Algoritmos y Estructuras de Datos 3: Práctica 3

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Octubre 2014

Objetivo de la práctica

En la primer parte de la materia, se explican varias **técnicas algorítmicas generales**:

- Recursión
- Divide and Conquer
- Algoritmos golosos
- Programación dinámica
- Backtracking

Objetivos de la práctica:

- Ejercitar el uso de las técnicas en la formulación de algoritmos.
- Ejercitar el análisis de los mismos.
  - Correctitud
  - Complejidad

Objetivo de la práctica

En la primer parte de la materia, se explican varias **técnicas algorítmicas generales**:

- Recursión
- Divide and Conquer
- Algoritmos golosos
- Programación dinámica
- Backtracking

Objetivos de la práctica:

- Ejercitar el uso de las técnicas en la formulación de algoritmos.
- Ejercitar el análisis de los mismos.
  - Correctitud
  - Complejidad

Objetivo de la práctica

En la primer parte de la materia, se explican varias **técnicas algorítmicas generales**:

- Recursión
- Divide and Conquer
- Algoritmos golosos
- Programación dinámica
- Backtracking

Objetivos de la práctica:

- Ejercitar el uso de las técnicas en la formulación de algoritmos.
- Ejercitar el análisis de los mismos.
  - Correctitud
  - Complejidad

División

No hay secciones marcadas explícitamente, pero los ejercicios se dividen claramente de acuerdo a las técnicas algorítmicas mencionadas:

- Recursión, 1 y 2
- Divide and Conquer, 3 y 4
- Algoritmos golosos, 5 y 6
- Programación dinámica, 7 al 13
- Backtracking, 14 al 16

Finalmente, 17 y 18 son ejercicios generales donde se pide analizar los algoritmos anteriores (de esta y otras prácticas).

Recursión

- Muchos algoritmos útiles son recursivos:
  - Usan soluciones a subproblemas íntimamente relacionados.
  - De la misma naturaleza que el problema principal.
- Varias de las técnicas algorítmicas estudiadas utilizan la recursión.
  - Divide and Conquer
  - Programación Dinámica
  - Backtracking
- Por eso es importante que el alumno entienda la recursión en sí misma antes intentar utilizar técnicas particulares más complejas que hagan uso de la recursión.

Recursión: Ejercicios 1 y 2

- Ejercicio 1: Cálculo de los números de Fibonacci.
  - Recursiva
  - Iterativa
  - Comparar ambas (complejidades)
- Ejercicio 2: Escribir un algoritmo para resolver problema de las torres de Hanoi recursivamente.
  - Ejercita el pensamiento recursivo.
  - Analizar complejidad.
  - Demostrar correctitud.

Divide and Conquer

- Muchos algoritmos **recursivos** responden a un esquema general de divide and conquer
  - El problema original es *dividido en subproblemas independientes*.
  - Estos se resuelven **recursivamente**.
  - Y luego *se combinan las soluciones* para resolver el original.
- Técnica fuertemente basada en la **recursión**.
- Es natural el paso de ejercicios sobre recursión a ejercicios sobre divide and conquer.

Divide and Conquer: Ejercicio 3

- Este ejercicio pide dar un algoritmo recursivo para encontrar el par de puntos más cercano de un conjunto de puntos en el plano.
- Está marcado con asterisco
  - Ejercicio opcional.
  - Geometría computacional.
  - Mayor dificultad que la mayoría de la práctica.
- Solución guiada mediante sugerencias paso a paso.
- Se pide complejidad: Teorema Maestro de AED2

Divide and Conquer: Ejercicio 4

Se pide armar un fixture para un torneo de n jugadores que se enfrentan todos contra todos una vez.

- Parte a) resolver el caso $n = 2^k$ (división en subproblemas independientes facilitada).
- Parte b) extender el algoritmo de a) para cualquier valor de n
- La diferencia entre a) y b) es una temática usual en divide and conquer (Ejemplo, algoritmo de Strassen)

Algoritmos golosos

- En general son fáciles de entender conceptualmente.
  - Mantienen un único estado actual (solución en construcción).
  - En cada paso utilizan algún criterio para elegir de manera golosa un candidato.
  - La solución se modifica de acuerdo a la elección y se repiten estos pasos.
- Por eso conviene colocarlos antes que programación dinámica.
  - Además veremos que algunos ejercicios de programación dinámica referencian ejercicios de secciones anteriores.
- No es tan fácil demostrar su correctitud.

