INFORME PROYECTO ALM

January 3, 2021

Victor Callejas Fuentes	David Alarcón	Jose Mira
4CO11	4CO11	4CO11
viccalfu@inf.upv.es	viccalfu@inf.upv.es	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 práctias de laboratorio.

Contents

1 Tarea 1 - Distancias de edición

Implementaión 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

```
    mat = matriz(x, y)
    res = np.zeros(shape=(len(x)+1,len(y)+1))
    for i in range(0,len(x)+1):
        for j in range(0,len(y)+1):
                if i==0 or j==0:
                    res[i,j] = res[i,j] + i + j
                else:
                    res[i,j] = min(
                        mat[i-1,j-1] + res[i-1,j-1],
                        1 + res[i-1,j],
                        1 + res[i,j-1]
                    )

    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

```
    INF = len(x) + len(y)

    mat = matriz(x, y)
    res = np.zeros(shape=(len(x)+1,len(y)+1))

    for i in range(0,len(x)+1):
        for j in range(0,len(y)+1):
                if i==0 or j==0:
                    res[i,j] = res[i,j] + i + j
                elif i == 1 or j == 1:
                    res[i,j] = min(
                        mat[i-1,j-1] + res[i-1,j-1],
                        1 + res[i-1,j],
                        1 + res[i,j-1],
                    )
                else:
                    res[i,j] = min(
                        mat[i-1,j-1] + res[i-1,j-1],
                        1 + res[i-1,j],
                        1 + res[i,j-1],
                        1 + res[i-2,j-2] + (mat[i-2,j-1] + mat[i-1,j-2]) * INF
                    )
    return res[len(x),len(y)]
```

1.3 Distancia de Damerau-Levensthtein intermedia

Considerando las operaciones de trasposición cuando:

$$|u| + |v| \leqslant cte \Leftarrow cte = 1 \tag{1}$$

Listing 3: Damerau-Levensthtein intermedio

```
    M = init_matriz(x, y)

    for i in range(1, len(x) + 1):
        for j in range(1, len(y) + 1):

            if x[i - 1] == y[j - 1]:
                initActual = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1])
            else:
                initActual = min(M[i-1, j] + 1, M[i, j-1] + 1, M[i-1][j-1] + 1)

            if j > 1 and i > 1 and x[i - 2] == y[j - 1] and x[i - 1] == y[j - 2]:
                M[i,j] = min(initActual, M[i-2][j-2] + 1)
            elif j > 2 and i > 1 and x[i-2] == y[j-1] and x[i-1] == y[j-3]:
                M[i,j] = min(initActual, M[i-2][j-3] + 2)
            elif i > 2 and j > 1 and x[i - 3] == y[j-1] and x[i-1] == y[j-2]:
                M[i,j] = min(initActual, M[i-3][j-2] + 2)
            else:
                M[i,j] = initActual

    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 algortimos es el adecuado.

2 Tarea 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

```
lower_y = max(1, i - threshold)
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

```
if min(M[i,:]) > threshold :
    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 algortimos 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

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

Listing 7: Condición 3

```
if term == word:
    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, ejécutamos nuestros algoritmos de distancias, esta vez implementados mediante la esdtructura 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([20,100,500,2500]) 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([10,50,250,500,2500]). Estas son palabras pertenecientes al diccionario sobre las que se realizan unas perturbaciones aleatorias.

Listing 8: Función para crear perturbaciones aleatorias

```
def perturbar(word):

    word = list(word)

    n_ops = random.randint(0, min(len(word)-1,MAX_PERTURBACIONES))

    for _ in range(0,n_ops):

        op = random.randint(0, 3)

        if op == 0: # borrar
            idx = random.randint(0,len(word)-1)
            word = word[:idx] + word[(idx+1):]
        elif op == 1: # cambiar
            idx = random.randint(0,len(word)-1)
            char = chr(random.randint(ord('a'),ord('z')+1))
            word[idx] = char
        elif op == 2: # trasposicion
            idx = random.randint(0,len(word)-2)
            tmp = word[idx]
            word[idx] = word[idx+1]
            word[idx+1] = tmp

