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
2.2 WIS - Weighted Interval Scheduling 17
2.3 LIS - Longest Increasing Subsequence 23
2.4 LICS - Longest Increasing Common Subsequence 27
2.5 Hateville 32
2.6 Distanza di Edit 38
2.7 Varianti di LCS, LIS e LICS 41
2.7.1 LDCS - Longest Decreasing Common Subsequence 41
2.7.2 LCS in cui non vi sono due simboli consecutivi che si ripetono . 42
2.7.3 LCS in cui si alternano numeri pari e numeri dispari 42
2.7.4 LACS - Longest Alternating Common Subsequence 42
2.7.5 LCS senza due pari consecutivi 45
2.7.6 LCSR: LCS con al massimo R simboli rossi 45
2.7.7 Stabilire se tutte le LCS di X e Y hanno almeno R simboli rossi 48
2.7.8 Stabilire se tutte le LCS di X ed Y hanno numero pari di simboli rossi 49
2.8 Interleaving 50
2.9 Rendere palindroma una stringa 55
2.10 Knapsack 58
2.10.1 Knapsack con oggetti colorati 63
2.10.2 Subset-Sum 66
2.10.3 LCS con ingombri 68

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

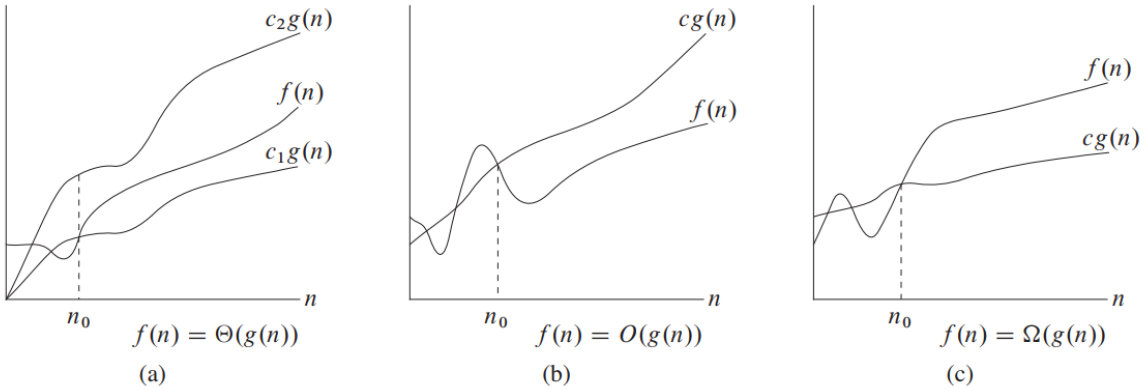
Le notazioni che usiamo per descrivere il tempo di esecuzione asintotico di un algoritmo sono definite in termini di funzioni il cui dominio è l'insieme dei numeri naturali $\mathbb{N} = \{0, 1, ...\}$. In particolare, indichiamo con:

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

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:

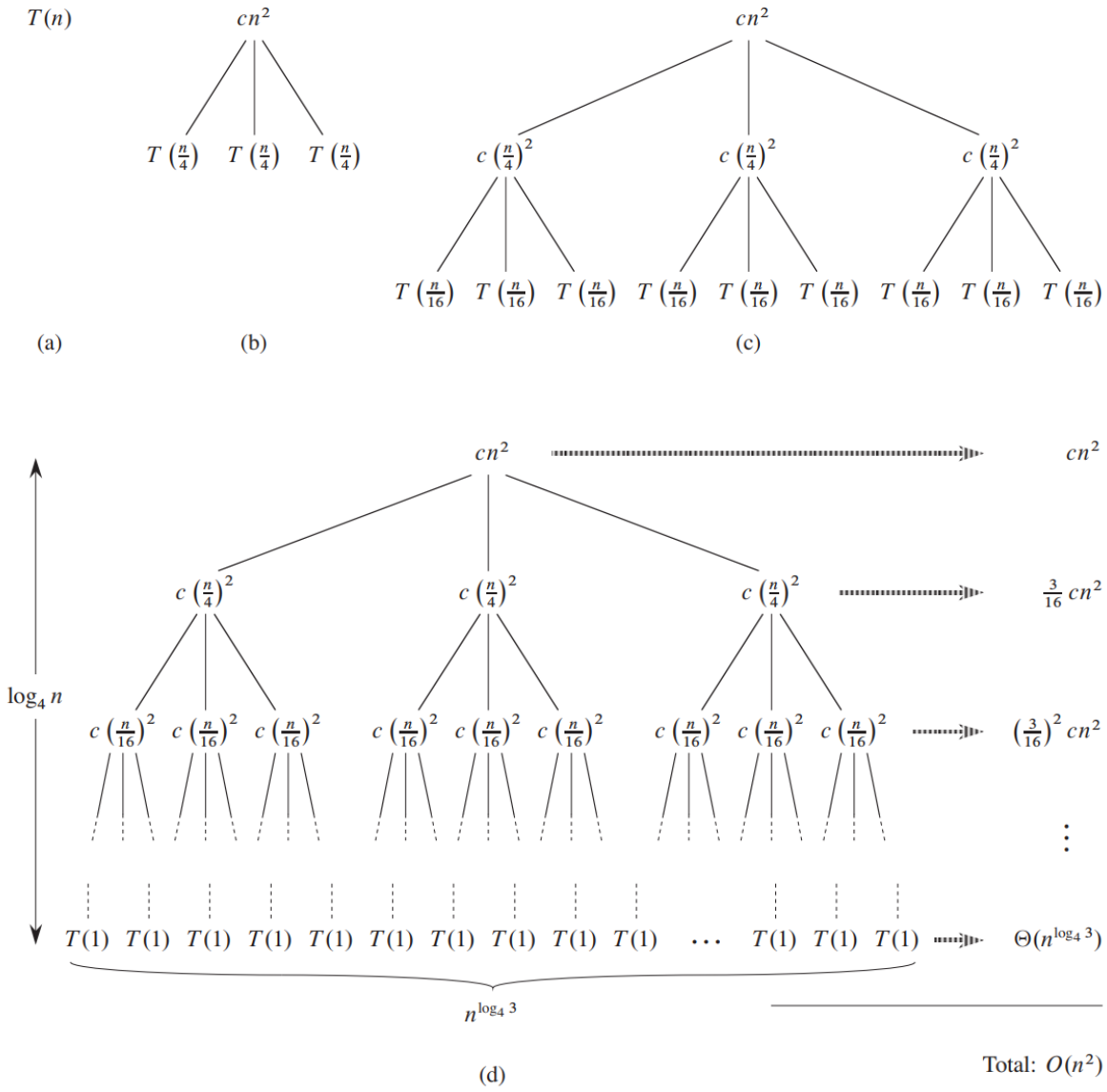
1. Ipotizzare la forma della soluzione
2. Usare l'induzione matematica per trovare le costanti e dimostrare che la soluzione funziona

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 ← 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:

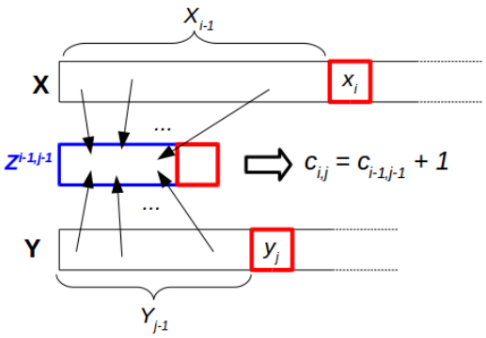
$$c_{i,j} = 0 \text{ se } i = 0 \vee j = 0$$

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:

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

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:

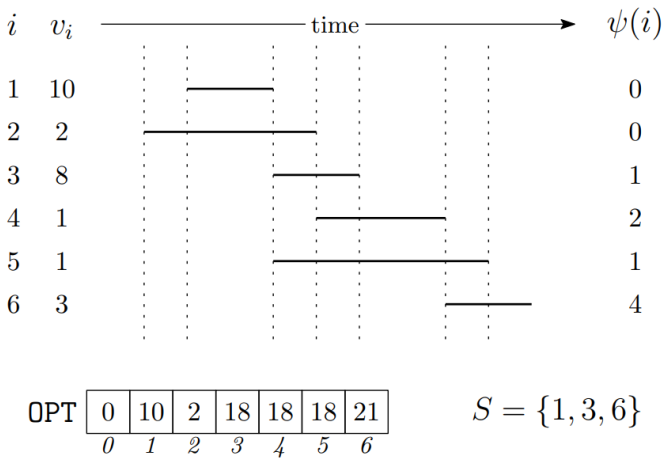
$$OPT_i = \max\{OPT_{i-1}, OPT_{\psi(i)} + v_i\}$$

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 $i \neq S'rightarrowS' \subseteq \{1, \ldots, i-1\}$, 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 sotto-problema**:

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.
ISTANZA: $X_n =< x_1, \ldots, x_n >$
SOLUZIONE: Z una più lunga sottosequenza strettamente crescete di X_n
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:

$$c_i = \text{lunghezza di una tra le più lunghe sottosequenze strettamente crescenti di } X_i$$

