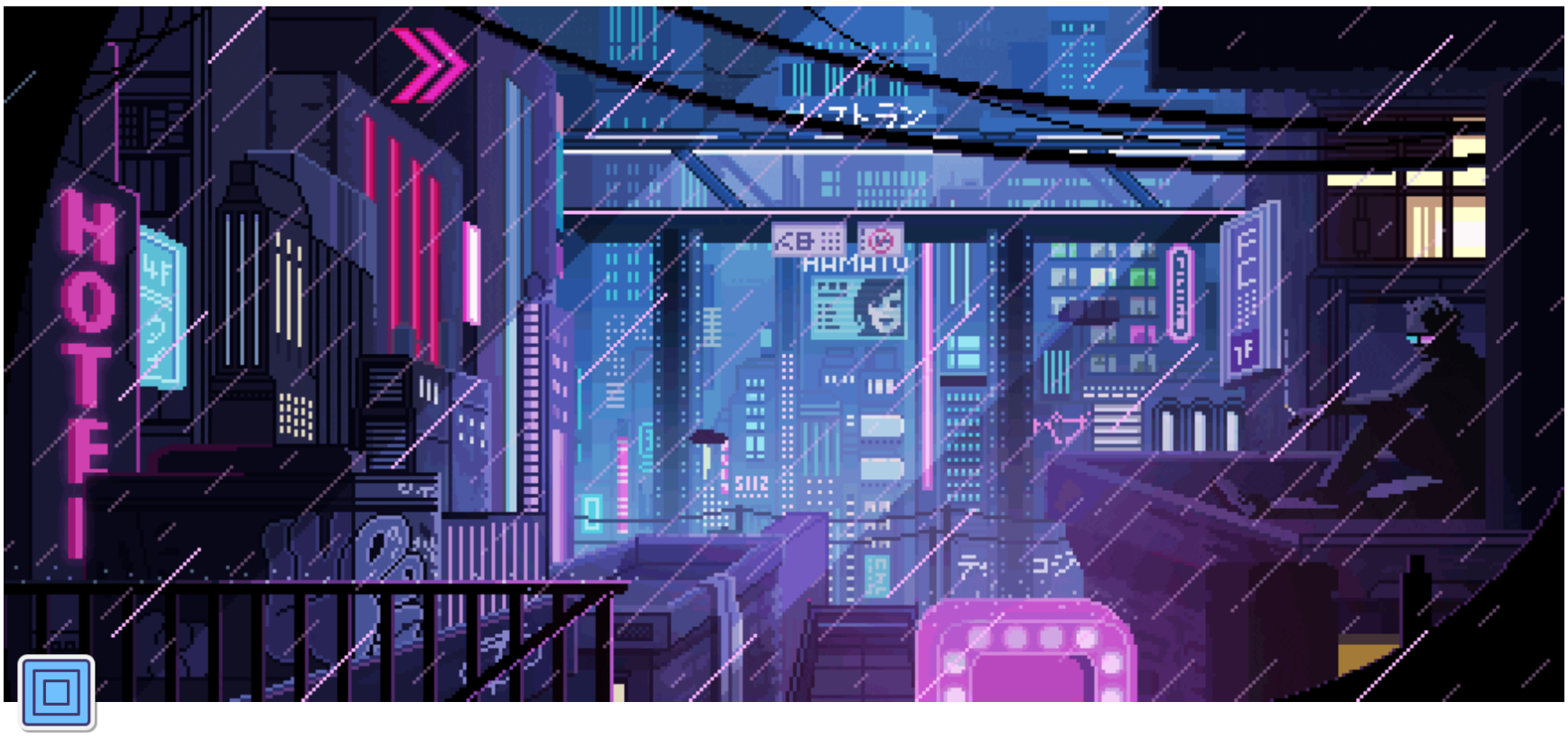


La ricorsione

▼ ***INDICE***

5 La ricorsione

Fino ad ora abbiamo studiato solo l'algoritmi in forma iterativi, ovvero che si basano sull'uso di cicli iterativi(come ciclo for, while, …).

Adesso andiamo a vedere l'uso di algoritmi ricorsivi, per capire meglio come strutturare un algoritmo in modo ricorsivo prendiamo come esempio la ricerca binaria.

