3° lezione algoritmi

	Luigi Domenico Castano
Due Date	@October 8, 2024
Materia	Algoritmi e Laboratorio
Status	Done

PROBLEMI RICORSIVI

Quando si ha un probleema e non si conosce la soluzione l' approccio ricorsivo può essere un metodo efficace. Questo approccio consiste nel suddividere il problema in sottoproblemi più semplici, risolverli e poi combinare le soluzioni per ottenere il risultato finale. La ricorsione è particolarmente utile quando il problema ha una struttura che si ripete su scala più piccola.

IL FATTORIALE

F(n) → !n

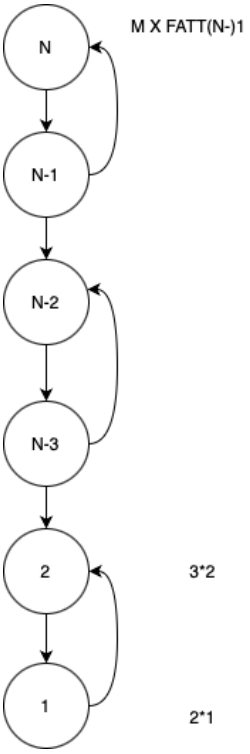
$1 * 2 * 3 + 4 * 5 * ... * n$

$$f(n) = \begin{cases} 1 & \text{se } n = 1 \\ n * f(n-1) & \text{se } n > 1 \end{cases}$$

La ricorsione è un concetto fondamentale in informatica e matematica. Nel caso del fattoriale, possiamo vedere come la funzione si richiami su se stessa con un input più piccolo fino a raggiungere il caso base.

```
FATTORIALE(N)
    IF N = 1 THEN RETURN 1 //CASO BASE
    RETURN N * FATTORIALE(N-1)
```

In ogni algorimo ricorsio viene generato un albero di ricorsione



VERSIONE INTERATIVA

```
FATTORIALE(N)
    P <- 1
    FOR I <- 2 TO N DO
        P <- P * I
    RETURN P
```

LA MOLTIPLICAZIONE

$mul(x,y) \rightarrow x * y \text{ con } \ x,y >= 1$

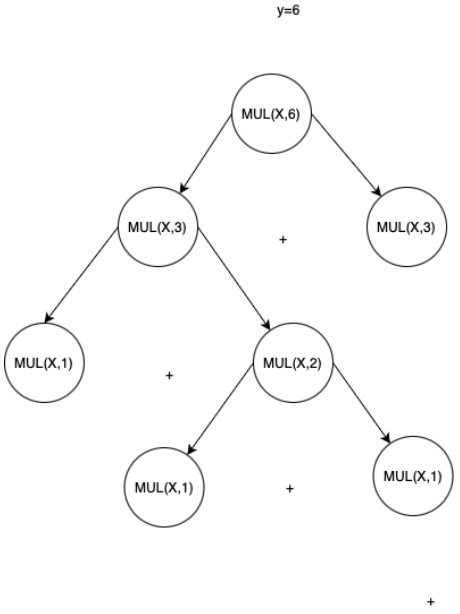
La moltilicazione si può ottenere sommando x qunte sono le y. Nel seguente modo:

$3 * 5 -> 5 * 5 * 5$ oppure $3 * 3 * 3 * 3 * 3$

$$f(n) = \begin{cases} 1 & \text{se } y = 1 \\ x + \text{mul}(x, y-1) & \text{se } y > 1 \end{cases}$$

```
MUL(X,Y)
    IF Y = 1 THEN RETURN X
    RETURN X + MUL(X,Y-1)

    soluzione con il ciling
    MUL(X,Y)
    IF Y = 1 THEN RETURN X
    RETURN MUL(X,Y/2)  + MUL(X,Y/2)
```



