Algorítmica y Lenguajes de Programación

Algoritmos voraces y “divide y vencerás”

Algoritmos voraces. Introducción (i)

- Las personas glotonas (voraces) intentan coger tanto como pueden en cada momento.
- Los algoritmos voraces intentar “coger” la solución que parece más adecuada en ese mismo momento.
- Los buenos algoritmos voraces pueden resolver correctamente los problemas.
- Un algoritmo voraz llega a una solución realizando selecciones, donde cada selección se hace en base de lo que es mejor en cada momento. Sin embargo, un buen algoritmo voraz puede llegar de este modo a una solución globalmente óptima.

Algoritmos voraces. Introducción (ii)

- Es uno de los esquemas más simples y al mismo tiempo de los más utilizados.
- Típicamente se emplea para resolver **problemas de optimización**:
  - existe una entrada de tamaño *n* que son los **candidatos** a formar parte de la solución;
  - existe un subconjunto de esos *n* candidatos que satisface ciertas restricciones: se llama **solución factible**;
  - hay que obtener la solución factible que maximice o minimice una cierta función objetivo: se llama **solución óptima**.
- Ejemplos:
  - encontrar la secuencia óptima para procesar un conjunto de tareas por un computador,
  - proporcionar una cantidad de dinero con el menor número de monedas
  - etc.

3

Algoritmos voraces. Introducción (iii)

- El **esquema voraz** procede por pasos:
  - inicialmente el conjunto de candidatos escogidos es vacío;
  - en cada paso, se intenta añadir al conjunto de los escogidos "el mejor" de los no escogidos (sin pensar en el futuro), utilizando una **función de selección** basada en algún criterio de optimización (puede ser o no ser la función objetivo);
  - tras cada paso, hay que ver si el conjunto seleccionado es **completable** (i.e., si añadiendo más candidatos se puede llegar a una solución);
    - si el conjunto no es completable, se rechaza el último candidato elegido **y no se vuelve a considerar en el futuro**;
    - si es completable, se incorpora al conjunto de escogidos **y permanece siempre en él**;
  - tras cada incorporación se comprueba si el conjunto resultante es una solución;
  - el algoritmo termina cuando se obtiene una solución;
  - el algoritmo es correcto si la solución encontrada es siempre óptima;

4

Algoritmos voraces. Ejemplo (i)

- Problema del cambio de monedas.
  - Se trata de devolver una cantidad de euros con el menor número posible de monedas.
  - Se parte de:
    - un conjunto de tipos de monedas válidas, de las que hay cantidad suficiente, y de
    - un importe a devolver.
- Elementos fundamentales del esquema:
  - **Conjunto de candidatos**: cada una de las monedas de los diferentes tipos que se pueden usar para realizar el desglose del importe dado.
  - **Solución**: un conjunto de monedas devuelto tras el desglose y cuyo valor total es igual al importe a desglosar.
  - **Completable**: la suma de los valores de las monedas escogidas en un momento dado no supera el importe a desglosar.
  - **Función de selección**: elegir si es posible la moneda de mayor valor de entre las candidatas.
  - **Función objetivo**: número total de monedas utilizadas en la solución (debe minimizarse).

Algoritmos voraces. Ejemplo (ii)

- Supongamos que tenemos monedas de 1, 2, 5, 10, 20 y 50 céntimos de euro. Estas monedas son nuestros **candidatos**.
- Queremos devolver 60 céntimos; esta es nuestra **solución**.
- La función de selección consiste en **coger la moneda de mayor valor disponible**.
- **Procedimiento:**
  - Al comenzar la moneda de mayor valor es la de 50 céntimos, la cogemos y la cantidad de que disponemos (50) no es mayor que la solución; así, la moneda de 50 se mantiene en la solución.
  - La siguiente moneda de mayor valor disponible sigue siendo de 50 céntimos, la cantidad que tenemos ahora es de 1 euro; al ser mayor que la cantidad a devolver la moneda de 50 céntimos no pasa a la solución y se elimina de la lista de candidatos.
  - Ahora se puede coger la moneda de 20 céntimos pero la cantidad total es de 70 céntimos así que no pasa a la solución y se elimina de la lista de candidatos.
  - Se coge la moneda de 10 céntimos, la cantidad no sólo no es mayor que la solución sino que es exactamente la solución. La moneda pasa a la solución y el algoritmo finaliza.

Algoritmos voraces. Ejemplo (iii)

- El algoritmo anterior funciona correctamente con las monedas de euro.
- Supongamos que en Palombia hay monedas de 12, 10, 5 y 1 céntimo de palombino.
- Nuestro algoritmo daría 16 céntimos de palombino como 12+1+1+1+1.
- Sin embargo, la solución óptima sería 10+5+1.
- **Moraleja:**
  - Los algoritmos voraces son fáciles de programar,
  - a menudo producen buenas (incluso óptimas) soluciones,
  - pero también pueden “fracasar”.

7

Algoritmos voraces. Consideraciones sobre su corrección (i)

- La selección de una candidata óptima en cada paso es una estrategia heurística que no siempre conduce a la solución óptima.
- En ocasiones, se utilizan heurísticas voraces para obtener soluciones subóptimas cuando el cálculo de las óptimas es demasiado costoso.
- **¿Puede saberse si una estrategia voraz servirá para resolver un problema concreto de optimización?**
  - La respuesta es: NO siempre.
  - Sin embargo, existen ciertos indicios...: la *propiedad de la selección voraz* y la existencia de una *subestructura óptima*.

8

Algoritmos voraces. Consideraciones sobre su corrección (ii)

- La propiedad de la selección voraz:
  - Se verifica esta propiedad cuando una solución globalmente óptima puede ser alcanzada mediante selecciones localmente óptimas que son tomadas en cada paso sin depender de las selecciones futuras;
  - en otras palabras, una estrategia voraz progresa de arriba hacia abajo, tomando una decisión voraz tras otra, reduciendo iterativamente el problema a otro más pequeño.

