Técnicas de Diseño de Algoritmos
Algoritmos y Estructuras de Datos III

Recuperatorio 1er Parcial - 1er C 2024. T. Noche

Nombre y Apellido	LU	#Ord
.....	.....	.....

Tema	Ej A / B	#hojas
A	.....	.....

Duración : 4 horas
*Este examen es a **libro cerrado**.*
Para aprobar el recuperatorio se debe alcanzar una nota de 50 *puntos.*

Preguntas opción múltiple (60 ptos.)

Pregunta 1 Julieta encontró una caja con n piezas distinguibles de dominó, y se propuso encontrar el camino de mayor longitud que puede hacerse con las mismas[*]. Su estrategia para generar todos los candidatos a caminos consiste en probar cada posible ordenamiento de las piezas con cada posible orientación de cada pieza en en la permutación, para luego buscar el camino más largo que quedó en esa configuración. Por suerte, conoce una técnica que le permite iterar por cada configuración, sin repetir, en tiempo constante. Tras unos minutos empleando esta estrategia se le ocurrió la siguiente idea: va a empezar de cero e ir agregando en cada paso una nueva pieza a la derecha del camino actual que está armando, probando una a una todas las piezas que sean compatibles con el camino armado hasta el momento (es decir, que se puedan orientar para que un lado coincida con la pieza más a la derecha del camino, considerando que la primera se puede orientar de las dos maneras). Cada vez que se quede sin piezas compatibles para seguir poniendo, volverá al último paso donde pueda poner alguna pieza que aún no haya probado en esa posición, y seguirá el camino utilizando esa pieza. Finalmente, se quedará con la mayor longitud que haya alcanzado formar durante este proceso. Decidir:

- ☐ La segunda estrategia propuesta por Julieta no es correcta, existen casos para los que proveerá una longitud de cadena de dominós menor a la máxima posible.
- ☒ La primera estrategia implica revisar $2^n \cdot n!$ soluciones candidatas, y toma $O(n)$ tiempo obtener el camino más largo en cada una.
- ☐ En la segunda estrategia, para todo $1 \leq k \leq n-1$, una vez que Julieta llegó a armar un camino de longitud k, para la $(k+1)$-ésima pieza puede tener $2^{n-k}(n-k)$ opciones distintas posibles a considerar.
- ☐ Una cota para la complejidad temporal de la segunda estrategia definida por Julieta es $O(2^n \cdot n)$.

[*]Las piezas de dominó son rectangulares y tienen un número de cada lado. Una pieza se puede unir a otra si coinciden en algún número, y se unen a través ese mismo número. Por lo tanto, el número de ese lado ya no se puede usar para unir otra pieza. Por ejemplo, si tenemos las piezas $(2,3)$, $(1,3)$ y $(1,4)$, podemos unrlas en el camino $(2,3),(3,1),(1,4)$. La longitud de un camino es igual a la cantidad de piezas en el camino.

Pregunta 2 Si se tiene una recursión de la forma

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

¿Cuáles de los siguientes valores para a, b y $f(n)$ aseguran que la complejidad final sea $\Theta(\sqrt{n}\log n)$?

- [x] $a = 4, b = 16, f(n) = n^{\frac{1}{2}}$.
- [ ] $a = 2, b = 2, f(n) = n^{\frac{1}{2}}\log n$.
- [x] $a = 2, b = 8, f(n) = n^{\frac{1}{2}}\log n$.
- [ ] $a = 4, b = 2, f(n) = 1$.
- [ ] $a = 3, b = 9, f(n) = n^{\frac{1}{4}}$.

Pregunta 3 Alfredo se fue de viaje una vez más, y a la vuelta espera comprar alfajores. Conoce las n ciudades que va a visitar de forma ordenada en su camino a Buenos Aires (y las numera de 1 a n), y en cada una de ellas puede comprar entre 0 y m alfajores. Denotamos como $p_{i,k}$ a lo que cuesta comprar k alfajores en la ciudad i, para $1 \leq i \leq n$ y $1 \leq k \leq m$, donde $p_{i,0} = 0$.
Inicia el viaje con un capital P, y su objetivo es maximizar la cantidad de alfajores que tiene al llegar a Buenos Aires. Para resolver este problema propone la siguiente función recursiva:

$$f(i,c) = \begin{cases} -\infty & c < 0 \\ 0 & i = n+1 \\ \text{máx}_{0 \leq k \leq m}\{f(i+1, c-p_{i,k}) + k\} & \text{caso contrario} \end{cases}$$

que (supuestamente) indica "La mayor cantidad de alfajores que Alfredo puede comprar empezando en la ciudad i, con capital c". Alfredo nos dice que la solución al problema se obtiene con el llamado $f(1, P)$. Decidir:

