Memoria Práctica 3: Partición de palabras

Alonso del Rincón de la Villa y Alberto Lardiés Getán

18 de abril de 2023

Índice

1. Diseño

Para identificar el problema como un problema resoluble con un esquema de programación dinámica necesitamos identificar las decisiones. Hemos identificado el hecho de añadir o no una palabra encontrada en el diccionario a la partición. Hay tantas decisiones óptimas como palabras que forman parte de cada partición (el problema puede tener como solución varias particiones). Por el tema de optimalidad de Bellman las soluciones óptimas que constituyen las soluciones de los subroblemas de n (longitud de la cadena a dividir) menor forman parte de las decisiones de la solución del subproblema de tamaño n (el problema original) y hayando las soluciones para los subproblemas de n menor conseguimos la del problema original.

2. Pruebas

2.1. Complejidad de los algoritmos

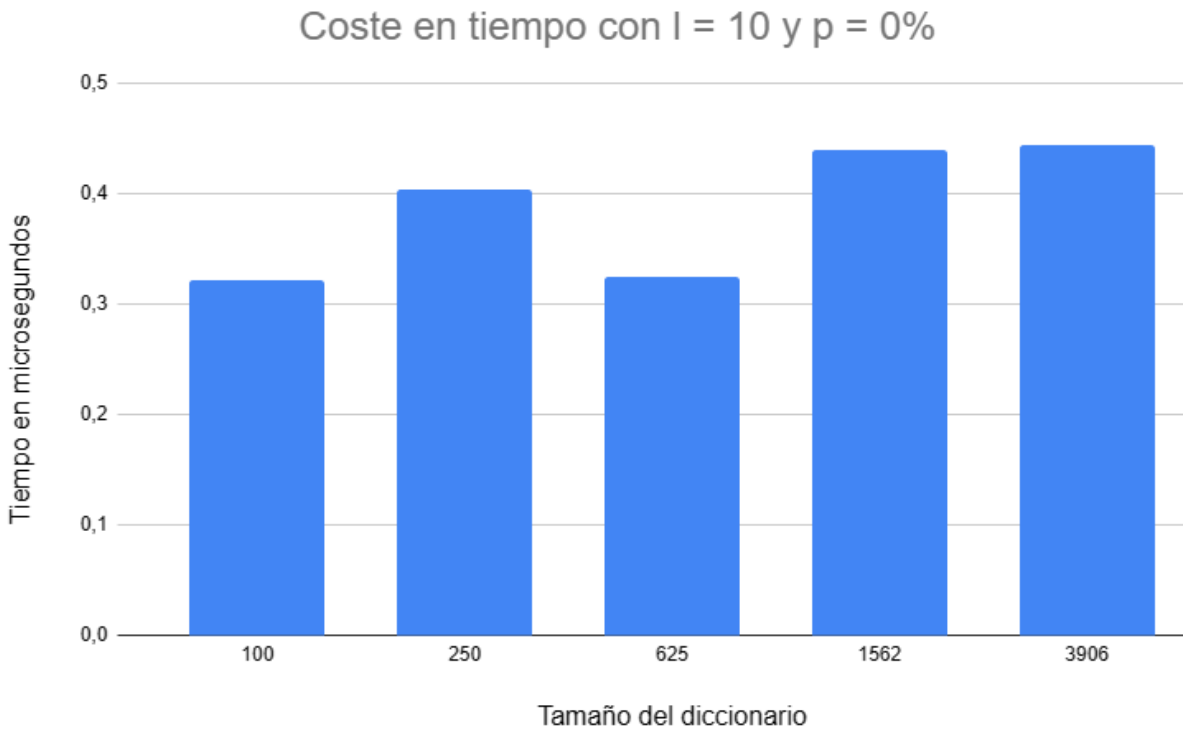
Para calcular la solución del problema original necesitamos rellenar la matriz de programación dinámica con las soluciones de los otros subproblemas. Como son n subproblemas en total necesitamos n iteraciones para el cálculo de todas. Cada subproblema necesita las soluciones de los subproblemas de tamaño menor para calcular su solución, al recorrerlas iteramos un total de 2n-1 veces. En el caso peor (todos los subproblemas de tamaño menor tienen alguna solución), para las 2n-1 iteraciones comprobaremos si el sufijo de la cadena se corresponde con alguna palabra del diccionario (O($log(D)$), siendo D el tamaño del diccionario). De estar siempre el sufijo (su separación de la cadena tiene coste O(n)) en el diccionario (caso peor) añadiremos una nueva solución al subproblema correspondiente por cada solución que tenga el subproblema que se está utilizando para hayar las soluciones del subproblema actual. Este último número depende de la naturaleza del diccionario y de la cadena a separar y no lo podemos estimar. Por lo tanto tenemos que el coste teórico temporal de nuestro algoritmo es O($(2n-1) * (log(D) + n + k)$), siendo k el coste temporal, que no podemos estimar, de iterar sobre las soluciones de un subproblema.

