.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Que parezca programación dinámica

5 de Mayo de 2025

Franco Guardia
109374

Mateo Requejo
110109

Santino
Regidor
112147

1. Introducción

Luego de haber ayudado al Gringo y a Amarilla Pérez a encontrar a *la rata* que les estaba robando dinero hemos sido ascendidos. El problema ahora es que Amarilla detectó que hay un soplón en la organización que parece que se contacta con alguien de la policía utilizando mensajes encriptados. Esta encriptación está hecha con todas las palabras juntas y sin espacios, lo cual complica la desencriptación y validar que el mensaje tenga sentido

Nos interesa saber la información que se está filtrando, el área de desencriptación nos dará posibles mensajes desencriptados y nuestra tarea es validar si son posibles mensajes o no. No es nuestra labor validar si el mensaje tiene sentido o no ya que de eso se encargará otro área

Se nos pide diseñar un algoritmo de programación dinámica que indique si la cadena desencriptada es un posible mensaje o no, teniendo un listado de palabras que conforman el "vocabularioçon el cual hay que validar cada una de las palabras para ver si la cadena se puede separar en estas o no. Y finalmente si esto es posible devolver el mensaje que estaba contenido en la cadena

2. Análisis del problema

En este trabajo realizaremos el análisis teórico y empírico de un algoritmo de programación dinámica que indique si podemos desencriptar una cadena encriptada en un posible mensaje o no, y en caso de que sí podamos, devolver dicho mensaje.

En la situación planteada recibiremos un listado de palabras que conformarán nuestro vocabulario y esas palabras son las que utilizaremos para verificar si la cadena recibida es un posible mensaje o no. Es decir, las palabras que podamos extraer de la cadena son las que estarán en ese listado y no otras. Si no fuimos capaces de formar una palabra con todas las letras que estaban en la cadena entonces concluiremos que no es un posible mensaje, pero si en cambio para todas las letras pudimos extraer una palabra del listado entonces concluiremos que sí es un mensaje válido. Esto lo sabremos luego de haber analizado toda la cadena

La dificultad del problema radica en tener en cuenta las diversas posibilidades que nos aparezcan en ese posible mensaje. Debemos tener el recaudo de que si optamos por desencriptar una parte de la cadena de determinada forma, que el resto de la cadena no nos quede sin solución cuando en realidad sí había.

2.1. Solución propuesta

El algoritmo de programación dinámica propuesto se basa en ir almacenando en una estructura valores booleanos que indiquen si el texto hasta determinado índice es un mensaje o no lo es.

Para ir almancenando los valores, recorreremos la cadena completa. Lo que hacemos en cada iteración es recorrer el texto desde nuestro índice actual hacia atrás. Si encontramos una palabra que pertenezca a nuestro diccionario, validamos que toda la cadena anterior al inicio de esa palabra sea efectivamente un mensaje válido, haciendo uso de nuestra estructura. Si lo es, podemos decir que toda la cadena (hasta el índice analizado) es un mensaje valido. Caso contrario, seguimos iterando hacia atrás para encontrar otras alternativas. Podríamos iterar hasta el final de la cadena, pero sería innecesario ya que conllevaría una complejidad extra. Podemos tener almacenado un valor que nos indique el largo de la palabra más larga de nuestro diccionario. Si iteramos nuestra cadena hacia atrás y nos excedemos de este valor, ya sabemos que no encontraremos una palabra posible en el diccionario.

Como primer paso del algoritmo de programación dinámica hay que plantear la ecuación de recurrencia, es decir aquella expresión que luego se aplicará iterativamente en la resolución del problema y donde queda explícito el cálculo del óptimo en el subproblema actual utilizando los resultados de los óptimos de los subproblemas más pequeños. Este algoritmo puede ser descripto mediante la siguiente ecuación de recurrencia:

$$OPT[j] = \bigvee_{i=max(0,j-L)}^{j} (OPT[i-1] \wedge texto[i-1:j] \in diccionario) \quad (1)$$

Siendo L el largo de la palabra más larga del diccionario.

