

Informatica A

Ricorsione

Sommario

- Definizioni basate sull'induzione
- L'iterazione e la ricorsione
- Che cosa significa "ricorsivo"
- Come si realizza una funzione ricorsiva
- Come si esegue una funzione ricorsiva
- Esempi

Definizioni induttive

- Sono comuni in matematica
- Esempio: il fattoriale di un naturale N ($N!$)
 - Se $N=0$ il fattoriale $N!$ è 1
 - Se $N>0$ il fattoriale $N!$ è $N * (N-1)!$
- Esempio: numeri pari
 - 0 è un numero pari
 - Se n è un numero pari anche $n+2$ è un numero pari

Dimostrazioni induttive

- Dimostriamo che $(2n)^2 = 4n^2$
 - Distributività del quadrato rispetto alla moltiplicazione

1. Se $n=1$: vero (per verifica diretta)
2. *Suppongo* sia vero per $n=k$ (ipotesi di induzione) e *dimostro* l'uguaglianza di cui sopra, per $n=k+1$:

$$\begin{aligned}(2n)^2 &= (2(k+1))^2 = (2k+2)^2 = (2k)^2 + 8k + 4 = \\ &(\text{per ipotesi di induzione}) \quad 4k^2 + 8k + 4 = \\ &4(k^2 + 2k + 1) = 4(k+1)^2 = 4n^2\end{aligned}$$

- (1) è il caso base, (2) è il passo induttivo

Iterazione e ricorsione

- Sono i due concetti informatici che nascono dal concetto di induzione
- L'iterazione si realizza mediante la tecnica del ciclo
- Per il calcolo del fattoriale:
 - $0! = 1$
 - $n! = n (n - 1)(n - 2).... 1$ (realizzo un ciclo)

Fattoriale – versione iterativa

```
int fattoriale (int n) {  
    int f = 1;  
    while (n > 0) {  
        f = f * n;  
        n--;  
    }  
    return f;  
}
```

Lo "spirito" del metodo ricorsivo

- Esiste almeno un *caso base* che rappresenta un sotto-problema di facile soluzione
 - Esempio: se $N=0$, so $N!$ in modo "immediato" (vale 1)
- Esiste almeno un *passo induttivo* che (prima o poi) riconduce il problema generale a un caso base
 - Consiste nell'esprimere la soluzione al problema (su dati di una dimensione generica n) in termini di operazioni semplici e della soluzione allo stesso problema su dati di dimensione $n-1$ (che, per tali dati, si suppone risolto per ipotesi)
 - Esempio: per n generico esprimo $n!$ in termini di n (è un dato direttamente accessibile) moltiplicato per (è una operazione semplice e nota) il valore di $(n-1)!$ (che non conosco ma so calcolare per ipotesi induttiva)

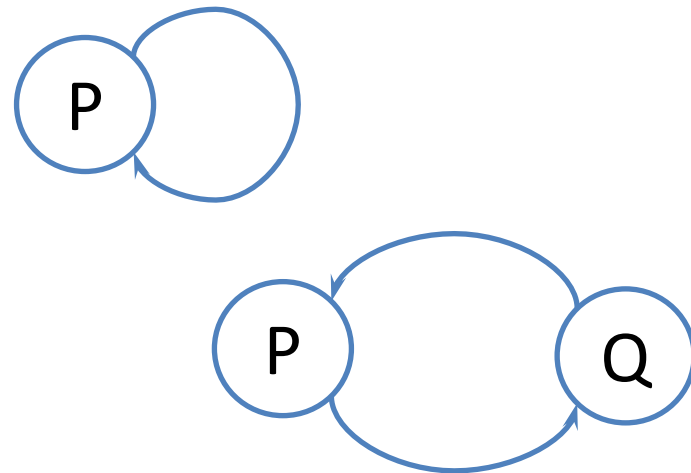
Fattoriale – versione ricorsiva

1. $n! = 1$ se $n = 0$
2. $n! = n * (n - 1)!$ se $n > 0$

- Riduce il calcolo a un calcolo più semplice
- Ha senso perché si basa sempre sul fattoriale del numero più piccolo, che io conosco
- Ha senso perché si arriva a un punto in cui non è più necessario riusare la definizione (2) e invece si usa la (1)
- (1) è il caso base, (2) è il passo induttivo

Ricorsione nei sottoprogrammi

- Dal latino **re-currere**
 - Ricorrere, fare ripetutamente la stessa azione
- Un sottoprogramma P invoca se stesso
 - **Direttamente**
 - P invoca P
 - **Indirettamente**
 - P invoca Q che invoca P



Formulazione ricorsiva in C

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

Simulazione del calcolo

Invocazione di: FattRic(3)

3 = 0? No

→ Calcola fattoriale di 2 e moltiplicalo per 3

2 = 0? No

→ Calcola fattoriale di 1 e moltiplicalo per 2

1 = 0? No

→ Calcola fattoriale di 0 e moltiplicalo per 1

0 = 0? Sì

→ Fattoriale di 0 è 1

→ Fattoriale di 1 è 1 per fattoriale di 0, cioè $1 \times 1 = 1$

→ Fattoriale di 2 è 2 per fattoriale di 1, cioè 2

→ Fattoriale di 3 è 3 per fattoriale di 2, cioè 6

Ma...

