.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo práctico 2: Que parezca programación dinámica

6 de mayo de 2025

Integrantes

Alumno	Padrón
Alan Valdevenito	107585
Ignacio Zanoni Mutti	110884
Francisco Gutierrez	103543

Índice

1. Análisis del problema

Amarilla detectó que hay un soplón en la organización, que parece que se contacta con alguien de la policía. En general eso no sería un problema, ya que la mafia trabaja en un distrito donde media policía está arreglada. El problema es que no parecería ser el caso. El soplón parece estarse contactando con mensajes encriptados.

No interesa saber quién es el soplón (de momento), sino más bien qué información se está filtrando. El área de desencriptación se encargará de intentar dar posibles resultados, y nosotros debemos validar si, en principio, es un posible mensaje. Es decir, si nos dan una cadena desencriptada que diga "estanocheenelmuellealassiete", este sería un posible mensaje, el cual correspondería a "esta noche en el muelle a las siete", mientras que "estamikheestado" no lo es, ya que no podemos separar de forma de generar todas palabras del idioma español (en cambio, si fuera "estamiheestado" podría ser "esta mi he estado"). No es nuestra labor analizar si el texto tiene potencialmente sentido o no, de eso se encargará otro área.

1.1. Problema

Dado un listado P de p palabras, y una cadena desencriptada, determinar si existe un posible mensaje (es decir, se lo puede separar en palabras del idioma), o no. Y si lo es, mostrar cómo queda formado.

1.2. Ecuación de recurrencia

Proponemos la siguiente ecuacion de recurrencia:

$$OPT[i] = \bigvee_{\substack{p \in P \\ |p| \leq i}} (\text{cadena}[i - |p| : i] = p \quad \wedge \quad OPT[i - |p|])$$

con el siguiente caso base para una cadena vacia:

$$OPT[0] = \text{True}$$

Mencionamos que:

- $OPT[i]$ es verdadero si existe una forma de segmentar los primeros i caracteres como una secuencia de palabras válidas.
- P es el conjunto de palabras del diccionario.
- cadena$[i - |p| : i] = p$ significa que los últimos caracteres coinciden con una palabra válida.

1.3. Solución propuesta

Nuestra solución para el problema es un algoritmo de programación dinámica que recorre la cadena desencriptada carácter por carácter, verificando en cada paso si es posible construir un mensaje válido utilizando únicamente palabras del diccionario.

Para ello, definimos el vector de memorizacion `mem`, donde cada posición `mem[i]` almacena un par: un valor booleano que indica si es posible llegar hasta la posición i formando palabras válidas, y la última palabra utilizada para alcanzar dicha posición.

Para cada posición i de la cadena, iteramos todas las palabras del diccionario. Si alguna palabra p coincide exactamente con los últimos $|p|$ caracteres que terminan en i, y además ya era posible construir un mensaje válido hasta la posición $i - |p|$, entonces actualizamos `mem[i]` con `True` y registramos la palabra p como la última palabra utilizada.

Una vez que construimos el vector de memorizacion, al momento de reconstruir la solucion verificamos si es posible llegar hasta el final de la cadena. En caso afirmativo, reconstruimos el mensaje original recorriendo hacia atrás las palabras registradas en `mem` hasta llegar a la posición 0.

2. Demostración

- Sea $S = s_1 s_2 \dots s_n$ la cadena desencriptada de longitud n, donde cada s_i es un caracter.
- Sea $P = \{p_1, p_2, \dots, p_m\}$ el diccionario de palabras válidas.
- La ecuación de recurrencia define $OPT[i]$, que indica si la subcadena $s_1 \dots s_i$ puede separarse en una secuencia de palabras $p_{i_1}, p_{i_2}, \dots, p_{i_k} \in P$ tal que

$$p_{i_1} + p_{i_2} + \dots + p_{i_k} = s_1 \dots s_i$$

- La ecuación es:

$$OPT[i] = \bigvee_{\substack{p \in P \\ |p| \leq i}} (\text{cadena}[i - |p| : i] = p \quad \wedge \quad OPT[i - |p|]),$$

  con caso base $OPT[0] = \text{True}$.

Tesis general: La ecuación de recurrencia determina siempre, de manera correcta, si la cadena S puede separarse en una secuencia de palabras del diccionario P. Es decir:

- Si existe al menos una separación válida, la ecuación asegura que $OPT[n] = \text{True}$.
- Si no existe una separación válida, la ecuación asegura que $OPT[n] = \text{False}$.

Parte 1: Si existe una separación válida

Hipótesis: Existe una separación válida, es decir, una secuencia de palabras $p_{i_1}, p_{i_2}, \ldots, p_{i_k} \in P$ tal que:

$$p_{i_1} + p_{i_2} + \cdots + p_{i_k} = s_1 s_2 \ldots s_n,$$

Tesis: Bajo esta hipótesis, la ecuación de recurrencia asegura que $OPT[n] = \text{True}$.

Consideremos cómo la ecuación evalúa las diferentes subcadenas consecutivas que pueden conformarse a partir de la cadena principal:

- **Caso base**: $OPT[0] = \text{True}$, lo cual es correcto, dado que vacío puede considerarse una separación válida sin palabras.
- **Evaluación de subcadenas**: Para cada posición i de 1 a n, la ecuación verifica si existe una palabra $p \in P$ con $|p| \leq i$ tal que:

  $$\text{cadena}[i - |p| : i] = p \quad \text{y} \quad OPT[i - |p|] = \text{True}.$$

  Si al menos una palabra de este estilo existe, $OPT[i] = \text{True}$; de lo contrario $OPT[i] = \text{False}$.

Sea $k_1, k_2, \ldots, k_t$ las posiciones donde terminan las palabras en la separación válida, con $k_t = n$, $k_0 = 0$, y $s_{k_{j-1}+1} \ldots s_{k_j} = p_{i_j}$ para $j = 1, 2, \ldots, t$.