Vediamo la seguente riformulazione ricorsiva dell'algoritmo:

- Ispeziona l'elemento centrale dell'array;
- Se è uguale a v restituisci il suo codice;
- Se è maggiore di v, riduci l'array alla sua metà inferiore ed ri-esegui la ricerca binaria su di essa;
- Se è minore di v, riduci l'array alla sua metà superiore e ri-esegui la ricerca binaria su di essa.

Come possiamo notare l'algoritmo risolve il problema riapplicando se stesso su un sotto-problema, ossia una versione "più semplice" di esso(nel nostro caso la versione più semplice è la metà dell'array).

Le funzioni ricorsive possono essere trovate anche nell'ambito matematico, per esempio il fattoriale di numero:

$$n! \begin{cases} n \cdot (n-1)! & \text{se } n > 0 \\ n & \text{se } n = 0 \end{cases}$$

$$n! = n \cdot (n-1)! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1! \cdot 0! = n \cdot ... \cdot 1.$$

Da come possiamo vedere il fattoriale di n, sarà il prodotto tra n stesso e il fattoriale di $(n-1)$, a sua volta composto tra $(n-1)$ e il fattoriale di $(n-2)$. Tale catena viene ripetuta fino a raggiungere il fattoriale di 0(che equivale ad 1).

In questo caso 0 sarà il nostro caso base del problema, ossia il sotto problema minimo.

La presenza di un caso base all'interno di un algoritmo è fondamentale, poiché senza di esso la ricorsione continuerebbe all'infinito.

> ✏️ ***DEFINIZIONE ALGORITMO RICORSIVO:***
>
> Un algoritmo ricorsivo ha sempre le seguenti proprietà:
>
> - La soluzione del sotto problema complessivo viene data risolvendo uno o più sotto-problemi di dimensione minore, per poi combinare le soluzioni ottenute;
> - La successione dei sotto-problemi, che sono sempre più piccoli, deve sempre convergere ad un sotto-problema che sostituisce un caso base, il quale va a terminare la ricorsione.

Andiamo ora a vedere qualche esempio di ricorsione:

📎 ***ESEMPIO DI CODICE RICORSIVO:***

Algoritmo ricorsivo per il calcolo del fattoriale di n:

```
def fattoriale(n):   #n = intero non negativo
  if(n==0): return 1 #caso base
  return n * fattoriale(n-1)
```

📎 ***ESEMPIO RICERCA SEQUENZIALE:***

Algoritmo ricorsivo della ricerca sequenziale:

```
def Ricerca_seq_ric(A, v, n=len(A)-1):
  if (A[n]==v):
    return n
  if (n==0):
    return -1
  else:
    return Ricerca_seq_ric(A, v, n-1)
```

📎 ***ESEMPIO RICERCA BINARIA:***

Algoritmo ricorsivo della ricerca binaria:

```
def Ricerca_bin_ric(A, v, i_min=0, i_max=len(A)):
  if(i_min>i_max):
    reuturn -1
  m = (i_min+i_max)//2
  if (A[m]==v):
    return m
  elif (A[m]==v):
    return Ricerca_bin_ric(A, v, i_min, m-1)
  else:
    return Ricerca_bin_ric(A, v, m+1, i_max)
```

5.1 Iterazione vs Ricorsione

Come abbiamo appena visto possiamo risolvere l'algoritmo iterativo usando la ricorsione.

A questo punto la domanda però sorge spontanea:

Perché preferire la ricorsione all'iterazione e viceversa?

Per capire ciò andiamo a vedere i seguenti punti:

1. Un algoritmo ricorsivo torna decisamente utile per esprimere la soluzione del problema in modo coerente con la sua natura(es. il fattoriale visto in precedenza), invece la soluzione iterativa, in questi casi, può risultare poco intuitiva e più complicata;
2. Usiamo un algoritmo iterativo solo se la sua versione ricorsiva non è altrettanto semplice e chiara. In altre parole, scegliamo l'approccio iterativo solo se non esiste una versione ricorsiva altrettanto chiara e semplice;
3. Utilizziamo un algoritmo iterativo quando vogliamo ottenere risultati rapidi e ottimizzati.