- ... se ogni volta la funzione richiama se stessa... perché la catena di invocazioni non continua all'infinito?
- Quando si può dire che una ricorsione è ben definita?
- Informalmente
 - Se per ogni applicazione del passo induttivo ci si avvicina alla situazione riconosciuta come caso base, allora la definizione non è circolare, e la catena di invocazioni termina

Un altro esempio: la serie di Fibonacci

- Leonardo Pisano, figlio di Bonaccio
Fi' Bonacci, 1202
- Partì dallo studio sullo sviluppo di una colonia di conigli in circostanze ideali
 - Partiamo da una coppia di conigli appena nati
 - La coppia diventa fertile dopo 1 mese e genera una nuova coppia ogni mese dopo il primo
 - Le nuove coppie si comportano in modo analogo
 - I conigli non muoiono mai (!)
 - Quante coppie ci sono dopo n mesi?

Un altro esempio: la serie di Fibonacci

- Dopo 1 mese una coppia di conigli sarà fertile
- Dopo 2 mesi ci saranno due coppie di cui una sola fertile
- Dopo 3 mesi ci saranno $2+1=3$ coppie perché solo la coppia fertile avrà generato; di queste tre, due saranno le coppie fertili, quindi
- Nel mese seguente (quarto mese dal momento iniziale) ci saranno $3+2=5$ coppie

Definizione ricorsiva della serie

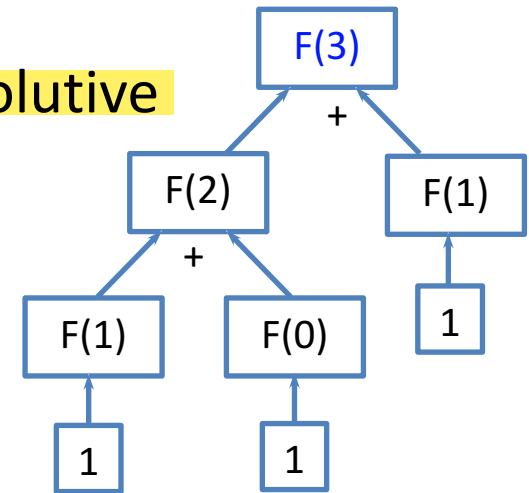
- I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

- $F = \{ f_0, \dots, f_n, \dots \}$

- $f_0 = 1$
- $f_1 = 1$ } 2 casi base

- Per $n > 1$, $f_n = f_{n-1} + f_{n-2}$ } 1 passo induttivo



- Notazione "funzionale"

- $f(i)$ invece di f_i

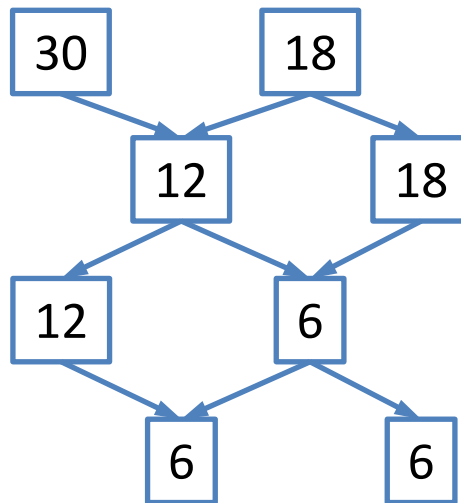
Numeri di Fibonacci in C

```
int fibo(int n) {  
    if (n == 0 || n == 1)  
        return 1;  
    else  
        return fibo(n-1) + fibo(n-2);  
}
```

Ovviamente supponiamo $n \geq 0$

Un altro esempio: MCD à-la-Euclide

- MCD tra M e N (M, N naturali positivi)
 - Se $M=N$ allora MCD è N 1 caso base
 - Se $M>N$ allora esso è il MCD tra N e $M-N$
 - Se $N>M$ allora esso è il MCD tra M e $N-M$ 2 passi induttivi



MCD versione **ricorsiva**

```
int EuclideRic (int m, int n) {  
    if (m == n)  
        return m;  
    if (m > n)  
        return EuclideRic(m - n, n);  
    else  
        return EuclideRic(m, n - m);  
}
```

MCD versione **iterativa**

```
int EuclideIter (int m, int n) {  
    while (m != n) {  
        if (m > n) {  
            m = m - n;  
        } else {  
            n = n - m;  
        }  
    }  
    return m;  
}
```

Funzione esponenziale (intera)

- Definizione iterativa

- $b^e = 1$ se $e = 0$
- $b^e = b * b * \dots b$
(e volte) se $e > 0$

Codice iterativo

```
int esp(int b, int e) {  
    int i, r = 1;  
    for (i = 1; i <= e; i++)  
        r = r * b;  
    return r;  
}
```

Codice ricorsivo

- Definizione induttiva

- $b^e = 1$ se $e = 0$
- $b^e = b * b^{(e-1)}$ se $e > 0$

```
int esp(int b, int e) {  
    if (e == 0)  
        return 1;  
    else  
        return b * esp(b, e-1);  
}
```

