Algorithm Conception and Analysis (CAL)
Master's in Informatics and Computation

# *Garbage Collection in Smart Cities - String Matching Algorithms*

**Final Report**

Students:

**Luís Cruz** - UP201303248@fe.up.pt

**Manuel Curral** - UP201403027@fe.up.pt

**Miguel Ramalho** - UP201403027@fe.up.pt

April 4, 2019

# Contents

## 1 Introduction

This report seeks to address the problem of optimizing a city's collection of trash through the monitorization of garbage containers. This is made possible by equipping said containers with volume sensors, therefore creating a way of knowing when a given container's content needs collecting.
This is the second of a series of reports on this project, and focuses specifically in the analysis of string matching algorithms, namely the *Exact Search Algorithm* and the *Approximate Search Algorithm*.
This work seeks to test both algorithms to satisfy the search of a container location by using two street's names to find it.

## 2 Problem Formalisation

Previous to the formalisation of the problem at hand, there is a need to establish a formal notation to use throughout this paper, as follows:

- $G$ represents the graph which abstracts an area's roads layout;
- $R$ represents the set of roads in the graph $G$;
- $R_i$ represents a specific road, referencing a physical location in a map;
- $V_ij$ represents a specific intersection between roads $R_i$ and $R_j$.

### 2.1 Input

The input to this model shall be a graph representation of the network which is to be analysed, as described before. Our data comes from the OpenStreetMap Project OSM. From this source, we extract an area's layout by using a tool, OSM2TXT Parser, that generates 3 files from a `.osm` file.
The first file contains information about the nodes: , latitude, longitude in degrees and radians. The second file contains information about the roads, ids, names and if it is two way road. Finally, the third text file contains information about the connection between two nodes, in other words, the information about the set of edges that defines a roads geometry.

### 2.2 Output

The output of the model is a essentially a boolean value indicating whether a given pair of roads $R_i$ and $R_j$ possess a garbage container at their intersection.

## 3 Approach 01 - Exact Search

### 3.1 Algorithm

This type of algorithm is part of the string-matching problems most basic solutions. The goal is to search and match the pattern array $P$ in text array $T$. We consider various possible occurrences in the text and mark the displacement at the beginning of the text, note the possibility of overlapped occurrence of same pattern on the text. The two algorithms most known are: Naive and KMP (Knuth Morris Pratt). The main difference is the use of value $lps[]$ and with this the KMP: skip characters while matching or decide upon the next characters to be matched.

#### 3.1.1 Pseudocode

**Algorithm 1** KMP-Matcher

```
procedure KMP(T,P)                ▷ Where T - string text, P - string pattern
    $num \leftarrow 0$            ▷ Number of times the pattern is in the text
    $n \leftarrow length[T]$
    $m \leftarrow length[P]$
    $prefix \leftarrow PRE - KMP(P)$          ▷ Where prefix - vector
    $q \leftarrow -1$
    for i = 1 to |n| do                       ▷ Scan the text from left to right.
        while  q > 0 and P[q+1]≠[i] do
            $q \leftarrow prefix(q)$          ▷ Next character does not match.
        if $P[q+1]$=T[i] then
            $q \leftarrow q + 1$              ▷ Next character matches.
        if $q$ = m-1 then                     ▷ Is all of P matched?
            print "Pattern occurs with shift" i - m
            $num \leftarrow num + 1$          ▷ Pattern matches.
            $q \leftarrow prefix(q)$
    return $num$
```

**Algorithm 2** PRE-KMP

```
procedure PRE-KMP(P,prefix)          ▷ Where P - string pattern,
prefix - vector
    $m \leftarrow length[P]$
    $prefix[0] \leftarrow -1$
    $k \leftarrow -1$
    for q = 1 to |m| do
        while  q > -1 and P[k+1]≠[q] do
            $k \leftarrow prefix(k)$
        if  $P[k+1]$=P[q] then
            $k \leftarrow k+1$
        $prefix(q) \leftarrow k$
    return $prefix$
```

### 3.1.2 Time efficiency

Time complexity of KMP algorithm is O(n) in the worst case + O(m) from PRE-KMP(P).

$O(n)$ or $O(m+n)$ in the worst case.

