
Nombre:

DNI / NIE:

Análisis y Diseño de Algoritmos

Examen final C3

Instrucciones:

- **Antes de comenzar el examen** poned vuestros datos en el cuadernillo de preguntas y también en la hoja de respuestas.
- **No olvidéis** poner la modalidad en la hoja de respuestas.
- **Dejad** encima de la mesa vuestro DNI o vuestra TIU.
- No podéis consultar ningún material ni hablar con nadie.
- Cada pregunta solo tiene una opción correcta, marcadla como se indica en la hoja de respuestas.
- Cada respuesta incorrecta resta la mitad de una correcta. Las preguntas sin contestar ni suman ni restan puntos.
- Tenéis 90 min. para hacer el examen.

Normativa: (*Reglamento para la evaluación de los aprendizajes*, 27-11-2015)

- Está prohibido acceder al aula del examen con cualquier tipo de dispositivo electrónico.
- Además de dos bolígrafos o lápices, del documento de identificación personal y del material suministrado por el profesorado, **no** se permite tener ningún objeto o documento, ni en la mesa ni en sus inmediaciones.
- Si tienes dudas acerca de un objeto o dispositivo concreto pregunta al profesor antes de que comience la prueba.
- El incumplimiento de esta normativa puede conllevar, entre otras, la expulsión del aula del examen sin posibilidad de realizar la prueba.

Algunas preguntas hacen referencia al problema con el que hemos trabajado en las sesiones de prácticas. Su enunciado común es el siguiente:

Problema del laberinto: *Se dispone de una cuadrícula $n \times m$ de valores $\{0, 1\}$ que representa un laberinto. Un valor 0 indica que la posición es inaccesible; por el contrario, con el valor 1 se simbolizan las casillas accesibles. Se pretende conocer la longitud del camino de salida más corto o un camino de salida con esa longitud (o ambas cosas).*
Se define *longitud del camino* como el número de casillas que lo componen. Se define *camino de salida* como un camino que partiendo del origen del laberinto $(0, 0)$, conduce a la salida $(n-1, m-1)$.
De este problema hemos distinguido dos versiones según los movimientos permitidos:

- **versión restringida:** Asumimos solo tres movimientos posibles: derecha; diagonal y abajo, siempre que la casilla de destino sea una posición válida.
- **versión general:** Desde una celda se puede acceder a cualquier de sus ocho colindantes siempre que la casilla destino que sea una posición válida.

Preguntas:

1. Se pretende resolver la versión general del problema del laberinto mediante ramificación y poda. Para obtener la cota optimista de un nodo cualquiera procedemos así: calculamos el número de casillas que quedan, por la diagonal derecha–abajo, hasta llegar a una pared del laberinto. A ese valor le sumamos el número de casillas que podrían quedar desde esa pared hasta la salida utilizando únicamente movimientos del tipo abajo o derecha, según corresponda. ¿Que podemos decir acerca de la cota optimista que se obtendría?
   (a) Que no cumple la condición para ser una cota optimista correcta.
   (b) Que no sería eficiente con respecto a otras cotas optimistas, ya que su cálculo requiere una complejidad temporal que no es constante.
   (c) Que es una cota optimista correcta y además puede obtenerse con complejidad temporal constante.
2. ¿Cuál de los siguientes enfoques es más eficiente para resolver la versión restringida del problema del laberinto?
   (a) Enfoque voraz.
   (b) Enfoque de programación dinámica.
   (c) Enfoque de vuelta atrás.
3. Con respecto a la técnica *poda con memoria*, solo una de las siguientes afirmaciones es cierta. ¿Cuál es?
   (a) No resulta eficaz para resolver la versión general del problema del laberinto mediante ramificación y poda.
   (b) No se puede aplicar para resolver la versión restringida del problema del laberinto.
   (c) En una solución de vuelta atrás para la versión general del problema del laberinto, sirve también para descartar ciclos.
4. Queremos aplicar la técnica de memoización a la siguiente función recursiva:

```
double f( double x ) {
  if( x <= 2 )
    return x;
  return f(sqrt(x-1)) + f(sqrt(x-2));
}
```