- [x] El algoritmo propuesto por Alfredo es correcto.
- [ ] Si D es una cota ajustada de la cantidad de posibles subllamados con argumentos distintos que ocurren al llamar a $f(1, P)$ en una implementación directa de la función f, entonces una cota ajustada para la complejidad de un algoritmo de programación dinámica implementando f es $O(D)$.
- [ ] La cantidad de llamados distintos que pueden hacerse a la función f en la recursión al llamar a $f(1, P)$ es del orden de $O(n^a)$ para alguna constante a (es decir, es polinomial en n).
- [x] Al llamar la función como $f(1, P)$ podrían ocurrir m^n subllamados.

Pregunta 4 ¿Cuál/es de las siguientes afirmaciones describe/n a las estrategias greedy?

- [x] Para poder afirmar fuera de toda duda que una estrategia greedy es un algoritmo que resuelve un problema dado necesitamos proveer una demostración.
- [x] Una forma de demostrar que una estrategia greedy da con el valor óptimo en un problema de optimización consiste en probar que la solución de la estrategia es por lo menos tan buena como cualquier solución posible.
- [ ] Una estrategia golosa encuentra una solución óptima para un problema dado si y solamente si se cuenta con una demostración que pruebe ese hecho.
- [ ] Las estrategias greedy aseguran que la solución final será una solución óptima.

Pregunta 5 Aye está remodelando su casa, y quiere cubrir el piso de su cuarto de largo n centímetros con hileras de placas de madera. Compró un tablón de n centímetros, el cual quiere usar para la primera hilera. Ella sabe que no puede poner un segmento de madera de más de ℓ centímetros en el piso, porque se rompe. Quiere entonces recortar este tablón de forma que cada segmento tenga a lo sumo tamaño ℓ. Además, en algunos lugares de la habitación va a poner muebles que cubran algunas partes del piso. Se le ocurrió que si cortaba el tablón sólo en lugares que vayan a quedar cubiertos, iba a quedar todo mucho más lindo. Aye ya tiene proyectado dónde va a poner los muebles, así que hizo marcas en la madera que corresponden a lugares en donde podría hacer los cortes. Aye quiere resolver el tema rápido, por lo que propone la siguiente estrategia para minimizar la cantidad de cortes:

1. Si el tablón tiene longitud menor o igual a ℓ, no hace nada.
2. En caso contrario, elige el punto de corte más lejano a distancia menor o igual que ℓ posible y corta allí.
3. Luego, repite este procedimiento con el tablón de madera restante a la derecha del corte.

Aye afirma que esta estrategia la lleva a una manera de cortar la madera la mínima cantidad de veces. Incluso, pensó la siguiente demostración de que cualquier esquema requiere tantos o más cortes que el suyo: Consideremos un esquema de k cortes válido cualquiera C y el esquema de q cortes de Aye C^A, y sean $c_1 < \ldots < c_k$ y $c_1^A < \ldots < c_q^A$ los puntos donde se hacen los cortes de C y C^A medidos desde el borde izquierdo del tablón, respectivamente. Si $q = 0$, es porque el tablón tiene longitud menor o igual a ℓ y Aye no hace ningún corte, así que $0 = q \leq k$. Si $q > 0$, por definición de C_A el tablón tiene longitud mayor a ℓ y en consecuencia todo esquema válido tiene al menos un corte. Sea i tal que c_i es el máximo corte en C con $c_i \leq \ell$. Vemos que $c_i \leq c_1^A$, pues por definición c_1^A está en la marca máxima que no se pase de ℓ. Si $i = k$, entonces $q = 1$, pues con el único corte c_1^A alcanza, por lo que $1 = q \leq k$. Si $i < k$, entonces $c_{i+1} - c_1^A \leq \ell$, pues $c_{i+1} - c_i \leq \ell$ por ser C es un esquema de corte válido y $c_i \leq c_i^A$. Luego, podemos obtener un esquema de corte C' que sea $c_1^A, c_{i+1}, \ldots, c_k$ tal que $|C'| \leq k$ y que comparte su primer corte con C^A. Aplicando esta idea sobre la parte derecha del tablón luego de cortar en c_1^A de manera inductiva, podemos deducir que C^A tiene una cantidad de cortes menor o igual que cualquier esquema posible.
¿Cuáles de estas afirmaciones son verdaderas?