Resumidamente, nuestro algoritmo busca para el índice actual hacia atrás si es posible encontrar una palabra del diccionario. Si lo consigue, verificamos que toda la cadena anterior a esa palabra sea efectivamente un mensaje válido.

Con los subproblemas identificados, lo que puede ocurrir es que para la instancia del problema que se esté analizando se pueda formar o no una palabra conocida. En caso de que sí se pueda entonces nos referimos a un óptimo calculado antes que debería ser una posición con un valor verdadero ya que eso expresa que hasta ese punto el problema ya fue resuelto y todas las letras que están hasta ese punto forman parte de una palabra conocida, y con ese valor calculamos el óptimo actual hasta la instancia actual del problema.

2.2. Demostración ecuación de recurrencia

Cuando decimos que una cadena es segmentable, estamos diciendo que se puede separar en palabras del diccionario para formar un mensaje. Sea L el largo máximo de palabra del diccionario. Sea *texto* una cadena de longitud n.

Inducción sobre $j \in \{0, \ldots, n\}$**:**

Paso base: $j = 0$.

La cadena vacía se define como opt(0) = True.

Hipótesis inductiva: $\forall k < j$, se cumple:

$$\text{opt}(k) = True \Leftrightarrow texto[0:k] \text{ es segmentable con palabras del diccionario.}$$

Paso inductivo: Queremos probar que:

$$\text{opt}(j) = True \Leftrightarrow \exists i \in [\text{máx}(0, j-L), j)\text{: opt}(i) = \text{True} \wedge texto[i-1:j] \in \text{diccionario.}$$

$\Rightarrow$ Si suponemos que existe ese i. Por Hipótesis Inductiva, $texto[0:i]$ es segmentable. Además, si $texto[i-1:j] \in$ diccionario, entonces $texto[0:j]$ también es segmentable. $\Rightarrow$ opt(j) = True.

Ahora supongamos que opt(j) = True. Entonces por definición, se encontró algún i tal que opt(i) = True y $texto[i-1:j] \in$ diccionario. Por Hipótesis Inductiva, $texto[0:i]$ es segmentable, entonces $texto[0:j]$ también es segmentable.

Conclusión: Por inducción, $\forall j \in \{0, \ldots, n\}, \text{opt}(j) = True \Leftrightarrow texto[0:j]$ es segmentable. Si la hipótesis planteada no logra cumplirse, podemos concluir que el $texto[0:j]$ no es segmentable y por lo tanto no es un mensaje válido.

3. Algoritmo

3.1. Desencriptación

A continuación, presentamos el código del algoritmo explicado.

```
1 def desencriptar(texto, diccionario, max_largo):
2     """
3     Desencripta el texto utilizando el diccionario proporcionado. Si el texto no
      puede ser desencriptado, devuelve "No es un mensaje". Ademas del texto y el
      diccionario, se pasa por parametro el largo de la palabra mas larga del
      diccionario para optimizar la busqueda.
4     """
```

```
    optimos = [False] * (len(texto)+1)

    optimos[0] = True
    optimos[1] = True if texto[0] in diccionario else False

    indices = [-1] * (len(texto) + 1) # Indica el indice donde empieza la palabra
    que termina en cada posicion j.
    for j in range(2,len(optimos)):
        for i in range(j, max(0, j - max_largo), -1):
            aux = texto[i-1:j]
            if aux in diccionario and optimos[i-1]:
                optimos[j] = True
                indices[j] = i-1
                break
    if not optimos[-1]:
        return "No es un mensaje"
    return reconstruccion(texto, optimos, indices)
```

3.2. Reconstrucción

```
def reconstruccion(texto, optimos, indices):
    """
    Devuelve el texto desencriptado con las palabras separadas mediante espacios.
    """
    palabras = []
    idx = len(texto)
    while idx > 0:
        i = indices[idx]
        palabras.append(texto[i:idx])
        idx = i
    palabras.reverse()
    return " ".join(palabras)
```

