Reporte Escrito
Segundo Proyecto

Víctor Barquero Miranda

I. Descripción del primer problema

El problema de la mochila es un problema que optimiza la combinatoria, en palabras un poco más simples, busca la mejor solución entre todas las posibles soluciones de un problema. El problema se modela de manera que buscar llenar una mochila que solo puede soportar un peso determinado, con todos los objetos o parte de ellos, cada uno de los objetos tiene asignado con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total buscando no exceder el peso máximo.

II. Descripción de la solución

En la primer parte de este proyecto busca implementar una solución en fuerza bruta y otra en programación dinámica para solucionar el problema de la mochila, por otra parte se debe realizar un análisis de estos dos para poder determinar el rendimiento en cada uno de ellos.

A. *Programación Dinámica:*

Para la solución del problema utilizando programación dinámica se utilizó el modelo bottom-up, se utilizó este modelo porque a diferencia del top-down, en el diseño bottom-up se parte del detalle y conforme se avanza se van enlazando para formar componentes más grandes, todo esto varias veces hasta que se logra llegar a una solución óptima. Las estrategias basadas en el flujo de información ”bottom-up” suelen ser más rápidas y no posee el problema del límite de recursividad, pero a la hora de entender el código puede ser un poco difícil.

Para esta implementación se utiliza una matriz inicializada en ceros ya que a la hora de recorrer la matriz debemos escoger el valor mayor en cada iteración por lo que al inicializar debe ser el número más bajo posible.

Luego se realiza una iteración por cada objeto en la lista de objetos, donde dentro de la misma iteración se realiza otra buscando todos y cada uno de los objetos que si caben dentro de la mochila, a partir de aquí se busca maximizar la cantidad de espacio utilizado en la mochila, por lo que se busca abarcar el mayor espacio por objeto dentro de la mochila, por lo mismo se utiliza la función max en cada iteración, decidiendo si lleva cierto objeto o no.

B. *Fuerza Bruta:*

La búsqueda por fuerza bruta, búsqueda combinatoria, búsqueda exhaustiva o simplemente fuerza bruta, es una técnica trivial pero a menudo usada, que consiste en buscar todas las posibles soluciones al problema, con el fin de verificar o encontrar la mejor solución.

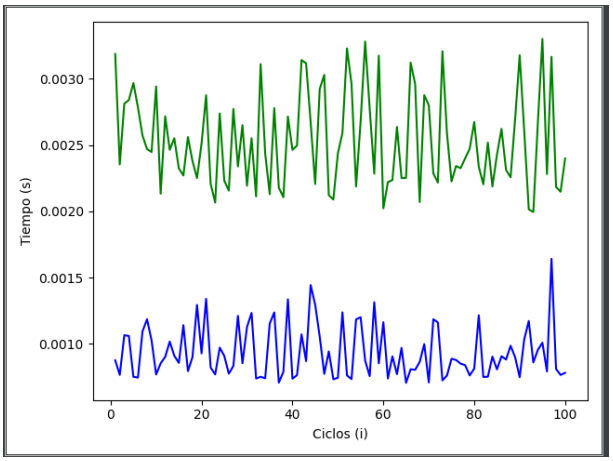
En esta solución simplemente se calculan todas las posibles combinaciones de objetos que caben dentro de la mochial y a partir de esto se escoge la que maximice la utilización de la mochila.

III. Experimento

El análisis del rendimiento de los diferentes algoritmos se ejecutó en el sistema operativo Ubuntu 20 y un Procesador Intel Core i7 de 3.2GHZ. Cada algoritmo se probó con un conjunto de datos aleatorios (pero utilizando los mismos datos de prueba en ambos algoritmos) varias veces y conforme se iba ejecutando se iban almacenando el tiempo o duración en cada ejecución.
Luego se procede a graficar la información obtenida utilizando la librería de Matplotlib de Python. El experimento se realizo con diferentes valores como por ejemplo tamaño de la mochila, cantidad de elementos, diferentes pesos, diferentes valores y cantidad máxima iteraciones por experimento.