   ¿Cuál sería un buen candidato para el almacén?
   (La función `sqrt()` obtiene la raíz cuadrada; `xMax` es el valor de `x` en la primera llamada.)
   (a) `vector< vector<double> > M(xMax+1,vector<double>(xMax+1))`
   (b) `vector<double> M(xMax+1)`
   (c) Ninguna de las otras dos opciones es válida.
5. ¿Cuál es la complejidad temporal en función de `n`, del siguiente fragmento:

```
for( int i = 0; i < n; i++ ) {
  A[i] = 0;
  for( int j = 0; j < 20; j++ )
    A[i] += B[j];
}
```

   (a) $\Theta(n \log n)$
   (b) $\Theta(n^2)$
   (c) $\Theta(n)$

6. Sea el vector $v = \{1, 3, 2, 7, 4, 6, 8\}$ cuyos elementos están dispuestos formando un montículo de mínimos. Posteriormente añadimos en la última posición del vector un elemento nuevo con valor 5. ¿Qué operación, de entre las que son válidas, hay que hacer para que el vector siga representando un montículo de mínimos?
   (a) Intercambiar el 7 con el 5.
   (b) Intercambiar el 8 con el 5.
   (c) No hay que hacer nada pues el vector $v = \{1, 3, 2, 7, 4, 6, 8, 5\}$ también es un montículo de mínimos.
7. Con los valores numéricos almacenados en un fichero, queremos construir un *heap* (montículo). ¿cuál es la forma más eficiente de proceder?
   (a) Almacenar esos valores en un vector y después, reorganizar sus elementos para que estén dispuestos en forma de *heap*.
   (b) Almacenar esos valores directamente en un *heap* que inicialmente está vacío y va creciendo por cada uno de los valores insertados.
   (c) Ambas formas de proceder son equivalentes en cuanto a eficiencia.
8. ¿Cuál es la complejidad, en función de `n`, del siguiente fragmento:
   (suponed que A está definido como `vector<int>A(n)` y `sort()` es la función de ordenación de la librería estándar de C++, que tiene la mejor complejidad, temporal y espacial, posible para un algoritmo de ordenación de propósito general.)

```
std::sort(begin(A), end(A));
int acc = 0;
for( auto i : A )
  acc += i;
```

   (a) $\Theta(n \log n)$
   (b) $\Theta(n^2)$
   (c) $\Theta(n)$
9. ¿Cuál es el coste de monticulizar (*heapify*) un vector de tamaño N?
   (a) $O(N \log N)$ y $\Omega(N)$
   (b) $\Theta(N)$
   (c) $O(N)$ y $\Omega(1)$
10. Una empresa de transportes dispone de M vehículos para repartir N paquetes, todos al mismo destino. Cada paquete i tiene un peso P_i y se tiene que entregar antes de que transcurra un tiempo T_{P_i}. Por otro lado, cada vehículo j puede transportar una carga máxima C_j, tarda un tiempo T_{V_j} para llegar al destino y consume una cantidad L_j de litros de combustible, independientemente de la carga que transporta. Imaginad un algoritmo de vuelta atrás que obtenga la manera en que se tienen que transportar los objetos (en qué vehículo j tiene que ir cada objeto i) para que el consumo sea el mínimo. ¿Cuál sería una buena cota optimista?
    (a) La solución voraz del problema de cargar cada paquete en el camión de menor consumo donde cada paquete llega a tiempo, sin tener en cuenta si el camión se sobrecarga o no.
    (b) La solución voraz del problema de cargar cada paquete en el camión de menor consumo, sin sobrecargarlo, sin tener en cuenta si el paquete llega a tiempo o no.
    (c) Ambas son cotas optimistas válidas.

11. Dada la versión restringida del problema del laberinto, si solo se desea conocer la longitud del camino de salida más corto, ¿cuál es la mejor complejidad temporal y espacial que se puede conseguir si se aplica programación dinámica?