3.3. Complejidad

Para analizar la complejidad debemos entender los diferentes pasos del algoritmo. En primer lugar, iteramos sobre cada posición j de la cadena de largo n. Esto es $O(n)$. En cada iteración se itera hacia atrás, como máximo hasta largo que pasamos por parámetro, que es el largo de la cadena más larga del diccionario. La llamaremos L. Esto es $O(L)$. Y además, para cada iteración se extrae el substring texto[i-1:j] que tiene costo $O(L)$ en el peor caso. Entonces, la complejidad total de nuestro algoritmo es: $O(nL^2)$.

La complejidad de la reconstrucción podemos despreciarla ya que recorre hacia trás los indices guardados extrayendo cada palabra una vez. En total se procesan todas las posiciones del texto y la suma de todos los substrings da como resultado el string original. Por lo tanto, esta parte tiene complejidad lineal.

4. Mediciones

A modo de complemento del análisis de complejidad se realizaron mediciones de tiempo del algoritmo de desecriptar con sets de datos generados con el módulo random semi-aleatorios y el siguiente algoritmo.

```

PORCENTAJE_ERRONEOS = 0
# Este parametro indica la probabilidad de que se le agregue un error al texto que
    en consecuencia es el porcentaje de errores para el sample generado.

ES_UNA_PALABRA = False
# Este parametro indica si se quiere generar el texto con una unica palabra

```

```python
def generar_datos(diccionario :dict, size :int, max_largo) -> tuple[str, dict]:

    """
    Genera una lista de textos encriptados (sin espacios) usando palabras del
    diccionario.

    devuelve una tupla: (texto, diccionario)
    """
    lista_palabras = list(diccionario)
    lista_palabras.pop(0)
    if ES_UNA_PALABRA:
        seleccionadas = [np.random.choice(lista_palabras)] * size
    else:
        seleccionadas = np.random.choice(lista_palabras, size)
    texto = "".join(seleccionadas)


    if np.random.randint(0, 100) <= PORCENTAJE_ERRONEOS:
        texto = agregar_errores(texto, size)
    return texto, diccionario, max_largo

def agregar_errores(texto: str, n: int) -> str:
    """
    Agrega errores al texto encriptado y lo devuelve.
    """
    caracteres = "abcdefghijklmnopqrstuvwxyz"
    texto = list(texto)
    for _ in range(n):
        error = np.random.choice(["insertar", "borrar", "sustituir"])
        pos = np.random.randint(0, len(texto)-1)

        if error == "insertar":
            texto.insert(pos, np.random.choice(list(caracteres)))

        elif error == "borrar":
            if pos == 0:
                texto.pop(0)
            else:
                texto.pop(pos)
        elif error == "sustituir":
            texto[pos] = np.random.choice(list(caracteres))

    return "".join(texto)
```

Como se puede ver en el algoritmo, las cadenas siempre se generan con las palabras del diccionario aunque, se puede agregar un porcentaje de errores en la variable PORCENTAJE_ERRONEOS para testear como afecta la variabilidad de los datos a las mediciones. Los resultados de las mediciones de tiempo y el error de la aproximación por cuadrados mínimos a una curva de complejidad $O(n * L^2)$fueron representados gráficamente con el uso de la biblioteca matplotlib de python.

