Problem 49

If the shortest string generated by a CFG is of length $n$, how could you find the lexicographically least such string?

Information

First, note that this problem is equivalent to finding the minimum string under the following ordering, $\leq_m$:

$$x \leq_m y \iff ((|x| < |y|) \text{ or } (|x| = |y| \text{ and } x \leq_y y))$$

Where $x \leq_y y$ means “$x$ is lexicographically less than $y$”.

In other words, $x$ is less than $y$ if $x$ is shorter than $y$, or, if $x$ and $y$ are the same length, if $x$ is lexicographically less than $y$.

Again, we need to find the minimum word $x$ generated by a CFG $G = (V, \Sigma, P, S)$ under $\leq_m$, but we also know that the length of $x$ is $n$.

Solution

First, if $\varepsilon \in L(G)$, then $x = \varepsilon$. Otherwise, we can put $G$ into CNF.

Now, the basic idea is this: Make a table $T$ with entries for each $A \in V$. At each stage, find the minimal string generated by each $A$ by substituting $T[A]$ into each production $A \rightarrow AB$, and taking the minimum string under the given ordering.

We can actually do quite a bit better.

First, remove any production of the form $A \rightarrow BC$ when $A = B$ or $A = C$, since this production would just increase the length of the substituted word. In other words, remove any production for $A$ where $A$ appears in the right-hand side.

Next, we only insert a value $y$ into $T[A]$ when we know that $y$ is the minimum string produced by $A$. If there are any productions of the form $A \rightarrow a$, this is easy, since it’s just the lexicographically least such symbol $a$.

Next, for all variables $A \in V$ (that don’t have any productions of the form $A \rightarrow a$), let this set be $P_A$, and make a table $T_A$ with an entry for each production $A \rightarrow BC$. The value in each $T_A[BC]$ is either empty, the value of the minimum string produced by this production, or marked off. If it is marked off, it means that another production produces a
less element, so this production isn’t usable. Since $T[A_i] = y$ only if $y$ is the minimum string produced by $A_i$, then $T_A[BC]$ can only be substituted into if the minimal strings produced by both $B$ and $C$ are known.

Also, whenever there is more than one $T_A[B_iC_i] = y_i$, find the minimal $y_i$, and mark off all other non-empty $T_A[B_iC_i]$. I’ll call this rule the “minimisation rule”. Also, since the minimal word is of length $n$, we can immediately mark off any $T_A[B_iC_i] = y_i$ when $|y_i| > n$. Therefore, each comparison takes at most $O(n)$ comparisons, since each possibly minimal string has length at most $n$.

Now, we know that if all but one $T_A[B_iC_i] = y_i$ is marked off, then $T[A] = y_i$, since every possible production rule $A \to BC$ has been tested with the minimum possible $B$ and $C$, so $[A] = \min x \in T_A$. I’ll call this rule the “completeness rule”. This means that there are at most $|P_A| - 1$ comparisons to be made, so finding each $T[A]$ requires at most $O(|P_A| \cdot n)$ comparisons.

Also, since $P \supseteq \bigcup_{A \in V} P_A$, there are at most $O(|P| \cdot n)$ comparisons required for the whole algorithm.

Also note that, if substituting $B = y$, $C = z$ into $A \to BC$ takes $O(|y| + |z|)$, then the total time spent on substitution is just $O(|P| \cdot n)$, since any substitution that would produce a word longer than $n$ can be marked off.

Since we can ignore any string that is longer than $n$, we want to be able to find out the length of a string quickly. Thus, we store the length of a string along with the string. This doesn’t change the running time of anything like comparison, but it does slightly change the running time of substitution. Since substituting $B = y$, $C = z$ into $BC$ results in a string with length $|y| + |z|$, we must calculate it, which takes $O(\log n)$ time. Therefore, substitution actually takes $O(|P| \cdot (n + \log n)) = O(|P| \cdot n)$ time.

Algorithm

Now: we know various aspects of the algorithm, but we still don’t know how to combine into the full algorithm.