2.2. Mediciones y conclusiones

1. En este primer caso se ha fijado la longitud (l) de la frase a particionar en 10 palabras, incrementando linealmente el tamaño del diccionario de búsqueda (D). Se utiliza la frase formada por palabras aleatorias del diccionario.

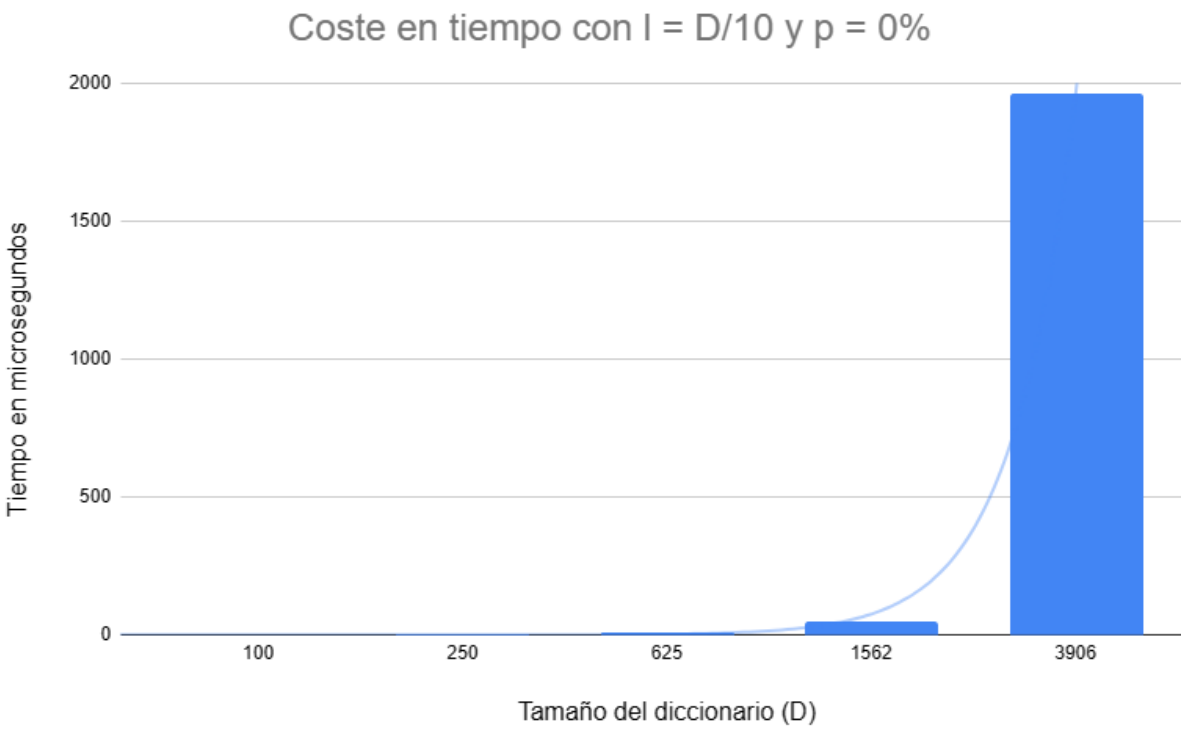
D	l	Prob	Tiempo de separación (ms)	Particiones
100	10	0	0,321818	1
250	10	0	0,404414	1
625	10	0	0,324234	1
1562	10	0	0,439781	1
3906	10	0	0,443855	1

   Esperabamos un incremento en el tiempo de separación, pero al no conocer en qué proporcion contribuye la búsqueda de palabras en el diccionario al tiempo de cada iteración puede ser posible que este resulte proporcionalmente bajo y que por ello no se aprecie.


