INFORME PROYECTO ALM

January 5, 2021

Victor Callejas	David Alarcón	Jose Mira
4CO11	4CO11	4CO11
viccalfu@inf.upv.es	daalse1@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

Realizado por José Mira

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

Listing 1: Algoritmo distancia de levenshtein

```
    def dp_levenshtein_backwards(x, 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
                    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

Realizado por David Alarcón

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

```
    def dp_restricted_damerau_backwards(x, y):
        # Simula el infinito
        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

Realizado por Víctor Callejas

Considerando las operaciones de trasposición cuando:

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

Listing 3: Damerau-Levensthtein intermedio

```
    def dp_intermediate_damerau_backwards(x, y):
        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, mediante los test proporcionados en las diapositivas, sino mostratá los test case en los que falla.

2 Tarea 2 - Distancias de edición con thresholds

Realizado por David Alarcón, Víctor Callejas y José Mira

Para 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, recorremos la matriz por filas.

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 filas, 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, sino mostratá los test case en los que falla.

3 Tarea 3 - Metodo Suggest

Realizado por David Alarcón, Víctor Callejas y José Mira

Este método, se compara cada palabra con todas las del diccionario, ello conlleva un gran coste temporal. Por ello se pide implementar alguna cota optimista que nos permita saber cuando una 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

Nosotros hemos decidido implementar las condiciones 1 y 3. Las condiciones 2 y 4, no las hemos implementado ya que estás no son cotas optimistas. La condición 5 tampoco, ya que conlleva un sobrecoste en el cálculo de la distancia con la tercera palabra.

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

Realizado por David Alarcón

Hemos modificado el archivo de leer_resultados.py lo que nos permite comparar los resultados proporcionados por el profesor con los resultados obtenidos por nuestro suggester.

Para ejecutar los test desde el directorio tarea 3 ejecutamos:

```
$ python leer_resultados.py
```

Además se permite especificar por parametros la distancia a usar *levenshtein*, *restricted* o *intermediate* y si queremos que sea no verbos *-nv*.

4 Tarea 4 - Implementación mediante Tries

Realizado por David Alarcón, Víctor Callejas y José Mira

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

En primer lugar hemos modifcado el metodo usado para inicializar la matriz ya que al tratrse de un trie el orden que este seguía no era igual al número de la columna en el que se encontraba por lo que había que recorrer todos los estados añadiendo su valor inicial a la matriz.

Listing 8: Método init_matriz_trie en */utils/utils.py*

```
def init_matriz_trie(term_trie, y):
    """
    Inicia la matriz para calculos con Tries
    """
    M = np.ones((term_trie.get_num_states(), len(y) + 1)) * np.inf

    M[0] = np.arange(len(y)+1)
    for i in range(1, term_trie.get_num_states()):
        M[i, 0] = M[term_trie.get_parent(i), 0] + 1

    return M
```

Internamente en las distancias nos hemos basado en reescribir las distancias de la tarea 4 usando la lógica del Trie.

- Para recoger todos los estados usamos *term_trie.get_num_states()*
- Para acceder al elemento *i-1* accedemos a *term_trie.get_parent(i)*
- Para verificar que es un estado final usamos *term_trie.is_final(i)*

Además en la distancia de leveshtein hemos añadido una optimización en la que si el valor del padre supera el threshold continuamos la ejecuciaón evitando ejecutar j iteraciones.

Listing 9: Optimización levenshtein tipo trie