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

Il problema LICS è definito come segue:
PROBLEMA: Date due sequenze X e Y, rispettivamente di m ed n numeri interi, si determini **UNA** tra le più lunghe sottosequenze crescenti comuni a X e Y.
ISTANZA: $X_m =< x_1, \ldots, x_m >$ e $Y_n =< y_1, \ldots, y_n >$
SOLUZIONE: Una più lunga sottosequenza crescente di X_m e Y_n.
Quello che si andrà a risolvere sarà comunque un **problema ridotto**, il quale è definito come segue:
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 crescenti comuni a X e Y.
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. Il sottoproblema generico è quindi individuato dalla 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 crescenti comuni al prefisso X_i e al prefisso Y_j.
Non consideriamo coppie (i, j) con $i = 0 \vee j = 0$. Dato che $1 \leq i \leq m$ e $1 \leq j \leq n$, si ottengono $m \cdot n$ sottoproblemi. Ad ogni sottoproblema **è associata una variabile**: considerato il sottoproblema di dimensione (i, j), la variabile ad esso associata è $c_{i,j}$ così definita:

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

per determinare la soluzione di un qualsiasi sottoproblema di dimensione (i, j), oltre all'input del problema, si utilizzeranno anche le soluzioni dei sottoproblemi di dimensione minore. Si noti, però, che ognuna delle variabili associate ai diversi sottoproblemi è da considerare come una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto.
Proviamo ad approcciarci al problema come abbiamo fatto con LCS: se $x_i = y_j$, la struttura della soluzione potrebbe essere:

$$S^{(i,j)} = S|x_i$$

con S una soluzione di un sottoproblema di dimensione minore. Tuttavia, **non sappiamo cosa è contenuto all'interno di** S, quindi **ci mancano delle informazioni necessarie per continuare**. Infatti, date solamente le variabili $\{c_{1,1}, \ldots, c_{i-1,j}, c_{i,j-1}\}$ e i prefissi X_i e Y_j (dei quali, in realtà, ci interessano solamente gli elementi x_i e y_j), non c'è alcun modo per poter comprendere se gli elementi x_i e y_j, nel caso fossero uguali, possano essere accodati alle sottosequenze crescenti relative ai sottoproblemi di dimensioni minore a (i, j): non sappiamo infatti con quale elemento termini ognuna di queste sottosequenze ma ne conosciamo solamente la lunghezza (nel caso del problema ridotto). Si rende quindi necessaria l'introduzione di un **problema ausiliario** nel quale introdurre l'informazione mancante necessaria. Il problema ausiliario è definito come segue:
PROBLEMA AUSILIARIO: Date due sequenze X e Y, rispettivamente di m ed n numeri interi, si determina la lunghezza di una tra le più lunghe sottosequenze crescenti comuni a X e Y la quale termini con x_m e y_n (se questi coincidono, 0 in caso contrario)

In altre parole, si richiede che la sottosequenza soluzione del problema termini con l'ultimo elemento delle sequenze X e Y nel caso siano lo stesso elemento (in caso contrario, **la soluzione non esiste**, il valore 0 per la lunghezza sta ad indicare tale fatto in questo contesto ed esso risulterà utile nel passo ricorsivo). Come per il problema ridotto, anche il problema ausiliario contiene $m \cdot n$ diversi sottoproblemi, ognuno associato ad una differente variabile. Il sottoproblema generico del problema ausiliario di dimensione (i, j) è definito come segue:
SOTTOPROB. GEN. PROB. AUSILIARIO: Date due sequenze X ed Y, rispettivamente di m ed n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze crescenti comuni a X_i e Y_j la quale termini con x_i e y_j (se questi coincidono).
Anche in questo caso, ad ogni sottoproblema è associata una variabile $c_{i,j}$ così definita:

$c_{i,j}$ = lunghezza di una tra le più lunghe sottosequenze crescenti comuni a X_i e Y_j la quale termina con x_i e y_j (se questi coincidono, 0 altrimenti)

Si noti che per un qualunque sottoproblema di dimensione (s, t), solo imponendo che la soluzione $c_{s,t}$ si riferisca ad una sottosequenza crescente comune a X_s e Y_t la quale termini con $x_s = y_t$ è possibile stabilire se un altro elemento $x_u = y_v$ con $u > s$ e $v > t$ possa essere accodato a tale sottosequenza (andando a verificare che $x_s = y_t \leq x_u = y_v$).
Prima di enunciare l'equazione di ricorrenza, può essere utile enunciare e dimostrare fin da subito il **teorema della proprietà della sottostruttura ottima per il problema ausiliario**:

Teorema 2.4.1 (PSO LICS Vincolata) *Sia X una sequenza di m numeri interi e sia X_i un suo prefisso di lunghezza i con $1 \leq i \leq m$. Sia Y una sequenza di n numeri interi e sia Y_j un suo prefisso di lunghezza j con $1 \leq j \leq n$. Sia $Z^{(i,j)}$ una tra le più lunghe sottosequenze crescenti comuni di X_i e Y_j la quale termina con $x_i = y_j$. Allora vale che $Z^{(i,j)} = Z^*|x_i$ (analogamente, $Z^{(i,j)} = Z^*|y_j$) con $Z^* \in \mathcal{W}_{i,j}$ e $|Z^*| = \max_{W \in \mathcal{W}_{i,j}}\{|W|\}$ dove $\mathcal{W}_{i,j}$ è l'insieme di tutte le sottosequenze crescenti comuni (non necessariamente le più lunghe) di X_h e Y_k le quali terminano con $x_h = y_k$ e a cui è*

possibile concatenare x_i (o analogamente y_i), ovvero:

$$\mathcal{W}_{i,j} = \bigcup_{\substack{1\leq h<i \\ 1\leq k<j \\ x_h=y_k<x_i=y_j}} \{W \textit{ sottosequenza crescente di } X_h \textit{ e } Y_k \textit{ che termina con } x_h = y_k\}$$

Si noti che Z^* sarà una soluzione di un qualche sottoproblema di dimensione minore a (i, j) (una più lunga tra le soluzioni di alcuni sottoproblemi più piccoli, in particolare quei sottoproblemi (h, k) con $h < i$ e $k < j$ tali che $x_h = y_k < x_i = y_j$).

Dimostrazione 2.4.1 *Per assurdo supponiamo che $Z^*|x_i$ non sia la soluzione del problema $(i, j) - esimo$. Allora, relativamente alla soluzione $Z^{(i,j)}$ del problema valgono le seguenti affermazioni:*

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

Dove Z' è una qualche sottosequenza comune crescente di un prefisso più piccolo di X_i e Y_j. Sia ora z' l'ultimo elemento di Z', vale quindi che $z' < x_i = y_j$, poiché è stato possibile concatenare x_i (o y_j) a Z'. Inoltre, siano $r < i$ e $s < j$ i ***più grandi indici tali per cui*** *$x_r = y_s = z'$. Di conseguenza, per come è stato definito $\mathcal{W}_{i,j}$, si ottiene che $Z' \in \mathcal{W}_{i,j}$. Infatti Z' è una sottosequenza comune crescente di X_r e Y_s, la quale termina con $x_r < x_i$. Ciò però porta ad una contraddizione: infatti dal punto 2 vale che $|Z'| > |Z^*|$, ma ciò è in contraddizione con l'ipotesi che $|Z^*| = max_{W\in\mathcal{W}_{i,j}}\{|W|\}$, quindi è* ***assurdo!****.*

Scriviamo ora **l'equazione di ricorrenza riferite al problema ausiliario**:
CASO BASE: Se $x_i \neq y_j$
Certamente fanno parte del caso bse tutte le coppie (i, j) tali che $x_i \neq y_j$, ossia quando i due prefissi considerati X_i e Y_j terminano con due elementi diversi. Questo caso, infatti, è semplice da risolvere: per definizione di $c_{i,j}$ gli elementi x_i e y_j devono coincidere, in caso contrario la soluzione è 0 (ossia non c'è nessuna sequenza soluzione). Il numero di sottoproblemi che compone il problema ausiliario sono quindi $m \cdot n$. Il caso base si ha per quei **sottoproblemi di dimensione** (i, j) **con** $i > 0 \wedge j > 0$ **tali che** $x_i \neq y_j$ ed è scrivibile come segue:

$$c_{i,j} = 0$$

Nota 2.4.1 *Quanti casi base invece esistono?* ***Dipende dalle sottosequenze****; infatti non posso saperlo a priori; l'unica informazione che ho al riguardo è che è un* ***numero in funzione dell'input****.*

PASSO RICORSIVO: (i, j) con $i > 0$ e $j > 0$ tali che $x_i = y_j$
IL passo ricorsivo si ha per un qualunque sottoproblema (i, j) tale che $x_i = y_j$, ossia quando i due prefissi X_i e Y_j considerati terminano con lo stesso elemento. In questo caso, la lunghezza della più lunga sottosequenza crescente comune fra X_i e Y_j è uguale alla lunghezza della più lunga sottosequenza comune calcolare per un sottoproblema di dimensione minore la quale termina con un carattere minore di $x_i = y_j$ aumentata

di uno (in quanto si accoda l'elemento comune $x_i = y_j$ alla sottosequenza crescente comune relativa al sottoproblema considerato). Il tutto può essere scritto come:

$$c_{i,j} = 1 + \max\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, x_h < x_i\}$$

