Word Equations: Sheet 2

Task 7 is very underspecified, but should be doable with the hints inside.

Task 1 Show that for a word equation with n occurrences of variables there are at most $2n$ different crossing pairs and at most $2n$ different letters with crossing blocks.

Task 2 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 3 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 4 Consider a word $w \in \Sigma^*$ such that none of its two consecutive letters are the same. 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 algorithm* book.

Task 5 Using tasks 2–4 give a linear-time algorithm for compressing a word based on the algorithm presented during the lecture. (Model assumption: all letters are integers, we can employ RadixSort on them.)

Task 6 Using tasks 2–4 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 7 Show that for a suitable constant c, if a word equation with at most n occurrences of variables and size at least cn^2 has a length-minimal solution S, then one of the following holds:

- there is a non-crossing pair ab in $S(u)$;
- for some non-crossing a there is a block a^ℓ in $S(u)$, for $\ell > 1$;
- there is a crossing ab such that uncrossing and compressing it does not increase the size of the equation;
- there is a with crossing blocks such that uncrossing and compressing a-blocks does not increase the size of the equation.

Employ this observation in an algorithm for word equations.

Task 8 Long and tedious, but not that difficult The goal of this task is to create a variant of algorithm that performs only compression of pairs, perhaps pairs of the same letter.

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 9 Suppose that the above algorithm was implemented, 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 size-minimal solution is at most doubly exponential.

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

Task 10 An SLP is a context-free grammar generating exactly one word, its size is the size of the right-hand sides of all productions.

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