    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

```
    start = time.time()

    for consulta in tqdm(consultas,total=len(consultas),leave=False,desc='Consultas: ',position=4):
        _ = iss.suggest(consulta,distance=alg,threshold=STATIC_THRESHOLD)

    end = time.time()
    elapsed = end - start
```

5.2 Resultados

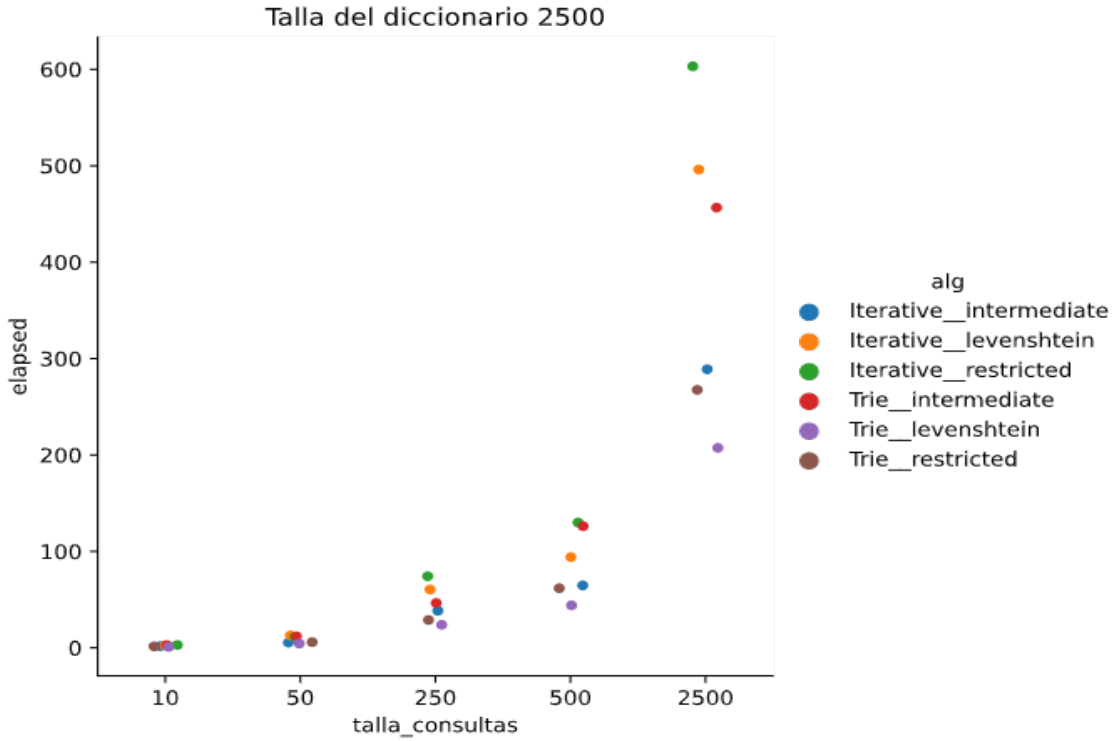
Una vez ejecutado el programa y realizado la medición de tiempos observamos que los tiempos medidos en las diferentes repeticiones con los mismos datos son parecidos, para ello obtenemos la media y desviaciones estándar.

Por ejemplo: para inglés, con talla de diccionario 20, talla de consultas 10 y el algoritmo Iterative intermediate obtenemos una media de 0.028734 y desviación estándard de 0.000572. Por lo que concluimos que los datos son válidos y procedemos a realizar la media de las diferentes repeticiones.

Realizamos el mismo procedimiento para los dos idiomas, inglés y castellano, observamos que el tiempo medidio con los mismos parametros son parecidos. Por lo tanto concluimos que los diferentes algoritmos se comportan igual en ambos idiomas y procedemos a realizar la media de ambas mediciones.

Finalmente, obtenemos para cada algoritmo una medición para cada talla de diccionario y talla de consulta.

Los resultados obtenidos muestran que todos los algoritmos escalan de manera lineal con la talla de consulta y de diccionario, y que el algoritmo más rápidos es Trie Levenhtein, seguido de Trie restricted.



6 Tarea 6 - Implementación proyecto SAR

6.1 TO BE CONTINUED

wqrteqwe