First, as stated before, for all variables $A$ that produce a set of terminals $\text{term}_A$, let $T[A] = \min(\text{term}_A)$. This requires

$$\sum_{A \in V} (|\text{term}_A| - 1) \leq |P|$$

comparisons, since each comparison between terminals only takes 1 comparison.

Now, we start the recursive phase. For all $A \in V$ such that $T[A]$ is empty, if we know that $A \to BC \in P_A$ and $T[B]$ and $T[C]$ are non-empty, then $T_A[BC] = (T[B]T[C])$. Since (at this point), the maximum string produced is of length 2, we know that any $y = \min(T_A)$ we just produced must be of length 2, and since the only way to create such a string is if $T[B]$ and $T[C]$ are terminals, so every possible string of length 2 has been created, so we can set $T[A] = y = \min(T_A)$. 


In fact, we can generalise this: by (the end of) phase $\ell$, all strings of length $\ell$ for each variable $A$ will have been generated (see appendix for proof).

What does this tell us? Well, if, in phase $\ell$, we find that there is some production such that $T_A[BC] = w$ and $|w| = \ell$, then $T[A] = y = \min(T_A)$, since for any string $z$ with $A \rightarrow^* z$ generated by a later phase, $|y| < |z|$ and thus $y \leq_m z$, so $y$ must be the minimal string generated by $A$. I’ll call this rule the “early-determination rule”.

Thus, the full algorithm would look like this (where “min” denotes the minimum under $\leq_m$):

$$\text{minWord}(G = (V, \Sigma, P, S), n)$$

1. Remove all productions of the form $p_i = (A \rightarrow \cdots)$ where $A$ appears in the right-hand side of $p_i$.

2. For $A_i \in V$, let $\text{term}_{A_i} := \{ a \in \Sigma : (A_i \rightarrow a) \in P \}$.

3. Set up a table $T$ with entries for each $A \in V$. If $\text{term}_{A} \neq \emptyset$, then set $T[A] = \min(\text{term}_{A})$. Otherwise, leave $T[A]$ empty.

4. For each $A$ such that $\text{term}_{A} = \emptyset$, set up another table, $T_A$ with entries for each production $(A \rightarrow \cdots) \in P$. At the moment, leave all entries empty.

5. Let $P_o$ be the set of transitions of the form $A \rightarrow BC$ such that $T[A]$ is empty and $T_A[BC]$ is empty and unmarked.

6. For $2 \leq \ell \leq n$:

   (a) For each $p_i \in P_o$, if $p_i = A \rightarrow BC$ and both $T[B]$ and $T[C]$ are non-empty, then set $T_A[BC] = T[B]T[C]$ and remove $p_i$ from $P_o$.

   (b) For each $T_A$ that was modified in the previous step, apply the minimisation rule to $T_A$.

   (c) Apply the completeness rule to all modified $T_A$.

   (d) Apply the early-determination rule for all $T_A$, and for any $T_A$ for which it applies, remove all $p_i = A \rightarrow \cdots$ from $P_o$.

8. return $T[S]$.

Since we have already calculated the total time taken up by calculations and substitutions, we can ignore the time taken by these in the loop.

Step (a) takes two operations for each $p_i$: one to lookup $T[B]$, and one to lookup $T[C]$. Thus, (a) takes $O(|P_o|) = O(|P|)$ time.

Step (b) requires looking at each $T_A$, and looking to see which are unmarked. Thus, it takes $O(|P|)$ time.

Step (c) again requires looking at each modified $T_A$, and seeing whether all but one entries are unmarked in $T_A$. Thus, it takes $O(|P|)$ time total.
Step (d) requires calculating whether \( m_A = \ell \), which takes \( \mathcal{O}(|V|) \) comparisons.
Therefore, the loop takes \( \mathcal{O}(|P| \cdot n + |V| \cdot n \log n) \) time.
Thus, since the substitutions and comparisons take \( \mathcal{O}(|P| \cdot n) \) time, the whole algorithm takes
\[
\mathcal{O} (|P| \cdot n + |V| \cdot n \log n)
\]
time.

