Analisi e Progetto di Algoritmi

spitfire

A.A. 2024-2025

Contents

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

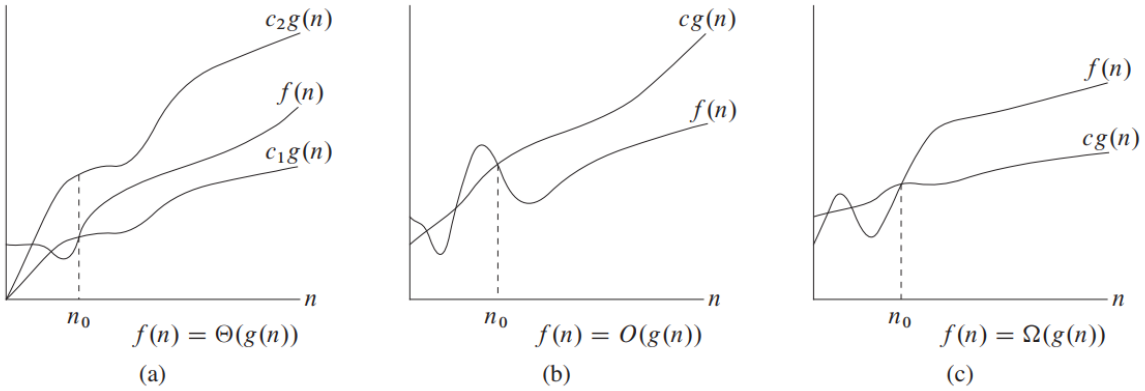
Le notazioni che usiamo per descrivere il tempo di esecuzione asintotico di un algoritmo sono definite in termini di funzioni il cui dominio è l'insieme dei numeri naturali $\mathbb{N} = \{0, 1, ...\}$. In particolare, indichiamo con:

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

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 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 $g : \mathbb{R} \to \mathbb{R}^+$ e $f : \mathbb{R}^+ \to \mathbb{R}^+$ due funzioni (che però applicheremo solo ai naturali). Diciamo che:

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

$g(n)$ 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 $f(n) \leq cg(n)$ 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 $cg(n) \leq f(n)$ 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**: fornice i limiti per le ricorrenze nella forma:

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

  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:*

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

dove $c_1, ..., c_r$ 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:*

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

dove $c_1, ..., c_r$ 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:*

$$x^r - c_1 x^{r-1} - ... - c_r$$

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:*

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

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

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

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:*

$$a_n = \alpha_0 + \alpha_1 n + \cdots + a_k n^k$$

Se 1 ***è una radice del polinomio caratteristico di molteplicità*** *μ, una soluzione particolare è del tipo $a_n = n^\mu(\alpha_0 + \alpha_1 n + \cdots + a_k n^k)$. Le costanti $\alpha_0, ..., \alpha_k$ 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:

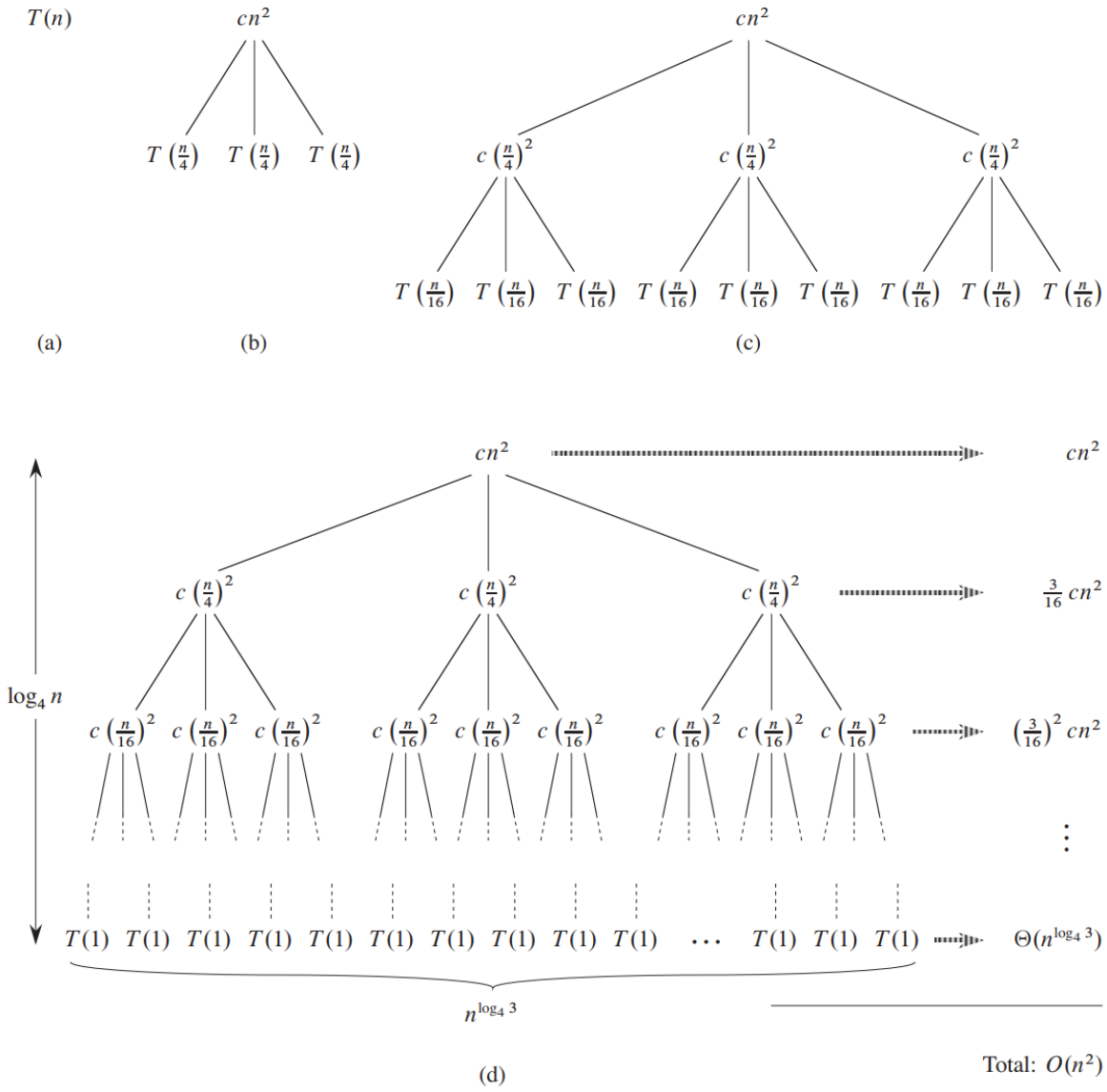
1. Ipotizzare la forma della soluzione
2. Usare l'induzione matematica per trovare le costanti e dimostrare che la soluzione funziona

