

**Luiss**

Libera Università Internazionale  
degli Studi Sociali Guido Carli

# **Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica**

## Lezione 4. Ricorsione, divide et impera

Alessio Martino

**20 Marzo 2023**

**LUISS**



# Reminder: Discord, OrientationLuiss

<https://discord.gg/2TTJCgrX>

- ***#domande-general*** (chat lezione)
- ***#codice*** (sharing is caring)

<https://orientation.luiss.it/>


- ***Forum Tematico*** (offline sharing is still caring)
- ***Materiale Didattico*** (slide lezioni e problemi settimanali)

**Nota:** la chat su WebEx non sarà monitorata durante la lezione!

# Lezioni

Puoi accedere alle lezioni tramite YouTube o Webex:

 **YouTube** (preferibile se hai problemi di connessione)

 **Cisco webex** (limitata a 1000 utenti; però puoi diventare ***Chad (Stacy) per un giorno*** condividendo la tua soluzione ai problemi che consideriamo di volta in volta!)



Chad and gigachad is for losers.

Oggi potrete ambire ad essere **terachad**.

Se siete collegati tramite Webex, alzate la mano per presentare le vostre soluzioni.

```
truth = "I am a Chad! "                                # or Stacy!
def chad(x):                                              # or def stacy(x):
    return chad(x*(10**3))
chad(truth)
```

**don't** try this at home, please

# Regole delle FantaOlimpiadi

## Punti bonus:

- Rispondere a domande sulla chat: +5
- Chad/Stacy per un giorno: +10
- Chad/Stacy per un giorno ringrazia professori Luiss: +10
- Superamento territoriali: +30
- Fase finale: Oro +100 , Argento +70, Bronzo +50

## Punti malus:

- Stare nella chat sbagliata: -10
- Dire "Fantaolimpiadi" su chat/video: -10
- Scrivere "buon pomeriggio" in chat appena inizia la lezione: -5
- Chiedere se si può usare Python alle territoriali: -10

# Programma di oggi

1. Soluzione di alcuni esercizi della scorsa settimana
2. **Ricorsione**
  1. Teoria
  2. Esempi
  3. Esercizio
3. **Divide et impera** (divide & conquer)
  1. Teoria
  2. Esempi
  3. Esercizio
4. Esercizi per casa

# **Soluzioni di alcuni degli esercizi della scorsa settimana**



# Rifiuti da riciclare (1/5)

Scrivete in chat per esporre la vostra soluzione:

[https://training.olinfo.it/#/task/oii\\_riciclo/statement](https://training.olinfo.it/#/task/oii_riciclo/statement)



# Torta di compleanno (2/5)

Scrivete in chat per esporre la vostra soluzione:

<https://training.olinfo.it/#/task/pre-egoi-torta/statement>

# Aggiornamento della macchina virtuale (3/5)

Scrivete in chat per esporre la vostra soluzione:

[https://training.olinfo.it/#/task/preoii\\_vm/statement](https://training.olinfo.it/#/task/preoii_vm/statement)

# Foglietto illustrativo (4/5)

Scrivete in chat per esporre la vostra soluzione:

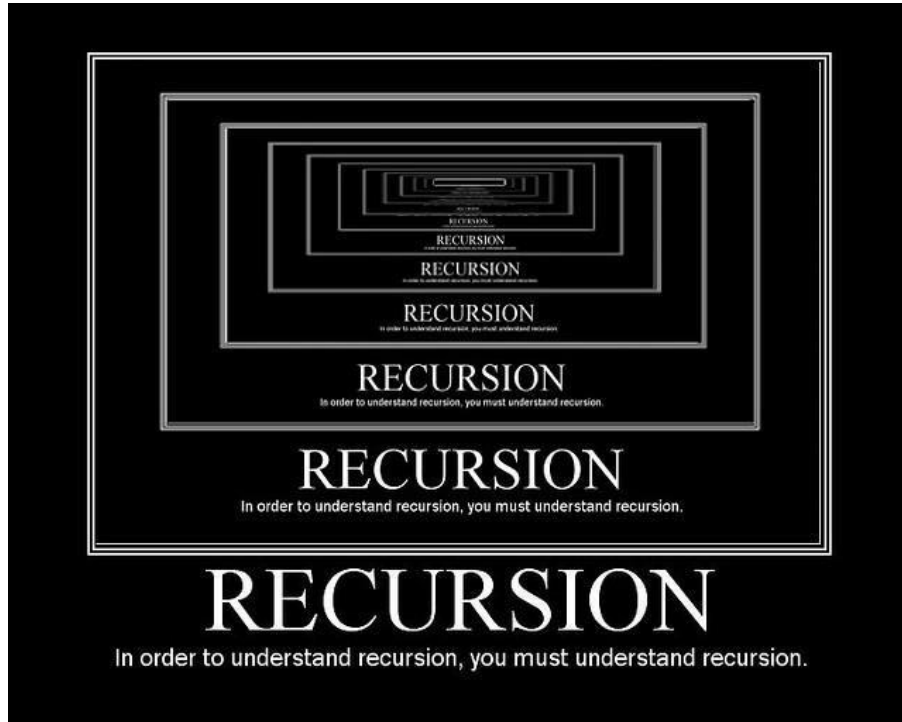
[https://training.olinfo.it/#/task/oii\\_foglietto/statement](https://training.olinfo.it/#/task/oii_foglietto/statement)

# Accampamento (5/5)

Scrivete in chat per esporre la vostra soluzione:

<https://training.olinfo.it/#/task/padrin/statement>

# Ricorsione



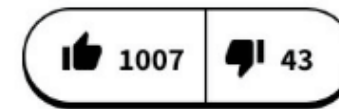
TOP DEFINITION



## recursion

[See](#) recursion.

by [Anonymous](#) December 05, 2002

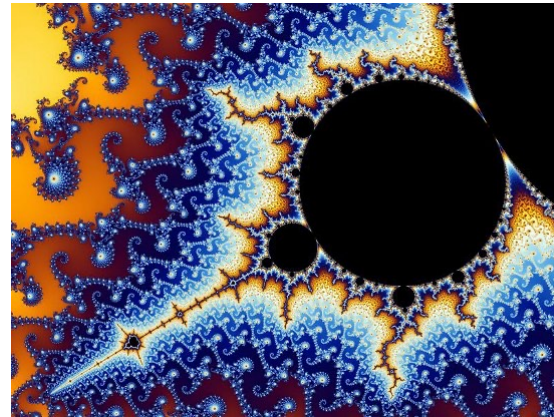
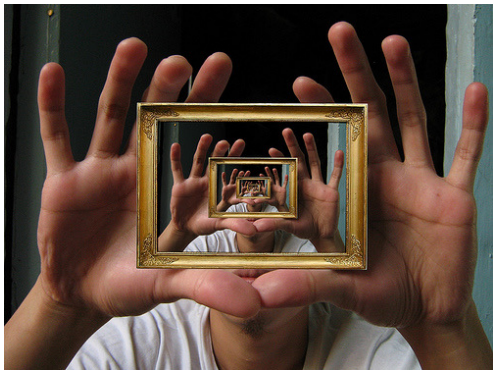


Spot on.

# Idea di base

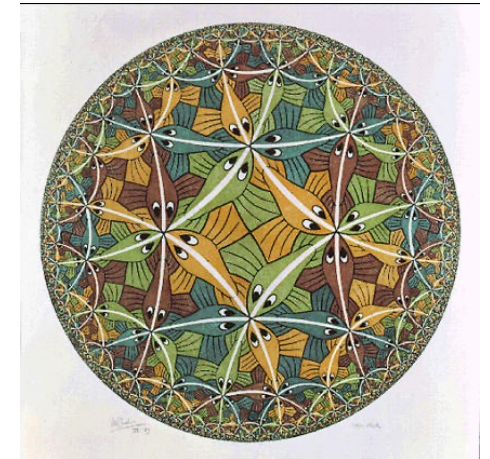
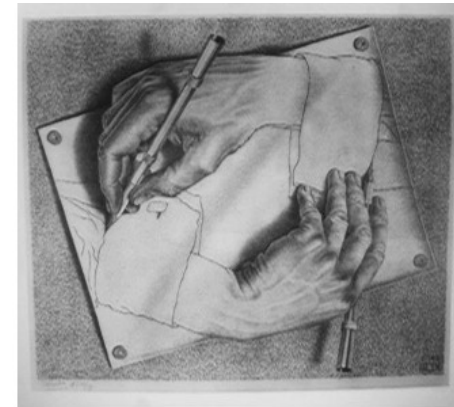
- Concetto semplice, ma molto potente
- Basato sulla nozione di **auto-similarità** e **ripetizione**

Mettete due specchi in  
parallelo...



I frattali sono forse la  
visualizzazione ricorsiva più  
nota (insieme di Mandelbrot)

Anche Escher usava la ricorsione con maestria!



# Ricorsione e problem solving

- Per usare la ricorsione nel progettare la soluzione di un problema, *cercate di risolvere il problema... assumendo che lo abbiate già risolto!*
- Come è possibile?

Definendo la soluzione di un'istanza del problema in termini della soluzione di istanze più piccole

- La funzione che calcola la soluzione *richiamerà se stessa* (una o più volte) all'interno del proprio corpo
- Spesso si ottiene codice estremamente sintetico ed elegante

# Gli elementi del procedimento ricorsivo

Per usare un procedimento ricorsivo:

1. Un problema deve essere scomponibile in sottoproblemi **dello stesso tipo**
2. Occorre trovare delle **relazioni** che legano un problema a sottoproblemi simili
3. È necessario conoscere la soluzione di un caso particolare del problema (**caso base** = condizione di terminazione)



# Esempio #1: il fattoriale

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$$

Calcoliamo 5!

$$\begin{aligned} 5! &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\ &\quad \underbrace{\hspace{1.5cm}}_{1!} \\ &\quad \underbrace{\hspace{1.5cm}}_{2! = 1! \cdot 2} \\ &\quad \underbrace{\hspace{1.5cm}}_{3! = 2! \cdot 3} \\ &\quad \underbrace{\hspace{1.5cm}}_{4! = 3! \cdot 4} \\ &\quad \underbrace{\hspace{1.5cm}}_{5! = 4! \cdot 5} \end{aligned}$$

$$\begin{aligned} n! &= (n-1)! \cdot n \\ 0! &= 1 \end{aligned}$$

# Esempio #1: il fattoriale

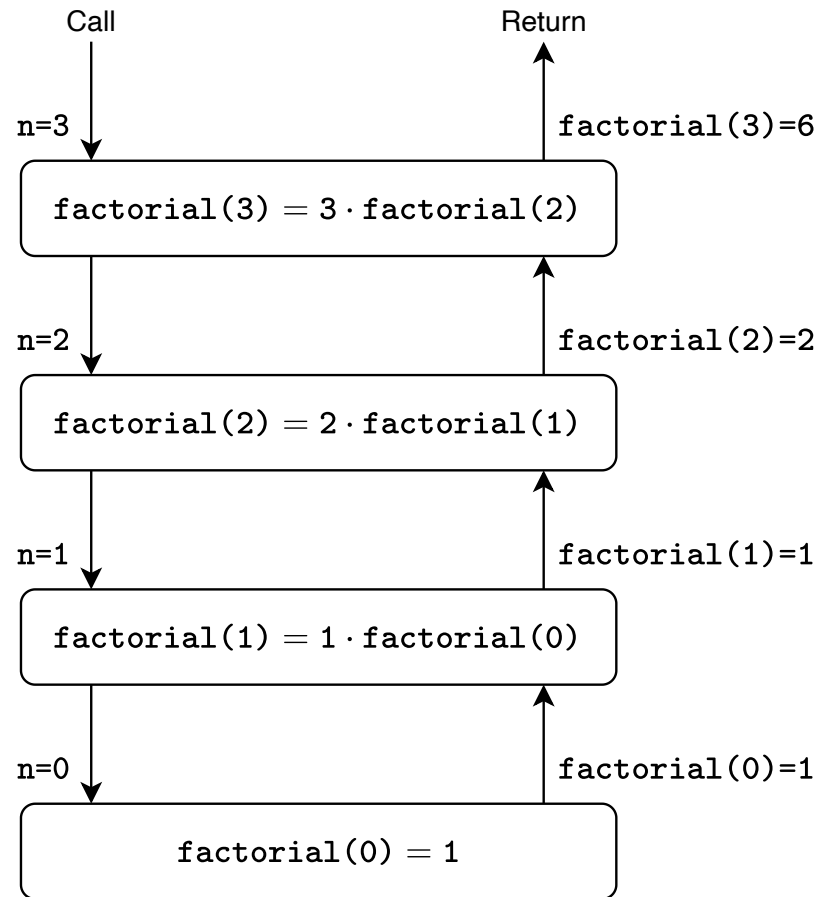


Figura 1: stack diagram per `factorial(3)`

# Esempio #2: Fibonacci

**Definizione.** Ogni numero della sequenza di Fibonacci si ottiene sommando i due numeri precedenti: 0, 1, 1, 2, 3, 5, 8, 11, ...