- [x] El algoritmo es correcto y la demostración es correcta.
- [ ] El algoritmo es incorrecto, existen soluciones con menos cortes.
- [ ] El algoritmo es correcto, pero la demostración es incorrecta.
- [x] La estrategia propuesta por Aye puede implementarse en $O(t)$, donde t es la cantidad de marcas que hizo Aye sobre la madera (y asumiendo que el input es una lista ordenada con estas posiciones).

Pregunta 6 Román quiere resolver el problema SUBSET-SUM: dada una lista de números naturales $A = \{a_1, \ldots, a_n\}$ de longitud par mayor a cero y un número natural $k \in \mathbb{N}_0$, quiere decidir si existe un subconjunto $S \subseteq A$ tal que $\sum_{a_i \in S} a_i = k$. Para esto, propone inicialmente un algoritmo que implementa la siguiente recursión:

$$ss(i,j) = \begin{cases} \texttt{True} & i = n+1 \wedge j = 0 \\ \texttt{False} & (i = n+1 \wedge j > 0) \vee (j < 0) \\ ss(i+1,j) \vee ss(i+1, j-a_i) & \text{caso contrario} \end{cases}$$

y dice que $ss(1,k)$ le dará la solución al problema. Decidir:

- ☐ Una cota ajustada para la complejidad temporal de este algoritmo es $O(n2^n)$.
- ☒ El algoritmo propuesto por Román es correcto.
- ☐ Para cualquier lista A de números naturales de longitud par vale que al llamar a la función ss con parámetros 1 y k habrá por lo menos 2^n subllamados.

Pregunta 7 Para mejorar su algoritmo, Román propone la siguiente estrategia: dividirá el conjunto A en dos mitades $A_1 = \{a_1, \ldots, a_{n/2}\}$ y $A_2 = \{a_{n/2+1}, \ldots, a_n\}$, y para cada mitad calculará, con backtracking, el conjunto T_i, que denota todas las sumas que se pueden obtener usando valores de A_i (formalmente, $T_i = \{m \in \mathbb{N}_0 : \exists S \subseteq A_i, \sum_{a_i \in S} a_i = m\}$). Luego, ordenará T_1 y T_2 y usará búsqueda binaria para para detectar si existen dos números $t_1 \in T_1$ y $t_2 \in T_2$ tales que $t_1 + t_2 = k$. Decidir:

- ☐ La complejidad espacial de este algoritmo es $O(n)$.
- ☒ Una cota para la complejidad temporal de este algoritmo es $O(n2^{\frac{n}{2}})$.
- ☒ Este nuevo algoritmo propuesto por Román es correcto.

Pregunta 8 Sea P_4 el grafo de 4 nodos que forma un camino, i.e. $P_4 = (\{v_1, v_2, v_3, v_4\}, \{v_1v_2, v_2v_3, v_3v_4\})$. Tuki quiere diseñar un algoritmo que detecte, dado un grafo $G = (V, E)$, si G contiene a P_4 como subgrafo inducido. Para esto, define el siguiente algoritmo:

Algoritmo 1 Procesar grafo

```
1: function DetectarP4(G = (V, E))
2:     for vw ∈ E do
3:         if N(v) \ N(w) ≠ ∅ and N(w) \ N(v) ≠ ∅ then
4:             return True
5:         end if
6:     end for
7:     return False
8: end function
```

Decidir:

- [x] Si G viene dado como matriz de adyacencias entonces el algoritmo se puede implementar en $O(n^3)$.
- [ ] Si el grafo G no tiene un P_4 como subgrafo inducido entonces el algoritmo propuesto por Tuki devuelve `False`.
- [x] Si el grafo G tiene un P_4 como subgrafo inducido entonces el algoritmo propuesto por Tuki devuelve `True`.
- [x] Si G viene dado como lista de adyacencias entonces el algoritmo se puede implementar en $O(nm)$.

Pregunta 9 Un grafo $G = (V, E)$ es **bipartito** si y sólo si sus vértices V se pueden dividir en dos conjuntos disjuntos U y W, de tal manera que cada arista en E conecta un vértice en U con un vértice en W. En tal caso, decimos que G se puede **bipartir** en U y W. ¿Cuál/es de las siguientes afirmaciones es/son verdadera/s para todo grafo bipartito G?