- **Para** $i = k_1$: la subcadena $s_1 \ldots s_{k_1} = p_{i_1}$. Evaluamos $OPT[k_1]$:
  - Existe $p = p_{i_1} \in P$ tal que $|p_{i_1}| = k_1$.
  - $\text{cadena}[k_1 - |p_{i_1}| : k_1] = \text{cadena}[0 : k_1] = p_{i_1}$, que es verdadero.
  - $OPT[k_1 - |p_{i_1}|] = OPT[0] = \text{True}$.
  - Por lo tanto, $OPT[k_1] = \text{True}$.
- **Para** $i = k_2$: La subcadena $s_1 \ldots s_{k_2} = p_{i_1} + p_{i_2}$. Evaluamos $OPT[k_2]$:
  - Existe $p = p_{i_2} \in P$ tal que $|p_{i_2}| = k_2 - k_1$.
  - $\text{cadena}[k_2 - |p_{i_2}| : k_2] = \text{cadena}[k_1 : k_2] = p_{i_2}$.
  - $OPT[k_2 - |p_{i_2}|] = OPT[k_1] = \text{True}$ (por el paso anterior).
  - Por lo tanto, $OPT[k_2] = \text{True}$.
- **Generalización**: Para $i = k_j$ (con $j = 1, 2, \ldots, t$):
  - la subcadena $s_1 \ldots s_{k_j} = p_{i_1} + \cdots + p_{i_j}$.
  - Existe $p = p_{i_j} \in P$ tal que $|p_{i_j}| = k_j - k_{j-1}$.
  - $\text{cadena}[k_j - |p_{i_j}| : k_j] = \text{cadena}[k_{j-1} : k_j] = p_{i_j}$.
  - $OPT[k_j - |p_{i_j}|] = OPT[k_{j-1}] = \text{True}$ (por los pasos anteriores).
  - Por lo tanto, $OPT[k_j] = \text{True}$.
- **Final**: Para $i = k_t = n$, $OPT[n] = \text{True}$, ya que $s_1 \ldots s_n = p_{i_1} + \cdots + p_{i_t}$, y la ecuación verifica que p_{i_t} coincide con $\text{cadena}[n - |p_{i_t}| : n]$ y $OPT[n - |p_{i_t}|] = OPT[k_{t-1}] = \text{True}$.

La ecuación evalúa todas las palabras $p \in P$ con $|p| \leq i$, sin omitir ninguna palabra que forme parte de la separacion válida. Por lo tanto, si existe una separacion, $OPT[n] = \text{True}$.

$\therefore$ La ecuación de recurrencia asegura que $OPT[n] = \text{True}$ cuando existe una separación válida. Queda demostrado por metodo directo que determina de forma correcta cuando la cadena es un mensaje válido.

Parte 2: No existe al menos una separación válida

Hipótesis: No existe una separacion válida, es decir, no es posible separar $S = s_1 s_2 \dots s_n$ en una secuencia de palabras $p_{i_1}, p_{i_2}, \dots, p_{i_k} \in P$ tal que $p_{i_1} + p_{i_2} + \cdots + p_{i_k} = S$.

Tesis: Bajo esta hipótesis, la ecuación de recurrencia asegura que $OPT[n] = \text{False}$.

Analicemos como la ecuación evalúa la cadena:

- **Caso base**: $OPT[0] = \text{True}$, como se definió.
- **Evaluación de subcadenas**: Para cada $i = 1, 2, \dots, n$, $OPT[i] = \text{True}$ solo si existe una palabra $p \in P$ con $|p| \leq i$ tal que:

$$\text{cadena}[i - |p| : i] = p \quad \text{y} \quad OPT[i - |p|] = \text{True}.$$

  Si no existe tal palabra, $OPT[i] = \text{False}$.

Supongamos que $OPT[n] = \text{True}$. Esto implica que existe una palabra $p \in P$ tal que:

- $\text{cadena}[n - |p| : n] = p$.
- $OPT[n - |p|] = \text{True}$.

Si $OPT[n - |p|] = \text{True}$, entonces existe otra palabra $p' \in P$ tal que:

- $\text{cadena}[(n - |p|) - |p'| : n - |p|] = p'$.
- $OPT[(n - |p|) - |p'|] = \text{True}$.

Si continuamos con este proceso llegamos a la posición $i_0 = 0$, donde $OPT[0] = \text{True}$. Esto quiere decir que existe una secuencia de palabras $p_1, p_2, \dots, p_k \in P$ tales que:

$$p_1 + p_2 + \cdots + p_k = s_1 s_2 \dots s_n.$$

Pero esto contradice la hipotesis de que no existe una separación válida. Por lo tanto, $OPT[n] = \text{False}$.

Punto de fallo: La ecuación solo asigna $OPT[i] = \text{True}$ si encuentra una palabra p que coincida con $\text{cadena}[i - |p| : i]$ y la subcadena $\text{cadena}[1 : i - |p|]$ es separable. Como no existe una separacíon valida para S, no se puede concretar esta condición para $i = n$.

Ejemplo:

- Diccionario: {"aguante", "lanus"}.
- Cadena: "aguanterlanus".
- $OPT[7] = \text{True}$, dado que $\text{cadena}[1 : 7] =$ "aguante" pertenece al diccionario de palabras conocidas y $OPT[0] = \text{True}$.
- $OPT[8] = \text{False}$, porque $\text{cadena}[8 : 8] =$ "r" no coincide con ninguna palabra del diccionario.
- Para $i = 13$, no hay p tal que $\text{cadena}[13 - |p| : 13] = p$ y $OPT[13 - |p|] = \text{True}$. Entonces, $OPT[13] = \text{False}$.

$\therefore$ Si no existe al menos una separación válida, la ecuación asegura que $OPT[n] = \text{False}$. Queda demostrado por método directo que determina de forma correcta cuando la cadena no es un mensaje válido.

