University of Waterloo
CS240 Winter 2020
Assignment 5

Due Date: Friday, April 3 at 5:00pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/w20/guidelines.pdf` for guidelines on submission. Submit your written solutions electronically as a PDF with file name a5Submit.pdf using MarkUs. We will also accept individual question files named a5q1.pdf, a5q2.pdf, ... , a5q6.pdf if you wish to submit questions as you complete them.

Problem 1 [4+6 (+6 Bonus) = 16 marks]

a) Draw a range tree that corresponds to the following set of 2D points:

$$S = \{(3,4),(10,2),(9,9),(5,7),(6,1),(0,3),(1,8)\}.$$

Draw the primary tree, and then all the auxiliary trees. Make the primary tree and all the auxiliary trees of the smallest height possible. Make sure to indicate to which node v an auxiliary tree $T(v)$ belongs to.

b) Consider the following variant of two-dimensional range reporting queries. We keep a set of two-dimensional points P in a data structure. All points have positive coordinates. For a query range $Q = [a, b] \times [0, c]$, we must find a lowest point $p \in P \cap Q$ and report it. In other words, we must find a point p such that $p.y$ is as small as possible, and $p.y \leq c$ and $a \leq p.x \leq b$. Here $p.x$ and $p.y$ denote the x- and y-coordinates of a point p. If $P \cap Q$ is empty, report $NULL$.

Describe a data structure that answers queries described above in $O(\log n)$ time and uses space $O(n)$.

c) (Bonus, 6 marks) Let S be a set of 2D points in general position. Design a data structure that needs $O(n)$ space and answers two-dimensional range reporting queries in $O(n^{1/4}\log n + s)$ time. `Hint:` Start by dividing S into equal size subsets S_i for $i = 1, \ldots, m$ and $m = n^{1/4}$ s.t. for any $p \in S_i$ and $q \in S_j$, the x-coordinate of p is smaller than the x-coordinate of q if and only if $i \leq j$.

Problem 2 [5 marks]

Let $M = 7$ be the prime number chosen by Rabin-Karp, $P = 118$ be the pattern to search for, and $h(k) = k \bmod M$. Give a text T, with length 9, that produces the worst-case number of comparisons for Rabin-Karp fingerprinting. Explain why T produces the worst-case.

Problem 3 [4+4+4 = 12 marks]

a) Compute the KMP failure array for the pattern $P = bbbabb$.

b) Show how to search for pattern $P = bbbabb$ in the text $T = bbbcbbacbbbbbabb$ using the KMP algorithm. Indicate in a table such as Table 1 which characters of P were compared with which characters of T, like the example in Module 9. Place each character of P in the column of the compared-to character of T. Put round brackets around characters if an actual comparison was not performed. You may need to add extra rows to the table.

b	b	b	c	b	b	a	c	b	b	b	b	b	a	b	b

Table 1: Table for KMP problem.

c) Given two words w and v, each of length n, explain how to use the KMP algorithm to find if one string is a cyclic shift of another in $O(n)$ time. For example, *alloy* is a cyclic shift of *loyal* but *alloy* is not a cyclic shift of *aloyl*.

Problem 4 [5+3 = 8 marks]

a) Draw the suffix tree for T = *deacacacaeacacacaedd*.

b) Trace a search for $P = acae$ in the suffix tree created in the previous part. To specify the trace, you can highlight the nodes visited in the suffix tree during search.

Problem 5 [4+4+3+4 = 15 marks]

a) [4 marks] Construct the last occurrence function L and suffix skip array S for pattern $P = adobodoa$. Let $\Sigma = a, b, c, d, o, t$.

b) [4 marks] Trace the search for P in $T = dotadotadotdotadobodoadot$ using the Boyer-Moore algorithm. Use the same convention as in lecture slides.

c) [3 marks] Modify the pseudocode for Boyer-Moore algorithm to find all occurrences of P in T. Note that if $T = baboraboraboraba$, $P = borabora$ occurs at index 2 and index 6. A trivial modification would be to start searching for pattern at index $i + 1$ if the previous occurrence was found at index i. Your algorithm should have better efficiency than this trivial modification. Hint: you can use ideas from KMP algorithm.

d) [4 marks] Let $T = abababab....ab$ of length n, i.e. sequence ab repeated $n/2$ times. Find pattern P that results in $\Omega(n^2)$ running time for Boyer-Moore algorithm if we use only the bad character heuristic.

Problem 6 [5+2+5+4 = 16 marks]

a) Construct the Huffman code for the following text $T =$ `bananablues` over alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{e}, \texttt{l}, \texttt{n}, \texttt{s}, \texttt{u}\}$.

Strictly follow these conventions for constructing the code: To break ties, first use the smallest letter (according to the alphabetical order), or the tree containing the smallest alphabetical letter. When combining two trees of different weights, place the lower-weighted tree on the left (corresponding to a `0`-bit). When combining trees of equal weight, place the one containing the smallest letter to the left.

The final code (a table with the codewords for all letters) is sufficient for full marks. You are highly encouraged, though, to provide intermediate steps to get partial marks in the presence of mistakes.

b) Use your Huffman code to decode the following bitstring:

`1000100010010110011110110111111110010`

It is sufficient to list the final decoded string.

(If you get stuck or obtain an unreadable sequence of letters you probably made a mistake in part (a). Make sure to fix it before moving on.)

c) Huffman codes are in general not unique, but we can make them so using tie breaking rules (as ours above). To allow decoding, we have to store the code alongside the compressed file. A simple way would be to store all codewords explicitly, or to store the characters frequencies, but it actually suffices to store the *lengths* of the codewords only (which requires much fewer bits, namely at most $|\Sigma|\lceil\log(|\Sigma|)\rceil$ bits) using so-called *canonical Huffman codes.* These schemes uniquely determine the codewords given the sorted list of codeword lengths. A concise implementation of a canonical Huffman code scheme is given below:

```
canonicalHuffmanCode(symbol, codelen, s):
   // Assumes symbol[i], i=0,...,s-1 has codeword length codelen[i].
   // Assumes codelen is sorted ascendingly.
   int code = 0
```

```
    for i = 0,...,s-1
        print (symbol[i], binaryString(code, codelen[i])
        // binaryString returns code in binary with codelen[i] digits/bits
        if (i == s-1) break
        code = (code + 1) << ( codelen[i+1] - codelen[i] )
        // << is the left-shift operation
```

For example, the codeword lengths $\{1, 3, 3, 3, 3\}$ result in the codewords `0`, `100`, `101`, `110` and `111` being printed.

Compute the canonical Huffman code for the text $T =$ `bananablues` based on the codeword lengths you obtained in (a). When sorting the characters by codeword length, break ties by the alphabetical order of the characters. Also draw the corresponding code trie for the canonical code.

d) Prove that Huffman's algorithm for constructing code tries can in general *not* be used to directly construct canonical Huffman codes(!): To do so, show that the tree corresponding to the codewords of the canonical Huffman code from part (c) *cannot* be the result of an execution of Huffman's tree construction algorithm.

(We therefore have to use a two-stage approach: First, we compute codeword lengths using Huffman's algorithm. Second, we compute the actual codewords to use with `canonicalHuffmanCode`).