



Analisi e Progetto di Algoritmi

spitfire

A.A. 2024-2025

Contents

1	Introduzione	3
1.1	Crescita delle funzioni	3
1.2	Metodi di risoluzione delle ricorrenze	4
1.2.1	Relazioni di ricorrenza lineari	4
1.2.2	Metodo di sostituzione	6
1.2.3	Metodo dell'albero di ricorsione	7
1.2.4	Metodo dell'esperto	7
2	Programmazione dinamica	8
2.1	LCS - Longest Common Subsequence	10

1 Introduzione

Prima di vedere gli argomenti del corso, facciamo un breve ripasso delle nozioni fondamentali per lo studio degli algoritmi.

1.1 Crescita delle funzioni

Le notazioni che usiamo per descrivere il tempo di esecuzione asintotico di un algoritmo sono definite in termini di funzioni il cui dominio è l'insieme dei numeri naturali $\mathbb{N} = \{0, 1, \dots\}$. In particolare, indichiamo con:

$$T : \mathbb{N} \rightarrow \mathbb{R}^+$$

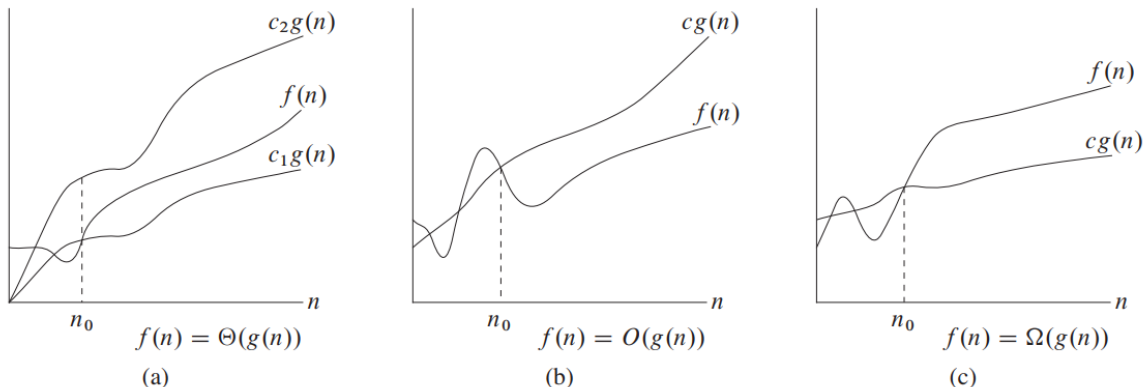
la funzione che indica i tempi di calcolo di un algoritmo. Assumiamo che n sia la dimensione dell'input per un certo algoritmo, allora $T(n)$ indica il **tempo di calcolo** (costo computazionale) dell'algoritmo in base alla dimensione dell'input. È molto raro avere un'espressione ben definita per T ; infatti molto più spesso interessa maggiormente "come cresce T " o più precisamente " T cresce come quale funzione?". Siano $g : \mathbb{R} \rightarrow \mathbb{R}^+$ e $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ due funzioni (che però applicheremo solo ai naturali). Diciamo che:

$$\Theta(g(n)) = \{f(n) \mid \exists c_1 > 0, c_2 > 0, n_0 \in \mathbb{N} \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$g(n)$ quindi si dice un **limite asintotico stretto** per f . Una funzione $f(n)$ quindi appartiene a $\Theta(g(n))$ se esistono delle costanti positive c_1, c_2 tali che essa possa essere "racchiusa" fra $c_1 g(n)$ e $c_2 g(n)$ per valori sufficientemente grandi di n .

Se vale solamente che $f(n) \leq c g(n)$ per una qualche costante c allora si dice che g è un **limite asintotico superiore** per f e si indica con $\mathcal{O}(g(n))$.

Se vale solamente che $c g(n) \leq f(n)$ per una qualche costante c allora si dice che g è un **limite asintotico inferiore** per f e si indica con $\Omega(g(n))$.



Riportiamo qui di sotto la gerarchia di crescita delle funzioni:

$$\text{costante} < \log n < n < n \cdot \log n < n^k < 2^n < n! < n^n \text{ con } k > 0$$

1.2 Metodi di risoluzione delle ricorrenze

Una ricorrenza è un'equazione o disequazione che descrive una funzione in termini del suo valore con input più piccoli. Vi sono diversi modi per risolvere una ricorrenza:

- **Metodo di sostituzione:** si ipotizza un limite e poi si utilizza l'induzione matematica per dimostrare che l'ipotesi è corretta
- **Metodo dell'albero di ricorsione:** Converte la ricorrenza in un albero i cui nodi rappresentano i costi ai vari livelli della ricorsione.
- **Metodo dell'esperto:** fornisce i limiti per le ricorrenze nella forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

dove $a \geq 1, b > 1$ e $f(n)$ è una funzione data.