Esecuzione di funzioni ricorsive

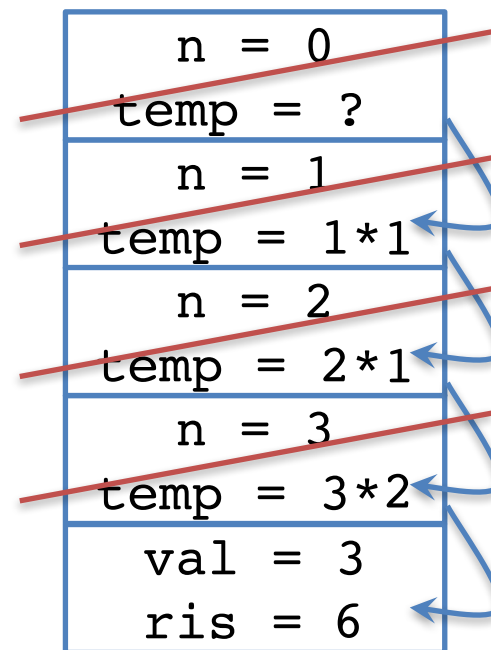
- In un certo istante possono essere in corso diverse attivazioni dello stesso sottoprogramma
 - Ovviamente sono tutte sospese tranne una, l'ultima invocata, all'interno della quale si sta svolgendo il flusso di esecuzione
- Ogni attivazione esegue lo stesso codice ma opera su copie distinte dei parametri e delle variabili locali

Il modello a runtime: esempio

```
int FattRic(int);

int main() {
    int val, ris;
    printf("dammi un naturale ");
    scanf("%d", &val);
    ris = FattRic(val);
    printf("\nfattoriale= %d", ris);
    return 0;
}

int FattRic (int n) {
    int temp;
    if (n == 0)
        return 1;
    else {
        temp = n * FattRic(n-1);
        return temp;
    }
}
```



temp

- Cella temporanea per memorizzare il risultato della funzione chiamata

Assumiamo $val = 3$

Terminazione (ancora!)

- Attenzione al rischio di catene infinite di chiamate
- Occorre che le chiamate siano soggette a una condizione che assicura che prima o poi la catena termini
- Occorre anche che l'argomento sia “progressivamente ridotto” dal passo induttivo
 - In modo da tendere prima o poi al caso base

Esempio: lunghezza stringhe

Soluzione iterativa

```
int  lunghezzaI(char s[]) {  
    int i = 0;  
    while ( s[ i ] != '\0' )  
        i++;  
    return i;  
}
```

Soluzione ricorsiva

```
int  lunghezzaR(char *s) {  
    if (*s == '\0') return 0;  
    else return 1 + lunghezzaR(s+1);  
}
```


Le parole palindrome

- Palindromo è una parola che si può leggere indifferentemente da destra a sinistra e viceversa
- Tutte le parole di un solo carattere sono considerate palindrome
 - "a" -> sì
 - "db" -> no
 - "Anna" -> no (Se l'analisi è case insensitive)
 - "anno" -> no
 - "anilina" -> sì
 - "restereste" -> no (Anche se composta di due metà "uguali")
 - "onorarono" -> sì
 - "lagerregal" -> sì (Scaffale del magazzino, in tedesco)
 - "saippuakivikauppias" -> sì (Mercante di steatite, in finlandese)

Esercizio

- Si scriva un programma che memorizza in un array di caratteri una parola letta da stdin e verifica se la parola è o non è palindroma
- `int palindromo(char parola[], ...)`
 - Restituisce 1 o 0
 - Si chiede, in particolare, di farne una versione **iterativa** e una versione **ricorsiva**
- Versione iterativa: confronto tra tutte le coppie di lettere simmetriche rispetto al "centro"
 - Attenzione, la parola può avere un numero pari o dispari di caratteri
- Versione ricorsiva: un palindromo è tale se ...

Palindromi in versione ricorsiva

- Un palindromo è tale se:

- La parola è di lunghezza 0 o 1, oppure

Caso base

- Il primo e l'ultimo carattere della parola sono uguali e inoltre la sotto-parola che si ottiene ignorando i caratteri estremi è a sua volta un palindromo

Passo induttivo

Il passo induttivo riduce la dimensione del problema!

Palindromi in versione ricorsiva

```
int palindromoR(char par[], int da, int a) {  
    if (da >= a)  
        return 1;  
    else  
        return (par[da] == par[a] &&  
                palindromoR(par, da+1, a-1));  
}
```

Palindromi ricorsivi: variante

```
int palindromoR(char par[], int da, int a) {  
    if (da >= a)  
        return 1;  
    else  
        return (palindromoR(par, da+1, a-1)  
                && par[da] == par[a]);  
}
```

- Qual è la differenza?

Attenzione

- Come si fa la prima chiamata?
- Si suppone che il chiamante conosca (o calcoli) la lunghezza della stringa.
 - Esempio: `palindromoR("oro", 0, 2)`
 - Oppure: `palindromoR(str, 0, strlen(str)-1)`
 - Oppure ancora...

Introduciamo un'altra funzione:

```
int palindromo(char *s) {  
    return palindromoR(s, 0, strlen(s)-1);  
}
```

Palindromi in versione iterativa

```
int palindromoI( char par[] ) {  
    // Assegna gli indici del primo e  
    // ultimo carattere della parola  
    int da = 0;  
    int a  = strlen( par ) - 1;  
    while (da < a) {  
        // Scandisce la parola facendo muovere  
        // gli indici verso il centro di par  
        if (par[da] != par[a])  
            // Dice FALSE alla prima differenza  
            return 0;  
        ++da;  
        --a;  
    }  
    // Se arriva alla fine senza differenze, TRUE  
    return 1;  
}
```

Parametri passati per indirizzo

```
/* Incrementa il primo parametro  
   del valore del secondo */  
void incrementa(int *n, int m) {  
    if (m != 0) {  
        (*n)++;  
        incrementa(n, m-1);  
    }  
    return;  
}
```

