

Palabra de Menor Distancia

Profesor:

Cuadrado Estrebou, María Fernanda

Grupo:

Rosso, Máximo - LU:1121122

Sánchez Paredes, Diego Andrés – LU: 1121456

Stella, Gabriel - LU: 1118523

Taborda, Franco - LU:1092629

Buenos Aires, 16 de Junio de 2021.-

Tabla de Contenidos

Introducción	3
Descripción del Problema	3
Estrategia de Resolución	3
Pseudocódigo del Algoritmo	4
Análisis de Complejidad Temporal	7
Conclusiones	8

Introducción

En algunas ocasiones cometemos errores de tipeo, ya sea por apuro, distracción o simplemente preferimos acortar una palabra para lograr mandar un mensaje corto.

Por esta razón se diseñó un algoritmo que logre comparar las palabras con errores de tipeo con la palabra bien escrita. Dicho algoritmo devolverá la palabra con menos diferencias con respecto a la original y la cantidad de diferencias que tiene.

Descripción del Problema

El problema por resolver consiste en que dada una palabra “X” de entrada y una lista de palabras “Y” determinar cuál palabra de la lista “Y” es la que menos diferencias tiene con la palabra “X” y cuantas diferencias tiene.

Para la palabra más cercana mostrar el número mínimo de sustituciones, inserciones o eliminaciones que se deben llevar a cabo para transformarla en “X”.

Estrategia de Resolución

El método a utilizar es Backtracking.

En primer lugar, vamos a recorrer la colección de palabras y llamar calcular la distancia con la palabra origen, pasandole además cuál es el valor menor de modificaciones necesarias calculado hasta el momento. Para la primera palabra le pasaremos un valor muy alto.

Empezando por el primer nivel, comparamos las letra de ambas palabras. En caso de que sea igual, vamos a avanzar al siguiente nivel.

Si son diferentes, vamos a abrir en tres opciones posibles:

- Reemplazar la letra por la correspondiente de la original
- Ingresar la misma letra
- Eliminar la letra equivocada

Cada una de las ramas va a devolvernos cuantas modificaciones son necesarias de realizar y nos quedaremos con la menor.

En caso de superar el valor máximo actual, se realiza la poda de esa rama y no se prosigue el cálculo.

