Task sheet 2

Task 10. Show that for a word equation with m occurrences of variables the sum of numbers of different crossing pairs and different letters with crossing blocks is at most $2m$.

Task 11. Let s be a length-minimal solution of a word equation $u = v$. Show that

- Let ab occur in $s(u)$. Show that ab has a crossing or explicit occurrence in $s(u)$ or $s(v)$ (with respect to s).
- Let a occur in $s(u)$. Show that a occurs in u or v, i.e. it has an explicit occurrence.
- Let a^ℓ be a maximal block in $s(u)$. Show that it has a crossing, explicit occurrence or it is a prefix or suffix of some $s(X)$ (so in other words: it touches the cut). It might help to look at $ba^\ell c$.

Task 12. Show that we can uncross and compress all blocks of all letters in parallel, i.e. as one procedure that pops at most one prefix and one suffix per occurrence of variable.

Task 13. A *partition* of an alphabet Σ is a pair (Σ_1, Σ_2) such that $\Sigma_1 \cup \Sigma_2 = \Sigma$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$.

Show that we can uncross and compress a set of pairs $\{a_i b_i\}_{i \in I}$ in parallel, assuming that $a_i \in \Sigma_1$ and $b_i \in \Sigma_2$ for each $i \in I$.

Task 14. Consider a word $w \in \Sigma^*$ such that none of its two consecutive letters are the same. Occurrence of letters from an occurrence of a pair ab in w is *covered* by a partition (Σ_1, Σ_2) if $a \in \Sigma_1$ and $b \in \Sigma_2$. Show that there is a partition of Σ such that it covers at least $\frac{|w|-1}{2}$ letters in w. Show that it can be computed in linear time.

Generalise this observation to a word equation with a solution s (and at most n occurrences of variables).

Hint: Reduce this problem to calculation of a maximal (weighted) cut in a graph. It has a simple randomised solution which can be derandomised using expected value approach. It is described in Michael Mitzenmacher, Eli Upfal *Probability and computing* book [¿] as well as in Vijay Vazirani *Approximation algorithms* book [¿].

Task 15. Using Tasks 12–14 devise an algorithm for word equation that keeps a linear-size equation; the algorithm can use more memory when processing the equations, moreover, at some point it will have to store blocks a^{cn}, but we treat them as size-1. (The latter is a cheat, but we will learn how to deal with this later on).

Task 16. Using Task 15 devise an algorithm that verifies the equivalence of SLPs in polynomial time.

You should probably have it run in two modes: one aims at reducing $|\mathcal{A}|$ by a constant fraction and the other at reducing $|\operatorname{val}(\mathcal{A})|$ by a constant fraction; here $\operatorname{val}(\mathcal{A})$ denotes the word derived by $\mathcal{A}$.

There are are some uniteresting details concerning the exact computational model, so you can ignore the $\log n$ factors.

Note: it is an open problem, whether this can be solved significantly faster than $\mathcal{O}(|\mathcal{A}| \log |\operatorname{val}(\mathcal{A})|)$.

Task 17. [(Long and tedious, but not that difficult), 2 points] The goal of this task is to create a variant of algorithm that performs only compression of pairs, perhaps pairs of the same letter.

Note: we do not use the bound on the exponent of periodicity.

The reason why we cannot use compression of pairs aa is that they can overlap and the compression is ambiguous, for instance consider an equation $aX = Xa$ (all its solutions have $s(X) \in a^*$). We cannot pair letters in X and in $s(aX)$ in the same way.

However, this can be walked around: observe, that a and X commute, as they both represent blocks of a. Thus we can change aX to Xa on the left-hand side, without affecting the equation.

Show, that if there is a particular letter a, such that each variable either:

1. has no a-prefix and no a-suffix *or*
2. is a block of a

then we can rearrange the variables and perform the aa-pair compression. This should pop at most 1 letter from each variable.

Show that afterwards 1–2 is satisfied for a', which represents aa.

To reach an equations satisfying 1–2 we pop a-prefixes and a-suffixes of variables, but represent them as variables.

However, this is not yet enough, as we pile up with many letters popped from variables. To remedy this, we *type* the letters that represent compressed blocks of a: initially we type a and variables satisfying 2; then we additionally perform pair compression for letters that are a-typed. Show that in this way 1 can be generalised: there is no prefix and suffix of a-typed letters.

This should be enough for the algorithm.

Task 18. Assume that Task 17 is correct, i.e. we are able to solve word equations in (non-deterministic) polynomial space performing only compressions of the form $ab \to c$. Show that this implies that the size of the length-minimal solution is at most doubly exponential, i.e. at most $2^{2^{p(n)}}$, where p is a polynomial.

This argument does not work that easily for variant with block compression. Can you say why?

Task 19. Show that the algorithm for word equations (in some variant: choose whichever you like) in fact generates an SLP of size $\mathrm{poly}(n, \log N)$ for some solution of a word equation of size N. How low can you make the dependency on $\log N$?

Task 11 should be helpful.