Técnicas de Diseño de Algoritmos
Primer Cuatrimestre 2024
Primer Parcial (T. Mañana)

Nombre y Apellido	L.U.	N°Orden

*Duración: **3 horas**. Este examen es a **libro cerrado**. Todas las preguntas tienen $k \geq 1$ opciones correctas. Las respuestas donde se marquen exactamente esas k dan 2 puntos, aquellas donde se marquen al menos $\lceil k/2 \rceil$ correctas y más correctas que incorrectas dan 1 punto, las que tengan menos de $\lceil k/2 \rceil$ correctas o igual o más incorrectas que correctas dan 0, y si todas las marcadas son incorrectas dan -1 punto. Para aprobar el parcial se deben sumar $p \geq 12$ puntos y la nota será $5p/12$.*

Pregunta 1 ¿Cuáles de estas afirmaciones son verdaderas para soluciones de un problema basadas en la técnica de backtracking?

En algunos casos anticipan que ciertos subconjuntos de soluciones candidatas del problema no serán factibles y evita considerarlos.

En algunos casos evalúan todas las soluciones candidatas posibles del problema.

Tienen una complejidad temporal estrictamente menor que una solución por exploración exhaustiva ("fuerza bruta").

Se pueden usar en problemas para los que no se conoce un algoritmo polinomial.

Explotan el fenómeno de superposición de subproblemas.

Siempre su complejidad temporal es exponencial respecto al tamaño de la entrada.

Pregunta 2

Dada una matriz $M \in \mathbb{Z}^{n \times m}$ y $p_0 = (f_0, c_0)$ $(0 \leq f_0 < n,\ 0 \leq c_0 < m)$, un *$k$-camino de d-saltos desde p_0* es una secuencia S de posiciones $p_1, \ldots, p_k$ de M tal que $p_i \neq p_j$ para $0 \leq i < j \leq k$, y p_i está a d lugares de p_{i-1} en alguna dirección (arriba, abajo, izquierda, derecha) para $1 \leq i \leq k$. El *valor* v de un k-camino de d-saltos S es $M[f_0, c_0] + \sum_{(f,c) \in S} M[f, c]$. Por ejemplo, sea M la matriz de 4×5 a continuación y consideremos dos 3-caminos de 2-saltos desde $p_0 = (0, 0)$.

$$M = \begin{pmatrix} -3 & 2 & 6 & 5 & -1 \\ 4 & 1 & -3 & 2 & 6 \\ 3 & 5 & -4 & 2 & 2 \\ 4 & 6 & 1 & 7 & -5 \end{pmatrix}$$

$S_1 = [(2,0),(2,2),(0,2)], v(S_1) = 2$
$S_2 = [(0,2),(0,4),(2,4)], v(S_2) = 4$
$D_1 = (\text{abajo}, \text{derecha}, \text{arriba})$
$D_2 = (\text{derecha}, \text{derecha}, \text{abajo})$

Dados M, k, d, y p_0, podemos considerar el problema de determinar el k-camino de d-saltos desde p_0 de valor máximo. Por ejemplo, el óptimo para M, con $p_0 = (0, 0)$, $d = 2$ y $k = 4$ es S_2. Para resolver este problema usando backtracking, podemos considerar cada secuencia S de k posiciones (f, c) $(0 \leq f < n,\ 0 \leq c < m)$, construída posición a posición, como una solución candidata cuya factibilidad se verifica al final (por ejemplo en M podría construir la posibilidad $S^* = [(3,0),(3,3),(1,1)]$ y finalmente descartarla por hacer movimientos ilegales). Alternativamente, podemos codificar cada solución candidata como una secuencia D de direcciones (Para $q \in D$, $q \in \{$"arriba","abajo","izquierda","derecha"$\}$), construirla agregando una dirección por vez (en el ejemplo se muestran las codificaciones D_1 y D_2 para S_1 y S_2), y nuevamente verificar al completarla que la secuencia sea un k-camino de d-saltos. Suponiendo que no se aplican podas, ¿Cuál es la cota más ajustada para la cantidad de hojas del árbol de backtracking en cada codificación?