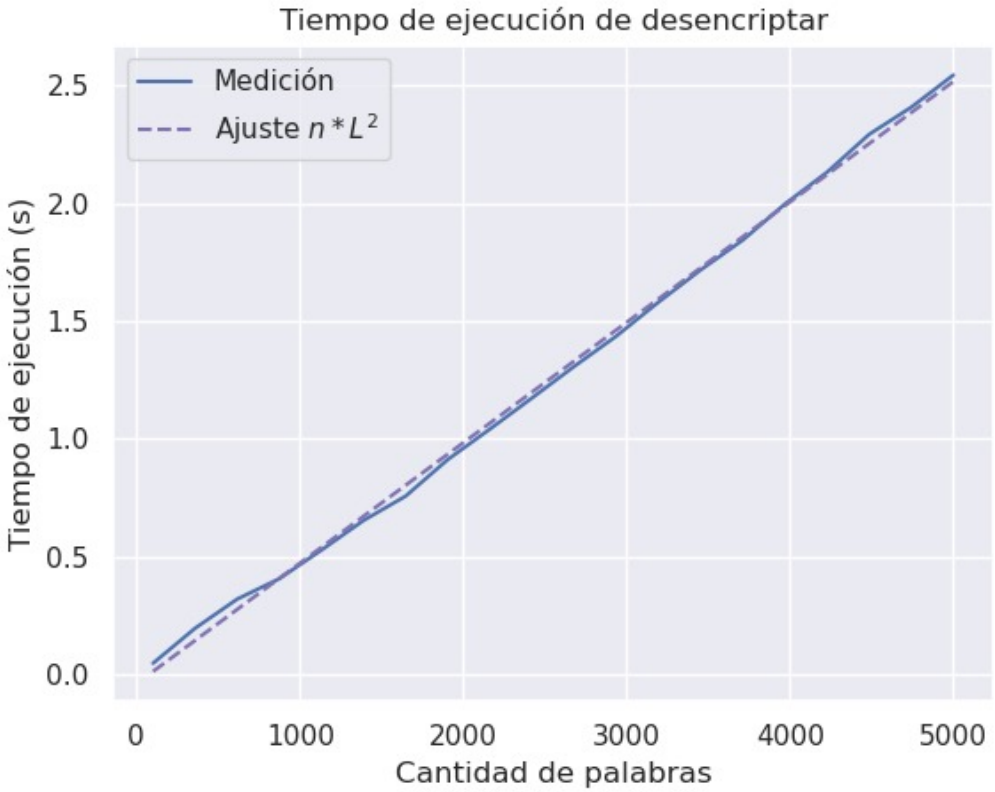

Figura 1: Tiempo de medición

Este gráfico sirve como respaldo empírico del análisis de complejidad que hicimos previamente ,ya que se puede observar como la medición es muy similar al ajuste $n * L^2$.

Figura 2: Error absoluto

El segundo gráfico nos muestra el error de la aproximación que hicimos respecto a los resultados obtenidos. Se puede observar que los valores obtenidos son bastante bajos lo que nos dice que el ajuste $n * L^2$ es bastante bueno respecto de la medición.

4.1. Variabilidad en los valores de entrada

Hay dos parámetros de entrada que afectan significativamente el tiempo de ejecución del algoritmo:

1. La diferencia entre la longitud máxima y mínima de las palabras del diccionario.
2. El porcentaje de mensajes que contienen errores.

En todos los gráficos que se presentan a continuación, se testearon 100 mensajes con longitudes crecientes desde 100 hasta 50.000 caracteres, y se promediaron los resultados sobre diez muestras. Para los dos primeros gráficos se utilizó un diccionario con 15 palabras. En cambio, para los dos últimos, el diccionario contenía 250 palabras.