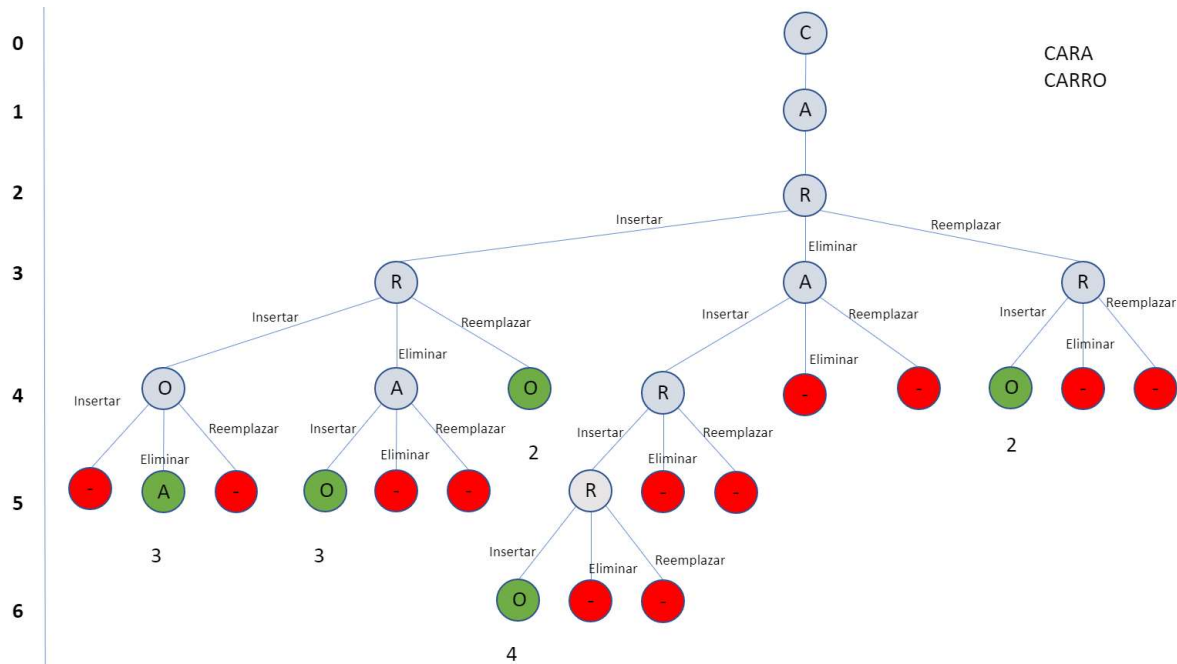


Figura 1 - Diagrama de estrategia.

En la *Figura 1* tenemos una representación gráfica de la estrategia utilizada. En esta se ven todas las modificaciones que *pueden* ser realizadas hasta igualar las palabras. En nuestra implementación al detectar que las modificaciones por una rama serían mayores al mejor resultado obtenido hasta el momento, se realiza una **poda**, reduciendo así la cantidad de operaciones considerablemente.

Pseudocódigo del Algoritmo de Resolución del Problema

Algoritmo CalcularDistancia

Entrada: origen: string, candidatas: vector <string>

Salida: MejorDistancia (Objeto con la mejor distancia y palabra correspondiente)

```

resultado ← nuevo MejorDistancia()
resultado.diferencia ← infinito
para cada candidata perteneciente a candidatos // O(n)
    nivel ← 0
    diferenciaActual ← 0
  
```

```
    resultadoComparacion ← DiferenciaEntreDos(origen, candidata,
nivel, resultado.diferencia, diferenciaActual)
    si resultadoComparacion < resultado.diferencia
        resultado.diferencia ← resultadoComparacion
        resultado.palabra ← candidata
    fin si
fin para

    retorna resultado
fin CalcularDistancia
```

Algoritmo DiferenciaEntreDos

Entrada: base: string, objetivo: string, nivel: entero,
diferenciaMinima: entero, diferenciaActual: entero
salida: entero (cantidad de modificaciones)

```
    si nivel ≥ max(longitud(base), longitud(objetivo)) //Caso base 0(1)
        retorna diferenciaActual
    sino si caracterEsIgual(base, objetivo, nivel) // Pasamos al sig.
nivel de una vez
        retorna DiferenciaEntreDos(base, objetivo, nivel+1,
diferenciaMinima, diferenciaActual)
    sino si diferenciaActual+1 ≥ diferenciaMinima // Poda 0(1)
        retorna diferenciaMinima
    sino
        diferenciaActual ← diferenciaActual + 1

    si nivel < longitud(base)
        string baseEliminando ← eliminaCaracter(base, nivel) // 0(1)
        entero difEliminando ← DiferenciaEntreDos(baseEliminando,
objetivo, nivel, diferenciaMinima, diferenciaActual)
        diferenciaMinima ← min(diferenciaMinima, difEliminando) //
Por si tenemos una diferencia mas corta 0(1)
    fin si
    si nivel < longitud(objetivo)
        string baseInsertando ← insertaCaracter(base, objetivo,
nivel) // 0(1)
        entero difInsertando ← DiferenciaEntreDos(baseInsertando,
objetivo, nivel+1, diferenciaMinima, diferenciaActual)
        diferenciaMinima ← min(diferenciaMinima, difInsertando)
```

```
    fin si
    si nivel < longitud(base) Y nivel < longitud(objetivo)
        string baseReemplazando ← reemplazaCaracter(base, objetivo,
nivel) // 0(1)
        entero difReemplazando ← DiferenciaEntreDos(baseReemplazando,
objetivo, nivel+1, diferenciaMinima, diferenciaActual)
        diferenciaMinima ← min(diferenciaMinima, difReemplazando)
    fin si

    retorna diferenciaMinima
fin si
fin DiferenciaEntreDos
```

```
Algoritmo caracterEsIgual // 0(1)
Entrada: cadena1; string, cadena2: string, indice: entero
Salida boolean
    si indice < min(longitud(cadena1), longitud(cadena2))
        retorna (cadena1[nivel] es igual a cadena2[nivel])
    sino
        retorna falso
    fin si
fin caracterEsIgual
```

```
Algoritmo reemplazaCaracter // 0(1)
Entrada: base: string, objetivo: string, indice: entero
Salida: string
    retorna base.subcadena(0, indice-1) + objetivo[indice] +
base.subcadena(indice+1, longitud(base)-1)
fin reemplazarCaracter
```

```
Algoritmo eliminaCaracter // 0(1)
Entrada: base: string, indice: entero
Salida: string
    retorna base.subcadena(0, indice-1) + base.subcadena(indice+1,
longitud(base)-1)
fin eliminaCaracter
```

```

Algoritmo insertaCaracter // 0(1)
Entrada: base: string, objetivo: string, indice: entero
Salida: string
    retorna base.subcadena(0, indice) + objetivo[indice] +
base.subcadena(indice+1, longitud(base)-1)
fin insertaCaracter