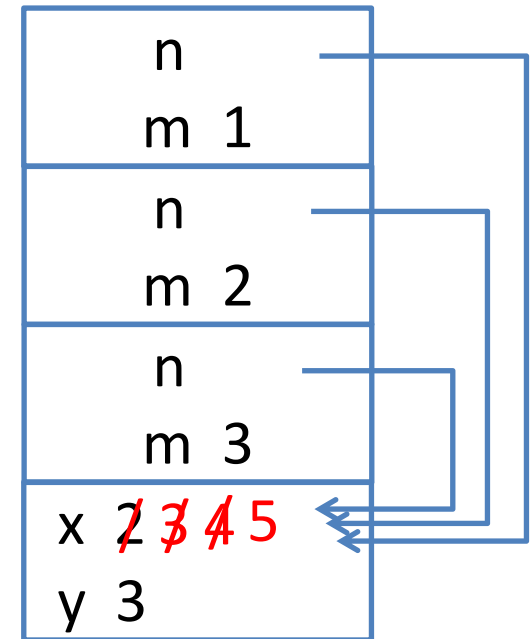
```
int x, y;  
...  
x = 2;  
y = 3;  
incrementa(&x, y);
```

```
/* NB la chiamata iniziale  
   ha forma diversa dalla  
   chiamata ricorsiva */
```


Modello a run-time

```
void incrementa(int *n, int m) {  
    if (m != 0) {  
        (*n)++;  
        incrementa(n, m-1);  
    }  
    return;  
}
```

```
int x, y;  
...  
x = 2;  
y = 3;  
incrementa(&x, y);
```



Ancora stringhe palindrome

- Stringa palindroma se:
 - Stringa vuota, oppure
 - Ha un solo carattere, oppure
 - Il suo cuore è una stringa palindroma, racchiusa tra caratteri uguali (primo e ultimo)

```
int Palindrome(char *PC, char *UC);
```

```

#define LunghMaxStringa 100

int  Palindrome(char *PC, char *UC);

int main() {
    char  str[LunghMaxStringa+1];
    int  LunghStr;
    scanf("%s", str); /* NB assumiamo non ci siano spazi */
    LunghStr = strlen(str);
    if (LunghStr == 0)
        printf("La stringa è palindroma");
    else {
        printf("La stringa");
        if (! Palindrome(str, str + LunghStr - 1))
            printf(" NON");
        printf(" è palindroma\n");
    }
    return 0;
}

```

```

int Palindrome(char *PC, char *UC) {
    if (PC >= UC) /* Stringa vuota o di 1 carattere */
        return 1;
    if (*PC != *UC) /* Primo e ultimo car. diversi */
        return 0;
    else
        /* Chiamata ricorsiva escludendo
           primo e ultimo carattere */
        return Palindrome(PC+1, UC-1);
}

```

```

int f(char *p, char *u) {
    return p>u || *p==*u && f(p+1,u-1);
}

```

Osservazioni sul programma

- Legge una stringa (termina con '\0')
- Ne calcola la lunghezza con strlen()
- Se stringa vuota o di un carattere → palindroma
- Se caratteri agli estremi diversi → *non* lo è
- Altrimenti applica la funzione Palindrome alla stringa privata degli estremi
 - Elegante uso di due puntatori ...
 - Indirizzi del primo e ultimo carattere della parte non ancora esaminata
 - ... spostati avanti e indietro a ogni chiamata ricorsiva

Altri tipi di palindromi

- **Palindromi a parola**
 - "Fall leaves after leaves fall"
- **Palindromi a frase** (ignorando spazi e punteggiatura)
 - "I topi non avevano nipoti"
 - "Avida di vita, desiai ogni amore vero, ma ingoiai sedativi, da diva"
 - "Sun at noon, tan us!"
 - "Was it a (b|c|r)at I saw?"
 - "Was It a car or a cat I saw?"
- **Esempi notevoli:**
 - G. Perec, "9691" (> 5000 caratteri)
 - "Trace l'inégal palindrome. Neige [...] ne mord ni la plage ni l'écart."
 - G. Varaldo, "11 Luglio 1982" (> 4000 caratteri)
 - Ai lati, a esordir, dama [...] a Madrid, rosea Italia!
 - Batman, "Una storia italiana" (> 1000 caratteri)
 - O idolo, se vero, mal onori parole [...] rapirono l'amore, v'è sol odio.

Altri tipi di palindromi

- Palindromi a riga

- J. A. Lindon, "Doppelganger"

- "Entering the lonely house with my wife,
I saw him for the first time
peering furtively from behind a bush
[...]

- Peering furtively from behind a bush,
I saw him, for the first time
entering the lonely house with my wife."

- Esercizio: implementare funzioni di verifica per palindromi a parola, a frase, a riga

Esempio: inversione stringa

- Scrivere una funzione che costruisca la stringa inversa di una stringa data nello stesso vettore
- Il chiamante deve invocare la funzione con parametri: la stringa da invertire, `str[]`, e la lunghezza, `n`, della stessa
- Il ragionamento induttivo che applichiamo è:
 - Se abbiamo il risultato dell'inversione della sottostringa ottenuta da quella di partenza senza il primo e l'ultimo carattere (che sarà lunga $n-2$ caratteri) ...
 - ... possiamo completare l'inversione della stringa di partenza, scambiando il primo e l'ultimo carattere

Esempio: inversione stringa

```
void inversione(char str[], int n) {  
    if (n <= 1) return;           // Caso base  
    inversione(&str[1], n-2);    // Passo induttivo  
    char temp = str[n-1];  
    str[n-1] = str[0];  
    str[0] = temp;  
}
```

