

## 008AA – ALGORITMICA E LABORATORIO

Verifica dell'8 aprile 2010

Cognome Nome:

N. Matricola:

Corso: A B

**Esercizio 1.** (5+4 punti) Data la seguente funzione ricorsiva `foo`, trovare la corrispondente relazione di ricorrenza e risolverla utilizzando il teorema principale.

```
foo( n ){
  if ( n < 5 ) return 1;
  if ( n <= 200 ) {
    temp = 0;
    for ( i = 1; i <= n; i = i + 1 )
      for ( j = 1; j <= n; j = j + 1 )
        for ( k = 1; k <= n; k = k + 1 )
          for ( m = 1; m <= n; m = m + 1 )
            tmp = tmp + i * j * k * m;
    return tmp + foo( n/5 ) + foo( n/2 );
  }
  tmp = 0;
  for ( i = 1; i*i < n; i = i + 1 )
    tmp = tmp + i*i;
  return tmp * foo( n/2 ) * foo( n/2 ) * foo( n/2 ) * foo( n/2 );
}
```

La relazione di ricorrenza risulta  $T(n) = O(1)$  per  $n < 5$ ;  $T(n) = O(n^4) = O(1)$  per  $5 \leq n \leq 200$ ; e  $T(n) = 4T(n/2) + \Theta(\sqrt{n})$  per  $n > 200$ .

Soluzione 1: Si nota che  $n^{\log_b a} = n^{\log_2 4} = n^2$ , e  $f(n) = \Theta(\sqrt{n})$ . Siamo dunque nel primo caso del teorema principale (secondo il libro CLR) in quanto  $f(n) = O(n^{2-\epsilon})$  per un certo  $\epsilon > 0$  e precisamente basta prendere  $\epsilon \in ]0, 3/2]$ .

Soluzione 2: posto  $T(n) \leq 4T(n/2) + c\sqrt{n}$  per una costante  $c > 0$ , vale  $\alpha = 4$ ,  $\beta = 2$  e  $f(n) = c\sqrt{n}$ . Esiste una costante  $\gamma > 0$  tale che  $\alpha f(n/\beta) = \gamma f(n)$ , ossia  $\gamma = \frac{\alpha f(n/\beta)}{f(n)} = 4 \frac{c\sqrt{n/2}}{c\sqrt{n}} = 4/\sqrt{2} > 1$ . Siamo quindi nel caso in cui la soluzione è  $T(n) = O(n^{\log_\beta \alpha}) = O(n^{\log_2 4}) = O(n^2)$ .

Quindi la soluzione della relazione di ricorrenza risulta  $T(n) = O(n^2)$ .

Cognome Nome:

N.Matr:

**Esercizio 2.** (6+3 punti) Sia dato un array  $A$  di  $n$  interi distinti e positivi. Progettare un algoritmo ricorsivo basato sulla tecnica *divide et impera* che conta il numero di elementi che sono minimi relativi del vettore  $A$  (escludendo dal conteggio  $A[0]$  e  $A[n-1]$ ). Si definisce minimo relativo un elemento che è minore dei suoi adiacenti. Per esempio,  $A = [3, 4, 2, 8, 7, 10, 12, 15, 14, 20]$  ha tre minimi relativi che sono  $\{2, 7, 14\}$ .

Si valuti inoltre la complessità in tempo e spazio al caso pessimo dell'algoritmo proposto.

*Suggerimento: Prestare attenzione agli estremi dei sotto-vettori nelle chiamate ricorsive.*

Soluzione: Basta invocare la seguente funzione ricorsiva sulla parte interna di  $A[0, n-1]$ , ossia  $\text{Conta}(A, 1, n-2)$ , avendo prima verificato che  $n > 2$ .

```
Conta(B,i,j){
  if (i == j) {
    if (B[i] < B[i-1]) && (B[i] < B[i+1]) )
      return 1;
    else
      return 0;
  } else {
    c = (i+j)/2; // parte intera inferiore
    return Conta(B,i,c) + Conta(B,c+1,j);
  }
}
```

In alternativa, il caso base in C può anche essere scritto come

```
Conta(B,i,j){
  if (i == j) {
    return (B[i] < B[i-1] && B[i] < B[i+1]) ? 1 : 0;
  } else {
    c = (i+j)/2; // parte intera inferiore
    return Conta(B,i,c) + Conta(B,c+1,j);
  }
}
```

Il costo dell'algoritmo è  $T(n) = 2T(n/2) + O(1) = O(n)$ . Lo spazio aggiuntivo è  $O(\log n)$  perchè questa è la profondità delle chiamate ricorsive.

