Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Prof. Ghezzi, Lanzi e Morzenti
Prova di recupero
17 Settembre 2004

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili. **Barrare le caselle relative alle parti recuperate. Non consegnare altri fogli.**

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

RECUPERO ☐
I PARTE
Es. 1,2,3

RECUPERO ☐
II PARTE
ES. 4,5,6

Spazio riservato ai docenti

Esercizio 1. Parte I. Considerando il seguente programma:

```
int k=2, p=4;          //@@1
int func1()
{
    int k=6;
    int z =5;          //@@2
    int func2()
    {
        while (k>=p)   //@@3
        {
           k = k - 2;
           p = p + 2;
        }
        return func1();
    }

    while ( k < p )    //@@4
    {
        p = p - 4;
    }
    return func2();
}

void main()
{
   int z;
   z = 2 * p;          //@@5
   k = 5 * k + func1()*3 + z;
   print k;
}
```

Quesito 1. Specificare le copie distanza-offset per le variabili utilizzate nelle istruzioni identificate dale etichette @@.

Soluzione

@@1 k=<0,0> p=<0,1>
@@2 z=<0,4>
@@3 k=<1,3> p=<2,1>
@@4 k=<0,3> p=<1,1>
@@5 z=<0,3> p=<1,1>

Esercizio 1. Parte I. (continua).
Quesito 2. Schizzare lo stato della macchina SIMPLESEM per il programma subito dopo la **terza** chiamata della funzione **func1()** quando viene eseguito l'assegnamento "int k=6".

	0	Current	29
	1	Free	35
	2	k	
	3	p	
main()	4	RP	
	5	DL	
	6	SL	2
	7	z	8
	8	RV	
func1()	9	RP	
	10	DL	4
	11	SL	2
	12	k	
	13	z	
	14	RV	
func2()	15	RP	
	16	DL	9
	17	SL	9
	18	RV	
func1()	19	RP	
	20	DL	15
	21	SL	2
	22	k	
	23	z	
	24	RV	
func2()	25	RP	
	26	DL	19
	27	SL	19
	28	RV	
func1()	29	RP	
	30	DL	25
	31	SL	2
	32	k	
	33	z	
	34	RV	

Esercizio 2. Parte I. Si realizzi, tratteggiando due programmi in Java e in Ada, il sistema concorrente descritto nel seguito. Esso è costituito da due moduli M1 ed M2, e da due buffer B0 e B1, identici, ognuno dei quali può memorizzare un singolo numero intero.
Il comportamento di M1 ed M2 è descritto come segue: M1 ripete indefinitamente le seguenti operazioni: legge da tastiera un numero intero n e se n è pari lo scrive in B0 mentre se n è dispari lo scrive in B1; similmente M2 ripete indefinitamente le seguenti operazioni: legge da tastiera un numero intero n e se n è pari legge il valore memorizzato in B0 e lo scrive a video, mentre se n è dispari fa la stessa cosa su B1.
L'accesso a ognuno dei buffer è in mutua esclusione; M1 rimane bloccato se il buffer a cui accede è pieno, M2 se è vuoto; si assuma inoltre che inizialmente entrambi i buffer siano vuoti.

Risposta. Vedi pagina successiva.

Descrivere brevemente in quali casi può accadere che M1 ed M2 rimangano definitivamente bloccati senza possibilità di procedere. Illustrare con un semplice esempio.

Risposta. M1 ed M2 rimangono bloccati quando uno dei due legge ripetutamente un numero pari e l'altro legge ripetutamente un numero dispari, e quindi M1 rimane bloccato nel tentativo di scrivere in un buffer pieno, mentre M2 rimane bloccato nel tentativo di leggere da un buffer vuoto. Per esempio se i primi due numeri letti da M1 sono pari e il primo numero letto da M2 è dispari, M1 ed M2 rimangono bloccati.

Indicare delle ipotesi, il più possibile semplici, sotto le quali si può avere le certezza che non si verifichi tale situazione di blocco tra M1 ed M2.

Risposta. L'esecuzione di M1 ed M2 procede senza blocchi se, per esempio, la loro velocità di esecuzione è tale per cui essi si alternano nella lettura dei numeri da tastiera, iniziando da M1, e la lettura di un numero pari o rispettivamente dispari da parte di M1 è sempre seguita dalla lettura di un numero pure pari o rispettivamente dispari da parte di M2.

