
Esempi di ricorsione: Fibonacci e Hanoi

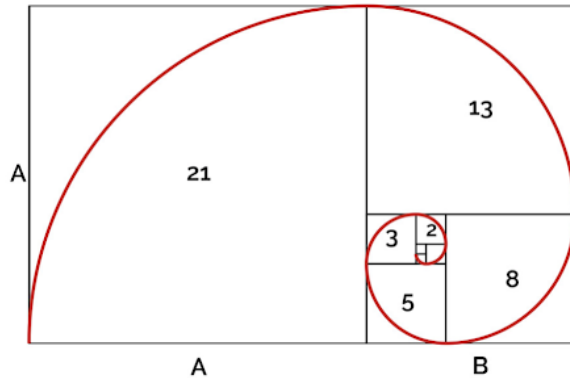
Filippo Cugini

Serie di Fibonacci

La serie di Fibonacci inizia con 0 e 1

Ha la proprietà che ogni successivo numero di Fibonacci è la somma dei due numeri precedenti

0, 1, 1, 2, 3, 5, 8, 13, 21, ...



Serie di Fibonacci

La serie di Fibonacci si può definire ricorsivamente:

$$\text{fibonacci}(0) = 0$$

$$\text{fibonacci}(1) = 1$$

$$\text{fibonacci}(n) = \text{fibonacci}(n - 1) + \text{fibonacci}(n - 2)$$

Serie di Fibonacci

```
#include <stdio.h>
//prototipo di funzione:
unsigned long long int fibonacci (unsigned int n);

int main(void)
{
    unsigned int number; // numero inserito da utente

    // ottieni un intero dall'utente
    printf("%s", "Enter an integer: ");
    scanf("%u", &number);
    // calcola il valore di fibonacci
    unsigned long long int result = fibonacci(number);

    // stampa il risultato
    printf("Fibonacci(%u) = %llu\n", number, result);
}
```

Serie di Fibonacci

```
// Definizione ricorsiva della funzione fibonacci
unsigned long long int fibonacci(unsigned int n)
{
    // caso di base
    if (0 == n || 1 == n) {
        return n;
    }
    else { // passo ricorsivo
        return fibonacci (n - 1) + fibonacci (n - 2);
    }
}
```

Serie di Fibonacci

```
Enter an integer: 0  
Fibonacci(0) = 0
```

```
Enter an integer: 1  
Fibonacci(1) = 1
```

```
Enter an integer: 2  
Fibonacci(2) = 1
```

```
Enter an integer: 3  
Fibonacci(3) = 2
```

```
Enter an integer: 10  
Fibonacci(10) = 55
```

```
Enter an integer: 40  
Fibonacci(40) = 102334155
```

Serie di Fibonacci

La chiamata a `fibonacci` da `main` non è una chiamata ricorsiva, ma tutte le chiamate successive a `fibonacci` sono ricorsive




```
unsigned long long int fibonacci(unsigned int n)
{
    // caso di base
    if (0 == n || 1 == n) {
        return n;
    }
    else { // passo ricorsivo
        return fibonacci (n - 1) + fibonacci (n - 2);
    }
}
```

Serie di Fibonacci

Ogni volta che `fibonacci` è chiamata, essa verifica immediatamente il caso di base, cioè se `n` è uguale a 0 o a 1

Se questo è vero, viene restituito `n`



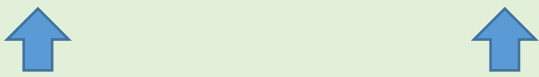
```
unsigned long long int fibonacci(unsigned int n)
{
    // caso di base
    if (0 == n || 1 == n) {
        return n;
    }
    else { // passo ricorsivo
        return fibonacci (n - 1) + fibonacci (n - 2);
    }
}
```