3. Algoritmo

3.1. Codigo

```python
INDEX_BOOL = 0
INDEX_PALABRA = 1

def encontrar_soplon_dinamico(palabras, cadena_desencriptada):
    n = len(cadena_desencriptada)
    mem = [(False, "")] * (n + 1)
    mem[0] = (True, "")

    for i in range(1, n + 1):
        for p in palabras:
            j = len(p)

            if (cadena_desencriptada[i - j:i] == p) and mem[i - j][INDEX_BOOL]:
                mem[i] = (True, p)
                break

    return reconstruir_solucion(mem, n)

def reconstruir_solucion(mem, n):

    if not mem[n][INDEX_BOOL]:
        return "No es un mensaje"

    solucion = []

    while n > 0:

        if mem[n][INDEX_PALABRA]:
            solucion.append(mem[n][INDEX_PALABRA])
            n -= len(mem[n][INDEX_PALABRA])

        else:
            return "No es un mensaje"

    solucion.reverse()
    return " ".join(solucion)
```

3.2. Complejidad: Construccion del vector de memorizacion

El algoritmo de construccion del vector de memorizacion para cada posición i de la cadena desencriptada, itera todas las palabras del diccionario.

Para cada palabra p del diccionario se fija si esta palabra coincide exactamente con los ultimos j caracteres que terminan en i de la cadena desencriptada donde $j = len(p)$ y si es posible construir un mensaje valido hasta la posicion $i - j$.

Por lo tanto, la función `encontrar_soplon_dinamico` tiene una complejidad de $\mathcal{O}(n \cdot p)$, donde n es el largo de la cadena desencriptada y p es la cantidad de palabras en el diccionario.

Adicionalmente, mencionamos que en un principio consideramos que la funcion tenia una complejidad de $\mathcal{O}(n \cdot p \cdot L)$, donde L es el largo maximo de las palabras.

¿Por que incluimos L en la complejidad? Debido a que en cada iteracioón interna donde vemos si la palabra p coincide exactamente con los ultimos j caracteres que terminan en i de la cadena desencriptada estamos realizando un slicing de longitud j y comparando strings de longitud j. Esta operacion cuesta $O(j)$ y teniendo en cuenta que las palabras pueden tener hasta un largo maximo L entonces esta operacion cuesta hasta $O(L)$.

Sin embargo, como nos encontramos trabajando con palabras en español y particularmente el diccionario de palabras que nos provee la catedra tiene palabras con longitud maxima 26 todas las comparaciones se hacen sobre strings cuya longitud es menor o igual a 26. Teniendo en cuenta esto y que la cantidad de palabras puede ser enorme y el largo de la cadena desencriptada también, asumimos que la longitud de las palabras es despreciable en el analisis de complejidad ya que no hay palabras de longitud arbitraria (tienen una longitud maxima fija) que puedan hacer que el tiempo de comparacion crezca.

3.3. Complejidad: Reconstruccion de la solucion

El algoritmo de reconstruccion en cada iteracion resta a n el largo de una palabra. En el peor caso, tendriamos palabras de longitud 1 con lo cual se ejecutarian n iteraciones.

Además, utiliza las funciones `reverse` y `join` que tienen una complejidad lineal en la cantidad de caracteres.

Por lo tanto, la función `reconstruir_solucion` tiene una complejidad de $\mathcal{O}(n)$ donde n es el largo del vector **mem**.

4. Análisis de la variabilidad de valores

¿Afecta a los tiempos del algoritmo planteado la variabilidad de los valores (mensajes, palabras del idioma, largos de las palabras, etc.)?.

4.1. Largos de las palabras

Como hemos mencionado anteriormente en el analisis de la complejidad de nuestro algoritmo de PD, asumimos que la longitud de las palabras es despreciable ya que no hay palabras de longitud arbitraria (tienen una longitud maxima fija) que puedan hacer que el tiempo de comparacion crezca.

Sin embargo, si tuviesemos palabras de longitud variable y potencialmente grandes (sin conocer su longitud maxima) entonces si afectaria a los tiempos del algoritmo planteado.

Esto por lo mismo que comentamos en el analisis de la complejidad y es debido a que en cada iteracion interna donde vemos si la palabra p coincide exactamente con los ultimos j caracteres que terminan en i de la cadena desencriptada estamos realizando un slicing de longitud j y comparando strings de longitud j. Esta operacion cuesta $O(j)$ y teniendo en cuenta que las palabras pueden tener hasta un largo maximo L entonces esta operacion cuesta hasta $O(L)$.

En este supuesto escenario la complejidad resultaria ser $\mathcal{O}(n \cdot p \cdot L)$, donde n es el largo de la cadena desencriptada, p es la cantidad de palabras en el diccionario y L es el largo maximo de las palabras.

4.2. Idioma

El idioma de las palabras puede influir en los tiempos del algoritmo ya que dependiendo del idioma podemos tener mas o menos palabras.

No solo esto, sino que tambien el idioma afecta el largo de las palabras y podriamos tener un idioma con palabras muy largas en promedio y otro idioma con palabras mucho mas cortas.

Otro caso a mencionar, es que para este trabajo consideramos encontrar la primera palabra posible. Si tendríamos que haber buscado específicamente una frase o tendríamos que haber considerado varias frases o palabras posibles a la hora de analizar la cadena, la complejidad sería mayor y el problema sería más difícil. En el español no sucede mucho pero dependiendo del idioma, esto se hace aún más complicado, como por ejemplo en el alemán.