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**, sopratutto 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 sottoproblema** 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:

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

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*

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

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 $T(n) = \Theta(n^{lob_b a})$*
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 informazione 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 ≤ 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 $X =< x_1, ..., x_m >$ una sequenza con elementi provenienti da un alfabeto Σ, quindi $x_i \in \Sigma \; \forall i = 1, ..., m$.

Definizione 2.1.1 *$Z =< z_1, \ldots, z_k >$ è **sottosequenza** di X se e solo se esiste una sequenza **strettamente crescente di indici** $< i_1, \ldots, i_k >$ tali che $\forall i \in [1, k]$ $z_j = x_{i_j}$*

Nota 2.1.1 *Non devono per forza essere consecutivi!*

Sia ora $X =< x_1, \ldots, x_n >$ una sequenza e sia $i \in \{1, \ldots, n\}$; allora **il prefisso i-esimo di** X viene indicato con $X_i =< x_1, \ldots, x_i >$ 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: $X =< x_1, \ldots, x_m >$ e $Y =< y_1, \ldots, y_n >$
SOLUZIONE: Z sottosequenza sia di X che di Y tale che
$|Z| = \max\{|W| : \text{W è sottosequenza sia di X che di Y}\}$

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: $X_i =< x_1, ..., x_i >$ e $Y_j =< y_1, ..., y_j >$
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:

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

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

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 chè 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 $LCS(X_{m-1}, Y_n)$. Sia Z' una $LCS(X_{m-1}, Y_n)$, allora $|Z'| > k$. Inoltre, Z' è anche sottosequenza di X_m e Y_n, tuttavia ciò implicherebbe che Z_k non può essere una $LCS(X_m, Y_n)$, 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 ∨ j = 0 then
        return ε
    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| ≥ |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 ← 0 to n do
        S[0,j] := ε
    end
    for i ← 0 to m do
        S[i,0] := ε
    end
    for i ← 1 to m do
        for j ← 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]) ≥ 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
    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 soluzione 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:

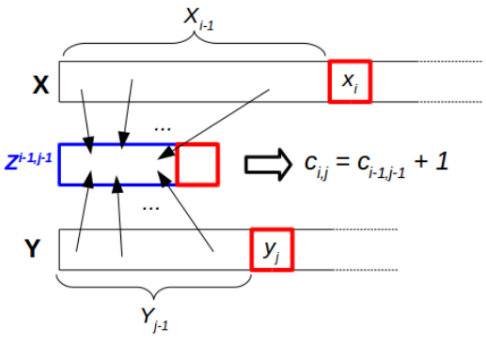
$$c_{i,j} = 0 \text{ se } i = 0 \vee j = 0$$

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 $X_i =< x_1, \ldots, x_i$ e $Y_j =< y_1, \ldots, y_j >$ 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 $\{c_{0,0}, \ldots, c_{i-1,j}, c_{i,j-1}\}$

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:

  - Se $z_k \neq x_i$, allora si deve considerare la soluzione $c_{i-1,j}$, ossia la soluzione del sottoproblema di dimensione $(i-1, j)$ con input il prefisso X_{i-1} e il prefisso Y_j
  - Se $z_k \neq y_j$, allora si deve considerare la soluzione $c_{i,j-1}$, ossia la soluzione del sottoproblema di dimensione $(i, j-1)$ con input il prefisso X_i e il prefisso Y_{j-1}

