

INFORME PROYECTO ALM

December 29, 2020

Victor Callejas Fuentes
4CO11
viccalfu@inf.upv.es

David Alarcón
4CO11
viccalfu@inf.upv.es

Jose Mira
4CO11
jomigar4@inf.upv.es

Abstract

En este documento se recoge el código desarrollado así como los resultados obtenidos y decisiones tomadas durante la realización de las prácticas de laboratorio.

Contents

1 Tarea 1 - Distancias de edición	3
1.1 Distancia de Levensthtein	3
1.2 Distancia de Damerau-Levensthtein restringida	3
1.3 Distancia de Damerau-Levensthtein intermedia	4
1.4 Testing	4
2 Distancias de edición con thresholds	5
2.1 Calcular alrededor de la diagonal	5
2.2 Detener ejecución sino promete	5
2.3 Testing	5
3 Tarea 3 - Metodo Suggest	6
3.1 Límites teóricos superior e inferior	6
3.2 Implementación	6
3.3 Testing	6
4 Tarea 4 - Implementación mediante Tries	7
4.1 Implementacion	7
4.2 Testing	7
5 Tarea 5 - Estudio experimental	8
5.1 Condideraciones	8
5.2 Resultados	9
6 Tarea 6 - Implementación proyecto SAR	10
6.1 TO BE CONTINUED	10

1 Tarea 1 - Distancias de edición

Implementación de distancias de edición entre cadenas de forma iterativa y mediante programación dinámica

1.1 Distancia de Levensthtein

Considerando las operaciones de inserción, borrado y sustitución con coste = 1.

Listing 1: Algoritmo distancia de levenshtein

```
1  mat = matriz(x, y)
2  res = np.zeros(shape=(len(x)+1,len(y)+1))
3  for i in range(0,len(x)+1):
4      for j in range(0,len(y)+1):
5          if i==0 or j==0:
6              res[i,j] = res[i,j] + i + j
7          else:
8              res[i,j] = min(
9                  mat[i-1,j-1] + res[i-1,j-1],
10                 1 + res[i-1,j],
11                 1 + res[i,j-1]
12             )
13
14  return res[len(x),len(y)]
```

1.2 Distancia de Damerau-Levensthtein restringida

Se añade la operación de trasposición. En esta versión una vez intercambiados dos símbolos, éstos no se pueden utilizar en otras operaciones de edición.

Listing 2: Sample Python code – Damerau-Levensthtein restringido

```
1  INF = len(x) + len(y)
2
3  mat = matriz(x, y)
4  res = np.zeros(shape=(len(x)+1,len(y)+1))
5
6  for i in range(0,len(x)+1):
7      for j in range(0,len(y)+1):
8          if i==0 or j==0:
9              res[i,j] = res[i,j] + i + j
10         elif i == 1 or j == 1:
11             res[i,j] = min(
12                 mat[i-1,j-1] + res[i-1,j-1],
13                 1 + res[i-1,j],
14                 1 + res[i,j-1],
15             )
16         else:
17             res[i,j] = min(
18                 mat[i-1,j-1] + res[i-1,j-1],
19                 1 + res[i-1,j],
20                 1 + res[i,j-1],
21                 1 + res[i-2,j-2] + (mat[i-2,j-1] + mat[i-1,j-2]) * INF
22             )
23  return res[len(x),len(y)]
```

1.3 Distancia de Damerau-Levensthtein intermedia

Considerando las operaciones de trasposición cuando:

$$|u| + |v| \leq cte \Leftrightarrow cte = 1 \quad (1)$$

Listing 3: Damerau-Levensthtein intermedio

```
1  M = init_matriz(x, y)
2
3  for i in range(1, len(x) + 1):
4      for j in range(1, len(y) + 1):
5
6          if x[i - 1] == y[j - 1]:
7              initActual = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1])
8          else:
9              initActual = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1] + 1)
10
11         if j > 1 and i > 1 and x[i - 2] == y[j - 1] and x[i - 1] == y[j - 2]:
12             M[i,j] = min(initActual, M[i-2][j-2] + 1)
13         elif j > 2 and i > 1 and x[i-2] == y[j-1] and x[i-1] == y[j-3]:
14             M[i,j] = min(initActual, M[i-2][j-3] + 2)
15         elif i > 2 and j > 1 and x[i - 3] == y[j-1] and x[i-1] == y[j-2]:
16             M[i,j] = min(initActual, M[i-3][j-2] + 2)
17         else:
18             M[i,j] = initActual
19
20     return M[len(x), len(y)]
```

1.4 Testing

En el código se encuentra una test flag, por defecto activada, para que a la hora de importar este módulo se compruebe que el comportamiento de los algoritmos es el adecuado.

2 Distancias de edición con thresholds

En orden de optimizar el algoritmo se han implementado dos mejoras sobre los tres métodos de la tarea 1.

2.1 Calcular alrededor de la diagonal

Establecemos límites en el recorrido para que solamente se calculen aquellas partes del grafo de dependencias que tengan sentido para dicho umbral. Es decir, zonas relativamente cercanas a la diagonal principal de la matriz. En nuestro caso, lo realizamos por columnas sobre la matriz.