S: $O((nm)^k)$	S: $O(m^{nk})$	D: $O(2^{2k})$	D: $O(mn^{2k})$
S: $O(k^{mn})$	S: $O(n^{mk})$	D: $O(2^{mn})$	D: $O(mn^2)$

Pregunta 3 Supongamos tres vectores de N enteros no negativos $p_1, \ldots, p_N$, $b_1, \ldots, b_N$ y $d_1, \ldots, d_N$, y sea Φ un problema que se puede resolver haciendo el llamado $f(N, K)$ a la función recursiva f definida como

$$f(i,k) = \begin{cases} -\infty & \text{si } k < 0 \\ i & \text{si } i \leq 1 \\ \max\{f(i-1, k-p_i) + f(i-2, k) + b_i, f(i-2, k-p_i) + f(i-1, k) + d_i\} & \text{c.c.} \end{cases}$$

Queremos implementar una solución por medio de programación dinámica para computar f, usando una tabla M como estructura de memoización. Nuestro objetivo es minimizar la complejidad espacial preservando los accesos a M en $O(1)$, ¿Cuáles son las dimensiones mínimas que podríamos darle a M?

$O(N \times K)$	$O(N^2)$
$O(N)$	$O(K)$

Pregunta 4 Considere las siguientes afirmaciones y determine si son verdaderas para la estrategia de resolución *top-down* o *bottom-up* (o a ambas) de una solución de programación dinámica para un problema determinado.

1. Permite, en las circunstancias adecuadas, ahorrar complejidad espacial.

   Top-Down Bottom-Up Ninguna

2. Explota el fenómeno de *superposición de subproblemas*.

   Top-Down Bottom-Up Ninguna

3. Su implementación más directa puede, en las circunstancias adecuadas, no requerir computar todas las subinstancias del problema con parámetros más cercanas al caso base.

   Top-Down Bottom-Up Ninguna

4. Es siempre la estrategia más eficiente de las dos en complejidad temporal.

   Top-Down Bottom-Up Ninguna

5. En problemas de optimización combinatoria, además de devolver el valor del óptimo, permite armar la lista de decisiones que llevan a ese valor.

   Top-Down Bottom-Up Ninguna

Pregunta 5 Se tiene un array A de n elementos (por simplicidad, consideramos n potencia de 2) del cual se quiere obtener su máximo y su mínimo. Ciertamente, una manera de lograrlo es recorriendo A secuencialmente y comparando cada elemento contra el máximo y el mínimo que hayamos encontrado. Si el tipo de los elementos en A tiene una comparación costosa, tiene sentido considerar una manera alternativa de obtener estos valores. Se nos propone utilizar el *método del torneo*, donde recursivamente obtenemos el máximo y el mínimo entre los elementos de la primera mitad de A, el máximo y el mínimo de los elementos de la segunda mitad de A, y los comparamos apropiadamente. Sea $C(n)$ la cantidad de comparaciones que requiere el método del torneo para un array de n elementos. Indique el valor asintótico de $C(n)$ y decida si efectivamente se reduce el orden de comparaciones respecto al recorrido lineal del arreglo.

$C(n) = C(n/2) + O(1)$	Se reduce el orden de comparaciones.
$C(n) = 2C(n/2) + O(1)$	Las cantidades de comparaciones son iguales en orden.
$C(n) = 2C(n/2) + O(n)$	Se incrementa el orden de comparaciones.
$C(n) = C(n/2) + O(n)$	

Pregunta 6 Para las siguientes recurrencias marque el orden de complejidad correcto.

- $T(n) = 16T(n/4) + n$

$\Theta(\log n)$	$\Theta(n \log n)$	$\Theta(n^2 \log n)$
$\Theta(n)$	$\Theta(n \log^2 n)$	$\Theta(n^2)$

- $T(n) = 6T(n/3) + n^2 \log n$

$\Theta(n)$	$\Theta(n^{\log_3(6)})$	$\Theta(n^2)$
$\Theta(\log n)$	$\Theta(n \log n)$	$\Theta(n^2 \log n)$

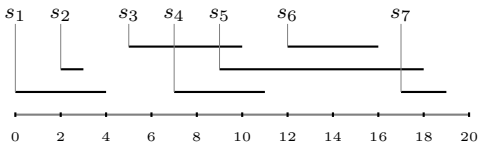
- $T(n) = 3T(n/3) + \sqrt{n}$

$\Theta(\log n)$	$\Theta(\sqrt{n})$	$\Theta(n^2)$
$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2 \log n)$

