APP4 Morse code

PATOTSKAYA Alisa PATOTSKAYA Yulia
REY Ruben SID-LAKHDAR Riyane
PAHEVICH Alexander

December 3, 2015

Abstract

TODO

Contents

1 Polynomial algorithm: second approach

1.1 Data structure

Let us first encode each word of the given English dictionary into a Morse sequence by applying the encryption function to each character. Each character can be encoded using at most 4 Morse characters (. and -). Therefore, each encoded word's length is bounded by $4 * M$ (where M is the length of the Morse sequence to process).

Let put those words into a hashmap data structure W. The keys of this hashmap are the Morse-encoded dictionary words. The value corresponding to each Morse key word k_M is the number of different English word with the same Morse transcription k_M.

For instance, let's suppose a sequence ”-....–.-” encodes the following words: ”BAC”, ”BANN”, and ”DUC”. Then W[”-....–.-”] = 3. We can build this data structure with a linear complexity O(N * M) where N is the number of words of the dictionary and M the maximum size of such a word.

1.2 Algorithmic principle

Let S the Morse sequence of length L that we are trying to decode. We suppose that a partition P of the string S exists such as $P = [s_1, s_2, ..., s_{lp}]$. Then the number of ways to decode S following to this partition is given by:

$$C(P) = \prod_{i=0}^{lp} W[s_i]$$

We can easily notice that the total number of ways to decode the string S equals to the sum of $C(P)$ for all the possible partitions P.
We can also notice, that we are only interested in the partitions P such as $\forall s_i \in P, s_i$ is a key of the hashmap W. For all the other partitions P' we consider that $C(P') = 0$.

****
Let us consider the last sub-string of any partition P. It should be present in W. In particular, that means there should be a suffix of length t (1 ¡= t ¡= 4 * M), that is present in W. Notice, there are no suffixes of length ¿ 4 * M, that might be present in W, and thus we are not interested in partitions that have the last substring of this length.
****

Since we don't have any keywords longer that 4 * M, we can only consider sufixes s of of length ¡= 4 * M (all the rest summands will be equal to zero). This formula can be easily prooved by induction on length(S), taking F(0) = 1 as a basis for induction.

1.3 Algorithm

```
algorithm(S: morse code of length M)
{
    Build a hashmap W.
    Allocate an array F[0..M] to store values of the F function.
    Set F[0] = 1, a basis for the induction.
    for i = 1 to M do
        F[i] = 0
        for suffix_len = 1 to min(i, 4 * M) do
            factor = W[S.substring(i - suffix_len + 1 .. i)]
            if factor > 0 then
                F[i] += F[i - suffix_len] * factor
            end
        end
    end
    return F[L]
 }
```

Figure 1: Greedy algorithm for the hole drilling problem.

1.4 Optimality ??????

1.5 Complexity

The time complexity of the previous algorithm is clearly the sum of the complexity of building the hash map and the complexity of computing the function F.

- As we stated above, the time complexity of building the hash map is $O(N * M_{english})$ where N is the number of words of the dictionary and $M_{english}$ the maximum size of such a word (in English). The space complexity is $O(N * (M_{english} + M_{morse})$ where M_{morse} is the maximum size of a morse word ().
- The time complexity of the function F is $O(N*(4*M))*W_{lookupcomplexity} = O(N * M^2) = O(N * M^2)$. While its space complexity is O(M)

2 Conclusion

TODO