Trabajo Práctico: Juegos de Hermanos

El presente trabajo busca evaluar el desarrollo y análisis de diferentes tipos de algoritmos. La fecha de entrega del mismo es el 02/12/24.

Este trabajo práctico contará la historia de vida de dos hermanos.

Primera parte: Introducción y primeros años

Cuando Mateo nació, Sophia estaba muy contenta. Finalmente tendría un hermano con quien jugar. Sophi tenía 3 años cuando Mateo nació. Ya desde muy chicos, ella jugaba mucho con su hermano.

Pasaron los años, y fueron cambiando los juegos. Cuando Mateo cumplió 4 años, el padre de ambos le explicó un juego a Sophia: Se dispone una fila de n monedas, de diferentes valores. En cada turno, un jugador debe elegir alguna moneda. Pero no puede elegir cualquiera: sólo puede elegir o bien la primera de la fila, o bien la última. Al elegirla, la remueve de la fila, y le toca luego al otro jugador, quien debe elegir otra moneda siguiendo la misma regla. Siguen agarrando monedas hasta que no quede ninguna. Quien gane será quien tenga el mayor valor acumulado (por sumatoria).

El problema es que Mateo es aún pequeño para entender cómo funciona esto, por lo que Sophia debe elegir las monedas por él. Digamos, Mateo está "jugando". Aquí surge otro problema: Sophia es muy competitiva. Será buena hermana, pero no se va a dejar ganar (consideremos que tiene 7 nada más). Todo lo contrario. En la primaria aprendió algunas cosas sobre algoritmos greedy, y lo va a aplicar.

Consigna

1. Hacer un análisis del problema, y proponer un algoritmo greedy que obtenga **la solución óptima** al problema planteado: Dados los n valores de todas las monedas, indicar qué monedas debe ir eligiendo Sophia para sí misma y para Mateo, de tal forma que se asegure de **ganar siempre**. Considerar que Sophia siempre comienza (para sí misma).
2. Demostrar que el algoritmo planteado obtiene siempre la solución óptima (desestimando el caso de una cantidad par de monedas de mismo valor, en cuyo caso siempre sería empate más allá de la estrategia de Sophia).
3. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a los tiempos del algoritmo planteado.
4. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a la optimalidad del algoritmo planteado.
5. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.
6. Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Agregar los casos de prueba necesarios para dicha corroboración. Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos.
7. Agregar cualquier conclusión que les parezca relevante.

Segunda parte: Mateo empieza a Jugar

Pasan los años. Mateo ahora tiene 7 años. Los mismos años que tenía Sophia cuando comenzaron a jugar al juego de las monedas. Eso quiere decir que Mateo también ya aprendió sobre algoritmos greedy, y lo comenzó a aplicar. Esto hace que ahora quién gane dependa más de quién comience y un tanto de suerte.

Esto no le gusta nada a Sophia. Ella quiere estar segura de ganar siempre. Lo bueno es que ella comenzó a aprender sobre programación dinámica. Ahora va a aplicar esta nueva técnica para asegurarse ganar siempre que pueda.

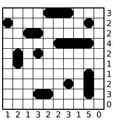
Consigna

1. Hacer un análisis del problema, plantear la ecuación de recurrencia correspondiente y proponer un algoritmo por programación dinámica que obtenga la solución óptima al problema planteado: Dada la secuencia de monedas $m_1, m_2, \cdots, m_n$, sabiendo que Sophia empieza el juego y que Mateo siempre elegirá la moneda más grande para sí entre la primera y la última moneda en sus respectivos turnos, definir qué monedas debe elegir Sophia para asegurarse obtener el **máximo valor acumulado posible**. Esto no necesariamente le asegurará a Sophia ganar, ya que puede ser que esto no sea obtenible, dado por cómo juega Mateo. Por ejemplo, para `[1, 10, 5]`, no importa lo que haga Sophia, Mateo ganará.
2. Demostrar que la ecuación de recurrencia planteada en el punto anterior en efecto nos lleva a obtener el **máximo valor acumulado posible**.
3. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta a los tiempos del algoritmo planteado la variabilidad de los valores de las monedas.
4. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.
5. Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Agregar los casos de prueba necesarios para dicha corroboración (generando sus propios sets de datos). Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos.
6. Agregar cualquier conclusión que parezca relevante.

