University of Waterloo
CS240E, Winter 2021
Assignment 5

Due Date: Wednesday, Apr 7, 2021 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 assessment in this course you must read and sign an Academic Integrity Declaration (AID) before you start working on the assessment and submit it **before the deadline of April 7th** 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/~cs240e/w21/guidelines/guidelines.pdf` for guidelines on submission. **Each written question solution 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 but can be submitted to MarkUs after the deadline for feedback if you email cs240e@uwaterloo.ca and let the ISAs know to look for it.**

1. (4 marks)

   We have shown that the Knuth-Morris-Pratt algorithm (Algorithm 9.5) runs in $O(m+n)$ time. This is non-trivial since the while-loop may execute repeatedly without increasing i. Show an example where this happens. More precisely, give a text T and a pattern P and argue that when searching for P in T, there exists some value of i for which line 5 (of Algorithm 9.5) is reached $\Omega(m)$ times without changing the value of i. Your construction should be in terms of $m = |P|$ and $n = |T|$, and work for any sufficienly large values m, n with $m < n$.

2. (3+3+3=9 marks) We are searching for pattern P in text T where $|T| = n$, $|P| = m$, and $n \geq m \geq 1$.

   (a) Show that any pattern matching algorithm must look at $\geq \lfloor n/m \rfloor$ characters of T in the worst case.

   (b) 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 comparisons done by Boyer-Moore until it mismatches for the first time or returns with success. Show that $E[X] \leq 2$.

(c) Consider the same setup as in the previous part. Assume you just had a mismatch. Show that 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 comparisons is hence $\approx 2\frac{n}{m-1}$ (i.e., roughly within a factor 2 of the lower bound) because you do 2 comparisons until a mismatch and then shift forward by $m-1$ characters.

3. (13 marks) Assume that you have bitstrings and you *know* that all runs in each bitstring have odd length. Give a lossless compression scheme for such bitstrings that always achieves compression ratio smaller than 1.

   Formally, if the source text S has length n, and consists of runs of length $k_1, k_2, \ldots, k_d$ for some $d \geq 1$, then all k_i's are odd. Explain how to find an encoding C of S that has length *less than* n and show that you can uniquely recover S, given C. You do not need to discuss the run-time of encoding/decoding (but it must be doable in polynomial time).

   You may assume that n is sufficiently large. You should make no assumptions about $k_1, \ldots, k_d$ (other than that they are odd), but part-marks may be given if you do.

4. (2+4+2=8 marks) This question concerns Lempel-Ziv encoding of the word A^n, which is the word consisting of n copies of the character A. In the following, use as alphabet the 128 ASCII characters, stored with code-words 0 up to 127. In particular, 'A' has code-word 65, and the first code-word you can use for strings added to the dictionary is 128.

   (a) Give the encoding (as list of numbers, *not* as a bit-string) of A^{16}. Show your work.

   (b) Recall that traditional Lempel-Ziv-Welch converts integers into 12-bit strings. This means that when we add codeword 4096 to the dictionary, this would result in an overflow-error.

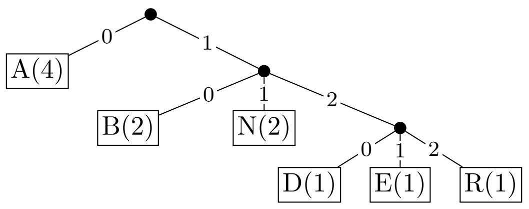
   When encoding A^n, what is the smallest n for which we get this overflow-error? Justify your answer theoretically (i.e., the answer "I implemented LZW and it used code 4096 at $n = X$" will *not* give you credit.)

   (c) Let X be the answer that you got in part (b). Prove that for *any* ASCII-string of length X or more, using Lempel-Ziv-Welch leads to a dictionary-overflow.

5. (2(+5)+18=20(+5) marks) Recall that the Huffman-trie uses a *binary* encoding, i.e., the encoding-alphabet is $\Sigma_C = \{0, 1\}$. But we can easily generalize it to a d-way encoding where $\Sigma_C = \{0, \ldots, d-1\}$ for some integer $d \geq 2$. One would build the trie for this with a natural greedy-algorithm shown in Algorithm 1.