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{se } n \geq 2 \\ n & \text{se } n = 0, 1 \end{cases}$$

Calcoliamo  $F_5$

$$\begin{array}{c} 5 = 3 + 2 \\ \quad \underbrace{\quad} \quad \underbrace{\quad} \\ \quad 2+1 \quad 1+1 \\ \quad \underbrace{\quad} \\ \quad 1+1 \end{array}$$

# Esempio #2: Fibonacci

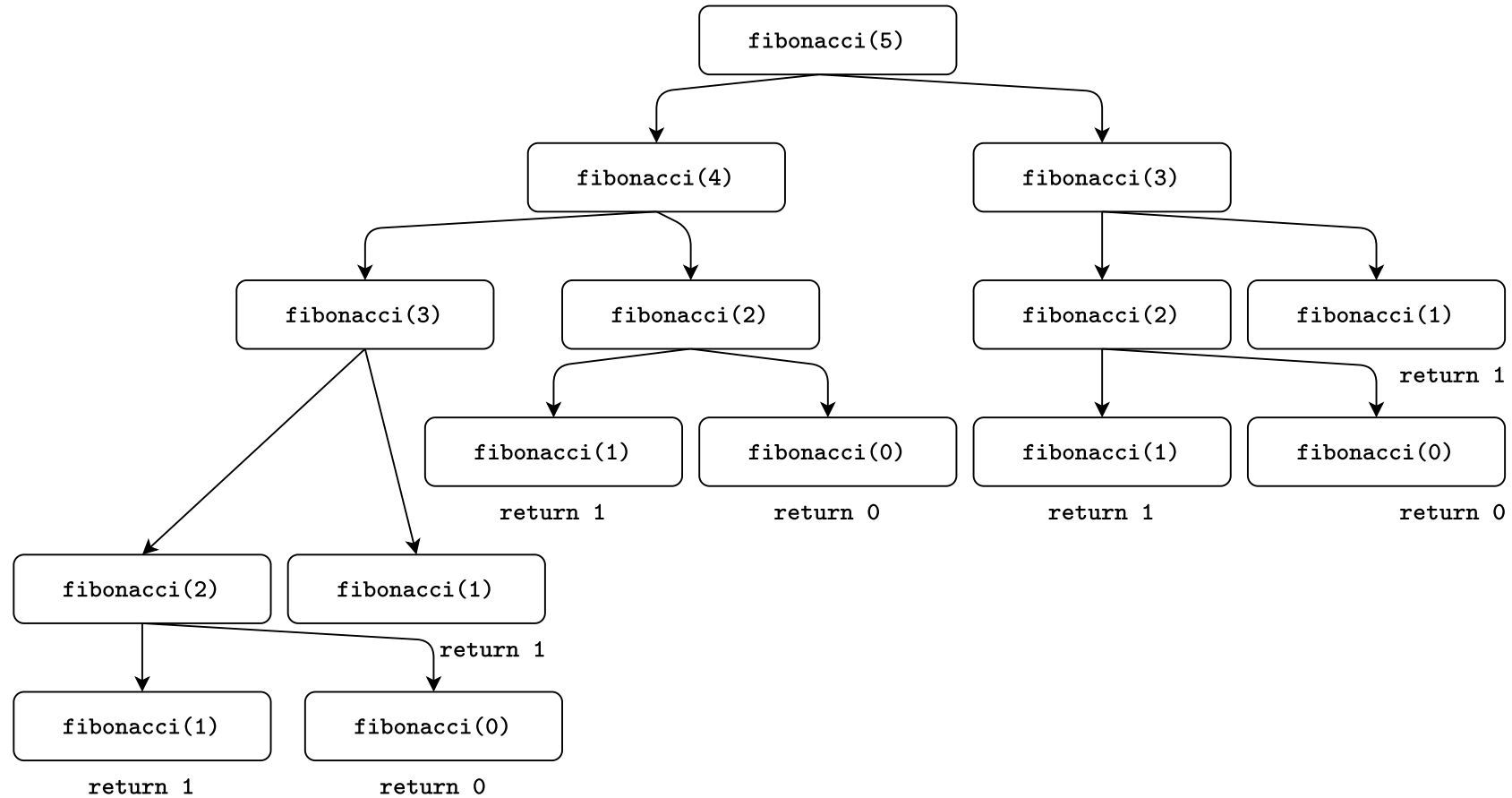


Figura 2: call stack diagram per `fibonacci(5)`

# Esempio #2: Fibonacci

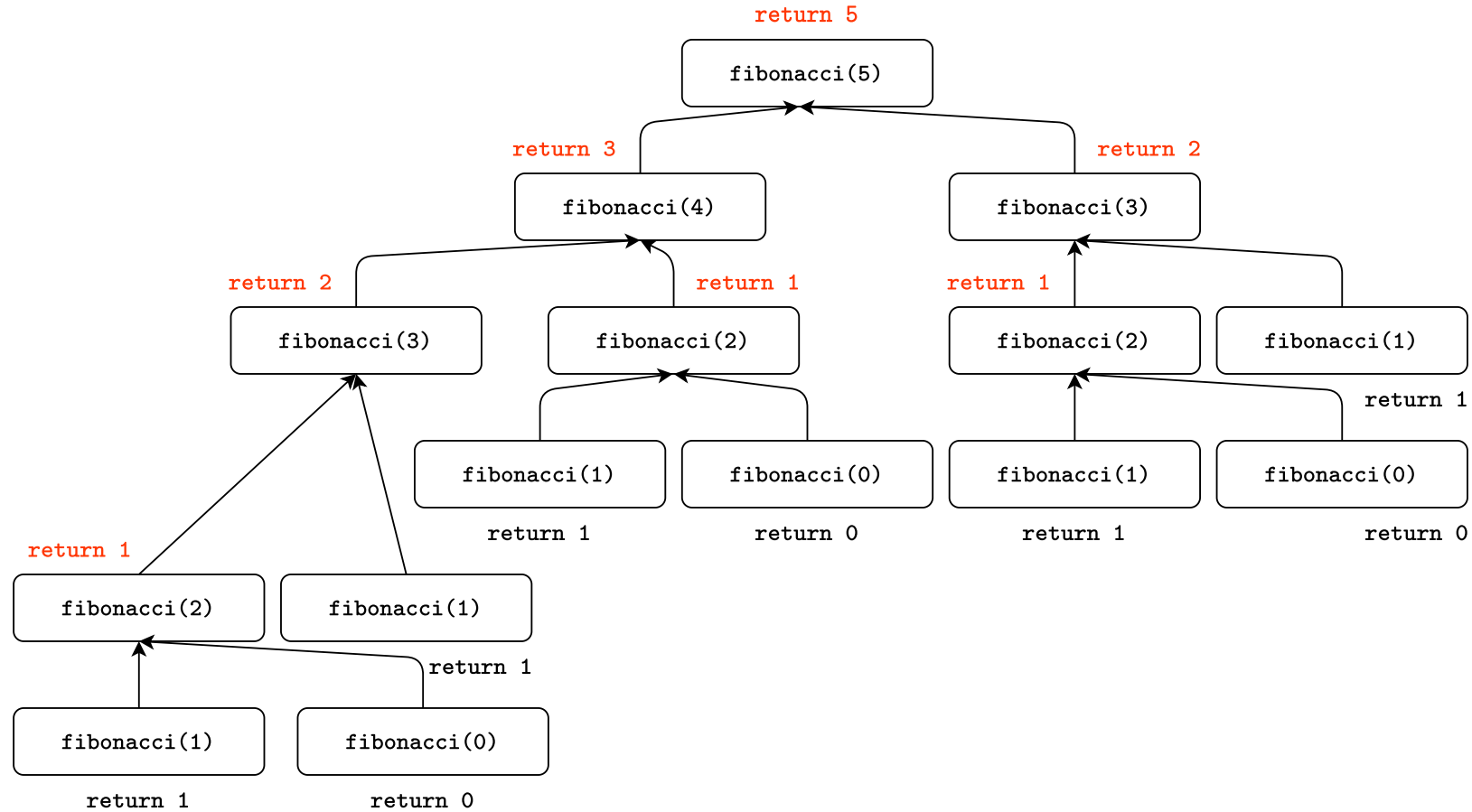


Figura 3: return stack diagram per fibonacci (5)

## Esempio #2.1: Fibonacci (optimized)

**Osservazione:** la precedente implementazione di Fibonacci con ricorsione è altamente inefficiente

- per `fibonacci(5)` ho 3 call a `fibonacci(2)` e 2 call a `fibonacci(3)`
- l'albero delle chiamate è un albero binario
- time complexity per  $F_n$ :  $O(2^n)$

Possiamo fare di meglio?

- sì, se usiamo un approccio iterativo  $\rightarrow$  time complexity per  $F_n$ :  $O(n)$

Possiamo fare di meglio, sempre usando un approccio ricorsivo?

- sì

# Esempio #2.1: Fibonacci (optimized)

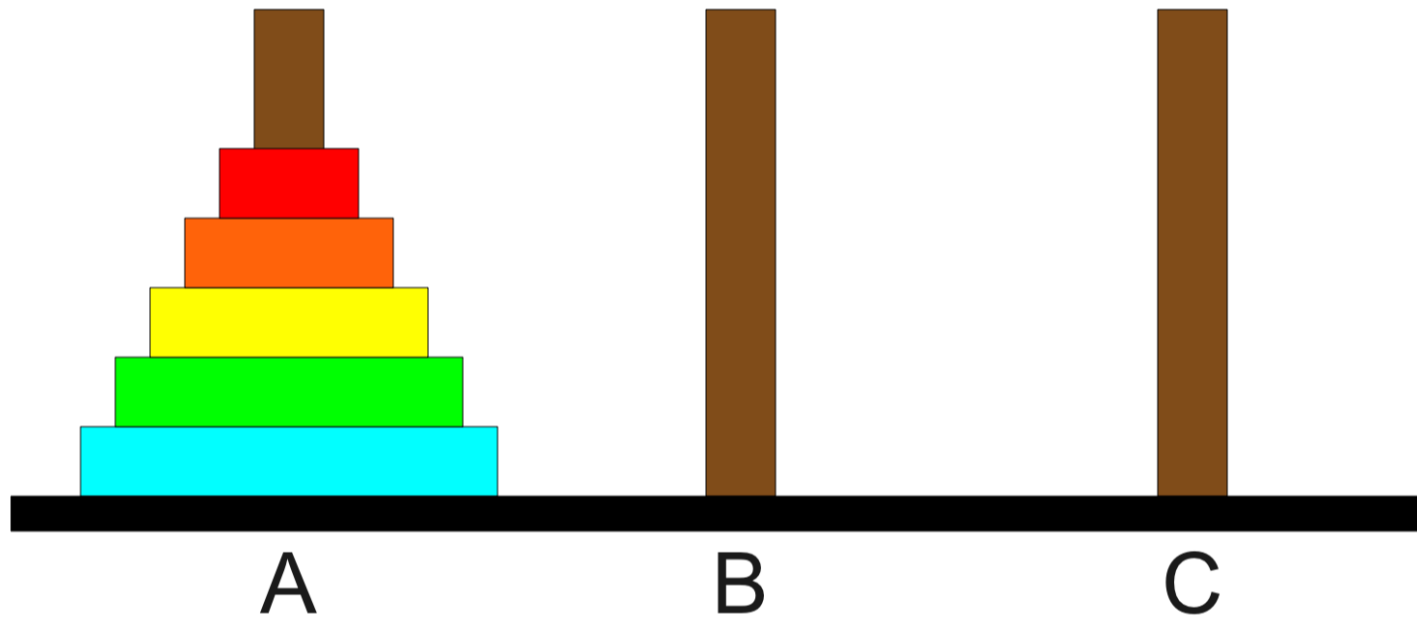
**Caching:** possiamo salvare i valori di `fibonacci()` già calcolati

