TEORIA BLOQUE 2 DE ALGORITMICA

Jaime Lorenzo Sánchez

18 de diciembre de 2021

Capítulo 1

Cuestiones de recursividad

1. Explica en qué consiste un algoritmo recursivo

Un algoritmo recursivo consiste en descomponer un problema en subproblemas similares sobre datos más simples que los datos del problema original.

2. Indica cuál es la principal desventaja de los algoritmos recursivos. Justifica tu respuesta

La posible repetición de llamadas recursivas, pues puede provocar el desbordamiento de la memoria

3. ¿Cuál es el principal inconveniente de la versión recursiva del algoritmo de la sucesión de Fibonacci? ¿Cómo se podría modificar dicha versión recursiva, sin eliminar la recursividad, para eliminar ese inconveniente?

El principal inconveniente de la versión recursiva del algoritmo de la sucesión de Fibonacci es que, a medida que el número de llamadas recursivas crece exponencialmente, el número de repeticiones innecesarias se puede desbordar y dar lygar a tiempos de cómputo elevadísimos.

Para eliminar este inconveniente, se puede realizar antes de ejecutar una llamada una comprobación de si la llamada ha sido ejecutado.

4. ¿Por qué la versión recursiva del algoritmo de Fibonacci consume más memoria que su versión iterativa?

Porque a medida que disminuye el valor de n, el número de llamadas recursivas crece exponencialmente.

5. ¿Por qué la versión recursiva del cálculo del factorial de un número tarda más tiempo que la iterativa, pese a tener la misma complejidad computacional?

Porque el número de repeticiones innecesarias del algoritmo se puede desbordar, dando lugar a tiempos de cómputo elevadísimos.

6. ¿Qué inconvenientes podría tener el uso de una tabla o de una estructura de datos adicional para evitar la repetición de llamadas en un algoritmo recursivo?

Se produciría un gran consumo de memoria debido a que estaría usando la estructura de datos adicional o tabla al realizar cada llamada al algoritmo.

7. Indica cuál es la principal ventaja de la recursividad. Cita al menos un par de algoritmos donde esa ventaja es patente

Soluciona problemas basados en la repetición anidada de ciertos procedimientos con diferentes parámetros.

Algoritmos: Torres de Hanoi y factorial recursivo.

8. ¿En qué casos valdría la pena traducir un algoritmo recursivo a uno iterativo?

En casos de que la versión iterativa sea mucho más eficiente, y además dicha versión no sea muy complicada de implementar.

9. Si implementamos el cálculo del factorial de un número mediante un algoritmo iterativo y mediante un algoritmo recursivo. ¿Cuál sería el orden de complejidad de cada implementación? ¿Cuál de los 2 consumiría más tiempo? Justifica esta última respuesta

La implementación del cálculo del factorial de un número mediante un algoritmo iterativo tiene un orden de complejidad de O(n).

La implementación del algoritmo recursivo es de orden de complejidad de O(n).

Consumiría más tiempo el algoritmo recursivo, porque a medida que disminuye el valor de n, el número de llamadas recursivas crece exponencialmente.

10. Si implementamos el cálculo del término n de la sucesión de Fibonacci mediante un algoritmo iterativo y mediante un algoritmo recursivo. ¿Cual sería el orden de complejidad de cada implementación?. ¿Cual de los dos consumiría más tiempo?. Justifica esta última respuesta.

El cálculo del término n de la sucesión de Fibonacci mediante un algoritmo recursivo tiene un orden de complejidad exponencial; mientras que un algoritmo iterativo tiene un orden de complejidad lineal.

Consumiría más tiempo el algoritmo recursivo, porque a medida que disminuye el valor de n, el número de llamadas recursivas crece exponencialmente.

11. Si implementamos el cálculo del término n de la sucesión de Fibonacci mediante un algoritmo iterativo y mediante un algoritmo recursivo que evitase la repetición de las llamadas recursivas. ¿Cual sería el orden de complejidad de cada implementación?. ¿Cual de los dos consumiría más tiempo?. Justifica esta última respuesta.

