Técnicas de Diseño de Algoritmos
Ejercicios Tipo Parcial

Duración el día del exámen: 3 horas.
Este exámen será a ***libro cerrado****.*

Las respuestas correctas dan un puntaje de $\frac{10}{n}$ *puntos, las incorrectas dan un puntaje de* $-\frac{5}{n}$ *puntos. Para aprobar el parcial se debe alcanzar una nota de* 5 *(cinco) (este criterio puede sufrir modificaciones, será debidamente informado)*

Pregunta 1 Se tiene un conjunto con n elementos, a los cuales se quiere separar en subconjuntos de tamaño $n/2$ disjuntos A y B. Para cada elemento se conoce el puntaje de ponerlo en el conjunto A y el puntaje de ponerlo en el conjunto B. Se quiere buscar una asignación que maximice el puntaje total de los elementos en el conjunto A, en primer lugar, y los de B, en segundo. Si se quiere resolver este problema usando backtracking, y suponiendo que *no hay podas*, ¿Qué cantidad de hojas tiene el árbol de backtracking?

- ☐ 2^{2n}
- ☐ 2^n
- ☐ $n!$
- ☐ $2n^2$

Pregunta 2 Dado un vector P de n valores $p_1, \ldots, p_n$, deseamos implementar un algoritmo de programación dinámica que responde la siguiente formulación recursiva donde $c, j \leq n$.

$$\text{mgn}(c, j) = \begin{cases} -\infty & \text{si } c < 0, c > j \\ 0 & \text{si } c = j = 0 \\ \max\{\text{mgn}(c-1, j-1) - p_i, \text{mgn}(c+1, j-1) + p_i, \text{mgn}(c, j-1)\} & \text{c. c.} \end{cases}$$

Para esto, agregamos una estructura de memoización M de tamaño $(n+1) \times n$, inicializada en un valor reservado para $-\infty$, y guardamos el resultado de $\text{mgn}(c, j)$ en $M(c, j)$. Suponiendo la instancia donde $P = [3, 2, 5, 6]$, donde acabamos de resolver $\text{mgn}(0, 4)$, nos interesa saber qué valores hay en determinadas posiciones de M dependiendo de si la implementación utilizada es *top-down* o *bottom-up*.
Para la posición $M[3, 2]$, en el algritmo top-down, el valor es:

Top-Down ☐ $-\infty$ ☐ 0 ☐ -5 ☐ 2

Para la posición $M[1, 2]$, en el algritmo bottom-up, el valor es:

Bottom-Up ☐ $-\infty$ ☐ 0 ☐ -5 ☐ 2

Pregunta 3 El algoritmo de selección del elemento de i-ésimo orden basado la mediana de medianas (nombrado SELECT en el libro de Cormen et. al.) requiere que el arreglo a explorar tenga todos sus elementos distintos para garantizar que la selección del pivot particiona el arrelgo de manera balanceada. De tener que aplicarlo en un arreglo que tiene repetidos, se puede aplicar la estrategia de agregar un segundo valor a cada elemento que corresponda con su índice. La implementación que corre el mismo algoritmo con esta modificación tiene como complejidad más ajustada

- ☐ $O(n^2)$
- ☐ $O(n \log^2 n)$
- ☐ $O(n)$
- ☐ $O(n \log n)$

Pregunta 4 El tiempo de ejecución de un determinado algoritmo responde a la siguente recurrencia

$$T(n) = T(n/2) + n$$

Para determinar la complejidad de este algoritmo, se nos presenta la siguiente demostración por sustitución de que la recurrencia es $O(n \log n)$.

$$T(n) = T(n/2) + n = (T(n/4) + n) + n = ((T(n/8) + n) + n) + n = \ldots = T(n/2^k) + kn$$

Luego, cuando $k = \log n$, obtenemos que $T(n) = T(n/2^{\log n}) + n \log n = O(n \log n)$. ¿Es correcta esta solución?