- usiamo una *unordered\_map* della forma  $\{n : F_n\}$ 
  - la ricerca e l'inserimento in una *unordered\_map* hanno complessità  $O(1)$
- usiamo una *map* della forma  $\{n : F_n\}$ 
  - la ricerca e l'inserimento in una *map* hanno complessità  $O(\log(n))$

Pseudocodice:

```
unsigned long long int fibonacci(int n) {  
    se n==0 oppure n==1 → return n;           \\ base case  
    se n è nella map → return map[n];         \\ leggo la cache  
    map[n] = fibonacci(n - 1) + fibonacci(n - 2);  
    return map[n];  
}
```

## Esempio #3: le torri di Hanoi



Risolvete lo interattivamente!

<https://discrete-math-puzzles.github.io/puzzles/hanoi-towers/index.html>



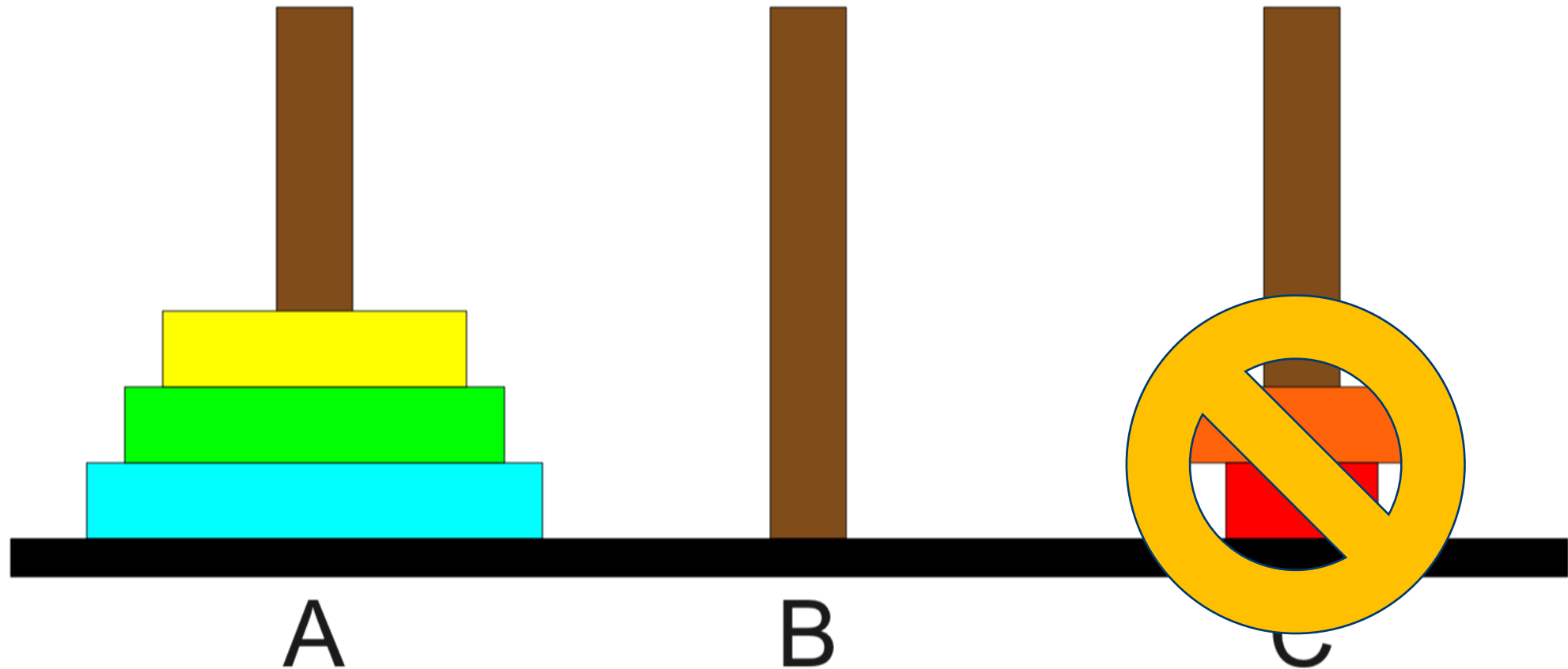
# Esempio #3: le torri di Hanoi

**Scopo del gioco:** spostare il cono di dischi dal paletto di sinistra a quello di destra.

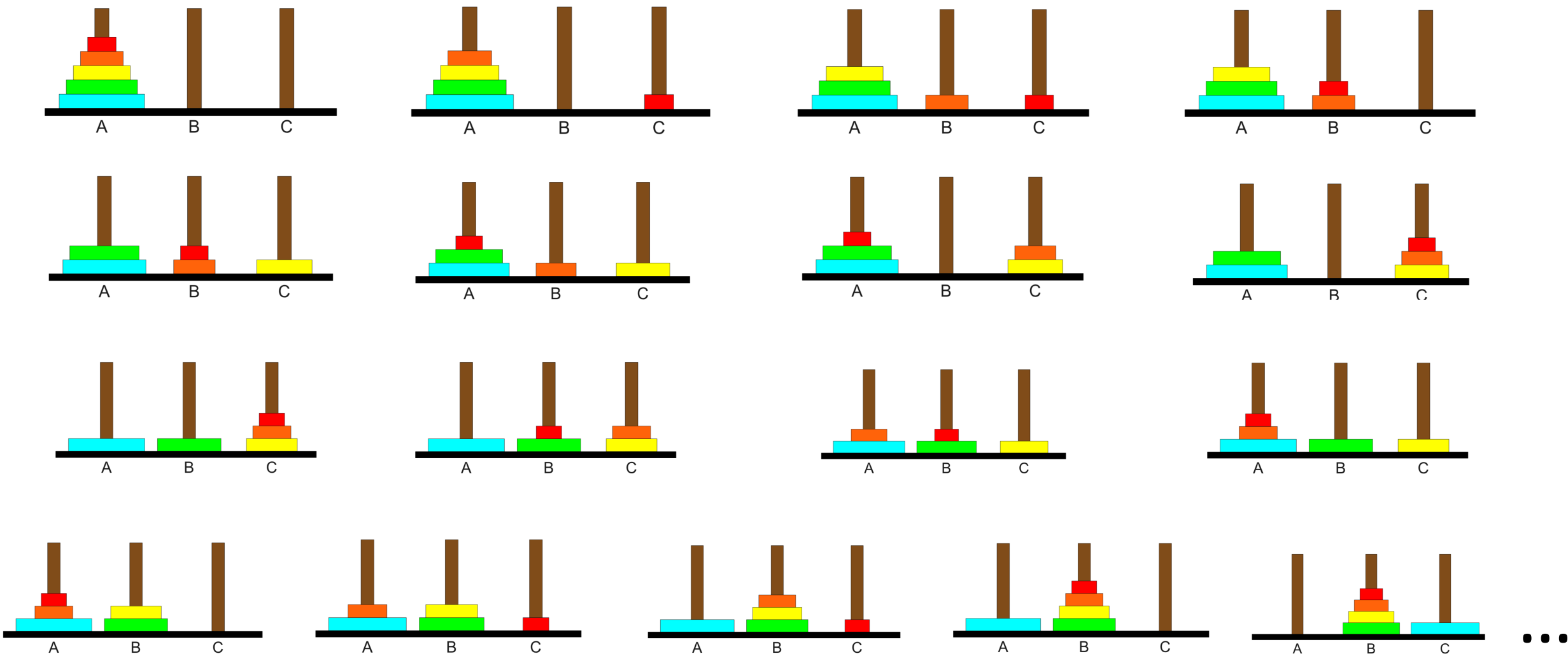
**Regole del gioco:**

1. si può spostare solo un disco alla volta
2. si può spostare solo il disco che sta sulla sommità della pila di dischi e lo si può sistemare
  - sulla sommità di un'altra pila di dischi, oppure
  - su un paletto vuoto
3. si può mettere un disco solo sopra un disco più grande, mai su uno più piccolo

# Configurazioni proibite



# Soluzione step-by-step

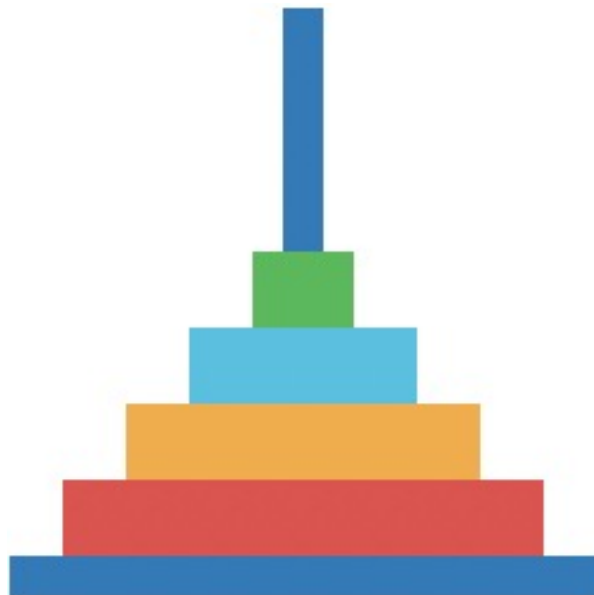


# Soluzione ricorsiva

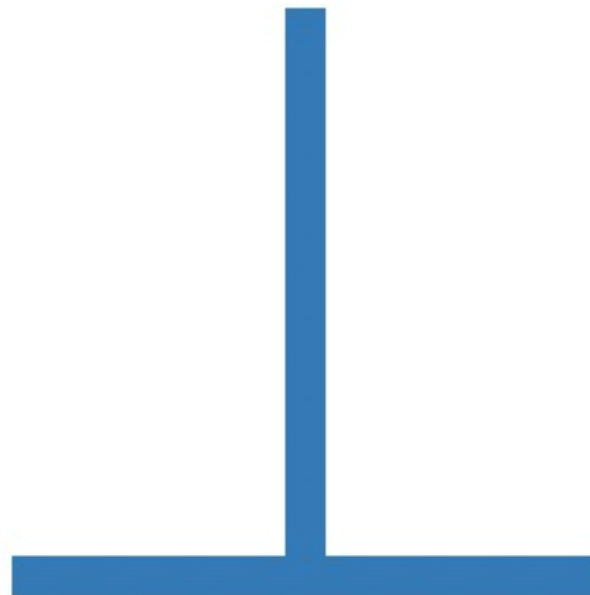
```
towerOfHanoi (int n, char from_stick, char to_stick, char aux_stick)
```

- `n=4`, il numero di dischi
- `from_stick = 'A'`, il paletto di partenza
- `to_stick = 'C'`, il paletto destinazione
- `aux_stick = 'B'`, il paletto ausiliario

