- This problemset has *three* questions.
- For other questions, either send the solutions to `gawry1+aos@gmail.com`, or leave them in the envelope attached to the doors of my office (room 321).

1. You are given a LZW parse (so, a sequence of n blocks, each block being either a single letter, or a previously defined block concatenated with a single character). Construct a small structure allowing you to access any letter of the corresponding text efficiently. For full credit, construct a structure of size $\mathcal{O}(n)$ allowing answering any such query in $\mathcal{O}(\log n)$ time, for partial credit the size of your structure can be larger, but should be $o(n^2)$.

   Solution: The structure should consist of two parts. The first part allows to locate the block a given position belongs to, and the second allows to extract the k-th letter of any block. For the first part, just store a sorted list of starting positions of all blocks, and use binary search to locate in $\mathcal{O}(\log n)$ the corresponding block. For the second part, you really need to solve the following problem: preprocess a tree of size n so that the k-th ancestor of any node can be extracted in $\mathcal{O}(\log n)$ time. For this you can, for example, use the power-of-two trick, and store at each node a pointer to the 2^0-th, 2^1-th,... $2^{\log n}$-th ancestor. Then the space is $\mathcal{O}(n \log n)$ and any ancestor can be accessed by following $\log n$ links.

2. Prove that for **any** text of length n over a fixed finite alphabet Σ, its LZW parse consists of at most $\mathcal{O}(\frac{n}{\log n})$ blocks (the constant hidden under the big-O depends on $|\Sigma|$, though).

   Solution: Every block in the LZW parse is unique. We split them into short, meaning of length $\leq t$, and long, meaning of length $> t$. Because no two blocks can be the same, there can be at most $|\Sigma| + |\Sigma|^2 + ... + |\Sigma|^t = \Sigma\frac{|\Sigma|^t - 1}{|\Sigma| - 1} \leq 2|\Sigma|^t$ short blocks. We choose $t = \frac{1}{2}\frac{\log n}{\log |\Sigma|}$ so that the number of short blocks is at most $n^{0.5}$. The number of long blocks is clearly at most $\frac{n}{t}$, so the total number of blocks is bounded by $n^{0.5} + 2\frac{n \log |\Sigma|}{\log n} = \mathcal{O}(\frac{n}{\log n})$.

3. A primitive square is a word of the form xx with x being primitive, which means that it is not possible to write $x = y^k$ with $k > 1$. For instance abaaba is a primitive square, but abababab is not. We want to get a good bound on the maximal number of subwords of a word of length n that are primitive squares. Note that if the same primitive square occurs multiple times in the word, we count it multiple times.

   (a) Prove a simplified version of the "three squares" lemma: if there are three primitive words x, y, z such that yy is a proper prefix of xx, and zz is a proper prefix of yy, then $|x| \geq 2|z|$. Hint: assume that $|x| < 2|z|$, draw a picture, then try to apply the periodicity lemma to deduce that z is actually not primitive.

Solution: We have that zz is a prefix of yy and yy is a prefix of xx. Because z is a prefix of both y and x, so it's a prefix of the second y in yy, and the second x in xx, we get that $|x|-|y|$ and $|y|-|z|$ are both periods of z. Now $|x|-|y|+|y|-|z| = |x|-|z| < |z|$ from the assumption, so $\gcd(|x|-|y|, |y|-|z|)$ is a period of z, too. Let's denote the period of z by d and write $z = r^{\alpha}r'$, where $|r| = d$, $\alpha \leq 1$, and $|r'| < d$, where r' is a prefix of r. Remember that $d \leq \gcd(|x|-|y|, |y|-|z|)$.

We want to deduce that actually $r' = \epsilon$ and $\alpha > 1$, because it will contradict the assumption that z is primitive. Observe that if we can show that $r' = \epsilon$, then it cannot happen that $\alpha = 1$ because it would imply $d = |z|$ and $d \leq \gcd(|x|-|y|, |y|-|z|) \leq |y|-|z| < |x|-|z| < |z|$. So, we only have to deduce that $r' = \epsilon$.

Because d divides $|y|-|z|$, zr is a prefix of y. So, z is a prefix of the second x in xx, and zr is a prefix of the second y in yy. So we have zr aligned with z with an offset of $|x|-|y|$ (z starts at the $(|x|-|y|)$-th character of zr). This offset is a multiple of d, say βd with $\beta \geq 1$, and furthermore it's less than z, because otherwise $|z| \leq |x|-|y|$ so $|x| \geq |z|+|y| > 2|z|$. Hence $r^{\alpha-\beta}r'r$ is a prefix of r^{α}. So $r'r$ is a prefix of $r^{\beta}r'$, which means that $r'r = rr'$. But then if $|r'| > 0$ then $|r'|$ is a period of z, too, so $\gcd(d, r') < d$ is a period of r, so d is not the shortest period.

(b) Prove the following "three squares" lemma: if there are three primitive words x, y, z such that yy is a proper prefix of xx, and zz is a proper prefix of yy, then $|x| \geq |y| + |z|$.

Hint: this is tricky and for extra credit.

Solution: `http://igm.univ-mlv.fr/~mac/Articles-PDF/CR95algo-squares.pdf`, see Lemma 10.

(c) Use the above lemma (in either version) to bound the number of primitive squares that all begin at the same position. Observe that multiplying this bound by n gives you a bound on the total number of primitive squares.

Solution: Let the primitive squares starting at a fixed position be of lengths $\ell_1 < \ell_2 < \ldots < \ell_k$. The simplified version gives us that $\ell_{i+2} \geq 2\ell_i$ for all $i = 1, 2, \ldots, k-2$. Hence $\ell_k \geq 2^{\lfloor (k-1)/2 \rfloor}\ell_1 \geq 2^{\lfloor (k-1)/2 \rfloor}$. But $\ell_k \leq n$, so $n \geq 2^{\lfloor (k-1)/2 \rfloor}$, and $k \leq 1 + 2\log n$. So, there are $\mathcal{O}(\log n)$ primitive squares starting at any position, and there are n positions, so the total number of primitive squares is $\mathcal{O}(n \log n)$. The stronger version of the three squares observation gives the same (asymptotically) bound.

(SPOJ) 4. Given a word $w[1..n]$ find the largest k such that w contains a substring of the form u^k, for some nonempty word u.

Hint: $\mathcal{O}(n \log n)$ is enough. Guess $|u|$ and try to compute the largest k in $\mathcal{O}(\frac{n}{|u|})$ time using some longest common prefix/suffix queries.

Solution: Iterate through all possible $|u|$. Split w into fragments of length $|u|$ and consider how an occurrence of u^k could look like. It doesn't have to consist of full blocks, but it must start with a suffix of length ℓ of some block, then a number of identical blocks, and then a prefix of length $|u| - \ell$ of the next block. By looking at a picture one can figure out that with one longest common prefix and one longest common suffix query it's possible to compute the longest such word intersecting a given boundary between two blocks. Assuming we can answer such queries in constant time, the whole running time becomes $n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \ldots = n \log n$. To answer the queries, we can use the reduction to RMQ. Given that we are spending $\mathcal{O}(n \log n)$ time anyway, it's possible to use the simpler solution with constant query time but $\mathcal{O}(n \log n)$ preprocessing. In fact seems that answering the queries $\mathcal{O}(\log n)$ time is fast enough, which can be achieved with, for example, hashing and binary searching for the longest common prefix/suffix.