El orden de complejidad del algoritmo iterativo es de O(n). El orden de complejidad del algoritmo recursivo es de O(n)

12. ¿De qué orden de complejidad es el algoritmo recursivo del juego de la rayuela?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?

El orden de complejidad del juego de la rayuela es $O(2^n)$

Esta complejidad se podría reducir utilizando una estructura de datos que almacenara las llamadas recursivas del algoritmo, de modo que al realizar una llamada recursiva solo se realizaría dicha llamada si no esta almacenada en la estructura de datos.

13. ¿Qué ocurriría si en el juego de la rayuela se permitiese retroceder un adoquín?

Aumentaría el orden de complejidad del algoritmo

14. ¿Cómo sería los casos particulares y el caso general en el juego de la rayuela si se admitiese un movimiento para pasar del adoquín n al n+3?

Caso particular: nCaminos(1) = 1, ncaminos(2) = 2, nCaminos(3) = 4

Caso general: nCaminos(n) = nCaminos(n-1)+nCaminos(n-2)+nCaminos(n-3)

15. ¿De qué orden de complejidad es el algoritmo recursivo para obtener el término general de la sucesión de Fibonacci?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?

Es de orden exponencial. Esta complejidad se podría reducir, utilizando una estructura de datos que almacenara las llamadas recursivas, de modo que solo se realizarian las llamadas recursivas si no estan almacenadas en la estructura de datos (si una llamada recursiva no esta almacenada en la estructura de datos, se almacena dicha llamada en la estructura de datos).

16. Indica la fórmula recursiva (caso particular y general) para el problema del plano de la ciudad si se permitieran movimientos de un solo paso a la derecha, a la izquierda y en diagonal noreste.

Caso particular: nCaminos(x,0) =1, nCaminos (0,y) = 1

Caso general: nCaminos(x,y) =nCaminos(x-1,y)+nCaminos(x+1,y)+nCaminos(x-1,y-1)

17. ¿De qué orden de complejidad es el algoritmo recursivo del plano de la ciudad?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?

El orden de complejidad del plano de la ciudad es de orden $O(2^n)$

Se podría reducir utilizando una tabla de datos auxiliar que almacenara las llamadas ya realizadas.

18. ¿De qué orden de complejidad es el algoritmo de las torres de Hanoi?. Justifica tu respuesta.

En el algoritmo de las Torres de Hanoi, cada llamada de orden n genera 2 llamadas de orden n-1. Por tanto, el orden de complejidad del algoritmo es de $O(2^n)$

19. ¿Se podría utilizar una tabla para evitar llamadas repetidas en el algoritmo de las torres de Hanoi para reducir su tiempo de ejecución?. Justifica tu respuesta

No se podría utilizar una tabla para evitar llamadas repetidas en el algoritmo de las torres de Hanoi debido a que no se puede evitar la repetición de llamadas recursivas en dicho algoritmo.

20. ¿Qué recurso se utiliza para numerar las varillas en el algoritmo de las torres de Hanoi?

Se utiliza el recurso 6 - i - j

Capítulo 2

Cuestiones de divide y vencerás

1. Describe brevemente en qué consiste el método divide y vencerás.

Consiste en descomponer el problema a resolver en subproblemas que son subcasos del problema.

2. ¿Qué tres condiciones se deben dar para que se aplique de forma eficiente el método divide y vencerás?.

1. Se ha de seleccionar de forma eficiente cuando usar el algoritmo básico en vez de seguir descomponiendo el problema.
2. Ha de ser posible la descomposición del problema en subproblemas y recomponer las soluciones a dichos problemas de una manera eficiente.
3. Los subproblemas han de ser aproximadamente del mismo tamaño.

3. ¿Por qué la recursividad se puede aplicar a la mayoría de los problemas basados en el método divide y vencerás?

Porque la recursividad es un método que permite descomponer un problema en subproblemas similares sobre datos más simples que los datos del problema original

4. ¿Porqué el algoritmo de la búsqueda binaria se basa en el método divide y vencerás?

