Memoria
Prácticas
Algoritmia Básica

Práctica 3

Autor:

Hugo Mateo Trejo, 816678

Paula Oliván Usieto, 771938

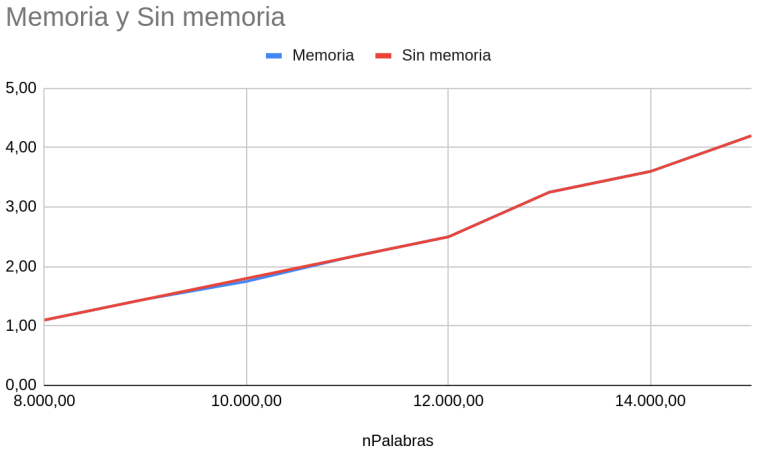
Análisis de los resultados:

Como se verá posteriormente en las pruebas, el algoritmo escala exponencialmente con el número de palabras en la entrada.
Además es altamente sensible a las palabras compuestas, es decir, palabras que contienen otras palabras en su interior (Ej: “migala”, “mi, gala”). Esto provoca que el algoritmo tenga que encontrar recursivamente todas las posibles combinaciones de frases.

El tamaño del diccionario es irrelevante, dado que para almacenarlo se utiliza un set hash cuya operación exists es O(1).
Los benchmarks se han hecho con frases completas existentes. Si la frase no existe, es equivalente a realizar el algoritmo con las palabras existentes iniciales y recorrer linealmente el resto de letras.

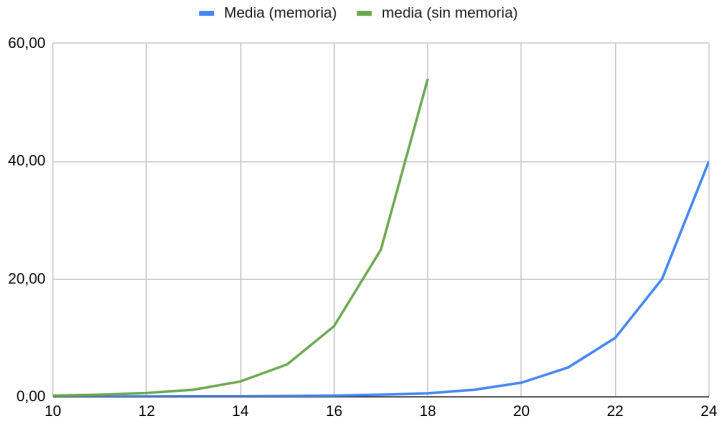
Benchmarks:

Tiempo de ejecución en segundos al aumentar el número de palabras no compuestas



Se puede apreciar que el algoritmo escala linealmente para las palabras no compuestas, dado que simplemente aumenta la longitud del camino único. Como la memoria sirve solo para caminos ya explorados, no hay diferencia entre ambas implementaciones.

Media de tiempo de ejecución, en segundos, al aumentar el número de palabras “migala” al principio de la frase:

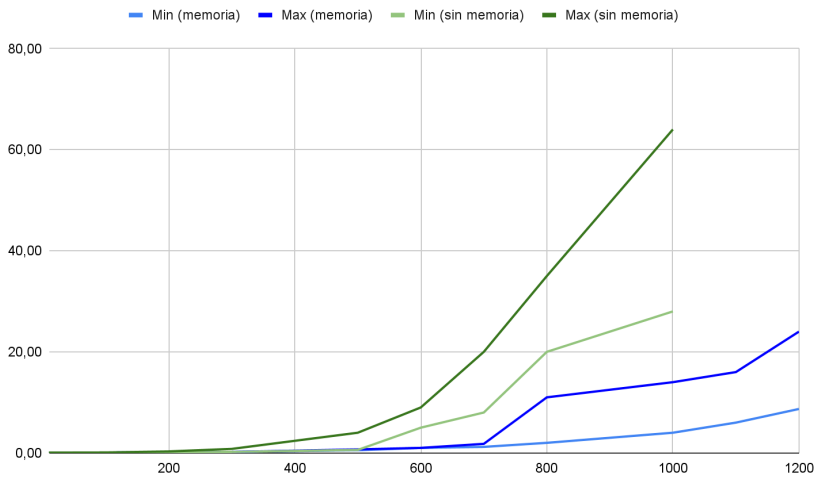


Cada “migala” provoca que el camino se subdivida en [“mi”, “gala” + resto] y en [“migala” + resto]. Al final de la frase, hay 10 palabras aleatorias del diccionario para añadir relleno (en la mayoría de casos, sin palabras compuestas)

Se puede apreciar claramente cómo ambos algoritmos escalan exponencialmente en las palabras compuestas. Al aumentar el número de “migalas” el número de caminos aumenta en 2^n.

El algoritmo con memoria evita volver a recorrer los caminos enteros cada vez que hay una bifurcación, reduciendo la penalización en las palabras compuestas y consiguiendo que la exponencial sea más suave.

Tiempo de ejecución en segundos al aumentar el número de palabras (elegidas aleatoriamente) en la entrada, máximo y mínimo observado:



Se puede observar una gran variabilidad entre el máximo y el mínimo. Esto se debe a que el benchmark genera combinaciones aleatorias de palabras del diccionario. Si la salida

contiene mayor cantidad de palabras compuestas, el algoritmo tarda mucho más en recorrer todas las combinaciones.

En cantidades pequeñas de palabras aparecen muy pocas palabras compuestas, por lo que al igual que en la primera gráfica ambas implementaciones dan los mismos resultados.

No se ve claramente ni la escala exponencial ni la lineal, dado que en realidad se da una combinación de ambas.
Los mínimos tienen poca cantidad de palabras compuestas y se acercan al crecimiento lineal, y los máximos tienen gran cantidad y se acercan al exponencial.

El algoritmo con memoria, al reducir la penalización en las palabras compuestas, tarda mucho más en acercarse al crecimiento exponencial. La mejora es suficiente como para ocultar las penalizaciones, y hacer que el coste se asemeje al lineal.

Pruebas:

Para comprobar la corrección del algoritmo se ejecuta 10 veces con 100 palabras elegidas aleatoriamente, con probabilidad de que alguna letra cambie según el enunciado.
Al final, si alguna letra había mutado se comprueba que la salida fuera No, y si no que fuera Si.
Se lanza mediante python3 test.py

La probabilidad es alta, concretamente 1/10. No depende de la longitud de la palabra, dado que una probabilidad de 1/(10*LF) repetida LF veces da un total de p=1/10, siempre.