# A LEVENSHTEIN TÁVOLSÁG[⊗]

# THE LEVENSHTEIN DISTANCE

**KOS Victor**

asist. univ.
Universitatea din Oradea, Facultatea de Stiinde, Departamentul de Matematica Informatica,
Str. Universitatii nr.1, Oradea, Bihor, Romania
vkos@uoradea.ro

***Kivonat:*** *A Levenshtein távolság két karaktersorozat közötti távolság mérésére szolgál. Ez esetben a távolság a hasonlóságok illetve különbségek mérését jelenti a két karaktersorozat között. A Levenshtein távolság szerkesztési távolságként is ismert. A Levenstein távolságot két karaktersorozat között a két string közötti minimális számú, egyetlen egy karakter törlése, beillesztése vagy helyettesítési műveletek száma adja meg.*

***Kulcsszavak****: Levesnstein távolság, szerkesztési távolság, C# alkalmazás a Levesnstein távolság kiszámolásához.*

***Abstract:*** *Levenshtein distance is a metric for measuring the amount of difference between two sequences. A metric, in this case, measure similarity or dissimilarity (distance) between two text strings for approximate matching (fuzzy string search) or comparison. The term edit distance is often used to refer specifically to Levenshtein distance. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character.*

***Keywords:*** *Levenshtein distance, Edit distance, C# implementation of Levenshtein distance.*

## 1. BEVEZETÉS

E cikk célja egy egyszerű és könnyen megközelíthető leírása a Levenshtein távolságnak, és annak implementálása C# .NET – ben.

Vladimir Levenshtein, orosz matematikus, vezeti be ezt a fogalmat 1965-ben. A Levenshtein távolság ismert még, mint Edit Distance (ED) [1].

## 2. DE MI IS A LEVENSHTEIN TÁVOLSÁG?

A Levesnstein távolság (LD) két karaktersorozat (azonos rögzített véges értékkészletű véges sorozatok) közötti távolság mérésére szolgál. Legyen (s) a forrás string és (t) a cél string. A Levenstein távolságot a két string közötti törlés, beillesztés vagy helyettesítő műveletek száma adja meg, úgy hogy (s) sorozatot (t) sorozattá alakítsuk át.

- Ha (s) string = „próba” és (t) string = „próba”, akkor LD (s, t) = 0, mert nincs szükség semmilyen átalakításra.
- Ha (s) string = „próba” és (t) string = „prba”, akkor LD (s, t) = 1, mert (s) string-ből ki kell törölni az „ó” betűt, hogy azonos legyen a (t) string-el.

## 3. A LEVENSHTEIN TÁVOLSÁG ALKALMAZÁSAI:

- **Helyesírás ellenőrzése (fuzzy)** Ha egy szöveg egy olyan szót tartalmaz, amelyik nem található meg az alkalmazás szótárában, akkor az alkalmazás javasolni fogja a legkisebb Levenshtein

[⊗] Szaklektorált cikk. Leadva: 2011. május 15. Elfogadva: 2011. szeptember 15.
Reviewed paper. Submitted: 15 May 2011. Accepted: 15 September 2011.
Lektorálta: Prof. Dr. POKORÁDI László / Reviewed by Prof. Dr. László POKORÁDI

távolsággal rendelkező szót (vagy szavakat) az eredeti szóhoz képest. A hétköznapi életben a leggyakoribb gépelési hiba a szavakon belüli betűsorrend felcserélése. E hibáknak a klasszikus kijavítása egy törlés és egy beszúrás kombinációja. Ez könnyen optimálható egyetlen egy műveletre.

- **File-ok verziókezelése.** UNIX rendszer alatt létezik a diff f1 f2 parancs, amelyik megtalálja két file között a különbségeket, és létrehoz egy konvertáló script-et f1 és f2 között.
- **DNS analízis** A Levenshtein távolság hozzávetőleges eredményeket ad két szekvencia hasonlóságáról. Ugyanezt a módszert használják két DNS vagy fehérje szekvencia közötti hasonlóság meghatározására, amit különböző célokra használnak fel ( pl. rokonsági fokozat meghatározása vagy evolúciós családfa elemzése).
- **a plagizálás észlelése**. A Leveshtein távolság alkalmazásával felfedezhető két szöveg közötti feltűnő hasonlóság.
- **Hangfelismerés** Egyes hangfelismerő rendszerek ugyanezt a módszert használják a meglévő hangminta és az új hangminta összehasonlítására.

## 4. AZ ALGORITMUS

### 1. Lépés

Legyen n az s sorozat hossza.
Legyen m a t sorozat hossza.
ha n == 0, akkor LD(s,t) = m, kilép
ha m == 0, akkor LD(s,t) = n, kilép
létrehozunk egy M(m,n) mátrixot  n sorral és m oszloppal.

### 2. Lépés

Inicializáljuk az első sort 0..m értékekkel, és
hasonlóan inicializáljuk az első oszlopot 0..n értékekkel.

### 3. Lépés

összehasonlítunk minden s[i], i=1..m karaktert.

### 4. Lépés

minden t[j], j=1..m karakterrel.