Proof

Proof of Correctness (Incomplete)

Most important to prove are the two invariants:

1. That \( T[A] \) is the minimal word generated by \( A \).
2. That \( T[A][BC] \) is the minimal word generated by the production \( A \to BC \).

We know that, at the start, (1) is true, because if a variable can produce terminals, then the least word produced by that variable is the least terminal.

Next, inductively, assume that in phase \( \ell \), before applying the completeness/early-determination rules, that invariant (2) holds, and that invariant (1) holds at the moment. We need to show that invariant (1) still holds after applying the completeness and early-determination rules.

Since \( T[A][BC] = y \) is the minimal word generated by \( A \to BC \), we know that if it is the only unmarked word in \( T[A] \), then \( y \leq_m z \), for any word produced by any production of the form \( A \to \cdots \). Therefore, \( y \) is the minimal word produced by \( A \). Thus, inserting \( y \) into \( T[A] \) preserves the invariant.

Now, for early-determination:

If \( T[A][BC] = w \) and \( |w| = m \), then there exist words \( y, z \) such that \( T[B] = y \) and \( T[C] = z \).
This means that \( y \) and \( z \) are the minimal words produced by \( B \) and \( C \) respectively.

Now, for a contradiction, assume that \( m = \ell \) and that in some later phase, there will be another word, \( w' \leq_m w \) with \( |w'| = m = \ell \).

Now, again, there are words \( y', z' \) and variables \( B', C' \) such that \( T[B'] = y' \) and \( T[C'] = z' \) and \( T[A][B'C'] = w' \). Now, \( y' \) and \( z' \) must have been generated some time after phase \( |y'| \) and \( |z'| \) respectively; otherwise, they could be produced in an earlier phase. This remains true for every decomposition of \( y' \) and \( z' \). Eventually, you will get to the point where some terminal must have been produced some time after the first phase, which is a contradiction.

Therefore, if \( z \) is the minimal word produced by \( A \) where \( |z| = \ell \), then \( T[A] = z \) by the end of the \( \ell \)-th phase.
Appendix: More Formal Proofs/Lemmas

1. For all $A \in V$, if the minimal string $z$ produced by $A$ is of length $\ell$, then $T[A] = z$ by the end of the $\ell$-th phase.

   To see this, look at this for the first few $\ell$:

   By the end of phase 1, each variable $A$ that can produce a terminal generate that/those terminal(s), and the minimum terminal is chosen. A terminal cannot show up in a later phase, since that would mean that $BC \rightarrow^* a$, which isn’t possible, since $G$ is in CNF.

   Next, in phase 2, $T_A[BC] = bc$ if $T[B] = b$ and $T[C] = c$. Since all minimal strings of length 1 have already been generated, then all strings of length 2 will be generated by this phase, since all $T[A]$ are non-empty if $T \rightarrow^* a$.

   Now, for the inductive step: assume that all strings of length $\ell$ are generated by the end of the $\ell$-th phase. Now, look at the strings generated/determined in phase $\ell + 1$. All strings generated by substitution this phase are of the form $w = yz$, where $|y|, |z| \leq \ell$, $T[A]$ is empty, $(A \rightarrow BC) \in P$, and $T[B] = y$ and $T[C] = z$. Now, for all words where $|w| = \ell + 1$, this must mean $|y| + |z| = \ell + 1$. Thus, there are $\ell$ possible decompositions for $w$, with the lengths of $(y, z)$ ranging from $(1, \ell)$ to $(\ell, 1)$. But since all possible minimal words of length $\leq \ell$ have already been generated by this point, this means that any minimal word $w$ with $|w| = \ell + 1$ could be generated by this phase or an earlier phase.

   Therefore, for all variables $A$ where the minimal string $z$ generated by $A$ is of length $\ell$, that string $z$ will be generated and inserted into $T[A]$ by the end of phase $\ell$. 