DOMANDA: Illustrare l'Algoritmo ricorsivo di somma di un array, approccio Divide et impera (ASOMRICDI)

RISPOSTA:
L'algoritmo ricorsivo di somma di un array, approccio Divide et impera, calcola la somma degli elementi di un array di dati. I dati possono essere numeri interi, numeri reali o qualsiasi altro insieme di dati su cui sia definita una operazione di somma.
L'ASOMRICDI ha come dati di input un array e il suo size, e ha come dato di output un dato scalare, che è il valore della somma di tutti gli elementi dell'array.
L'ASOMRICDI non fa uso di altri array o strutture dati.
L'ASOMRICDI è basato sull'applicazione dell'approccio Divide et impera al problema della somma degli elementi di un array.
L'approccio Divide et impera è una metodologia di progetto di algoritmi in cui la soluzione del problema viene determinata suddividendo ricorsivamente il problema in due istanze di dimensione dimezzata (fase Divide) e poi esprimendo la soluzione combinando opportunamente le due soluzioni delle istanze dimezzate (fase Impera).
La tecnica di programmazione ricorsiva, basata sull'autoattivazione di una function, consente di descrivere in modo naturale un algoritmo basato sull'approccio Divide et impera.
Nel caso del problema della somma degli elementi di un array, la soluzione del problema, cioè la somma, viene calcolata come la somma tra la somma degli elementi della porzione "di sinistra" e la somma degli elementi della porzione "di destra" in cui si può suddividere l'array.
Ogni autoattivazione suddivide una data istanza in due istanze di dimensioni dimezzate, crea un processo che risolve l'istanza del problema della somma sulla "porzione di sinistra" e un processo che risolve l'istanza del problema della somma sulla "porzione di destra" della porzione ricevuta in input e quindi restituisce la somma dei valori ottenuti dai due processi. A ogni autoattivazione, i due processi creati agiscono ognuno su una "porzione" di size dimezzato rispetto al precedente, fino ad arrivare al size 1, che è il size dell'istanza banale del problema della somma.
Diamo una rapida analisi della meccanica dell' ASOMRICDI. Alla prima attivazione si crea un processo che agisce sull'intero array A; esso crea a sua volta un processo che effettua la somma sulla porzione di sinistra dell'array A e un processo che effettua la somma sulla porzione di destra dell'array A; i valori restituiti da questi due processi sono sommati e restituiti. Le porzioni sono individuate dall'indirizzo base e dal size.
Il caso base della ricorsione è costituito dalla porzione di size 1, e in tal caso si deve restituire il valore dell'unico elemento della porzione.
Se l'istanza che il processo sta risolvendo non rientra nel caso base, allora si deve calcolare la posizione "centrale" della porzione; poi si deve autoattivare sulla porzione "di sinistra", cioè la porzione che ha lo stesso indirizzo base della porzione in input e size dimezzato, e autoattivare sulla porzione "di destra", cioè la porzione che ha come indirizzo base l'indirizzo della posizione immediatamente a destra della posizione centrale e size dimezzato; i due risultati di tali auto attivazioni devono essere sommati e restituiti al processo chiamante.
Un esempio aiuta a capire la dinamica dell'ASOMRICDI. Si consideri un array A di n=8 elementi interi.
Poiché il size 8 è diverso da 1, non siamo nel caso base.

41	31	27	28	52	36	11	13
0	1	2	mediano=3	4	5	6	7

Quindi, l'algoritmo deve continuare, cioè deve effettuare una prima autoattivazione che crea un processo che agisce sulla porzione di inizio 0 e size 4 e una seconda autoattivazione che crea un processo che agisce sulla porzione di inizio 4 e size 4; i valori restituiti da tali autoattivazioni devono essere sommati.
I due nuovi processi "vedono" rispettivamente le seguenti porzioni di array (si faccia attenzione agli indici locali)