  Siccome z_k non è noto, non possiamo sapere a priori quale dei due casi si verifichi. Quindi consideriamo la soluzione di entrambi i sottoproblemi di dimensione $(i-1, j)$ e $(i, j-1)$ e scegliamo quella di valore massimo. Formalmente, possiamo scrivere:

$$c_{i,j} = \max\{c_{i-1,j}, c_{i,j-1}\} \quad sex_i \neq y_j$$

Il passo ricorsivo è quindi descrivibile come:

$$c_{i,j} = \begin{cases} 1 + c_{i-1,j-1} & \text{se } x_i = y_j \\ \max\{c_{i-1,j}, c_{i,j-1}\} & \text{se } x_i \neq y_j \end{cases}$$

Una volta calcolati i valori di tutte le soluzioni $(c_{0,0}, c_{0,1}, \ldots, c_{m,n})$, si hanno a disposizione tutte le lunghezza delle sottosequenze comuni massimali fra qualsiasi prefisso di X e qualsiasi prefisso di Y. La **soluzione del problema ridotto** è quindi

$$c_{m,n}$$

		0	1	2	3	4	5	6	7	8	**j**
		ε	2	7	4	23	21	14	1	8	y_j
0	ε	0	0	0	0	0	0	0	0	0	
1	2	0	1	1	1	1	1	1	1	1	
2	4	0	1	1	2	2	2	2	2	2	
3	7	0	1	2	2	2	2	2	2	2	
4	11	0	1	2	2	2	2	2	2	2	
5	21	0	1	2	2	2	3	3	3	3	
6	14	0	1	2	2	2	3	4	4	4	
7	1	0	1	2	2	2	3	4	5	5	
i	x_i										

Scriviamo quindi l'algoritmo bottom-up per calcolare la lunghezza di una LCS fra X e Y

Algorithm 5: Algoritmo iterativo che calcola la lunghezza di una LCS di X e Y

Procedure `LCS-LEN`(X, Y)**:**

```
m := LENGTH(X)
n := LENGTH(Y)
for j ← 0 to n do
    c[0,j] := 0
end
for i ← 0 to m do
    c[0,j] := 0
end
for i ← 1 to m do
    for j ← 1 to n do
        if x_i = y_j then
            c[i,j] := c[i-1, j-1] + 1
            b[i,j] := "↖"
        else
            if c[i − 1, j] ≥ c[i, j − 1] then
                c[i,j] := c[i-1,j]
                b[i,j] := "↑"
            else
                c[i,j] := c[i, j-1]
                b[i,j] := "←"
            end
        end
    end
end
return c e b
```

Questo algoritmo ha lo stesso costo computazionale dell'algoritmo 4, cioè $T(n) = \Theta(m \cdot n)$. Si noti che si utilizza una matrice ausiliaria b di dimensioni $m \times n$ per memorizzare **l'informazione utile a ricostruire una LCS di X e Y**; in particolare, andremo a salvare **quale sottoproblema è stato usato** tramite una freccia:

- $\uparrow$ indica che è stato usato il sottoproblema $(i-1, j)$
- $\leftarrow$ indica che è stato usato il sottoproblema $(i, j-1)$
- $\nwarrow$ indica che è stato usato il sottoproblema $(i-1, j-1)$

Tramite la matrice b, abbiamo tutte le informazioni necessarie per definire un algoritmo ricorsivo che **ricostruisca una LCS di X e Y**:

Algorithm 6: Algoritmo ricorsivo che stampa una LCS di X e Y

```
Procedure PRINT-LCS(b, X, i, j):
    if i ≠ 0 ∧ j ≠ 0 then
        if b[i, j] = " ↖ " then
            PRINT-LCS(X, b, i-1, j-1)
            Print(x_i)
        else
            if b[i, j] = " ↑ " then
                PRINT-LCS(X, b, i-1, j)
            else
                PRINT-LCS(X, b, i, j-1)
            end
        end
    end
```

La procedura impiega un tempo $T(n) = \Theta(m+n)$ perché a ogni chiamata ricorsiva essa decrementa almeno uno dei valori i e j.

2.2 WIS - Weighted Interval Scheduling

Sia $X = \{1, \ldots, n\}$ un insieme di n attività, con $n \in \mathbb{N}$. Ad ogni attività $i \in X$ sono associate le seguente informazioni $([s_i, e_i), v_i)$ dove:

- s_i indica il tempo di inizio dell'attività
- e_i con $e_i \geq s_i$ indica il tempo di fine dell'attività (si noti che e_i non appartiene all'intervallo che specifica la durata dell'attività)
- v_i indica il valore dell'attività

Due attività $i, j \in X$ sono dette **compatibili** se non si sovrappongono:

$$[s_i, e_i) \cap [s_j, e_j) = \emptyset$$

Un insieme X di attività **contiene attività mutualmente compatibili** se e solo se per tutte le coppie (i, j) con $i \neq j$, l'attività i è compatibile con l'attività j:

$$\forall i, j \in X \text{ con } i \neq j, \text{se } [s_i, e_i) \cap [s_j, e_j) = \emptyset$$

Definiamo ora la seguente funzione:

$$COMP : \mathcal{P}(\{1, \ldots, n\}) \to \{True, False\}$$

che associa ad ogni sottoinsieme di attività $A \subseteq \{1, \ldots, n\}$, **True** se X contiene attività mutualmente compatibili, **False** altrimenti. Formalmente: $\forall A \subseteq \{1, \ldots, n\}$:

$$COMP(A) = \begin{cases} True & \text{se } \forall i, j \in A \text{ con } i \neq j, [s_i, e_i) \cap [s_j, e_j) = \emptyset \\ False & \text{Altrimenti} \end{cases}$$

