) = w_1 + w_2 + w_3 \bmod 10$. Show each comparison, and indicate how many "false-positives" there are in total (a "false-positive" is where $h(w) = h(w')$ but $w \neq w'$). The table below may be helpful in writing up your solution:

7	9	2	3	6	5	7	4	0	1	6	5	2	4	1	0	6	9	3	1	7	8	3	0	1	2	3

b) Repeat part (a), except use the hash function $h(w) = h(w_1w_2w_3) = 4w_1 + 2w_2 + w_3 \bmod 53$.

Problem 3 Suffix Trie [4+4=8marks]

a) Draw the suffix tree for $T = deacacaeacacaedd$.

b) Trace a search for $P = aca$ in the suffix trie created in the previous part.

Problem 4 Huffman coding [4+4+4=12 marks]

We will define the *weighted path length* (denoted as WPL) of an encoding tree as

$$\mathrm{W}PL(T) = \sum_{c \in T} f(c) \cdot d(c),$$

where $f(c)$ is the frequency of the character c and $d(c)$ is the depth of the character c (i.e., the edge distance from the root of the tree).

Recall the class convention for constructing Huffman trees: To break ties, choose the smallest-alphabetical letter, or tree containing the smallest-alphabetical letter. Also, when combining two trees of different values, place the lower-valued tree on the left.

a) Give the Huffman tree (called T) for the following string: `AAABBCCCD`. Calculate WPL(T).

b) Give another encoding tree T' for `AAABBCCCD` that *cannot* be created by Huffman's algorithm, yet WPL(T) =WPL(T'). Justify why your encoding tree cannot be built by Huffman's algorithm.

c) Suppose c_1 and c_2 are two characters of frequencies f_1 and f_2 in text w, and let d_1 and d_2 be the depths of c_1 and c_2 in a Huffman encoding tree for w. Show that if $f_1 > f_2$, then $d_1 \leq d_2$.

Problem 5 Lempel-Ziv [14 marks]

Implement a program that computes how many codewords the Lempel-Ziv algorithm uses for a given string; i.e. implement the method `int LempelZiv(string s)` which performs the Lempel-Ziv encoding on string s and returns how many codewords were used. (It does not need to return that actual codewords, though you may want to do that for testing purposes.) Your Lempel-Ziv implementation must use a trie (that you implement) for the dictionary of codewords.

Your program will read a string from stdin and output the codeword count by printing it to stdout.

Submit a file `lzcount.cpp`.