Possiamo dividere le ricorrenze in due tipi:

1.2.1 Relazioni di ricorrenza lineari

Definizione 1.2.1 Una relazione di ricorrenza lineare di ordine r è una relazione del tipo:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_r a_{n-r} + f(n)$$

dove c_1, \dots, c_r sono costanti e f è una funzione di n

Definizione 1.2.2 Una relazione di ricorrenza lineare è **omogenea** di ordine r se è una relazione del tipo:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_r a_{n-r}$$

dove c_1, \dots, c_r sono costanti

Chiaramente, ogni relazione di ricorrenza lineare omogenea ha la successione identicamente nulla come soluzione. Analogamente a quanto capita per le equazioni lineari omogenee, si verifica facilmente che combinazioni lineari di soluzioni di una relazione omogenea sono ancora soluzioni.

Definizione 1.2.3 Diciamo **polinomio caratteristico** di una relazione di ricorrenza lineare omogenea R_0 di ordine r il polinomio:

$$x^r - c_1 x^{r-1} - \dots - c_r$$

Vale la seguente proposizione:

Proposizione 1.2.1 Sia λ una radice del polinomio caratteristico di una relazione lineare omogenea. Allora la successione $(\lambda^n)_n$ è una soluzione della relazione.

Vale inoltre, in generale, il seguente teorema:

Teorema 1.2.1 Si consideri una relazione di ricorrenza lineare omogenea di ordine r :

1. Supponiamo che la radice λ del polinomio caratteristico abbia molteplicità μ . Allora:

$$\lambda^n, \lambda^n n, \dots, \lambda^n n^{\mu-1}$$

sono soluzioni della relazione di ricorrenza. Al variare di λ tra le radici del polinomio caratteristico si ottengono r soluzioni di questo tipo, dette le **soluzioni-base** della relazione.

2. La soluzione generale della relazione è data da tutte le combinazioni lineari (a coefficienti complessi) delle r soluzioni-base della relazione.

Analizziamo ora una generica relazione lineare di ordine r

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_r a_{n-r} + f(n)$$

Chiameremo ancora polinomio caratteristico della relazione il polinomio caratteristico della **relazione omogenea associata**.

Proposizione 1.2.2 *La soluzione generale di una relazione di ricorrenza lineare si ottiene aggiungendo una soluzione particolare alla soluzione generale della sua parte omogenea.*

Proposizione 1.2.3 *Si consideri una relazione di ricorrenza lineare con parte non omogenea f .*

1. Sia $f(n) = cq^n$ con c costante e $q \neq 0$. Se q **non è una radice del polinomio caratteristico**, allora vi è una soluzione particolare del tipo $a_n = \alpha q^n$. Se q è **una radice del polinomio caratteristico di molteplicità μ** , vi è una soluzione particolare del tipo $a_n = \alpha n^\mu q^n$. La costante α si determina imponendo che la successione $(a_n)_n$ verifichi la relazione.
2. Sia $f(n)$ un polinomio in n di grado k . Se 1 **non è una radice del polinomio caratteristico**, una soluzione particolare è un polinomio di grado k del tipo:

$$a_n = \alpha_0 + \alpha_1 n + \dots + \alpha_k n^k$$

Se 1 è **una radice del polinomio caratteristico di molteplicità μ** , una soluzione particolare è del tipo $a_n = n^\mu (\alpha_0 + \alpha_1 n + \dots + \alpha_k n^k)$. Le costanti $\alpha_0, \dots, \alpha_k$ si determinano imponendo che la successione $(a_n)_n$ verifichi la relazione

Corollario 1.2.1 *Una relazione di ricorrenza lineare il cui termine non omogeneo è costante ammette:*

- Una soluzione costante se 1 **non è radice del polinomio caratteristico**
- Una soluzione del tipo αn^μ se 1 è **radice di molteplicità μ del polinomio caratteristico**