Porque las estrategia del algoritmo de la búsqueda binaria implica la división del entorno del problema inicial en 3 subentornos, que generan 3 problemas del mismo carácter que el de partida

5. ¿Cual es el orden de complejidad de la búsqueda binaria?. Justifica tu respuesta.

Para calcular la complejidad de este algoritmo, inicialmente en número de elementos por analizar es n. Tras la primera división, el número será como mucho n/2 (pues nos hemos quedado con la mitad de elementos); tras la segunda división, el número será como mucho n/4; y así sucesivamente.

Por lo general, tras la división número i, el número de elementos por analizar será como mucho: $\frac{n}{2^i}$

El peor caso se da cuando el elemento a buscar no se encuentra en el vector (es decir, cuando tras dividir los elementos por analizar nos quedemos con un número menor a 1). Por lo tanto, el número máximo de llamadas a realizar es el menor número m tal que: $\frac{n}{2^m} < 1$.

Transformando esta fórmula a un logaritmo en base 2, tenemos que: $n < 2^m$ y que $\log n < m$.

Por tanto, el orden de complejidad de la búsqueda binaria es de orden logarítmico ($O(logn)$

6. En el algoritmo del cálculo del mínimo y máximo elemento de un vector basado en divide y vencerás ¿se mejora la complejidad computacional del algoritmo clásico?. Justifica tu respuesta.

Sí, porque al aplicar el método divide y vencerás los valores máximo y mínimo del entorno inicial se calculan comparando los valores máximo de los subentornos, y mínimos de los subentornos respectivamente (en caso de que el entorno inicial contenga un solo elemento, no es necesario dividir el entorno inicial).

De este modo, el número de comparaciones aplicando divide y vencerás es menor que el número de comparaciones del algoritmo clásico.

7. ¿Porqué el algoritmo del quicksort se basa en el método divide y vencerás?

Porque el algoritmo del quicksort genera 2 subproblemas del mismo carácter que el original pero para un ámbito más reducido.

8. ¿Cual es el orden de complejidad del quicksort?. ¿Cual sería el caso peor en el quicksort?. ¿Qué complejidad tendría este caso peor?. Justifica tus respuestas.

El algoritmo quicksort consiste en dividir el vector en 2 subvectores en función de un valor aleatorio pivote (un subvector con los elementos menores que el pivote, en el lado izquierdo a la posición del pivote, y el otro con los elementos mayores que el pivote en el lado derecho a la posición del pivote). Una vez obtenidos los subvectores, tenemos que ordenarlos.

Por tanto, el orden del algoritmo es $O(n * logn)$

El peor caso ocurre cuando el pivote divide al subvector en 2 subvectores, uno con un elemento y el otro con el resto de elementos, pues los tamaños de los subvectores no son del mismo tamaño.

Por tanto, el orden del algoritmo es $O(n^2)$

9. En el algoritmo de los k-menores ¿qué hay que hacer si al aplicar el algoritmo de la partición, el pivote queda situado en la posición k-2? Justifica tu respuesta.

Si la posición del pivote p ocupa una posición menor que k, entonces tendríamos una parte compuesta por p elementos de los k menores que estamos buscando. Por tanto, volveríamos a aplicar el algoritmo de partición a los n-p elementos que quedan a la derecha del pivote obtenido.

10. En el algoritmo de los k-menores ¿qué hay que hacer si al aplicar el algoritmo de la partición, el pivote queda situado en la posición k+2? Justifica tu respuesta.

Si la posición del pivote p ocupa una posición mayor a la posición k, entonces dentro del subvector que componen los p menores están los k menores. Por tanto, volveríamos a aplicar el algoritmo de partición en el subvector de p elementos.

11. ¿Cual es el orden de complejidad del algoritmo de los k-menores?. Justifica tu respuesta.

En el algoritmo de los k-menores, se obtiene la posición de un pivote k, de forma aleatoria, y se ordena los elementos de 2 subvectores (uno con los elementos de valor menor al pivote, y el otro con los elementos de valor mayor al pivote).

