Repeating sub-string finder algorithm

Contents

1 Introduction

This document purposes a viable algorithm for finding all the repetitions of all the repeating sub-strings in an arbitrary string. Repeating sub-strings are defined by this document as follows:

Definition 1.1. *Repeating sub-string Z.*

1. *Let A be an arbitrary selected alphabet.*
2. *Let S be an arbitrary selected string that consists of two or more characters that are elements of A.*
3. *Let S' be a sub-string of S.*
4. *If there is another sub-string, S'', in S that is identical to S' but does not start at the index of S' in S then the sub-string S' is repeated in S and the repeating content of the sub-strings is then denoted by Z. The sub-strings S', S'', etc are donated as Z_1,Z_2, etc which is generalised to Z_i where i indicates the sequential index of the repeating sub-string in relation to all repeating sub-strings of Z.*
5. *Z_1 indicates the first occurrence of Z in S from left to right and Z_N indicates the last occurrence of Z in S. Where N is defined as the number of occurrences of a particular repeating term.*

The document will further propose an extension of the algorithm that will enable it to find the all the X-Y-X repetitions that occur in the arbitrary sting S. A X-Y-X repetition is defined by this document as follows:

Definition 1.2. *X-Y-X Repetition*

1. *Let A be an arbitrary selected alphabet.*
2. *Let S be an arbitrary selected string that consists of two or more characters that are elements of A.*
3. *Let X be a non-zero length sub-string of S which repeats M times in S. Where $M > 1$. This is the repeating term of the X-Y-X repetition.*

4. *Let Y be any of the M - 1 sub-strings of S, which consists of zero or more characters and which starts directly after the end of some occurrence of X_i and ends directly before the occurrence of X_j. Where $j = i + 1$.*

5. *Hence a X-Y-X repetition is defined as the M identical occurrences of X in S that are each separated by one of the M - 1 varying sub-strings Y.*

The above definitions and terms lay the bases of the algorithm and will be used throughout the document.

2 Algorithm

In order to provide a formal description of the algorithm purposed above the *Repeating sub-string* definition must be used in conjunction with following observations:

Observation 2.1. *Repeating sub-strings are identical to the prefix, of length $|Z|$, of the sub-string of S starting at the starting index of the first occurrence of Z, Z_1, in S and ending at the last character in S.*

1. *Let R be the set of all non-identical repeating sub-string content Z in S.*
2. *Let R_i be the set of repeated sub-strings $Z_1, ..., Z_N$ for a particular sub-string content Z.*
3. *Let S' be the sub-string of S which starts at the index of Z_1 in S of some set R_i and extends to last index in S.*
4. *Therefore S' contains the N occurrences of Z in S. Since $Z1$ to Z_N all in S and are sequential.*
5. *Let P be the prefix of S' with a length of $|Z|$ of R_i.*
6. *Since Z_1 is the start of S'. It is also P.*
7. *It can be observed that each occurrence of Z, Z_i, is identical to the P.*

Observation 2.2. *Each occurrence of Z in the string S forms a cluster of $\frac{|Z|(2+(|Z|-1))}{2}$ repeating sub-strings of S.*

1. *Z has two or more occurrences in S by definition.*
2. *Each occurrence, Z_i, contains one or more characters by definition.*
3. *Splitting the occurrence,Z_i, into J layers denoted by L, where J is $|Z_i|$ so that each layer L_j, where $0 \leq j < J$, contains $J - j$ partitions of the sub-string Z. Each partition has a length of $j + 1$ each, where each partition's starting index is pairwise disjoint with any other partition's starting index on the same layer.*

4. *Each of the partitions on each layer can be seen as a repeating sub-string in S since there is more than one occurrences of it in S since there is more then one occurrence of Z as shown above.*

5. *Since each layer adds $J - j$ to the total partitions for a particular value of J. Hence the number of partitions contained by a sub-string Z is the summation of the layers which is $J + J - 1 + J - 2 + ... + 2 + 1$. This is an arithmetic sequence that can be described by the explicit formula $a_n = a_1 + (n-1)d$. Thus the summation of these partitions can be described by the explicit formula $\frac{J(2+(J-1))}{2}$.*

6. *Therefore Z forms N clusters containing $\frac{|Z|(2+(|Z|-1))}{2}$ repeated sub-strings. Since $J = |Z_i| = |Z|$.*

Observation 2.3. *If any two repeating sub-strings, R' and R'', of any of the repeated sub-strings in S overlap at some leftmost position, p, and rightmost position ,q, in S, then $\frac{(q-p)(2+((q-p)-1))}{2}$ repeating sub-strings are in the set of $R'_{\text{repeating sub-strings}} \cup R''_{\text{repeating sub-strings}}$.*

1. *Let R' be the leftmost repeating sub-string in S of any two overlapping arbitrary selected repeating sub-strings of any of the repeating sub-strings in S.*

2. *Let R'' be the rightmost repeating sub-string in S of any two overlapping arbitrary selected repeating sub-strings of any of the repeating sub-strings in S.*

3. *Let p be the character index in S which indicates the leftmost overlapping character.*

4. *Let q be the character index in S which indicates the rightmost overlapping character.*

5. *Therefore it can be seen that $R' - p$ characters overlap, form an overlapped sub-string S' in S.*

6. *Since both R' and R'' form repeating sub-string clusters it can be that S' forms a repeating sub-string cluster since it consists of only characters that are repeated.*