Serie di Fibonacci

Se n è maggiore di 1, il passo di ricorsione genera *due* chiamate ricorsive

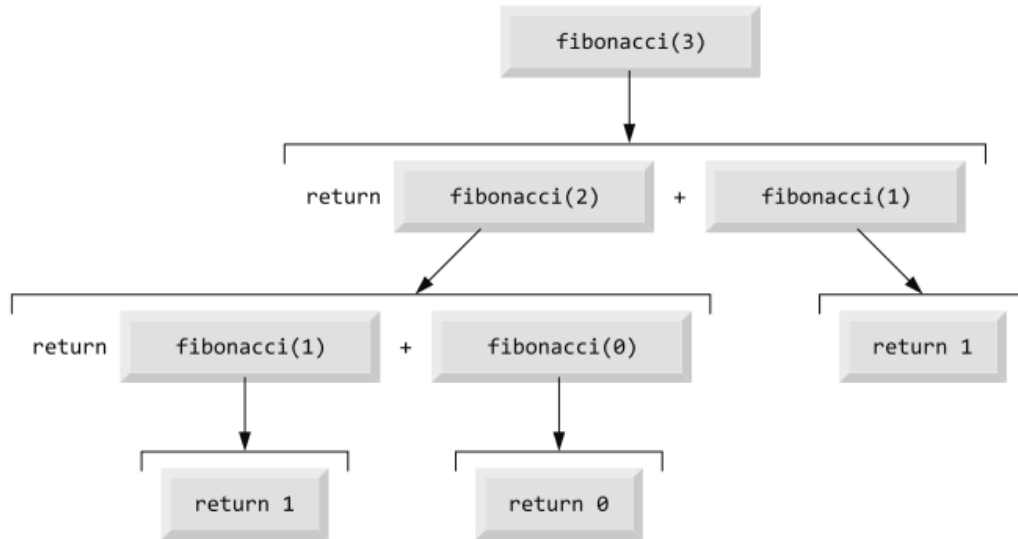
Ogni chiamata risolve un problema leggermente più semplice dell'originaria chiamata a `fibonacci`

```
unsigned long long int fibonacci(unsigned int n)
{
    // caso di base
    if (0 == n || 1 == n) {
        return n;
    }
    else { // passo ricorsivo
        return fibonacci (n - 1) + fibonacci (n - 2);
    }
}
```



Serie di Fibonacci

Insieme di chiamate ricorsive per `fibonacci(3)`:



Complessità esponenziale

Un avvertimento è d'obbligo riguardo ai programmi ricorsivi come quello che usiamo qui per generare i numeri di Fibonacci

Ogni livello di ricorsione nella funzione `fibonacci` ha un effetto di raddoppio sul numero delle chiamate

Il numero delle chiamate ricorsive che saranno eseguite per calcolare l' n^{mo} numero di Fibonacci è dell'ordine di 2^n

La situazione può rapidamente sfuggire di mano

Calcolare il 20^{mo} numero di Fibonacci richiede un numero dell'ordine di $2^{20} \rightarrow$ circa un milione di chiamate

Calcolare il 30^{mo} numero richiede circa un miliardo di chiamate, e così via

Si parla di complessità esponenziale

L'esempio mostrato usa una soluzione intuitivamente attraente per calcolare i numeri di Fibonacci, ma esistono approcci migliori

Serie di Fibonacci non ricorsiva

```
#include <stdio.h>
#define MAX 93

//prototipo di funzione:
unsigned long long int fibonacci (unsigned int n);

int main(void)
{
    // calcola e stampa il valore di Fibonacci
    // per numeri da 0 a MAX
    for (unsigned int loop = 0; loop <= MAX; ++loop) {
        printf("fibonacci(%u)=%llu\n", loop, fibonacci(loop));
    }
}
```