    (a) Temporal $\Theta(nm)$ y espacial $\Theta(\text{mín}\{n, m\})$

    (b) Ninguna de las otras dos opciones es cierta.

    (c) Temporal $\Theta(nm)$ y espacial $\Theta(nm)$

12. Se pretende resolver la versión general del problema del laberinto mediante un algoritmo de búsqueda y enumeración. Para reducir el número de nodos explorados, ¿qué mecanismo de los relacionados garantiza encontrar la solución y resulta a priori más eficaz?

    (a) Usar una matriz de booleanos para descartar caminos que llegan a una casilla ya visitada. Cada casilla (i, j) de la matriz se inicializa con el valor *false*.

    (b) Usar una matriz de enteros donde se almacena la longitud del mejor camino encontrado hasta el momento que llega a cada casilla del laberinto. Cada casilla (i, j) de la matriz se inicializa con el valor infinito.

    (c) Usar una matriz de enteros donde se almacena la longitud del mejor camino encontrado hasta el momento que llega a cada casilla del laberinto. Cada casilla (i, j) de la matriz se inicializa con la solución de programación dinámica para la versión restringida del problema asumiendo que el destino es (i, j).

13. Dada la versión general del problema del laberinto ¿Qué ocurre si la cota optimista de un nodo resulta ser el valor que se obtiene de una solución factible pero que no es la mejor en el subárbol generado por ese nodo?

    (a) Nada especial, las cotas optimistas se corresponden con soluciones factibles que no tienen por qué ser las mejores.

    (b) Que el algoritmo sería incorrecto pues podría descartarse el nodo que conduce a la solución óptima.

    (c) Que el algoritmo sería más lento pues se explorarían más nodos de los necesarios.

14. ¿Qué inconveniente presenta la siguiente función?

```
void maze(const Maze &maze, vector<vector<bool>> &visited, Point &currentPoint,
          size_t currentLength, size_t &currentBestLength) {

  if (currentPoint == maze.exit()) {
    currentBestLength = min(currentBestLength, currentLength);
    return;
  }

  for (auto step : allSteps) {
    Point next = currentPoint.next(step);
    if (maze.is_valid(next) && !visited[next.x()][next.y()] &&
        currentLength < currentBestLength) {
      visited[next.x()][next.y()] = true;
      maze(maze, visited, next, currentLength + 1, currentBestLength);
      visited[next.x()][next.y()] = false;
    }
  }
  return;
}
```

    (a) Que no detecta todos los subproblemas repetidos que pueden aparecer.

    (b) Que la poda basada en la mejor solución hasta el momento se puede mejorar fácilmente.

    (c) Las otras dos opciones son ambas ciertas.

15. Dada la versión restringida del problema del laberinto, ¿cuál de las estrategias siguientes proveería de una cota optimista para ramificación y poda?

    (a) Suponer que en adelante todas las casillas del laberinto son accesibles.

    (b) Las otras dos estrategias son ambas válidas.

    (c) Suponer que ya no se van a realizar más movimientos.

16. Indica cuál es la complejidad temporal en función de `n`, donde `A` es un vector de enteros y `k` es una constante que no depende de `n`, del fragmento siguiente:

```
for( int i = k; i < n - k; i++){
  A[i] = 0;
  for( int j = i - k; j < i + k; j++ )
    A[i] += B[j];
}
```

    (a) $\Theta(k)$

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

    (c) $\Theta(n)$

17. Se pretende resolver la versión general del problema del laberinto mediante ramificación y poda y para ello se usa una estrategia que consiste en priorizar las expansiones de los nodos que contienen un camino explorado más corto. ¿Qué podemos decir del algoritmo resultante?

    (a) Que la primera hoja a la que se llegue es la solución del problema y por lo tanto, ya no será necesario explorar más nodos de la lista de nodos vivos, aunque no esté vacía.