Esercizio 2. Parte I.

Soluzione Ada.

```
With TEXT_IO, Ada.Integer_text_IO;
use Ada.Integer_text_IO, TEXT_IO;
procedure M1M2 is

task type BUF is
entry PUT (X: in INTEGER);
entry GET (X: out INTEGER);
end BUF;

B0, B1: BUF;
task M1;
task M2;

task body BUF is
  I: INTEGER;
  FULL : BOOLEAN := FALSE;
begin
  loop
    select
      when not FULL =>
          accept PUT(X: in INTEGER) do
            Q := X;
           FULL := TRUE;
        end PUT;
     or
      when FULL =>
        accept GET(X: out INTEGER) do
           X := Q;
           FULL := FALSE;
        end GET;
    end select;
  end loop;
end BUF;

task body M1 is
    n : INTEGER;
  begin
    loop
      TEXT_IO.PUT("n? >");
      Ada.Integer_Text_IO.GET(n);
      if n REM  2 = 0
          then B0.PUT(n);
          else B1.PUT(n);
      end if;
    end loop;
end M1;

task body M2 is
    n, b : INTEGER;
  begin
    loop
       TEXT_IO.PUT("n? >");
       Ada.Integer_Text_IO.GET(n);
       if n REM  2 = 0
          then B0.GET(b);
          else B1.GET(b);
       end if;
       Ada.Integer_Text_IO.PUT(b);
       TEXT_IO.PUT_LINE(" ");
    end loop;
end M2;

begin -- corpo
   null;
end M1M2;
```

Soluzione Java.

```
public class Buf {
    private int q;
    private boolean full;
    Buf() {  full = false;  }
    public synchronized void put (int item) {
        while (full)
           try { wait(); }  catch (InterruptedException e) { }
       q = item;
      full = true;
      notify();
   }
   public synchronized int get () {
      while (!full)
          try { wait(); }
          catch (InterruptedException e) { }
      full = false;
      notify();
      return q;
   }
}

public class M1 extends Thread {
  private Buf b0, b1;
  M1 (Buf br0, Buf br1) {
    b0 = br0;
    b1 = br1;
  }

  public void run() {
      int n;
      while (true) {
          n = System.in.ReadInt();
         if (n % 2 == 0)
             b0.put(n);
          else b1.put(n);
      }
  }
}

public class M2 extends Thread {
  private Buf b0, b1;
 M2 (Buf br0, Buf br1) {
    b0 = br0;
    b1 = br1;
  }
  public void run() {
      int n, b;
      while(true) {
          n = System.in.ReadInt();
         if (n % 2 == 0)
               b = b0.get();
          else b = b1.get();
          System.out.println("Buff " + b);
  }
}
}

public class M1M2 {
    public static void main (String [] args){
        Buf b0 = new Buf();
        Buf b1 = new Buf();
        M1 m1 = new M1(b0, b1);
        M2 m2 = new M2(b0, b1);
        m1.start();
        m2.start();
    }
}
```

Esercizio 3. Parte I. Si consideri il seguente frammento di codice:

```
int i = 0;
int a[2] = {1,1};

void foo(int x)
{
  a[0] := 6;
  i := 1;
  x:= x + 3;
}

main()
{
  a[1]:= 2;
  foo(a[i]);
}
```

Scrivere quali sono i valori di a[0] e a[1] al termine dell'esecuzione nel caso in cui i parametri siano passati per valore, per valore-risultato, per indirizzo e per nome.

Soluzione

- per valore, a[]={6,2};
- per valore-risultato a[]={4,2};
- per indirizzo a[] = {9,2};
- per nome a[] = {6,5}.