Por tanto, el orden de complejidad del algoritmo es $O(logn)$

12. ¿Qué inconveniente tiene el algoritmo de los k-menores?

Los elementos del vector no están ordenados.

13. Sea K-menores(vector v, int k) una función para calcular los k-menores. Usando llamadas a dicha función, ¿cómo sería la función que calcula los k-mayores? ¿y para calcular los k centrales?

Para calcular los k-mayores, la función sería: $K - menores(v, k)$

Para calcular los k centrales, la función sería: $K - menores(v, k)$

14. Explica como se descomponen los números a multiplicar, y como se multiplican, en el algoritmo de la multiplicación de enteros grandes para poder usar un algoritmo recursivo.

Los números a multiplicar se descomponen de la siguiente manera:

$s = \frac{n}{2}$, $w = \frac{u}{10^s}$, $x = umod10^s$, $y = \frac{v}{10^s}$, $z = vmod10^s$

Dichos números se multiplican de la siguiente manera:

si $pequeno(n) = cierto : u * v$

si $pequeno(n) = falso : Multiplicar(w, y) * 10^{2^s}) + (Multiplicar(w, z) + Multiplicar(x, y)) * 10^s + Multiplicar(x, z)$

15. ¿Cual es el orden de complejidad del algoritmo de la multiplicación de enteros grandes?

Orden cuadrático

16. Explica en qué se basa la mejora del algoritmo de la multiplicación de enteros grandes

Se basa en reducir los 4 productos del algoritmo original en 3, a base de realizar más sumas.

17. Explica brevemente en qué consiste el algoritmo de ordenación por fusión.

Consiste en descomponer el vector en 2 subvectores de tamaños similares, ordenar dichos subvectores mediante llamadas recursivas y fusionar las soluciones de cada parte de forma que se mantenga el orden.

18. ¿Por qué el algoritmo de ordenación por fusión se basa en el método divide y vencerás?

Porque cuando el tamaño del vector es lo suficiente grande, divide el vector en 2 subvectores de tamaño similar y los resuelve recursivamente.

19. ¿Cual es el orden de complejidad del algoritmo de ordenación por fusión? ¿Cual sería el peor de los casos y su complejidad?

El orden de complejidad del algoritmo es $O(n * logn)$.

El peor caso sería si los subvectores obtenidos son de tamaños muy distinto. En este caso, el orden de complejidad sería $O(n^2)$

20. Explica el caso general de la fórmula recursiva del algoritmo de la exponenciación basado en divide y vencerás

El caso general de la fórmula recursiva de la exponenciación basado en divide y vencerás se basa en que un valor n tiene exactamente $[\log_2 n] + 1$ dígitos en base 2.

Por tanto, si n es par entonces $a^n = (a^{n/2})^2$ y, si n es impar, tendremos que $a^n = a * a^{n-1}$

21. ¿Cual es el orden de complejidad del algoritmo recursivo de la exponenciación basado en divide y vencerás en el caso de que se aplicaras a números y en el caso de que se aplicara a matrices?. Justifica tu respuesta

Si aplicamos el algoritmo a números enteros y, dado que un valor n tiene exactamente $[\log_2 n] + 1$ dígitos en base 2, sólo necesitamos realizar un máximo de O(log n) de multiplicaciones.

Si aplicamos el algoritmo en matrices, tendremos que realizar n multiplicaciones de la matriz (matriz elevada a la potencia n). Por tanto, debemos de realizar un máximo de O(log n) de multiplicaciones.

Capítulo 3

Algoritmos voraces

1. ¿De dónde procede el nombre de voraz en los algoritmos voraces?

El nombre voraz proviene del hecho de seleccionar el bocado más apetecible en cada etapa, sin preocuparse por lo que pueda ocurrir más adelante.

