

RICORSIONE

(Come, Quando e Perché)

Perché

- i programmi ricorsivi sono più chiari, più semplici, più brevi e più facili da capire delle corrispondenti versioni iterative.
- il programma spesso riflette fedelmente la strategia di soluzione del problema.
- spesso la soluzione trovata può poi trasformarsi, più o meno meccanicamente, in una soluzione iterativa equivalente ma più efficiente.

RICORSIONE

Svantaggi

- **spazio:** ogni chiamata di funzione può richiedere spazio per i parametri e le variabili locali, oltre che l'indicazione del punto di rientro della chiamata. Queste informazioni (**record di attivazione**) sono memorizzate in una pila dalla quale vengono automaticamente cancellate non appena la chiamata di una funzione è terminata .
- **tempo:** le operazioni coinvolte in una chiamata (allocazione e rilascio della memoria, copia dei valori dei parametri nella memoria locale, rientro dalla chiamata) contribuiscono tutte ad aumentare il tempo di calcolo.

Il principio di induzione

- Induzione (semplice)

Se $P(0)$ e, per ogni n , $P(n)$ implica $P(n + 1)$

Allora $P(n)$ per ogni n

- Induzione forte o completa

Se $(P(i)$ per $0 < i < n$) implica $P(n)$ per ogni n

Allora $P(n)$ per ogni n

Esempio

Calcolo ricorsivo di Fattoriale

```
int fatt(int n)
{ if (n==0) return 1;
  return n * fatt(n-1);
}
```

Corretto?

Esempio

Calcolo ricorsivo di Fattoriale

```
int fatt_rec(int n)
/*prec: n ≥ 0
postc: restituisce il
fattoriale di n */
{ if (n == 0) return 1;
  return n * fatt_rec(n-1);
}
```

Esempio

Calcolo iterativo di Fattoriale

```
int fatt_iter(int n) {
  int i, f = 1;
  for (i = 2; i <= n; i++)
    f *= i;
  return f;
}
```

Esempio: algoritmo di Eulero

$\text{mcd}(p, 0) = p$

$\text{mcd}(p, q) = \text{mcd}(q, p \bmod q)$

```
int euler_rec(int p, int q) {  
    if (q == 0)  
        return p;  
    return euler_rec(q, p % q);  
}
```

Esempio: algoritmo di Eulero iterativo

$\text{mcd}(p, 0) = p$

$\text{mcd}(p, q) = \text{mcd}(q, p \bmod q)$

```
int euler_iter(int p, int q){  
    int r;  
    while (q != 0) {  
        r = p % q;  
        p = q;  
        q = r;  
    }  
    return p;  
}
```

Calcolo ricorsivo di Potenza

$$(b^0 = 1 \quad e \quad b^i = b * b^{i-1})$$

```
int potenza(int x, int i)
/*prec: i ≥ 0
postc: restituisce bi */
{ if (i == 0) return 1;
  return x * potenza(x, i-1);
}
```

Calcolo ricorsivo di Potenza

$$(b^0 = 1 \quad e \quad b^{2n+1} = b * b^n * b^n \quad e \quad b^{2n} = b^n * b^n)$$

```
int potenza2_d(int x, int n) {
  if (n == 0)
    return 1;
  else if (n % 2 == 1)
    return x * potenza2_d(x, n/2) * potenza2_d(x, n/2);
  else
    return potenza2_d(x, n/2) * potenza2_d(x, n/2);
}
```

Calcolo ricorsivo di Potenza

Notato la duplicazione delle chiamate?

```
int potenza2(int x, int n) {
    if (n == 0)
        return 1;
    else {
        int y = potenza2(x, n/2);
        if (n % 2 == 1)
            return x * y * y;
        else
            return y * y;
    }
}
```

potenza2_d richiede n chiamate ricorsive, potenza2 solo $\sim \log n$ (profondità dell'albero delle chiamate)

Calcolo iterativo di Potenza

```
int potenza_it(int x, int n) {
    float res = 1.0;
    while (n > 0) {
        res = x * res;
        n--;
    }
    return res;
}

int potenza2_it(int x, int n) {
    float res = 1.0;
    int i;
    if (n <= 0)
        return 1;
    for (i = 1, res = x; 2*i < n; i *= 2)
        res *= res;
    for (; i < n; i++)
        res *= x;
    return res;
}
```

Esempio classico di ricorsione: Fibonacci

$\text{fib}(0)=0; \text{fib}(1)=1; \text{fib}(n+2)=\text{fib}(n+1)+\text{fib}(n)$

```
int fib(int n)
```

```
/*prec:  $n \geq 0$ 
```

```
postc: restituisce l'n-simo
```

```
numero di Fibonacci*/
```

```
{
```

```
if( $n==1 \parallel n==0$ ) return n;
```

```
return fib(n-1) + fib(n-2);
```

```
}
```

