University of Waterloo
CS240 Fall 2020
Assignment 5

Due Date: Wednesday, Dec 2 at 5pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every asessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of December 2nd** along with your answers to the assignment; i.e. **read, sign and submit A05-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f20/guidelines.pdf` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a5q1.pdf, a5q2.pdf, ... , a5q5.pdf .

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute. **Remember, late assignments will not be marked.**

Problem 1 Range Trees [3+4+4=11 marks]

a) Draw a 2-dimensional range tree of minimal height for the following set of points:

$$\{(7, 88), (12, 19), (22, 33), (27, 29), (28, 9), (31, 99), (42, 66)\}$$

b) Assume that we have a set of n numbers (not necessarily integers) and we are interested only in the number of points that lie in a range rather than in reporting all of them. Describe how a 1-dimensional range tree (i.e., a balanced BST) can be modified such that a range counting query can be performed in $O(\log n)$ time (independent of s). Briefly justify that your algorithm is within the expected runtime.

c) Next, consider the 2-dimensional case where we have a set of n 2-dimensional points. Given a query rectangle R, we only want to find the number of points inside R, not the points themselves. Explain how to modify the Range Tree data structure discussed in class such that you can answer any of these counting queries in $O((\log n)^2))$ time. Briefly justify that your algorithm is within the expected runtime.

Problem 2 Rabin-Karp [2+5+3=10 marks]

A deranged instructor once posed the following question on a final exam:
Note: this is not the problem you are asked to solve.

Given a text T of length n and an array P of patterns of size ℓ where each $P[i]$, for $0 \leq i < \ell$, is a pattern, i.e. an array of characters, where:

- the length of $P[0]$ is m
- the length of $P[k]$ is $(k+1)m$ for $k \geq 0$
- $P[k-1]$ is a prefix of $P[k]$ for all $k > 0$

You may assume m is much smaller than n.

State how to modify the Rabin-Karp algorithm so that it returns (i, k) where k is maximal such that $P[k]$ is found in T; i.e. $P[k+1]$ is not found in T. The returned i is the position in the text where pattern $P[k]$ is found. For simplicity, you may assume that $\Sigma = \{0, 1, 2, \ldots, 9\}$ and the hash function is the standard one described on Slide 8 of Module 9.

One possible solution is to do the following:

In the pre-processing step, create an array V of hash values for each pattern (of increasing length) where $V[i]$ is the hash value for $P[i]$.

The algorithm then uses a binary search approach on the pattern hash value array to find the longest pattern found in the text. For the initial step, let $m =$ the midpoint of $(0, \ell)$, then the Rabin-Karp algorithm is applied using hash value $V[m]$ and its corresponding pattern. If the pattern is found in the text, then a new pattern of V is chosen at the midpoint of $(m+1, \ell)$; otherwise the pattern of V at the midpoint of $(0, m-1)$ is chosen. The Rabin-Karp algorithm is then applied again with this new pattern. The algorithm continues in this way until the maximal length pattern found in T is determined. You may assume the approriate values are returned (these details are not important for this question).

The questions for you to answer are below.

a) Give an algorithm to efficiently create array V and briefly justify the runtime.

For parts b) and c) below, do not include the pre-processing time from part a).

b) Analyze the algorithm given and give the following runtimes:

- Best-case runtime
- Worst-case runtime

- Best-case expected runtime
- Worst-case expected runtime

Explain how the runtimes were determined.
Note: The runtime function parameters are: n, m and ℓ.

c) Suppose the solution is modified so that if a pattern corresponding to $V[j]$ is found, Rabin-Karp is then executed using $V[j+1]$ and its corresponding pattern. If this pattern is not found, we can conclude that the pattern for $V[j]$ is maximal and terminate. Otherwise, the algorithm continues as originally stated.
Does this change either of the runtimes from part b)? Briefly justify your answer and give the new runtime(s) if they differ from the previous part.

Problem 3 KMP [3+3=6 marks]

a) Compute the failure array for the pattern $P =$`ababac`.

b) Show how to search for pattern $P =$`ababac` in the text $T =$`abcaabaababababacabcaa` using the KMP algorithm. Indicate in a table such as Table 1 which characters of P were compared with which characters of T. Follow the style in the example on slide 16 in Module 9.

- Place each character of P in the column of the compared-to character of T.
- Put brackets around the character if an actual comparison was not performed.
- Use a new row when sliding the pattern.
- You may not need all space in the table.

Problem 4 Boyer-Moore [3+3+3+3+3=15 marks]

a) Construct the last occurrence function L and good suffix array S for pattern $P = ramammam$ where $\Sigma = a, k, m, r$.

c	a	k	m	r
L(c)				

i	0	1	2	3	4	5	6	7
P[i]	r	a	m	a	m	m	a	m
S[i]								

b) Trace the search for P in $T = ramaamamkmamramammam$ using the Boyer-Moore algorithm. Indicate in a table such as Table 2 which characters of P were compared with which characters of T. Follow the example on slides 23-24 in Module 9.

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

Table 1: Table for KMP problem.

- Place each character of P in the column of the compared-to character of T.
- Put brackets around the character if they are known to match from the previous step (similar to the examples in the slides).
- Use a new row when sliding the pattern.
- You may not need all space in the table.

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 $\Theta(n/m)$ 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.

r	a	m	a	a	m	a	m	k	m	a	m	r	a	m	a	m	m	a	m

Table 2: Table for Boyer-Moore problem.

Problem 5 Huffman coding [3+3+3=9 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 $d_1 \leq d_2$.