```

Análisis de Complejidad Temporal

Para **caracterEsIgual**, **reemplazaCaracter** y **eliminaCaracter**

Todas las operaciones que realizan estos algoritmos son de costo temporal constante, independiente del tamaño de las entradas, por lo tanto los tres algoritmos son $\Theta(1)$.

Para **DiferenciaEntreDos**

Es una función recursiva donde sus entradas disminuyen por forma de sustracción, siempre reduciendo la entrada de a 1, en nuestro caso representa la evaluación de un carácter menos y en el peor de los casos se puede realizar un llamado recursivo hasta 3 veces.

Todas las operaciones realizadas además de los llamados recursivos son de complejidad temporal constante.

De lo anterior obtenemos que **a=3**, **b=1** y **k=0**.

Por los casos de sustracción en la *Figura 2*.

$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < 1 \\ \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n \div b}) & \text{si } a > 1 \end{cases}$$

Figura 2. Cálculo de complejidad temporal para funciones recursivas que disminuyen por sustracción.

Vemos que entramos por el caso de $\Theta(a^{n \div b})$ ya que **a=3 > 1**. Por lo tanto la complejidad temporal del algoritmo es $\Theta(3^{n+1}) = \Theta(3^n)$ donde **n** es la longitud mayor entre las cadenas de entrada **base** y **objetivo**.

Para CalcularDistancia

El ciclo principal tiene una complejidad de $\Theta(n)$ donde **n** es la cantidad de palabras candidatas más el llamado a la función **DiferenciaEntreDos** cuya complejidad temporal es de $\Theta(3^n)$. Por lo tanto el algoritmo tiene una complejidad temporal de $\Theta(n3^m) = \Theta(n^m)$ donde **n** es la cantidad de palabras candidatas y **m** es la longitud máxima entre las palabras candidatas y la palabra origen.

Conclusiones

El problema implica obtener cual es la menor cantidad de modificaciones necesarias por cada palabra candidata para llegar a obtener la palabra origen.

Por esto, elegimos como estrategia utilizar backtracking, para poder generar una nueva rama por cada opción posible en caso de encontrar diferencias con la letra equivalente a cada posición de la palabra original. Luego, de cada rama podemos obtener mediante recursividad el camino más corto para finalmente obtener cual es la distancia mínima entre la palabra origen y la candidata.

Mediante la implementación en java del pseudocódigo que planteamos en base a nuestra estrategia, pudimos llevarlo a la práctica. Pudimos comprobar con los casos de prueba que efectivamente el algoritmo funciona y obtuvimos satisfactoriamente la palabra de menor distancia de la colección.