- $T(n) = 4T(n/2) + n^2$

$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2 \log n)$
$\Theta(\log n)$	$\Theta(n \log^2 n)$	$\Theta(n^2)$

Pregunta 7 Últimamente se ha instalado la difusión de contenido por medio de *streaming*, lo cual ha derivado en una oferta cada vez más grande de canales e *influencers* para mirar. Teodoro armó un conjunto $A = s_1, \ldots, s_m$ con los *streams* en los que está interesado y sus horarios, pero pronto encontró que muchos de ellos se superponen. Como no quiere cambiar de stream durante una transmisión, se propone determinar, al menos, la cantidad de streams que puede ver en el día sin cambiarse una vez que comenzó a ver a uno. Por ejemplo, si tenemos



$s_1 = [0, 4]$ $s_6 = [12, 16]$
$s_2 = [2, 3]$ $s_7 = [17, 19]$
$s_3 = [5, 10]$
$s_4 = [7, 11]$ Aquí, el máximo de streams
$s_5 = [9, 18]$ es 4 (s_1, s_3, s_6, s_7).

Teodoro considera que encontró un algoritmo basado en una estrategia *greedy* que le permite conocer el número de streams máximo que puede ver en un día. Su estrategia es elegir el stream s_1 que comience primero en el día, mirarlo hasta que termine, y elegir aquel que comience primero una vez que termine s_1. Su fundamento para decir que esto es correcto hace inducción global en el cardinal del conjunto A de streams que quedan por ver.
Caso base: Claramente, si hay un stream, es el primero.
Paso inductivo: Si hay más de un stream, ciertamente alguno de ellos es el primero, y, si quito ese stream más todos los que compartan horario con él, tendré un conjunto de streams A' de cardinal estrictamente menor. Por hipótesis inductiva, la estrategia me da la cantidad máxima de streams que puedo ver en A', y como todos los elementos que había quitado eran incompatibles, claramente la máxima cantidad de streams que se pueden ver es 1 más de la cantidad máxima que se puedan ver de los de A'.
¿Es correcto el algoritmo propuesto por Teodoro? ¿Vale el argumento que presenta?

- El algoritmo no es correcto, hay un error lógico en el paso inductivo.
- El algoritmo no es correcto, pero el paso inductivo de la demostración sería correcto si se hubieran probado otros casos base.
- El algoritmo y la demostración son correctos.

¿Qué estrategia(s) greedy da en efecto el valor óptimo buscado por Teodoro?

- Seleccionar el stream más corto (cualquiera si hay más de uno), descartar todos los streams en conflicto con este, y realizar la misma selección para el conjunto de streams restantes.
- Seleccionar el stream que termine primero (cualquiera si hay más de uno), descartar todos los streams en conflicto con este, y realizar la misma selección para el conjunto de streams restantes.
- El propuesto por Teodoro.
- Seleccionar el stream que empiece último (cualquiera si hay más de uno), descartar todos los streams en conflicto con este, y realizar la misma selección para el conjunto de streams restantes.

Pregunta 8 Se tiene un conjunto de archivos $A_1, \ldots, A_n$ de tamaños $s_1, \ldots, s_n$ (dos archivos pueden tener el mismo tamaño), a los que se les quiere aplicar un *merge*, es decir, fusionarlos en un solo archivo de tamaño $\sum_{i=1}^{n} s_i$. Para esto, contamos con una función MERGE que toma dos archivos A_i y A_j y realiza el merge entre ellos, generando un nuevo archivo con tamaño $k = s_i + s_j$, y costo $O(k)$. La estrategia con la que elegimos los pares de archivos a los que hacerles merge cambia el costo total; por ejemplo, si tenemos los archivos $A_1, \ldots, A_4$ con tamaños 40, 50, 70 y 90, respectivamente, podríamos hacer MERGE(A_1, A_2) para generar A_5 de tamaño 90, luego MERGE(A_3, A_5) para generar A_6 de tamaño 160, y finalmente MERGE(A_4, A_6) generando A_7, el archivo final, de tamaño 250. El costo total en este caso habrá sido de $90 + 160 + 250 = 500$. Alternativamente, si hiciéramos $A_5 =$ MERGE(A_1, A_2), $A_6 =$ MERGE(A_4, A_5), y $A_7 =$ MERGE(A_3, A_6), el costo total sería 520. En el caso general, ¿Cuál(es) de estas estrategias nos permite(n) hacer el merge de los archivos con el menor costo y cuál es su complejidad ajustada?

