Corso di Algoritmi e Strutture Dati—Modulo 2

Esercizi su Divide-et-Impera – 5 Aprile 2023

Moreno Marzolla e Jocelyne Elias

Esercizio 1. Consideriamo un array $A[1..n]$ composto da $n \geq 0$ valori reali, non necessariamente distinti. L'array è ordinato in senso non decrescente. Scrivere un algoritmo ricorsivo di tipo divide-et-impera che restituisca *true* se e solo se A contiene valori duplicati. Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione.

```
DUPLICATI( real A[1..n], integer i, integer j ) → bool
  if ( i ≥ j ) then
      return false;
  else
      integer m ← Floor( ( i + j ) / 2 );
      return ( DUPLICATI(A, i, m) or DUPLICATI(A, m+1, j) or A[m] = A[m+1] );
  endif
```

Nel caso in cui il sottovettore $A[i..j]$ sia vuoto oppure contenga un singolo elemento ($i \geq j$), l'algoritmo restituisce *false*, in quanto non possono esistere duplicati. Altrimenti, l'algoritmo restituisce true se e solo se una delle tre seguenti alternative è vera:

1. Esistono duplicati nel sottovettore $A[i..m]$; oppure
2. Esistono duplicati nel sottovettore $A[m+1..j]$; oppure
3. I due elementi "a cavallo" dei sottovettori precedenti ($A[m]$ e $A[m+1]$) sono uguali.

In base al Master Theorem, l'algoritmo ha costo $T(n) = \Theta(n)$.

Esercizio 2. Scrivere un algoritmo ricorsivo di tipo divide-et-impera che, dato un array $A[1..n]$ di valori reali, restituisce *true* se e solo se A è ordinato in senso non decrescente, cioè se $A[1] \leq A[2] \leq \ldots \leq A[n]$. Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione. L'algoritmo seguente ritorna true se e solo se il sottovettore $A[i..j]$ è ordinato in senso non decrescente; per controllare se l'intero vettore è ordinato, sarà sufficiente invocarlo come ORDINATA(A, 1, n).

```
ORDINATA( real A[1..n], integer i, integer j ) → boolean
  if ( i ≥ j ) then
      return true;
  else
      integer m ← FLOOR( (i + j) / 2 );
      return ( A[m] ≤ A[m+1] and ORDINATA(A, i, m) and ORDINATA(A, m+1, j) );
  endif
```

Il costo $T(n)$ dell'algoritmo nel caso peggiore è espresso dalla seguente equazione di ricorrenza:

$$T(n) = \begin{cases} c_1 & \text{se } n \leq 1 \\ 2\mathrm{T}(n/2) + c_2 & \text{altrimenti} \end{cases}$$

che in base al Master Theorem ha soluzione $T(n) = \Theta(n)$.

Esercizio 3. Si consideri un array $A[1..n]$ composto da $n>=1$ interi distinti ordinati in senso crescente ($A[1] < A[2] < \ldots < A[n]$). Scrivere un algoritmo efficiente che, dato in input l'array A, determina un indice i, se esiste, tale che $A[i] = i$. Nel caso esistano più indici che soddisfano la relazione precedente, è sufficiente restituirne uno qualsiasi. Determinare il costo computazionale dell'algoritmo.

Soluzione. È possibile utilizzare il seguente algoritmo ricorsivo, molto simile a quello della ricerca binaria (i e j rappresentano rispettivamente gli indici estremi del sottovettore $A[i..j]$ in cui effettuare la ricerca, all'inizio la funzione va invocata con $i = 1, j = n$):

```
PuntoFisso( integer A[1..n], integer i, integer j ) → integer
  if (i > j) then
      return -1;
  else
      integer m ← Floor( (i + j) / 2 );
      if ( A[m] = m ) then
          return m;
      elseif ( A[m] > m ) then
          return PuntoFisso(A, i, m - 1);
      else
          return PuntoFisso(A, m + 1, j);
      endif
  endif
```

L'algoritmo proposto è una semplice variante dell'algoritmo di ricerca binaria, e ha lo stesso costo computazionale $T(n) = \Theta(\log n)$.