Cognome Nome:

N.Matr:

**Esercizio 3.** (*4+1 punti*) Una sequenza *bilanciata* di parentesi può essere definita ricorsivamente come segue: la sequenza vuota è bilanciata; se  $S$  e  $T$  sono sequenze bilanciate, allora  $(S)T$  e  $S(T)$  sono sequenze bilanciate. Per esempio,  $()(())(())$  e  $((())(())(())$  sono sequenze bilanciate, mentre  $((())(())(())$  e  $)()((())$  non lo sono.

Progettare un algoritmo efficiente che, presa in ingresso una sequenza di parentesi, calcola il minimo numero di parentesi da aggiungere per renderla bilanciata. Per esempio, la risposta è 0 per  $()(())(())$  e  $((())(())(())$  in quanto sono già bilanciate, mentre è 3 per  $((())(())(())$  e 2 per  $)()((())$ . L'algoritmo deve essere spiegato anche a parole.

Valutare inoltre la complessità in tempo al caso peggio dell'algoritmo proposto.

Soluzione: Si assuma che la sequenza di parentesi sia memorizzata entro un vettore  $A[0, n - 1]$ .

Occorre prestare attenzione al fatto che una sequenza può avere conteggio 0, ma non essere bilanciata, quindi non è sufficiente contare le parentesi aperte e chiuse e confrontare i due valori. Si consideri per esempio la sequenza  $)()$  oppure la sequenza  $)()((())$  indicata nel testo.

L'algoritmo proposto scandisce  $A$  mantenendo un *contatore*  $c$  che viene incrementato di 1 ogni qualvolta si incontra una parentesi aperta, e viene decrementato di 1 ogni qualvolta si incontra una parentesi chiusa. Il valore del contatore, se negativo, indica quante parentesi chiuse non sono bilanciate da altrettante parentesi aperte corrispondenti. La precedente sequenza ha  $c = -1$  in prima posizione, e poi  $c = 0$ ; infatti in prima posizione abbiamo una parentesi chiusa non bilanciata.

Possiamo affermare che una sequenza è bilanciata se il contatore è sempre non-negativo e il suo valore finale è nullo. Abbiamo già osservato che: se esistono parentesi chiuse non bilanciate allora il contatore  $c$  sarà negativo quando le incontra. Quindi basta prendere il minimo valore assunto da  $c$  nella scansione di  $A$  (sia questo  $m$ ) per calcolare quante parentesi chiuse necessitano di essere bilanciate con l'aggiunta di altrettante parentesi aperte.

D'altra parte, se aggiungessimo queste  $m$  parentesi alla sinistra di  $A$ , avremmo che il contatore alla fine varrebbe  $|m| + c$ , che è sicuramente  $\geq 0$  (visto che le parentesi chiuse sono tutte state bilanciate). Se  $> 0$ , indicherebbe quante parentesi aperte non sono bilanciate. Ne consegue quindi che dobbiamo aggiungere altrettante parentesi chiuse per bilanciare la sequenza. In totale dunque dobbiamo aggiungere  $2|m| + c$  parentesi.

```
Bilancia(A,n)
{
  c=0; m = 0;
  for(i=0; i<n; i++) {
    if (A[i] = '(') { c++; } else { c--; }
    if (c < m) m = c;
  }
  return 2 * abs(m) + c;
}
```

In alternativa, possiamo usare un contatore **npar** che incrementiamo ogni volta che il contatore  $c$  diventa negativo, forzando poi  $c$  a zero (perché abbiamo concettualmente ribilanciato la parentesi chiusa). Le parentesi aperte che eventualmente rimangono dopo essere usciti dal ciclo ( $c > 0$ ) vanno messe nel conteggio di **npar** perché sono da ribilanciare.

```
Bilancia(A,n)
{
  c=0; npar = 0;
  for(i=0; i<n; i++) {
    if (A[i] = '(') c++; else c--;
    if (c < 0) { npar++; c = 0; }
  }
  return npar + c;
}
```

Cognome Nome:

N.Matr:

**Esercizio 4.** (*3+4+2 punti*) Scrivere la formula ricorsiva di programmazione dinamica per risolvere il problema della sottosequenza comune più lunga (*LCS*). Applicare tale formula alle due sequenze  $X = \text{DAEABCDE}$  e  $Y = \text{ZADBCDYE}$ , costruendo la relativa tabella di programmazione dinamica. Infine, mostrare sulla tabella come effettuare il percorso a ritroso per trovare una sottosequenza comune più lunga, ed indicarla.

Si veda il libro di testo.