Esercizio 4. Parte II. Si consideri una hash table con *n* slot, numerati da 0 a *n-1*, su cui si deve mappare una chiave *K*. Per ciascuna delle seguenti definizione della funzione *h(k)*

1. *h(k) = k/n* (con *k* e *n* sono interi)
2. *h(k) = 1*
3. *h(k) = (k+Random(n)) mod n*, dove *Random(n)* restituisce un numero intero fra *0* e *n-1*
4. *h(k) = k mod n*, con *n* numero primo.

dire se la funzione *h(k)* è una funzione di hash accettabile, ovvero se l'inserimento e la ricerca possono funzionare correttamente utilizzando tale funzione. In caso affermativo dire se la funzione è una buona funzione di hash. Motivare brevemente tutte le risposte.

Soluzione

1. non accettabile, k/n può restituire un indice al di fuori della tabella
2. accettabile, ma non è una buona funzione di hash: tutte le chiavi sono mappate nella stessa posizione provocando collisioni per ogni inserimento
3. non accettabile, la posizione determinata dalla funzione di hash è casuale; risulta quindi impossibile ritrovare l'elemento.
4. accettabile, dato che n è primo h(k) può essere considerata una buona funzione di hash

Esercizio 5. Parte II. Si consideri l'algoritmo per effettuare la moltiplicazione di due numeri decimali composti da n cifre che viene insegnato alle scuole elementari. Si indichi la sua complessità asintotica in funzione di n, assegnando costo unitario alla moltiplicazione tra due cifre decimali. Giustificare brevemente la risposta.

Risposta. La complessità è quadratica nel numero delle cifre, perché occorre fare per n volte n moltiplicazioni tra due cifre.

Si vuole adottare un approccio *divide et impera* per effettuare la moltiplicazione, supponendo per semplicità che n sia una potenza di 2. Si suddivide quindi ognuno dei due fattori X e Y nella parte "alta" e in quella "bassa" della rappresentazione decimale

$X = X_1 \cdot 10^{n/2} + X_0$
$Y = Y_1 \cdot 10^{n/2} + Y_0$

La moltiplicazione $X \cdot Y$ può quindi essere calcolata come

$X \cdot Y = (X_1 \cdot 10^{n/2} + X_0) \cdot (Y_1 \cdot 10^{n/2} + Y_0) = X_1 \cdot Y_1 \cdot 10^n + (X_1 \cdot Y_0 + X_0 \cdot Y_1) \cdot 10^{n/2} + X_0 \cdot Y_0$

Si noti che moltiplicare un numero per 10^k equivale a eseguire uno *shift* delle sue cifre di k posizioni a sinistra e si assuma che tale operazione abbia costo $\Theta(k)$, e che tale complessità sia anche quella dell'operazione di somma tra due numeri composti da k cifre. Si determini la complessità asintotica di un algoritmo divide et impera per la moltiplicazione tra due numeri basato sulla precedente relazione. Si ottiene un guadagno rispetto alla complessità dell'algoritmo "delle elementari"?

Risposta. Occorre fare 4 moltiplicazioni di numeri di $n/2$ cifre, più un certo numero (fissato) di somme e *shift*; vale quindi la relazione alle ricorrenze

$T(n) = 4 \cdot T(n/2) + \Theta(n)$ per $n>1$; $T(1) = 1$

Che ha come soluzione $T(n) = \Theta(n^{\log 4}) = \Theta(n^2)$, senza alcun guadagno.

Si osservi che valgono le seguenti relazioni: definendo P_1, P_2 e P_3 nel modo seguente

$P_1 = (X_1 + X_0) \cdot (Y_1 + Y_0)$
$P_2 = X_1 \cdot Y_1$
$P_3 = X_0 \cdot Y_0$

si ha che

$(X_1 \cdot Y_0 + X_0 \cdot Y_1) = P_1 - P_2 - P_3$

Si sfrutti tale osservazione per tratteggiare un algoritmo divide et impera per la moltiplicazione di due numeri che effettui un numero inferiore a 4 di moltiplicazioni tra numeri di $n/2$ cifre, e se ne determini la complessità asintotica.