Algoritmos golosos: Ejercicios 5 y 6

- Ejercicio 5: Se pide analizar tres estrategias golosas posibles de ordenamiento de los programas en una cinta para minimizar el tiempo promedio de carga del programa.
- Ejercicio 6: Problema de dar el vuelto con mínima cantidad de monedas (será revisitado con Programación Dinámica).
- En ambos se ejercita:
  - Mostrar con contraejemplos que ciertas estrategias pueden dar resultados muy lejanos al óptimo.
  - Demostrar la optimalidad de una estrategia correcta.

Programación Dinámica

- Muy similar a divide and conquer, pero **memorizando** resultados de subproblemas para poder reutilizarlos.
- Aprovecha **superposición de subproblemas**.
- Permite eliminar algunas complejidades exponenciales en recursiones (fibonacci).
- Resulta generalmente más difícil de entender que las anteriores:
  - Se debe plantear el espacio de estados (subproblemas) posibles.
  - Establecer una relación de recurrencia que permita calcular el resultado de unos estados en función de estados anteriores.
  - Estos estados o subproblemas no siempre se desprenden fácilmente del enunciado del problema en sí.
  - Se debe incorporar información intermedia propia de la solución que se quiere construir.

Programación Dinámica : Ejercicios 7,8 y 9

- Ejercicio 7: Cálculo de coeficientes binomiales (triángulo de Pascal).
  - Ilustra superposición de subproblemas.
  - Se sugiere repensar el ejercicio de Fibonacci usando esta idea.
- Ejercicio 8: Camino mínimo cruzando una matriz de esquina a esquina.
  - También ilustra superposición de subproblemas.
  - Primer ejemplo de uso del **principio del óptimo**.
  - Se ejercita reconstrucción del camino, además del valor.
- Ejercicio 9: Encontrar una secuencia de operaciones para obtener un resultado fijo (si es posible).
  - Mismos objetivos que el ejercicio anterior.
  - Ejemplo más complejo: la elección del espacio de subproblemas es más difícil y menos evidente del enunciado.

Programación Dinámica : Ejercicio 10

- Problema de optimización en una fábrica de radios, con muchas restricciones.

Este ejercicio está marcado con un asterisco (ejercicio opcional).

- Mismos objetivos que los anteriores.
- Pero tiene más complejidad debido a que hay mucha información en juego
- Más parámetros en los subproblemas, y muchos conjuntos de valores en la entrada:
  - Los subproblemas involucran el mes que se está considerando actualmente y la cantidad de radios actualmente en stock.
  - Hay que considerar **todas** las posibles cantidades de radios a fabricar durante el mes actual a la hora de escribir la relacion de recurrencia.
  - Para esto último entran en juego la cantidad de radios pedidas en el mes actual, el costo de producir en el mes actual, el costo de stock y el costo fijo de iniciar producción.

Programación Dinámica : Ejercicios 11, 12 y 13

- Ejercicio 11: Problema de dar el vuelto con la menor cantidad de monedas.
  - Se pide resolver el problema y comparar la solución con el goloso del ejercicio 6.
- Ejercicio 12: Distancia de edición entre dos palabras.
  - Suele explicarse en la teórica (o bien *máxima subsecuencia común*, que es un problema similar).
  - Aquí se pide dar el algoritmo preciso y analizarlo, incluyendo la reconstrucción de la secuencia de operaciones.
- Ejercicio 13: Ejercicio teórico sobre programación dinámica (por eso al final, luego de resolver ejercicios concretos)
  - Se pide dar ejemplos donde **no** valga el principio de optimalidad.
  - Esto sirve para apreciar cómo el principio de optimalidad es necesario para poder aplicar la técnica, al ver ejemplos donde no aplica.
  - Puente para llegar a backtracking, ya que esta técnica será aplicable en esos casos.