Infine, definiamo la funzione

$$V : \mathcal{P}(\{1, \ldots, n\}) \to \mathbb{R}^+$$

che associa ad ogni sottoinsieme di attività $A \subseteq \{1, \ldots, n\}$ il suo valore complessivo. Formalmente, $\forall A \subseteq \{1, \ldots, n\}$:

$$V(A) = \begin{cases} \sum_{i \in A} v_i & \text{se } A \neq \emptyset \\ 0 & \text{Altrimenti} \end{cases}$$

Definiamo quindi il problema:
PROBLEMA: Dato un insieme $\{1, \ldots, n\}$ di attività, trovare un sottoinsieme di attività mutualmente compatibili di valore massimo
ISTANZA: $\{1, \ldots, n\}$ con $([s_i, e : i), v_i)$ $\forall i \in \{1, \ldots, n\}$
SOLUZIONE: $S \subseteq \{1, \ldots, n\}$ tale che:

$$COMP(S) = TRUE \wedge V(S) = \max_{COMP(A)=TRUE} \{V(A) | A \subseteq X\}$$

Possiamo direttamente risolvere questo problema **senza definire un problema ridotto**. Il problema contiene **diversi sottoproblemi**, ognuno dei quali non ha come input l'insieme X con $([s_i, e_i)_i), \forall i \in X$ ma un suo sottoinsieme. Formuliamo quindi il problema in modo **ricorsivo**:

1. Dato $X_n = \{1, \ldots, n\}$ si vuole trovare una soluzione $S_n \subseteq X_n$ tale che:

$$COMP(S_n) = True \wedge V(S_n) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_n\}$$

2. Dato $X_{n-1} = \{1, \ldots, n-1\}$ si vuole trovare una soluzione $S_{n-1} \subseteq X_{n-1}$ tale che:

$$COMP(S_{n-1}) = True \wedge V(S_{n-1}) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_{n-1}\}$$

3. ...

4. Dato $X_2 = \{1, 2\}$ si vuole trovare una soluzione $S_2 \subseteq X_2$ tale che:

$$COMP(S_2) = True \wedge V(S_2) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_2\}$$

5. Dato $X_1 = \{1\}$ si vuole trovare una soluzione $S_1 \subseteq X_1$ tale che:

$$COMP(S_1) = True \wedge V(S_1) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_1\}$$

6. Dato $X_0 = \emptyset$ si vuole trovare una soluzione $S_0 \subseteq X_0$ tale che:

$$COMP(S_0) = True \wedge V(S_0) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_0\}$$

Quindi, **il generico sottoproblema del problema originale è individuato** da $i \in \{1, \ldots, n\}$; il **sottoproblema di dimensione i è definito come segue**:
SOTTOPROBLEMA GENERICO: Dato un insieme $\{1, \ldots, n\}$ di attività, trovare un suo sottoinsieme di attività mutualmente compatibili di valore massimo
Ossia:
ISTANZA DEL SOTTOPROBLEMA: $X_i = \{1, \ldots, i\}$ con $([s_i, e_i), v_j)$ $\forall i \in X_i$
SOLUZIONE DEL SOTTOPROBLEMA: $S_i \subseteq X_i$ tale che:

$$COMP(S_i) = True \wedge V(S_i) = \max_{COMP(A)=True} \{V(A) | A \subseteq X_i\}$$

Dato che $0 \leq i \leq n$ si ottengono $(n+1)$ sottoproblemi, ad ognuno dei quali è associata una **coppia di variabili**: considerato il sottoproblema di dimensione i, la coppia di variabili ad esso associata è (OPT_i, S_i) ed è cosi definita:

S_i, un sottoinsieme di X_i che contiene attività compatibili di peso massimo
$OPT_i = V(S_i)$, ossia il valore di un sottoinsieme di X_i che contiene attività mutualmente compatibili di peso massimo

Per determinare la soluzione di un qualsiasi sottoproblema di dimensione i, oltre all'input del problema, si utilizzeranno le soluzione dei sottoproblemi di dimensione minore e le loro variabili associate S_j e OPT_j. Si noti, però, che ognuna di queste variabili è da considerare come una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto. Scriviamo quindi l'equazione di ricorrenza del problema:
CASO BASE: $i = 0$
Il caso vase si ha per un qualunque sottoproblema di dimensione $i = 0$, ossia quando non ci sono attività da considerare. In questo caso è facile ottenere la soluzione del sottoproblema:

$$OPT_i = 0$$
$$S_i = \emptyset$$

Dunque il caso base è descrivibile come:

$$OPT_i = 0 \text{ e } S_i = \emptyset \text{ se } i = 0$$

PASSO RICORSIVO: $i > 0$ Per definire il passo ricorsivo, assumiamo che le attività in $X_n = \{1, \ldots, n\}$ siano ordinate rispetto ai **tempi di fine**, cioè che valga che $e_1 \leq e_2 \leq \cdots \leq e_i \leq \cdots \leq e_n$. Inoltre, dato $i \in X_n$, definiamo:

$$\psi(i) = \max\{j | j < i \text{ e } j \text{ è compatibile con } i\}$$

Assumiamo infine che $\max \emptyset = 0$.
Ora possiamo definire il passo ricorsivo. Esso si ha per un qualunque sottoproblema di dimensione i con $i > 0$, ossia quando si va a considerare un insieme che contiene almeno un'attività. I dati disponibili per calcolare la soluzione di un sottoproblema sono: l'input $X_i = \{1, \ldots, i\}$ (in particolare l'attività i) e tutte le soluzioni dei sottoproblemi di dimensione minore. Per risolvere un sottoproblema è necessario domandarsi: "Com'è fatto S_i rispetto a $S_{i-1}, S_{i-2}, \ldots, S_1, S_0$?" Per rispondere a questa domanda, bisogna considerare due casi:

- Se $i \notin S_i$, cioè se l'attività i non appartiene alla soluzione, allora è sufficiente considerare l'insieme di attività $X_{i-1} = \{1, \ldots, i-1\}$ e si può dunque considerare la soluzione del sottoproblema di dimensione subito minore, ossia:

$$OPT_i = OPT_{i-1} \text{ e } S_i = S_{i-1}$$

- Se $i \in S_i$, cioè se l'attività i appartiene alla soluzione, allora è necessario andare a determinare la soluzione del sottoproblema di dimensione minore ad i di peso massimo e il cui insieme di attività risulta essere compatible con l'attività i. È dunque possibile considerare la soluzione del sottoproblema di dimensione $\psi(i)$ e aggiungere a questa attività i. Si ottiene dunque:

$$OPT_i = OPT_{\psi(i)} + v_i \text{ e } S_i = S_{\psi(i)} \cup \{i\}$$

Siccome non possiamo sapere a priori se l'attività i appartenga o meno alla soluzione, consideriamo **entrambe le soluzioni e scegliamo quella di valore massimo**. Formalmente possiamo scrivere:

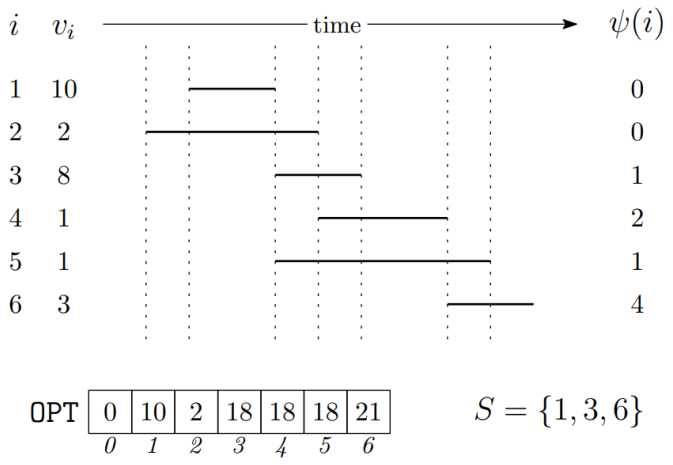
$$OPT_i = \max\{OPT_{i-1}, OPT_{\psi(i)} + v_i\}$$

e

$$S_i = \begin{cases} S_{i-1} & \text{se } OPT_{i-1} \geq OPT_{\psi(i)} + v_i \\ S_{\psi(i)} \cup \{i\} & \text{Altrimenti} \end{cases}$$

Si noti come l'insieme S_i venga definito a partire dalle variabili S_j ma sulla base dei valori delle variabili OPT_j relative ai sottoproblemi di dimensione minore (ossia con $j < i$). Una volta calcolati i valori $OPT_0, OPT_1, \ldots, OPT_n$, è possibile determinare la **soluzione del problema** andando a considerare la coppia:

$$(OPT_n, S_n)$$



Enunciamo quindi il teorema della **proprietà della sottostruttura ottima**:

Teorema 2.2.1 (PSO di WIS) *Sia $i \geq 1$. Siano $S_0, S_1, \ldots, S_{i-1}$ soluzioni dei sottoproblemi di dimensione $0, 1, \ldots, i-1$ di WIS. Allora vale che:*

$$S_i = \begin{cases} S_{\psi(i)} \cup \{i\} & \text{se } i \in S_i \\ S_{i-1} & \text{se } i \notin S_i \end{cases}$$

Dimostrazione 2.2.1 *Distinguiamo i due casi:*

- *Supponiamo che $i \notin S_i$; devo allora dimostrare che $S_i = S_{i-1}$. Supponiamo per assurdo che S_{i-1} non sia soluzione del problema i, cioè $S_i \neq S_{i-1}$. Allora esiste $S' \neq S_{i-1}$ che è soluzione del problema i; allora $V(S') > V(S_{i-1})$. Inoltre $i \neq S' rightarrow S' \subseteq \{1, \ldots, i-1\}$, ma allora S_{i-1} non potrebbe essere la soluzione del sottoproblema $i-1$, **assurdo!**.*