Ogni funzione, che sia ricorsiva o meno, richiede memoria per essere eseguita. La memoria viene utilizzata per:

- Calcolare il codice della funzione;
- Passare i parametri;
- Ritornare i valori calcolati.

Come bene sappiamo le funzioni ricorsive sono funzioni che richiamano se stesse, il che significa che hanno bisogno di molta memoria per poter essere eseguite.

Quindi come si può bene capire si preferisce sempre l'iterazione alla ricorsione, a meno che il problema non sia intuitivamente risolvibile con la ricorsione, come nel seguente esempio:

📎 ***ESEMPIO RICORSIONE INTUITIVA:***

Un esempio può essere il calcolo dell'n-esimo numero di Fibonacci, il quale può essere definito in modo ricorsivo in modo naturale:

- Se $n = 0$, il numero di Fibonacci sarà:

$$F(0) = 0$$

- Se $n = 1$, il numero di Fibonacci sarà:

$$F(1) = 1$$

- Altrimenti (se $n >= 1$) il numero di Fibonacci è la somma dei due numeri di Fibonacci precedenti, in altre parole:

$$F(n) = F(n-1) + F(n-2)$$

📖 ***NOTA:***

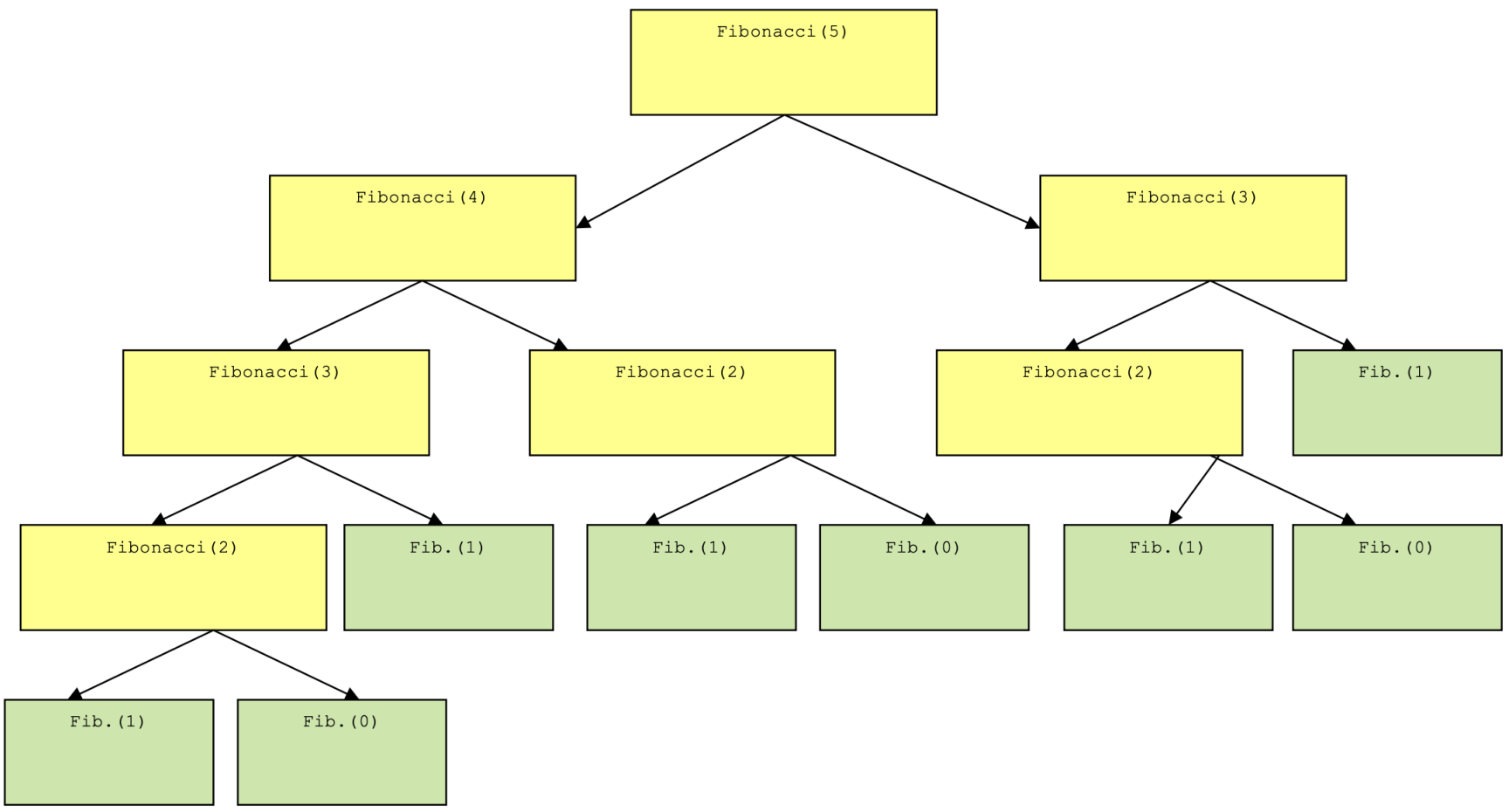
Questo può essere tradotto in un algoritmo ricorsivo la cui funzione stessa viene chiamata con argomenti più piccoli fino a quando non viene raggiunto il caso base($n = 0\ o\ n = 1$).