CALCOLO DEL N-ESIMO NUMERO NELLA SEQUENZA DI FIBONACCI

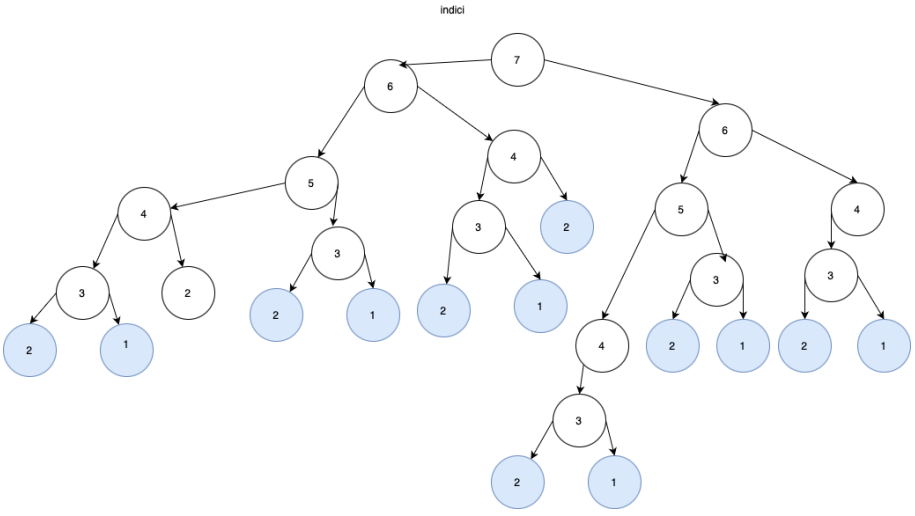
La sequenza di Fibonacci è una serie di numeri in cui ogni numero è la somma dei due precedenti. Matematicamente, può essere definita come segue:

$$F(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

```
FIBONACCI(N)
    IF  N <=2 THEN RETURN N
    RETURN FIBONACCI(N-1) + FIBONACCI(N-2)
```

- soluzione iterativa: complesità temporale millesimi di secondo
- soluzione ricorsiva: coplessità tempoarale che cresce esponenzialmente

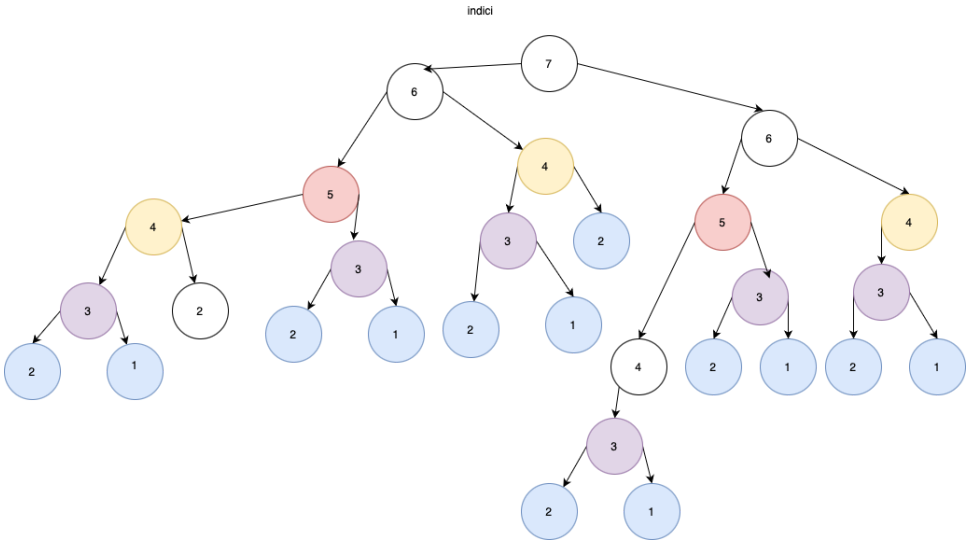
ALBERO DI RICORSIONE PER FIBONACCI



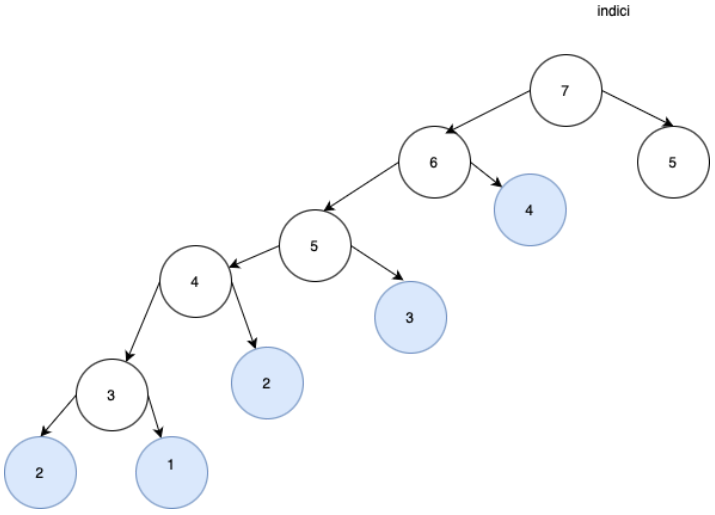
I nodi colorati in blu rappresnetano il caso base.

