University of Waterloo
CS240E, Winter 2025
Assignment 5

Due Date: Tuesday, April 1, 2025 at 5pm

Be sure to read the assignment guidelines (`https://student.cs.uwaterloo.ca/~cs240e/w25/assignments.phtml#guidelines`). Submit your solutions electronically to Crowdmark.

Grace period: submissions made before 11:59PM on Apr. 1 will be accepted without penalty. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [3+3=6 marks]

We are searching for pattern P in text T where $|T| = n$, $|P| = m$, and $n \geq m \geq 1$.

a) Consider pattern $P = 0^m$ and let text T be a string of $n \geq m$ bits that were randomly chosen to be 0 or 1 with equal probability. Let X be the number of checks done by Boyer-Moore until it mismatches for the first time or returns with success. (The check that leads to a mismatch is included in this count.) Show that $E[X] \leq 2$.

b) Consider the same setup as in the previous part. Assume that you just had a mismatch. Show that the expected amount by which you shift the guess forward is at least $m - 1$.

Motivation: For the special string $P = 0^m$, the expected number of checks is hence $\approx 2\frac{n}{m-1}$, because you expect to do at most 2 checks until a mismatch and then shift forward by $m - 1$ characters. We clearly need to check at least $\approx \frac{n}{m}$ characters, otherwise we might miss an occurrence of P. Therefore Boyer-Moore is within a factor of 2 of the minimum number of checks, at least for this special string on randomly chosen texts.

Question 2 [4 marks]

Let T be a text of length n. Recall that the suffix tree of T has $O(n)$ nodes and height $O(n)$, and the trie of suffixes of T has $O(n^2)$ nodes and height $O(n)$.

Show that these bounds are tight for some text T, even if the alphabet is small. To do so, give (for all n that are divisible as needed) a bitstring T of length n such that its trie of suffixes has $\Omega(n^2)$ nodes and its suffix tree has height $\Omega(n)$. Justify your answer by explaining the structure of both tries.

Question 3 [2+2+6=10 marks]

a) (Warm-up.) Consider the text $A\,C\,A\,G\,A\,T\,A\,T\,A\,C\,A\,C\,A\,A\,C\,G$ over alphabet $\Sigma = \{A, C, G, T\}$.

What is the cost of the corresponding Huffman-encoding? Show how you obtained your answer, and also write the length of the code-word for each character.

b) Given some probabilities $p_1, \ldots, p_s$ (with $0 < p_i < 1$ and $\sum_{i=1}^{s} p_i = 1$), the *entropy* is defined to be

$$H(p_1, \ldots, p_s) = -\sum_{i=1}^{s} p_i \log_2(p_i).$$

For a text S over alphabet $\Sigma = \{x_1, \ldots, x_s\}$, we define the entropy $H(S)$ to be

$$|S| \cdot H(p_1, \ldots, p_s),$$

where $p_i = \frac{1}{|S|}$(frequency of x_i) is used as "probability" of character x_i for $i = 1, \ldots, s$.

Compute $H(S)$ for the text from part (a). Show how you obtained the answer (in particular, list the probabilities).

c) Let S be a text such that the length of S and the frequency of the characters $x_1, \ldots, x_k$ in S are powers of 2. In other words, there exist integers $\ell_0, \ldots, \ell_k$ such that $|S| = 2^{\ell_0}$ and x_i has frequency $f_i = 2^{\ell_i}$ for $i = 1, \ldots, k$. Show that the Huffman-encoding of S has cost $H(S)$.

Hint: The text from (a) satisfies the assumption. Study its Huffman-trie: what can you say about the length of the encoding of x_i? (For ease of description, assume that the naming is such that $\ell_1 \geq \cdots \geq \ell_k$.)

Motivation: Based on Shannon's information-theoretic lower bound, one can argue that *any* encoding of S as bitstring (whether obtained via prefix-free binary encoding or otherwise) has length at least $H(S)$. So in the special case where the frequencies are powers of 2, Huffman-encoding gives the minimum-length encoding that is possible.

Detour: Set-up for Questions 4-6

The next three questions all have the same motivation and setup, explained here. They are otherwise entirely unrelated, so you should try each of them even if you do not know how to solve the others.

Professor E. Responsible converted a text T into a code-text C with some method. The professor now wants to do pattern matching on the text T with a pattern P, but they have accidentally run the command `sudo rm -rf /*` and deleted their operating system. This means that the text T is lost, but fortunately they still have the code-text C saved on a backup device. They also have the suffix-array of C, so they can efficiently search for patterns in C. But does this help for doing pattern matching in T?

Formally, you have a code-text C (obtained with the method specified in each question below). C is given to you as an array $C[0..N-1]$, and as usual there is an end-sentinel at $C[N]$. You also have the suffix-array $S[0..N]$ of C. You are given a pattern $P = P[0..m-1]$ over alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Also assume that $m > 1$, so P is not just a single character.