   The *d-way Huffman-encoding* of a text S is obtained by building this trie T (using the frequencies in S) and then encoding as with any prefix-free code. The *cost* of this encoding is $(\log d) \sum_{c \in S} f(c) \cdot d_T(c)$, where $f(\cdot)$ and $d_T(\cdot)$ denote the frequency and the depth as before, and the log-factor is added to compensate for the larger coding-alphabet. For example, text `BANANABREAD` could have the following encoding-trie:

```
Algorithm 1: dWayHuffman::buildTrie($\Sigma_S, f, d$)
1 $Q \leftarrow$ min-oriented priority queue
2 forall $c \in \Sigma_S$ with $f[c] > 0$ do
3     $Q.insert$(single-node trie for $c$ with weight $f[c]$)
4 while $Q.size > 1$ do                                   // build encoding trie
5     $k \leftarrow \min\{d, Q.size\}$
6     for $i = 1, \ldots, k$ do $T_i \leftarrow Q.deleteMin()$, $f_i \leftarrow$ weight of $T_i$
7     $T \leftarrow d$-way trie with $T_1, \ldots, T_k$ as sub-tries and weight $f_1 + \ldots + f_k$
8     $Q.insert(T)$
9 return $Q.deleteMin()$
```



This gives encoding 100110110101221210120 with length 21 and cost $21 \log 3 \approx 33.28$.

(a) Find a text S such that the cost of the 3-way Huffman encoding of S is *better* than the cost of the 2-way Huffman encoding (i.e., the standard Huffman-encoding from class). Justify your answer by showing intermediate steps of how you obtained the costs for your chosen string.

(b) (Bonus) Assume you have a text S of length n that uses characters $\Sigma = \{c_1, \ldots, c_s\}$, and character c_i has frequency $f_i = n/d^{\ell_i}$ times, for some integer $\ell_i > 0$. (A different way to think of it is that each character c_i has probability $p_i = 1/d^{\ell_i}$.) Show that the d-way Huffman encoding has cost

$$n \sum_{1 \le i \le s} p_i(-\log p_i).$$

Motivation: The strange sum is called the *entropy* of these probabilities, and the bound can be shown to be a lower-bound for *any* encoding of texts that have these frequencies.

(c) (Programming, due April 14) Implement d-way encoding and experiment with resulting ~~compression ratios~~ costs. Specifications are given below.

Programming assignment details

- The provided stub `HuffmanEncoding.cpp` contains methods that you must implement without changing the signatures:

  - `dWayEncoding(S,d)` returns the encoding of text S with d-way Huffman encoding. Here S has ASCII-characters between 32 and 126, and d is an integer with $2 \leq d \leq 10$. You may assume that S has at least two distinct characters. You can break ties arbitrarily, but your encoding *must* be the ~~the shortest-possible d-way encoding for text S (which is obtained using Algorithm 1)~~ no longer than the d-way encoding you would get from Algorithm 1. You should not assign code-words to characters that do not occur in S.
  - `getCodeWord(c)` returns the code-word of one ASCII-character c (between 32 and 126). This method will only be called if `dWayEncoding` has been called previously, and should return the same code-word for c as has been used during that encoding.

  Submit an updated file `HuffmanEncoding.cpp`. If you need additional classes, then (contrary to good programming practices) include them in this file as well.

- Run experiments on some texts and compare the costs of doing d-way encoding (for d between 3 and 10) to the cost of doing 2-way encoding. Submit a file *experiments.pdf* where you report average ratios of these costs, and explain briefly what texts you used.

  You should *not* be using randomly generated text; it does not compress well since the character-frequencies would likely all be the same. Instead, use some human-generated texts (e.g. English or some programming or markup code). You are specifically allowed to download such texts from the internet, but acknowledge your sources. Also, make sure that your texts are long enough that the character-frequencies are uneven.

- You may use the following `std` classes/methods: `min, pair, list, stack, queue, priority_queue, vector` as well as the ones used in the stub. No other external libraries are allowed in your submitted code.