7. *Therefore by definition and observation 2.2 the new repeating sub-string cluster contains* $\frac{(q-p)(2+((q-p)-1))}{2}$ *repeating sub-strings.*

8. *Therefore both* R' *and* R'' *contain* $\frac{(q-p)(2+((q-p)-1))}{2}$ *identical repeating sub-strings.*

Observation 2.4. *If any two repeating sub-strings,* R' *and* R''*, of any of the repeated sub-strings in S overlap in such a way that one of them completely overlaps all the characters of the other then the longer of the two contains all the repeating sub-strings in the cluster of the smaller as well as it owns. Thus the longer absorbs the smaller.*

The above observations forms the core characteristics of a repeating sub-string and the string that contains it which will be exploited by the algorithm. The algorithm primarily consists of two stages. A third stage can be added that allows for the extension of the algorithm as stated in the introduction of this document.

2.1 Stage 1: Finding maximum longest prefixes starting at each character index in S

The stage 1 algorithm analyses the string S in such away that a numeric array U, which has the same length as S, is produced. Each element of U indicates the length of the longest repeating sub-string starting at the particular character index the element corresponds to.

1. The first stage splits the arbitrary string S into a array of strings, K, of $|S|$ sub-strings. Where the first sub-string in the array is S. Each of the sub-strings in K have the following properties:

   (a) Is donated by K_i, where i is its index in the array and $0 \leq i < |S|$.

   (b) Has a length $T = |K_{i-1}| - 1$ or $T = |S| - i$, one less the length of the sub-string prior to it in K.

   (c) Is a sub-string of S which starts at the character $C_{|S|-T}$ in S and ends at character $C_{|S|}$ in S.

2. Next the algorithm creates a numeric array, U, which has a length of $|S|$ and all of its elements are initialised to zero. U has the following properties:

(a) Each element, E_i is mapped to the corresponding character C_i in S, where $0 \leq i < |S|$.

(b) Each element has numeric value ≥ 0.

(c) Each element represents the maximum longest prefix of all the longest prefixes found in each of the sub-strings in K which have a character index y which can be mapped to the character index C_i in S. Where the character index y donates the starting index of the prefix of particular sub-string in K. The mapping is defined as $i = y + (|S| - z)$, where z is the index of sub-string in K and y is the character index in K_z.

3. Next each sub-string, K_z, in K is then scanned iteratively for the longest prefix sub-string* starting at each character, K_{zy}. If the value of the element in U which is mapped to by the character index y of the sub-string K_z, is less than the length of longest prefix starting at K_{zy} then it is replaced by the length of the longest prefix starting at K_{zy}.

4. Since the array elements captures the maximum length of longest prefixes starting at each character index this implies that the elements following the particular starting element will contain a value one less than the previous. This due to the fact that a prefix string covers $|prefix - string|$ elements in U and the cluster repeating sub-strings in the prefix. Thus next the array U is scanned iteratively from left to right in order to remove this unnecessary information. If one of the elements is not zero and does not follow the linearly decreasing sequence above then the element indicates the starting point of a repeating sub-string that is not completely overlapped by the other. In order to remove the unnecessary information, elements containing implied values can be changed to zero. The sub-routine proposed to do so can be found in the pseudo code below.

*In order to find the longest prefixes a modified version of the O(n) longest prefix sub-string finder algorithm was used. The algorithm was created by Michael G. Main and Richard J. Lorentz and documented in their research article *An O(n log n) Algorithm for finding all repetitions in a String** which was published in the Journal of Algorithms in 1984.
The pseudo code of the stage 1 algorithm follows:

2.2 Stage 2: Extracting all repeating sub-strings from U

In order to find all the repeating sub-strings in S the algorithm needs to extract them from the array U. Since each non-zero element in U indicates a the starting position of the longest repeating sub-string with a length equal to the value of the element, l. It also indicates the starting position of a cluster of $\frac{(l)(2+(l-1))}{2}$ repeating sub-strings. Thus in order to extract the repeating sub-strings this stage needs to scan the the array U from left to right and at every element that is not zero extract the sub-string in S starting at the corresponding character index. This string needs to be partitioned into the $\frac{(l)(2+(l-1))}{2}$ partitions and recorded. In order to avoid duplicate repeating sub-strings from being created by non-fully overlapped repeating sub-strings a special case must be considered. To cater for both cases the partitioning should occur from left to right in terms of starting position and bottom to top in terms of layers.

The pseudo code for the stage 2 algorithm follows:

```
S ← String beginning scanned
U ← Array of longest repeating sub-string starting positions
L1 ← Empty string array
L2 ← Empty numeric array

//Scan U for repeating sub-strings starting positions
FOR i = 0 to (|U| − 1)
BEGIN
  //Check if element of U is non-zero
  IF U_i > 0
  BEGIN

    //Extract repeating sub-strings
    FOR j = 0 to (U_i − 1)
    BEGIN

      //Check for second case repeating sub-strings
      IF (U_{i+j} > 0) AND (j NOT 0)
      BEGIN
        Stop extracting partitions for current sub-string
      END

      FOR k = 0 to (U_i − j)
      BEGIN
        Grow L1 and L2 by one element so their new size is n
        L1_n ← Sub-string in S which starts at (i + j) and end
   at (k + 1)
        L2_n ← Starting position of the repeating sub-string (i
    + j)
      END

    END

  END
END
```