2 Schema divide-et-impera

Una tecnica generale per risolvere un problema complesso e' di decomporlo in problemi *dello stesso tipo*, ma di dimensioni minori. Formalmente, tale tecnica corrisponde a definire ricorsivamente una funzione (parziale) $f: I \rightharpoonup O$, in termini di due funzioni ausiliarie $d: I \to I\ list$ per decomporre un problema in sottoproblemi e $c: I \times (O\ list) \to O$ per costruire la soluzione al problema a partire dalle soluzioni ai sottoproblemi. In ML lo schema generale del divide-et-impera e' espresso mediante il seguente funzionale ricorsivo

```
DC : (I -> I list) -> (I * O list -> O) -> I -> O
fun DC D C x = C(x,map (DC D C) (D x));
fun F x = DC D C x; (* o piu' direttamente *)
fun F x = C(x,map F (D x));
```

In pseudo-Pascal l' algoritmo ricorsivo per calcolare f, supponendo di aver gia' disponibili degli algoritmi per d e c, e'

```
F(x)
  (x_1,...,x_k) <- D(x)          {dividi}
  for i=1 to k do y_i <- F(x_i)  {risolvi sotto-problemi}
  y <- C(x,(y_1,...,y_k))          {combina}
  return(y)
```

- Dimostrazione di terminazione degli algoritmi, ovvero f e' totale, per induzione su $|x|$. Basta verificare che
  - $(x_{i\in k}) = d(x)$ implica $\forall i \in k.|x_i| < |x|$

  si osservi che quando $|x| = 0$ bisogna avere $k = 0$.
- Dimostrazione di correttezza parziale, cioe' $P(x, f(x))$ se $f(x)$ definita, per induzione su "altezza dell'albero delle chiamate ricorsive per il calcolo di $f(x)$". Basta verificare che
  - $(x_{i\in k}) = d(x)$ e $\forall i \in k.P(x_i, y_i)$ implica $P(x, c(x, (y_{i\in k})))$

  se si sa' anche che f e' totale, allora consegue la correttezza totale.

Molto spesso e' sufficiente uno schema divide-et-impera piu' semplice, in cui esiste un k prefissato t.c. il numero di sottoproblemi in cui si decompone x e' o 0 o k.

```
F(x)
  if trivial(x)   {test}
  then y <- solve(x)      {risolvi immediatamente}
  else (x_1,...,x_k) <- D(x)              {dividi}
       for i=1 to k do y_i <- F(x_i)     {risolvi sotto-problemi}
       y <- C(x,y_1,...,y_k)              {combina}
  return(y)
```

Le tecniche di dimostrazione illustrate sopra si adattano facilmente a questa variante, per esempio:

- Dimostrazione di correttezza parziale, per induzione su "altezza dell'albero delle chiamate ricorsive per il calcolo di $f(x)$". Basta verificare che
  - $trivial(x)$ implica $P(x, solve(x))$
  - $\neg trivial(x)$ e $(x_{i\in k}) = d(x)$ e $\forall i \in k.P(x_i, y_i)$ implica $P(x, c(x, y_{i\in k}))$.

Example 2.1 Applicare le idee generali del divide-et-impera per analizzare Mergesort. Trattare le sequenze come un tipo di dato astratto. Parametrizzare lo studio della correttezza e dell'analisi di complessita' di Mergesort in funzione di una specifica delle proprieta' delle operazioni di composizione e decomposizione.

2.1 Complessita' di algoritmi ricorsivi e Master Method

Per poter fare stime di complessita' di algoritmi ricorsivi basati sulla tecnica del divide-et-impera, occorre anzitutto definire la **complessita' esatta** $T_f: I \rightharpoonup M$ dell'algoritmo F (dove M e' uno dei monoidi ordinati precedentemente introdotti per misurare la complessita'). L'approccio piu' ovvio e' di definire T_f ricorsivamente, in analogia con la definizione di f, supponendo di disporre di funzioni $T_d: I \to M$ e $T_c: I \times (O\ list) \to M$ per la complessita' esatta degli algoritmi che calcola le funzioni d e c. La definizione ricorsiva di T_f in funzione di f, T_d e T_c e' data da (dove il + e' quello del monoide ordinato M)