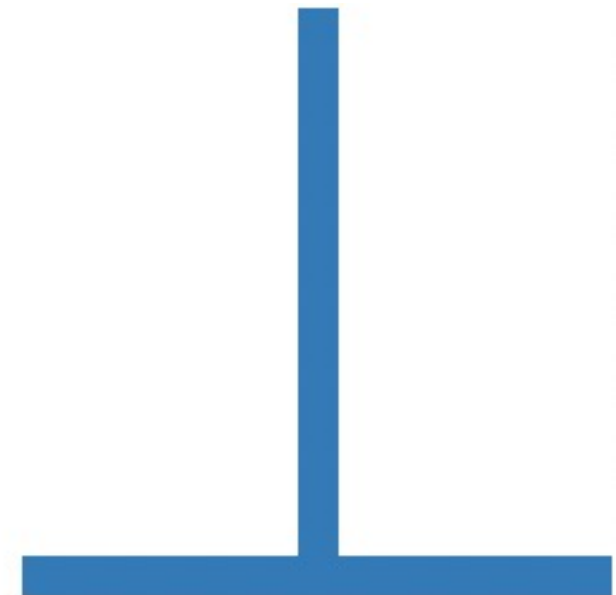
```
towerOfHanoi (4, 'A', 'C', 'B')
```



A



B

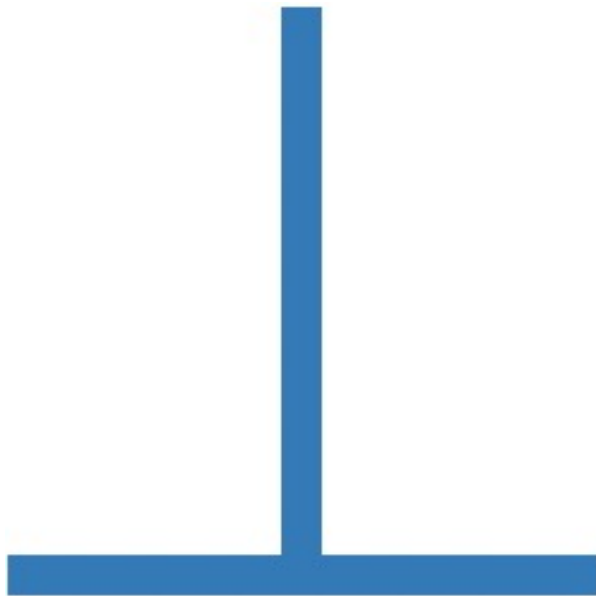


C

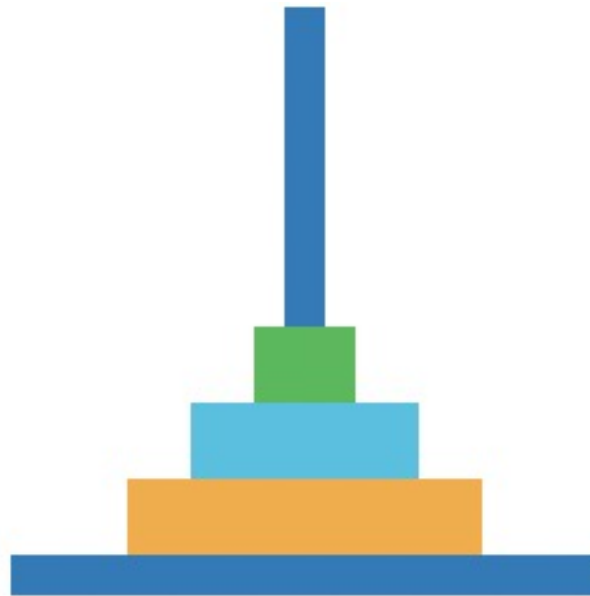
# Soluzione ricorsiva

Dividiamo il problema in sotto-problemi:

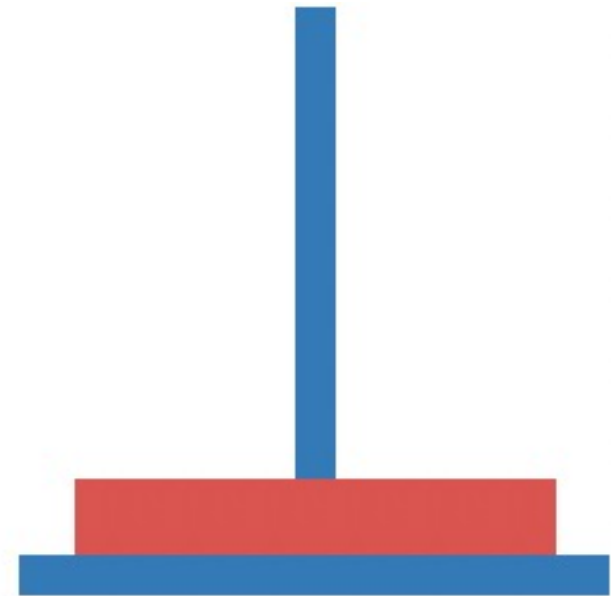
1. Prendere i primi tre dischi e spostarli su B
2. Prendere il disco rosso e spostarlo su C
3. Prendere i tre dischi da B e spostarli su C



A



B

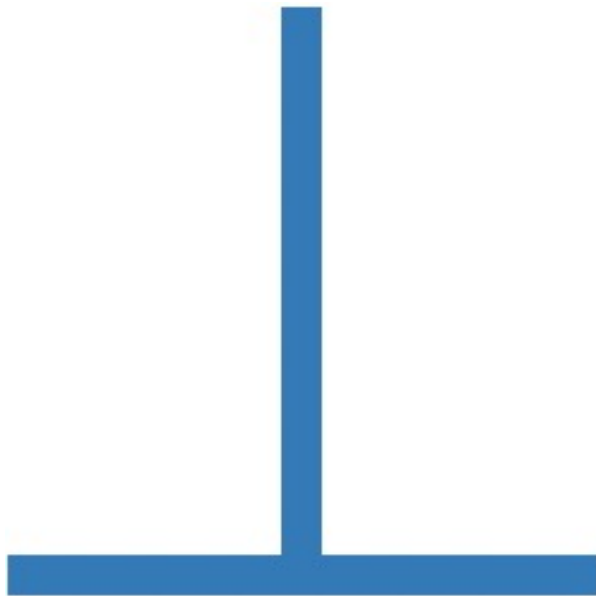


C

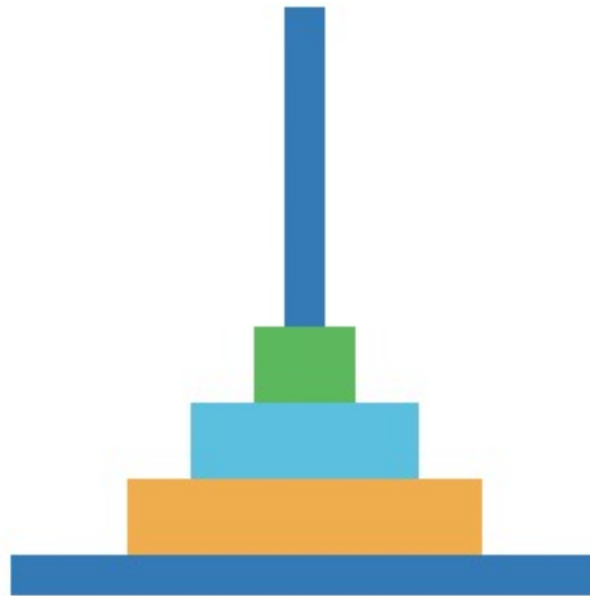
# Soluzione ricorsiva

## Attenzione:

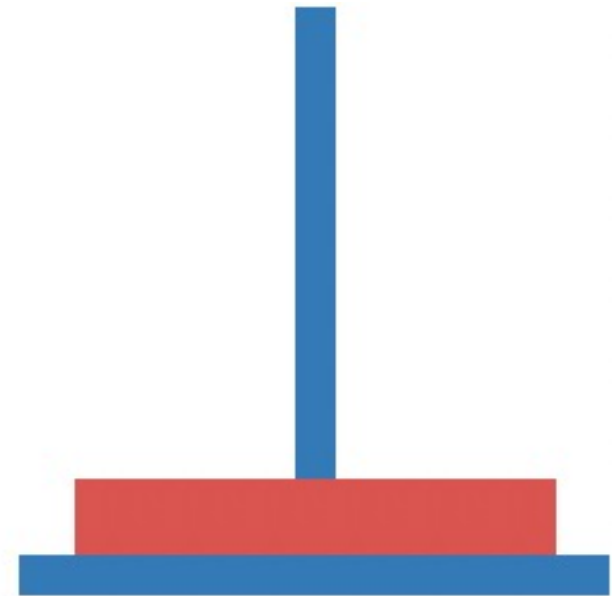
- nello spostare i 4 dischi, ci si pone il problema di spostare 3 dischi!
- spostamento dei tre dischi da A a B e poi da B a C dopo aver mosso il disco rosso
- ... e come faccio a spostare i 3 dischi? ... spostando prima i primi due ... do you see where I'm going?



A



B



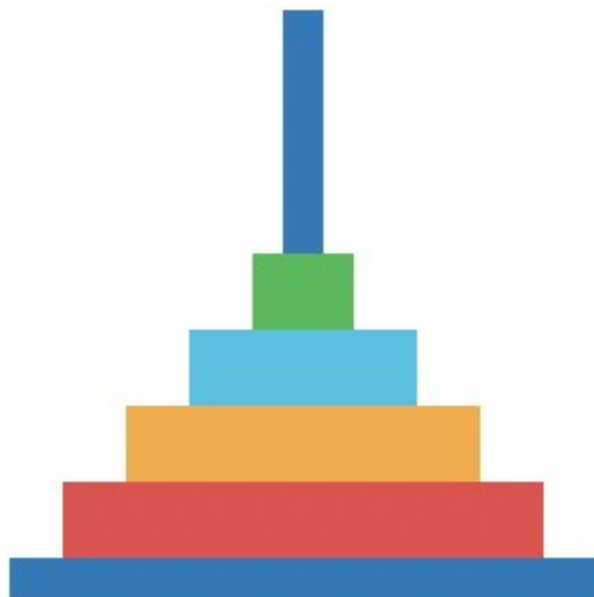
C

# Soluzione ricorsiva

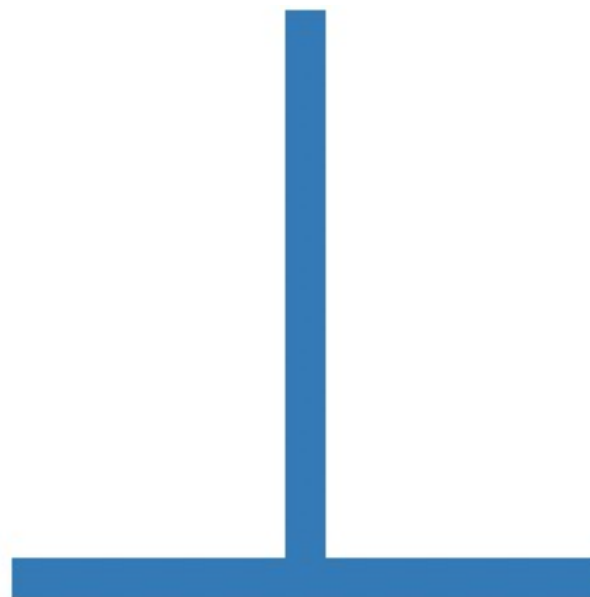
```
towerOfHanoi(4, 'A', 'C', 'B')  
{towerOfHanoi(3, 'A', 'B', 'C')}
```