Ordenar la lista de archivos por tamaño y juntarlos de a pares para hacerles MERGE. Luego tomar la nueva lista de $n/2$ elementos y repetir el proceso hasta que quede sólo un archivo (se puede asumir que n es potencia de 2).	$O(n^2)$
Mantener una estructura de datos ordenada por tamaño de archivo y, hasta que quede un solo archivo, tomar los dos más chicos para hacerles MERGE y reinsertar el resultado en la estructura.	$O(n^2 \log n)$
Ordenar los archivos por tamaño, tomar el mínimo, y usarlo de acumulador haciendo MERGE con cada uno del resto en orden de tamaño.	$O(n \log n)$

Pregunta 9 Se cuenta con la siguiente afirmación sobre grafos: "Para todo grafo G de n vértices, si G tiene exactamente $n - 1$ aristas, entonces G es un árbol.", y se presenta una demostración por inducción para la misma.
Caso Base: $n = 1$, el grafo trivial tiene 0 aristas y es un árbol.
Paso Inductivo: Sea G con n vértices y $n - 1$ aristas. Queremos ver qué sucede con el grafo G' resultante de agregar un vértice v a G. Si v tiene grado distinto de 1, entonces G' tiene una cantidad distinta de $(n+1) - 1 = n$ aristas y la propiedad vale por falsedad del antecedente. Si v tiene grado 1, entonces observar que por hipótesis inductiva G es un árbol, y al agregarle una hoja, sigo teniendo un árbol.
¿Son la afirmación y su demostración correctas?

La demostración no es correcta porque el paso inductivo requiere que el caso base sea distinto al elegido.	La afirmación no es verdadera.
La demostración es correcta.	La afirmación es verdadera.
La demostración no es correcta porque el paso inductivo no está correctamente planteado.	

Pregunta 10 Sea G un grafo. Suponiendo que puedo implementar a G como lista de adyacencias o como matriz de adyacencia (sin estructuras adicionales), determinar en cuál(es) se pueden realizar las siguientes operaciones en la complejidad descrita.

- Dados dos vértices v, w, decidir si son vecinos en $O(1)$ tiempo.

  Lista de Adyacencias Matriz de Adyacencias Ninguna

- Listar todas las aristas del grafo en $O(n + m)$ tiempo.

  Lista de Adyacencias Matriz de Adyacencias Ninguna

- Devolver la lista de aristas del grafo complemento de G en $O(n^2)$ tiempo.

  Lista de Adyacencias Matriz de Adyacencias Ninguna

- Devolver una representación de G por lista de adyacencias tal que para todo $v \in V(G)$, $N(v)$ esté ordenado por grado ascendente, en $O(n + m)$ tiempo.

  Lista de Adyacencias Matriz de Adyacencias Ninguna

Pregunta 11 Cuando se realiza un recorrido DFS sobre un grafo conexo no dirigido G, se obtiene un árbol T que cumple que toda arista (v, w) de G que no está en T es una *back-edge*, es decir, que v es ancestro de w o viceversa. Si hacemos el algoritmo de DFS sobre un grafo dirigido D, desde un vértice r que alcanza a todos los vértices, obtenemos un árbol T'. Una arista dirigida $v \to w$ de D que no pertenece a T', ¿De cuál(es) de estos tipos pueden ser?

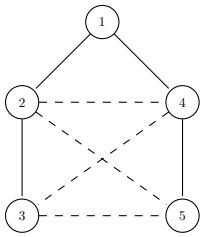
Cross-edge a derecha (w pertenece a una rama de T' que se explora después que la de v).

Forward-edge (v es ancestro de w).

Back-edge (w es ancestro de v).

Cross-edge a izquierda (w pertenece a una rama de T' que se explora antes que la de v).

Pregunta 12 Dados el grafo G y un árbol generador T de G obtenido por medio de BFS, como se muestra en la figura, indicar desde cuál(es) vértice(s) es posible que se haya eraizado el recorrido.



(Las líneas sólidas representan aristas pertenecientes al AG T las punteadas, a $G \setminus T$.)

2

1

Ninguno

3

5

4