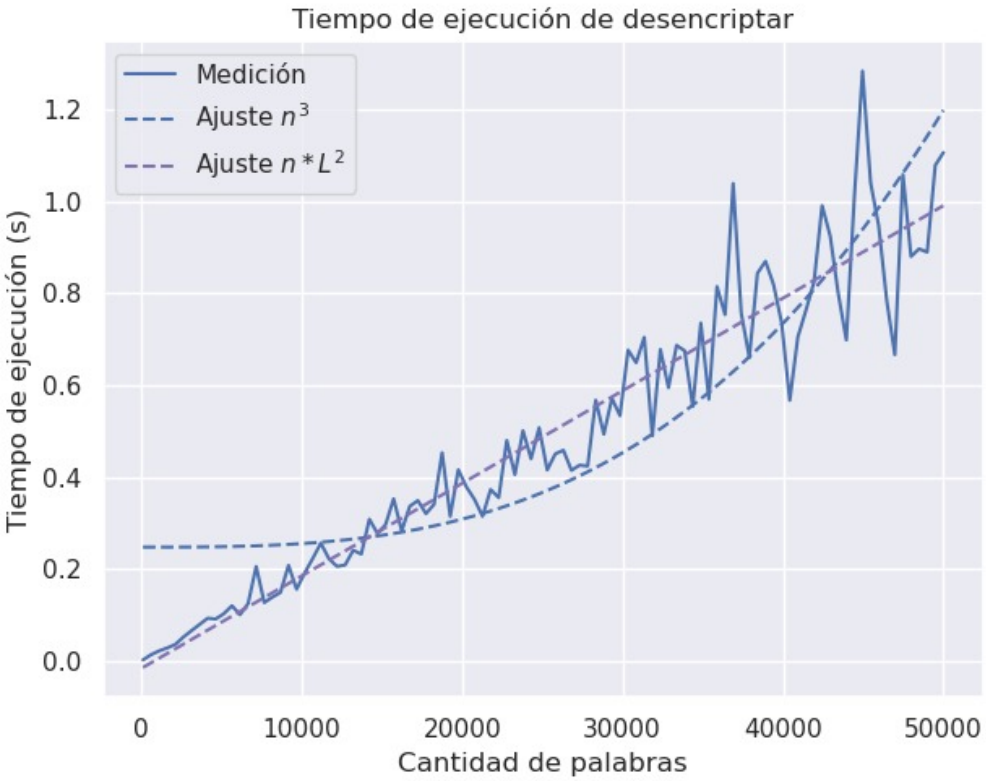

Figura 3: 30 % Errores

Como se puede observar, cuando un porcentaje significativo de los mensajes tiene errores, el tiempo de ejecución se aleja de la curva esperada de complejidad de $n * L^2$, especialmente a medida que aumenta el tamaño de los mensajes. Esto se debe a que los errores interrumpen las segmentaciones válidas, obligando al algoritmo a explorar los L subproblemas.

A diferencia del caso anterior, cuando no hay errores en los mensajes, el tiempo de ejecución se ajusta mucho mejor a la curva teórica de $n * L^2$. En este caso no necesita explorar todos los L subproblemas.