Coste en tiempo con l = 10 y p = 0%
0,5
0,4
0,3
0,2
0,1
0,0
Tiempo en microsegundos
100
250
625
1562
3906
Tamaño del diccionario

2. Para el segundo caso se incrementa linealmente el tamaño del diccionario de búsqueda (D), calculando en cada una la longitud de la frase (l) a buscar como D/10. Al igual que en el caso anterior se utiliza la frase compuesta por concatenación de palabras del diccionario.

D	l	p	Tiempo de separación (ms)	Particiones
100	10	0	0,318193	1
250	25	0	0,879124	1
625	62	0	6,35003	1
1562	156	0	49,1608	1
3906	390	0	1963	1536

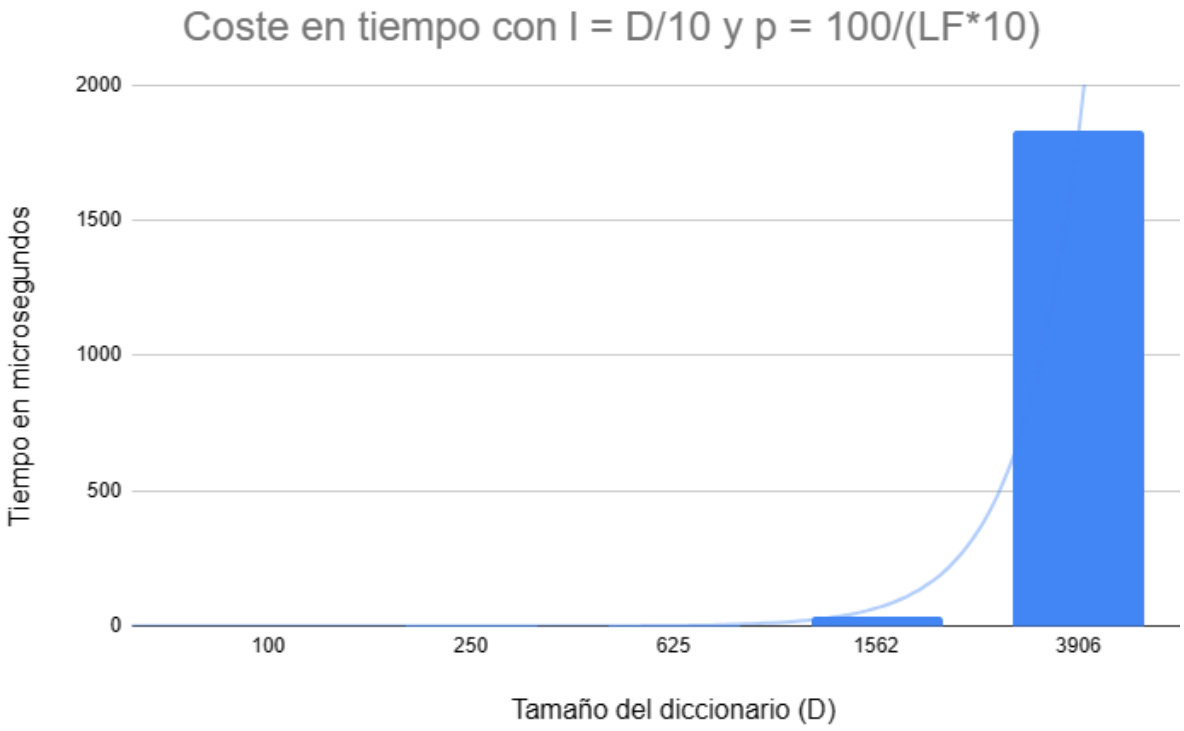


En este caso esperabamos un crecimiento lineal del tiempo pero el incremento es superior. Podría explicarse si el número de soluciones de cada subproblema (k) creciera con n tendríamos un coste asintótico polinómico o $n * log(n)$ que podría explicar el crecimiento.

Al súbito incremento del número de particiones no le encontramos explicación. Al no variar el número de caracteres distintos usados en las palabras ni la longitud de las palabras (nuestro generador de prueba las cre con longitudes entre 2 y 10) una cadena más larga compuesta de palabras de un diccionario más grande debería contener fruto de la concatenación más palabras que las otras cadenas más cortas compuestas de palabras de diccionarios más pequeños, pero no conseguimos comprender por que el cambio es tan súbito (de 1 a 1536).

3. Para el último caso se mantiene el incremento lineal del tamaño del diccionario (D) y de la longitud de la frase (l), pero a diferencia de los casos anteriores se incluye una probabilidad constante de modificar cada carácter por otro aleatorio. Llamamos LF a la longitud en caracteres de la frase y a partir de un valor $p = 10$ se calcula esta probabilidad como $1/(LF * p)$.

