Práctica: Técnicas algorítmicas

Concurso JTP Simple Algoritmos. Agustín Santiago Gutiérrez

Los ejercicios marcados con **(*)** se consideran "Menos importantes / Opcionales".

Dividir y conquistar

1. Sea $v = (v_1, v_2, \cdots, v_n)$ un vector de n objetos. Un elemento mayoría de v es un valor que aparece en v **más** de $\frac{n}{2}$ veces.

   a) Si $n \geq 2$, dado algún $2 \leq i \leq n$, podemos partir a v en dos arreglos: $a = (v_1, v_2, \cdots, v_{i-1})$ y $b = (v_i, v_{i+1}, \cdots, v_n)$. Probar que si v_m es elemento mayoría de v, entonces v_m es elemento mayoría de a o de b (o de ambas partes).

   b) Utilizar el ítem anterior para escribir un algoritmo de dividir y conquistar que determine si un arreglo dado tiene elemento mayoría, y de ser así, lo calcule. Debe tener complejidad $O(n \lg n)$.

2. **(*)** Hay que organizar un torneo que involucre a n competidores. Cada competidor debe jugar exactamente una vez contra cada uno de sus oponentes. Además cada competidor debe jugar un partido por día con la sola posible excepción de un día en el cual no juegue.

   a) Si n es potencia de 2 dar un algoritmo que use la técnica de dividir y conquistar para armar el fixture para que el torneo termine en $n - 1$ días.

   b) Si $n > 1$ no es potencia de 2 dar un algoritmo para armar el fixture para que el torneo termine en $n - 1$ días, si n es par o en n, si n es impar.

   Demostrar que las cantidades de días indicadas son mínimas.

Algoritmos golosos

3. Se tiene inicialmente una hilera de n casillas blancas consecutivas. Se deben pintar algunas casillas de negro, de manera tal que no haya nunca dos consecutivas pintadas. Algunas casillas ya fueron pintadas de antemano, cumpliendo con la restricción.

   Dar un algoritmo goloso que pinte algunas de las restantes, de modo tal que la cantidad total de casillas pintadas sea máxima. Demostrar que el algoritmo es correcto y calcular su complejidad.

4. Se tiene un listado de m intervalos, de modo tal que para cada $1 \leq i \leq m$, el intervalo i-ésimo es $[A_i, B_i]$, siendo $0 \leq A_i \leq B_i \leq n$, para cierto valor n fijo. Se desea seleccionar algunos de los intervalos, de modo tal que su unión sea todo el intervalo $[0, n]$ (es decir, se desea "cubrir" el $[0, n]$ usando algunos intervalos de la entrada).

   Dar un algoritmo goloso que calcule la mínima cantidad de intervalos necesaria, o indique que esta tarea es imposible. Calcular su complejidad en términos de m y n.

5. **(*)** Se quiere dar el vuelto a un cliente usando el mínimo número de monedas posibles. Se tiene un suministro ilimitado de monedas de 1, 5, 10 y 25 centavos. Dado el siguiente algoritmo goloso:

   ```
   Elegir en cada paso la moneda de mayor valor entre las disponibles (sin pasarse).
   ```

   a) Probar que este algoritmo siempre da el vuelto utilizando la mínima cantidad posible de monedas.

   b) ¿Ocurre lo mismo si se elimina la moneda de 5, de forma tal que las denominaciones posibles sean 1, 10 y 25 centavos? En caso afirmativo, demostrar. En caso negativo, dar un contraejemplo para el algoritmo.

Programación dinámica

6. La sucesión de Fibonacci se define matemáticamente de la siguiente manera:

   $f_0 = f_1 = 1$

   $f_n = f_{n-1} + f_{n-2}$ para $n \geq 2$

   Escribir una función recursiva (implementación "directa") que calcule f_n dado n.

   a) Determinar la complejidad de esta función. ¿Es polinomial? ¿Es exponencial?

   b) Explicar por qué se puede decir que tenemos en este caso *superposición de subproblemas*.

   c) Teniendo en cuenta lo anterior, utilizar programación dinámica para mejorar la complejidad de la función de modo que esta sea $O(n)$.

7. Se tienen n números enteros positivos en fila, de izquierda a derecha. Se deben elegir algunos de ellos, pero con la restricción de que no se permite elegir nunca dos elementos que sean consecutivos.

   a) Utilizar programación dinámica para calcular la máxima suma posible de los elementos elegidos.

   b) ¿Hay alguna relación entre este problema y el ejercicio 3?

   c) Plantear un algoritmo goloso para este problema, y decidir si es correcto o no.

8. Sea $M \in \mathbb{N}^{m \times n}$ una matriz de números naturales. Se desea obtener un camino que empiece en la casilla superior izquierda ($[1, 1]$), termine en la casilla inferior derecha ($[m, n]$), y tal que minimice la suma de los valores de las casillas por las que pasa.