Una relazione di ricorrenza lineare può essere risolta anche **osservando che r^n è una soluzione per particolari valori di r** . Per relazioni di ricorrenza della forma:

$$x_n = Ax_{n-1} + Bx_{n-2}$$

si ha la soluzione r^n per la quale:

$$r^n = Ar^{n-1} + Br^{n-2}$$

dividendo tutti i termini per r^{n-2} si ottiene:

$$r^2 = Ar + B$$

ossia

$$r^2 - Ar - B = 0$$

che viene chiamata **equazione caratteristica della relazione di ricorrenza**. Essa fornisce per r due radici λ_1, λ_2 . Se tali radici sono **distinte** si ha la soluzione:

$$x_n = C\lambda_1^n + D\lambda_2^n$$

se invece le due radici **coincidono**, cioè se $A^2 + 4B = 0$ si ha:

$$x_n = C\lambda^n + Dn\lambda^n$$

dove C e D sono costanti arbitrarie che possono essere ricavate da "condizioni al contorno" che tipicamente sono date nella forma:

$$x_0 = a, \quad x_1 = b$$

1.2.2 Metodo di sostituzione

Il metodo di **sostituzione** per risolvere le ricorrenze richiede due passi:

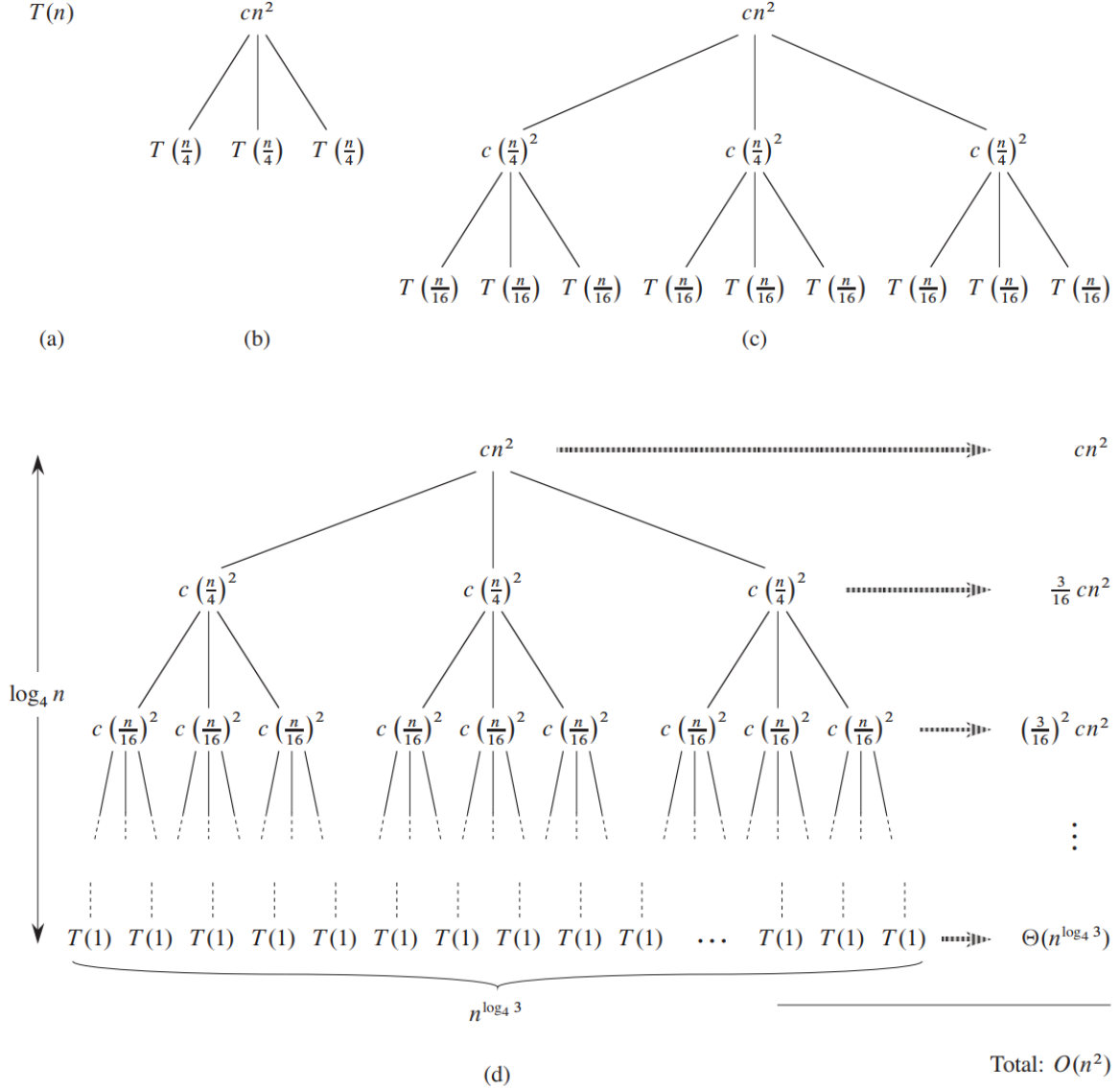
1. Ipotizzare la forma della soluzione
2. Usare l'induzione matematica per trovare le costanti e dimostrare che la soluzione funziona

