Divide & Conquer

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Octubre 2012

¿En qué consiste un algoritmo de divide & conquer?

- Muchos algoritmos de utilidad son recursivos: para resolver un problema, se utilizan las soluciones a subproblemas fuertemente relacionados.
- Estos algoritmos suelen ajustarse a un esquema de divide and conquer: Se divide el problema en varios subproblemas que luego se resuelven y se combinan las soluciones obtenidas para resolver el original.

¿En qué consiste un algoritmo de divide & conquer?

- Muchos algoritmos de utilidad son recursivos: para resolver un problema, se utilizan las soluciones a subproblemas fuertemente relacionados.
- Estos algoritmos suelen ajustarse a un esquema de divide and conquer: Se divide el problema en varios subproblemas que luego se resuelven y se combinan las soluciones obtenidas para resolver el original.

El esquema general

- Los algoritmos de divide and conquer se organizan en 3 pasos que responden al siguiente esquema general:
- **Divide:** Se divide el problema en subproblemas más pequeños (más fáciles), similares al problema original.
- **Conquer:** Se resuelven cada uno de los subproblemas: Esto se hace típicamente de manera recursiva, invocando el mismo algoritmo de divide and conquer, pero al llegar a instancias suficientemente pequeñas se utiliza algún otro método como caso base de la recursión.
- **Combine:** Se combinan las soluciones de los subproblemas para obtener una solución al problema original

El esquema general

- Los algoritmos de divide and conquer se organizan en 3 pasos que responden al siguiente esquema general:
- **Divide:** Se divide el problema en subproblemas más pequeños (más fáciles), similares al problema original.
- **Conquer:** Se resuelven cada uno de los subproblemas: Esto se hace típicamente de manera recursiva, invocando el mismo algoritmo de divide and conquer, pero al llegar a instancias suficientemente pequeñas se utiliza algún otro método como caso base de la recursión.
- **Combine:** Se combinan las soluciones de los subproblemas para obtener una solución al problema original

Referencias

- Introduction to Algorithms, 2nd Edition. MIT Press. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Página 28: 2.3.1 The divide-and-conquer approach

El problema de armar fixtures

Ejercicio 3.4, práctica 3 de Algoritmos y Estructuras de Datos 3:

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.

El problema de armar fixtures

Ejercicio 3.4, práctica 3 de Algoritmos y Estructuras de Datos 3:

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.

El problema de armar fixtures (cont)

Supondremos que tenemos disponible una operación:
`jugar(fecha, jugador1, jugador2)`
que reporta que se debe jugar un partido entre esos jugadores en esa fecha (Podría consistir en loguear el evento por ejemplo, o en agregarlo a una lista de partidos para ser procesada luego).
Numeraremos los jugadores con enteros consecutivos desde 0 hasta $n - 1$, e igualmente numeraremos las fechas con enteros desde 0.

Solución (parte a)

Utilizando divide and conquer:

```
1  reportarPartidos(listaJugadores):
2      n <- cantidad de jugadores
3      // Si hay un solo jugador, no hace falta jugar nada.
4      SI n > 1:
5          j1 <- lista de los primeros n/2 jugadores de listaJugadores
6          j2 <- lista de los ultimos n/2 jugadores de listaJugadores
7          // Armamos las primeras fechas del fixture recursivamente
8          reportarPartidos(j1)
9          reportarPartidos(j2)
10         // Y armamos las últimas n/2 fechas para obtener el fixture final
11         PARA i <- 0 HASTA n/2 - 1:
12             PARA j <- 0 HASTA n/2 - 1:
13                 jugar(n/2 - 1 + i, j1[j], j2[(i+j) %(n/2)])
```

Solución (parte b)

¿Cómo se puede adaptar la idea de la parte a para resolver la parte b?

Solución (parte b)

¿Cómo se puede adaptar la idea de la parte a para resolver la parte b?

- Para resolver los casos impares, podemos agregar un jugador "centinela", de manera que ahora una fecha donde un jugador no juega se interpreta como una fecha donde juega contra el centinela. El problema equivale ahora a armar un fixture con los $n + 1$ jugadores que quedan, y podemos reducir el problema al caso par.
- En el divide and conquer del caso par, ahora puede pasar que $\frac{n}{2}$ resulte ser impar (aunque n sea par), y por lo tanto los subproblemas contendrán partidos con jugadores libres. Si emparejamos en cada fecha a los jugadores libres de cada torneo, podemos proceder similar a como hacíamos en la parte a y armar un fixture en $n - 1$ fechas.