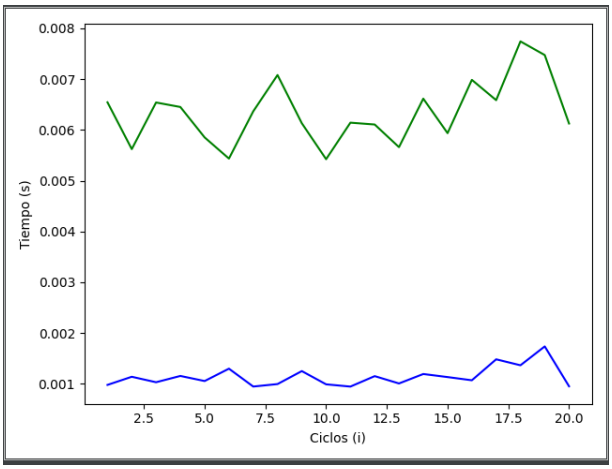
IV. Gráficos

A. Gráfico 1



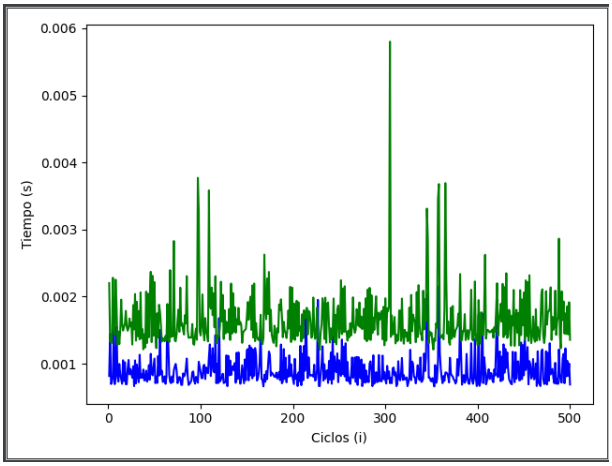
Este experimento se ejecutó con python3 generador.py -p 1 20 10 5 100 10 70 1 5, una mochila de tamaño 200, 10 objetos para escoger y 20 iteraciones.
En la linea azul está representado el algoritmo en Programación Dinámica y en verde el algoritmo en Fuerza Bruta, se puede notar la gran diferencia en tiempo entre un algoritmo y otro.

B. Gráfico 2



Este experimento se ejecutó con python3 contenedor.py -p 1 20 10 5 100 10 70 1 5, una mochila de tamaño 200, 10 objetos para escoger y 300 iteraciones, de igual forma se puede observar como el algoritmo de Programación Dinámica es mucho más eficiente que el de Fuerza Bruta.

C. Gráfico 3



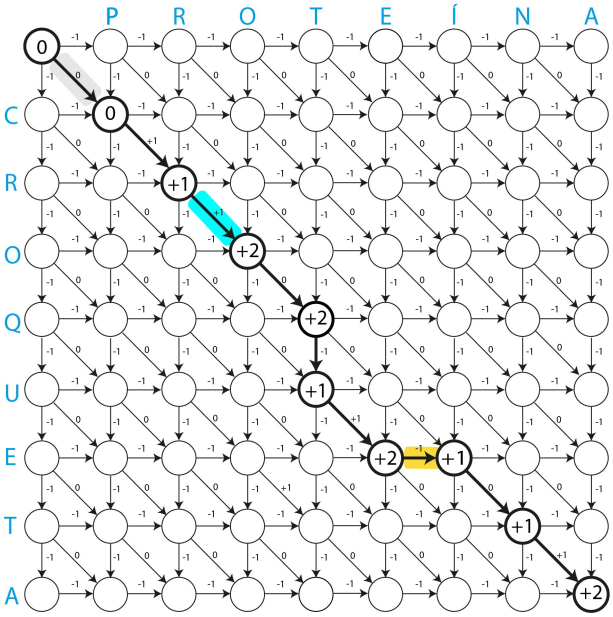
Este experimento se ejecutó con python3 contenedor.py -p 1 20 10 5 100 10 70 1 5, una mochila de tamaño 200, 10 objetos para escoger y 500 iteraciones, en este experimento se nota como hay varios picos de duración por parte del algoritmo de fuerza bruta donde se nota que en ciertos casos se comporta de una manera inestable, ya que el algoritmo de programación dinámica no tiene estos

grandes cambios en los mismos casos, incluso en el pico de duración más alto del algoritmo de fuerza bruta, el algoritmo de programación dinámica se mantuvo bajo.

V. Análisis y Conclusiones Problema 1

Realizando un análisis empírico de estos problemas se puede concluir que al implementar programación dinámica la eficiencia aumenta en grandes cantidades. También cabe destacar que la computadora solía pegarse un poco a la hora de correr el algoritmo de fuerza bruta debido a la alta demanda en el procesador, sin embargo utilizando programación dinámica esto no solía pasar gracias a que el procesador podía funcionar de una manera más libre sin tanto esfuerzo.