Esercizio: riscrivere la funzione, con prototipo

```
void inversione(char str[], int inizio, int fine);
```

Esempio: inversione stringa

Metodo alternativo di inversione

- Il chiamante passa anche il **puntatore alla stringa** che conterrà l'inversa, cioè il prototipo che vogliamo è:

```
char * inversione(char *s, char *t);
```

- Si restituisce la stringa costruita (puntatore)

- Ci serviamo di

```
char * strcpy(char *dest, char *src)
```

– Copia la stringa src nella stringa dest e restituisce il ptr dest

```
char * strncat(char *dest, char *src, int n)
```

– Aggiunge i primi n caratteri di src in coda a dest e restituisce il ptr dest

Esempio: inversione stringa

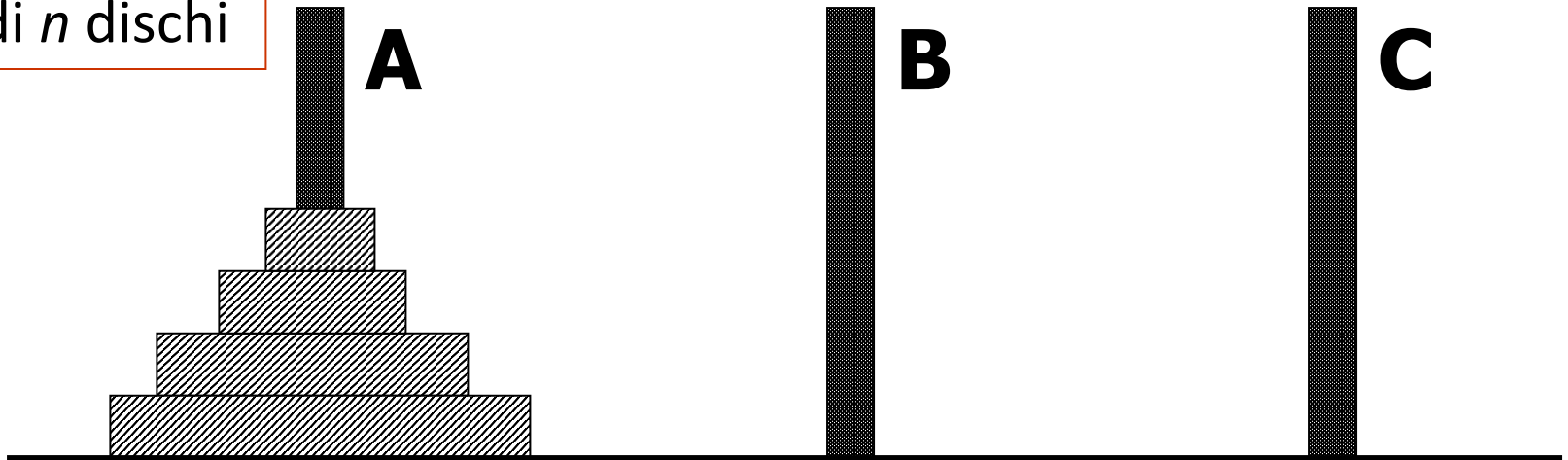
```
char * inversione(char *s, char *t) {  
    if (strlen(s) <= 1)    // Caso base  
        return strcpy(t, s);  
    else                    // Passo induttivo  
        return strncat(inversione(s+1,t), s, 1);  
}
```

Attenzione: la memoria che contiene la stringa t è allocata a cura del programma che effettua la prima chiamata (strncat(...) e inversione(...)) *non allocano* memoria)

Le torri di Hanoi

Spostare tutta la torre da A a C spostando un cerchio alla volta e senza mai mettere un cerchio più grosso su uno più piccolo