2. Explica en qué consiste un algoritmo voraz, detallando los distintos pasos que sigue un algoritmo voraz

Una aproximación voraz consiste en que cada elemento a considerar se evalúa una única vez, siendo descartado (no forma parte de la solución ni volverá a ser considerado para la misma) o seleccionado (forma parte de la solución)

Los pasos que sigue un algoritmo voraz son los siguientes:

1. Inicialmente, partimos de una solución vacía.
2. En cada paso se escoge el siguiente elemento para añadir a la solución, entre los candidatos. Una vez tomada esta decisión no se podrá deshacer
3. El algoritmo acabará cuando el conjunto de elementos seleccionados constituya una solución.

3. ¿Cual es el orden de complejidad habitual en los algoritmos voraces?. Justifica tu respuesta. (No basta con decir el orden)

Al calcular el tiempo de complejidad de los algoritmos voraces, se deben tener en cuenta lo siguiente:

1. El conjunto de candidatos en cada etapa es de cardinalidad menor al conjunto de candidatos de partida.

2. La función objetivo y de selección pueden consumir un tiempo constante o lineal.

3. El número de candidatos que formarán parte de la solución será una fracción del conjunto de candidatos de partida.

Como el orden de complejidad máximo viene dado por el orden de complejidad de la función objetivo y de selección, el orden de complejidad de los algoritmos será $O(n^2)$ si el tiempo consumido por la función objetivo y de selección es constante; y a lo sumo $O(n^3)$ si el tiempo consumido por la función objetivo y de selección es de orden lineal.

4. Los algoritmos voraces son O(n^2) o O(n^3). ¿De qué depende de que sean de un orden o de otro?

Los algoritmos voraces son de $O(n^2)$ si la función objetivo y de selección consume un tiempo constante; y de $O(n^3)$ si la función objetivo y de selección consume un tiempo lineal. ´

5. ¿Se obtiene con un algoritmo voraz una solución óptima en todos los casos?. Justifica tu respuesta e indica un caso donde se verifique tu respuesta.

No, porque hay casos en que el algoritmo voraz no obtiene la solución óptima.

Por ejemplo, en el problema de la mochila la solución óptima debe llenar por completo la mochila. Sin embargo, hay casos en los que no se llena por completo la mochila.

6. ¿Porqué el algoritmo tradicional del cambio es un algoritmo voraz?

Porque en cada paso selecciona la mayor de las monedas que se puede escoger, sin preocuparse si esta decisión es correcta a la larga. Además, una vez escogida una moneda ésta

forma parte de la solución final.

7. ¿Qué es un sistema monetario perfecto?

Es un sistema de cambio que proporciona el menor número de monedas.

8. ¿Cómo se obtiene la solución al problema de la mochila?

Se seleccionan las monedas del mayor valor posible, sin pasarnos de la cantidad establecida, hasta obtener dicha cantidad.

9. ¿Qué condición se ha de cumplir para que el algoritmo voraz de la mochila proporcione una solución óptima?

Se debe de llenar por completo la mochila.

10. ¿En qué casos sería la solución al problema de la mochila un subconjunto de los datos de partida?. Justifica tu respuesta

1. La solución será una tupla de reales $(X_1, X_2, ..., X_n)$ que indicará la cantidad de cada material que introducimos en la mochila, de forma que $0 \leq X_i \leq V_i$ y $\sum_{i=1}^{n} x_i = x_i = V$

2. Los candidatos serían pares (i, X_i) con $1 \leq i \leq n$ y $0 \leq x_i \leq v_i$, que indican la cantidad x_i que se utiliza del material m_i.

11. ¿En qué casos sería la solución al problema de la mochila no sería un subconjunto de los datos de partida?. Justifica tu respuesta

Cuando el último material se seleccione parcialmente para completar la mochila, porque dicho material no pertenece al subconjunto solución (dicho subconjunto está formado por los materiales que utilicen todo su volumen).

12. Si en una consulta médica se tuviesen que atender a una serie de personas y se conoce el tiempo de atención que necesita cada una. ¿En qué orden deberían ser atendidos para minimizar el tiempo en el que el médico está en la consulta?

Orden decreciente del tiempo de atención.