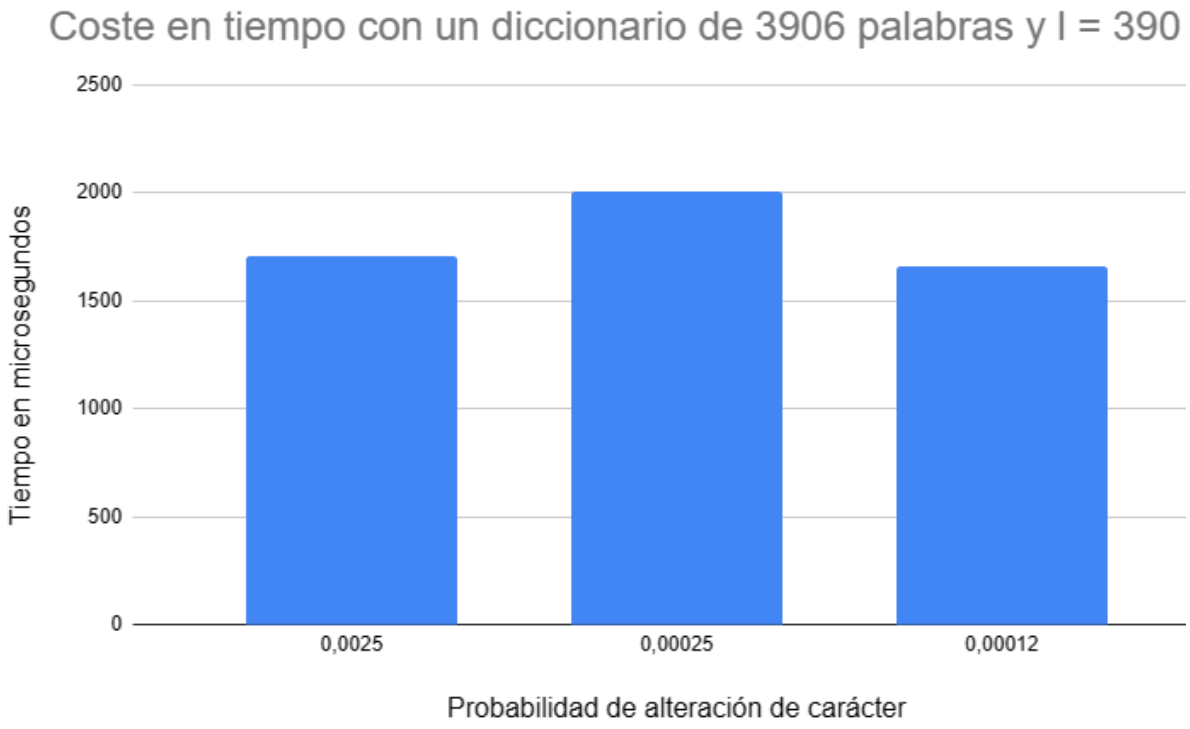
D	l	p	Tiempo de separación (ms)	Particiones
100	10	0,008	0,329946	1
250	25	0,0034	1,0274	1
625	62	0,0014	5,05573	1
1562	156	0,0006	37,4329	4
3906	390	0,0002	1829	512



No se aprecia un impacto en el tiempo de ejecución respecto al caso anterior. Entendemos que una ofuscación de la cadena producirá menos particiones, pues la cadena ya no está hecha de palabras en su totalidad y los datos parecen corroborarlo, pues se encuentran 3 veces menos particiones.

4. La probabilidad de sustituir un caracter de la cadena es muy baja. Si la utilizamos para calcular la probabilidad de que ningun caracter cambie vemos que en todo caso, da igual el tamaño del diccionario, es aproximadamente 0.9. Por lo tanto esperamos que reducir la probabilidad no repercuta pero que un aumento (con el parámetro p = 1 conseguimos que la probabilidad de que no cambie ninguno baje a 0.003) suponga una diferencia.

D	p	probabilidad	Tiempo de separación (ms)	Particiones
3906	1	0,0025	1706,23	192
3906	10	0,00025	2006,38	128
3906	20	0,00012	1656,4	96



Nuestra predicción se cumple. Entendemos que debido a las palabras de nuestro diccionario son concatenaciones de caracteres aleatorios la sustitución de caracteres en la cadena no tiene tanto impacto como si el diccionario fuera de palabras reales.