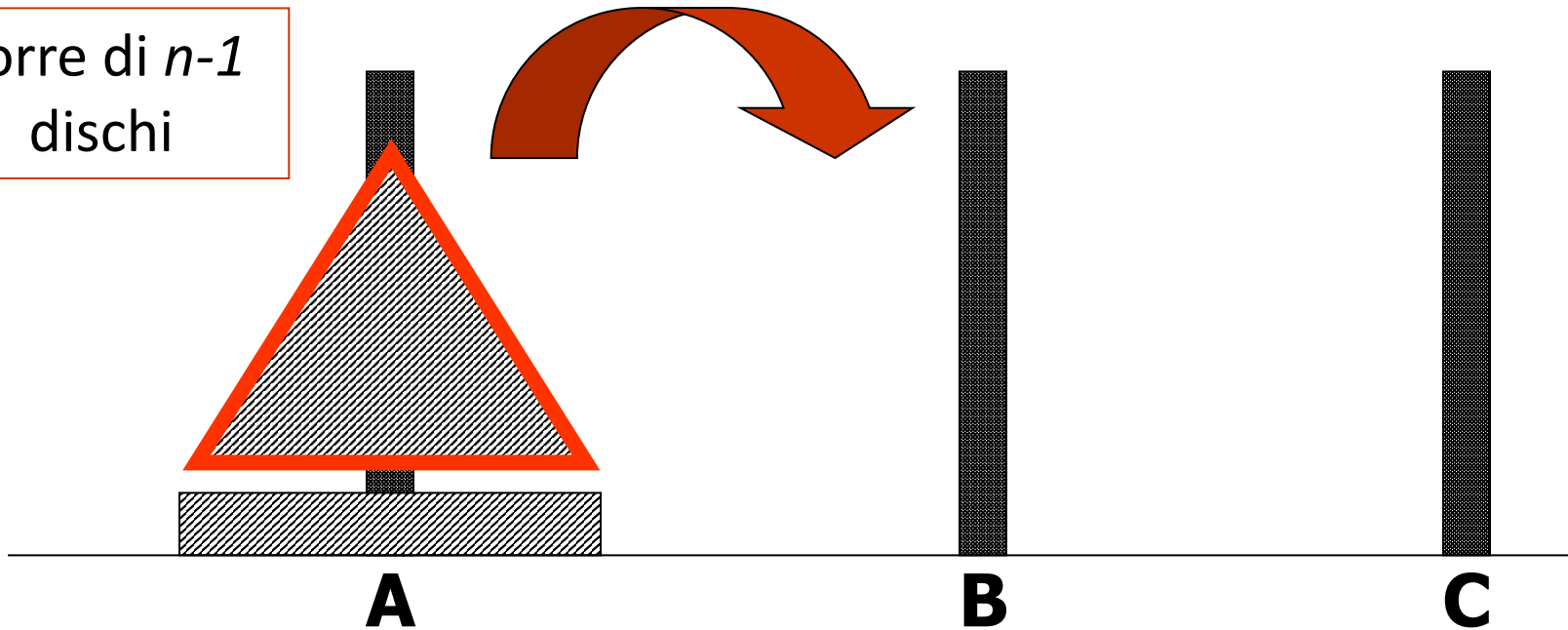
Torre di n dischi



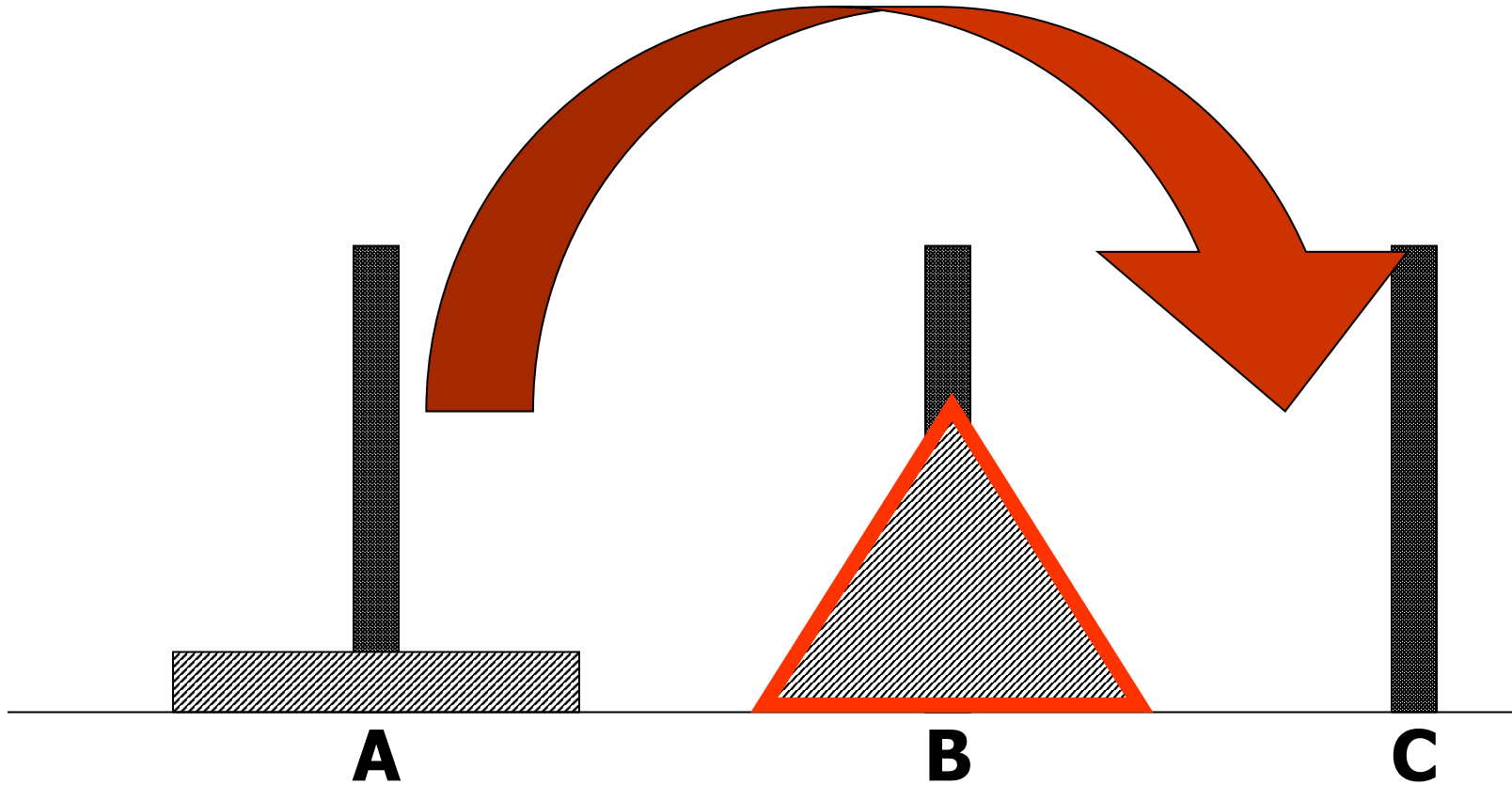
Formulazione ricorsiva?