Risposta. Si ha evidentemente

$X \cdot Y = P_1 \cdot 10^n + (P_1 - P_2 - P_3) \cdot 10^{n/2} + P_3$
quindi bastano tre moltiplicazioni più un numero fissato di somme e shift, e l'equazione alle ricorrenze diventa

$T(n) = 3 \cdot T(n/2) + \Theta(n)$ per $n>1$; $T(1) = 1$

che porta alla complessità $T(n) = \Theta(n^{\log 3}) = \Theta(n^{1.59})$

Esercizio 6. Parte II. Si considerino le seguenti quattro versioni di BubbleSort, che usano un metodo scambia definito a parte (e qui ignorato) per scambiare tra di loro i valori memorizzati in due posizioni di un array.

```
static void bubsort1(Elem[] array) {
    for (int i=0; i<array.length-1; i++)
        for (int k=0; k<array.length-2; k++)
            if (array[k]>array[k+1])
                scambia(k, k+1)
}
```

```
static void bubsort2(Elem[] array) {
    for (int i=0; i<array.length-1; i++)
        for (int k=array.length-2; k>=i; k--)
            if (array[k]<array[k+1])
                scambia(k, k+1)
}
```

```
static void bubsort3(Elem[] array) {
    swap=true;
    while (swap) {
        swap=false;
        for (int k=0; k<array.length-1; k++)
            if (array[k]>array[k+1])   {
                scambia(k, k+1);
                swap=true;
            }

}
```

```
static void bubsort4(Elem[] array) {
    for (int i=array.length-1; i>=0 i--)
        for (int k=array.length-2; k>=0; k--)
            if (array[k]>array[k+1])
                scambia(k, k+1)
}
```

Quesito 1: Si confrontino i valori di complessità nel caso medio, pessimo e ottimo riempiendo questa tabella. Motivare brevemente tutte le risposte.

bubsort1
caso ottimo $\Theta(n^2)$
caso medio $\Theta(n^2)$
caso pessimo $\Theta(n^2)$

Sia nel caso ottimo (array già ordinato in partenza) sia nel caso pessimo (array ordinato in senso inverso) il corpo del ciclo interno viene eseguito n^2 volte. Chiaramente il caso medio, essendo compreso tra i due estremi rappresentati da caso ottimo e pessimo avrà una complessità $\Theta(n^2)$

bubsort2
caso ottimo $\Theta(n^2)$
caso medio $\Theta(n^2)$
caso pessimo $\Theta(n^2)$

Di nuovo, il caso ottimo e il caso pessimo hanno la stessa complessità, perché i cicli vengono eseguiti lo stesso numero di volte indipendentemente dalla disposizione dei numeri nelle celle. Infatti:

$$\sum_{i=0}^{n-2}\sum_{j=i}^{n-3} c = \sum_{i=0}^{n-2} c(n-1-i) =$$

$$= cn(n-1) - c(n-1) - c\frac{(n-2)(n-1)}{2} - cn = \Theta(n^2)$$

Bubsort3
caso ottimo $\Theta(n)$
caso medio $\Theta(n^2)$
caso pessimo $\Theta(n^2)$

In questa versione se l'array di partenza è già ordinato (caso ottimo), il ciclo interno viene eseguito n volte mentre quello esterno viene eseguito una volta sola (swap non diventa mai vero) e quindi avremo una complessità lineare. Nel caso pessimo invece dovremo comunque eseguire n^2 scambi. Per quanto riguarda il caso medio, la complessità è ancora $\Theta(n^2)$ perché avremo in media $\frac{n^2}{2}$ scambi.

bubsort4
caso ottimo $\Theta(n^2)$
caso medio $\Theta(n^2)$
caso pessimo $\Theta(n^2)$

In questo caso valgono le considerazioni esposte per Bubsort1, in quanto l'algoritmo è sostanzialmente identico a meno dell'ordine seguito per scandire l'array (dall'ultima cella verso la prima in questo caso)

Quesito 2. Quali versioni del bubblesort risultano stabili?

Un algoritmo di sortine viene definito stabile se non viene alterato l'ordine di due elementi identici. Un esempio di algoritmo di sortine NON stabile è il QuickSort perché se viene scelto come pivot uno dei due elementi identici, l'altro può venire spostato a destra o sinistra del primo. Il BubbleSort (in tutte e quattro le versioni presentati) è invece stabile perché, utilizzando una disuguaglianza stretta come condizione per lo scambio, gli elementi identici mantengono lo stesso ordine. Se, al contrario, si fosse utilizzata una disuguaglianza non stretta (array[k] >= array[k+1]), l'algoritmo sarebbe stato instabile.