- *Supponiamo che $i \in S_i$; devo dimostrare che $S_i = S_{\psi(i)} \cup \{i\}$ Supponiamo per assurdo che $S_i \neq S_{\psi(i)} \cup \{i\}$ e che quindi $S_{\psi(i)} \cup \{i\}$ non è soluzione del problema i. Allora esiste $S' \neq S_{\psi(i)} \cup \{i\}$ che è soluzione del problema i; allora $V(S') > V(S_{\psi(i)} \cup \{i\})$. Inoltre $i \in S'$, quindi $S' = S'' \cup \{i\}$ con S'' formato da attività mutualmente compatibili. Inoltre $S'' \subseteq \{1, \ldots, \psi(i)\}$, quindi:*

  $$V(S') = V(S'') + v_i > V(S_{\psi(i)}) + v_i = V(S_{\psi(i)} \cup \{i\})$$

  cioè

  $$V(S'') > V(S_{\psi(i)})$$

  *ma quindi $S_{\psi(i)}$ non può essere soluzione del sottoproblema $S_{\psi(i)}$, **assurdo!***

Utilizzando le equazioni di ricorrenza presentate precedentemente, possiamo definire in maniera immediata un algoritmo ricorsivo che ritorni **la soluzione di ogni sottoproblema**:

Algorithm 7: Algoritmo ricorsivo che ritorna la soluzione del problema WIS

```
Procedure WIS-Ric(i):
    if i = 0 then
        return (0, ∅)
    else
        (V_1, S_1) ← WIS-Ric(i − 1) ;
        (V_2, S_2) ← WIS-Ric(ψ(i)) ;
        V_2 ← V_2 + v_i ;
        if V_1 ≥ V_2 then
            return (V_1, S_1)
        else
            return (V_2, S_2 ∪ {i})
        end
    end
```

Questo algoritmo **ha complessità esponenziale**, dato che ogni sottoproblema viene risolto più volte. È Anche facile scrivere un algoritmo ricorsivo che calcoli il valore di OPT_i:

Algorithm 8: Algoritmo ricorsivo che calcola il valore di OPT_i

```
Procedure OPT-Ric(i):
    if i = 0 then
        return 0
    else
        return max{OPT-Ric(ψ(i)) + v_i, OPT-Ric(i − 1)}
    end
```

Possiamo utilizzare una tecnica **bottom-up** per risolvere il problema WIS: questa tecnica infatti permette di calcolare la soluzione di ogni sottoproblema solo una volta. In questo modo si ottiene un algoritmo con **complessità spaziale maggiore**, in quanto devo memorizzare le varie soluzioni, ma con **complessità di tempo minore rispetto all'algoritmo ricorsivo**. L'algoritmo bottom-up per il calcolo della soluzione del problema WIS è il seguente:
Prima di tutto definisco un algoritmo iterativo per calcolare i valori di OPT legati alle soluzioni; per farlo mi avvalgo di un array M di dimensione n:

Algorithm 9: Algoritmo iterativo per il calcolo di OPT

Procedure `OPT-IT(`n`):`

$M[0] := 0$;
for $i \leftarrow 1$ *to nell* **do**
  $M[i] := \max\{M[\psi(i)] + v_i, M[i-1]\}$
end

dopodiché definisco l'algoritmo bottom-up per il calcolo della soluzione al problema WIS: per farlo mi avvalgo di un array S di dimensione n che contiene le soluzioni dei sottoproblemi (cioè gli insiemi di attività mutualmente compatibili)

Algorithm 10: Algoritmo iterativo per il calcolo della soluzione del problema WIS

Procedure `WIS-IT(`n`):`

$S := \emptyset$;
for $i \leftarrow 1$ *to* n **do**
  $V_1 := M[i-1]$;
  $V_2 := M[\psi(i)] + v_i$;
  if $V_1 \geq V_2$ **then**
    $S[i] := S[i-1]$
  else
    $S[i] := S[\psi(i)] \cup \{i\}$
  end
end
return $(M[n], S[n])$

Algorithm 11: Algoritmo di ricostruzione della soluzione di WIS

Procedure `PRINT-WIS(`i, M`):`

if $i \neq 0$ **then**
  if $M[\psi(i)] + v_i \geq M[i-1]$ **then**
    PRINT-WIS($\psi(i)$, M)
    Print(i)
  else
    PRINT-WIS($i-1$, M)
  end
end

La complessità computazionale dell'algoritmo 10 è:

$$T(n) = \mathcal{O}(n)$$

tuttavia, esso occupa uno spazio in memoria pari a $\mathcal{O}(n + n^2)$ poiché necessita di un

vettore per contenere i vari OPT_i e un vettore per contenere i vari insieme S_i, ognuno dei quali può contenere al massimo n elementi.

2.3 LIS - Longest Increasing Subsequence

Il problema LIS è definito come segue:
PROBLEMA: Data una sequenza X di n numeri interi, si determini **UNA** tra le più lunghe sottosequenze strettamente crescenti di X.
In particolare, come nel caso di LCS, andremo a risolvere un **problema ridotto**:
PROBLEMA RIDOTTO: Data una sequenza X di n numeri interi, si determini **la lunghezza** di una tra le più lunghe sottosequenze strettamente crescenti di X
Il problema ridotto contiene **diversi sottoproblemi** ognuno dei quali non ha come input la sequenza X ma un suo prefisso. Il **generico sottoproblema** di dimensione i è definito come segue:
SOTTOPROBLEMA GENERICO: Data una sequenza X di n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze strettamente crescenti di X_i.
Dato che $1 \leq i \leq n$, si ottengono n sottoproblemi (non si considera in questo caso il valore $i = 0$). Ad ogni sottoproblema del problema ridotto **è associata una variabile**: considerato il sottoproblema di dimensione i, la variabile ad esso associata è c_i, ed è così definita:

c_i = lunghezza di una tra le più lunghe sottosequenze strettamente crescenti di X_i