VI. Problema "Alineamiento de Secuencias"



VII. Descripción del segundo problema

Un alineamiento de secuencias en bio-informática es una forma de representar y comparar dos o más secuencias o cadenas de ADN, ARN, o estructuras primarias proteicas para resaltar sus zonas de similitud, que podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas consultados. Este problema lo que busca resolver es la forma en la que dos cadenas de secuencias estén alineadas entre sí de la mejor manera, las cadenas están conformadas por Adenina representada con una A, Guanina representada con una G, Timina con una T y Citosina con una C, donde en dos hileras comparadas entre sí puede haber un match (caso donde ambas letras son iguales en la misma posición) un missmatch (ambas letras son diferentes) o puede haber un GAP (caso donde hay una letra y un espacio en blanco en la otra cadena) la idea es buscar maximizar la cantidad de match en la cadena para que las mismas estén lo mas alineadas posible.

VIII. Descripción de la solución

En esta sección del proyecto se implementaron dos soluciones una en fuerza bruta y la otra utilizando programación dinámica para resolver el problema del alineamiento de secuencias.

A. *Programación Dinámica:*

Para la solución del problema utilizando programación dinámica se utilizó el modelo top-down, se utilizó este modelo porque en el problema anterior opté por Bottom Up y así implementar ambos modelos de programación dinámica, en el diseño bottom-up se parte del detalle y conforme se avanza se van enlazando para formar componentes más grandes, en este caso se hizo lo contrario se comenzó armando la matriz de manera que se calculaban todos los posibles caminos a tomar al armas la matriz, luego la secuencia se arma al realizar el traceback del recorrido óptimo tal como lo vimos en clase.

B. *Fuerza Bruta:*

Para la implementación en fuerza bruta se utilizó una función recursiva que abarca los tres posibles

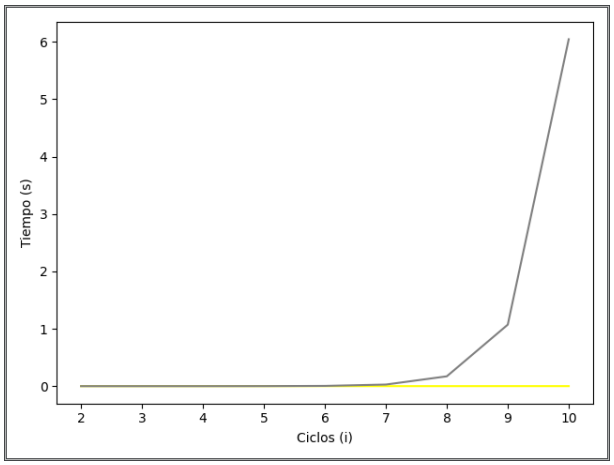
casos al comparar una posición dada en las hileras, al ser recursiva se abarcan todas las posibles combinaciones con las hileras resultantes, se le asigna un valor a cada hilera gracias una función externa donde se evalúa cada hilera resultante y al final se utiliza la función max de python para determinar cuál de todas las hileras tiene mejor valor.

IX. Experimento

Para el análisis se utilizó la misma máquina de los análisis pasados, en este caso se realizaron dos experimentos unos asociados con la cantidad de ciclos realizados y ver la diferencia en la media de tiempo entre ambos algoritmos y otro donde se busca ver a partir de que largo de hilera empieza a tener problemas cada uno de los algoritmos.