5. Ejemplos de ejecucion

5.1. Generador

Para realizar ejemplos de ejecucion y realizar las mediciones de tiempo, generamos datos para nuestro algoritmo de Programacion Dinamica. Para ello utilizamos la siguiente funcion encargada de generar datos aleatorios y que además permite configurar el tamaño de los datos y la validez de los mismos.

```
URL_PALABRAS = "https://raw.githubusercontent.com/JorgeDuenasLerin/diccionario-espanol-txt/refs/heads/master/0_palabras_todas.txt"

def generar_entrada_aleatoria(cantidad_palabras, cantidad_palabras_desencriptadas, largo_maximo_palabras, valido):
    palabras = requests.get(URL_PALABRAS).text.splitlines()

    palabras_validas = [p.lower() for p in palabras if p.isalpha() and len(p) >= 3 and len(p) <= largo_maximo_palabras]

    palabras_elegidas = random.sample(palabras_validas, cantidad_palabras)

    if valido:
        palabras_aleatorias = random.choices(palabras_elegidas, k = cantidad_palabras_desencriptadas)
        cadena_desencriptada_elegida = ''.join(palabras_aleatorias)

    else:
        palabras_aleatorias = random.choices(palabras_elegidas, k = cantidad_palabras_desencriptadas - 1)
        cadena_desencriptada_elegida_parcial = ''.join(palabras_aleatorias)

        ruido = ''.join(random.choices(string.ascii_lowercase, k=random.randint(1, 10)))
        indice_ruido = random.randint(0, len(cadena_desencriptada_elegida_parcial))

        cadena_desencriptada_elegida = (
            cadena_desencriptada_elegida_parcial[:indice_ruido] +
            ruido +
            cadena_desencriptada_elegida_parcial[indice_ruido:]
        )

    return palabras_elegidas, cadena_desencriptada_elegida
```

¿Como se generan los datos?. Analicemos cada una de las lineas de codigo.

1. Descargamos el contenido del archivo de palabras desde la URL proporcionada por la catedra y dividimos el texto en una lista, separando por cada linea (una palabra por linea).
2. Creamos una nueva lista con palabras filtradas que solo contengan letras, tenga al menos 3 caracteres y no excedan el largo maximo elegido.
3. Seleccionamos una cantidad de palabras al azar sin repetir de la lista de palabras filtradas.
4. Si se eligio generar datos validos, entonces selecionamos la cantidad de palabras elegida y las unimos en una sola cadena, sin espacio entre las palabras.
5. Si se eligio generar datos de tal forma que la cadena no sea un mensaje, se genera una cadena invalida con ruido agregado. El ruido se agrega en una posicion aleatoria de la cadena.

Finalmente, el generador devuelve una lista con las palabras generadas y la cadena desencriptada generada.

¿Como generar datos?. Ejecutar el archivo *generador.py* indicando como parametro la cantidad de palabras del diccionario, la cantidad de palabras de la cadena desencriptada, el largo maximo de las palabras y la validez del mensaje. Esto generara datos en la ruta *tests/gen*.

5.2. Verificador

Para poder validar nuestros datos generados y de esta forma corroborar lo encontrado, utilizamos el siguiente verificador. Mencionamos que nuestro verificador recibe una lista de palabras, una cadena desencriptada y posible mensaje.

```
def verificador(palabras, cadena_desencriptada, mensaje):

    for palabra in mensaje.split():
        if palabra not in palabras:
            return False

    mensaje_concatenado = "".join(mensaje.split())
    return cadena_desencriptada == mensaje_concatenado
```

¿Que condiciones deben cumplirse para que el posible mensaje sea valido?.

1. Todas las palabras que conforman el posible mensaje se deben encontrar en el diccionario de palabras.
2. La concatenacion del posible mensaje debe ser exactamente igual a la cadena desencriptada.

5.3. Generar entrada y corroborar su resultado

Una forma de realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado es:

1. **G**enerar entrada utilizando el generador.
2. **E**jecutar el algoritmo de PD con los archivos generados en el paso anterior y utilizar el flag $-v$ al momento de ejecutarlo para utilizar el verificador.

6. Mediciones

Se realizaron mediciones de tiempo con el objetivo de verificar la complejidad teórica propuesta. Para ello, se generaron gráficos y se utilizó la técnica de cuadrados mínimos, siguiendo la explicación detallada proporcionada por la cátedra.

Para el grafico del error, se utiliza el error cuadratico total (SSE) el cual se calcula como

$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

donde y_i es el valor real, $\hat{y}_i$ es el valor predicho y n la cantidad de puntos.

Mencionamos tambien que, tal y como se indica en la consigna, tomamos más de una medición de la misma muestra y nos quedamos con el promedio para reducir el ruido en la medición. Esto utilizando el archivo util.py que nos proporciona la catedra y colocando la constante `RUNS_PER_SIZE` con valor 20.

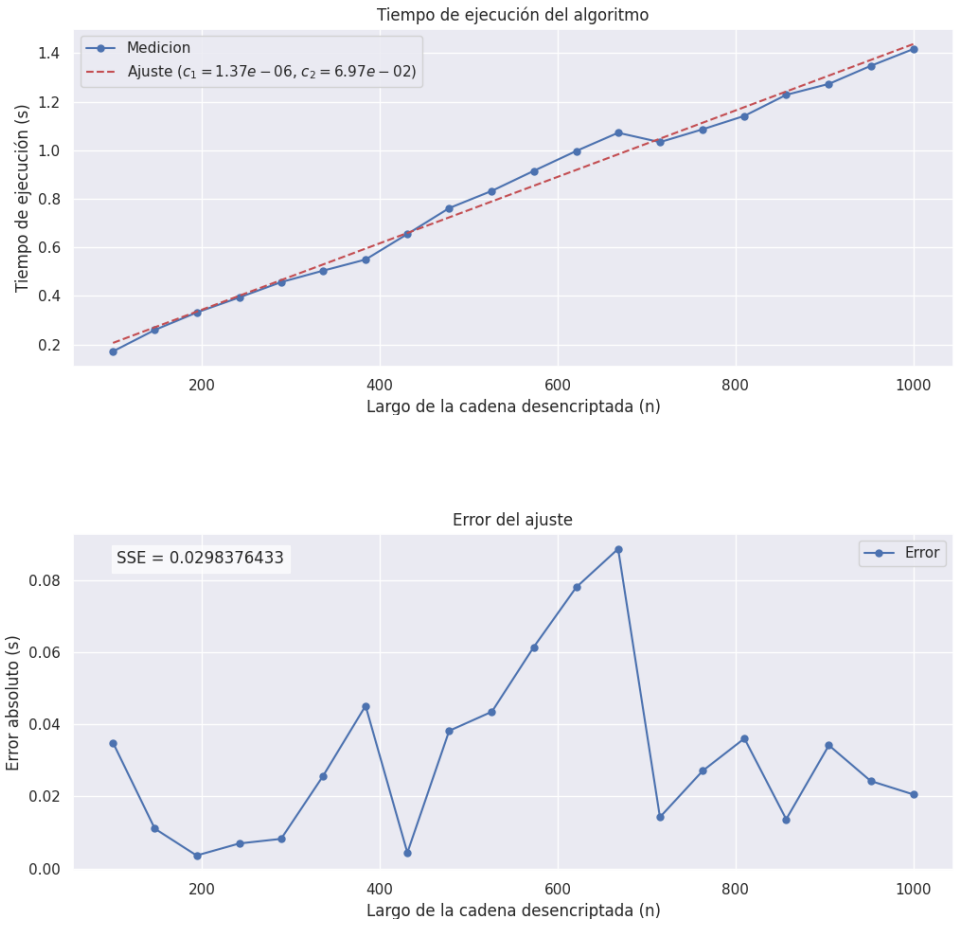
Por ultimo, y teniendo en cuenta que nuestra complejidad teorica propuesta es $\mathcal{O}(n \cdot p)$, si bien para el analisis de cuadrados minimos utilizamos la funcion $f(n, p) = c_1 \cdot n \cdot p + c_2$ para realizar los graficos sin complicaciones hemos fijado una de las dos variables de entrada. Dicho esto, a continuacion podremos ver graficos donde:

1. Hemos variado la cantidad de palabras en la cadena desencriptada (es decir, su largo) y fijado la cantidad de palabras del diccionario.
2. Hemos fijado la cantidad de palabras en la cadena desencriptada (es decir, su largo) y variado la cantidad de palabras del diccionario.

6.1. Variando el largo de la cadena desencriptada

6.1.1. Medicion 1

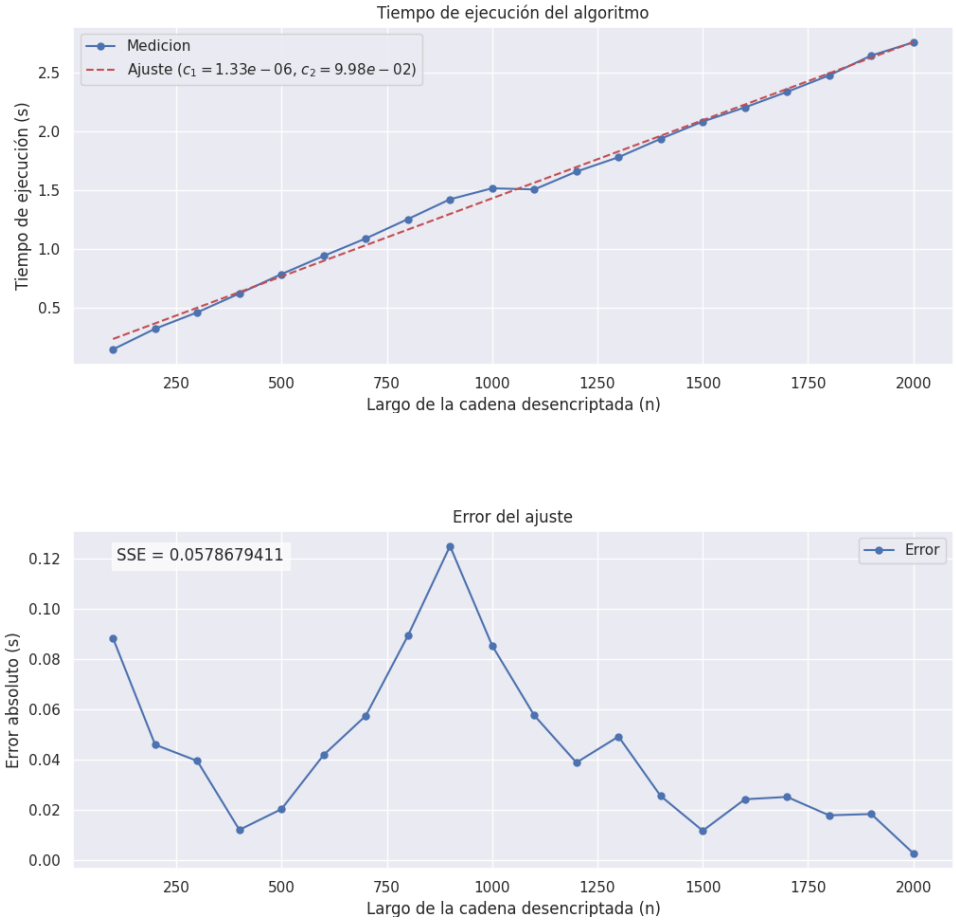
Consideremos el siguiente grafico generado considerando un largo de hasta 1000 palabras en la cadena desencriptada:



El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.030 s^2.