rosso da A a C

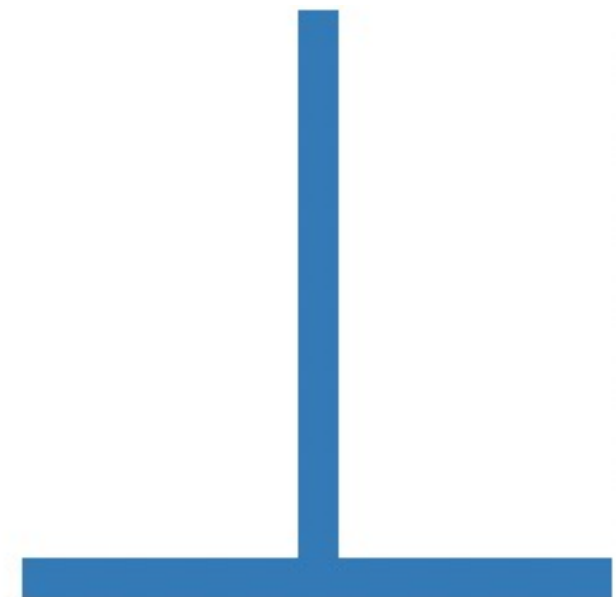
```
towerOfHanoi(3, 'B', 'C', 'A')}
```



A



B



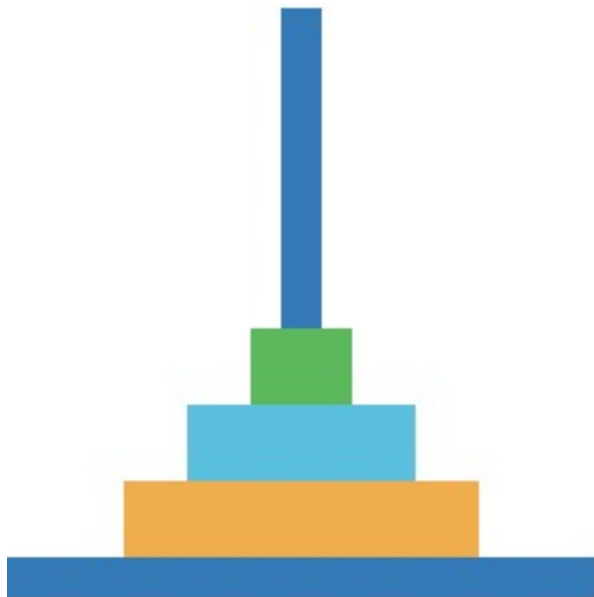
C

# Soluzione ricorsiva

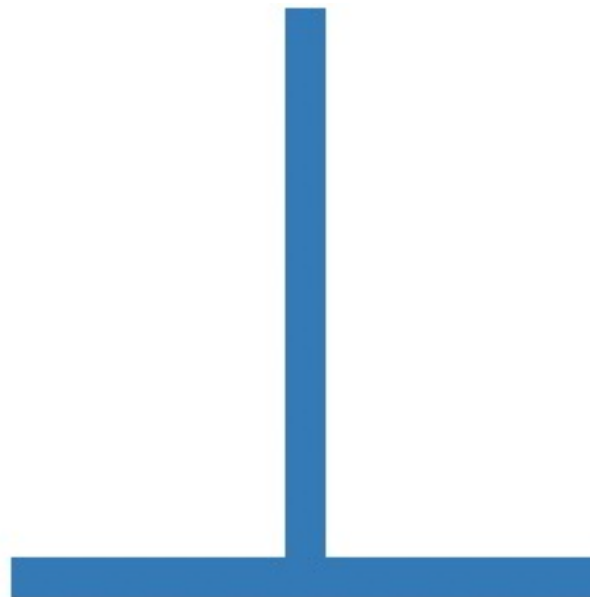
```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  }}
```

rosso da A a C  
arancio da A a B

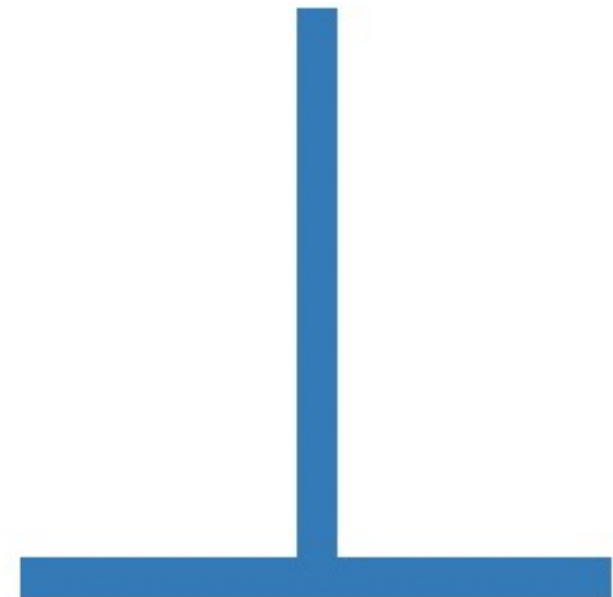
```
towerOfHanoi(3, 'B', 'C', 'A') }  
towerOfHanoi(2, 'C', 'B', 'A') }
```



A



B



C

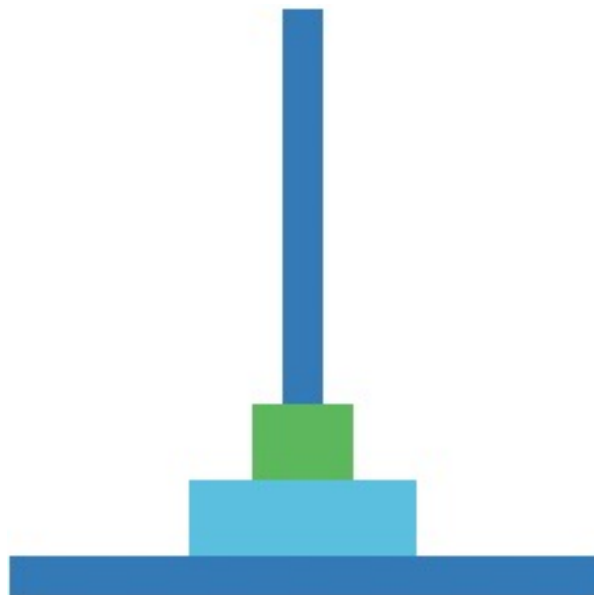


# Soluzione ricorsiva

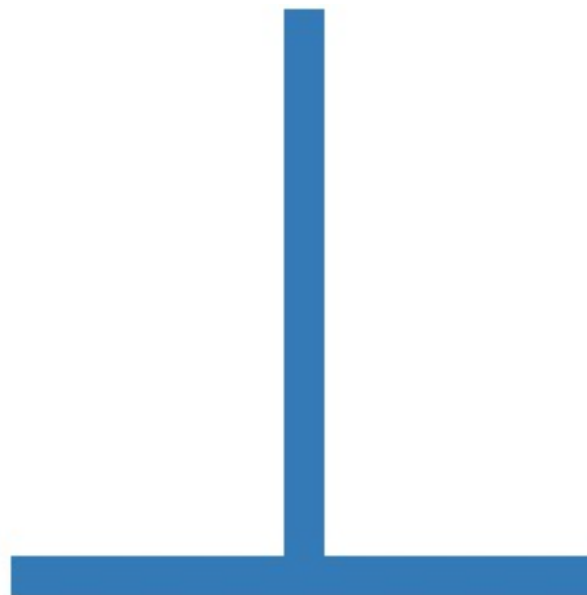
```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  
    {  
      towerOfHanoi(1, 'A', 'B', 'C')    }  
  }  
}
```

rosso in C  
arancio in B  
azzurro in C

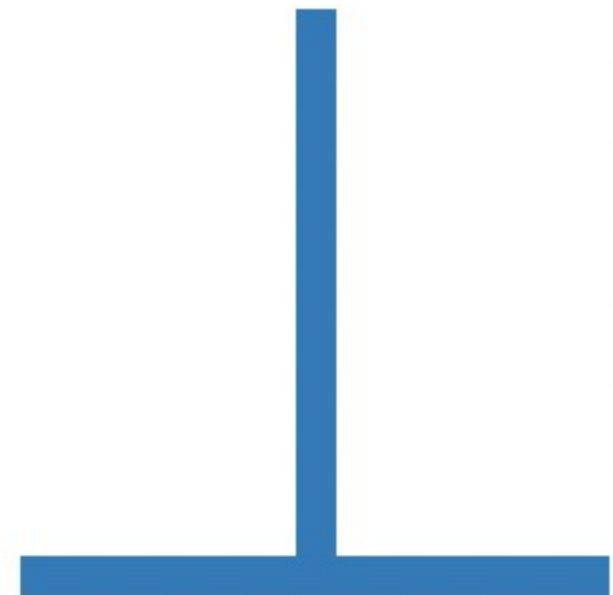
```
towerOfHanoi(3, 'B', 'C', 'A')}  
towerOfHanoi(2, 'C', 'B', 'A')}  
towerOfHanoi(1, 'B', 'C', 'A')}
```



A



B

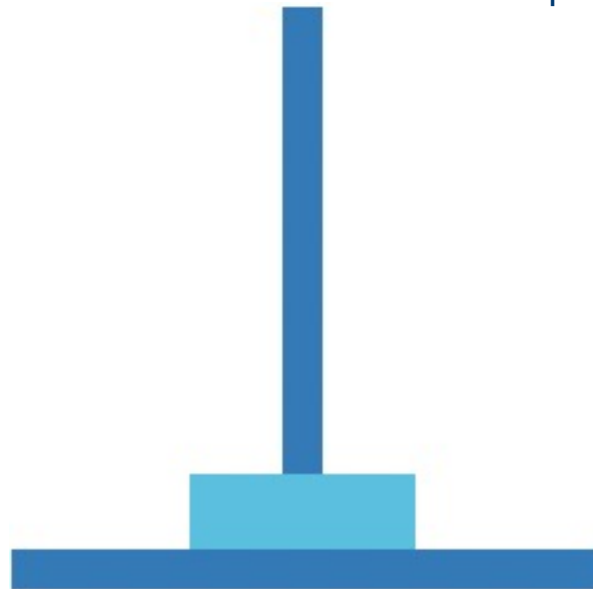


C

# Soluzione ricorsiva

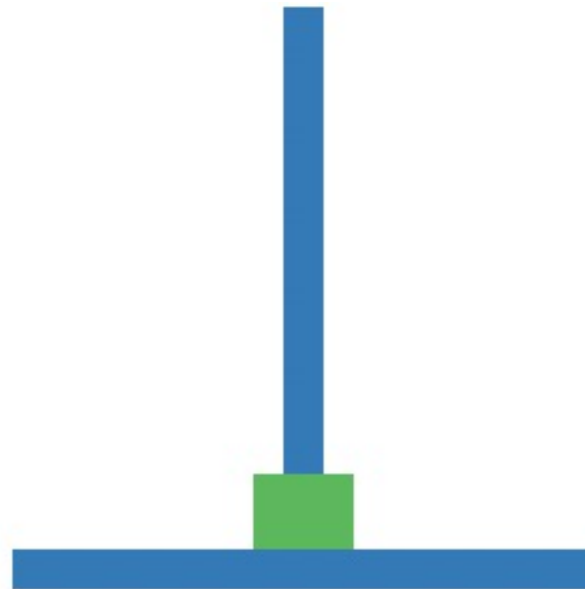
```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  
    {  
      towerOfHanoi(1, 'A', 'B', 'C')  
    }  
  }  
}
```

Abbiamo un solo disco. Prendiamo e spostiamolo.