    (b) Que el recorrido en el árbol de búsqueda será equivalente a un recorrido por niveles, por lo que no es necesario utilizar una cola de prioridad.

    (c) Las otras dos opciones son ambas ciertas.

18. Dada la versión general del problema del laberinto, se pretende resolver mediante vuelta atrás haciendo uso de un mecanismo de poda basado en la mejor solución encontrada hasta el momento. ¿Cuándo se podaría un nodo?

    (a) Cuando el valor de su cota optimista supere al valor de la mejor cota pesimista encontrada hasta el momento.

    (b) Cuando el valor de su cota pesimista supere al valor de su cota optimista.

    (c) Cuando el valor de su cota optimista supere al valor de su cota pesimista.

19. Dada la versión general del problema del laberinto, tratamos de completar un nodo cualquiera del árbol de búsqueda con el camino que, en caso de existir, obtendríamos asumiendo solo los tres movimientos de la versión restringida. ¿Qué obtendríamos en el caso de completar el nodo?

    (a) Una cota optimista para ese nodo.

    (b) Un nodo hoja del subárbol generado por aquel nodo.

    (c) Nada de interés, puesto que el camino resultante no contemplaría todos los movimientos permitidos.

20. ¿Cómo se utiliza la cola de prioridad en el algoritmo de ramificación y poda para resolver el problema del laberinto?

    (a) Se agregan los nodos a la cola de prioridad en orden aleatorio para explorar todas las posibilidades.

    (b) Los nodos se agregan a la cola de prioridad según una estimación heurística para priorizar los caminos más prometedores.

    (c) Los nodos se extraen de la cola de prioridad según se vayan obteniendo.

21. Se dispone de un conjunto de n valores numéricos dispuestos en forma de montículo y se desea obtener el valor de la suma de todos los que al menos tienen un hijo (es decir, no son nodos hoja). ¿Cuál es la complejidad temporal del mejor algoritmo que se puede escribir?

    (a) $O(n)$

    (b) $O(\log n)$

    (c) $O(n \log n)$

22. Se dispone de un conjunto de n valores numéricos dispuestos en un vector sin orden preestablecido. Se desea escribir una función que reciba ese vector y un valor k $(n/2 \leqslant k \leqslant n)$ y que devuelva los k valores más pequeños dispuestos en otro vector de manera ordenada. ¿Cuál es la complejidad temporal del mejor algoritmo que se puede escribir?

    (a) $O(kn)$

    (b) Ninguna de las otras dos opciones es cierta.

    (c) $O(k \log n)$

23. Para resolver la versión general del problema de la mochila con n objetos y carga máxima W, hemos escrito un algoritmo de divide y vencerás que, sucesivamente, divide el problema en dos subproblemas; cada uno de ellos toma la mitad de los objetos y la mitad de la carga máxima de la mochila. El caso base ocurre cuando solo hay un objeto que se añade a la solución si cabe en la fracción de carga máxima que corresponde a ese subproblema, y si no cabe se descarta. Asumiendo que n y W son potencias exactas de 2, ¿qué podemos decir de esta solución?

    (a) Que con los resultados de los subproblemas no siempre se puede componer la solución del problema original.

    (b) Que no cumple el teorema de reducción.

    (c) Que, aunque con los resultados de los subproblemas se puede componer la solución del problema original, esta formulación no mejora la solución estudiada en clase.

24. Dada la solución *naive* de la versión restringida del problema del laberinto, en general, ¿cuántas veces se llega al caso base de la recursión?

    (a) Solo una.

    (b) Un valor que puede crecer exponencialmente con el tamaño del laberinto.

    (c) Un valor que a lo sumo es $n \cdot m$.

25. Se pretende resolver el problema del viajante de comercio (*travelling salesman problem*) mediante el esquema de vuelta atrás. ¿Cuál de los siguientes valores se espera que se comporte mejor como cota optimista para un nodo?

    (a) La suma de los pesos de las k aristas restantes más cortas, donde k es el número de ciudades que quedan por visitar.