Poiché può accadere che l'insieme $\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, x_h < x_i\}$ sia vuoto (il che corrisponde al fatto che l'elemento $x_i = y_j$ è minore di tutti gli elementi precedenti e, quindi, non può essere accodato a nessuna più lunga sottosequenza crescente relativa a sottoproblemi di dimensione minore), assumiamo per definizione che $\max \emptyset = 0$, così che il corrisponde valore di $c_{i,j}$ risulti uguale a 1. Inoltre, si noti che l'insieme $\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, x_h < x_i\}$ corrisponde all'insieme vuoto anche se $i = 1 \vee j = 1$, inquanto i casi con $i = 0 \vee j = 0$ non sono considerati e, pertanto, non esistono sottoproblemi di dimensione minore a quella del sottoproblema considerato (ossia quello con $i = 1 \vee j = 1$).
Una volta calcolati i valori $c_{1,1}, c_{2,1}, c_{1,2}, \ldots, c_{m,n}$ si hanno a disposizione le lunghezze delle sottosequenze comuni massimali fra qualsiasi prefisso di X e qualsiasi prefisso di Y le quali terminano con l'ultimo elemento di entrambi i prefissi $x_i = y_j$. La **soluzione al problema ausiliario** è $c_{m,n}$, mentre **la soluzione al problema ridotto è**:

$$\max\{c_{i,j} | 1 \leq i \leq m \wedge 1 \leq j \leq n\}$$

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

Algorithm 15: Algoritmo ricorsivo che risolve un generico sottoproblema (i, j) di LICS

```
Procedure LICS-RIC(i,j):
    if x_I ≠ y_j then
        return 0
    else
        max := 0 for h ← 1 to i − 1 do
            for k ← 1 to j − 1 do
                if x_h < x_i then
                    S := LICS-RIC(h, k)
                    if S > max then
                        max := S
                    end
                end
            end
        end
    end
    return 1 + max
```

l'algoritmo ricorsivo sopra determina la soluzione $c_{i,j}$ di ogni sottoproblema (si assume di poter accedere alle sequenze X e Y). Questo algoritmo presenta una complessita **esponenziale**, dato che ogni sottoproblema verrà risolto più volte. Per questa ragione, presentiamo anche un algoritmo scritto tramite una tecnica bottom-up che permette di calcolare la soluzione di ogni ogni sottoproblema solamente una volta.

Algorithm 16: Algoritmo iterativo che calcola una LICS tra due sequenze X e Y

```
Procedure LICS-IT(X, Y):
    max := 0
    for i ← 1 to m do
        for j ← 1 to n do
            if x_i ≠ y_j then
                c[i, j] := 0
                b[i, j] := (0, 0)
            else
                temp := 0
                for h ← 1 to i − 1 do
                    for k ← 1 to j − 1 do
                        if (x_h < x_i) ∧ (c[h, k] > temp) then
                            temp := c[h, k]
                            b[i, j] := (h, k)
                        end
                    end
                end
                c[i, j] := 1 + temp
            end
            if c[i, j] > max then
                max := c[i, j]
                maxIdx := (i, j)
            end
        end
    end
    return max
```

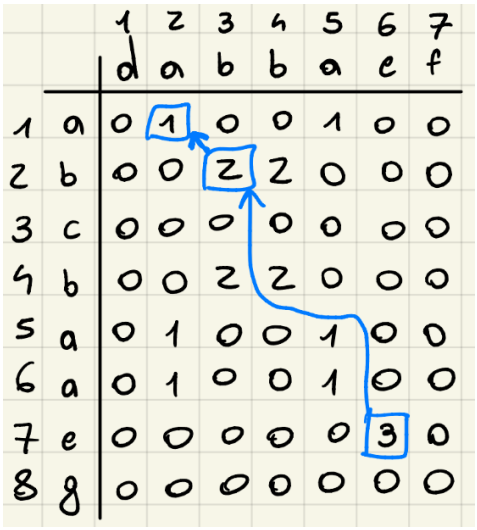
Algorithm 17: Algoritmo di ricostruzione di una LICS tra X e Y

```
Procedure PRINT-LICS(i, j, b, c, X):
    if c[i, j] = 0 then
        print("non esiste nessuna LCS di X_i e Y_j la quale termini con x_i = y_j")
    else
        if b[i, j] ≠ (0, 0) then
            PRINT-LICS(b[i, j], b, c, X)
            print(x_i)
        else
            print(x_i)
        end
    end
```

L'algoritmo che calcola una LICS permette di risolvere il problema in tempo:

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

occupando $\Theta(m \cdot n)$ spazio in memoria (ossia una matrice che contiene i vari $c_{i,j}$; ovviamente non stiamo contando la matrice b). Si noti che abbiamo utilizzato una matrice b per salvare informazioni utili alla ricostruzione



2.5 Hateville

Hateville è una città formata da n case dispose lungo un'unica strada e numerate da 1 a n, in cui si sta organizzando una colletta fondi per la costruzione di un nuovo ospedale. Ogni abitante i della città è interessato a partecipare e ha già comunicato la quantità di denaro che intende versare, ovvero d_i. Ad Hateville però, **ognuno odia i propri vicini, da entrambi i lati**. Ovvero, l'abitante della casa i odia i vicini della casa $i-1$ e $i+1$. Per questo motivo, nessuno vuole partecipare ad una colletta a cui partecipano uno o entrambi i propri vicini. Si vuole quindi determinare **la massima quantità di denaro che può essere raccolta rispettando questa regola**.
Prima di definire il problema, è necessario introdurre alcune definizioni preliminari. Sia $X_n = \{1, \ldots, n\}$ l'insieme che rappresenta gi n abitanti di Hateville e $A \subseteq X_n$ un suo qualunque sottoinsieme. Inoltre, sia $(d_1, \ldots, d_n)$ un vettore di n elementi con $d_i \in \mathbb{N}$ che rappresenta la quantità di denaro che l'abitante i è disposto a versare. Possiamo quindi definire il concetto di **compatibilità** di un sottoinsieme di X_n:

Un sottoinsieme $A \subseteq X_n$ è **compatibile** se e solo se $\forall i \in A$ vale che $i-1 \notin A$ e $i+1 \notin A$ (se esistono)

Ovvero, un sottoinsieme A è compatibile se e solo se per nessuno abitante in A, A contiene un suo vicino. Possiamo rappresentare questo concetto con la funzione $COMP(A)$:

$$COMP(A) = True \text{ se e solo se } \forall i \in A \text{ vale che } i-1 \notin A \text{ e } i+1 \notin A \text{ (se esistono)}$$
$$COMP : \mathcal{P}(X_n) \to \{True, False\}$$

Infine, possiamo definire il concetto di **denaro raccolto** da un sottoinsieme di X_n:

Il **denaro raccolto** da un sottoinsieme $A \subseteq X_n$ è la somma delle quantità di denaro che ogni abitante in A è disposto a versare

Ovvero, il denaro raccolto da un sottoinsieme A è la somma dei valori d_i per ogni $i \in A$. Possiamo rappresentare questo concetto con la funzione $D(A)$:

$$D : \mathcal{P}(X_n) \to \mathbb{R}^+$$

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

Il problema può essere quindi definito come segue:
PROBLEMA: Dato $n \in \mathbb{N}$ il numero degli abitanti, rappresentati nell'insieme $X_n = \{1, 2, \ldots, n\}$ e $d_i, \forall i \in X_n$ la quantità di denaro che l'abitante i è disposto a versare, determinare un sottoinsieme $S \subseteq X_n$ tale che:

$$COMP(S) = True \wedge D(S) = \max_{\substack{A \subseteq X_n \\ COMP(A) = True}} \{D(A)\}$$

Il problema può essere formalizzato come segue:
ISTANZA: $X_n = \{1, \ldots, n\}$ e $d_i, \forall i \in X_n$
SOLUZIONE: $S \subseteq \{1, \ldots, n\}$ tale che:

$$COMP(S_n) = True \wedge D(S_n) = \max_{\substack{A \subseteq X_n \\ COMP(A) = True}} \{D(A)\}$$

Il problema contiene diversi sottoproblemi ognuno dei quali non ha come input l'insieme X_n con $d_i, \forall i \in X_n$ ma un suo sottoinsieme:

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

$$COMP(S_n) = True \wedge D(S_n) = \max_{\substack{A \subseteq X_n \\ COMP(A) = True}} \{D(A)\}$$

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

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

3. ...

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

$$COMP(S_2) = True \wedge D(S_2) = \max_{\substack{A \subseteq X_2 \\ COMP(A) = True}} \{D(A)\}$$

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

$$COMP(S_1) = True \wedge D(S_1) = \max_{\substack{A \subseteq X_1 \\ COMP(A) = True}} \{D(A)\}$$

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

$$COMP(S_0) = True \wedge D(S_0) = \max_{\substack{A \subseteq X_0 \\ COMP(A)=True}} \{D(A)\}$$

Quindi il generico sottoproblema è individuato da $i \in \{0, \ldots, n\}$. Il **generico sottoproblema di dimensione** i è definito come segue:
SOTTOPROBLEMA GENERICO: Dato un insieme $X_i = \{1, \ldots, i\}$ rappresentante i primi i abitanti di Hateville, trovare un suo sottoinsieme, che rispetti la compatibilità, che massimizzi il denaro raccolto.
Ossia:
ISTANZA SOTTOPROB. GEN: $X_i = \{1, \ldots, i\}$ con $d_j, \forall j \in X_i$
SOLUZIONE SOTTOPROB. GEN: $S_i \subseteq \{1, \ldots, i\}$ tale che:

$$COMP(S_i) = True \wedge D(S_i) = \max_{\substack{A \subseteq X_i \\ COMP(A)=True}} \{D(A)\}$$

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) così definita:

$OPT_i = D(S_i)$, ossia il valore di un sottoinsieme di $X_i = \{1, \ldots, i\}$, che rispetti la compatibilità, che massimizzi il denaro raccolto
S_i, un sottoinsieme di X_i che rispetti la compatibilità e che massimizzi il denaro raccolto, pari a OPT_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. Si noti che ognuna delle variabili associate ad un sottoproblema è da considerarsi una **black-box**: si può utilizzare, ma non è possibile conoscerne il contenuto. Scriviamo le equazioni di ricorrenza:
CASO BASE: Se $i = 0 \vee i = 1$
Il caso base si ha per un qualunque sottoinsieme di dimensione i con $i = 0 \vee i = 1$. Nel caso in cui $i = 0$, il sottoproblema di dimensione i è un problema **vuoto**, ossia non ci sono abitanti in Hateville e quindi non c'è nessuno che può partecipare alla colletta. La soluzione del problema vuoto è quindi $OPT_i = 0$ e $S_i = \emptyset$. Nel caso in cui $i = 1$ invece, significa che c'è solo un abitante in Hateville e quindi non è necessario fare nessuna scelta per per evitare di inserire i vicini. La soluzione del sottoproblema di dimensione i considererà quindi solamente l'abitante 1, di conseguenza $OPT_i = d_1$ e $S_i = \{1\}$. Dunque, il caso base è scrivibile come:

$$OPT_i = \begin{cases} 0 & \text{se } i = 0 \\ d_1 & \text{se } i = 1 \end{cases}$$

$$S_i = \begin{cases} \emptyset & \text{se } i = 0 \\ \{1\} & \text{se } i = 1 \end{cases}$$

PASSO RICORSIVO: Se $i > 1$
Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione i con $i > 1$,

ossia quando ci sono almeno due abitanti in Hateville. I dati disponibili per calcolare la soluzione del sottoproblema sono: l'input X_i (in particolare la quantità d_i) e tutte le soluzioni dei sottoproblemi di dimensione minore. Per risolvere un sottoproblema, è necessario domandarsi quali sia la struttura di S_i rispetto a $S_{i-1}, \ldots, S_0$. Distinguiamo due casi:

- Se $i \notin S_i$ allora la soluzione del problema di dimensione i sarà uguale a quella del problema di dimensione S_{i-1}; infatti è sufficiente considerare l'insieme di abitanti $\{1, \ldots, i-1\}$. Di conseguenza:

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

- Se $i \in S_i$ allora non è possibile che il vicino $i-1$ partecipi alla colletta, per la regola della compatibilità. Al contrario però, possiamo aggiungere d_i ad OPT_{i-2} e i a S_{i-2} sicuramente senza violare la compatibilità e, dato che OPT_{i-2} era il massimo per il problema di dimensione $i-2$, aggiungendo d_i avremo anche il massimo per il problema di dimensione i (se $i \in S_i$). Di conseguenza abbiamo che:

$$OPT_i = OPT_{i-2} + d_i \; S_i = S_{i-2} \cup \{i\}$$

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

$$OPT_i = \max\{OPT_{i-1}, OPT_{i-2} + d_i\}$$

e

$$S_i = \begin{cases} S_{i-1} & \text{se } OPT_{i-1} \geq OPT_{i-2} + d_i \\ S_{i-2} \cup \{i\} & \text{altrimenti} \end{cases}$$

	0	0	{}
10	1	10	{1}
5	2	10	{1}
5	3	15	{1,3}
10	4	20	{1,4}
d_i	i	OPT(i)	S_i

Una volta calcolati i valori $OPT_0, OPT_1, \ldots, OPT_n$, è possibile determinare **la soluzione** al problema andando a considerare

$$(OPT_n, S_n)$$

Enunciamo e dimostriamo il **teorema della proprietà della sottostruttura ottima**:

Teorema 2.5.1 (PSO Hateville) *Sia $i \geq 2$. Siano $S_0, S_1, \ldots, S_{i-1}, S_{i-1}$ le soluzioni dei sottoproblemi $0, 1, \ldots, i-2, i-1$. Sia S_i la soluzione del sottoproblema i. Allora:*

$$S_i = \begin{cases} S_{i-2} \cup \{i\} & \text{se } i \in S_i \\ S_{i-1} & \text{se } i \notin S_i \end{cases}$$

Dimostrazione 2.5.1 *Distinguiamo i due casi:*

- *Supponiamo che $i \notin S_i$, devo dimostrare che $S_i = S_{i-1}$. Per assurdo, supponiamo che $S_i \neq S_{i-1}$. Allora deve esistere $S' \neq S_{i-1}$ tale che $D(S') > D(S_{i-1})$; inoltre $i \notin S_i$ quindi $S' \subseteq \{1, \ldots, i-1\}$, ma così S_{i-1} non può più essere soluzione del problema $i-1$-esimo,* ***assurdo!****.*

- *Supponiamo che $i \in S_i$, devo dimostrare che $S_i = S_{i-2} \cup \{i\}$. Per assurdo, supponiamo che $S_i \neq S_{i-2} \cup \{i\}$; allora deve esistere $S' \neq S_{i-2} \cup \{i\}$ tale che S' è soluzione del problema i, quindi tale che $D(S') > D(S_{i-2} \cup \{i\})$; inoltre $S' \subseteq \{1, \ldots, i\}$ perché $i \in S_i$ e, inoltre, $COMP(S_i) = True$. Infine, $S' = S'' \cup \{i\}$ con $S'' \subseteq \{1, \ldots, i-1\}$. Quindi:*

$$D(S') > D(S_{i-2} \cup \{i\})$$

$$D(S'' \cup \{i\}) > D(S_{i-2} \cup \{i\})$$

$$D(S'') + d_i > D(S_{i-2}) + d_i$$

$$D(S'') > D(S_{i-2})$$

Ma quindi S_{i-2} non può più essere soluzione del problema $i-2$; ***assurdo!****.*

Utilizzando l'equazione di ricorrenza, possiamo definire un algoritmo ricorsivo che calcoli la soluzione di ogni sottoproblema:

Algorithm 18: Algoritmo ricorsivo che risolve un generico sottoproblema i di Hateville

```
Procedure HATEVILLE-RIC(i):
    if i = 0 then
        return (0, ∅)
    else
        (V_1, S_1) := HATEVILLE-RIC(i − 1)
        (V_2, S_2) := HATEVILLE-RIC(i − 2)
        V_2 := V_2 + d_i
        S_2 := S_2 ∪ {i}
        if V_1 ≥ V_2 then
            return (V_1, S_1)
        else
            return (V_2, S_2)
        end
    end
```

Questo algoritmo ha **complessità esponenziale**, visto che ogni sottoproblema viene ricalcolato più volte. Definiamo quindi un algoritmo bottom-up che calcoli la soluzione al problema:

Algorithm 19: Algoritmo iterativo che calcola la soluzione al problema Hateville

Procedure `HATEVILLE-IT(`n`)`**:**
- $OPT[0] := 0$
- $S[0] := \emptyset$
- $OPT[1] := d_1$
- $S[1] := \{1\}$
- **for** $i \leftarrow 1$ *to* n **do**
  - $V_1 := OPT[i-1]$
  - $V_2 := OPT[i-2] + d_i$
  - **if** $V_1 \geq V_2$ **then**
    - $S[i] := S[i-1]$
    - $OPT[i] := V_1$
  - **else**
    - $S[i] := S[i-2] \cup \{i\}$
    - $OPT[i] := V_2$
  - **end**
- **end**
- **return** $(OPT[n], S[n])$

L'algoritmo ha tempo di esecuzione pari a:

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

occupando però $\mathcal{O}(n+n^2)$ spazio in memoria (ossia un vettore che contiene i vari OPT_i e un vettore che contiene i vari insiemi S_i, ognuno dei quali contiene al massimo n elementi). Scriviamo l'algoritmo di ricostruzione:

Algorithm 20: Algoritmo che stampa l'insieme S_i di Hateville

Procedure `PRINT-HATEVILLE(`i`)`**:**
- **if** $i \neq 0$ **then**
  - **if** $i = 1$ **then**
    - print(i)
  - **else**
    - **if** $OPTi - 2 + d_i \geq OPT[i-1]$ **then**
      - PRINT-HATEVILLE($i-2$)
      - print(i)
    - **else**
      - PRINT-HATEVILLE($i-1$)
    - **end**
  - **end**
- **end**

2.6 Distanza di Edit

Il problema della distanza di edit può essere definito come segue:
PROBLEMA: Date due sequenze $X =< x_1, \ldots, x_m >$ e $Y =< y_1, \ldots, y_n >$, rispettivamente di lunghezza m ed n definite su un alfabeto Σ, si determini la **minima sequenza delle seguenti operazioni elementari** che permette di trasformare X in Y operando una scansione su X:

- **insert(a)**: Inserisce a nella posizione corrente della sequenza X
- **delete(a)**: cancella a dalla posizione corrente della sequenza X
- **replace(a,b)**: sostituisce a con b nella posizione corrente della sequenza X

Quello che si andrà a risolvere, comunque, è una versione ridotta del problema; la quale è definita come segue:
PROBLEMA RIDOTTO: Date due sequenze $X =< x_1, \ldots, x_m >$ e $Y =< y_1, \ldots, y_n >$, rispettivamente di lunghezza m ed n definite su un alfabeto Σ, si determini il **minimo numero di operazioni elementari** che permette di trasformare X in Y
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**. Il sottoproblema generico è quindi individuato dalla coppia (i, j) ed è definito come segue:
SOTTOPROBLEMA GENERICO: Date due sequenze $X =< x_1, \ldots, x_m >$ e $Y =< y_1, \ldots, y_n >$, rispettivamente di lunghezza m ed n, si determini il minimo numero di operazioni elementari che permette di trasformare X_i in Y_j.
Dato che $0 \leq i \leq m$ e $0 \leq j \leq n$, si ottengono $(m+1) \cdot (n+1)$ sottoproblemi. Ad ogni sottoproblema **è associata una variabile**: considerato il sottoproblema di dimensione (i, j), la variabile ad esso associata è $\delta_{i,j}$ così definita:

$d_{i,j}$ = numero minimo di operazioni elementari che permette di trasformare X_i in Y_j

Per determinare la soluzione di un qualsiasi sottoproblema (i, j), oltre all'input del problema, si utilizzeranno le soluzioni dei sottoproblemi di dimensione minore. Scriviamo le **equazioni di ricorrenza**:
CASO BASE: Se $i = 0 \vee j = 0$
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 cso, è facile ottenere il valore della variabile $\delta_{i,j}$ distinguendo tre casi:

- $i = 0 \wedge j = 0$ allora i prefissi sono entrambi la sequenza vuota; quindi non è necessario alcuna operazione elementare per trasformare X_i in Y_j. Allora

$$\delta_{i,j} = 0$$

- $i = 0 \wedge j > 0$ allora X_i è la sequenza vuota; quindi è necessario eseguire un numero di operazioni elementare pari alla lunghezza del prefisso Y_j (in quanto è necessario inserire tutti gli elementi di Y_j in X_i). Quindi:

$$\delta_{i,j} = j$$

- $i > 0 \wedge j = 0$; allora Y_j è la sequenza vuota; quindi è necessario eseguire un numero di operazioni elementari pari alla lunghezza del prefisso X_i (in quanto è necessario eliminare tutti gli elementi di X_i). Quindi:

$$\delta_{i,j} = i$$

Il caso base è quindi scrivibile come segue:

$$\delta_{i,j} = \begin{cases} 0 & \text{se } i = 0 \wedge j = 0 \\ j & \text{se } i = 0 \wedge j > 0 \\ i & \text{se } i > 0 \wedge j = 0 \end{cases}$$

PASSO RICORSIVO: Se $i > 0 \wedge j > 0$
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 e Y_j entrambi diversi dalla sequenza vuota. I dati disponibili per calcolare $\delta_{i,j}$ sono: l'input X ed in particolare l'elemento x_i, l'input Y ed in particolare l'elemento y_j e tutte le variabili $\{\delta_{0,0}, \ldots, \delta_{i-1,j}, \delta_{i,j-1}\}$. Per calcolare $\delta_{i,j}$ e quindi risolvere il problema di dimensione (i, j) è necessario distinguere i seguenti casi:

- Se $x_i = y_j$; se i due elementi considerati sono identici, allora il numero minimo di operazioni elementari per trasformare X_i in Y_j è uguale al numero minimo di operazioni per trasformare il prefisso X_{i-1} in Y_{j-1}, ovvero:

$$\delta_{i,j} = \delta_{i-1,j-1} \text{ se } x_i = y_j$$

- Se $x_i \neq y_j$, allora il numero minimo di operazioni è dato dalla soluzione di **uno dei sottoproblemi di dimensione minore**. Sarà quindi necessario distinguere in base **all'ultime operazione utilizzata per trasformare X_i in Y_j**:

  - **insert**(y_j): in questo caso, il numero minimo di operazioni per trasformare X_i in Y_j si ottiene a partire dal numero $\delta_{i,j-1}$ di operazioni per trasformare X_i in Y_{j-1} a cui si aggiunge l'operazione di inserimento dell'elemento y_j in X_i; ovvero:

$$\delta_{i,j} = \delta_{i,j-1} + 1$$

    dove 1 rappresenta il contributo dato dall'operazione **insert**(y_j)

  - **delete**(x_i): in questo caso, il numero minimo di operazioni per trasformare X_i in Y_j si ottiene a partire dal numero $\delta_{i-1,j}$ di operazioni per trasformare X_{i-1} in Y_j a cui si aggiunge l'operazione di cancellazione dell'elemento x_i da X_i; ovvero:

$$\delta_{i,j} = \delta_{i-1,j} + 1$$

    dove 1 rappresenta il contributo dato dall'operazione **delete**(x_i).

  - **replace**(x_i, y_j): in questo caso, il numero minimo di operazioni per trasformare X_i in Y_j si ottiene a partire dal numero $\delta_{i-1,j-1}$ di operazioni per trasformare X_{i-1} in Y_{j-1} a cui si aggiunge l'operazione di sostituzione dell'elemento x_i con y_j in X_i; ovvero:

$$\delta_{i,j} = \delta_{i-1,j-1} + 1$$

    dove 1 rappresenta il contributo dato dall'operazione **replace**(x_i, y_j).

Siccome la serie di operazioni **non è nota**, non possiamo sapere a priori quale dei tre casi si verifichi. Quindi consideriamo la soluzione dei tre possibili sotto-problemi e scegliamo quella di valore minimo. Formalmente, possiamo scrivere:

$$\delta_{i,j} = \min\{\delta_{i,j-1} + 1, \delta_{i-1,j} + 1, \delta_{i-1,j-1} + 1\} \text{ se } x_i \neq y_j$$

Il passo ricorsivo è quindi scrivibile come:

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

Nota 2.6.1 *L'equazione di ricorrenza* ***è la proprietà della sottostruttura ottima***

Una volta calcolati i valori $\delta_{0,0}, \delta_{0,1}, \ldots, \delta_{m,n}$ si hanno a disposizione tutte le lunghezze delle sequenze di operazioni per rendere ciascun prefisso X_i uguale al prefisso Y_j. La **soluzione del problema** è quindi il valore $\delta_{m,n}$.

		0	1	2	3	4	5	6	j
		ε	P	R	E	S	T	O	y_j
0	ε	0	1	2	3	4	5	6	
1	R	1	1	1	2	3	4	5	
2	I	2	2	2	2	3	4	5	
3	S	3	3	3	3	2	3	4	
4	O	4	4	4	4	3	3	3	
5	T	5	5	5	5	4	3	4	
6	T	6	6	6	6	5	4	4	
7	O	7	7	7	7	6	5	4	
i	x_i								

Algorithm 21: Algoritmo ricorsivo che calcola la distanza di edit per due prefissi X_i e Y_j

```
Procedure ED-RIC(i,j):
    if i = 0 ∨ j = 0 then
        if i = 0 then
            return j
        else
            return i
        end
    else
        if x_i = y_j then
            return ED-RIC(i − 1, j − 1)
        else
            ins := 1 + ED-RIC(i, j − 1)
            del := 1 + ED-RIC(i − 1, j)
            rep := 1 + ED-RIC(i − 1, j − 1)
            return MIN(ins, del, rep)
        end
    end
```

L'algoritmo sopra calcola ricorsivamente la soluzione ad un generico sottoproblema (i, j); tuttavia il suo tempo di calcolo è **esponenziale**. Scriviamo quindi un algoritmo bottom-up che calcoli la soluzione al problema:

Algorithm 22: Algoritmo iterativo che calcola la distanza di edit tra due sequenze X e Y

```
Procedure ED-IT(X, Y):
    for i ← 0 to m do
        δ[i, 0] := i
    end
    for j ← 0 to n do
        δ[0, j] := j
    end
    for i ← 1 to m do
        for j ← 1 to n do
            if x_i = y_j then
                δ[i, j] := δi − 1, j − 1
            else
                δ[i, j] := 1 + MIN(δ[i, j − 1], δ[i, j − 1], δ[i − 1, j − 1])
            end
        end
    end
    return δ[m, n]
```

L'algoritmo bottom-up ha complessità:

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

e occupa spazio in memoria pari a $\Theta(m \cdot n)$ (ossia una matrice per contenere i vari $\delta_{i,j}$).

2.7 Varianti di LCS, LIS e LICS

Presentiamo di seguito, in maniera più o meno formale, delle varianti dei problemi LCS, LIS e LICS.

2.7.1 LDCS - Longest Decreasing Common Subsequence

In questa variante di LICS, avviene la seguente variazione all'interno della definizione dell'equazione di ricorrenza:

- Se $x_i \neq y_j$ allora
$$c_{i,j} = 0$$
- Se $x_i = y_j$ allora
$$c_{i,j} = \max\{c_{h,k} | 1 \leq h < i \wedge 1 \leq k < j \wedge x_h > x_i\} + 1$$

la modifica dell'algoritmo bottom-up, per fare in modo che esso rispetti la nuova definizione della variabile $c_{i,j}$, risulta quindi triviale.

2.7.2 LCS in cui non vi sono due simboli consecutivi che si ripetono

In questa variante di LCS, vi è la seguente variazione nell'equazione di ricorrenza:

$$c_{i,j} = \max\{c_{h,k} | 1 \leq h < i \wedge 1 \leq k < j \wedge x_h \neq x_i\} + 1$$

la modifica dell'algoritmo bottom-up, per fare in modo che esso rispetti la nuova definizione della variabile $c_{i,j}$, risulta quindi triviale.

2.7.3 LCS in cui si alternano numeri pari e numeri dispari

In questa variante di LCS, vi è la seguente variazione nell'equazione di ricorrenza:

$$c_{i,j} = \max\{c_{h,k} | 1 \leq h < i \wedge 1 \leq k < j \wedge x_h \bmod 2 \neq x_i \bmod 2\} + 1$$

la modifica dell'algoritmo bottom-up, per fare in modo che esso rispetti la nuova definizione della variabile $c_{i,j}$, risulta quindi triviale.