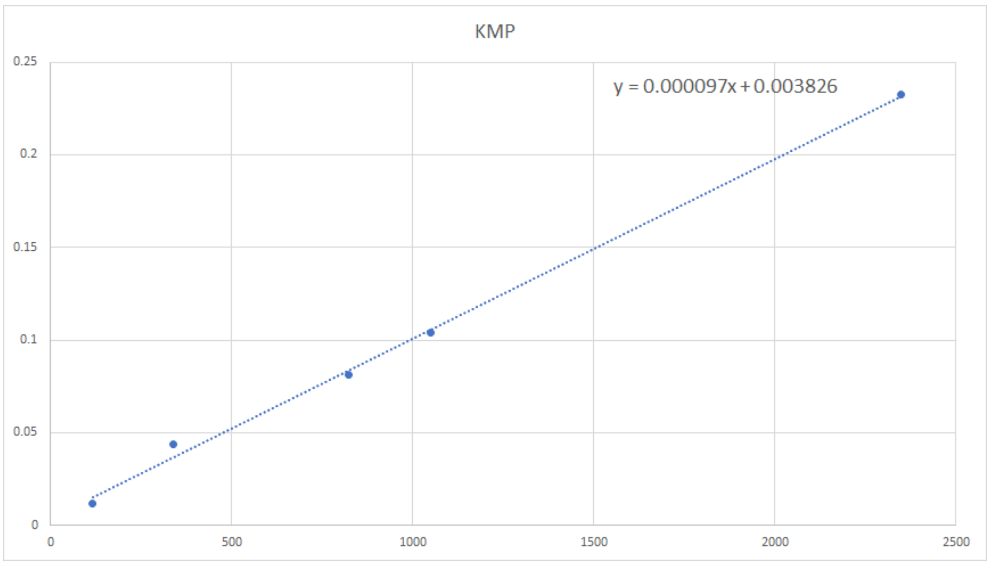

Figure 1: Empiric time efficiency for the KMP algorithm

### 3.1.3 Space efficiency

Space efficiency is only related to how many variables are necessary to compute the algorithm, in this case it is m, where m is the length of

Pattern:

# 4 Approach 02 - Approximate Search

## 4.1 Algorithm

This algorithm is a dynamic programming algorithm used to calculate the distance between two different strings. The distance is represented by the number of changes needed to transform the text string into the pattern string. This changes can be, additions, deletions or substitutions of a character from the pattern string.

### 4.1.1 Pseudocode

**Algorithm 3** Edit Distance

```
procedure EDIT DISTANCE(Vertex P)
    for j=0 to n do D[j] = j do                              ▷ inicialization
    for i=1 to m do do                                       ▷ recurrency
        old = D[0]                                           ▷ stores D[i-1,0]
        D[0]= i                                              ▷ inicializes D[i, 0]
        for j=1 to n do do
            if $m[i] == n[j]$ then
                $new = old$
            else
                $new = 1 + min(old, D[j], D[j-1])$
                $old = D[j]$
                $D[j] = new$
    return $D[n]$
```

### 4.1.2 Time efficiency

As we can see in the edit distance pseudocode, we have two nested loops, one iterates through |n| the other iterates through |m|. Thus being the main algorithm on the approximate string matching it defines the time efficiency factor of these approach. The Approximate search time efficiency will be:

$$O(n \times m)$$

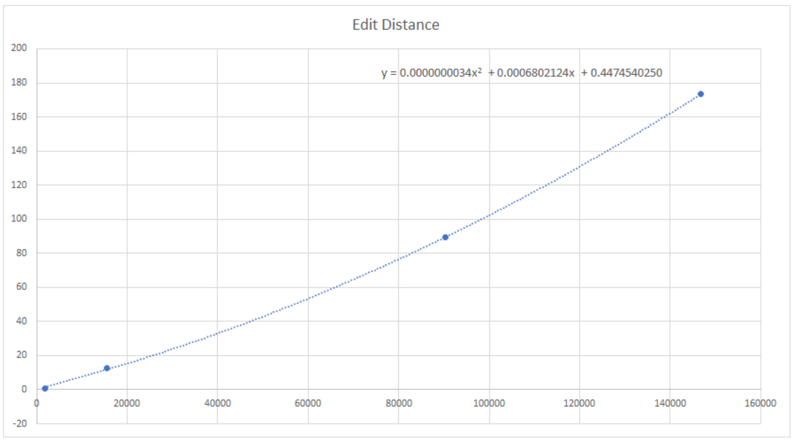

Figure 2: Empiric time efficiency for the Edit Distance algorithm

### 4.1.3 Space efficiency

To run this algorithm we need to create one integer, n for the index of the text string. This way the space efficiency will be:

$$O(1)$$

## 5 Conclusion and Final Remarks

The second part of this project was very helpful in the understanding of the string search algorithms and their applications, namely the use of approximate search to display suggestions and similar searches to users, allowing the effect of human error in text input to be minimized in an efficient and, globally, helpful way.

## 6 References

[1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990). Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill. ISBN 0-26-03141-8. See in particular Section 32, "String Matching", pp.906-932. [2] Johnson, D. S.; McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF). In Aarts, E. H. L.; Lenstra, J. K. *Local Search in Combinatorial Optimisation.* London: John Wiley and Sons Ltd. pp. 215–310. [3] Crochemore, Maxime; Rytter, Wojciech (2003). Jewels of stringology. Text algorithms. River Edge, NJ: World Scientific. pp. 20–25 [4] Baeza-Yates, R.; Navarro, G. (June 1996). "A faster algorithm for approximate string matching". In Dan Hirchsberg; Gene Myers. Combinatorial Pattern Matching (CPM'96), LNCS 1075. Irvine, CA. pp. 1–23 [5] Gusfield, Dan (1997). Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge, UK: Cambridge University Press