$$T_f(x) = T_d(x) + (\sum_{i \in k} T_f(x_i)) + T_c(x, (y_{i \in k})) \quad \text{dove } (x_{i \in k}) = d(x) \text{ e } y_i = f(x_i)$$

raggruppando in un unico termine $t_f(x)$ il costo di divisione e combinazione associato ad x, cioe'

$$t_f(x) = T_d(x) + T_c(x, (y_{i \in k})) \quad \text{dove } (x_{i \in k}) = d(x) \text{ e } y_i = f(x_i)$$

possiamo riscrivere la definizione della complessita' esatta T_f nel modo seguente

$$T_f(x) = (\sum_{i \in k} T_f(x_i)) + t_f(x) \quad \text{dove } (x_{i \in k}) = d(x)$$

Nel seguito supporremo che la funzione di decomposizione produce sotto-problemi di dimensione strettamente minore, cioe' $\forall i \in k.|x_i| < |x|$. Tra le altre cose questa ipotesi garantisce che f, t_f e T_f sono totali.
Per dimostrare che $T: N \to M$ e' una maggiorazione della complessita' esatta T_f, cioe' $\forall x \in I.T_f(x) \le T(|x|)$, basta verificare che

- $(x_{i \in k}) = d(x)$ implica $(\sum_{i \in k} T(|x_i|)) + t_f(x) \le T(|x|)$

Il precedente criterio fornisce un criterio sufficiente per dimostrare che T e' una maggiorazione di T_f, ma non fornisce alcun aiuto per trovare T. Diamo adesso un metodo per definirsi *implicitamente* un buon cadidato per T. Supponiamo che

- $t: N \to M$ sia una maggiorazione di t_f, cioe' $t_f(x) \le t(|x|)$
- $a: N \to N$ sia una maggiorazione sul numero dei sottoproblemi, cioe' $k \le a(|x|)$ dove $x_{i \in k} = d(x)$
- $s: N \to N$ sia una **buona** maggiorazione per la dimensione dei sottoproblemi, cioe' $\forall i \in k.|x_i| \le s(|x|) < |x|$ dove $x_{i \in k} = d(x)$ e $|x| > 0$
- infine supponiamo che le funzioni t, a e s siano **monotone**

allora la funzione T definita (per induzione su n) da

$$T(0) = t(0) \quad \text{e} \quad T(n) = a(n) * T(s\ n) + t(n) \quad \text{se } n > 0$$

e' **monotona** ed e' una maggiorazione di T_f. In genere siamo interessati a stimare la complessita' degli algoritmi a meno di fattori costanti, cioe' a sapere se $T \in O(g)$. Per alcune T definite mediante relazioni di ricorrenza vi sono delle formule preconfezionate, che vanno sotto il nome di Master Method. Supponiamo che T sia definita dalla seguente relazione di ricorrenza, dove $a \ge 1$ e $b > 1$, mentre t e' una funzione data

$$T(1) = t(1) \quad \text{e} \quad T(n) = a * T(\lceil n/b \rceil) + t(n) \quad \text{se } n > 1$$

e sia $c \stackrel{\Delta}{=} \log_b a$, allora valgono le seguenti implicazioni:

- se $t(n) \in O(n^{c-\epsilon})$ allora $T(n) \in \Theta(n^c)$,
- se $t(n) \in \Theta(n^c)$ allora $T(n) \in \Theta(n^c * \log n)$,
- se $t(n) \in \Omega(n^{c+\epsilon})$ (e vale una condizione di regolarita' su t) allora $T(n) \in \Theta(t)$.

Per esempio quando si analizza un algoritmo divide-et-impera, $t(n)$ corrisponde al costo di divisione e combinazione di un problema di dimensione n, a e' il numero di sottoproblemi, e b misura di quanto si riducone le dimensioni dei sottoproblemi. Il primo caso si verifica quando il costo della ricorsione predomina su quello della divisione e combinazione (quindi anche se esse fossero a costo 0 non si otterrebbero miglioramenti significativi), il terzo caso si verifica quando il costo della divisione e combinazione e' predominante. Non abbiamo esplicitato la condizione di regolarita', poiche' tutte le funzioni che si usano in pratica la soddisfano.