2.7.4 LACS - Longest Alternating Common Subsequence

Sia C un insieme di colori e $col : \Sigma \to C$ una funzione che attribuisce ad ogni simbolo dell'alfabeto Σ un colore. Possiamo quindi definire così il problema:
PROBLEMA: Data due sequenze X e Y, rispettivamente di m e n numeri interi, si determini **UNA** tra le più lunghe sottosequenze comuni a X e Y la quale non abbia elementi consecutivi dello stesso colore.
Quello che si andrà a risolvere, comunque, è una versione ridotta del problema, la quale è definita come segue:
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 che non abbia elementi consecutivi dello stesso colore.
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. Il generico sottoproblema (i, j) è quindi così definito
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 a X_i e Y_j la quale non abbia elementi consecutivi dello stesso colore.
Non consideriamo le coppie $i = 0 \vee j = 0$. Dato che $1 \leq i \leq m$ e $1 \leq j \leq n$, si ottengono $m \cdot n$ sottoproblemi. Ad ogni sottoproblema (i, j) è associata **una variabile** così definita:

$$c_{i,j} = \text{lunghezza di una tra le più lunghe sottosequenze comuni a } X_i \text{ e } Y_j \text{ la quale non ha elementi consecutivi dello stesso colore}$$

Per determinare la soluzione di un qualsiasi sottoproblema di dimensione (i, j), oltre all'input del problema, si utilizzeranno le soluzioni dei sottoproblemi di dimensione minore. 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. Il problema così definito **non è risolvibile poiché ci manca informazione**: date solamente le variabili $\{c_{1,1}, \ldots, c_{i-1,j}, c_{i,j-1}\}$ e i prefissi X_i e Y_j (dei quali, in realtà, ci interessano solo gli elementi x_i e y_j), non c'è alcun modo per poter comprendere se

gli elementi x_i e y_j, nel caso fossero uguali, possano essere accodati alle sottosequenze comuni alternanti relative ai sottoproblemi di dimensione minore a (i, j): non sappiamo **con quale elemento termini ognuna di queste sottosequenze**, ma ne conosciamo solamente la lunghezza. Risulta necessario quindi introdurre un **problema vincolato** nel quale introdurre l'informazione mancante necessaria:
PROBLEMA VINCOLATO: 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 la quale non abbia elementi consecutivi dello stesso colore e la quale termini con x_m e y_n (se questi coincidono, 0 in caso contrario).
Come per il problema ridotto, anche il problema vincolato contiene $m \cdot n$ diversi sottoproblemi, ognuno **associato ad una differente variabile**. Il sottoproblema generico del problema vincolato di dimensione (i, j) è definito come segue:
SOTTOPROB. GEN. PROBL. VINCOLATO: 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_i e Y_j la quale non abbia elementi consecutivi dello stesso colore e la quale termini con x_i e y_j (se questi coincidono, 0 in caso contrario).
il sottoproblema è associato alla variabile $c_{i,j}$ così definita:

$c_{i,j}$ = lunghezza di una tra le più lunghe sottosequenze comuni a X_i e Y_j la quale non ha elementi consecutivi dello stesso colore e la quale termina con x_i e y_j (se questi coincidono)

Si noti quindi che per un qualunque sottoproblema di dimensione (s, t), solo imponendo che la soluzione $c_{s,t}$ si riferisca ad una sottosequenza comune a X_s e Y_t la quale non presenta elementi consecutivi dello stesso colore e la quale termina con $x_s = y_t$ è possibile stabilire se un altro elemento $x_u = y_v$ con $u > s$ e $v > t$ possa essere accodato a tale sottosequenza (andando a verificare che $col(x_s) = col(y_t) \neq col(x_u) = col(y_v)$).
Scriviamo le **equazioni di ricorrenza**:
CASO BASE: (i, j) con $i > 0 \wedge j > 0$ tali che $x_i \neq y_j$
Certamente fanno parte del caso base tutte le coppie (i, j) tali che $x_i \neq y_j$, ossia quando i due prefissi X_i e Y_j terminano con due elementi diversi. Questo caso, infatti, è semplice da risolvere: per definizione di $c_{i,j}$, gli elementi x_i e y_j devono coincidere, in caso contrario la soluzione è 0 (ossia non c'è nessuna soluzione). Il numero di sottoproblemi che compone il problema vincolato possono essere ridotti a $m \cdot n$. Allora $c_{i,j}$ è scrivibile come:

$$c_{i,j} = 0$$

PASSO RICORSIVO: (i, j) con $i > 0 \wedge j > 0$ tali che $x_i = y_j$
Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione (i, j) tale che $x_i = y_j$, ossia quando i due prefissi X_i e Y_j considerati terminano con lo stesso elemento. In questo caso, la lunghezza della più lunga sottosequenza con colori alternanti comune fra X_i e Y_j è uguale alla lunghezza della più lunga sottosequenza comune con colori alternanti calcolata per un sottoproblema di dimensione minore e che termina con un carattere con colore diverso da quello di $x_i = y_j$ aumentata di uno. Il tutto può essere scritto come:

$$c_{i,j} = 1 + \max\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, col(x_i) \neq col(x_h)\}$$

Poiché può accadere che l'insieme $\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, col(x_i) \neq col(x_h)\}$ sia vuoto (il che corrisponde al fatto che l'elemento $x_i = y_j$ è colorato allo stesso modo

di tutti gli elementi precedenti e, quindi, non può essere accodato a nessuna più lunga sottosequenza alternante relativa a sottoproblemi di dimensione minore), assumiamo per definizione che $\max \emptyset = 0$, così che il corrispondente valore di $c_{i,j}$ risulti uguale a 1. Inoltre, si noti che l'insieme $\{c_{h,k} | 1 \leq h < i, 1 \leq k < j, col(x_i) \neq col(x_h)\}$ corrisponde all'insieme vuoto anche se $i = 1 \vee j = 1$, in quanto i casi $i = 0 \vee j = 0$ non sono considerati e, pertanto, non esistono sottoproblemi di dimensione minore a quella del sottoproblema considerato.
Una volta calcolati i valori $c_{1,1}, c_{2,1}, \ldots, c_{m,n}$, si hanno a disposizione tutte le lunghezze delle sottosequenze comuni massimali fra qualsiasi prefisso di X e qualsiasi prefisso di Y le quali non hanno elementi consecutivi dello stesso colore e le quali terminano con l'ultimo elemento di entrami i prefissi. La **soluzione al problema vincolato** è $c_{m,n}$ mentre la **soluzione al problema ridotto** è:

$$\max\{c_{i,j} | 1 \leq i \leq m \wedge 1 \leq j \leq n\}$$

			1	2	3	4	5	6	7	8	**j**
			2	7	4	23	21	14	1	8	y_j
			r	b	r	g	g	g	r	b	$\varphi(y_j)$
1	2	r	1	0	0	0	0	0	0	0	
2	4	r	0	0	1	0	0	0	0	0	
3	7	b	0	2	0	0	0	0	0	0	
4	11	g	0	0	0	0	0	0	0	0	
5	21	g	0	0	0	0	3	0	0	0	
6	14	g	0	0	0	0	0	3	0	0	
7	1	r	0	0	0	0	0	0	4	0	
i	x_i	$\varphi(x_i)$									

Utilizzando le equazioni di ricorrenza, scriviamo un algoritmo ricorsivo che calcola la soluzione ad un generico sottoproblema (i, j):

Algorithm 23: Algoritmo ricorsivo che calcola la soluzione ad un gen. sottoprob. (i, j) di LACS

```
Procedure LACS-RIC(i, j):
    if x_i ≠ y_j then
        return 0
    else
        max := 0
        for h ← 1 to i − 1 do
            for k ← 1 to j − 1 do
                if col(x_h) ≠ col(x_i) then
                    S := LACS-RIC(h, k)
                    if S > max then
                        max := S
                    end
                end
            end
        end
    end
    return 1 + max
```

l'algoritmo sopra ha **complessità esponenziale**; scriviamo quindi un algoritmo bottom-up che calcoli la soluzione al problema:

Algorithm 24: Algoritmo bottom-up che calcola la soluzione al problema LACS

```
Procedure LACS-IT(X, Y):
    max := 0
    for i ← 1 to m do
        for j ← 1 to n do
            if x_i ≠ y_j then
                c[i, j] := 0
            else
                temp := 0
                for h ← 1 to i − 1 do
                    for k ← 1 to j − 1 do
                        if (col(x_h) ≠ col(x_i)) ∧ (c[h, k] > temp) then
                            temp := c[h, k]
                        end
                    end
                end
                c[i, j] := 1 + temp
            end
            if c[i, j] > max then
                max := c[i, j]
            end
        end
    end
    return max
```

L'algoritmo ha complessità computazionale pari a:

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

e occupa uno spazio pari a $\Theta(m \cdot n)$ (ossia una matrice che contiene i vari $c_{i,j}$).

2.7.5 LCS senza due pari consecutivi

In questa variante di LCS, vi è la seguente variazione alla definizione della variabile:

$$c_{i,j} = \max\{c_{h,k} | 1 \leq h < i \wedge 1 \leq k < j \wedge x_h \bmod 2 \neq 0 \wedge x_i \bmod 2 \neq 0\} + 1$$

la modifica dell'algoritmo bottom-up, per fare in modo che esso rispetti la nuova definizione della variabile $c_{i,j}$, risulta quindi triviale.