6.1.2. Medicion 2

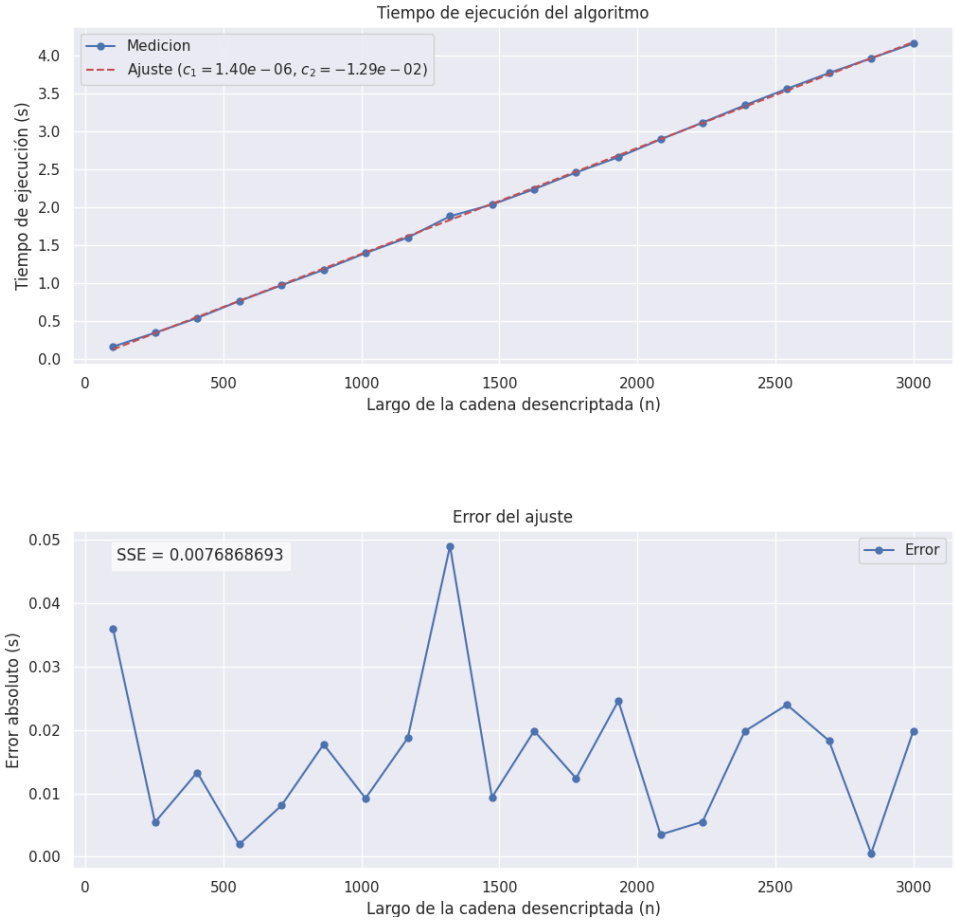
Consideremos el siguiente grafico generado considerando un largo de hasta 2000 palabras en la cadena desencriptada:



El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.058 s^2.

6.1.3. Medicion 3

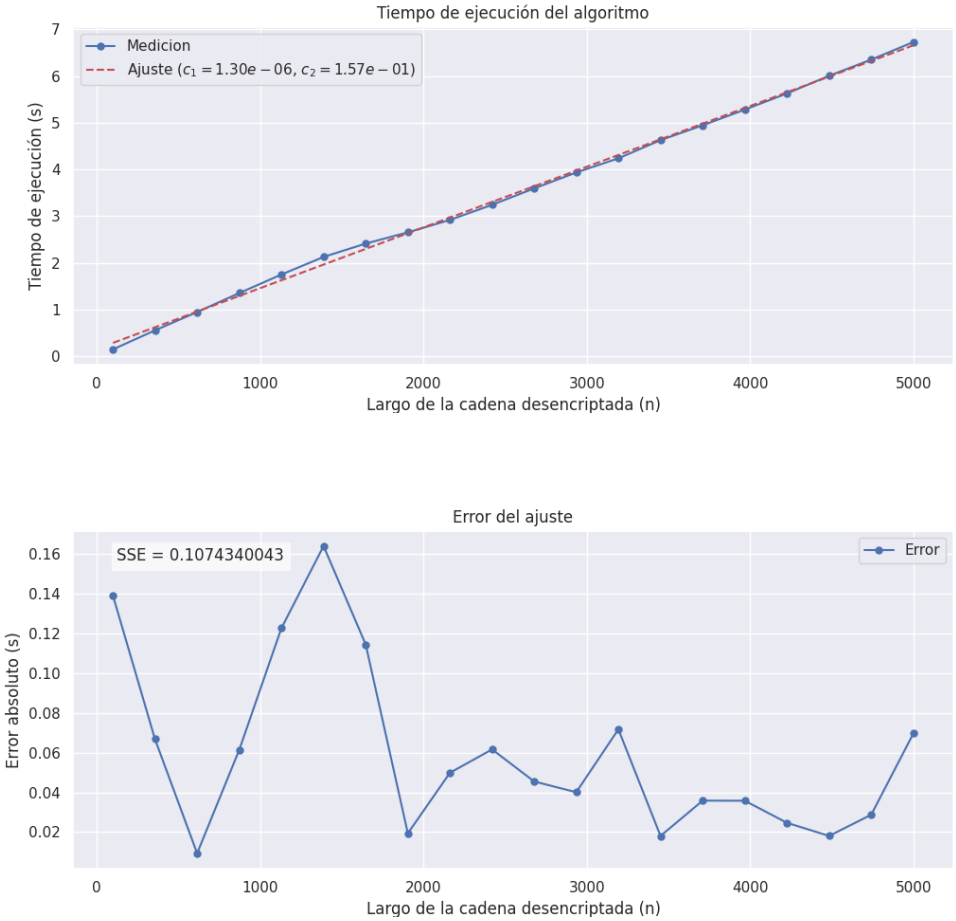
Consideremos el siguiente grafico generado considerando un largo de hasta 3000 palabras en la cadena desencriptada:



El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.0077 s^2.

6.1.4. Medicion 4

Consideremos el siguiente grafico generado considerando un largo de hasta 3000 palabras en la cadena desencriptada:

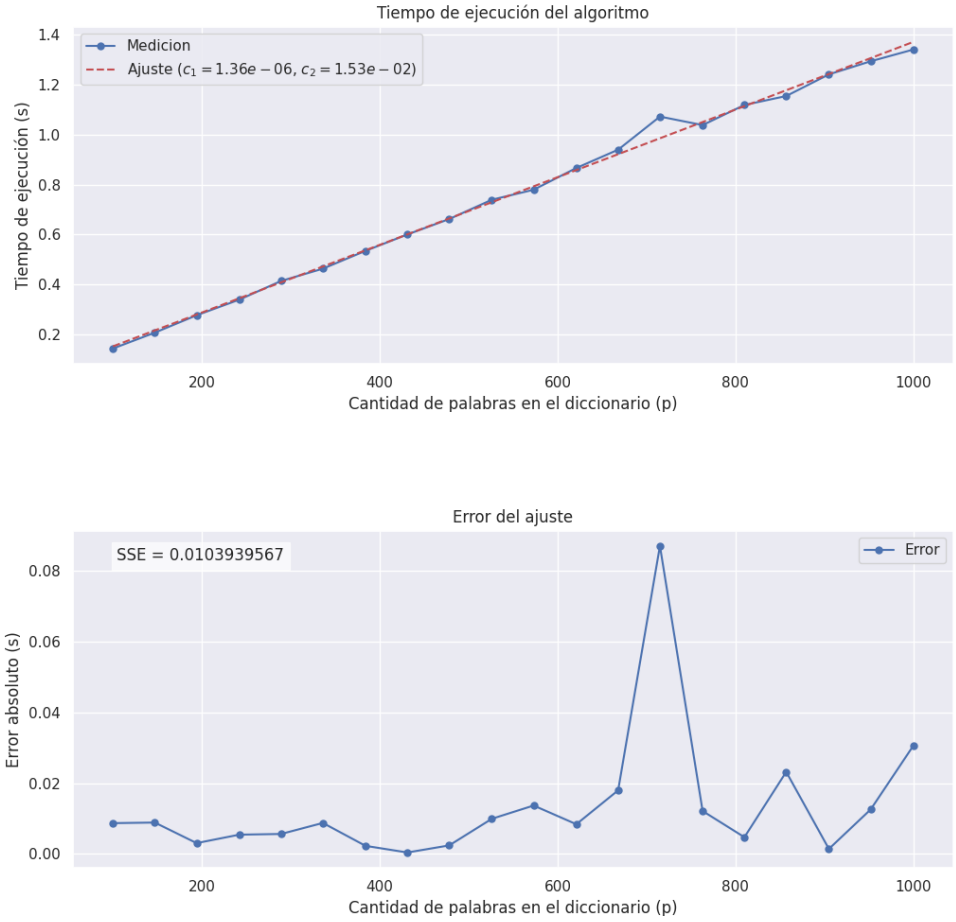


El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.11 s^2.

6.2. Variando la cantidad de palabras en el diccionario

6.2.1. Medicion 1

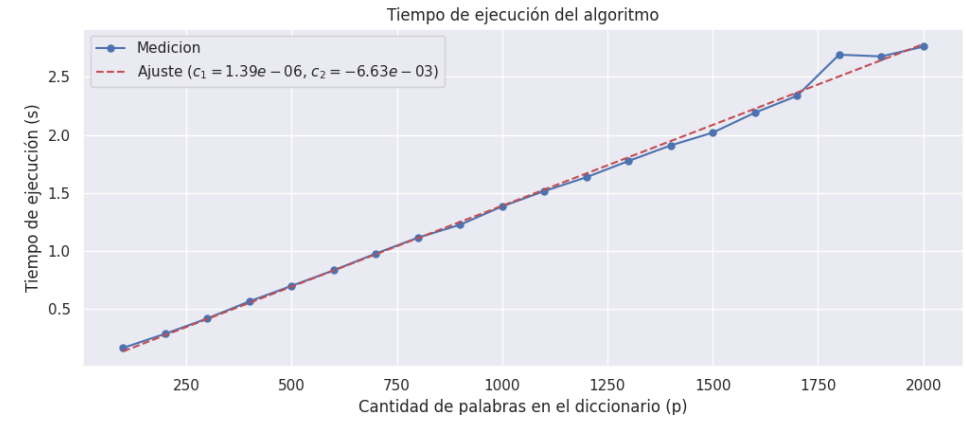
Consideremos el siguiente grafico generado considerando hasta 1000 palabras en el diccionario:

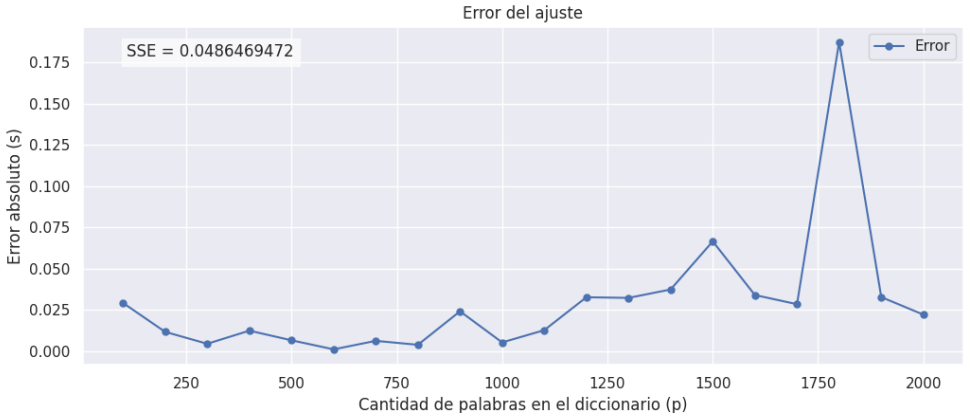


El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.010 s^2.

6.2.2. Medicion 2

Consideremos el siguiente grafico generado considerando hasta 2000 palabras en el diccionario:

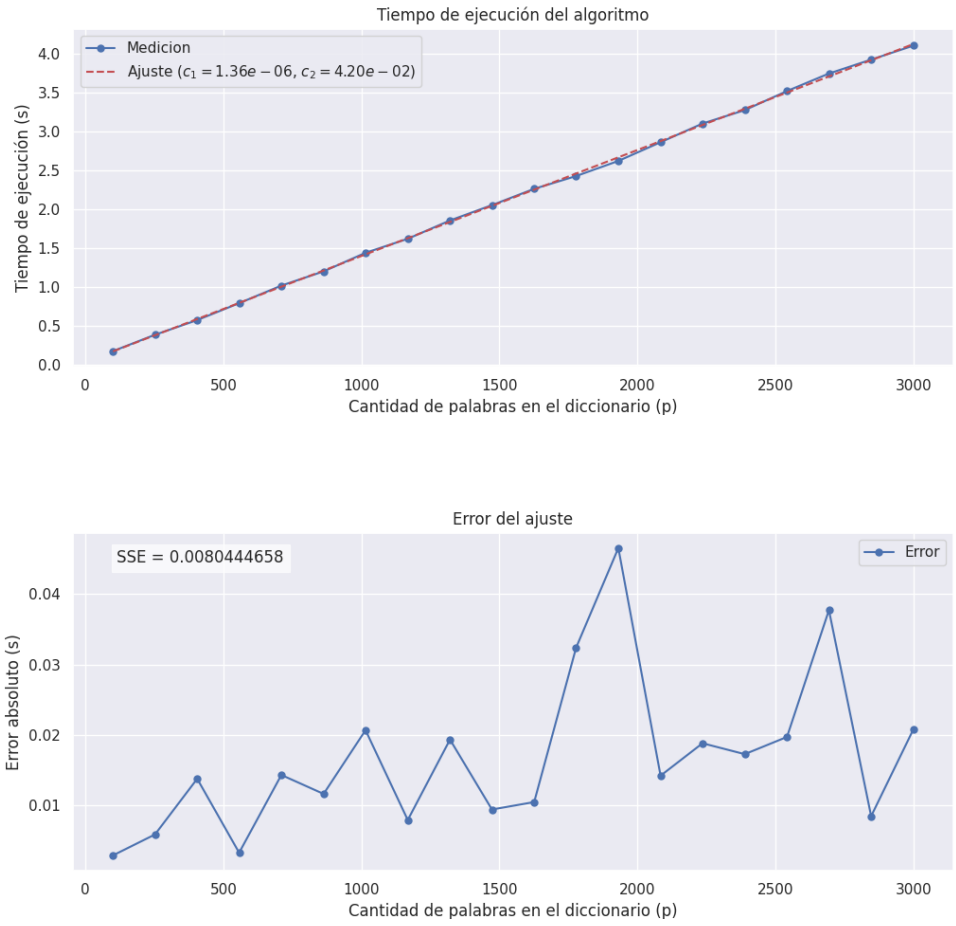




El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.49 s^2.

6.2.3. Medicion 3

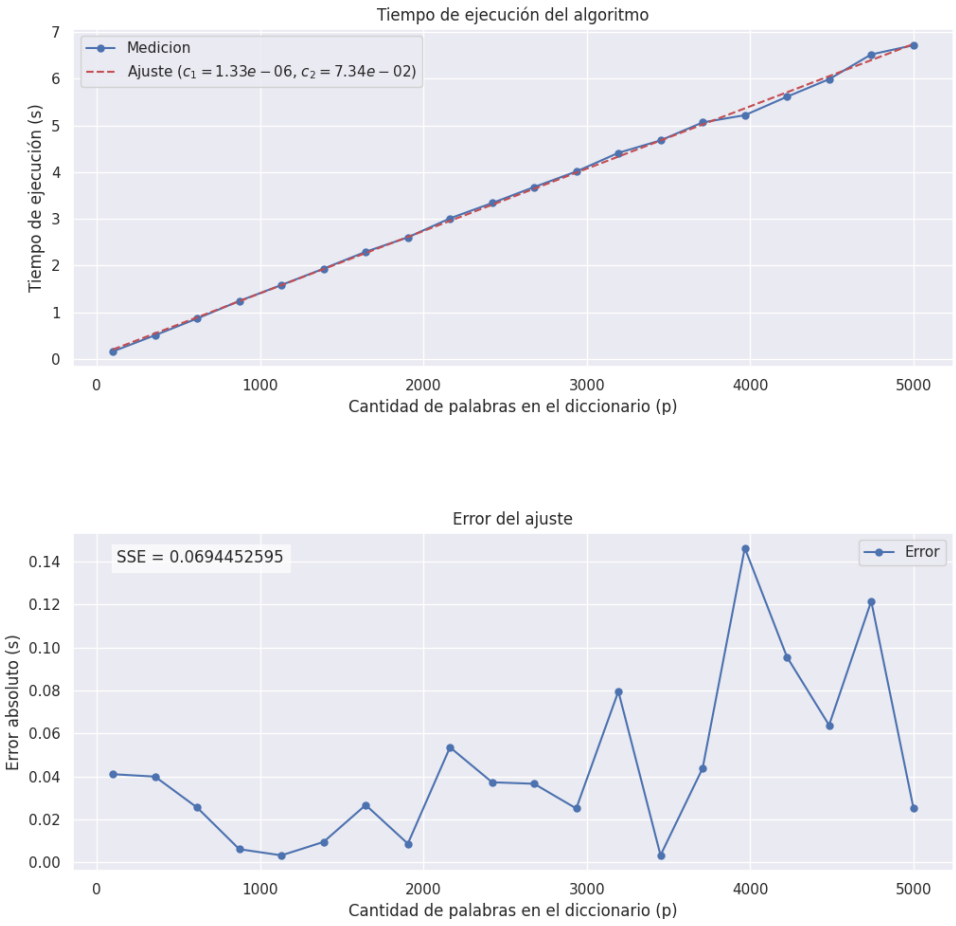
Consideremos el siguiente grafico generado considerando hasta 3000 palabras en el diccionario:



El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.0080 s^2.

6.2.4. Medicion 3

Consideremos el siguiente grafico generado considerando hasta 3000 palabras en el diccionario:



El error cuadratico total con las mediciones indicadas anteriormente para nuestra aproximacion es 0.070 s^2.

7. Conclusiones

En las mediciones realizadas se puede apreciar claramente que los tiempos de ejecucion del algoritmo dependen de la cantidad de palabras del diccionario de palabras y tambien del largo de la cadena desencriptada ya que al variar una y dejar fija la otra pudimos ver como hay una dependencia. Esto confirma que la complejidad del algoritmo es efectivamente O(n $\cdot p$) como se habia teoricamente propuesto ya que los tiempos de ejecucion dependen de ambas variables.