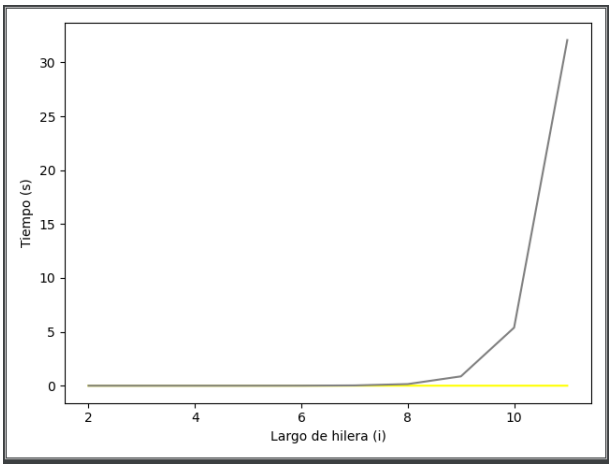
X. Gráficos

A. Gráfico 4



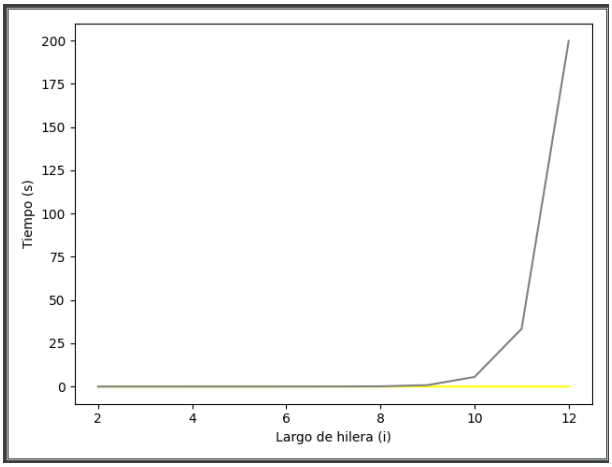
Este experimento se ejecutó con python3 generador.py -p 2 10 10, lo que significa que se realizaron dos hileras de tamaño 10. Lo más interesante de este gráfico es como se ve que a partir de hileras de tamaño 8 empieza a subir exponencialmente el tiempo que tarda el algoritmo de fuerza bruta mientras que el de programación dinámica se mantiene bastante estable en el tiempo sin importar el largo de la hilera.

B. Gráfico 5



Este experimento se ejecutó con python3 generador.py -p 2 11 11, lo que significa que se realizaron dos hileras de tamaño 11. Vemos como en el caso anterior el tiempo máximo que tardó el sistema fue de solamente 6 segundos, sin embargo aumentando el tamaño de la hilera en solamente uno, pasa de tardar 6s a 30s queriendo decir que se está quintuplicando el tiempo.

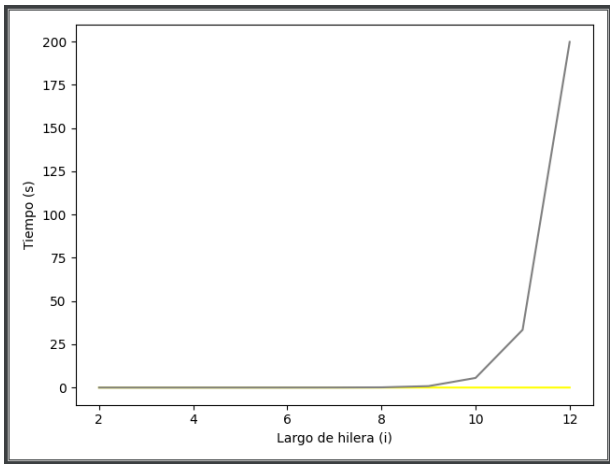
C. Gráfico 6



Este experimento se ejecutó con python3 generador.py -p 2 12 12, lo que significa que se realizaron dos hileras de tamaño 12 de forma aleatoria. En este caso ya logramos ver como la eficiencia del algoritmo realizado usando programación dinámica es mucho mayor, ya que la primera, mantiene el

tiempo sin importar el tamaño de la hilera utilizada, sin embargo en el algoritmo de fuerza bruta ya está tardando 200s en poder resolver el mismo problema, donde esto implica un aumento en 6.7 veces el tiempo de duración con respecto al problema aleatorio de largo 12.

D. Gráfico 7



Este experimento se ejecutó con python3 generador.py -p 2 13 13, lo que significa que se realizaron dos hileras de tamaño 13 de forma aleatoria. Aquí ya se nos sale de control la cantidad de minutos esperados para obtener un resultado por parte del algoritmo de fuerza bruta, tardando 20 minutos en darnos una respuesta, manteniendo la tendencia de aumentar cerca de 6 veces el tiempo tardado en el experimento anterior.

XI. Análisis y Conclusiones Problema 1

Por último a partir de los resultados obtenidos en el experimento y pensando que va a seguir la tendencia de subir en aproximadamente en 6 veces el tiempo cada vez que se aumenta en uno el tamaño de la hilera se estima que la duración para una hilera de largo 14 podría tardar 120 minutos y una de largo 15 podría tardar cerca de 720 minutos, sin embargo la implementación de programación dinámica en ninguno de los casos tardó más de un segundo por lo que se logra ver con mucha claridad la diferencia en cuanto a implementar estos algoritmos se refiere.