Listing 4: Calculo de los umbrales

```
1 lower_y = max(1, i - threshold)
2 upper_y = min(len(y), i + threshold)
```

2.2 Detener ejecución sino promete

Detenemos el algoritmo si, tras calcular una etapa en nuestro caso por columnas, se puede asegurar que el coste superará el umbral.

Listing 5: Parar ejecución sino promete

```
1 if min(M[i,:]) > threshold :
2     return threshold + 1
```

2.3 Testing

Al igual que en la tarea 1, en el código se encuentra una test flag, por defecto activada, para que a la hora de importar este módulo se compruebe que el comportamiento de los algoritmos es el adecuado.

3 Tarea 3 - Metodo Suggest

Este método, iterativo, compara cada palabra con todas las del diccionario, ello conlleva un gran coste temporal. Para ello se pide implementar alguna cota optimista que nos permita saber que dicha distancia será mayor al umbral establecido.

3.1 Límites teóricos superior e inferior

La distancia de Levenshtein tiene varios límites superior e inferior simples. Éstas incluyen:

1. Es al menos la diferencia de tamaños de las dos cuerdas.
2. Es como máximo la longitud de la cuerda más larga.
3. Es cero si y solo si las cadenas son iguales.
4. Si las cuerdas son del mismo tamaño, la distancia de Hamming es un límite superior en la distancia de Levenshtein. La distancia de Hamming es el número de posiciones en las que los símbolos correspondientes en las dos cadenas son diferentes.
5. La distancia entre dos cadenas Levenshtein no es mayor que la suma de sus distancias levenshtein de una tercera cadena (desigualdad triangular).

3.2 Implementación

Listing 6: Condición 1

```
1 if threshold is not None:
2     lower = abs(length - len(word))
3     if lower > threshold:
4         continue
```

Listing 7: Condición 3

```
1 if term == word:
2     results[word] = 0
```

3.3 Testing

Ejécutamos nuestros algoritmos de distancias con los mismos parámetros que el profesor y comparamos los resultados, cerciorandonos de que son iguales.

4 Tarea 4 - Implementación mediante Tries

En esta parte, extendemos los algoritmos de cálculo de distancias que originalmente trabajan con pares de cadenas, para calcular la distancia entre una cadena y un trie

4.1 Implementacion

Yo tengo poca idea de esta parte

4.2 Testing

Al igual que en la tarea 3, ejecutamos nuestros algoritmos de distancias, esta vez implementados mediante la estructura de datos Trie, con los mismos parámetros y comparamos los resultados, cerciorandonos de que son iguales que los de la tarea 3 y los del profesor.

5 Tarea 5 - Estudio experimental

En esta parte se pide realizar un estudio experimental de medidas de tiempo para determinar qué versiones de las anteriores son las más eficientes para unos datos concretos

5.1 Condideraciones

Utilizamos dos diccionarios (castellano e inglés) ambos contruidos mediante el texto de la declaración de derechos humanos.

Sobre estos creamos múltiples diccionarios con las $N([1000, 5000, 10000])$ palabras más frecuentes. Para ello sobrecargamos el constructor de la clase Suggest de maneara que acepte un path al corpus y un set de palabras ya creado.

Sobre cada uno de estos se crean N consultas $([10, 100, 500, 1000, 5000, 10000])$. Estas son palabras pertenecientes al deccionario sobre las que se realizan unas perturbaciones aleatorias.

Listing 8: Función para crear perturbaciones aleatorias

```
1 def perturbar(word):
2
3     word = list(word)
4
5     n_ops = random.randint(0, min(len(word)-1, MAX_PERTURBACIONES))
6
7     for _ in range(0, n_ops):
8
9         op = random.randint(0, 3)
10
11         if op == 0: # borrar
12             idx = random.randint(0, len(word)-1)
13             word = word[:idx] + word[(idx+1):]
14         elif op == 1: # cambiar
15             idx = random.randint(0, len(word)-1)
16             char = chr(random.randint(ord('a'), ord('z')+1))
17             word[idx] = char
18         elif op == 2: # trasposicion
19             idx = random.randint(0, len(word)-2)
20             tmp = word[idx]
21             word[idx] = word[idx+1]
22             word[idx+1] = tmp
23
24     return "".join(word)
```

Estas consultas se realizan con todos los algoritmos desarrollados en las tareas previas.

Utilizamos el mismo diccionario y consultas con todos los algoritmos (datos apareados).

Repetimos 3 veces la medición de tiempo de cada algoritmo para cada talla de diccionario y consultas para luego poder explorar los resultados y ver que son consistentes y válidos.

Listing 9: medición de tiempos

```
1     start = time.time()
2
3     for consulta in tqdm(consultas, total=len(consultas), leave=False, desc='Consultas: ', position=4):
4         _ = iss.suggest(consulta, distance=alg, threshold=STATIC_THRESHOLD)
5
6     end = time.time()
7     elapsed = end - start
```


5.2 Resultados

6 Tarea 6 - Implementación proyecto SAR

6.1 TO BE CONTINUED

wqrteqwe