2.7.6 LCSR: LCS con al massimo R simboli rossi

ISTANZA: X_m e Y_n sequenze
SOLUZIONE: Calcolare la lunghezza di una più lunga sottosequenza di X_m e Y_n la quale ha al massimo R simboli colorati di rosso.
Si assume di avere una funzione $col : \Sigma \to \{Red, Blue\}$ e che ad ogni simbolo all'interno

delle due sequenze ha un colore che è il rosso oppure il blu.
Ogni **sottoproblema generico** è quindi **identificato da una tripla** (i, j, r) con:

$$i \in \{0, \ldots, m\}$$

$$j \in \{0, \ldots, m\}$$

$$r \in \{0 \ldots, R\}$$

ad ogni sottoproblema è **associata una variabile** così definita:

$c_{i,j,r}$ = la lunghezza di una più lunga sottosequenza comune di X_i e Y_j con r simboli rossi

Definiamo le **equazioni di ricorrenza**:
CASO BASE: (i, j, r) con $i = 0 \vee j = 0, \forall r \in \{0, \ldots, R\}$
In questo caso, una (o entrambe) le sottosequenze sono **vuote**, quindi la lunghezza di una più lunga sottosequenza comune a X_i e Y_j con r simboli rossi è:

$$c_{i,j,r} = 0$$

PASSO RICORSIVO: (i, j, r) con $i > 0 \wedge j > 0$ e con r qualsiasi
Visto che $i > 0 \wedge j > 0$, allora entrambe i prefissi considerati **non sono la sequenza vuota**. Dobbiamo distinguere due casi:

- Se $x_i \neq y_j$ allora significa che dobbiamo prendere il valore massimo tra i due problemi di dimensione $(i-1, j, r)$ e $(i, j-1, r)$; quindi la variabile avrà valore

$$c_{i,j,r} = \max\{c_{i-1,j,r}, c_{i,j-1,r}\}$$

- Se $x_i = y_j$ allora significa che $x_i = y_j$ potrebbe appartenere alla sottosequenza soluzione del problema. Dobbiamo quindi distinguere altri due sotto-casi:
  - Se $col(x_i) = Red$ allora dobbiamo stare attenti a **quanti simboli rossi sono già presenti all'interno della sottosequenza**
    - Se $r = 0$ allora
    $$c_{i,j,r} = c_{i-1,j-1,0}$$
    cioè **non posso aggiungere il simbolo** x_i poiché "non c'è più spazio" per ulteriori simboli rossi
    - Se $0 < r \leq R$ allora
    $$c_{i,j,r} = c_{i-1,j-1,r-1} + 1$$
  - Se $col(x_i) \neq Red$ allora
  $$c_{i,j,r} = c_{i-1,j-1,r} + 1$$

Il passo ricorsivo è quindi riassumibile come segue:

$$c_{i,j,r} = \begin{cases} \max\{c_{i-1,j,r}, c_{i,j-1,r}\} & \text{se } x_i \neq y_j \\ \begin{cases} \begin{cases} c_{i-1,j-1,0} & \text{se r} = 0 \\ c_{i-1,j-1,r} + 1 & \text{se } 0 < r \leq R \end{cases} & \text{se } col(x_i) = Red \\ c_{i-1,j-1,r} + 1 & \text{se } col(x_i) \neq Red \end{cases} & \text{se } x_i = y_j \end{cases}$$

Scriviamo quindi l'algoritmo bottom-up per calcolare la soluzione al problema

Algorithm 25: Algoritmo iterativo che calcola al soluzione al problema LCSR

```
Procedure LCSR-IT(X,Y,R):
    m := LENGTH(X)
    n := LENGTH(Y)
    for i ← 0 to m do
        for r ← 0 to R do
            c[i,0,r] := 0
        end
    end
    for j ← 0 to n do
        for r ← 0 to R do
            c[0,j,r] := 0
        end
    end
    for i ← 1 to m do
        for j ← 1 to n do
            for r ← 0 to R do
                if x_i ≠ y_j then
                    c[i,j,r] := max{c[i-1,j,r], c[i,j-1,r]}
                else
                    if col(x_i) ≠ Red then
                        c[i,j,r] := c[i-1,j-1,r] + 1
                    else
                        if r = 0 then
                            c[i,j,r] := c[i-1,j-1,0]
                        else
                            c[i,j,r] := c[i-1,j-1,r-1] + 1
                        end
                    end
                end
            end
        end
    end
    return c[m,n,R]
```

l'algoritmo ha complessità computazionale pari a:

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

Presentiamo inoltre l'algoritmo di ricostruzione:

Algorithm 26: Algoritmo che stampa la soluzione al problema LCSR

```
Procedure PRINT-LCSR(X,Y,i,j,r):
    if i ≠ 0 ∧ j ≠ 0 then
        if x_i ≠ y_j then
            if c[i − 1, j, r] ≥ c[i, j − 1, r] then
                PRINT-LCSR(X, Y, i-1, j, r)
            else
                PRINT-LCSR(X, Y, i, j-1, r)
            end
        else
            if col(x_i) ≠ Red then
                PRINT-LCSR(X, Y, i-1, j-1, r)
                print(x_i)
            else
                if r = 0 then
                    PRINT-LCSR(X,Y, i-1, j-1, 0)
                else
                    PRINT-LCSR(X, Y, i-1, j-1, r-1)
                    print(x_i)
                end
            end
        end
    end
```

2.7.7 Stabilire se tutte le LCS di X e Y hanno almeno R simboli rossi

Siano X e Y due sequenze, rispettivamente di m ed n simboli, costruite sull'alfabeto Σ e sia *col* la funzione:

$$col : \Sigma \to \{Red, Blue\}$$

Il problema è quindi così definito:
PROBLEMA: Stabilire se tutte le LCS di X ed Y hanno almeno R simboli rossi
Ogni sottoproblema generico è individuato da una tripla (i, j, r) con $r \in \{0, \ldots, R\}$ ed è così definito
SOTTOPROBLEMA GENERICO: Stabilire se tutte le LCS di X_i e Y_j hanno almeno R simboli rossi.
Ad ogni sottoproblema è **associata una variabile** così definita:

$$e_{i,j,r} = \begin{cases} True & \text{Se tutte le LCS di } X_i \text{ e } Y_j \text{ hanno almeno } R \text{ simboli rossi} \\ False & \text{Altrimenti} \end{cases}$$

Scriviamo **l'equazione di ricorrenza**:
CASO BASE: (i, j, r) con $(i = 0 \vee j = 0)$ e $\forall r \in \{0, \ldots, R\}$
In questo caso, abbiamo che:

$$e_{i,j,r} = \begin{cases} True & \text{se } r = 0 \\ False & \text{se } r > 0 \end{cases}$$

PASSO RICORSIVO: (i, j, r) con $i > 0 \wedge j > 0$ e $\forall r \in \{0, \ldots, R\}$
Assumiamo di aver **già calcolato le lunghezze di tutte le LCS tra X_i e Y_j** e di averle salvate nelle variabili $c_{i,j}$ (asintoticamente, questo non modifica i tempi dell'algoritmo). Dobbiamo in questo caso distinguere diversi casi:

- Se $x_i \neq y_j$ allora:

$$e_{i,j,r} = \begin{cases} e_{i-1,j,r} & \text{Se } c_{i-1,j} > c_{i,j-1} \\ e_{i,j-1,r} & \text{Se } c_{i,j-1} > c_{i-1,j} \\ e_{i-1,j,r} \wedge e_{i,j-1,r} & \text{Se } c_{i-1,j} = c_{i,j-1} \end{cases}$$

- Se $x_i = y_j$ dobbiamo distinguere altri 3 sotto-casi:
  - Se $col(x_i) \neq Red$ allora:
  $$e_{i,j,r} = c_{i-1,j-1,r}$$
  - Se $col(x_i) = Red \wedge r > 0$ allora:
  $$e_{i,j,r} = c_{i-1,j-1,r-1}$$
  - Se $col(x_i) = Red \wedge r = 0$ allora:
  $$e_{i,j,r} = c_{i-1,j-1,0} = True$$

La **soluzione al problema** sarà in $e_{m,n,R}$.

2.7.8 Stabilire se tutte le LCS di X ed Y hanno numero pari di simboli rossi

Assumiamo di essere nella stessa situazione del paragrafo precedente per quanto riguarda X, Y e col. Il problema è quindi così definito:
PROBLEMA: Stabilire se tutte le LCS di X e Y hanno un numero pari di simboli rossi.
Ogni sottoproblema è individuato da una tripla (i, j, p) con $p = 0$ che indica che il sottoproblema ha **un numero pari di simboli rossi** e $p = 1$ che indica l'opposto. Ad ogni sottoproblema è associata **una variabile** così definita:

$$e_{i,j,p} = \begin{cases} True & \text{Se } p = 0 \\ False & \text{Se } p = 1 \end{cases}$$

Scriviamo **l'equazione di ricorrenza**:
CASO BASE: (i, j, p) con $i = 0 \vee j = 0$ e $p \in \{0, 1\}$
In questo caso, la variabile è così definita:

$$e_{i,j,p} = \begin{cases} True & \text{Se } p = 0 \\ False & \text{Se} P = 1 \end{cases}$$

PASSO RICORSIVO: (i, j, p) con $i > 0 \wedge j > 0$ e $p \in \{0, 1\}$
Assumiamo di aver calcoalto la lunghezze di tutte le LCS tra X ed Y e di averle salvate nelle variabili $c_{i,j}$. In caso, dobbiamo distinguere i diversi casi:

- Se $x_i \neq y_j$ allora la variabile è così definita, per ogni valore di p:

$$e_{i,j,p} = \begin{cases} e_{i-1,j,p} & \text{Se } c_{i-1,j} > c_{i,j-1} \\ e_{i,j-1,p} & \text{Se } c_{i-1,j} < c_{i,j-1} \\ e_{i,j-1,p} \wedge e_{i-1,j,p} & \text{Se } c_{i-1,j} = c_{i,j-1} \end{cases}$$

- Se $x_i = y_j$ allora dobbiamo distringuere altri due sottocasi:
  - Se $col(x_i) = Red$ allora la variabile è così definita:

$$e_{i,j,p} = \begin{cases} e_{i-1,j-1,1} & \text{Se } p = 0 \\ e_{i-1,j-1,0} & \text{Se } p = 1 \end{cases}$$

  - Se $col(x_i) \neq Red$ allora la variabile è così definita:

$$e_{i,j,p} = e_{i-1,j-1,p}$$

La **soluzione** del problema è nella variabile $e_{m,n,0}$.