Le torri di Hanoi

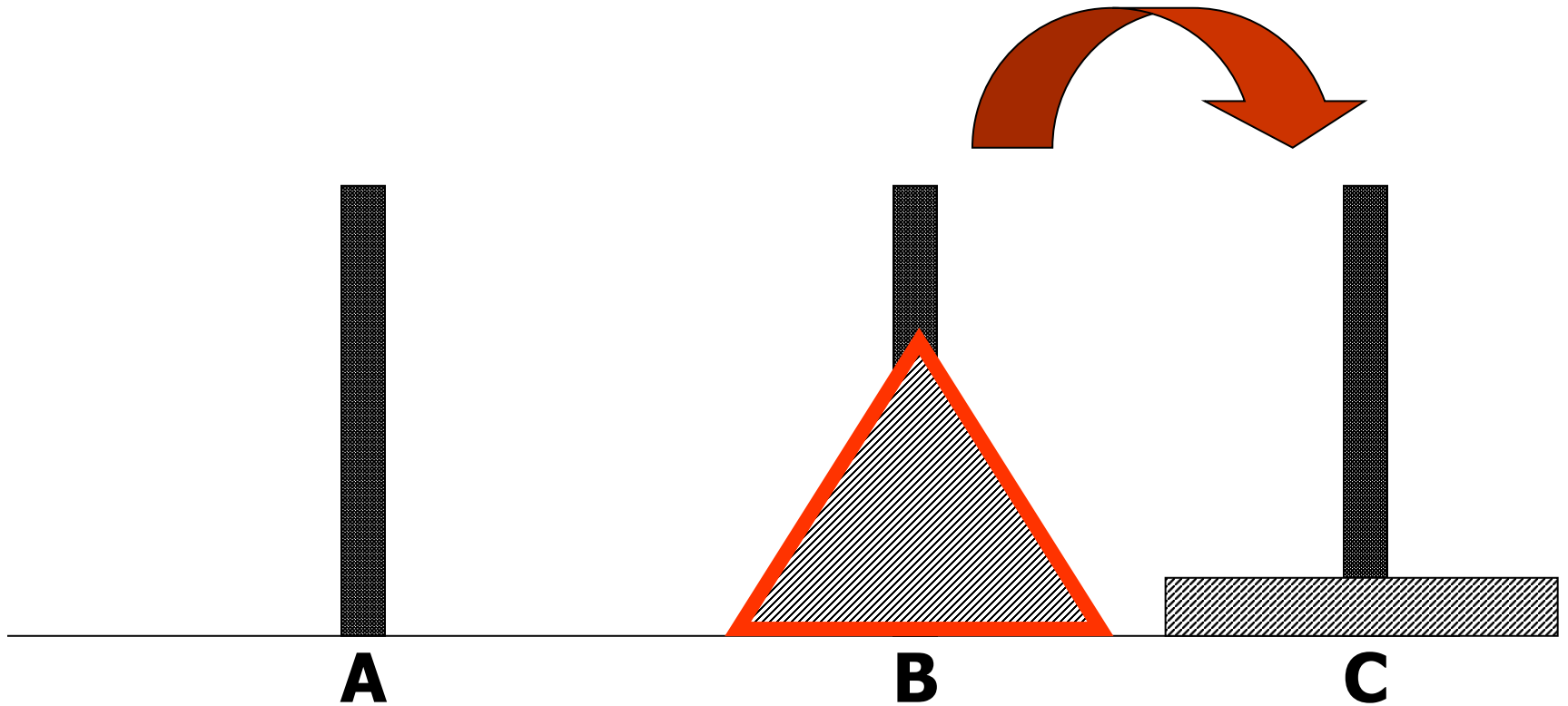
Torre di $n-1$
dischi



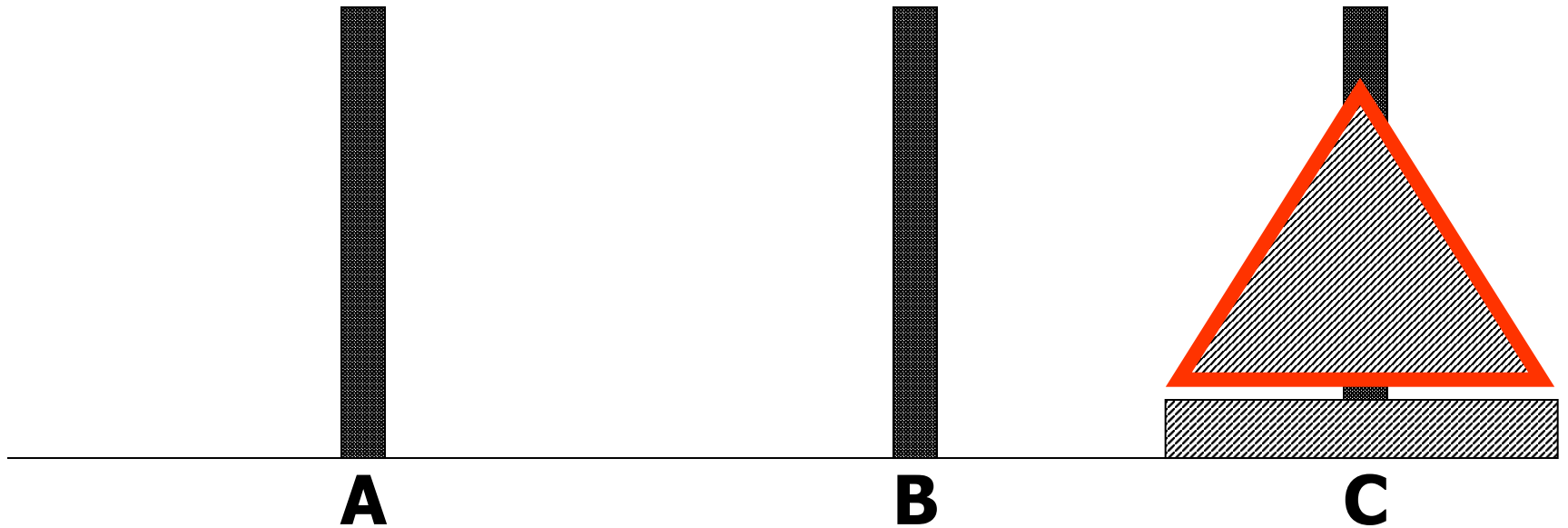
Le torri di Hanoi



Le torri di Hanoi



Le torri di Hanoi



Il metodo

- Il programma deve stampare una serie di comandi di spostamento
 - So spostare la torre di 0 elementi da A a C
 - Caso di base
 - Per spostare la torre di N elementi da A a C
 - Sposto la torre di N-1 cerchi da A a B
 - Sposto il cerchio restante in C
 - Sposto la torre di N-1 elementi da B a C
- Gli spostamenti da un piolo all'altro avvengono usando "il piolo residuo" come piolo di appoggio

Chiamata iniziale

```
hanoi(12, 'A', 'C', 'B')  
/* significa che abbiamo una torre di 12 cerchi  
   da trasferire da A a C "appoggiandoci" a B */
```

```
void hanoi (int n, char from, char to, char with) {  
    if ( n != 0 ) {  
        hanoi( n-1, from, with, to );  
        printf("Sposta un cerchio da %c a %c\n", from, to);  
        hanoi( n-1, with, to, from );  
    }  
}
```

Una variante più "stringata"

```
hanoi (int n, int from, int to);
```

```
/*
```

Significa che abbiamo una torre di n cerchi da trasferire da $from$ a to ; adottiamo una codifica dei nomi dei tre pioli che permetta di ricavare in modo immediato il "nome" di ogni piolo partendo dai "nomi" degli altri due: i pioli sono ora indicati da *interi* 1, 2, 3, e non dei caratteri 'A', 'B', 'C'

```
*/
```

```
void hanoi (int n, int from, int to) {  
    if (n != 0) {  
        hanoi(n-1, from,  $6 - from - to$ );  
        printf("sposta cerchio da %d a %d", from, to);  
        hanoi(n-1,  $6 - from - to$ , to);  
    }  
}
```

Hanoi: soluzione iterativa

- Non è così evidente ...
- Stabiliamo un "senso orario" tra i pioli: 1, 2, 3 e poi ancora 1, ecc.
- Per muovere la torre nel prossimo piolo in senso orario bisogna ripetere questi due passi:
 - Sposta il disco più piccolo in senso orario
 - Fai l'unico altro spostamento possibile con un altro disco

Ricorsione o iterazione?

- Spesso le soluzioni ricorsive sono eleganti