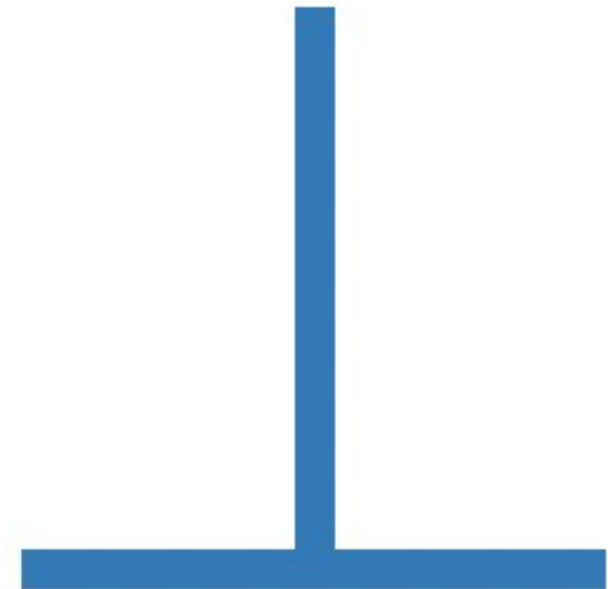
A

rosso in C  
arancio in B  
azzurro in C



B

```
towerOfHanoi(3, 'B', 'C', 'A')}  
towerOfHanoi(2, 'C', 'B', 'A')}  
towerOfHanoi(1, 'B', 'C', 'A')}
```



C

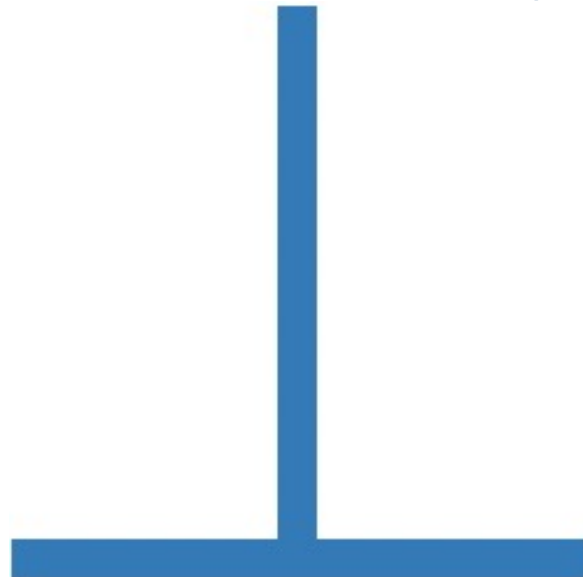
# Soluzione ricorsiva

```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  
    {  
      towerOfHanoi(1, 'A', 'B', 'C')    }  
  }  
}
```

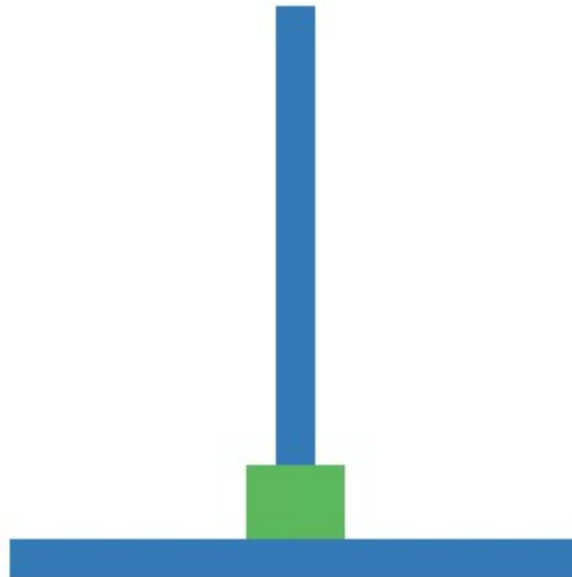
rosso in C  
arancio in B  
azzurro in C

```
towerOfHanoi(3, 'B', 'C', 'A')}  
towerOfHanoi(2, 'C', 'B', 'A')}  
towerOfHanoi(1, 'B', 'C', 'A')}
```

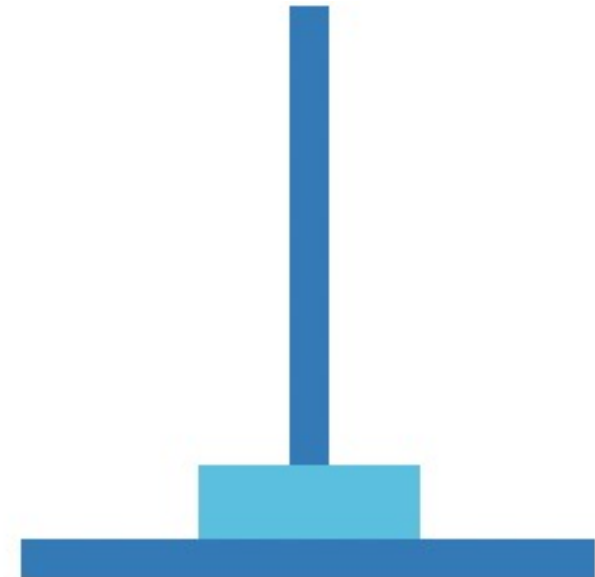
Abbiamo un solo disco. Prendiamo e spostiamolo.



A



B



C

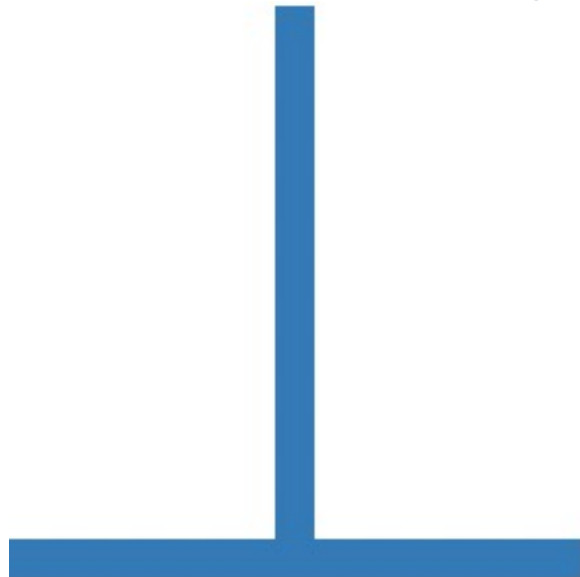
# Soluzione ricorsiva

```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  
    {  
      towerOfHanoi(1, 'A', 'B', 'C')    }  
  }  
}
```

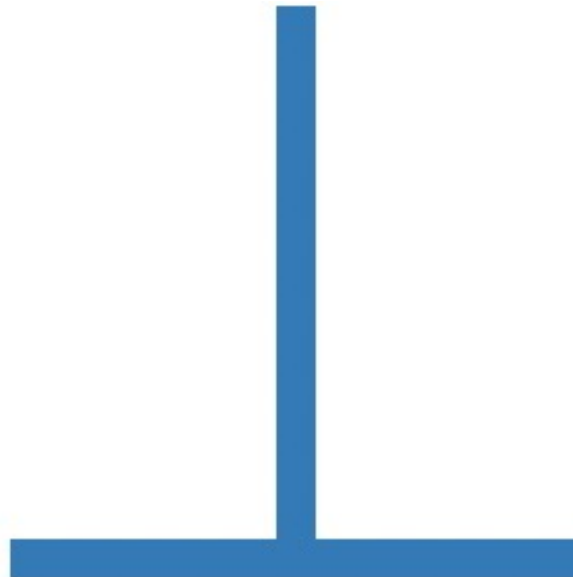
rosso in C  
arancio in B  
azzurro in C

```
towerOfHanoi(3, 'B', 'C', 'A') }  
towerOfHanoi(2, 'C', 'B', 'A') }  
towerOfHanoi(1, 'B', 'C', 'A') }
```

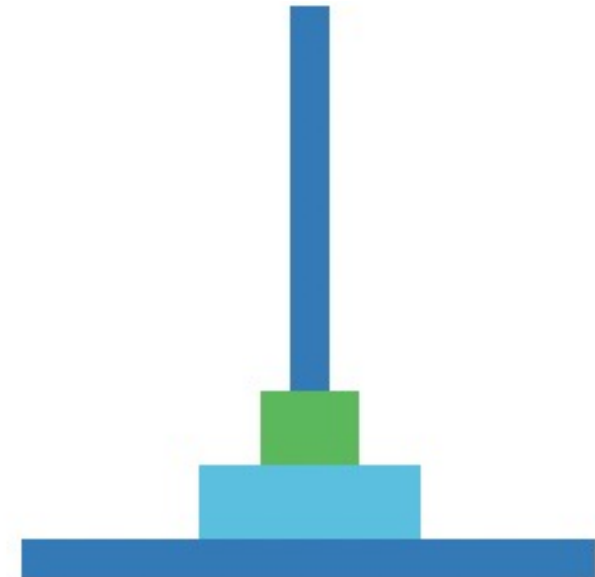
Abbiamo un solo disco. Prendiamo e spostiamolo.



A



B



C

# Soluzione ricorsiva

```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  }}
```

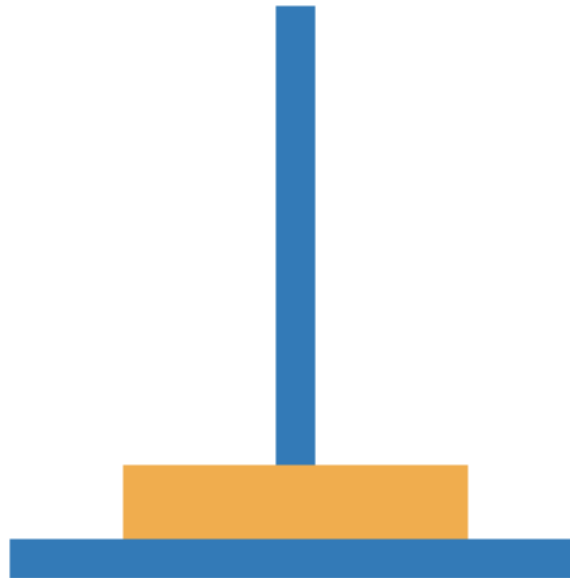
rosso in C  
arancio in B

```
towerOfHanoi(3, 'B', 'C', 'A') }  
towerOfHanoi(2, 'C', 'B', 'A') }
```

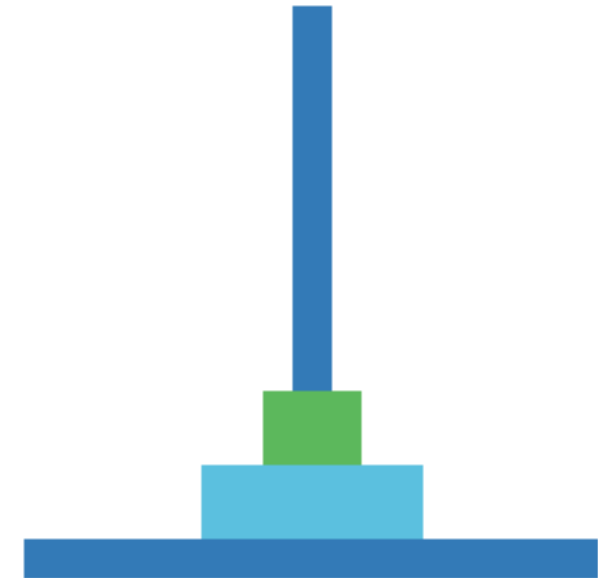
Abbiamo un solo disco. Prendiamo e spostiamolo.



