Correzione Primo Esonero 2014, Esercizio 2

Esercizio 2 *Si supponga di avere a disposizione un esecutore la cui unica abilità aritmetica sia la seguente funzione:*

```
int scomponi(int n, int *f1, int *f2)
```

che ha il sequente comportamento: sotto la precondizione che il parametro `n` *sia un numero intero strettamente positivo restituisce come risultato 0 se* `n` *contiene un numero primo, e* `1` *se* `n` *contiene un numero composto. In tal caso, carica nei parametri* `f1` *ed* `f2` *due fattori (maggiori strettamente di 1) il cui prodotto è il valore passato nel parametro* `n`*.*

Ad esempio, siano `x` *ed* `y` *due variabili intere. La chiamata* `scomponi(17, &x, &y)` *restituisce 0. Viceversa la chiamata* `scomponi(24, &x, &y)` *restituisce 1 e carica* `x` *ed* `y` *con due fattori di 24, per esempio 4 e 6, oppure 8 e 3, oppure 12 e 2 (ma non possiamo fare ipotesi su quali essi siano).*

1. *Scrivere una funzione* ricorsiva `maxPrimoRec(int m)` *che usa* `scomponi` *per calcolare il massimo fattore primo di un numero positivo* `m`*;*
2. ★*scrivere una funzione* iterativa `maxPrimoIt(int m)` *che usa* `scomponi` *per calcolare il massimo fattore primo di un numero positivo* `m` *[*SUGGERIMENTO*: aiutarsi con un array ausiliario dove memorizzare i fattori via via calcolati].*

Soluzione: Per fare questo esercizio è innanzitutto necessario capire che la funzione `scomponi` dà come risultato una coppia di fattori qualsiasi e noi non possiamo in nessun modo speculare sulle proprietà di questi fattori. Spero che la cosa non vi sorprenda: a volte una specifica è rilassata e noi non possiamo fare ipotesi sul fatto che una funzione dia una particolare soluzione canonica. Spesso ci sono buoni motivi per operare in questo modo. Nel nostro caso, potrebbe essere l'uso di una procedura che tenta di scomporre un numero facendo tentativi causali (cosa realmente utile nei test di divisibilità).

Fatta questa premessa, è chiaro che se n è primo, allora, lui è anche il suo massimo fattore primo. Viceversa il massimo fattore primo di un numero $n = fat_1 \cdot fat_2$ sarà il maggiore tra il massimo fattore primo di fat_1 e il massimo fattore primo di fat_2.

Abbiamo il nostro schema induttivo, con tanto di caso base.

```
int maxPrimo(int n){
    int f1, f2;
    if (scomponi(n, &f1, &f2)) return max(maxPrimo(f1), maxPrimo(f2));
       else return n;
}
```

Errori Tipici L'errore più interessante, è scrivere questa funzione, che, innocentemente, sembra del tutto equivalente a quella scritta sopra:

```
int maxPrimoE(int n){
    int *f1, *f2;
    if (scomponi(n, f1, f2)) return max(maxPrimoE(*f1), maxPrimoE(*f2));
        else return n;
}
```

E invece no! Il problema è che le dichiarazioni `int *f1;` e `int *f2;` dichiarano due puntatori, ma *non allocano memoria*! Viceversa `int f1;` e `int f2;` allocano spazio per due variabili intere. In questo caso quindi, `f1` ed `f2` possono essere puntatori qualsiasi, e un errore di `Segmentation Fault` è il risultato più probabile di questa funzione.

Soluzione Iterativa La versione iterativa, viceversa, è estremamente più complicata e richiede di memorizzare tutti i fattori trovati in una struttura dati (ad esempio un vettore di lunghezza $\log_2 n$, visto che un numero ha al più $\log_2 n$ fattori) e poi continuare a scomporli finchè non si arriva a fattori primi o fattori più piccoli del più grande primo già calcolato. Vediamo una possibile soluzione:

```
int maxPrimoIt(int n){
    int f1, f2, m;
    int f[n];    /* vettore per i fattori trovati */
    int fn=1;    /* numero di fattori             */
    int maxP = 1;
    f[0]=n;
    while (fn>0){ /* finche' non ho esaminato tutti i fattori       */
         m=f[fn--]; /* estraggo il primo fattore in cima alla pila */
     if (scomponi(m, &f1, &f2)) { /* f1,f2 > 1 */
              f[fn++]=f1;
              f[fn++]=f2;
              } /* endif */
         /* altrimenti ho trovato un primo, verifico se
            sia maggiore del massimo primo finora trovato */
         else if (m>maxP) maxP=m;
    } /* endwhile */
    return maxP:
}
```

Ma tutta l'informazione che la funzione iterativa deve mantenere, dove si trova nella versione ricorsiva (che non necessita di strutture dati)? Chiaramente, i fattori ancora da analizzare si trovano memorizzati nelle variabili `f1` ed `f2`, sullo *stack di attivazione* delle funzioni, dove convivono numerose attivazioni della funzione `maxPrimo` e quindi ci sono numerose copie di queste variabili! (di fatto, una *pila*)!