Processo 1

41	31	27	28
0	1	2	3

Processo 2

52	36	11	13
0	1	2	3

Poiché il size 4 è diverso da 1, non siamo nella prima situazione del caso base. Il primo processo calcola la posizione centrale è (0+3)/2, cioè 1, e attiva due nuovi processi che agiscono rispettivamente sulla porzione 0..1 e sulla porzione 2..3 della propria porzione ricevuta. Analogamente, Il secondo processo calcola la posizione centrale è (0+3)/2, cioè 1, e attiva due nuovi processi che agiscono rispettivamente sulla porzione 0..1 e sulla porzione 2..3 della propria porzione ricevuta.
I quattro nuovi processi "vedono" rispettivamente le seguenti porzioni di array (si faccia attenzione agli indici locali)

Processo 1.1

41	31
0	1

Processo 1.2

27	28
0	1

Processo 2.1

52	36
0	1

Processo 2.2

11	13
0	1

Poiché il size 2 è diverso da 1, non siamo nella prima situazione del caso base. Il primo processo calcola la posizione centrale è (0+1)/2, cioè 0, e attiva due nuovi processi che agiscono rispettivamente sulla porzione 0 e sulla porzione 1 della propria porzione ricevuta. Analogamente, il secondo, il terzo e quarto processo calcolano la posizione centrale è (0+1)/2, cioè 0, e attivano ognuno due nuovi processi che agiscono rispettivamente sulla porzione 0 e sulla porzione 1 della propria porzione ricevuta.
Gli otto nuovi processi "vedono" rispettivamente le seguenti porzioni di array (si faccia attenzione agli indici locali)

Processo 1.1.1

41
0

Processo 1.1.2

31
0

Processo 1.2.1

27
0

Processo 1.2.2

28
0

Processo 2.1.1

52
0

Processo 2.1.2

36
0

Processo 2.2.1

11
0

Processo 2.2.2

13
0

Poiché il size è 1, siamo nella prima situazione del caso base. Quindi tutti gli otto processi restituiscono il valore dell'unico elemento della propria porzione ricevuta.
A questo punto i processi sospesi 1.1, 1.2, 2.1, 2.1 sommano i valori ricevuti e restituiscono tali somme ai processi sospesi 1 e 2, che a loro volta sommano i valori ricevuti e restituiscono tali somme al processo sospeso iniziale, che effettua la somma e restituisce la soluzione del problema.

Questa è l'implementazione in C dell'ASOMRICDI, per un array di tipo `int`:

```
int somma_a_ricDI(int a[],int n)
{
 int mediano;
/* soluzione del caso base */
 if (n == 1)
     return a[0];
   else
    /* autoattivazioni */
     {
      mediano = (n-1)/2;
      return somma_a_ricDI(a,mediano+1) + ...
             somma_a_ricDI(a+mediano+1,n-mediano-1);
      }
}
```