Il nome del metodo deriva dalla sostituzione della soluzione ipotizzata al posto della funzione quando l'ipotesi induttiva viene applicata a valori più piccoli. Questo metodo è potente, ma ovviamente può essere applicato solamente a ricorrenze di cui sia facile immaginare la forma della soluzione. Inoltre, non esiste un metodo generale per effettuare una buona ipotesi, quindi bisogna basarsi molto spesso sull'esperienza oppure controllare se la ricorrenza considerata è simile a ricorrenze di cui si conoscono già i limiti asintotici. Altri metodi per risolvere le problematiche di questo metodo sono:

- **Effettuare ipotesi "lasche"** e man mano diminuire il grado di incertezza (cioè, restringere le ipotesi fino ad arrivare ad un limite stretto)
- **Sottrarre un termine di grado inferiore**, soprattutto in ricorrenze in cui l'ipotesi induttiva non è abbastanza forte per dimostrare il limite esatto per colpa di una costante
- **Sostituzioni di variabili**

1.2.3 Metodo dell'albero di ricorsione

In un **albero di ricorsione**, ogni nodo rappresenta il costo di un **singolo sotto-problema** da qualche parte nell'insieme delle chiamate ricorsive di funzione. Sommiamo i costi all'interno di ogni livello dell'albero per ottenere un insieme di costi per livello; poi sommiamo tutti i costi per livello per determinare il costo totale di tutti i livelli della ricorsione. Un albero di ricorsione è un ottimo modo per **ottenere una buona ipotesi** che verrà poi verificata tramite il **metodo di sostituzione**; tuttavia può essere usato anche come metodo risolutivo diretto.



1.2.4 Metodo dell'esperto

Il metodo dell'esperto permette di risolvere le ricorrenze della forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

dove $a \geq 1, b > 1$ sono costanti e $f(n)$ è una funzione **asintoticamente positiva**. Il metodo dell'esperto dipende dal seguente teorema:

Teorema 1.2.2 (Teorema dell'esperto) *Date le costanti $a \geq 1, b > 1$ e la funzione $f(n)$, sia $T(n)$ una funzione definita sugli interi non negativi dalla ricorrenza*

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

dove $\frac{n}{b}$ rappresenta $\lfloor \frac{n}{b} \rfloor$ o $\lceil \frac{n}{b} \rceil$. Allora $T(n)$ può essere asintoticamente limitata nei seguenti modi:

1. Se $f(n) = \mathcal{O}(n^{\log_b a - \varepsilon})$ per qualche costante $\varepsilon > 0$, allora $T(n) = \Theta(n^{\log_b a})$
2. Se $f(n) = \Theta(n^{\log_b a})$ allora $T(n) = \Theta(n^{\log_b a} \log n)$
3. Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ per qualche costante $\varepsilon > 0$ e se $f(\frac{n}{b}) \leq cf(n)$ per qualche costante $c < 1$ e per ogni n sufficientemente grande, allora $T(n) = \Theta(f(n))$

Questo metodo si utilizza principalmente per risolvere le ricorrenze che descrivono i tempi di calcolo degli algoritmi **dividi-et-impera**

2 Programmazione dinamica

La programmazione dinamica risolve i problemi combinando le soluzioni dei sotto-problemi. La tecnica dividi-et-impera, divide il problema in sotto-problemi **indipendenti**, li risolve in modo ricorsivo e, poi, combina le loro soluzioni per risolvere il problema originale. La programmazione dinamica, invece, può essere applicata quando **i sotto-problemi non sono indipendenti**, ovvero quando i sotto-problemi hanno **in comune dei sotto-problemi**. In questo contesto, un algoritmo dividi-et-impera **svolge molto più lavoro del necessario**, risolvendo **ripetutamente i sotto-problemi comuni**. Un algoritmo di programmazione dinamica invece **calcola una sola volta i risultati dei sotto-problemi** e li salva in una tabella, evitando quindi di ricalcolarli ogni volta che essi si presentano. La programmazione dinamica tendenzialmente si applica ai **problemi di ottimizzazione**. Per questi problemi ci possono essere molte soluzioni possibili; quindi si vuole trovare una soluzione con **valore ottimo** (minimo o massimo). Precisiamo che abbiamo detto **UNA** soluzione ottima e non **LA** soluzione ottima poiché ci possono essere più soluzioni che raggiungono il valore ottimo. Il processo di sviluppo di un algoritmo di programmazione dinamica può essere suddiviso in una sequenza di quattro fasi:

1. Caratterizzare la struttura di una soluzione ottima
2. Definire in modo ricorsivo il valore di una soluzione ottima
3. Calcolare il valore di una soluzione ottima, di solito con uno schema bottom-up (dal basso verso l'alto)
4. Costruire una soluzione ottima dalle informazioni calcolate (algoritmo di **ricostruzione**)

Durante la fase 4 possiamo memorizzare anche **informazioni aggiuntive** utili a semplificare il processo di ricostruzione. Facciamo un esempio: consideriamo un algoritmo ricorsivo che dia in output l'n-esimo numero di fibonacci:

Algorithm 1: Algoritmo ricorsivo che calcola l'n-esimo numero di fibonacci

Procedure Fib-Ric(n):

```

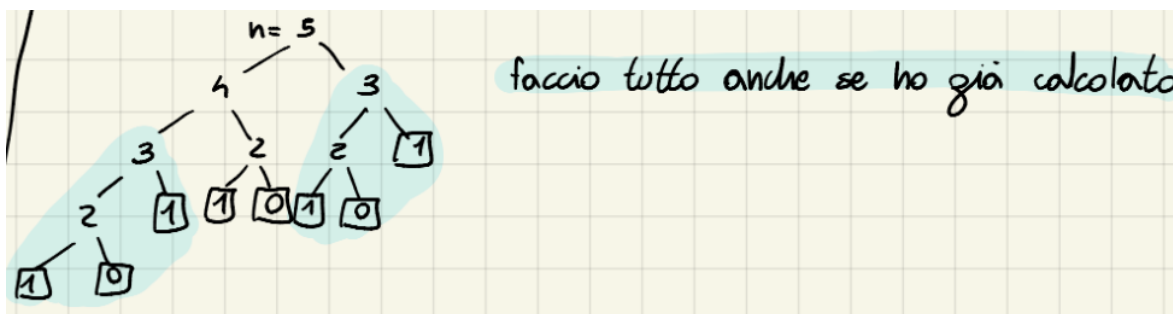
if  $n \leq 1$  then
  | return 1
else
  | return Fib-Ric( $n-1$ ) + Fib-Ric( $n-2$ )
end

```

La sua equazione di ricorrenza è:

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

Se provassi a sviluppare la ricorrenza, finirei in un **loop infinito**. Proviamo quindi il metodo dell'albero di ricorsione; supponiamo $n = 5$:



notiamo quindi che l'algoritmo **non si accorge che alcuni calcoli si ripetono**. Notiamo quindi che l'equazione di ricorrenza è una **ricorrenza lineare non omogenea**; consideriamo quindi l'equazione di ricorrenza omogenea associata:

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

supponiamo che r^n è una soluzione:

$$r^n = r^{n-1} + r^{n-2}$$

moltiplichiamo entrambi i lati per r^2 e otteniamo:

$$r^2 \cdot r^n = r \cdot r^n + r^n$$

dividiamo entrambi i lati per r^n e otteniamo:

$$r^2 = r + 1 \Rightarrow r^2 - r - 1 = 0$$

il quale è anche il polinomio caratteristico della ricorrenza. Il discriminante di questa equazione di secondo grado è $\Delta = 5$, quindi le due radici del polinomio sono:

$$r_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

notiamo quindi che la supposizione $T(n) = r^n$ è vera! Poiché $r_1 \neq r_2$ allora l'equazione di ricorrenza omogenea associata diventa:

$$T(n) = c_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + c_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

per risolvere la ricorrenza, dobbiamo aggiungere una soluzione particolare alla soluzione generale dell'equazione di ricorrenza omogenea associata (Proposizione 1.2.2), cioè quindi dobbiamo trovare un certo k tale che:

$$T(n) = c_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + c_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n + k$$

notiamo che la parte non omogenea $f(n)$ della ricorrenza iniziale è costante, quindi dal Corollario 1.2.1 sappiamo che una soluzione particolare della ricorrenza è **costante**, in particolare essa è -2 . Quindi

$$T(n) = c_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + c_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n - 2 = \Theta \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n \right)$$

come possiamo riscrivere iterativamente l'algoritmo?

Algorithm 2: Algoritmo iterativo che calcola l' n -esimo numero di fibonacci

Procedure Fib-It(n):

```

    Sia F[0,...,n] un array
    F[0] := 1
    F[1] := 1
    for  $i \leftarrow 2$  to  $n$  do
        | F[i] := F[i-1] + F[i-2]
    end
    return F[n]
```

Il tempo di calcolo di questo algoritmo è $T(n) = \Theta(n)$, tuttavia viene "sprecato" dello spazio in memoria per **memorizzare l'array**. Notiamo che l'istruzione all'interno del for è **praticamente uguale alla relazione di ricorrenza presentata in precedenza**; la programmazione dinamica è quindi **strettamente legata alla ricorsione** e a come essa **definisce la STRUTTURA della soluzione**.