13. Si tuvieses que almacenar en una cinta de casette una serie de canciones de distinta duración, ¿En qué orden las almacenarías?. Justifica tu respuesta. Recuerda que para escuchar una canción se han de recorrer todas las anteriores

Al ordenar 2 elementos desordenados de una permutación dada, se disminuye el tiempo total de espera en el sistema. Por tanto, se deben ordenar en orden decreciente de tiempos, para que dicha mejora sea posible.

14. Describe cómo se obtendría la solución óptima en el problema de la planificación de tareas a plazo fijo

Para obtener la solución óptima en el problema de la planificación de tareas a plazo fijo, debemos encontrar un conjunto A de tareas que sean puntuales en la secuencia óptima (secuencia óptima).

Una vez encontrado dicho conjunto de tareas, la secuencia se crea ordenando los elementos de A por plazos crecientes y, a continuación, las tareas tardías en cualquier orden.

15. En el algoritmo voraz de la planificación de tareas a plazo fijo, ¿cual es la condición necesaria y suficiente para que un conjunto de tareas sea factible?

Debe existir al menos una secuencia que permite realizar todas las tareas en sus plazos respectivos.

16. ¿Cómo se demuestra que si un conjunto de tareas es factible, la permutación en orden creciente de plazos de dicho conjunto también lo es?

Demostrando que el conjunto de tareas es factible si y sólo si la permutación es factible.

17. En el algoritmo voraz de la planificación de tareas a plazo fijo ¿cuál es la única tarea que siempre formaría parte de la solución?. Justifica tu respuesta

La primera tarea, porque se utiliza la posición 0 como centinela de los vectores de plazos de cada tarea y del vector solución, y se ordenan dichos vectores en orden decreciente de beneficios.

18. Describe como funciona el algoritmo de Kruskal indicando porqué es un algoritmo voraz.

La solución del algoritmo es un subconjunto del conjunto del problema formado por los lados y los nodos del grafo. Inicialmente, dicha solución está vacía y, a medida que el algoritmo avanza, se incorporan nuevos lados al conjunto solución, de modo que el lado incorporado en cada etapa es el lado de coste mínimo aún no seleccionado, que enlaza dos componentes conexas distintas, con lo cual el número de componentes conexas se reduce en una unidad.

Cuando finaliza el algoritmo, en el conjunto solución hay una sola componente conexa de coste mínimo que enlaza los nodos del grafo, formando el árbol de coste mínimo que enlaza los nodos del grafo.

Por todo esto, podemos concluir que el algoritmo de Kruskal es un algoritmo voraz.

19. ¿Podría obtenerse más de una solución en el algoritmo de Kruskal?. Justifica tu respuesta

No, porque al obtenerse los componentes conexos cada vez más grandes, se van tomando los lados del grafo en orden creciente de pesos siempre y cuando se enlacen 2 componentes conexas distintas.

20. ¿Porqué no podría haber más de n-1 lados en la solución del algoritmo de Kruskal?

Porque en cada etapa se incorpora en la solución el lado de coste mínimo, aún no seleccionado, que enlaza 2 componentes conexas distintas.

21. Explica en qué consiste el algoritmo voraz para resolver el problema del viajante de comercio

En cada iteración seleccionamos el lado más corto aún no seleccionado que cumpla las siguientes 2 condiciones:

1. No forma un ciclo con los lados ya seleccionados, excepto en la última iteración para cerrar el recorrido.

2. No es el tercer lado que incide en el mismo nodo.

22. ¿Es óptima la solución obtenida por el algoritmo voraz del viajante de comercio?

No, porque sería inviable estimar la mínima posibilidad para un número elevado de ciudades.

23. ¿En qué tipo de grafos es posible que el algoritmo voraz del viajante de comercio no proporcione una solución?

En grafos no denso como una red de carreteras, debido a que se puede llegar a un callejón sin salida en el cual no encontremos ningún lado que no forme ciclo o que los lados que no forman ciclos, son los terceros lados incidentes en un nodo.