    (b) El valor que se obtiene de multiplicar k por el peso de la arista más corta de entre las restantes, donde k es el número de ciudades que quedan por visitar.

    (c) La suma de los pesos de las aristas que completan la solución paso a paso visitando el vértice más cercano al último visitado.

26. La distancia de Manhattan (d) entre dos puntos (x_1, y_1) y (x_2, y_2) viene dada por la expresión $d = |x_1 - x_2| + |y_1 - y_2|$. ¿Podría utilizarse como cota optimista para resolver la versión general de problema del laberinto?

    (a) No, puesto que incumple la condición para ser una cota optimista correcta.

    (b) Sí, pero hay otros estimadores que se suelen comportar mejor ante este problema.

    (c) Sí, y además tiene la ventaja de que se puede calcular con complejidad temporal constante.

27. Estamos resolviendo la versión restringida del problema del laberinto. Ya hemos obtenido el almacén de resultados parciales. ¿Con qué complejidad temporal podemos obtener un camino de salida de longitud mínima si es que existe?

    (a) $\Theta(m+n)$

    (b) $\Omega(\text{mín}(n,m))$ y $O(mn)$

    (c) $\Theta(mn)$

28. Dados dos nodos cualesquiera del árbol de búsqueda de ramificación y poda, en general, ¿se puede saber con certeza cuál está más cerca de la solución óptima del problema a resolver?

    (a) Sí, el que tiene mejor cota optimista.

    (b) Sí, el que tiene mejor cota pesimista.

    (c) Sí, pero solo si ambos nodos son hoja.

29. De las siguientes expresiones, o bien dos son ciertas y una es falsa, o bien al contrario, una es cierta y dos son falsas. Marca la que en este sentido es diferente a las otras dos.

    (a) $\sum_{i=1}^{\frac{n}{2}} \sum_{j=1}^{i} 2^j \in O(n \log n)$

    (b) $\sum_{i=1}^{n} \sum_{j=1}^{\log i} 2^j \in O(n^2)$

    (c) $\sum_{i=1}^{\log n} \sum_{j=1}^{n} 2^j \in O(n \log n)$

30. Dada la versión restringida del problema del laberinto, ¿se podría modificar la solución *naive* de divide y vencerás para que obtenga el número total de caminos diferentes, de cualquier longitud, que conducen a la salida desde el origen?

    (a) No se puede, ya que no se trataría de un problema de optimización.

    (b) No se puede, puesto que se trata de un nuevo problema que no cumple las condiciones de aplicación de divide y vencerás.

    (c) Sí, pero puesto que se trata de un nuevo problema, hay que cambiar la recurrencia matemática inicial.

31. Con respecto a los algoritmos estudiados durante el curso que encuentran el árbol de recubrimiento de mínimo coste, de las afirmaciones siguientes, o bien dos son verdaderas y una es falsa, o bien dos son falsas y una es verdadera. Marca la que (en este sentido) es diferente de las otras dos.

    (a) El algoritmo de Kruskal va construyendo un bosque de árboles que va uniendo hasta que acaba con un árbol de recubrimiento de coste mínimo.

    (b) El algoritmo de Prim se puede acelerar notablemente si los vértices se organizan en una estructura *union-find*.

    (c) La complejidad temporal del algoritmo de Prim es cúbica con respecto al número de vértices del grafo.

32. ¿Qué hace la siguiente función?

```
void f( vector<int>&A ) {
 priority_queue<int>pq;
 for( auto i : A )
  pq.push(A[i]);
 A.clear();
 while( !pq.empty() ) {
  A.push_back(pq.top());
  pq.pop();
 }
}
```

(a) Invierte el vector `A` (el último elemento quedará el primero).

(b) Nada, deja el vector `A` como estaba.

(c) Ordena el vector `A`.