Serie di Fibonacci non ricorsiva

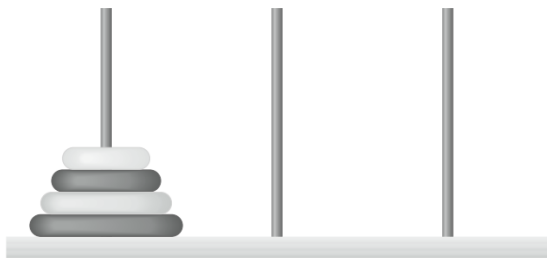
```
unsigned long long int fibonacci(unsigned int n)
{
    unsigned long long int fib1 = 0;
    unsigned long long int fib2 = 1;

    // ripeti per trovare l'ennesimo valore di Fibonacci
    for (unsigned int j = 2; j <= n; ++j) {
        if (j % 2 == 0)
            fib1 += fib2;
        else
            fib2 += fib1;
    }
    // restituzione dell'ennesimo valore di Fibonacci
    if (n % 2 == 0)
        return fib1;
    else
        return fib2;
}
```

Torre di Hanoi

La leggenda narra che in un tempio dell'Estremo Oriente alcuni sacerdoti tentino di spostare una pila di dischi da un piolo a un altro

La pila iniziale aveva 64 dischi infilati in un piolo, disposti in dimensione decrescente dal basso verso l'alto

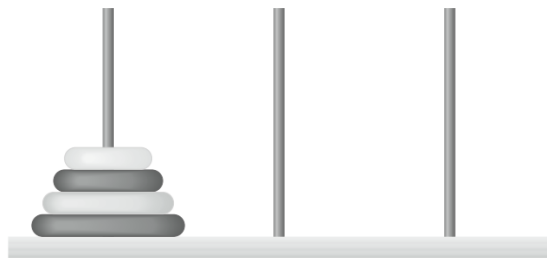


Torre di Hanoi

I sacerdoti hanno il vincolo che:

- 1) si sposti esattamente un solo disco alla volta e
- 2) che nessun disco più grande possa essere collocato sopra un disco più piccolo

Un terzo piolo è disponibile per contenere temporaneamente i dischi



Se affrontassimo questo problema coi metodi convenzionali, ci troveremmo in difficoltà

Con la ricorsione, diventa immediatamente trattabile

Spostare n dischi si può vedere in termini dello spostamento di solo $n - 1$ dischi (da cui la ricorsione):

- a) Spostare $n - 1$ dischi dal piolo 1 al piolo 2, usando il piolo 3 come supporto temporaneo
- b) Spostare l'ultimo disco (il più grande) dal piolo 1 al piolo 3
- b) Spostare gli $n - 1$ dischi dal piolo 2 al piolo 3, usando il piolo 1 come supporto temporaneo

Scrivete un programma per risolvere il problema delle Torri di Hanoi

Usate una funzione ricorsiva con quattro parametri:

- a) Il numero di dischi da spostare
- b) Il piolo su cui questi dischi sono inizialmente infilati
- c) Il piolo nel quale spostare questa pila di dischi
- d) Il piolo da usare come supporto temporaneo

Il programma deve stampare le istruzioni necessarie a spostare i dischi dal piolo di partenza al piolo di arrivo

```
1 → 3 (spostare un disco dal piolo 1 al piolo 3)
1 → 2
```

Torre di Hanoi

```
#include <stdio.h>

// prototipo di funzione
void tower(int c, int start, int end, int temp);

int main()
{
    printf("%s", "Enter the starting number of disks: ");
    int n; // numero di dischi
    scanf("%d", &n);
    // stampa di istruzioni per spostare i dischi dal
    // piolo 1 al piolo 3
    // usando il piolo 2 per deposito temporaneo
    tower(n, 1, 3, 2);
}
```

Torre di Hanoi

```
// tower stampa ricorsivamente istruzioni per lo
// spostamento dei dischi dal piolo iniziale al piolo
// finale usando il piolo temp per deposito temporaneo
void tower(int c, int start, int end, int temp)
{
    // caso di base
    if (1 == c) {
        printf("%d --> %d\n", start, end);
        return;
    }
    // sposta c - 1 dischi da start a temp
    tower(c - 1, start, temp, end);
    // sposta l'ultimo disco da start a end
    printf("%d --> %d\n", start, end);
    // sposta c - 1 dischi da temp a end
    tower(c - 1, temp, end, start);
}
```