### 5. Lépés

Ha s[i] == t[j], akkor cost = 0, másképp cost = 1

### 6. Lépés

M[i,j] = min (a, b, c ), ahol:
a = M[i-1, j] + 1, felső tag + 1, ez a törlés
b = M[i,j-1] + 1, baloldali tag + 1, a beszúrás
c=  M[i-1,j-1] + cost, felső baloldali tag + cost, a helyettesítés

### 7. Lépés

Miután a 3, 4, 5 es 6 -ik lépések iterációi véget értek, az eredményt (LD(s,t)) megtalálhatjuk a mátrix utolsó cellájában az az M[n,m]

## 5. EGY C# ALKALMAZÁS LEÍRÁSA

The LevenshteinClass:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LevenshteiDistance
{
    public class Distance
    {
        // Kiszamoljuk a Levenshtein tavolsagot
        public int LD(string s, string t)
        {
            int n = s.Length; // s sorozat hossza
            int m = t.Length; // t sorozat hossza
            int[,] d = new int[n + 1, m + 1]; // a matrix
            int cost; // cost

            // 1 lépés
            if (n == 0) return m;
            if (m == 0) return n;

            // 2 lépés
            for (int i = 0; i <= n; i++)
                d[i, 0] = i;
            for (int j = 0; j <= m; j++)
                d[0, j] = j;

            // 3 lépés
            for (int i = 1; i <= n; i++)
            {

                // 4 lépés
                for (int j = 1; j <= m; j++)
                {

                    // 5 lépés
                    cost = (t.Substring(j - 1, 1) == s.Substring(i - 1, 1) ? 0 : 1);

                    // 6 lépés
                    d[i, j] = System.Math.Min(System.Math.Min(d[i - 1, j] + 1, d[i, j
- 1] + 1), d[i - 1, j - 1] + cost);
        // d[i - 1, j] + 1 törlés
        // d[i, j - 1] + 1 beszúrás
        // d[i - 1, j - 1] + cost a helyettesítés
                }
            }

            // lasd a matrixot
            for (int i = 0; i <= n; i++)
            {
                for (int j = 0; j <= m; j++)
                {
                    Console.Write(d[i, j]);
                    Console.Write(„ „);
                }
                Console.WriteLine();
            }
```

```
            // 7 lépés
            return d[n, m];
        }
    }
}
```

Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LevenshteiDistance
{
    class Program
    {
        static void Main(string[] args)
        {
            string s, t;
            Distance d;
            d = new Distance();
            s = Console.ReadLine();
            t = Console.ReadLine();
            Console.WriteLine(d.LD(s, t));
            Console.ReadLine();
        }
    }
}
```

## 6. EGY EGYSZERŰ PÉLDA

A következő példában végig követjük a Levenshtein módszerrel hogyan jutunk el s=„MARIA” sortól a t=„MARINA” sorig.

1. és 2. lépések:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | | | | | | |
| A | 2 | | | | | | |
| R | 3 | | | | | | |
| I | 4 | | | | | | |
| A | 5 | | | | | | |

1. Táblázat

Lépések 3-tól 6-ig, ahol i = 1:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | | | | | | |
| R | 3 | | | | | | |
| I | 4 | | | | | | |
| A | 5 | | | | | | |

2. Táblázat

Lépések 3-tól 6-ig, ahol i = 2:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| R | 3 | | | | | | |
| I | 4 | | | | | | |
| A | 5 | | | | | | |

3. Táblázat

Lépések 3-tól 6-ig, ahol i = 3:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| R | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| I | 4 | | | | | | |
| A | 5 | | | | | | |

4. Táblázat

Lépések 3-tól 6-ig, ahol i = 4:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| R | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| I | 4 | 3 | 2 | 1 | 0 | 1 | 2 |
| A | | | | | | | |

5. Táblázat

Lépések 3 -tól 6 -ig, ahol i = 5:

| | | M | A | R | I | N | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| M | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| R | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| I | 4 | 3 | 2 | 1 | 0 | 1 | 2 |
| A | 5 | 4 | 3 | 2 | 1 | 1 | 1 |

6. Táblázat

Az eredmény a mátrix utolsó sorának utolsó oszlopában található, vagyis a két sor közötti távolság az egyenlő 1-el.

Ugyanerre az eredményre jutunk, ha intuitív módszerrel oldjuk meg a problémát, vagyis az „N” betű beszúrásával az eredeti “MARIA” sor utolsó előtti pozíciójára a “MARINA” szót kapjuk. Tehát egy beszúrási művelet történt, így elmondhatjuk, hogy a Levenshtein távolság a két sor között az 1. Következtetésképp elmondhatjuk, hogy minél nagyobb eredményhez jutunk, annál nagyobb a két sor közötti különbség.

## 7. ÖSSZEFOGLALÁS

Az előző fejezetekben a Levenshtein távolság egy egyszerű és könnyen megközelíthető leírására vállalkoztunk, így lehetőséget nyújtva számos gyakorlati probléma optimált megoldásához.
A fentiekben bemutatott algoritmus, és C# programot további optimálásoknak lehet alávetni, és parallelizálása is elképzelhető további időnyereség eléréséhez.

A későbbiekben egy parallelizált implementációt szeretnénk létrehozni és lemérni. Az így elért időnyereséget több konfigurációjú számítógépen (például egy, két, három illetve négy magos {processzor} számítógépen).

További cél lehet egy matematikailag optimált fuzzy távolság létrehozása.

## 8. FELHASZNÁLT IRODALOM:

[1] http://en.wikipedia.org/wiki/Levenshtein_distance
[2] http://www.levenshtein.net/levenshtein_demos.htm
[3] Appeared in English as:Levenshtein VI (1966). „Binary codes capable of correcting deletions, insertions, and reversals”. *Soviet Physics Doklady* **10**: 707–10. В.И. Левенштейн (1965). „Двоичные коды с исправлением выпадений, вставок и замещений символов”. *Доклады Академий Наук СССР* **163** (4): 845–8.
[4] http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit/
[5] http://www.merriampark.com/ld.htm