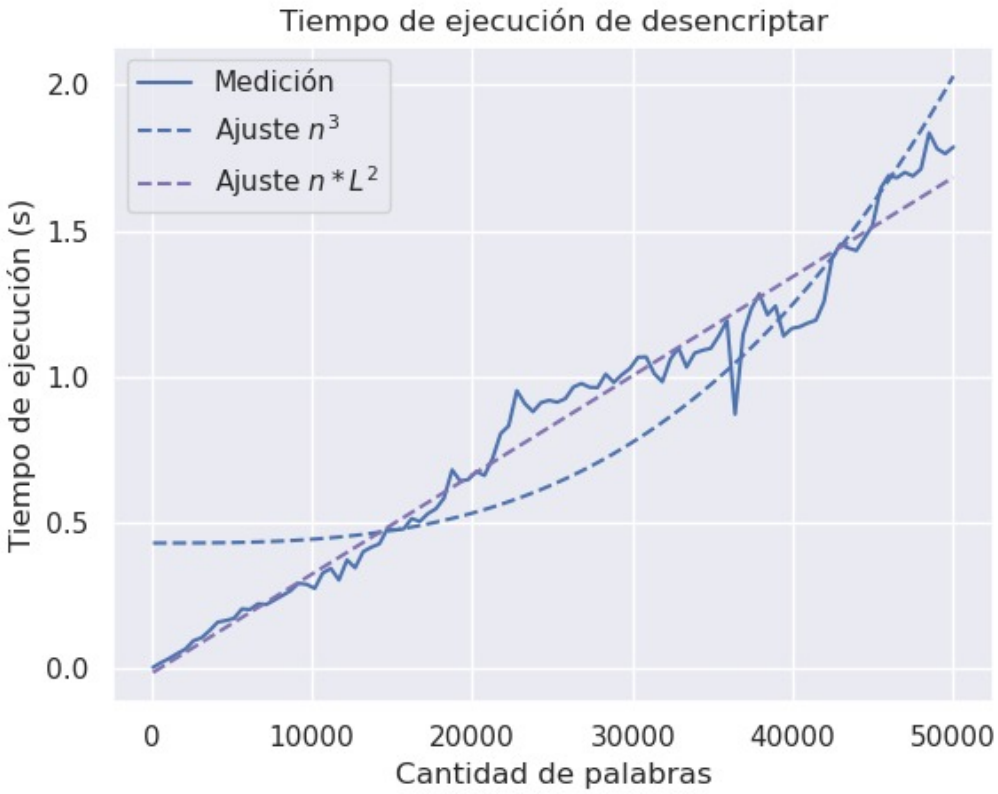

Figura 4: 0 % Errores

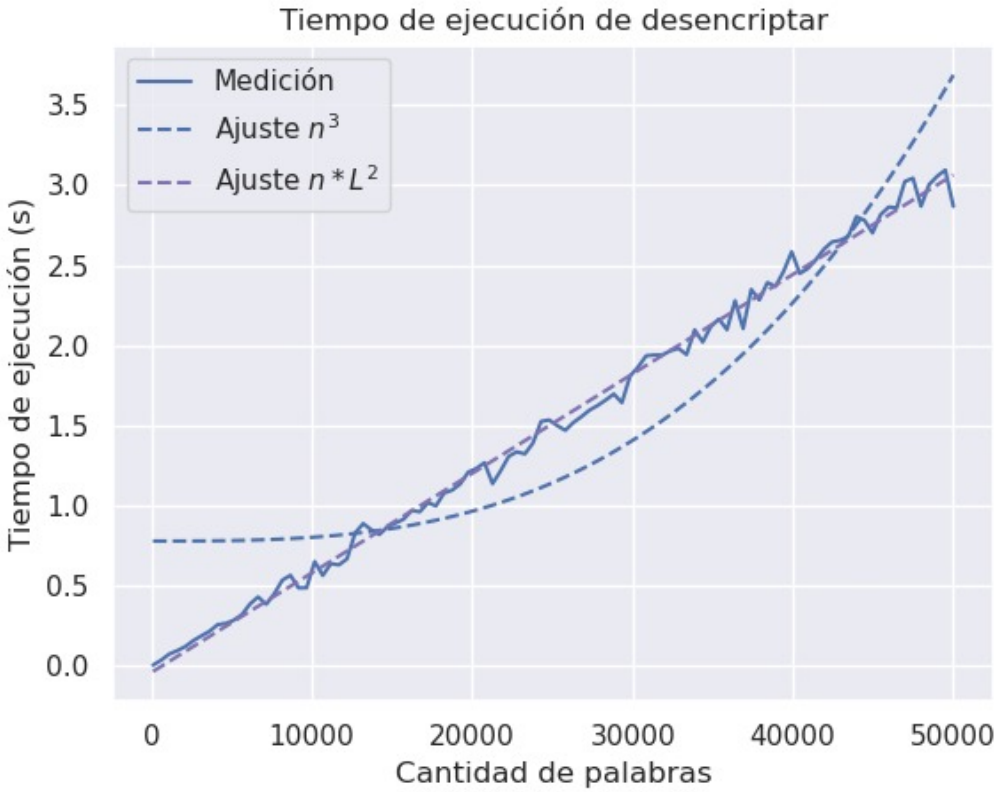

Figura 5: Diccionario más grande y 0 % Errores

Para esta prueba se utilizó un diccionario con 250 palabras y no se introdujeron errores. Al haber una mayor variedad de palabras, es menos probable que el algoritmo utilice palabras de longitud L, lo cual reduce el tiempo promedio de búsqueda. Como resultado, el tiempo de ejecución se estabiliza aún más alrededor de la complejidad teórica esperada.