Backtracking

- Alternativa a la fuerza bruta.
- En lugar de probar todas las soluciones posibles individualmente, se va construyendo una solución de manera incremental.
- Se detiene el proceso (backtrack) cuando se detecta que esta solución parcial no se va a completar a una solución global.
- Es mucho mejor que la fuerza bruta, y los ejercicios que vienen pretenden mostrar eso.
- Generalmente da lugar a algoritmos exponenciales, y se la usa para tratar problemas computacionalmente difíciles.

Backtracking (cont)

- No es muy difícil de entender conceptualmente.
- Suele ser difícil implementar buenos algoritmos de backtracking.
- Como generalmente se utiliza recursión en la implementación, requiere un buen manejo de recursión.
- Hay que identificar y mantener consistente la información de la solución que se está construyendo, que va mutando permanentemente.

Teniendo en cuenta esto y que se la suele aplicar a problemas computacionalmente difíciles, es razonable poner esta técnica al final.

Backtracking : Ejercicios 14, 15 y 16

- Ejercicio 14: Problema de las 8 reinas en un tablero de ajedrez.
- Ejercicio 15: Problema de la mochila. Este problema es un ejemplo de problema NP completo que se verá más adelante en la materia.
- Ejercicio 16: Problema de la suma de subconjuntos, también NP completo.

En todos los casos:

- Se pide comparar contra el algoritmo de fuerza bruta.
- Al dar problemas computacionalmente difíciles, se va mostrando que la técnica de backtracking es aplicable a estos problemas.
- Se pide analizar las complejidades, que resultarán exponenciales en estos casos.

Ejercicios teóricos generales

Finalmente, terminan la práctica dos ejercicios generales para clasificar y analizar los algoritmos ya realizados en ejercicios anteriores.

Ejercicios teóricos generales : Ejercicio 17

- Se pide decidir cuales de los algoritmos de las prácticas 2 y 3 son:
  - Divide and conquer.
  - Recursivos.
- Además se pregunta si siempre es mejor usar una implementación recursiva o una iterativa.
- En base a las complejidades obtenidas en ejercicios anteriores, el alumno debería poder concluir que no siempre es mejor una versión recursiva, y tampoco es siempre mejor una versión no recursiva (ejemplos, ejercicios 1 y 3 de esta práctica).

Ejercicios teóricos generales : Ejercicio 18

Este ejercicio pide clasificar todos los ejercicios de esta práctica en “buenos” y “malos”.

- En las teóricas se define un buen algoritmo como un algoritmo de complejidad temporal polinomial.
- Por lo tanto, revisando los análisis de complejidad de los distintos algoritmos ya propuestos, el alumno debería poder realizar esta clasificación.
- Esto va sirviendo de introducción para el posterior estudio de las clases de complejidad P y NP.

Ejercicio 5: Enunciado

Sean $P_1, P_2, \cdots, P_n$ programas que se quieren almacenar en una cinta. El programa P_i requiere s_i Kb de memoria. La cinta tiene capacidad para almacenar todos los programas. Se conoce la frecuencia π_i con que se usa el programa P_i. La densidad de la cinta y la velocidad del drive son constantes. Después que un programa se carga desde la cinta la misma se rebobina hasta el principio. Si los programas se almacenan en orden $i_1, i_2, \cdots, i_n$ el tiempo promedio de carga de un programa es

$$T = c \sum_j \left(\pi_{i_j} \sum_{k \leq j} s_{i_k} \right)$$

donde la constante c depende de la densidad de grabado y la velocidad del dispositivo. Queremos construir un algoritmo goloso para determinar el orden en que se almacenan los programas que minimice T.
Analizar las siguientes estrategias de almacenamiento:

I. en orden no decreciente de los s_i
II. en orden no creciente de los π_i
III. en orden no creciente de $\frac{\pi_i}{s_i}$

¿Cómo se podría demostrar que la última estrategia es la mejor? (Sugerencia: analizar en cada caso qué ocurre cuando dos elementos contiguos desordenados se ordenan.)