- Sono vicine alla definizione del problema
- Però a volte possono essere inefficienti
 - Chiamare un sottoprogramma significa allocare memoria a run-time
- N.B. è sempre possibile trovare un corrispondente iterativo di un programma ricorsivo

Calcolo numeri di fibonacci

```
int fibonacci (int n) {  
    if (n == 0 || n == 1)  
        return 1;  
    else return (fibonacci(n-1) +  
                fibonacci(n-2));  
}
```

Inefficiente!

Calcola più volte l'i-esimo numero di fibonacci!

Numeri di Fibonacci

- La prima volta che si calcola un dato numero di Fibonacci lo si memorizza in un array
- Dalla seconda volta in poi, anziché ricalcolarlo, lo si legge direttamente dall'array
- Occorre un valore “innocuo” con cui inizializzare l'array per indicare che il numero di Fibonacci corrispondente non è stato ancora calcolato
 - Qui si può usare ad esempio 0

Inizializzazione dell'array memo

```
#define MAX 100
long int memo[MAX];
main() {
    int i, n;
    for (i=2; i < MAX; i++) // inizializzazione
        memo[i]=0;
    memo[0] = memo[1] = 1; // i due casi base
    printf("Un intero: ");
    scanf("%d",&n);
    printf("fib(%d) = %d", n, fib(n));
}
```


Calcolo con memo

```
long fib(int n) {  
    if (memo[n] != 0)  
        return memo[n];  
    memo[n] = fib(n-1) + fib(n-2);  
    return memo[n];  
}
```

Drastica riduzione della *complessità* (aumento di efficienza)

Questa soluzione richiede un tempo *lineare* in n

La soluzione precedente richiede un tempo *esponenziale* in n

Calcolo con memo

E se per caso ci servisse un $\text{fib}(k)$ con $k \geq \text{MAX}$??

```
long fib( int n ) {  
    if (n >= MAX)  
        return fib(n-1) + fib(n-2);  
    if (memo[n] != 0)  
        return memo[n];  
    memo[n] = fib(n-1) + fib(n-2);  
    return memo[n];  
}
```

Con questo accorgimento non c'è limite a n e l' "effetto memo" è limitato ai soli valori $\leq \text{MAX}$
(che sono comunque quelli che sarebbero ricalcolati con maggiore frequenza in assenza di memo)