Con qusta soluzione si ha una crescita esponenziale della complessità temporale. In quanto alcuni sotto problemi vengono risolti più volte.

bisogna trovare una soluzione per far capire quale numeri già sono stati calcolati



RICORSIONE CON MEMORIZZAZIONE

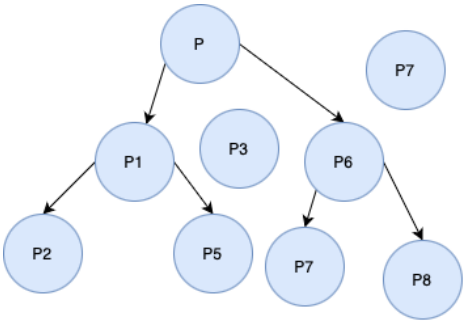


È un tipo particolare di ricorso, dove si si accorge che i problemi vengono esporati tutti, ma alcuni sono visitati più volte. Per eviatare questo bisogna meorizzare le soluzioni intermedie. Per fare ciò si usa un vettore in questo caso in quanto le infomazioni intermedie da memorizzarem è solo una. Il vettore prima va inizializato con dei valori di default come -1 da 1 fino ad N.

```
FIBONACCI(N)
    IF  N <=2 THEN RETURN N
        IF F[N-1] = -1 THEN F[N-1]=FIB(N-1)
    RETURN FIBONACCI(N-1) + FIBONACCI(N-2)
```

```
    IF F[N-1] = -1 THEN F[N-1]=FIB(N-1)
```

Nel if non si controlla anche F[N-2] perche quando si va a chiamare F[N-1] se serve viene gia calcolato.



Lo spazio delle soluzioni ai sottoproblemi non è stato espoarato tutto ma solo quelle che servono per ottenere la soluzione. Il vantaggio principale della ricorsione è che risolve i probleimi on the mand cioè solo se servono. La soluaione iterativa ha un approcio botton-up mentre la ricorsione ha un approccio top-down. Quando si approcia alla rislozione di un problema si deve capire se si sa risolvere e se bisogna applicare la ricorsione con memorizzazione. In vase a come è fatto il problema

KNAPSACK PROBLEM (problema dello zaino)

immaginavo di essere dei ladri che accendono a casa di qualcuno con uno zaino in spalla. Lo zaino ha capienza massima che non si può sorpassare, perché se no si rompe lo zaino**(K)**. Definiamo anche il nome di questi oggetti (V) che non devono superare un certo valore massimo fissato.

▼ **1° formulazione**

In questa formaulazione si considera il peso K

3	2	3	3	4	5	6	6

Tutti gli ogetti hanno il medesimo valore. Bisogna trovare una strategia per guadagnare di più.

Obiettivo di questo problema è riuscire a portare il sottoinsieme più grande possibile di elementi il cui peso non supera il valore massimo K.

Si va a prendere sempre l'oggetto con il peso minore, continuando a fare così finché gli oggetti non finiscono oppure si è raggiunto il peso massimo K.

Si procede nei seguenti passi ricorsivi:

1. **Si ordina un re dei pesi in ordine crescente(dal più piccolo al più grande)**
2. **Si va a trovare il minimo ad ogni interazione**
3. **Si termina l'algoritmo quando gli oggetti non finiscono oppure si raggiunge il peso massimo.**

$$Z(K,N) = \begin{cases} 0 & \text{se } N = 0, K = 0 \\ 0 & \text{se } P[N] > K \\ P[N] + (K - P[N], N-1) & \text{se } n > 1 \end{cases}$$

Se, invece, si ordina l'array al contrario si ha che

3	2	3	3	4	5	6	6

6	6	5	4	3	3	3	2

Bisogna capire se l'ultimo elemento va preso oppure no, va preso se non supero la dimensione, viceversa non andrà preso

```
KNAPSACK(K,N)
    SORT() // max to min
    IF N = 0  OR K=0
```

```
        RETURN 0
    IF P[N] > K
        RETURN 0
    RETURN (P[N] +KNAPSACK(K-P[N],N-1))
```

▼ **2° formalazione**

In questo caso il valre dell'ogetto è proprzionale al suo peso.

