Master 1 MoSIG

Algorithmic Problem Solving

APP4 Report

Deciphering the Morse

Team:
SACAD

Members:
Andrey **SOSNIN**
Majdeddine **ALHAFEZ**
Antoine **COLOMBIER**
Eman **AL-SHAOUR**
Son Tung **DO**

Grenoble, 1 December, 2017

1 Notations

D is the dictionary of Latin-character words, $\mathcal{D}$ is the translation of the words in D into Morse, such that $|D| = |\mathcal{D}|$ (*i.e.* two Latin-character words which have the same translation into Morse are considered as two different elements of $\mathcal{D}$). We denote the Morse sequence by $m = (m_0 \dots m_{L-1})$. $m[i,j]$ denotes the subsequence $(m_i \dots m_{j-1})$.

2 Introduction

Our goal was to propose an algorithm which would give the exact solution. One could easily imagine a naive approach - to perform the whole space exploration which would consist in assuming the first word of the sequence to be w' for all $w' \in \mathcal{D}$ matching the beginning of the sequence and then repeating recursively on the rest of the sequence - which would lead to an exponential time complexity. Instead we take a dynamic programming approach whose principle is explained in the next section. We suppose that $\epsilon \notin D$.

3 Principle

The proposed dynamic procedure is based on the following observation:
Let w be a word of the dictionary such that its unique translation into Morse alphabet w' of length $|w'|$ occurs in m at index $i > 0$ (i.e. $m[i, i + |w'|] = w'$). Then the number $C_{i,w'}$ of interpretations of $m[0, i + |w'|]$ whose last word is w, equals the number C_i of interpretations of $m[0,i]$.
This leads us to establish a recursive formula for C_i:

$$C_i = \sum_{\substack{w' \in \mathcal{D} \\ m[0,i] = \alpha w'}} C_{i-|w'|,w'} + F_i = \sum_{\substack{w' \in \mathcal{D} \\ m[0,i] = \alpha w'}} C_{i-|w'|} + F_i$$

where $C_0 = 0$, "$m[0,i] = \alpha w'$" means that w' is a strict suffix of $m[0,i]$ and

$$F_i = |\{w' \in \mathcal{D} \mid m[0,i] = w'\}| \quad (note\ that\ F_i = 0\ for\ i > 4 \times M)$$

4 Algorithm

The following pseudocode is a straightforward application of the recursive formula, with the value of interest being C_{L-1}. We compute the values of C_i for i going from 1 to $L - 1$ and store them in a cache array:

```
procedure compute_number_sequences(m, L, 𝒟)
    C ← create_array(L, 0);
    for i = 1 ... L − 1 do
        for w′ ∈ 𝒟 do                          ▷ compute the sum and store it into C[i]
            if m[i − |w′|, i] = w′ then                ▷ the word matches the sequence
                if |w′| = i then                          ▷ is it the first word in m?
                    C[i] = 1
                else
                    C[i]+ = C[i − |w′|]
                end if
            end if
        end for
    end for
    return C[L − 1]
end procedure
```

5 Complexity analysis

Time complexity

At every iteration inside the innermost loop, aside from some constant-cost operations, we perform an equality check between two strings of length bound by M: it costs $O(M)$. In total there are $L \times N$ of those iterations: the total procedure's time cost is in $O(L \times N \times M)$. It is polynomial in all variables.

Space complexity

We have to produce an array of size L. Also, if we consider that the dictionary $\mathcal{D}$ of words translated into Morse is not available at the beginning, storing it would cost an additional $O(N \times M)$ overhead – however, iterating on words of D (rather than $\mathcal{D}$) and translating them on-the-fly would neither increase the asymptotic space complexity nor the time complexity (as it would take $O(M)$ operations each time and would count additively).

6 Improvements

The first improvement to the space complexity would consist in storing at any given point only last 80 computed values of C, as there can be no words longer than $4 \times M = 80$ Morse symbols. For that we should replace $C[x]$ by $C[x \ mod \ 4 \times M]$ for all x (as well as $create\_array(L, 0)$ by $create\_array(4 \times M, 0)$).

Another point of view would be to consider that at the beginning $\mathcal{D}$ is available as the tree whose root is ϵ and where a word x is the parent of a word y if and only if x is a (strict) suffix of y. An example is presented in the figure below. We could exploit this structure to reduce the average number of considered words and thus, computations to check if a word is a suffix of the Morse sequence: if x is an ancestor of y and x does not match the end of m then neither does y. To implement this improvement we have to replace the iteration *for* $w' \in \mathcal{D}$ by a depth-first search in the suffix tree (omitting useless branches).

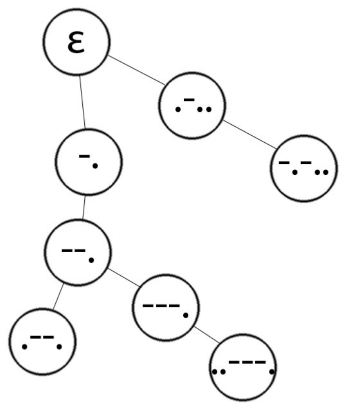

Figure 1: The suffix tree representation of the dictionary $\{-., \ .-.., \ --., \ -.-.., \ .--.---., \ ..---.\}$