33. ¿De qué clase de complejidad es la solución de la siguiente relación de recurrencia?
$$f(n) = n(n-1) + f(n-1) \quad \text{si } n > 0$$
$$f(0) = 1 \quad \text{si } n = 0$$

(a) $f(n) \in \Theta(n^2)$

(b) $f(n) \in \Theta(n^3)$

(c) Ninguna de las otras dos opciones es cierta.

34. Un fontanero tiene una jornada de Q cuartos de hora (es así como se organiza la agenda) y tiene C clientes. El trabajo del cliente i tarda q_i cuartos de hora y el fontanero le cobra un precio p_i. Es posible que no pueda atender todos los clientes en la jornada, que nunca puede alargar. Este problema tiene una solución bien conocida que permite elegir qué clientes visitar para que la suma cobrada al final de la jornada sea la máxima. ¿Qué podemos decir de esta solución?

(a) Que se ha de implementar forzosamente con un algoritmo de búsqueda y enumeración como el de vuelta atrás.

(b) Que la organización de la agenda en cuartos de hora permite obtener una solución de complejidad temporal $\Theta(QC)$ y complejidad espacial $\Theta(Q)$.

(c) Que no se puede implementar con una solución de "divide y vencerás" con memoización.

35. ¿Cuál es la característica principal de un algoritmo voraz aplicado al problema del laberinto?

(a) Explora exhaustivamente todas las posibles soluciones del laberinto.

(b) Toma decisiones locales óptimas en cada paso sin considerar el panorama general.

(c) Utiliza una estrategia basada en estimaciones heurísticas para encontrar la solución óptima.

36. Dada la versión restringida del problema del laberinto, estamos interesados en obtener, para cada casilla accesible del laberinto, el camino más corto entre el origen y esa casilla. ¿Qué esquema es el más apropiado en este caso?

(a) La versión iterativa de programación dinámica.

(b) Memoización.

(c) Ambas técnicas resultan equivalentes.

37. El problema de la moneda consiste a formar una suma M con el número mínimo de monedas tomadas (con repetición) de un conjunto C donde hay una cantidad suficientemente grande de monedas con cada posible valor facial $C = \{c_1, c_2, \ldots, c_{|C|}\}$, con $c_1 = 1$. ¿Cuál de estas afirmaciones sobre un algoritmo recursivo de la forma

$$n_{\text{opt}}(M) = 1 + \min_{1 \leqslant i \leqslant |C|} n_{\text{opt}}(M - c_i); \quad n_{\text{opt}}(0) = 0; \quad n_{\text{opt}}(x) = \infty \text{ para } x < 0$$

es falsa?

(a) Dependiendo de cuáles sean los valores faciales y la suma, puede ser que el algoritmo recursivo no encuentre solución.

(b) Tiene un coste temporal prohibitivo, ya que puede calcular $n_{\text{opt}}(x)$ para el mismo valor de x más de una vez.

(c) Encuentra siempre la solución óptima.

38. ¿Qué obtenemos con la siguiente declaración de C++: `priority_queue<nodo> pq;` ?

(a) Un *heap* o montículo de máximos.

(b) Un *heap* o montículo de mínimos.

(c) Un *heap* o montículo sin orden establecido ya que no se ha definido la función de comparación.

39. En un algoritmo de búsqueda y enumeración, ¿qué podemos decir acerca de la heurística que se utiliza para determinar si un nodo debe expandirse o no?

(a) Que puede equivocarse, por eso se le llama heurística.

(b) Que también puede usarse como estrategia de búsqueda.

(c) Las otras dos opciones son ambas ciertas.

40. En cuanto a la posibilidad de aplicar la técnica de programación dinámica iterativa para resolver un problema:

(a) Se debe conocer de antemano todos los posibles subproblemas y además, se debe disponer de una ordenación entre todos ellos según tamaño.

(b) No necesariamente ha de conocerse de antemano todos los posibles subproblemas pero sí debe saberse, dados dos de ellos cualesquiera, cuál es más pequeño.

(c) Se debe conocer de antemano todos los posibles subproblemas pero no necesariamente se debe disponer de una ordenación entre todos ellos según tamaño.