Ecco un esempio di codice in Python per calcolare l'n-esimo numero di Fibonacci utilizzano un approccio ricorsivo:

```
def Fibonacci(n):
  if n <= 1: return n  #0(1)
  return Fibonacci(n-1)+Fibonacci(n-2)  #T(n-1)+T(n-2)
```

Come visto dal precedente esempio una soluzione iterativa risulta molto più complessa da sviluppare, rispetto ad una soluzione ricorsiva.

Proviamo però ad analizzare lo sviluppo della catena di ricorsione nel caso avessimo come input $n = 5$:



Notiamo subito che, nonostante l'input di piccole dimensioni, le chiamate ricorsive effettuate siano molte, inoltre gli stessi calcoli vengono ripetuti più volte.

Andiamo dunque a vedere una versione iterativa dell'algoritmo:

```
def Fibonacci(n):
  if n <= 1: return n      #0(1)
  fib_0 = 0                #0(1)
  fib_1 = 1                #0(1)
  for i in range(2, n+1):  #0(n)
    fib_i = fib_0 + fib_1   #0(1)
    fib_0 = fib_1           #0(1)
    fib_1 = fib_i           #0(1)
  return fib_i             #0(1)
```

Possiamo quindi calcolare il costo computazionale dei 2 codici:

- Codice ricorsivo: considerando $T(n)$ come il costo computazionale della funzione, possiamo intuire che
  - Il costo del primo sotto-problema è $T(n-1)$;
  - Il costo del secondo sotto-problema è $T(n-2)$.

  Quindi:

$$T = \begin{cases} T(n) = \Theta(1) + T(n-1) + T(n-2) & \text{se} n > 1 \\ T(1) = \Theta(1) & \text{se } n \leq 1 \end{cases}$$

- Codice iterativo: in questo caso come ben sappiamo il costo computazionale sarà $\Theta(n)$.

Esercizi

🧮 ***Progettare una funzione ricorsiva che, dati due interi k ed n, $0 \leq k \leq n$, calcoli il valore del coefficiente binomiale n su k utilizzando la seguente relazione:***

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \; con \; \binom{n}{0} = \binom{n}{n} = 1$$

```
def Coefficiente_bin(k, n):
  if k==0 or k==n: return 1
  return Coefficiente_bin(k,n-1) + Coefficiente_bin(k-1,n-1)
```

Costo:

$$T = \begin{cases} T(n,0) = \Theta(1), T(n,n) = \Theta(1) \\ T(k,n) = \Theta(1) + T(k,n-1) + T(k-1,n-1) \end{cases}$$

🧮 ***Progettare un algoritmo ricorsivo che, dati due interi x e y, $x > y > 0$, ne calcoli il massimo comune divisore utilizzando il seguente procedimento (di Euclide):***

- ***se $y = 0$ allora $MCD(x,y) = x$;***
- ***altrimenti $MCD(x,y) = MCD(y, x\%y)$ dove $x\%y$ rappresenta il resto della divisione tra x ed y.***

```
def MCD(x,y):
  if y==0: return x
  return MCD(x%y,y)
```

Costo:

$$T = \begin{cases} T(x,0) = \Theta(1) \\ T(x,y) = \Theta(1) + T(x\%y, y) \end{cases}$$