Per determinare la soluzione di un qualsiasi sottoproblema di dimensione i, oltre all'input del problema, si utilizzeranno le soluzioni dei sottoproblemi di dimensione minore. Proviamo ad impostare la risoluzione del problema come abbiamo fatto con LCS: siano allora:

$$\begin{gathered} X_n =< x_1, \ldots, x_n > \\ X_{n-1} =< x_1, \ldots, x_{n-1} > \\ \ldots \\ X_1 =< x_1 > \\ X_0 = \varepsilon \end{gathered}$$

La soluzione allora potrebbe avere la seguente struttura:

$$S_n = S_{n-k}|x_n \text{ con } 0 < k < n$$

per sapere se questa assunzione sia vera o meno, mi basterebbe sapere l'ultimo elemento di S_{n-k}; tuttavia **NON POSSIAMO SAPERE IL VALORE DI** s_{n-k} poiché le soluzioni ai sottoproblemi sono delle **black-box** e quindi **NON NE CONOSCIAMO IL VALORE!**. Per risolvere il problema dobbiamo quindi introdurre un **PROBLEMA VINCOLATO**:
PROBLEMA VINCOLATO: Data una sequenza X di n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze strettamente crescenti di X, la quale termina con x_n.
In altre parole, si richiede che la sottosequenza crescente più lunga termini con l'ultimo elemento della sequenza di input. Come per il problema ridotto, **anche il problema**

vincolato contiene n **diversi sottoproblemi**, ognuno associato ad una differente variabile. Il sottoproblema generico del problema vincolato di dimensione i è definito come segue:
SOTTPROB. GEN. PROBLEMA VINCOLATO: Data una sequenza X di n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze strettamente crescenti di X_i, la quale termina con x_i
ogni sottoproblema generico è associato ad una variabile c_i così definita:

c_i = lunghezza di una tra le più lunghe sottosequenze crescenti di X_i, la quale termina con x_i

Si noti quindi che per un qualunque sottoproblema di dimensione s, solo imponendo che la soluzione c_s si riferisca ad una sottosequenza strettamente crescente di X_s la quale termina con x_s è possibile stabilire se un altro elemento x_u con $u > s$ possa essere accodato a tale sottosequenza (andando a verificare che $x_s < x_u$).
Prima di procedere con le equazioni di ricorrenza, può essere utile enunciare e dimostrare già adesso il teorema della proprietà della sottostruttura ottima **per il problema vincolato**:

Teorema 2.3.1 (PSO LIS Vincolato) *Sia X una sequenza di n numeri interi e sia X_i un suo prefisso di lunghezza i con $1 \leq i \leq n$. Sia Z^i una tra le più lunghe sottosequenze strettamente crescenti di X_i la quale termina con x_i. Allora vale che $Z^i = Z^*|x_i$, con $Z^* \in \mathcal{W}_i$ e $|Z^*| = \max_{W \in \mathcal{W}_i}\{|W|\}$, dove $\mathcal{W}_i$ è l'insieme di tutte le sottosequenze crescenti (**non necessariamente più lunghe**) di X_j le quali finiscono con x_j e a cui è possibile concatenare x_i, ovvero:*

$$\mathcal{W}_i = \bigcup_{\substack{1 \leq j \leq i \\ x_j < x_i}} \{W \text{ sottosequenza di } X_j \text{ la quale termina con } x_j\}$$

Si noti che Z^* **sarà anche soluzione di un qualche sottoproblema di dimensione minore a** i (una più lunga tra le soluzioni di alcuni sottoproblemi più piccoli, in particolare quei sottoproblemi $j < i$ tali che $x_j < x_i$).

Dimostrazione 2.3.1 *Per assurdo, si supponga che $Z^*|x_i$ non sia la soluzione del problema i-esimo. Allora, relativamente alla soluzione Z^i del problema valgono le seguente affermazioni:*

1. *Z' tale che $Z^i = Z'|x_i$*
2. *$|Z'| > |Z^*|$ (dato che Z^i non si ottiene a partire da Z^* ma da Z', necessariamente Z' sarà più lunga di Z^*)*

*Dove Z' è una qualche sottosequenza crescente di un prefisso più piccolo di X_i. Sia ora z' l'ultimo elemento di Z'. Vale quindi che $z' < x_i$ poiché è stato possibile concatenare x_i a Z'. Inoltre, sia $h < i$ **il più grande indice tale che** $x_h = z'$. Di conseguenze, per come è stato definito $\mathcal{W}_i$, si ottiene che $Z' \in \mathcal{W}_i$. Infatti Z' è una **sottosequenza strettamente crescente** di X_h, la quale termina con $x_h < x_i$. Ciò pero porta ad una contraddizione: infatti dal punto 2 vale che $|Z'| > |Z^*|$, ma ciò non può valere con l'ipotesi che $|Z*| = \max_{W \in \mathcal{W}_i}\{|W|\}$, quindi **assurdo!***