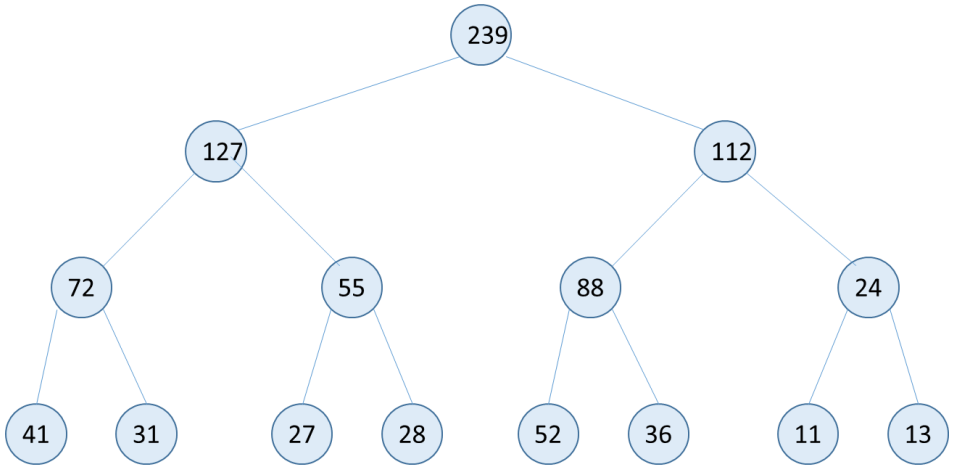
Un breve commento al codice. La function ha la classica struttura delle function ricorsive, cioè un `if-the-else` che distingue il caso base dalle istanze non banali.
Il blocco `else` gestisce l'autoattivazione. La prima autoattivazione agisce sulla semi-porzione di sinistra, che ha lo stesso indirizzo base della porzione ricevuta in input e size uguale al valore dell'indice **`mediano+1`**; la seconda autoattivazione agisce sulla semi-porzione di destra, che ha come indirizzo base quello della posizione immediatamente a destra della posizione centrale della porzione ricevuta in input e size uguale al valore **`n-mediano-1`**. La soluzione dell'istanza è calcolata sommando i valori restituiti dalle due autoattivazioni ed è restituita al chiamante.
Poiché una espressione è eseguita da "sinistra verso destra", l'autoattivazione sulla porzione di sinistra (che appare come primo addendo nell'espressione **`somma_a_ricDI(a,mediano+1) + somma_a_ricDI(a+mediano+1,n-mediano-1)`**) è eseguita prima dell'autoattivazione sulla porzione di destra. Quindi l'esecuzione dell'ASOMRICDI dà luogo alla seguente sequenza di attivazione/sospensione dei processi nello stack (per semplicità si usano gli indici globali; ogni colonna "fotografa" lo stack dopo ogni autoattivazione):

		P(0)	P(1)		P(2)	P(3)				
		P(0..1)	P(0..1)		P(2..3)	P(2..3)				P(4..5)
	P(0..3)	P(0..3)	P(0..3)	P(0..3)	P(0..3)	P(0..3)	P(0..3)		P(4..7)	P(4..7)
P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)

P(4)	P(5)		P(6)	P(7)		
P(4..5)	P(4..5)		P(6..7)	P(6..7)		
P(4..7)	P(4..7)	P(4..7)	P(4..7)	P(4..7)	P(4..7)	
P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)	P(0..7)

Analizziamo la complessità dell'ASOMRICDI. La complessità di spazio è n, perché l'algoritmo opera completamente "in place".
Passiamo alla complessità di tempo. L'operazione dominante è l'operazione di somma tra due numeri.
Il modo più semplice per determinare la complessità di tempo dell'ASOMRICDI consiste nell'esaminare il cosiddetto albero binario delle operazioni associato all'esecuzione dell'algoritmo per un dato problema.
I nodi foglia di tale albero sono gli elementi dell'array, i nodi interni contengono i valori delle somme delle varie porzioni in cui l'array viene suddiviso durante le successive autoattivazioni, e il livello in cui appaiono nell'albero corrisponde al livello di autoattivazione. In altre parole, la radice dell'albero contiene la soluzione del problema, ottenuta sommando dal basso verso l'alto le varie coppie di numeri.

41	31	27	28	52	36	11	13
0	1	2	3	4	5	6	7



E' chiaro esaminando l'albero che per ottenere il valore di ciascuno dei nodi interni (cioè tutti i nodi escluse le foglie) è necessario effettuare esattamente una somma; quindi il numero totale di somme è uguale al numero dei nodi interni. Poiché il numero delle foglie è n, in base a una delle proprietà degli alberi binari (completi), si ha che il numero dei nodi interni è n-1.
In conclusione, la complessità di tempo dell'ASOMRICDI è $T(n) = n - 1$ somme.
Si noti che si tratta di una complessità assoluta, cioè indipendente dal valore degli elementi dell'array, e che tale complessità è identica a quella del classico algoritmo non ricorsivo di somma degli elementi di un array basato sull'approccio incrementale.