9

Algoritmos voraces. Consideraciones sobre su corrección (iii)

- **¿Cómo se comprueba si se verifica la propiedad de la selección voraz?**
  - Normalmente, se examina una solución globalmente óptima,
  - se trata de demostrar que esa solución puede ser manipulada de forma que se obtiene tras una primera selección voraz (localmente óptima),
  - y esa selección reduce el problema a otro similar pero más pequeño;
  - se aplica inducción para demostrar que se puede usar una selección voraz en cada paso
  - **hay que demostrar que una solución óptima posee una "subestructura óptima"**
- **Subestructura óptima:**
  - Un problema posee una subestructura óptima si una solución óptima de ese problema incluye soluciones óptimas de subproblemas.

10

Divide y vencerás. Introducción (i)

- Técnica de diseño de algoritmos "divide y vencerás":
  - descomponer, en un tiempo máximo *g(n)*, el ejemplar a resolver, de tamaño *n*, en un cierto número, *x*, de subejemplares de tamaño *n/y*, cada uno del mismo tipo que el problema original;
  - resolver independientemente, y generalmente de forma recursiva, cada uno de los *x* subejemplares en un tiempo *T(n/y)*;
  - combinar los resultados obtenidos para los *x* subproblemas para construir la solución del ejemplar original en un tiempo *h(n)*.

11

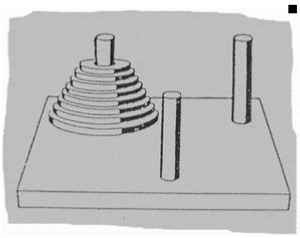
Divide y vencerás. Introducción (ii)

- Así, la ecuación de recurrencia resultante es de la forma
  - $T(n) = x \cdot (n/y) + f(n)$
  - Donde $f(n) = g(n) + h(n)$
- En otras palabras, "divide y vencerás" consiste en "romper" un problema de tamaño n en problemas más pequeños de tal forma que la solución de dichos problemas permita construir fácilmente una solución para el problema original.

12

Divide y vencerás. Ejemplo (i)

- El juego de las torres de Hanoi es un juego oriental muy antiguo que hoy se conoce en todo el mundo.
- Consta de tres columnas y una serie de discos de distintos tamaños. Los discos están acomodados de mayor a menor en una de las columnas.
- El juego consiste en pasar todos los discos a otra de las columnas y dejarlos como estaban: de mayor a menor.
  - Las reglas del juego son las siguientes:
    - Sólo se puede mover un disco cada vez.
    - Para cambiar los discos de lugar se pueden usar las tres columnas.
    - Nunca deberá quedar un disco grande sobre un disco pequeñó.



13

Divide y vencerás. Ejemplo (ii)

- El problema de las torres de Hanoi se puede resolver de forma muy sencilla con un enfoque divide y vencerás.
- El problema consiste en desplazar *n* discos de la columna *i* de inicio a la columna *f* de fin utilizando la columna *t* como un "almacenamiento" temporal.
- El problema menor consiste en mover *n-1* discos de *i* a *t* utilizando *f* como soporte temporal.
- De esta forma es posible mover un disco (el que queda) desde *i* hasta *f* y, después, mover los *n-1* discos de *t* a *f*.

14

Divide y vencerás. Ejemplo (iii)

```
acción Hanoi (n Î entero, i,f,t Î soporte)
inicio
  si n=1 entonces
    mover un disco de i a f
  si no
    llamar Hanoi (n-1,i,t,f)
    mover un disco de i a f
    llamar Hanoi (n-1,t,f,i)
  fin si
fin
```

15

Algoritmos voraces. Resumen

- Es uno de los esquemas más simples y al mismo tiempo de los más utilizados.
- Típicamente se emplea para resolver **problemas de optimización**:
  - existe una entrada de tamaño *n* que son los **candidatos** a formar parte de la solución;
  - existe un subconjunto de esos *n* candidatos que satisface ciertas restricciones: se llama **solución factible**;
  - hay que obtener la solución factible que maximice o minimice una cierta función objetivo: se llama **solución óptima**.
- El **esquema voraz** procede por pasos:
  - inicialmente el conjunto de candidatos escogidos es vacío;
  - en cada paso, se intenta añadir al conjunto de los escogidos “el mejor” de los no escogidos (sin pensar en el futuro), utilizando una **función de selección** basada en algún criterio de optimización (puede ser o no ser la función objetivo);
  - tras cada paso, hay que ver si el conjunto seleccionado es **completable** (i.e., si añadiendo más candidatos se puede llegar a una solución);
  - tras cada incorporación se comprueba si el conjunto resultante es una solución;
  - el algoritmo termina cuando se obtiene una solución;
  - el algoritmo es correcto si la solución encontrada es siempre óptima;

16

Algoritmos “divide y vencerás”. Resumen

- Técnica de diseño de algoritmos “divide y vencerás”:
  - descomponer, en un tiempo máximo *g(n)*, el ejemplar a resolver, de tamaño *n*, en un cierto número, *x*, de subejemplares de tamaño *n/y*, cada uno del mismo tipo que el problema original;
  - resolver independientemente, y generalmente de forma recursiva, cada uno de los *x* subejemplares en un tiempo *T(n/y)*;
  - combinar los resultados obtenidos para los *x* subproblemas para construir la solución del ejemplar original en un tiempo *h(n)*.
- En otras palabras, “divide y vencerás” consiste en “romper” un problema de tamaño n en problemas más pequeños de tal forma que la solución de dichos problemas permita construir fácilmente una solución para el problema original.

17