A



B



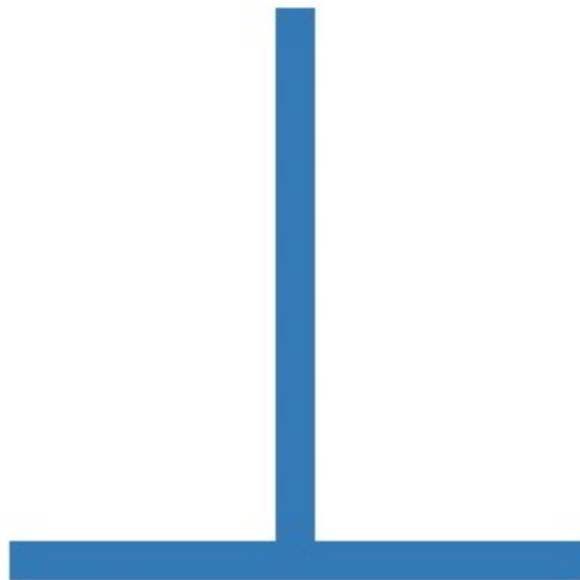
C

# Soluzione ricorsiva

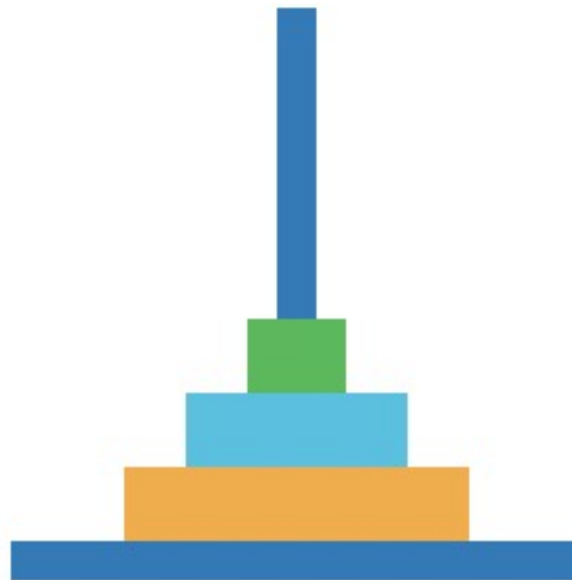
```
towerOfHanoi(4, 'A', 'C', 'B')  
{  
  towerOfHanoi(3, 'A', 'B', 'C')  
  {  
    towerOfHanoi(2, 'A', 'C', 'B')  }}
```

rosso in C  
arancio in B

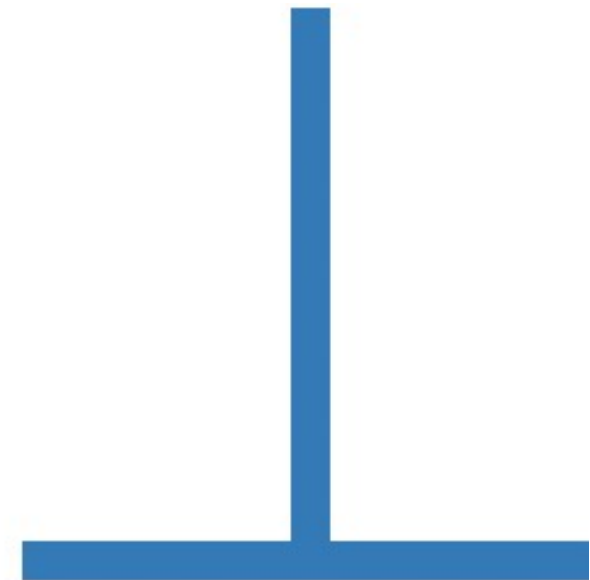
```
towerOfHanoi(3, 'B', 'C', 'A') }  
towerOfHanoi(2, 'C', 'B', 'A') }
```



A



B



C

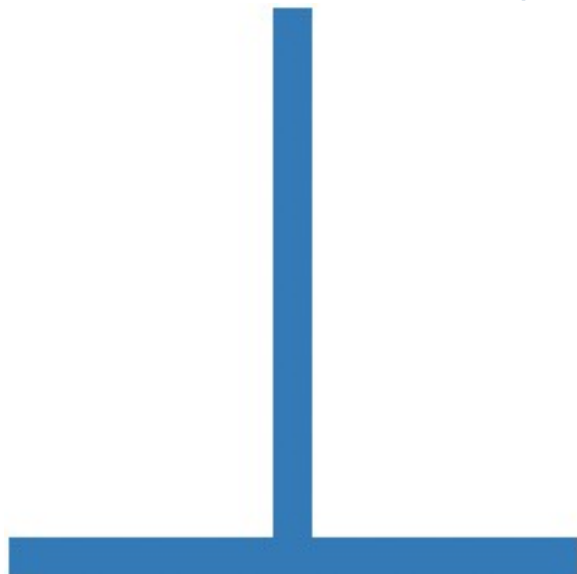
# Soluzione ricorsiva

```
towerOfHanoi(4, 'A', 'C', 'B')  
{towerOfHanoi(3, 'A', 'B', 'C')}
```

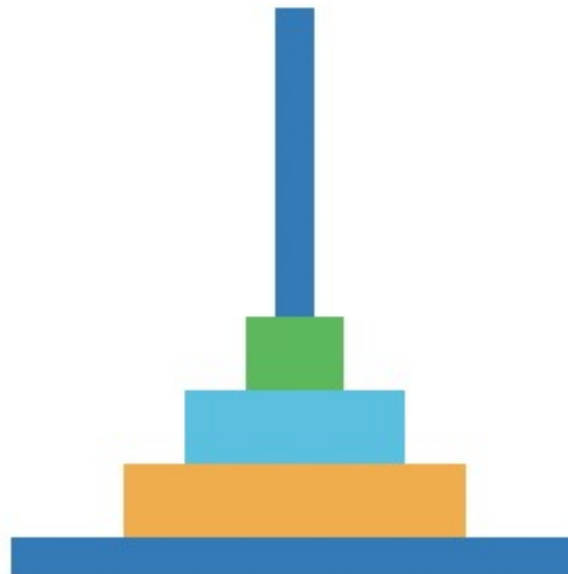
rosso in C

```
towerOfHanoi(3, 'B', 'C', 'A')}
```

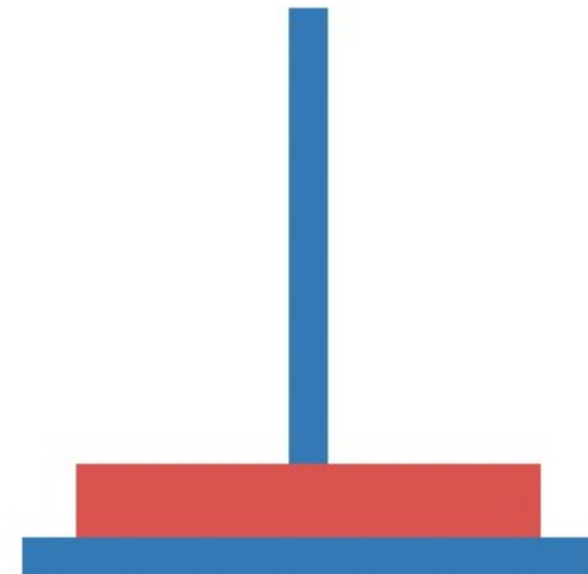
Abbiamo un solo disco. Prendiamo e spostiamolo.



A



B



C

# Soluzione ricorsiva

```
towerOfHanoi(4, 'A', 'C', 'B')  
{towerOfHanoi(3, 'A', 'B', 'C')}
```

rosso in C

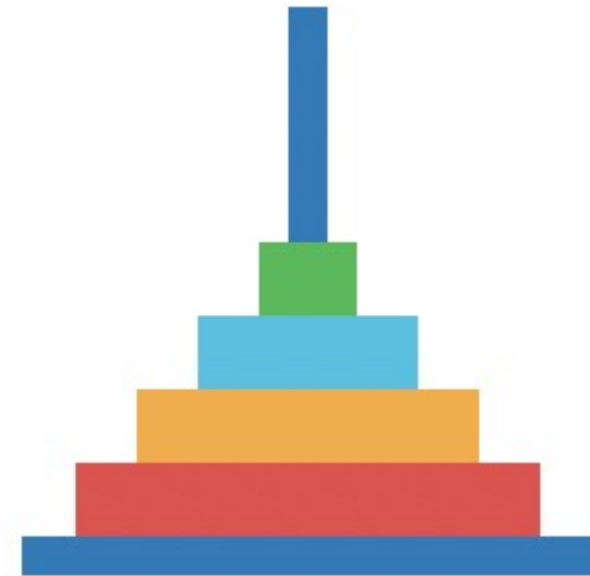
towerOfHanoi(3, 'B', 'C', 'A')



A



B



C



# Un'implementazione elegante (e sintetica!)

```
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 0) {
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod <<
        " to rod " << to_rod << endl;
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}
```

# Quando usare la ricorsione?

- Problemi di tipo “**divide et impera**” (next): la soluzione si ottiene suddividendo il problema in due o più sottoproblemi simili, che vengono risolti separatamente ricombinando poi le soluzioni per ottenere la soluzione al problema di partenza
- Soluzioni “**a forza bruta**”: per generare tutti i casi possibili tramite **backtracking** (ad esempio, tutti i sottoinsiemi, tutte le permutazioni...) e scegliere il caso ottimo per il problema in questione
- Problemi di **programmazione dinamica** (prossima lezione)

# Problemi con la ricorsione

## 1. Ricorsione infinita

- Cosa accade se manca il caso base o se la funzione viene richiamata con parametri non corretti?
- Ad esempio, cosa restituisce `factorial(-1)`?

## 2. Non è immediato calcolare il tempo di esecuzione di funzioni ricorsive

- Si richiedono tecniche matematiche sofisticate (equazioni di ricorrenza)

## 3. Siate cauti con le chiamate ricorsive: fare più di una chiamata **potrebbe** far aumentare di molto i tempi di esecuzione (anche in modo esponenziale, e.g., $2^n$ ). Un esempio? Fibonacci!

- Valutate sempre i vincoli del problema (dimensione delle istanze di input)

# Problema

Mappa Antica (Territoriali 2008)

<https://training.olinfo.it/#/task/mappa/statement>



## Descrizione del problema

Topolino è in missione per accompagnare una spedizione archeologica che segue un'antica mappa acquisita di recente dal museo di Topoinia. Raggiunta la località dove dovrebbe trovarsi un prezioso e raro reperto archeologico, Topolino si imbatte in un labirinto che ha la forma di una gigantesca scacchiera quadrata di  $N \times N$  lastroni di marmo.

Nella mappa, sia le righe che le colonne del labirinto sono numerate da 1 a  $N$ . Il lastrone che si trova nella posizione corrispondente alla riga  $r$  e alla colonna  $c$  viene identificato mediante la coppia di interi  $(r, c)$ . I lastroni segnalati da una crocetta '+' sulla mappa contengono un trabocchetto mortale e sono quindi da evitare, mentre i rimanenti sono innocui e segnalati da un asterisco '\*'.

Topolino deve partire dal lastrone in posizione  $(1, 1)$  e raggiungere il lastrone in posizione  $(N, N)$ , entrambi innocui. Può passare da un lastrone a un altro soltanto se questi condividono un lato o uno spigolo (quindi può procedere in direzione orizzontale, verticale o diagonale ma non saltare) e, ovviamente, questi lastroni devono essere innocui.

Tuttavia, le insidie non sono finite qui: per poter attraversare incolume il labirinto, Topolino deve calpestare il minor numero possibile di lastroni innocui (e ovviamente nessun lastrone con trabocchetto). Aiutate Topolino a calcolare tale numero minimo.

## File di input

Il programma deve leggere da un file di nome `input.txt`. La prima riga contiene un intero positivo che rappresenta la dimensione  $N$  di un lato del labirinto a scacchiera. Le successive  $N$  righe rappresentano il labirinto a scacchiera: la  $r$ -esima di tali righe contiene una sequenza di  $N$  caratteri '+' oppure '\*', dove '+' indica un lastrone con trabocchetto mentre '\*' indica un lastrone sicuro. Tale riga rappresenta quindi i lastroni che si trovano sulla  $r$ -esima riga della scacchiera: di conseguenza, il  $c$ -esimo carattere corrisponde al lastrone in posizione  $(r, c)$ . I caratteri NON sono separati da degli spazi.