2.1 LCS - Longest Common Subsequence

Sia $X = \langle x_1, \dots, x_m \rangle$ una sequenza con elementi provenienti da un alfabeto Σ , quindi $x_i \in \Sigma \forall i = 1, \dots, m$.

Definizione 2.1.1 $Z = \langle z_1, \dots, z_k \rangle$ è **sottosequenza** di X se e solo se esiste una sequenza **strettamente crescente di indici** $\langle i_1, \dots, i_k \rangle$ tali che $\forall i \in [1, k] z_i = x_{i_i}$

Nota 2.1.1 Non devono per forza essere consecutivi!

Sia ora $X = \langle x_1, \dots, x_n \rangle$ una sequenza e sia $i \in \{1, \dots, n\}$; allora il **prefisso i -esimo di X** viene indicato con $X_i = \langle x_1, \dots, x_i \rangle$ con $X_0 = \varepsilon$ che indica

la **sequenza vuota**. Facciamo due considerazioni:

- X_i è **sottosequenza** di X
- $X_n = X$ se X ha lunghezza n

Diamo ora la formulazione formale del problema:

PROBLEMA: Date due sequenze X e Y , rispettivamente di m ed n numeri interi, si determini **UNA** tra le più lunghe sottosequenze comuni a X e Y .

ISTANZA: $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$

SOLUZIONE: Z sottosequenza sia di X che di Y tale che
 $|Z| = \max\{|W| : W \text{ è sottosequenza sia di } X \text{ che di } Y\}$

Nota 2.1.2 *Notare che il la definizione del problema non è ricorsiva!*

Come possiamo quindi "sfruttare" la ricorsione per raggiungere una soluzione del problema? Dobbiamo **individuare un sottoproblema**; che in questo caso "lavorerà" con i **prefissi** di X e Y . Un sottoproblema del problema originale è quindi il seguente:

ISTANZA DEL SOTTOPROBLEMA: $X_i = \langle x_1, \dots, x_i \rangle$ e $Y_j = \langle y_1, \dots, y_j \rangle$

SOLUZIONE: $LCS(X_i, Y_j) = S^{(i,j)}$

Quindi per risolvere un problema ricorsivo, **devo immaginare di aver già risolto tutti i sottoproblemi più piccoli**, quindi di aver già risolto:

$$S^{m-1,n}, S^{m-2,n}, \dots, S^{0,n}$$

$$S^{m-1,n-1}, S^{m-2,n-1}, \dots, S^{0,n-1}, \dots$$

Quindi come andiamo a definire $S^{(m,n)}$? essa contiene **tutti i sottoproblemi risolti, combinati in una certa maniera**.

Il sottoproblema generico è quindi individuato da una coppia (i, j) tale che:

$$0 \leq i \leq m$$

$$0 \leq j \leq n$$

Definiamo allora il problema ricorsivamente:

CASO BASE: $i = 0 \vee j = 0$; allora $S^{(i,j)} = \varepsilon$

PASSO RICORSIVO: $i > 0 \wedge j > 0$

Dobbiamo distinguere due casi:

- Se $x_i = y_j$ allora $S^{(i,j)} = S^{(i-1,j-1)} | x_i$ (con $|$ che significa "accodo")
- Se $x_i \neq y_j$ allora:

$$S^{(i,j)} \begin{cases} S^{(i-1,j)} & \text{Se } |S^{(i-1,j)}| > |S^{(i,j-1)}| \\ S^{(i,j-1)} & \text{Altrimenti} \end{cases}$$

Questa però **non è una dimostrazione**; per dimostrare che effettivamente la soluzione abbia questa struttura dobbiamo introdurre il concetto di **proprietà della sottostruttura ottima (PSO)**:

Teorema 2.1.1 (PSO della LCS) Siano X_m, Y_n le sequenze e sia Z_k una LCS di X_m e Y_n . Allora:

1. Se $x_m = y_n$ allora $z_k = x_m = y_n$ e Z_{k-1} è una LCS di X_{m-1} e Y_{n-1}
2. Se $x_m \neq y_n$ e $z_k \neq x_m$ allora Z_k è una LCS di X_{m-1} e Y_n
3. Se $x_m \neq y_n$ e $z_k \neq y_n$ allora Z_k è una LCS di X_m e Y_{n-1}

Dimostrazione 2.1.1 Dimostriamo tutti i casi del teorema sopra:

1. Se $x_m = y_n$
 - (a) Facciamo vedere che $z_k = x_m = y_n$. Se per assurdo $z_k \neq x_m \vee z_k \neq y_n$ allora potrei costruire una sequenza $Z_k|x_m$; tuttavia essa sarebbe una LCS di lunghezza maggiore di Z_k , il che è **assurdo** poiché Z_k è la LCS tra X_m e Y_n .
 - (b) Facciamo vedere che $Z_{k-1} = LCS(X_{m-1}, Y_{n-1})$. Assumiamo per assurdo che Z_{k-1} non è una LCS di X_{m-1} e Y_{n-1} . Sia Z' una LCS di X_{m-1} e Y_{n-1} , allora $|Z'| > k - 1$; quindi potrei costruire una sequenza $Z'|x_m$, la quale è una sottosequenza comune di X_m e Y_n . Inoltre, $|Z'|x_m| > k$, tuttavia ciò contraddice il fatto che Z_k sia una LCS tra X_m e Y_n , quindi è **assurdo**
2. Se $x_m \neq y_n \wedge z_k \neq x_m$, allora devo far vedere che $Z_k = LCS(X_{m-1}, Y_n)$. Assumiamo per assurdo Z_k non è una LCS di X_{m-1} e Y_n . Sia Z' una LCS di X_{m-1} e Y_n , allora $|Z'| > k$. Inoltre, Z' è anche sottosequenza di X_m e Y_n , tuttavia ciò implicherebbe che Z_k non può essere una LCS di X_m e Y_n , il che è **assurdo**
3. Simmetrica a quella del punto 2

Algorithm 3: Algoritmo ricorsivo che stampa la LCS tra due sequenze X_m e Y_n

```

Procedure LCS-Ric( $X, Y, i, j$ ):
    if  $i = 0 \vee j = 0$  then
        | return  $\varepsilon$ 
    else
        if  $x_i = y_j$  then
            | return LCS-Ric( $X, Y, i-1, j-1$ ) $|x_i$ 
        else
            A := LCS-Ric( $X, Y, i-1, j$ )
            B := LCS-Ric( $X, Y, i, j-1$ )
            if  $|A| \geq |B|$  then
                | return A
            else
                | return B
            end
        end
    end

```

Notiamo che seppur tratti della struttura della soluzione, **non possiamo derivare un algoritmo basandoci unicamente sulla PSO**; infatti io non conosco il valore di z_k !

Tuttavia, se nella caratterizzazione della sottostruttura ottima sia si utilizzano sia gli input che le caratteristiche della sottostruttura ottima, **posso comunque arrivare ad un algoritmo anche in caso di mancanza di informazione**, a patto che **tutti i casi siano coperti**. Poiché è questo il caso, possiamo derivare l'algoritmo sopra riportato. La complessità dell'algoritmo 3 tuttavia è **altissima**; abbiamo però a disposizione **tutte le informazioni necessarie per applicare la programmazione dinamica e scrivere un algoritmo iterativo che calcoli la LCS tra X_m e Y_n** :

Algorithm 4: Un algoritmo iterativo che stampa una LCS tra X_m e Y_n

Procedure LCS-It(X, Y):

```

    m = LENGTH(X)
    n = LENGTH(Y)
    for  $j \leftarrow 0$  to  $n$  do
        |  $S[0,j] := \varepsilon$ 
    end
    for  $i \leftarrow 0$  to  $m$  do
        |  $S[i,0] := \varepsilon$ 
    end
    for  $i \leftarrow 1$  to  $m$  do
        for  $j \leftarrow 1$  to  $n$  do
            if  $x_i = y_j$  then
                |  $S[i,j] := S[i-1, j-1]||x_i$ 
            else
                if  $LENGTH(S[i-1,j]) \geq LENGTH(S[i, j-1])$  then
                    |  $S[i, j] := S[i-1, j]$ 
                else
                    |  $S[i,j] := S[i, j-1]$ 
                end
            end
        end
    end
    end
    return  $S[m,n]$ 

```

Ci avvaliamo quindi di una **matrice** $m \times n$ S per **salvare i risultati dei sottoproblemi**. La soluzione del problema si troverà quindi nella cella $[m,n]$ della matrice. Il **tempo di calcolo** di questo algoritmo è:

$$T(n) = \Theta(m \cdot n)$$

tuttavia, vi è uno **spreco di memoria per memorizzare la matrice**, vista che ogni cella contiene una **sottosequenza**. Possiamo quindi lavorare come una **versione ridotta del problema**, il quale ha la seguente formulazione:

PROBLEMA RIDOTTO: Date due sequenze X e Y , rispettivamente di m ed n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze comuni a X e Y .

