AED 3: Técnicas algorítmicas

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Abril 2018

Objetivo de la práctica

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

- Dividir y conquistar
- Algoritmos golosos
- Programación dinámica
- Búsqueda exhaustiva
- Algoritmos probabilísticos (mencionado, pero fuera de programa)

Objetivos de la práctica:

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

Objetivo de la práctica

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

- Dividir y conquistar
- Algoritmos golosos
- Programación dinámica
- Búsqueda exhaustiva
- Algoritmos probabilísticos (mencionado, pero fuera de programa)

Objetivos de la práctica:

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

Objetivo de la práctica

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

- Dividir y conquistar
- Algoritmos golosos
- Programación dinámica
- Búsqueda exhaustiva
- Algoritmos probabilísticos (mencionado, pero fuera de programa)

Objetivos de la práctica:

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

División

Dividimos la práctica en 5 partes, por técnica algorítmica utilizada. Las presentamos en el siguiente orden:

- Dividir y conquistar: 2 ejercicios, “repaso” AED II
- Algoritmos golosos: 3 ejercicios
- Programación dinámica: 7 ejercicios
- Búsqueda exhaustiva 2 ejercicios
- Incluimos al final una sección marcada **opcional**, con dos ejercicios sobre algoritmos probabilísticos.

A lo largo de la práctica se marcan con **(*)** varios ejercicios que no introducen **nuevas** ideas importantes, sino que combinan y refuerzan ejercicios anteriores.

Dividir y conquistar

- 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**.
- Conocida de Algoritmos II

Dividir y conquistar: Ejercicio 1

- Este ejercicio pide dar un algoritmo de D&C para encontrar el **elemento mayoría** de un arreglo.
- Se ejercita descomposición en subproblemas
- Se pide complejidad: Teorema Maestro de AED2, o analogía con Merge-Sort.

Dividir y conquistar: Ejercicio 2

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

- Marcado opcional (D&C es repaso, y ejercitado en 1)
- Relación con problemas de grafos posteriores de la materia:
  - Coloreo de aristas en K_n
  - Coloreo de aristas en $K_{n,n}$

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:
  - En los 3 ejercicios se ejercita demostrar la correctitud de una estrategia correcta.

Algoritmos golosos: Ejercicios 3 y 4

- Ejercicio 3: Pintar casillas sin colisionar con vecinos.
- Fácil de intuir algoritmo goloso correcto.
- Demostración simple, pero representativa de los golosos más clásicos.
- Ejercicio 4: Cubrir con intervalos el $[0, N]$

Algoritmos golosos: Ejercicio 5

- Ejercicio 6: Problema de dar el vuelto con mínima cantidad de monedas (será revisitado con Programación Dinámica).
- Se pide probar que es óptimo para las monedas usuales.
- Se pide dar un contraejemplo para otro caso.
- Marcado opcional: se profundizará en DP, y ya se han ejercitado dos demostraciones de correctitud.

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 6,7,8

- Ejercicio 6: Calcular Fibonacci con recursion directa, y con DP.
  - Ejemplifica explícitamente superposición de subproblemas.
- Ejercicio 7: Elegir números con suma máxima, sin elegir consecutivos
  - Ejemplo sencillo de recursión para empezar.
  - Similar a Fibonacci, pero en un problema **de optimización**.
  - Primer ejemplo de uso del **principio del óptimo**.
  - Se compara explícitamente con el 3, que generaliza.
  - Se pregunta explícitamente si aquí funciona el goloso usado en 3.
- Ejercicio 8: Camino mínimo cruzando una matriz de esquina a esquina.
  - También ilustra superposición de subpr. y princ. del óptimo.
  - Se ejercita **reconstrucción del camino**, además del valor.

Programación Dinámica : Ejercicio 9

- Ejercicio 9: Cubrir $[0, n]$ con intervalos **con costos**.
- Generaliza 4, y se pide mostrar que en esta variante el goloso no es correcto.
- Se pide resolver en $O(m^2)$ con programación dinámica.
- Este problema aparece en prácticas posteriores de AED 3 como problema de caminos mínimos/máximos en DAG.

Programación Dinámica : Ejercicios 10, 11 y 12

- Ejercicio 10: 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 5.
- Ejercicio 11: Identificar todas las subcadenas palindrómicas.
  - Ejemplo donde los subproblemas son **intervalos** dentro del rango de índices.
  - La solución está guiada pues se sugiere la tabla de subproblemas a utilizar.
- Ejercicio 12: Descomponer una cadena en la mínima cantidad de palíndromos.
  - Opcional: toma ideas de ejercicios 11 y 9.
  - El patrón de la recursión es similar al ejercicio 9.

Búsqueda exhaustiva

- Tema central: Backtracking como 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 en los ejercicios se pretende 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 13 y 14

- Ejercicio 13: Problema de suma de subconjunto.
  - Se pide resolver con fuerza bruta y con backtracking
  - Se pide explícitamente analizar las diferencias entre ambas
  - Se propone buscar otra manera de resolverlo con programación dinámica (muy similar a ejercicio 10)
- Ejercicio 14: Problema de la partición
  - Opcional, no hay conceptos importantes nuevos
  - Se propone relacionar con el problema anterior (ejemplo temprano de reducción entre problemas)

Algoritmos probabilísticos

Finalmente, terminan la práctica dos ejercicios opcionales sobre ejemplos de algoritmos probabilísticos:

- Ejercicio 15: Algoritmo Las Vegas para el Problema de la Partición
  - Consiste en aleatorizar en qué conjunto se intenta ubicar cada elemento inicialmente.
  - Se describe, y se propone implementar y comparar con el determinista.
- Ejercicio 16: Algoritmo Montecarlo para verificar si $AB = C$ con matrices.
  - Consiste en probar si $A(Bv) = Cv$ para varios v aleatorios.