Scriviamo ora le equazioni di ricorrenza:
CASO BASE: $i = 1$
Il caso base si ha quando $i = 1$, ossia quando il prefisso considerato è una sequenza composta da un singolo elemento. È facile ottenere il valore della variabile c_1: la lunghezza di una tra le più lunghe sottosequenze crescenti di una sequenza composta da un singolo elemento la quale termina con quell'elemento è 1. Il caso base è quindi scrivibile come:

$$c_i = 1 \text{ se } i = 1$$

PASSO RICORSIVO: $i > 1$
Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione i con $i > 1$, ossia quando si considera un prefisso della sequenza X in input di almeno due elementi. I dati disponibili per calcolare c_i sono: l'input X ed in particolare l'elemento x_i e tutte le variabili $\{c_1, \ldots, c_{i-1}\}$. Si ricorda che i valori $c_1, c_2, \ldots, c_{i-1}$ rappresentano **le lunghezza delle più lunghe sottosequenze strettamente crescenti di** $X_1, X_2, \ldots, X_{i-1}$ le quali terminano rispettivamente con $x_1, x_2, \ldots, x_{i-1}$. Tra queste ci saranno alcune sottosequenze alle quali possiamo accodare x_i (in quanto maggiore dell'ultimo elemento) e altre alle quali l'elemento x_i non può essere accodato (in quanto minore dell'ultimo elemento). Se prendiamo la più lunga sottosequenza strettamente crescente alla quale possiamo attaccare x_i e accodiamo ad essa x_i, otteniamo la più lunga sottosequenza strettamente crescete di X_i che termina con x_i. La lunghezza di tale sottosequenza sarà uguale alla lunghezza della sottosequenza crescente alla quale abbiamo accodato x_i aumentata di 1. Il passo ricorsivo è quindi scrivibile come:

$$c_i = 1 + \max\{c_h | 1 \leq i \wedge x_h < x_i\}$$

Poiché può accadere che l'insieme $\{c_h | 1 \leq i \wedge x_h < x_i\}$ **sia vuoto** (il che corrisponde al fatto che l'elemento x_i è minore di tutti gli elementi precedenti e, quindi, non può essere accodato a nessuna più lunga sottosequenza strettamente crescente relativa a sottoproblemi di dimensione minore), assumiamo per definizione che $\max \emptyset = 0$, così che il corrispondente valore di c_i risulti uguale a 1 in questo caso.

i		*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
x_i		2	4	7	6	11	13	21	14	1
		1	2	3	3	4	5	6	6	1

Una volta calcolati i valori $c_1, \ldots, c_n$ si hanno a disposizione tutte le lunghezze di una tra le più lunghe sottosequenze crescenti dei vari prefissi di X le quali terminano con l'ultimo elemento del prefisso. La **soluzione del problema vincolato** è c_n, mentre **la soluzione del problema ridotto** è:

$$\max\{c_i | 1 \leq i \leq n\}$$

Con le informazioni presentate fino ad ora, è immediato ricavare l'algoritmo ricorsivo per determinare la soluzione c_i di ogni sottoproblema (si suppone di aver accesso alla sequenza X). In ogni caso, questo algoritmo **presenterà una complessità esponenziale**, dato che ogni sottoproblema viene risolto più volte.

Algorithm 12: Algoritmo ricorsivo per risolvere un sottoproblema i di LIS

```
Procedure LIS-Ric(i):
    if i = 1 then
        return 1
    else
        max := 0
        for h ← 1 to i − 1 do
            if x_h < x_i then
                S := LIS-Ric(h)
                if S > max then
                    max = S
                end
            end
        end
        return 1 + max
    end
```

Possiamo quindi scrivere un algoritmo bottom-up per risolvere il problema. Utilizziamo anche un array b per salvare dati utili alla ricostruzione.

Algorithm 13: Algoritmo iterativo che calcola una LIS di una sequenza X

```
Procedure LIS-IT(X):
    c[1] := 1
    b[1] := 0
    max := c[1]
    for i ← 2 to n do
        tmp := 0
        ind := 0
        for h ← 1 to i − 1 do
            if (x_h < x_i) ∧ (c[h] > temp) then
                temp := c[h]
                ind := h
            end
            c[i] := 1 + temp
            b[i] := ind
            if c[i] > max then
                max := c[i]
                indMax := i
            end
        end
    end
    return max
```

L'algoritmo ha complessità computazionale

$$T(n) = \mathcal{O}(n^2)$$

e occupa spazio in memoria pari a $\Theta(n)$ (ossia un vettore che contiene i vari c_i; ovviamente non stiamo contando il vettore introdotto per la ricostruzione). L'algoritmo ricorsivo per la ricostruire la LIS è il seguente:

Algorithm 14: Algoritmo di ricostruzione di una LIS di X

```
Procedure PRINT-LIS(i, b, X):
    if b[i] ≠ 0 then
        PRINT-LIS(b[i], b, X)
        Print(x_i)
    else
        Print(x_i)
    end
```

Notare che per ottenere una LIS di X è necessario sapere **in che indice dell'array** c **è presente la massima lunghezza calcolata**; cioè bisogna sapere quale numero, se messo in coda, porta a formare una più lunga sottosequenza strettamente crescente di X (ovviamente ce ne potrebbero essere più di uno, ma noi ne basta uno). Nell'algoritmo bottom-up lo abbiamo salvato nella variabile $indMax$.

2.4 LICS - Longest Increasing Common Subsequence