   En cada casilla $[i, j]$ hay dos movimientos posibles: ir hacia abajo (a la casilla $[i + 1, j]$), o ir hacia la derecha (a la casilla $[i, j + 1]$).

   a) Diseñar un algoritmo eficiente basado en programación dinámica que resuelva este problema.

   b) Determinar la complejidad del algoritmo propuesto (temporal y espacial).

   c) Exhibir el comportamiento del algoritmo sobre la matriz que aparece a continuación.

   $$\begin{bmatrix} 2 & 8 & 3 & 4 \\ 5 & 3 & 4 & 5 \\ 1 & 2 & 2 & 1 \\ 3 & 4 & 6 & 5 \end{bmatrix}$$

9. Supongamos que tenemos la situación del ejercicio 4, pero ahora cada intervalo tiene un cierto **costo** c_i asociado, y queremos cubrir el $[0, n]$ con costo mínimo (sin importar la cantidad de intervalos).

   a) Mostrar que el mismo algoritmo goloso ya no es correcto para esta variante del problema.

   b) Dar un algoritmo de programación dinámica $O(m^2)$ para este problema (sugerencia: pensar primero un algoritmo $O(nm)$).

10. Dar un algoritmo de programación dinámica que resuelva el problema de dar el vuelto del ejercicio 5, que funcione correctamente para cualquier conjunto de denominaciones dado.

    El algoritmo recibirá la lista de valores de monedas existentes $v_1, v_2, \cdots, v_n$, y el vuelto T objetivo.

11. Dada una cadena de n caracteres $s_1, s_2, \cdots, s_n$, se desea conocer todas sus subcadenas palindrómicas[1]. Computar mediante programación dinámica una tabla de $n \times n$, que en el lugar $[i, j]$, para $i \leq j$, indique con un booleano si la subcadena $s_i, s_{i+1}, \cdots, s_j$ es palíndromo. La complejidad del algoritmo debe ser $O(n^2)$.

12. **(*)** Dada una cadena de n caracteres, determinar mediante programación dinámica la mínima cantidad de palíndromos en que puede descomponerse. Calcular la complejidad del algoritmo resultante.

[1] Una cadena es un palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, RADAR lo es pero RADA no lo es.

Búsqueda exhaustiva

13. En el problema de *suma de subconjunto*, se recibe un conjunto de n enteros positivos $S = \{a_1, a_2, \cdots, a_n\}$, y se debe determinar **si existe** algún subconjunto $S' \subseteq S$ cuya suma sea un cierto valor objetivo dado T.

    a) Describir un algoritmo de **fuerza bruta** para este problema.

    b) Describir un algoritmo de **backtracking** para este problema.

    c) ¿Cuál es la diferencia entre un algoritmo de fuerza bruta y uno de backtracking?

    d) Asumiendo que los números a_i no son excesivamente grandes, dar un algoritmo de programación dinámica eficiente para este problema.

14. **(*)** En el *problema de la partición*, se tiene un conjunto de n enteros positivos $\{a_1, a_2, \cdots, a_n\}$ y se deben particionar sus elementos en dos conjuntos de igual suma. Por ejemplo, para la entrada $n = 7$ con elementos $\{1, 2, 5, 6, 7, 10, 11\}$, las soluciones son $\{\{1, 2, 5, 6, 7\}, \{10, 11\}\}$ y $\{\{1, 2, 7, 11\}, \{5, 6, 10\}\}$.

    a) Escribir un algoritmo de backtracking que encuentre todas las soluciones posibles al problema de la partición.

    b) ¿Hay alguna relación entre este problema y el anterior?

Algoritmos probabilísticos (*)

15. **(*)** Un algoritmo de backtracking posible para el problema de la partición considera en cada paso, para cierto elemento de la entrada las dos posibilidades: incluirlo en el conjunto "de la derecha" o en el "de la izquierda". En una implementación determística usual, siempre se probará la misma opción primero, por ejemplo, siempre se intentará primero ubicar los números en el conjunto izquierdo, y solo al retroceder en el algoritmo se reintenta ubicarlos a la derecha.

    En la práctica esto puede tardar mucho en encontrar una solución, y si no hace falta buscar todas las soluciones sino solamente encontrar una, se puede utilizar un algoritmo probabilístico de tipo *Las Vegas*: En cada paso del backtracking, **se elige al azar** en cuál de los conjuntos ubicar al número. Si esa rama resulta no producir una solución, luego se intenta la otra opción.

    Implementar este algoritmo y comparar los tiempos de ejecución hasta encontrar alguna solución en instancias "grandes", con los del backtracking determinístico.

16. **(*)** Dadas 3 matrices $A, B, C \in \mathbb{R}^{n \times n}$, se desea verificar si es cierto que $A \times B = C$. Como multiplicar dos matrices es muy costoso[2], se desea realizar la verificación utilizando únicamente productos entre matrices y vectores: Si $M \in \mathbb{R}^{n \times n}$ y $v \in \mathbb{R}^{n \times 1}$, computar $M \times v$ toma solamente $O(n^2)$.

    Proponer un algoritmo probabilístico eficiente de tipo Montecarlo, que determine si es cierto que $A \times B = C$ con baja probabilidad de error.

[2]Existen algoritmos avanzados $O(n^{2,37})$, pero no se conocen algoritmos $O(n^2)$