- ☐ La complejidad no es ajustada, la recurrencia es $O(\log n)$.
- ☐ La complejidad no es ajustada, la recurrencia es $O(n)$.
- ☐ La demostración es correcta.
- ☐ La complejidad es ajustada, pero la demostración es incorrecta.

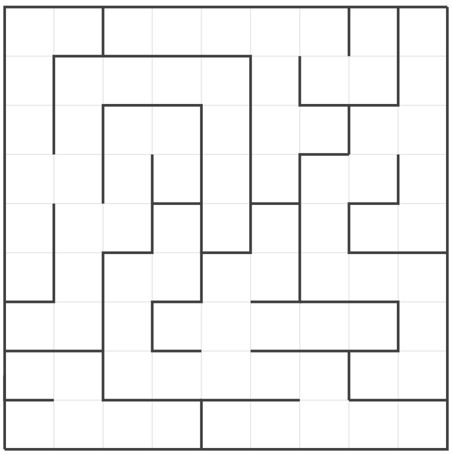
Pregunta 5 Seleccione las afirmaciones correctas respecto de la ténica algorítmica greedy (golosa, codiciosa).

- ☐ Se basa en explotar la propiedad de superposición de subproblemas.
- ☐ Para algunos problemas, un algoritmo de backtracking puede encontrar una solución mejor que la provista por la estrategia greedy.
- ☐ Si no podemos demostrar que la solución que proporciona una estrategia greedy es al menos tan buena como cualquier otra solución, entonces la estrategia no deriva en un algoritmo correcto.
- ☐ Si la formulación de un problema admite una estrategia greedy, esta obtiene una solución óptima.
- ☐ La técnica construye una solución probando en un principio con la opción localmente óptima y en caso de no hallar el óptimo global prueba con la siguiente mejor opción.

Pregunta 6 Se propone demostrar la siguiente propiedad sobre grafos: "Para todo grafo G con n vértices, G es conexo". La demostración es por inducción.
Caso base: Partimos del caso $n = 1$. Claramente, el grafo trivial es conexo.
Paso Inductivo: Sea G un grafo con $n+1$ vértices ($n+1 \geq 2$). Sean v, w vértices de G. Si consideramos $G^v = G \setminus \{v\}$, vemos que es un grafo con n vértices. Luego, por hipótesis inductiva, G^v es conexo, y por lo tanto para todo vértice x hay un camino entre w y x, y en consecuencia, hay un camino entre w y x en G. Análogamente, $G^w = G \setminus \{w\}$ es conexo y vemos que hay un camino entre v y x en G. Entonces, hay un camino de v a w en G, en particular pasando por x, mostrando que G es conexo.
¿Es verdadera la afirmación? ¿La demostración de la misma es correcta?

- ☐ La afirmación es verdadera y la demostración es correcta.
- ☐ La afirmación es falsa, falla en el paso inductivo.
- ☐ La afirmación es verdadera pero la demostración falla en el paso inductivo.
- ☐ La afirmación es verdadera pero la demostración falla en el caso base.
- ☐ La afirmación es falsa, la demostración falla en el caso base.

Pregunta 7 Se cuenta con un laberinto dividido por sus paredes en distintas regiones. Dentro de este, desde una celda podemos movernos a la adyacente arriba, abajo, a la izquierda o a la derecha, en tanto no se interponga una pared.



Nos gustaría contar con una función que dadas dos posiciones dentro del laberinto, pueda determinar en $O(1)$ si una de estas posiciones es alcanzable desde la otra. Por ejemplo, en el laberinto presentado, $(5,4)$ es alcanzable desde $(3,4)$, pero $(6,8)$ no es alcanzable desde $(8,8)$. Para obtener esta función, podemos hacer un precómputo con la información del laberinto, y tenemos disponibles tanto el algoritmo de DFS como el de BFS. ¿Podemos usar alguno de ellos para dar con la función buscada?

- ☐ Ni BFS ni DFS pueden ser utilizado para resolver este problema.
- ☐ Un algoritmo basado en BFS puede resolver este problema.
- ☐ Un algoritmo basado en DFS puede resolver este problema.