Esercizio 4. Consideriamo un insieme di n variabili $x_1, \ldots x_n$. Sono dati un insieme di vincoli di uguaglianza della forma "$x_i = x_j$", e un altro insieme di vincoli di disuguaglianza della forma "$x_i \neq x_j$". Il problema consiste nel capire se tutti i vincoli possono essere soddisfatti. Ad esempio, considerando quattro variabili x_1, x_2, x_3, x_4 soggette ai vincoli seguenti

$$x_1 = x_2;$$
$$x_2 = x_3;$$
$$x_3 = x_4;$$
$$x_1 \neq x_4$$

risulta che in questo caso i vincoli non sono soddisfacibili. Descrivere a parole un algoritmo efficiente che, dati in input il numero n di variabili e le liste dei vincoli di uguaglianza e disuguaglianza, restituisce *true* se e solo se i vincoli sono soddisfacibili.

Soluzione. Si costruisce un Mfset con n elementi che rappresentano le variabili. Per ogni vincolo di uguaglianza di fa la merge delle variabili corrispondenti. Poi si verifica che i vincoli di disuguaglianza siano tra variabili in componenti diverse. La complessità è $O(n \log n)$ con quick union by rank.

Esercizio 5. Si consideri un array $A[1..n]$ contenente valori reali ordinati in senso non decrescente; l'array può contenere valori duplicati. Scrivere un algoritmo ricorsivo di tipo divide-et-impera che, dato A e due valori reali $low < up$, calcola quanti valori di A appartengono all'intervallo $[low, up]$. Determinare il costo computazionale dell'algoritmo proposto.

Soluzione.

```
CONTA( real A[1..n], real low, real up, integer i, integer j ) → integer
  if ( i > j ) then
      return 0;
  elseif ( A[i] > up or A[j] < low ) then
      return 0;
  elseif ( i = j ) then
      return 1;
  else
      integer m ← FLOOR( ( i + j ) / 2 );
      return CONTA(A, low, up, i, m) + CONTA(A, low, up, m+1, j);
  endif
```

L'algoritmo viene invocato con CONTAINTERVALLO(A, low, up, 1, n). Il suo costo $T(n)$ soddisfa la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} c_1 & \text{se } n \le 1 \\ 2\,T(n/2) + c_2 & \text{altrimenti} \end{cases}$$

L'applicazione del Master Theorem porta alla soluzione $T(n) = \Theta(n)$. Questo caso pessimo viene effettivamente raggiunto, ad esempio quando tutto l'array fa parte dell'intervallo $[low, up]$. In questo caso infatti sommiamo sempre 1, e per ottenere n come risultato serve un numero di chiamate lineare.

La funzione qui sotto, estremamente simile, ha invece costo $T(n) = \Theta(\log n)$ nel caso peggiore. Per dimostrarlo si consideri il livello n-esimo di chiamate ricorsive: solo gli intervalli (1 o 2) che contengono low e high vanno considerati, per cui ad ogni livello abbiamo al massimo 2 chiamate ricorsive, per un tempo di elaborazione totale del livello costante. Visto che ad ogni chiamata ricorsiva la dimensione degli intervalli dimezza abbiamo un numero logaritmico di livelli, da cui il costo $T(n) = \Theta(\log n)$.

```
CONTA2( real A[1..n], real low, real up, integer i, integer j ) → integer
  if ( i > j ) then
      return 0;
  elseif ( A[i] > up or A[j] < low ) then
      return 0;
  elseif ( A[i] ≥ low and A[j] ≤ high ) then
      return j-i+1;
  else
      integer m ← FLOOR( ( i + j ) / 2 );
      return CONTA2(A, low, up, i, m) + CONTA2(A, low, up, m+1, j);
  endif
```