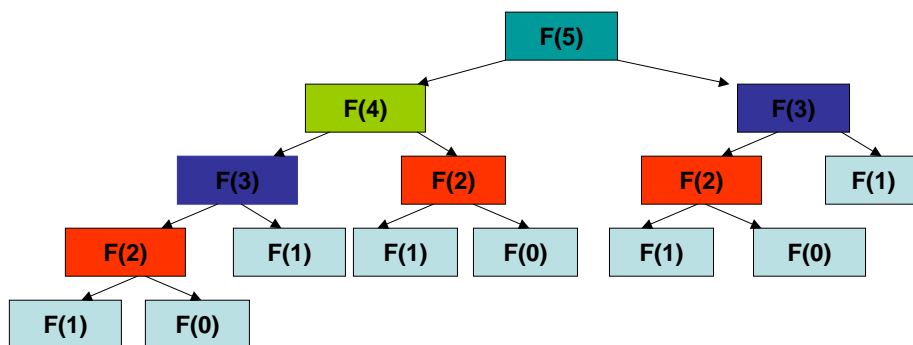
Altro esempio di
ricorsione multipla

Spesso utilizzato come
esempio di inefficienza
della ricorsione

Esempio classico di ricorsione: Fibonacci

Calcoli ripetuti e numero esponenziale di chiamate ricorsive

Per $n=42$, $\text{fib}(n) = 267914296$ calcolato con $866988873 (= \text{fib}(43))$ chiamate



Esempio classico di ricorsione: Fibonacci

$\text{fib}(0)=0; \text{fib}(1)=1; \text{fib}(n+2)=\text{fib}(n+1)+\text{fib}(n)$

```
int fib_iter(int n) {
    int aux, p0 = 0, p1 = 1;
    while (n > 0) {
        aux = p1;
        p1 = p0 + p1;
        p0 = aux;
        n--;
    }
    return p1;
}
```

Ecco una soluzione
(iterativa!) efficiente:
solo n iterazioni
(LINEARE)

Fattoriale rivisto: tail recursion

```
int fattoriale2(int n)
/*postc: se n ≥ 0 restituisce il fattoriale di n, altrimenti 0*/
{ int ris=1;
  if (n < 0) return 0;
  if (n > 1) fattor2(ris,n);
  return ris; }

int fattor2(int ris,int n)
/*prec: n > 0
Postc: restituisce il fattoriale di n */
{   if ( n == 1) return ris ;
    return fattor2(n*ris , n-1) ;
}
```

Fattoriale rivisto: versione più efficiente

```
int fattoriale3(int n)
/*postc: se  $n \geq 0$  restituisce il fattoriale di  $n$ , 0 altrimenti */
{ int ris=1;
  if (n < 0) return 0;
  if (n > 1) fattor3(&ris,n);
  return ris; }

void fattor3(int *ris , int n)
/*prec:  $n > 1$ 
Postc: restituisce il fattoriale di  $n$  */
{   if ( n == 1) return;
    *ris = n * (*ris);
    fattor3(ris,--n); }
```

/*Poiché ris è passato per riferimento le modifiche del suo valore nella chiamata di fattor3 sono presenti anche all'uscita da fattor3.*/

Altro Esempio

Occorrenze di un carattere in una stringa

```
int occCar(char* s, char c)
/*prec:  $s \neq \text{NULL}$ ,
postc: restituisce il numero di occorrenze del
carattere in  $c$  nella stringa  $s$ */
{ if (*s == '\0') return 0;
  if (*s == c)
    return 1 + occCar(++s,c);
  else
    return occCar(++s,c);
}
```

Occorrenze: tail recursion

```
int occCar2(char* s, char c)
/*postc: restituisce il numero di occorrenze del
carattere in c nella stringa s, se s != NULL, - 1 altrimenti*/
{ int ris = 0;
  if (s) ris = oCar(s, c, ris); else return -1;
  return ris;}

int oCar(char* s, char c,int ris)
/*prec: s!= NULL,
postc: restituisce il numero di occorrenze del carattere in c nella stringa s*/
{ if (*s == '\0') return ris;
  if (*s == c) return oCar(++s, c,ris+1);
  else return oCar(++s, c,ris);
}
```

Versione più efficiente

```
int occCar3(char* s, char c)
/*postc: restituisce il numero di occorrenze del
carattere in c nella stringa s, se s != NULL, -1 altrimenti*/
{ int ris=0;
  if (s) oCar2(s,c,&ris); else return -1;
  return ris;}

void oCar2(char* s, char c,int * ris)
/*prec: s!= NULL,
postc: restituisce il numero di occorrenze del carattere in c nella stringa s*/
{ if (*s != '\0')
  if (*s == c)
  {(*ris)++; oCar2(++s, c,ris);}
  else oCar2(++s, c,ris);}
```

Ricordate Fibonacci ?

$\text{fib}(0)=0; \text{fib}(1)=1; \text{fib}(n+2)=\text{fib}(n+1)+\text{fib}(n)$

```
int fib(int n)
/*prec: n ≥ 0
postc: restituisce l'n-simo numero di Fibonacci*/
{
if(n==1 || n==0) return n;
return fib(n-1) + fib(n-2);
}
```

Fibonacci: tail recursion

```
int fibon2(int n)
/*postc: se n ≥ 0 restituisce l'n-simo numero
di Fibonacci, -1 altrimenti*/
{ int ris , fibn=1 , fibnMeno1=0;
  if (n < 0) return -1;
  ris = fibnMeno1;
  if (n > 0) ris = fib2(fibn , fibnMeno1 , n);
  return ris;}

int fib2(int fibnpiu1, int fibn, int n)
{ if ( n == 0) return fibn;
return fib2(fibnpiu1 + fibn,fibnpiu1,n-1); }
```