## File di output

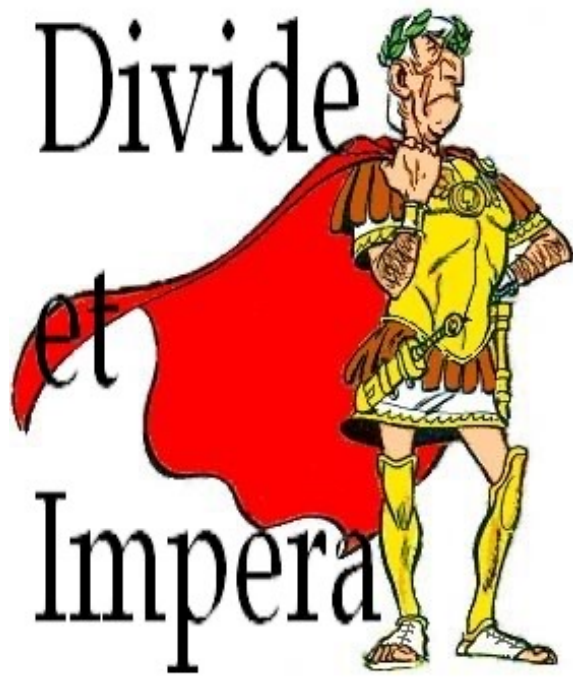
Il programma deve scrivere in un file di nome `output.txt`. Deve venire stampato il minimo numero di lastroni innocui (ossia indicati con '\*') che Topolino deve attraversare a partire dal lastrone in posizione  $(1, 1)$  per arrivare incolume al lastrone in posizione  $(N, N)$ . Notare che i lastroni  $(1, 1)$  e  $(N, N)$  vanno inclusi nel conteggio dei lastroni attraversati.

## Esempio di input/output

File input.txt	File output.txt
5 ***** +****+ *++++* +****+ +****+	5

# Divide et impera

in inglese, *divide & conquer*



# Approccio

Struttura ricorsiva:

1. **Dividere** il problema in sottoproblemi simili all'originale, ma più piccoli
2. **Conquistare** i sottoproblemi risolvendoli ricorsivamente. Se sono abbastanza piccoli, risolverli in modo diretto
3. **Combinare** le soluzioni dei sottoproblemi per ottenere la soluzione del problema originale



# Esempio #1: MergeSort

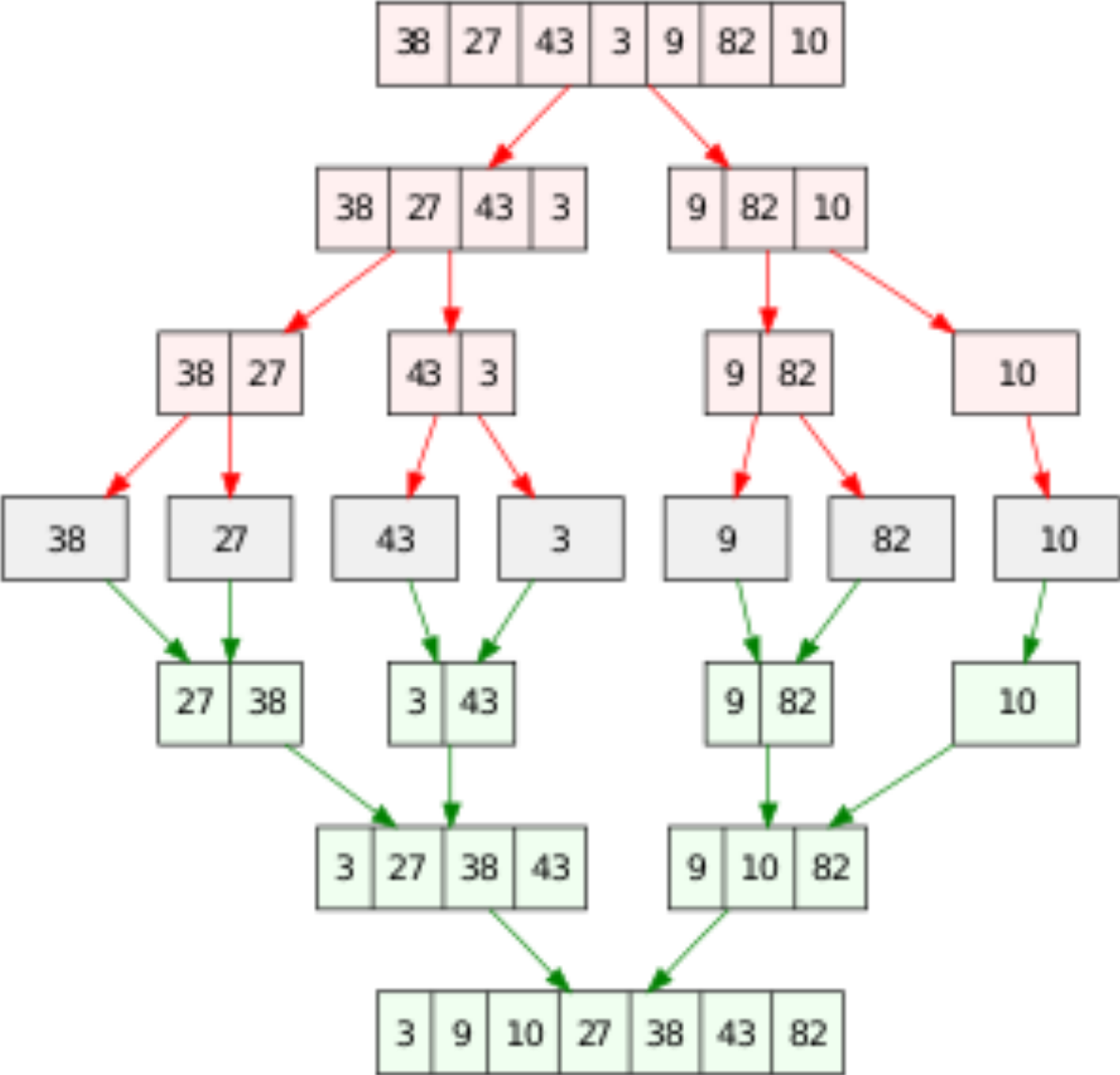
Il problema: ordinare una sequenza di  $n$  elementi

L'algoritmo:

1. **Dividere** la sequenza lunga  $n$  in due sottosequenze lunghe  $\lceil n/2 \rceil$  e  $\lfloor n/2 \rfloor$
2. **Ordinare** le due sottosequenze ricorsivamente usando mergesort (a meno che non abbiano un solo elemento)
3. **Combinare** le soluzioni **fondendo** le due sottosequenze ordinate in una sequenza ordinata lunga  $n$

Nota: la fusione di due sequenze ordinate è un problema facile che può essere risolto in tempo lineare

# Graficamente



# Implementazione

```
/* s e d sono gli indici sinistro e destro del
   sottoarray da ordinare */
void mergeSort(int arr[], int s, int d) {
    if (s < d) {
        int m = (s+d)/2;           // divide
        mergeSort(arr, s, m);      // conquer
        mergeSort(arr, m+1, d);    // conquer
        fondi(arr, s, m, d);       // combine
    }
}
```

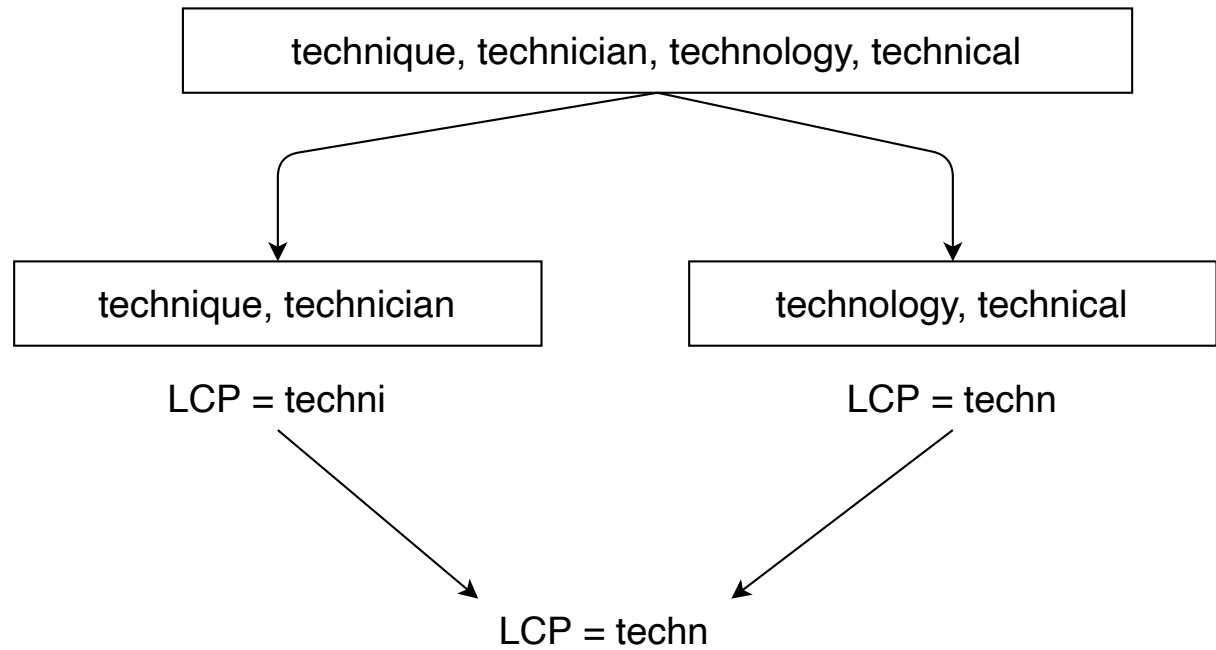
# Esempio #2: Longest Common Prefix

Il problema: trovare il prefisso comune più lungo dato un array di stringhe

L'algoritmo:

1. **Dividere** la sequenza lunga  $n$  in due sottosequenze lunghe  $\lceil n/2 \rceil$  e  $\lfloor n/2 \rfloor$
2. **Calcolare** ricorsivamente il LCP delle due sottosequenze (a meno che non abbiano un solo elemento)
3. **Combinare** i LCP fino a trovare il LCP dell'intero array

# Graficamente



# Problema

Ordinamento a paletta (OI 2015)

<https://training.olinfo.it/#/task/paletta/statement>



Romeo ha di recente assistito ad un'avvincente performance di un cuoco acrobatico, che gestiva una imponente grigliata di braciole ribaltandole a gruppi di tre per mezzo di una apposita paletta. Questo evento gli ha ispirato l'idea del *paletta-sort*, una nuova interessante procedura di ordinamento.

Dato un vettore  $V$  contenente gli interi da  $0$  a  $N-1$  (indicizzato da  $0$  a  $N-1$ ), l'unica operazione ammessa nel *paletta-sort* è l'operazione *ribalta*. Questa operazione sostituisce tre elementi  $A, B, C$  consecutivi di  $V$  con i corrispondenti ribaltati  $C, B, A$ . Aiuta Romeo a capire se è possibile ordinare il vettore  $V$ , e in caso affermativo quante e quali operazioni *ribalta* sono sufficienti!

### Esempi di input/output

input.txt	output.txt
5 2 0 4 3 1	-1
6 2 3 0 5 4 1	3

# Esercizi per casa

1. Domino massimale (ricorsione)

<https://training.olinfo.it/#/task/domino/statement>

2. Missioni segrete (ricorsione)

<https://training.olinfo.it/#/task/missioni/statement>

3. Controllo degli estintori (divide-et-impera)

[https://training.olinfo.it/#/task/ois\\_estintori/statement](https://training.olinfo.it/#/task/ois_estintori/statement)

4. Allenamento su ChinaForces (divide-et-impera)

[https://training.olinfo.it/#/task/preoii\\_allenamento/statement](https://training.olinfo.it/#/task/preoii_allenamento/statement)