```
if min(res[term_trie.get_parent(i), :]) > threshold:
    continue
```

4.2 Testing

Realizado por David Alarcón

Al igual que en la tarea 3, hemos modiciado el archivo leer_resultados.py con el objetivo de comprobar que verificar que los resultados obtenidos sean los mismos que los del profesor. A la hora de testear se siguen los mismo patrones que en la tarea 3

5 Tarea 5 - Estudio experimental

Realizado por Víctor Callejas

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 10: 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 11: 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

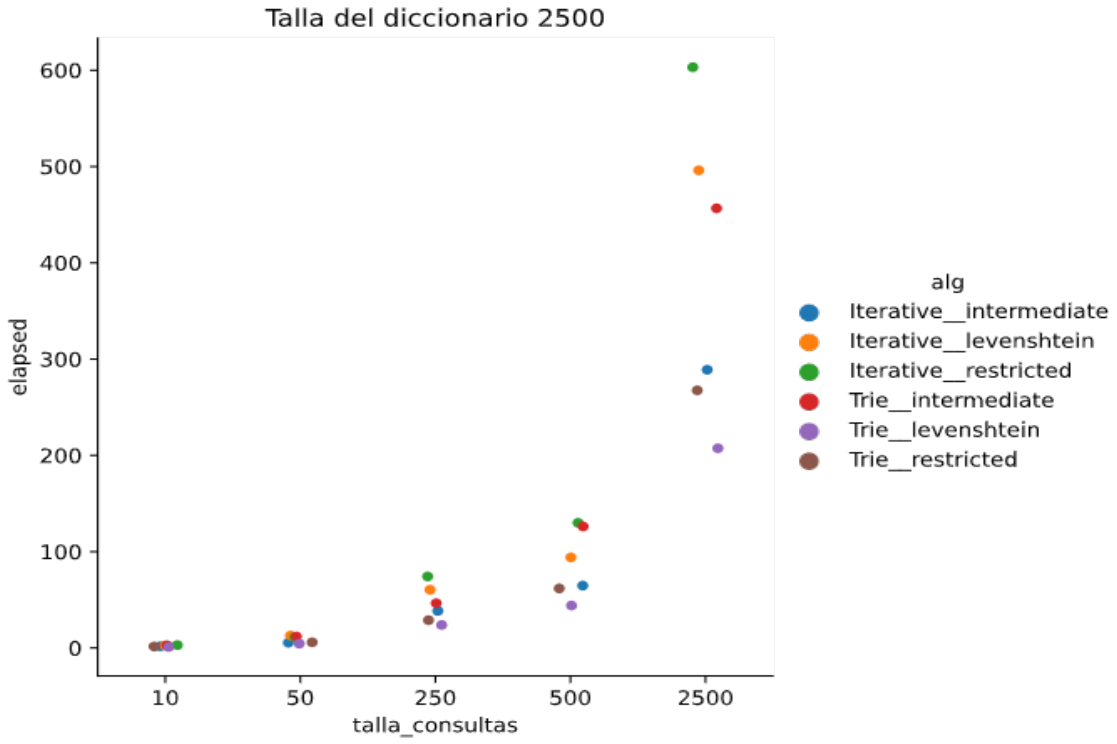
Una vez ejecutado el programa y realizado la medición de tiempos observamos que los tiempos medidos en las diferentes repeticiones con los mismos datos son parecidos, para ello obtenemos la media y desviaciones estándar.

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 los algoritmos más rápidos son Trie Levenhtein, seguido de Trie restricted.



6 Tarea 6 - Integración proyecto SAR

Realizado por David Alarcón, Víctor Callejas y José Mira

6.1 Implementación

Hemos creado un método al cual se le pasa el índice de noticias, un término, una distancia y un threshold. Este método devuelve la lista de indices de noticias en el que aparecen los términos que cumplen las condiciones de distancia especificadas, en base al termindo de búsqueda.

Listing 12: Método buscar índices de noticias con palabras aproximadas

```
def buscar_aproximados(index, term, distancia, threshold):
    suggesterObject = suggester(list(index.keys()))
    return list(dict.fromkeys(
        itertools.chain.from_iterable(
        [index[word] for word, _ in suggesterObject.suggest(term, distancia, threshold).items() ]
            )
        ))
```

Para que esto funcione con nuestro searcher hemos añadido un try catch a la hora de acceder al índice de una palabra si se produce un KeyError esto significa que el término no está en el índice por lo que hemos de recurrir a sus terminos sugeridos en base a la distancia especificada.

Listing 13: Método buscar índices de noticias con palabras aproximadas

```
try:
    l1 = index[term]
except KeyError:
    l1 = buscar_aproximados(index, term, distancia, threshold) if aproximada else  []
```

El funcionamiento de nuestro suggester puede variar dependiendo de los parámetros que se le pasen a este.

- *–index_file* El index file a usar.
- *–ql_file* El nombre del arcivo que contiene la lista de queries que queremos ejecutar.
- *–distancia* La distancia que se quiere usar por defecto se usa levenshtein ya que es con la que mejor resultados hemos obtenido en la tarea 5.
- *–threshold* El threhold que se desea usar.
- *–query* La query que se le desea pasar al buscador en caso de que se prediera pasar una query en vez de un archivo de queries.

Todas las distancias de nuestro buscador se ejecutan con la configuración de tipo Trie ya que han sido las que mejor resultados nos han dado en la tarea5.

6.2 Testing

Realizado por David Alarcón

Se incluye un archivo tester.py el cual contrasta los resultados obtenidos por nuestro searcher con los resultados referncia de poliformat.

Para ejecutar los test desde el directorio tarea 6 ejecutamos:

```
$ python tester.py
```

Si este no printea nada por consola indica que ha ejecutado todos los tests de forma correcta.