# Algoritmos y Estructuras de Datos 3: Práctica 3

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Octubre 2012

# Objetivo de la práctica

En la primer parte de la materia, antes de comenzar a estudiar grafos, se explican varias técnicas algorítmicas generales:

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

El objetivo en esta práctica es ejercitar el uso de dichas técnicas en la formulación de algoritmos, y el análisis de los mismos.

# Objetivo de la práctica

En la primer parte de la materia, antes de comenzar a estudiar grafos, se explican varias técnicas algorítmicas generales:

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

El objetivo en esta práctica es ejercitar el uso de dichas técnicas en la formulación de algoritmos, y el análisis de los mismos.

# Objetivo de la práctica

En la primer parte de la materia, antes de comenzar a estudiar grafos, se explican varias técnicas algorítmicas generales:

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

El objetivo en esta práctica es ejercitar el uso de dichas técnicas en la formulación de algoritmos, y el análisis de los mismos.

# División

Si bien los ejercicios están numerados en forma continua y no hay secciones marcadas, la práctica se encuentra dividida implícitamente en cuanto a la temática de los ejercicios siguiendo fielmente 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 los ejercicios 17 y 18 son ejercicios generales en los que no se trata un problema particular, sino que se pide analizar los algoritmos ya realizados en ejercicios anteriores de esta y otras prácticas.

# División

Si bien los ejercicios están numerados en forma continua y no hay secciones marcadas, la práctica se encuentra dividida implícitamente en cuanto a la temática de los ejercicios siguiendo fielmente 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 los ejercicios 17 y 18 son ejercicios generales en los que no se trata un problema particular, sino que se pide analizar los algoritmos ya realizados en ejercicios anteriores de esta y otras prácticas.

# Recursión

Muchos algoritmos útiles son recursivos: Esto quiere decir que utilizan soluciones a subproblemas íntimamente relacionados, de la misma naturaleza que el problema principal que se quiere resolver, para construir una solución al problema completo.

Varias de las técnicas algorítmicas utilizan la recursión (por ejemplo Divide and Conquer, Programación Dinámica y 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

Muchos algoritmos útiles son recursivos: Esto quiere decir que utilizan soluciones a subproblemas íntimamente relacionados, de la misma naturaleza que el problema principal que se quiere resolver, para construir una solución al problema completo.
Varias de las técnicas algorítmicas utilizan la recursión (por ejemplo Divide and Conquer, Programación Dinámica y 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: Ejercicio 1

En este ejercicio se pide dar una función recursiva y una no recursiva para el cálculo de los números de Fibonacci, comparando sus complejidades. La intención es que el alumno practique el diseño de algoritmos recursivos y su análisis de complejidad.

# Recursión: Ejercicio 2

Aquí se plantea el problema de las torres de Hanoi y se pide escribir un algoritmo recursivo para resolverlo, así como analizar su complejidad y demostrar su correctitud. Los objetivos son los mismos que en el problema anterior pero sobre un problema más complejo.

# Divide and Conquer

Muchos algoritmos recursivos responden a un esquema general de divide and conquer: el problema original es *dividido en subproblemas independientes* que se resuelven recursivamente, y luego *se combinan las soluciones* a esos subproblemas para construir una solución al problema completo.

Como esta técnica está fuertemente basada en la recursión, es natural el paso de ejercicios sobre recursión a ejercicios sobre divide and conquer.

# Divide and Conquer

Muchos algoritmos recursivos responden a un esquema general de divide and conquer: el problema original es *dividido en subproblemas independientes* que se resuelven recursivamente, y luego *se combinan las soluciones* a esos subproblemas para construir una solución al problema completo.
Como esta técnica está 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. Es un problema de bastante más dificultad que la mayoría de los ejercicios de la práctica: por eso está marcado con un asterisco. Además, se describen como sugerencia las ideas centrales de un algoritmo eficiente de divide and conquer para resolver el problema.
Como en la mayoría de los ejercicios y de acuerdo a los objetivos de la práctica se pide también analizar la complejidad del algoritmo. En este caso como en muchos algoritmos de divide and conquer, la idea es que utilizen el teorema maestro visto en AED2 para hallar la complejidad.

# Divide and Conquer: Ejercicio 4

En este problema se pide dar un algoritmo de divide and conquer que arme un fixture para un torneo de $n$ jugadores que se enfrentan todos contra todos una vez.
Está dividido en dos partes: La parte a) pide resolver el caso en que $n$ es potencia de 2, para facilitar la división del problema en dos subproblemas de igual tamaño y simplificar el algoritmo resultante. La parte b) pide extender el algoritmo de la parte a) para que funcione para cualquier valor de $n$, para lo cual se deberán manejar los casos adicionales que resultan cuando en algún punto del algoritmo se encuentra un $n$ es impar.

# Algoritmos golosos

Los algoritmos golosos son en general fáciles de entender conceptualmente, y en general la parte más difícil consiste en demostrar su correctitud (o en general determinar si son correctos o no).