Anche il problema ridotto contiene **diversi sottoproblemi**, ognuno dei quali non ha come input la coppia (X, Y) ma una coppia di prefissi di tali sequenze. Ogni sottoproblema è quindi **identificato da una coppia** (i, j) ed è definito come segue:

SOTTOPROBLEMA GENERICO: Date due sequenze X e Y , rispettivamente di

m ed n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze comuni al prefisso X_i e al prefisso Y_j .

Anche in questo caso, dato che $0 \leq i \leq m$ e $0 \leq j \leq n$, si ottengono $(m+1) \cdot (n+1)$ sottoproblemi (i e j possono valere 0 in quanto si deve considerare anche il caso in cui un prefisso sia la sequenza vuota). Ad ogni sottoproblema del problema ridotto è **associata una variabile**: considerato il sottoproblema di dimensione (i, j) , la variabile ad esso associata è $c_{i,j}$ ed è così definita:

$$c_{i,j} := \text{lunghezza di una tra le più lunghe sottosequenze comuni a } X_i \text{ e } Y_j$$

Per determinare la soluzione di un qualsiasi sottoproblema di dimensione (i, j) , oltre all'input, si utilizzeranno le soluzioni dei sottoproblemi di dimensione minore (come nel caso visto prima). Si noti però che ogni variabile associata ad un sottoproblema è da considerare come una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto. Scriviamo l'equazione di ricorrenza del problema ridotto:

CASO BASE: (i, j) con $i = 0 \vee j = 0$.

Il caso base si ha per un qualunque sottoproblema di dimensione (i, j) con $i = 0 \vee j = 0$, ossia quando uno dei due prefissi considerati è la sequenza vuota. In questo caso, è facile ottenere il valore della variabile $c_{i,j}$ in quanto la lunghezza di una tra le più lunghe sottosequenze comuni fra una qualsiasi fra una qualsiasi sequenza e la sequenza vuota è 0 (ossia la lunghezza della sequenza vuota). Per questa ragione, il caso base è scrivibile come:

$$c_{i,j} = 0 \text{ se } i = 0 \vee j = 0$$

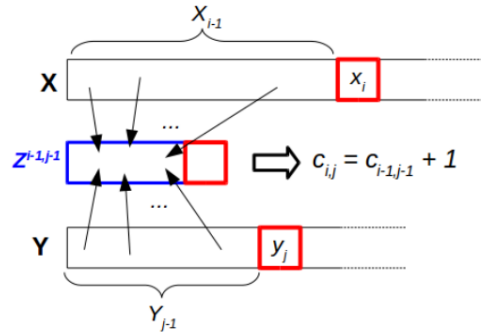
PASSO RICORSIVO: (i, j) con $i > 0 \wedge j > 0$

Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione (i, j) con $i > 0 \wedge j > 0$, ossia quando si vanno a considerare due prefissi $X_i = \langle x_1, \dots, x_i \rangle$ e $Y_j = \langle y_1, \dots, y_j \rangle$ entrambi diversi dalla sequenza vuota. I dati disponibili per calcolare $c_{i,j}$ sono:

- L'input X ed in particolare l'elemento x_i
- L'input Y ed in particolare l'elemento y_j
- Tutte le variabili $\{c_{0,0}, \dots, c_{i-1,j}, c_{i,j-1}\}$

Per calcolare $c_{i,j}$ è necessario applicare il **teorema della proprietà della sottostruttura ottima** (Teorema 2.1.1), di cui ciò che segue è una conseguenza:

- Se $x_i = y_j$: se i due elementi considerati sono identici allora la lunghezza della più lunga sottosequenza comune fra X_i e Y_j è uguale alla lunghezza della più lunga sottosequenza comune fra X_{i-1} e Y_{j-1} (ossia il valore di $c_{i-1,j-1}$) aumentata di uno (l'elemento comune x_i è accodato ad una più lunga sottosequenza comune a X_{i-1} e Y_{j-1} correlata al sottoproblema di dimensione $(i-1, j-1)$ e che quindi lunghezza $c_{i-1,j-1}$). In altri termini, se Z_k è una LCS fra X_i e Y_j , allora $z_k = x_i = y_j$ e $Z_{k-1} = LCS(X_{i-1}, Y_{j-1})$



quindi $c_{i,j} = 1 + c_{i-1,j-1}$ se $x_i = y_j$

- Se $x_i \neq y_j$ i due elementi considerati sono differenti, allora la lunghezza della più lunga sottosequenza comune è data dalla soluzione di uno dei sottoproblemi di dimensione minore. Siccome gli elementi finali dei due prefissi considerati sono differenti, non possono appartenere entrambi alla soluzione Z_k e risulta necessario considerare i seguenti due casi, sulla base di come è fatto l'ultimo elemento z_k della soluzione Z_k :

—