Tercera parte: Cambios

Los hermanos siguieron creciendo. Mateo también aprendió sobre programación dinámica, y cada uno aplicaba la lógica sabiendo que el otro también lo hacía. El juego de las monedas se tornó aburrido en cuánto notaron que siempre ganaba quien empezara, o según la suerte. Los años pasaron, llegó la adolescencia y empezaron a tener gustos diferentes. En general, jugaban a juegos individuales. En particular, Sophia estaba muy enganchada con un juego inventado en Argentina por Jaime Poniachik (uno de los fundadores de *Ediciones de Mente*) en 1982: La Batalla Naval Individual.

En dicho juego, tenemos un tablero de $n \times m$ casilleros, y k barcos. Cada barco i tiene b_i de largo. Es decir, requiere de b_i casilleros para ser ubicado. Todos los barcos tienen 1 casillero de ancho. El tablero a su vez tiene un requisito de consumo tanto en sus filas como en sus columnas. Si en una fila indica un 3, significa que deben haber 3 casilleros de dicha fila siendo ocupados. Ni más, ni menos. No podemos poner dos barcos de forma adyacente (es decir, no pueden estar contiguos ni por fila, ni por columna, ni en diagonal directamente). Debemos ubicar todos los barcos de tal manera que se cumplan todos los requisitos. A continuación mostramos un ejemplo de un juego resuelto:



Consigna

Para los primeros dos puntos considerar la versión de decisión del problema de La Batalla Naval: Dado un tablero de $n \times m$ casilleros, y una lista de k barcos (donde el barco i tiene b_i de largo), una lista de restricciones para las filas (donde la restricción j corresponde a la cantidad de casilleros a ser ocupados en la fila j) y una lista de restricciones para las columnas (símil filas, pero para columnas), ¿es posible definir una ubicación de dichos barcos de tal forma que se cumplan con las demandas de cada fila y columna, y las restricciones de ubicación?

1. Demostrar que el Problema de la Batalla Naval se encuentra en NP.

2. Demostrar que el Problema de la Batalla Naval es, en efecto, un problema NP-Completo. Si se hace una reducción involucrando un problema no visto en clase, agregar una (al menos resumida) demostración que dicho problema es NP-Completo. Para esto, recomendamos ver ya sea los problemas *3-Partition* o *Bin-Packing*, ambos en su versión unaria. Si bien sería tentador utilizar *2-Partition*, esta reducción no sería correcta. En caso de querer saber más al respecto, consultarnos :-)

3. Escribir un algoritmo que, por backtracking, obtenga la solución óptima al problema (valga la redundancia) en la versión de optimización: Dado un tablero de $n \times m$ casilleros, y una lista de k barcos (donde el barco i tiene b_i de largo) una lista de las demandas de las n filas y una lista de las m demandas de las columnas, dar la asignación de posiciones de los barcos de tal forma que se reduzca al mínimo la cantidad de demanda incumplida. Pueden no utilizarse todos los barcos. Si simplemente no se cumple que una columna que debería tene 3 casilleros ocupados tiene 1, entonces contará como 2 de demanda incumplida. Por el contrario, no está permitido exceder la cantidad demandada. Generar sets de datos para corroborar su correctitud, así como tomar mediciones de tiempos.

4. Escribir un modelo de programación lineal que resuelva el problema de forma óptima. Ejecutarlo para los mismos sets de datos para corroborar su correctitud. Tomar mediciones de tiempos y compararlas con las del algoritmo que implementa Backtracking.