Conviene colocar los algoritmos golosos antes de empezar con la sección de programación dinámica, ya que esta última es una técnica más compleja de entender y manejar correctamente, al involucrar guardar resultados intermedios y analizar múltiples opciones en cada paso, mientras que los algoritmos golosos utilizan algún criterio para elegir de manera única un candidato en cada paso sin tener en cuenta posibilidades alternativas de solución global.

# Algoritmos golosos

Los algoritmos golosos son en general fáciles de entender conceptualmente, y en general la parte más difícil consiste en demostrar su correctitud (o en general determinar si son correctos o no).
Conviene colocar los algoritmos golosos antes de empezar con la sección de programación dinámica, ya que esta última es una técnica más compleja de entender y manejar correctamente, al involucrar guardar resultados intermedios y analizar múltiples opciones en cada paso, mientras que los algoritmos golosos utilizan algún criterio para elegir de manera única un candidato en cada paso sin tener en cuenta posibilidades alternativas de solución global.

# Algoritmos golosos: Ejercicio 5

En este ejercicio se pide analizar tres estrategias golosas posibles de ordenamiento de los programas en una cinta, con la intención de minimizar el tiempo promedio de carga del programa de la cinta.
La idea del ejercicio es mostrar con ejemplos que dos de las estrategias pueden dar resultados muy lejanos al óptimo en casos desfavorables, y demostrar la optimalidad de la tercera estrategia.

# Algoritmos golosos: Ejercicio 6

En este ejercicio se plantea el problema de dar el vuelto con mínima cantidad de monedas. Se plantean concretamente dos situaciones: el conjunto usual de monedas en circulación y un conjunto modificado de monedas (por ejemplo de 12 centavos).
El objetivo del ejercicio es que en primer lugar el alumno demuestre la correctitud del enfoque goloso en el caso usual, y que muestre con ejemplos que este enfoque no resulta óptimo con los valores modificados.

# Programación Dinámica

La técnica de programación dinámica consiste en resolver un problema recursivamente, pero calculando la respuesta para cada valor posible de los parámetros del problema una única vez, y reutilizándo este valor ya calculado cuando sea necesario.
Cuando se aplica adecuadamente, la programación dinámica reduce dramáticamente el tiempo de ejecución de soluciones recursivas al plantear la reutilización de resultados ya calculados, aprovechando la superposición de subproblemas.

Esta técnica es generalmente más difícil de entender que las anteriores, ya que para resolver un ejercicio hay que lograr plantear el espacio de estados posibles del algoritmo, y establecer una relación de recurrencia que permita calcular el resultado de unos estados en función de estados anteriores. Este estado no siempre se desprende fácilmente del enunciado del problema en sí, sino que debe incorporar información intermedia propia de la solución que se quiere construir.

# Programación Dinámica

La técnica de programación dinámica consiste en resolver un problema recursivamente, pero calculando la respuesta para cada valor posible de los parámetros del problema una única vez, y reutilizándo este valor ya calculado cuando sea necesario.
Cuando se aplica adecuadamente, la programación dinámica reduce dramáticamente el tiempo de ejecución de soluciones recursivas al plantear la reutilización de resultados ya calculados, aprovechando la superposición de subproblemas.
Esta técnica es generalmente más difícil de entender que las anteriores, ya que para resolver un ejercicio hay que lograr plantear el espacio de estados posibles del algoritmo, y establecer una relación de recurrencia que permita calcular el resultado de unos estados en función de estados anteriores. Este estado no siempre se desprende fácilmente del enunciado del problema en sí, sino que debe incorporar información intermedia propia de la solución que se quiere construir.

# Programación Dinámica : Ejercicio 7

En este ejercicio se pide usar programación dinámica para calcular los coeficientes binomiales, con la fórmula recursiva del triángulo de Pascal. En este problema se ilustra claramente la superposición de subproblemas, y se revee el primer ejercicio sobre la sucesión de Fibonacci para observar que la misma usada en este ejercicio permite calcular dicha sucesión con programación dinámica.

# Programación Dinámica : Ejercicio 8

En este ejercicio se pide dar un algoritmo de programación dinámica para obtener un camino de suma mínima cruzando una matriz de enteros de esquina a esquina. También se ilustra la superposición de subproblemas y el uso del principio del óptimo para construir una recursión que resuelva el problema.
Un punto importante es que en este ejercicio se pide dar el camino óptimo, y no solo su valor, con lo cual en este ejercicio se ilustra cómo puede utilizarse la tabla de valores calculada con programación dinámica para reconstruir un camino óptimo.

# Programación Dinámica : Ejercicio 9

En este caso se pide dar un algoritmo de PD para encontrar una secuencia de operaciones a intercalar en una secuencia de números dada, con el fin de obtener un resultado fijo dado (si es posible). Se ilustran las mismas ideas que en el caso anterior pero en un ejemplo más complejo donde la eleccion del espacio de subproblemas es mas dificil.