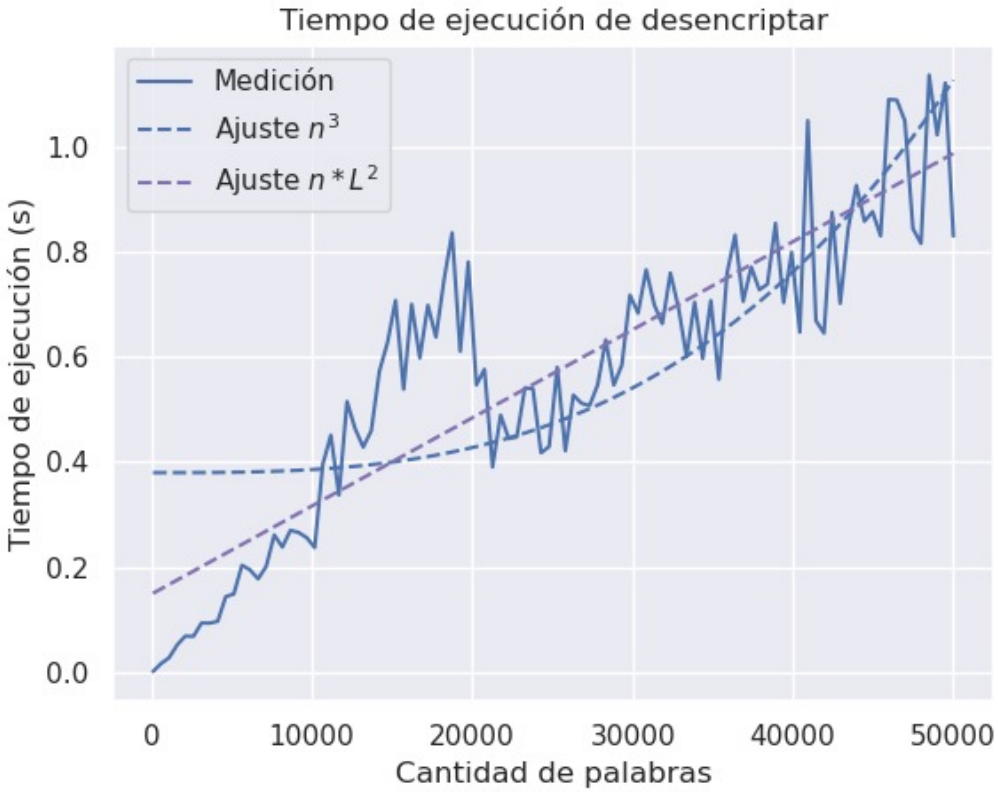

Figura 6: Todas las cadenas formadas por una palabra repetida

Por último, para mostrar un caso extremo, se generaron mensajes compuestos por una sola palabra repetida múltiples veces. Esa palabra fue elegida aleatoriamente del diccionario para cada mensaje. Esta configuración fuerza al algoritmo a realizar una búsqueda exacta de la longitud de la palabra elegida, lo que, en caso de ser muy cercana a L, aleja considerablemente la curva del comportamiento esperado.

5. Ejemplos

Para demostrar que el algoritmo elegido siempre obtiene la solución correcta realizamos algunas pruebas

Como primer prueba utilizamos los datos provistos por la cátedra, de los cuales ya sabíamos cual era el resultado esperado

Para los 14 archivos de prueba, la respuesta de si cada cadena recibida es un mensaje válido o no dio el resultado esperado:

- 10-in.txt: Resultado obtenido:
  es andar hola argentina compra andar plancha plancha reptiles es
  No es un mensaje
  No es un mensaje
  argentina reto compra plancha consistencia as reto argentina votar compra
  argentina argentina consistencia zanahoria argentina es compra hola consistencia argentina
  eso eso hola andar andar reptiles as plancha plancha andar
  No es un mensaje

argentina andar es hola as es votar plancha compra compra
plancha como como consistencia es votar zanahoria votar andar reto
votar argentina argentina andar votar votar como zanahoria hola eso
hola como andar reto votar compra zanahoria semana votar argentina
como semana eso zanahoria consistencia semana consistencia eso consistencia votar
No es un mensaje
andar es eso andar compra as consistencia consistencia andar votar
hola andar consistencia andar eso reptiles andar zanahoria eso reptiles
reto semana reto semana votar como andar votar hola reptiles
compra andar consistencia argentina eso reptiles reto plancha as reto
consistencia reto reto as consistencia semana eso semana hola eso
es hola hola reptiles plancha consistencia hola plancha votar como
compra zanahoria eso zanahoria consistencia reptiles votar compra hola hola

- 15-in.txt: Resultado obtenido: caro cera esmeralda esmeralda dromedario cerebro suelo turismo cera tu manchar revocar cera inundar tu tu doce revocar plural rubor dromedario invernadero tobogan eso doce manchar coro campesino guitarra adorno
No es un mensaje
No es un mensaje
No es un mensaje
No es un mensaje
tobogan soga guitarra rancho rubor arbustos adorno guitarra madurez nafta suelo mujer primero cerebro coro
coro cerebro monumento calendario caro calendario en pisar invernadero cicatriz monumento rubor doce trepar telescopio
pisar eso invierno violonchelo revocar pisar mosquito caro no guitarra mujer rubor telescopio coro invierno
calendario yo eso revocar yo peluquero pisar coro el sollosos copia algoritmo caro eso teclado plural inundar mujer monumento peluquero coro cicatriz viento nafta inundar campesino primero tobogan manchar en
mujer esmeralda guitarra coro momia mujer cicatriz rubor no plural mosquito peluquero mujer campesino adorno
trepar no guitarra madurez caro algoritmo pisar madurez sollosos telescopio telescopio tu dromedario manchar estudiar
campesino nafta rubor doce mafia mujer dromedario no violonchelo tobogan no turismo mosquito violonchelo violonchelo
No es un mensaje
...

En el resto de casos el resultado obtenido es igual al esperado, no se agrega por un tema de tamaño de la salida. Por otro lado también hicimos pruebas por nuestra cuenta con distintos sets de datos generados:

- 01-es.txt: Resultado obtenido:
elefante elefante elefante elefante elefante elefante elefante elefante elefante elefante elefante elefante elefante elefante
tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón tiburón
anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado anaranjado
magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta magenta
índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo índigo
mango mango

zorro zorro
león león león león león león león león león león león león león león león león león león león león león león león león león
cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo cocodrilo

- 02-no-es.txt: Resultado obtenido:
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  No es un mensaje
  ...

Con estos tests la idea era probar el funcionamiento del algoritmo en casos borde para asegurarnos que aunque haya varias asignaciones posibles para un caso concreto igualmente el algoritmo identificaba cual era la indicada de forma tal de identificar correctamente si la cadena recibida era válida o no. Para estos casos no teníamos un resultado esperado de antemano pero sabíamos que si insertábamos un caracter no válido en alguna cadena entonces esta se invalidaría, porque ya no sería posible extraer una palabra conocida con todas las letras que la conforman

6. Conclusiones

El desarrollo de este informe nos da demostrado el potencial de la programación dinámica a la hora de resolver problemas, ya que nos permitió resolver el problema planteado de forma correcta, eficiente y sobre todo con una complejidad relativamente baja, características que hubieran sido imposibles de obtener con la utilización de fuerza bruta.

A partir del análisis propuesto, se confirma que el algoritmo resuelve correctamente el problema de desencriptar cadenas encriptadas utilizando el diccionario proporcionado. El enfoque permite evaluar de manera eficiente la segmentación del texto en palabras válidas, memorizando los subresultados para hacer más óptimo el proceso.

Los resultados obtenidos con varios conjuntos de pruebas demostraron que el algoritmo identifica de manera precisa si una cadena puede segmentarse o no. Además, la demostración formal mediante inducción garantiza la optimalidad del algoritmo.