5. John Jellicoe (almirante de la Royal Navy durante la batalla de Jutlandia) nos propone el siguiente algoritmo de aproximación: Ir a fila/columna de mayor demanda, y ubicar el barco de mayor longitud en dicha fila/columna en algún lugar válido. Si el barco de mayor longitud es más largo que dicha demanda, simplemente saltearlo y seguir con el siguiente. Volver a aplicar hasta que no queden más barcos o no haya más demandas a cumplir.

   Este algoritmo sirve como una aproximación para resolver el problema de La Batalla Naval. Implementar dicho algoritmo, analizar su complejidad y analizar cuán buena aproximación es. Para esto, considerar lo siguiente: Sea I una instancia cualquiera del problema de La Batalla Naval, y $z(I)$ una solución óptima para dicha instancia, y sea $A(I)$ la solución aproximada, se define $\frac{A(I)}{z(I)} \leq r(A)$ para todas las instancias posibles. Calcular $r(A)$ para el algoritmo dado, demostrando que la cota está bien calculada. Realizar mediciones utilizando el algoritmo exacto y la aproximación, con el objetivo de verificar dicha relación. Realizar también mediciones que contemplen volúmenes de datos ya inmanejables para el algoritmo exacto, a fin de corroborar empíricamente la cota calculada anteriormente.

6. **Opcional**: Implementar alguna otra aproximación (o algoritmo greedy) que les parezca de interés. Comparar sus resultados con los dados por la aproximación del punto anterior. Indicar y justificar su complejidad. No es obligatorio hacer este punto para aprobar el trabajo práctico (pero sí resta puntos no hacerlo).

7. Agregar cualquier conclusión que parezca relevante.

Entrega

Completar el formulario de entrega con los integrantes y el link al repositorio donde se encuentre el código fuente, y donde debe encontrarse el informe en formato PDF. Debe ser claro cómo ejecutar el programa pasando por parámetro un set de datos como los que se dan de ejemplo. No debe ser necesario tener la solución esperada (sino, ¿para qué programaríamos un programa para resolver un problema cuya respuesta ya conocemos?). Esto puede ser aclarado dentro del `README.md` del repositorio, u otra ubicación que les parezca clara.

El informe debe ser:

- Autocontenido: es decir, no debe ser necesario ponernos a buscar el código por diferentes lugares. El código que debe ser incluido es el código de los algoritmos a desarrollar (no es de interés un main, o el procesamiento de archivos), y estos deben estar incluidos donde consideren correcto dado el desarrollo del informe, para su entendimiento.
- Tener todo el análisis correspondiente.
- Ser realizado en un formato profesional. Para esto, les brindamos un template en LaTeX para que puedan utilizar (también se encuentra en la home de la página del curso). No es necesario que lo sigan al pie de la letra, es simplemente un ejemplo que tiene varias cosas que pueden llegar a utilizar de LaTeX. Si ya conocen LaTeX no es necesario que lo utilicen, o mismo si utilizan algún otro formato (e.g. Markdown con Pandoc), pero recomendamos su revisión para que vean cosas que no deben de faltar. Por supuesto, pueden trabajar localmente como usar Overleaf o cualquier otra herramienta. El objetivo de darles el template no es la de limitar la creatividad, sino de asegurarnos que se cumplan lineamientos básicos sobre lo que se espera de una entrega de un informe en la facultad.
- En caso de ser necesarias reentregas, por favor agregar las modificaciónes en un Anexo al final del informe. No modificar lo hecho anteriormente. La excepción a esto sería si hay que rehacer una enorme mayoría de lo escrito.

La nota del trabajo práctico tendrá en cuenta tanto la presentación y calidad de lo presentado, como también el desarrollo del trabajo. No será lo mismo un trabajo realizado con lo mínimo indispensable, que uno bien presentado, analizado, y probado con diferentes volúmenes, set de datos, o estrategias de generación de sets, en el caso que corresponda.