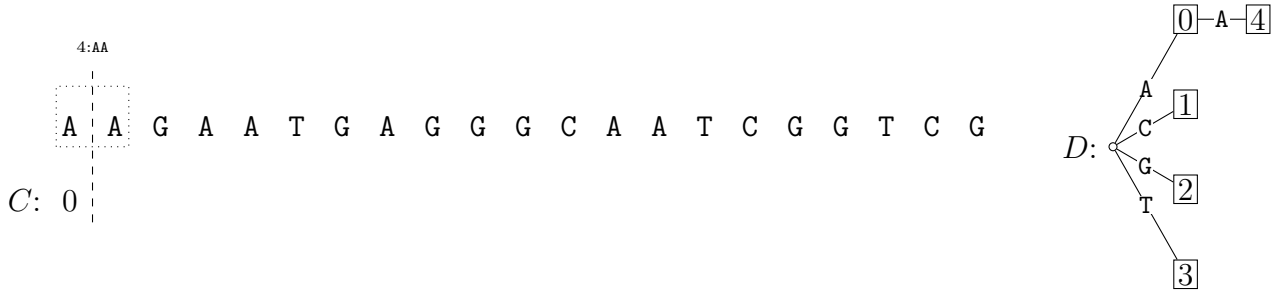
Question 4 [3 marks]

This question has been removed from the assignment.

Question 5 [2+2+3+2+2+5+2=18 marks]

In class we did Lempel-Ziv-Welch compression with alphabet ASCII and stopped adding to the dictionary once code-number 4095 has been assigned. For this question, use Lempel-Ziv-Welch where the initial dictionary only stores $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ and we stop adding to the dictionary once code-number 15 has been assigned.

a) Show the result of Lempel-Ziv-Welch compression applied to the string T given below. You should both show the output-string C (as a list of integers in $\Sigma_C := \{0, \ldots, 15\}$) and the dictionary D at the end. The first step was done for you already.



b) Consider the setup of the detour. Assume that code-text C was obtained from T using Lempel-Ziv-Welch compression. Give an algorithm to test whether P exists in the (unknown) T, with run-time $O(m \log N)$.

The following steps guide you towards the algorithm.[1] You can use steps for later part-questions even if you did not solve them.

i) The LZW encoding algorithm can be viewed as having two phases. In the first phase, we still add to dictionary D, while in the second phase dictionary D no longer changed. In particular, we can write $T = T_1 \frown T_2$ where T_i is the part of T that was encoded during Phase i, for $i = 1, 2$.

Show how compute T_1 in constant time. If $T_1 \neq T$ (i.e., LZW encoding reached Phase 2), then your algorithm should also return the encoding dictionary D as it was at the end of Phase 1.

[1] If you can find algorithm that solves the problem, but does not follow these steps, then this is also an acceptable solution (and will be graded out of 16 marks). But **state clearly** if that is what you are doing.

ii) Show how to test in $O(m)$ time whether there is an occurrence of P in T that overlaps at least one character of the string T_1 that was encoded in Phase 1.

iii) Assume that P has no occurrence that overlaps a character of T_1. Let $\mathcal{W}$ be the set of words stored by dictionary D. Show that P is *not* a substring of a word in $\mathcal{W}$.

iv) Assume that P occurs in T, but has no occurrence that overlaps T_1 and P is not a substring of a word in $\mathcal{W}$. Then P must be *split o across code-numbers*, i.e., we can write $P_1 \frown \ldots \frown P_k$ for some $k \geq 2$ such that P_1 is a suffix of a word $w_1 \in \mathcal{W}$, P_k is a prefix of a word in $w_k \in \mathcal{W}$, and $P_i = w_i$ for some word $w_i \in \mathcal{W}$ for $i = 2, \ldots, k-1$. (You need not prove this.)

Show how to compute in constant time all pairs (w_1, P_1) that could possibly fit such a split of P.

v) Show that there are $O(1)$ many tuples $(P_1, \ldots, P_k, w_1, \ldots, w_k)$ that could fit a split of P, and give an algorithm to compute them all in $O(m)$ time.

vi) Explain how to combine the above steps to test whether P occurs in the (unknown) T in $O(m \log N)$ time.

Question 6 [2+2+5=9 marks]

Assume that code-text C was obtained doing the Burrows-Wheeler transform. In particular, one of the characters in $C[0..N-1]$ is the end-sentinel of T (we use `#` for this), while $C[N] =$`$` (the end-sentinel of C). The alphabet-order is `$` < `#` < `A` < `C` < `G` < `T`.

a) Consider $C =$`AT#CGGAA$` (so $N = 8$). Show the suffix array S of C and the *auxiliary array* A that we create as part of BWT decoding and that stores character-index pairs. No justification needed.

Recall that each entry in S is normally just an integer k, but you may find it helpful for later parts to write, with each entry $S[i]$, also the corresponding suffix of C, or at least its leftmost character.

b) Show that for all $0 \leq i < N$, the character stored at $A[i]$ equals the character $C[S[i+1]]$.

c) Assume that (in addition to C and S) you have access to the auxiliary A of C. Give an algorithm to test whether P exists in the (unknown) T that has run-time $O(m \log N)$.