- [ ] G no contiene ciclos de longitud par.
- [x] G no contiene ciclos de longitud impar.
- [x] La suma de los grados de los vértices de U es igual a la suma de los de W.
- [x] Si aplico DFS desde cualquier vértice v de G, toda *back-edge* siempre unirá dos vértices en niveles del árbol de distinta paridad.
- [ ] Si los nodos de G se pueden bipartir en conjuntos U y W luego $|U|$ es par o bien $|W|$ lo es.

Pregunta 10 En una red de transporte aéreo se modelan n distintos aeropuertos y las conexiones entre ellos. Decimos que un aeropuerto a_1 está conectado con a_2 si existe un vuelo directo de a_1 a a_2 (notemos que esta relación puede no ser simétrica). Se nos pide encontrar la cantidad mínima de vuelos que se debe hacer para llegar desde algunos aeropuertos a Ezeiza, y para simplificarnos el trabajo nos aseguran que solo van a estar interesados en $r < n$ aeropuertos y que cada aeropuerto está conectado, a lo sumo, a k otros aeropuertos. ¿Cuál/es de los siguientes algoritmos puede/n usarse para resolver este problema?

- [x] BFS.
- [ ] DFS.

¿Y cuál es la cota más ajustada de complejidad en la que puede resolverse?

- [ ] $O(n^2)$
- [x] $O(kn)$
- [ ] $O(rk + n)$
- [ ] $O(rn)$

Ejercicio a desarrollar (40 ptos.)

Marcar con una cruz el cuadrado (☐) del ejercicio que vas a entregar.
Solo se entrega 1 de los 2 ejercicios.

☐ Problema A

A Alfredo le gustan tanto los alfajores que compró la prestigiosa fábrica de Guaimayén™. Ahora debe encargarse de la distribución de los alfajores a lo largo del Área Metropolitana de Buenos Aires.

Cada camión debe visitar una secuencia $k_1, \ldots, k_n$ de n kioscos, y los debe visitar en ese orden. Cuando salen de la fábrica llevan una cantidad A de alfajores, los cuales ofrecen a los distintos kioscos siguiendo el orden. Se sabe que el kiosco k_i quiere comprar c_i alfajores, y está dispuesto a pagar p_i por los mismos. Más aún, el kiosco k_i solo comprará si el camión le ofrece **exactamente** c_i alfajores (es decir, compra la cantidad c_i, o bien no compra nada).

Aparte, cada camión tiene un tanque de 90 litros de combustible, y en el camino entre un kiosco y su consecutivo gasta siempre 10 litros. Al llegar a un kiosco los camiones pueden decidir cargar combustible en vez de vender los alfajores, lo cual hacen gratis debido a un convenio entre Guaimayen™ y IP-F. Naturalmente, el camión no debe quedarse sin combustible.

Alfredo quiere saber cuál es la máxima ganancia que puede obtener un camión que sale de la fábrica con A alfajores y un tanque lleno (de 90 litros) teniendo en cuenta que va a visitar los kioscos $k_1, \ldots, k_n$ en orden, que cada kiosco k_i solo está dispuesto a comprar c_i alfajores a precio p_i y que el camión no debe quedarse sin combustible.

a) Dar una función recursiva que resuelva el problema, justificando su correctitud.

b) Indicar bajo qué condiciones la función propuesta tiene la propiedad de superposición de subproblemas.

c) Describir un algoritmo de programación dinámica de complejidad temporal $O(nA)$ que implementa la función del inciso *a)*. Justificar la complejidad del mismo.

☐ Problema B

Dado un dígrafo $G = (V, E)$ decimos que G es *casi DAG* si es un DAG o bien si existe una arista $e \in E$ tal que $G \setminus \{e\} = (V, E \setminus \{e\})$ es un DAG.

a) Dar un algoritmo que detecte, dado un dígrafo $G = (V, E)$ débilmente conexo, si G es *casi DAG* en complejidad temporal $O(|E|^2)$. Justificar su correctitud y complejidad.

b) Sea C un ciclo de G. Demostrar que G es un *casi DAG* si y solamente si existe una arista $e \in C$ tal que $G \setminus \{e\}$ es un DAG.

c) Dar un algoritmo de complejidad temporal $O(|V| + |E|)$ que dado un digrafo $G = (V, E)$ encuentre un ciclo del mismo, si es que lo tiene. Justificar su correctitud y complejidad.

d) Dar un algoritmo que resuelva el mismo problema que el del inciso *a*), pero de complejidad $O(|V||E|)$. Justificar su correctitud y complejidad.