L' obiettivo dell'algoritmo in questo caso è quello di massimizzare il peso. Si può essere il caso in cui si riempie lo zaino fino alla fine, ma anche il caso in cui non si riesca a riempire del tutto, quindi bisogna avvicinarsi sempre di più al peso massimo indicato con K.

Bisogna capire se il singolo elemento va inserire oppure no nello zaino.

Si hanno due casi:

1. **Inserire l'oggetto nello zaino**: In questo caso, il peso dello zaino aumenta del peso dell'oggetto attuale, e la capacità rimanente dello zaino si riduce.
2. **Non inserire l'oggetto nello zaino**: In questo caso, si continua con gli altri oggetti senza modificare il peso dello zaino.

L'obiettivo è scegliere una di queste due opzioni in modo da avvicinarsi il più possibile al peso massimo K

FORMULA RICORSIVA

$$Z(K,N) = \begin{cases} 0 & \text{se } N = 0 \text{ o } K = 0 \\ Z(K, N-1) & \text{se } P[N-1] > K \\ \max\left(P[N-1] + Z(K - P[N-1], N-1), Z(K, N-1)\right) & \text{se } P[N-1] \leq K \end{cases}$$

$Z(K,N)$ è il massimo peso che possiamo inserire nello zaino con caacità di K peso e N ogettti

$P[N-1]$ è il peso dell' ogetto N-eseimo

Se $P[N-1] > K$ l'ogetto corrente è troppo pesante per essere aggiunto allo zaino, quindi passiamo al prossimo oggetto

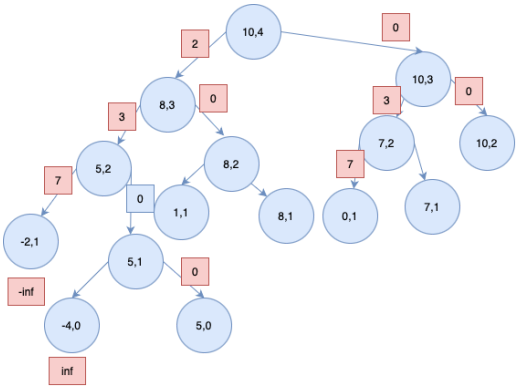
Se $P[N-1] \leq K$, confrontiamo due opzioni: in cui l'oggetto corrente nello zaino o non includerlo. Di queste operazioni, prendiamo il massimo tra i due risultati.

```
KNAPSACK(K, N)
    SORT()
    IF N = 0 OR K = 0 THEN
        RETURN 0
    ENDIF

    IF P[N-1] > K THEN
        RETURN KNAPSACK(K, N-1) // Ignora l'oggetto attuale
    ENDIF
        // Calcola il massimo tra includere o meno l'oggetto attuale
    RETURN MAX(P[N-1] + KNAPSACK(K - P[N-1], N-1),KNAPSACK(K, N-1))
```

K=10

1	2	3	4
9	7	3	2



Il primo valore è K 4 è N

▼ **3° formulazione**

Si ha un array di dimesione N che indica i pesi degli elementi dall'elemento 1 all' elemento n-esimo. Un array V da 1 a N che indica il valore di ogni ogetto. Il valore degli oggetti non è necessariamente proporzionale al peso, ma può cambiare perché ci sono oggetti di diverso materiale. Pertanto, ci possono essere oggetti pesanti con un valore basso, ed ogetti piccoli con valore alto. L'obiettivo di questo algoritmo è di riuscire a guadagnare il più possibile questi oggetti. Si inserisce nello zaino di sottoinsieme che massimizza il valore, ma il peso complessivo deve essere inferiore K.

P={.....} 1-N

V={....} 1-N

A→ insieme di tutti gli oggetti

$S \subseteq A$

S→ come gli oggetti che inseriscono nello zaino

$\sum_{x \in S} P(x) <= K$

$\sum_{x \in S} V(x)$ sia massima

$A = \{a_1, a_2, a_3, a_4, a_5, ..., a_n\}$

$$Z(K,N) = \begin{cases} 0 & \text{se } N = 0 \text{ o } K = 0 \\ -\infty & \text{se } K < 0 \\ \max\left(V[N-1] + Z(K - P[N-1], N-1), Z(K, N-1)\right) & \text{se } P[N-1] \leq K \end{cases}$$