# Programación Dinámica : Ejercicio 10

En este ejercicio se plantea una situación concreta donde se fabrican radios mensualmente, pero se puede imaginar un esquema general del problema con cantidad arbitraria de meses y valores distintos de los costos. Se pide obtener un plan óptimo de producción usando PD. Este ejercicio está marcado con un asterisco. Tiene más complejidad que otros debido a que hay mucha información en juego (más parámetros y datos diferentes que en otros problemas): la especificación de un subproblema involucra el mes que se está considerando actualmente y la cantidad de radios actualmente en stock, y 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 lo cual 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 : Ejercicio 11

En este problema se pide dar un algoritmo de programación dinámica para el problema de dar el vuelto con la menor cantidad de monedas, y analizar en qué casos funciona.
La idea de este problema es mostrar que a diferencia del algoritmo goloso presentado en el ejercicio 6, el algoritmo de programación dinámica funciona en todos los casos sin importar los valores de las monedas, a cambio de un costo mayor en cuanto a complejidad temporal.

# Programación Dinámica : Ejercicio 12

En este problema se pide dar un algoritmo de PD para calcular la distancia de edición entre dos palabras, y una secuencia mínima de operaciones para transformar una de las palabras en la otra.
Este problema se suele contar brevemente en las clases teóricas como ejemplo de uso de programación dinámica, pero aquí se pide a los alumnos dar el algoritmo preciso incluyendo la reconstrucción de una secuencia de operaciones que transforme una palabra en la otra.

# Programación Dinámica : Ejercicio 13

Este es un ejercicio teórico de programación dinámica y por eso va al final, cuando se supone que el alumno ya ha realizado ejercicios y entiende la técnica de programación dinámica. El ejercicio pide dar ejemplos de problemas donde no valga el principio de optimalidad, de manera que no sea posible usar programación dinámica directamente para resolverlo.

# Backtracking

La técnica de backtracking propone una alternativa a la fuerza bruta. En lugar de probar todas las soluciones posibles individualmente, se va construyendo una solución de manera incremental, y se detiene la construcción tan pronto como se detecte que esa solución parcial no puede llevar a una solución suficientemente buena para los propósitos del problema que se pretende resolver.

Esta técnica da lugar usualmente a algoritmos exponenciales, y se la usa para tratar problemas computacionalmente difíciles. Si bien no es particularmente difícil de entender conceptualmente, implementar o diseñar en detalle buenos algoritmos de backtracking requiere en general un buen manejo de la recursión. También hay que identificar y mantener consistentemente durante toda la ejecución la información relevante de la solución que se está construyendo.

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

# Backtracking

La técnica de backtracking propone una alternativa a la fuerza bruta. En lugar de probar todas las soluciones posibles individualmente, se va construyendo una solución de manera incremental, y se detiene la construcción tan pronto como se detecte que esa solución parcial no puede llevar a una solución suficientemente buena para los propósitos del problema que se pretende resolver.
Esta técnica da lugar usualmente a algoritmos exponenciales, y se la usa para tratar problemas computacionalmente difíciles. Si bien no es particularmente difícil de entender conceptualmente, implementar o diseñar en detalle buenos algoritmos de backtracking requiere en general un buen manejo de la recursión. También hay que identificar y mantener consistentemente durante toda la ejecución la información relevante de la solución que se está construyendo.
Teniendo en cuenta esto y que se la suele aplicar a problemas computacionalmente difíciles, es razonable poner esta técnica al final.

# Backtracking : Ejercicio 14

En este problema se pide implementar un algoritmo de backtracking para el clásico problema de las 8 reinas en un tablero de ajedrez, compararlo con un algoritmo de fuerza bruta para el mismo problema, y determinar si es un buen algoritmo (tiempo polinomial). De esta forma se introduce la idea de que backtracking es una técnica que permite encarar problemas computacionalmente difíciles aunque los algoritmos resultantes puedan ser exponenciales.

# Backtracking : Ejercicio 15

En este ejercicio se pide escribir un algoritmo de backtracking para el problema de la mochila. Este problema es NP completo, como se explicará al final de la materia, con lo cual se sigue mostrando que un backtracking es aplicable en general a problemas computacionalmente difíciles para los que no se conozca una solución mejor que explorar todas las posibilidades de solución exhaustivamente.

# Backtracking : Ejercicio 16

En este ejercicio se pide dar un algoritmo de fuerza bruta para el problema de suma de subconjuntos, que también es NP completo al igual que el del ejercicio anterior. Se pide además analizar la complejidad del algoritmo dado y se pregunta al alumno si se le ocurre una idea mejor. La idea es reforzar la diferencia de eficiencia entre un algoritmo de backtracking, que corta un subarbol de exploracion de soluciones completo tan pronto como es posible, y uno de fuerza bruta que explora todas las posibilidades una por una.

# Ejercicios teóricos generales

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

# Ejercicio teóricos generales : Ejercicio 17

En este ejercicio se pide determinar cuales ejercicios de las prácticas 2 y 3 son algoritmos de divide and conquer, y cuales son algoritmos recursivos. Además se pregunta si siempre es mejor usar una versión recursiva de un algoritmo o viceversa.
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. Se puede ver como ejemplos los ejercicios 1 y 3 de esta práctica: En uno es mejor un algoritmo recursivo, y en otro es mejor uno iterativo.

# Ejercicio 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.

# 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 $\pi_i$ 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 $\pi_i$
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.)