Homework #5

(3 points) Implement FM-index search using a wavelet tree with ordinary fixed-length and Huffman encoding. Let us recall that the FM-index consists of

- $L = bwt(T)$, i.e. the last column in the matrix of all rotations of T
- F – the first column; $F[c]$ = position where the lines starting with c begin
- a data structure that supports computing $\texttt{rank}_c$ over L
- SA – sampled suffix array of T.

Download the input file `https://people.ksp.sk/~kuko/ds/du/bwt/SH.txt` – we will do the entire homework with this example.

Since implementation of the entire FM-index would be a relatively large project, we will simplify it a bit:

We have already implemented the suffix array / BWT in the previous homework. You can use your own solution or one of these programs:

`https://people.ksp.sk/~kuko/ds/src/bwt/`

To achieve small memory footprint, FM-indices store just a small sample of the suffix array – for simplicity, we'll keep the whole thing.

Since the text contains < 32 different symbols, by fixed-length encoding, we mean any encoding where each symbol is encoded by exactly 5 bits, e.g., `\0` $\to$ 00000, ␣ $\to$ 00001, `a` $\to$ 00010, `b` $\to$ 00011, ...

Count all the symbol frequencies and construct their Huffman code. The first column F is simply an array of prefix sums of these frequencies.

For an efficient *binary* `rank`, you can use an existing library implementation – I recommend, for example, `https://github.com/simongog/sdsl-lite`, you can even compare different implementations: `rank_support_v`, `rank_support_v5`, `rank_support_rrr`, `rank_support_hyb`,...

Once we have the BWT, the chosen encoding (fixed-length or Huffman) and the chosen implementation of the rank, we can create a wavelet tree from the BWT text and implement the function $\texttt{rank}_c(L, i)$.

Finally, implement search: if we are searching for the string $P = p_0 \dots p_{m-1}$, we go backwards, from the last character, and maintain an interval of lines $[s_i, e_i)$ that start with $P_{i \dots m-1}$.

- Start – lines starting with symbol p_{m-1} are:
  $[s_{m-1}, e_{m-1}) \leftarrow [F[p_{m-1}], F[p_{m-1} + 1])$
- Step – going from $[s_{i+1}, e_{i+1})$ to $[s_i, e_i)$:
  $[s_i, e_i) \leftarrow [F[p_i] + \texttt{rank}_{p_i}(L, s_{i+1}), F[p_i] + \texttt{rank}_{p_i}(L, e_{i+1}))$
- End – we get the interval $[s_0, e_0)$ of rows starting with P
- Using the suffix array, we convert these indices to SA into positions in the text T (test that your program indeed found occurrences of P)

Submit:

- Huffman code and fixed-length code – i.e. write down how exactly you encoded each symbol
- what is the memory footprint (in bytes) of the wavelet tree with the two encodings
- how long does the search take (measure this for example for random strings or random substrings of T of length 10, 20, 30, ..., 100)