2.8 Interleaving

Il problema è definito come segue:
PROBLEMA: Date tre sequenze, definite su un alfabeto Σ, $X =< x_1, \ldots, x_m >$ di lunghezza m, $Y =< y_1, \ldots, y_n >$ di lunghezza n e $W =< w_1, \ldots, w_{m+n} >$ di lunghezza $m+n$, stabilire se W è un interleaving di X ed Y, ovvero se X e Y si possono trovare come due sottosequenze disgiunte in W. Ovverro:

- $\exists\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, m+n\}$ con $i_1 < i_2 < \cdots < i_m$
- $\exists\{j_1, \ldots, j_n\} \subseteq \{1, \ldots, m+n\}$ con $j_1 < j_2 < \cdots < i_n$

Tali che:

- $\{i_1, \ldots, i_m\} \cap \{j_1, \ldots, j_n\} = \emptyset$
- $\{i_1, \ldots, i_m\} \cup \{j_1, \ldots, j_n\} = \{1, \ldots, m+n\}$
- $\forall k \in \{1, \ldots, m\}, x_k = w_{i,k}$
- $\forall h \in \{1, \ldots, n\}, y_h = w_{j,h}$

Il problema contiene diversi sottoproblemi, ognugno dei quali non ha come input (X, Y, Z) ma i prefissi di tali sequenze. Ogni sottoproblema è quindi definito da una coppia (i, j) ed è definito come segue:
SOTTOPROBLEMA GENERICO: Date tre sequenze X, Y e W, rispettivamente di lunghezza m, n e $m+n$, si determini se W_{i+j} è interleaving di X_i e Y_j.
Dato che $0 \leq i \leq m$ e $0 \leq j \leq n$, si ottengono $(m+1) \cdot (n+1)$ sottoproblemi. Ad ogni sottoproblema **è associata una variabile** così definita:

$$s_{i,j} = \begin{cases} True & \text{Se } W_{i+j} \text{ è interleaving di } X_i \text{ e } Y_j \\ False & \text{Altrimenti} \end{cases}$$

Ovvero, la variabile $s_{i,j}$ è la soluzione del sottoproblema (i, j). Per determinare la soluzione di un qualsiasi sottoproblema di dimensione (i, j), oltre all'input del problema, si utilizzeranno le soluzioni dei sottoproblemi di dimensione minore. Si noti, però, che ognuna delle variabili associate ai sottoproblemi è una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto. Scriviamo le **equazioni di ricorrenza**:

- **PRIMO CASO**: (i, j) con $i = 0 \vee j = 0$
  Concentriamoci inizialmente su 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, il valore della variabile $s_{i,j}$ sarà, considerando i diversi casi possibili:

  - Se $i = 0 \wedge j = 0$ allora entrambi i prefissi sono la sequenza vuota, allora W_{i+j} è la sequenza vuota e di conseguenza sarà interleaving di X_i e Y_j. Quindi

    $$s_{i,j} = True$$

  - Se $i = 0 \wedge j > 0$ in questo caso bisognerà considerare altre due situazioni diverse:
    * Se $w_i = y_j$, allora W_{i+j} è interleaving di X_i e Y_j solo se lo era anche W_{i+j-1}; in questo caso quindi

      $$s_{i,j} = s_{i,j-1}$$

    * Se $w_j \neq y_j$ allora W_{i+j} non può essere interleaving dI X_i e Y_j, quindi:

      $$s_{i,j} = False$$

  - Se $i > 0 \wedge j = 0$, anche in questo caso bisogna considerare due situazioni:
    * Se $w_i = x_i$ allora W_{i+j} è interleaving di X_i e Y_j solo se lo era anche W_{i+j-1}; in questo caso quindi

      $$s_{i,j} = s_{i-1,j}$$

    * Se $w_i \neq x_i$ allora W_{i+j} non può essere interleaving di X_i e Y_j; in questo caso quindi

      $$s_{i,j} = False$$

  Possiamo quindi riassumere questo primo caso con la seguente equazione:

  $$s_{i,j} = \begin{cases} True & \text{Se } i = 0 \wedge j = 0 \\ s_{i,j-1} & \text{Se } i = 0 \wedge j > 0 \wedge w_j = y_j \\ False & \text{Se } i = 0 \wedge j > 0 \wedge w_j \neq y_j \\ s_{i-1,j} & \text{Se } i > 0 \wedge j = 0 \wedge w_i = x_i \\ False & \text{Se } i > 0 \wedge j = 0 \wedge w_i \neq x_i \end{cases}$$

- **SECONDO CASO**: (i, j) con $i > 0 \wedge j > 0$
  Ora invece consideriamo un qualunque sottoproblema di dimensione (i, j) con

$i > 0 \wedge j > 0$, ossia quando si vanno a considerare due prefissi X_i e Y_j entrambi diversi dalla sequenza vuota. I dati disponibili per calcolare $s_{i,j}$ sono: l'input X, ed in particolare l'elemento x_i, l'input Y, e in particolare l'elemento y_j, e tutte le variabili $\{s_{0,0}, \ldots, s_{i-1,j}, s_{i,j-1}\}$. Per calcolare $s_{i,j}$ e quindi risolvere il sottoproblema di dimensione (i, j) è necessario distingere i seguenti casi:

- Se $w_{i+j} \neq x_i \wedge w_{i+j} \neq y_j$ allora non possono essere accodati né x_i né y_j al simbolo in posizione w_{i+j}; allora W_{i+j} non può essere interleaving di X_i e Y_j e quindi:
$$s_{i,j} = False$$
- Se $w_{i+j} = x_i \wedge w_{i+j} \neq y_j$ allora l'unico carattere valido per la posizione $i + j$ in W è proprio x_i. Quindi W_{i+j} è interleaving di X_i e Y_j solo se lo era anche W_{i+j-1}; in questo caso quindi:
$$s_{i,j} = s_{i-1,j}$$
- Se $w_{i+j} \neq x_I \wedge w_{i+j} = y_j$ allora l'unico carattere valido per la posizione $i + j$ in W è proprio y_j. Quindi W_{i+j} è interleaving di X_i e Y_j solo se lo era anche W_{i+j-1}; in questo caso quindi:
$$s_{i,j} = s_{i,j-1}$$
- Se $w_{i+j} = x_i \wedge w_{i+j} = y_j$, allora possiamo accodare sia x_i che y_j al simbolo in posizione w_{i+j} in W. In questo caso quindi:
$$s_{i,j} = s_{i-1,j} \vee s_{i,j-1}$$

Possiamo quindi riassumere i casi precedenti nella seguente equazione:

$$s_{i,j} = \begin{cases} False & \text{Se } w_{i+j} \neq x_i \wedge w_{i+j} \neq y_j \\ s_{i-1,j} & \text{Se } w_{i+j} = x_i \wedge w_{i+j} \neq y_j \\ s_{i,j-1} & \text{Se } w_{i+j} \neq x_i \wedge w_{i+j} = y_j \\ s_{i-1,j} \vee s_{i,j-1} & \text{Se } w_{i+j} = x_i \wedge w_{i+j} = y_j \end{cases}$$

Riassumendo il tuto abbiamo:
CASO BASE:

$$s_{i,j} = \begin{cases} True & \text{Se } i = 0 \wedge j = 0 \\ False & \text{Se } i = 0 \wedge j > 0 \wedge w_j \neq y_j \\ False & \text{Se } i > 0 \wedge j = 0 \wedge w_i \neq x_i \\ False & \text{Se } w_{i+j} \neq x_i \wedge w_{i+j} \neq y_j \end{cases}$$

PASSO RICORSIVO:

$$s_{i,j} = \begin{cases} s_{i,j-1} & \text{Se } i = 0 \wedge j > 0 \wedge w_j = y_j \\ s_{i-1,j} & \text{Se } i > 0 \wedge j = 0 \wedge w_i = x_i \\ s_{i-1,j} & \text{Se } w_{i+j} = x_i \wedge w_{i+j} \neq y_j \\ s_{i,j-1} & \text{Se } w_{i+j} \neq x_i \wedge w_{i+j} = y_j \\ s_{i-1,j} \vee s_{i,j-1} & \text{Se } w_{i+j} = x_i \wedge w_{i+j} = y_j \end{cases}$$

Una volta calcolati valori di tutte le variabili associate ai sottoproblemi, la **solzuione** al problema è $s_{m,n}$.

Algorithm 27: Determina ricosrivamente se W è interleaving di X ed Y

```
Procedure INTERLEAVING-RIC(i,j):
    if i = 0 ∧ j = 0 then
        return True
    end
    if w_{i+j} ≠ x_i ∧ w_{i+j} ≠ y_j then
        return False
    end
    if i = 0 ∧ j > 0 then
        if w_j = y_j then
            return INTERLEAVING-RIC(i, j − 1)
        else
            return False
        end
    end
    if i > 0 ∧ j = 0 then
        if w_i = x_i then
            return INTERLEAVING-RIC(i − 1, j)
        else
            return False
        end
    end
    if w_{i+j} = x_i ∧ w_{i+j} ≠ y_j then
        return INTERLEAVING-RIC(i − 1, j)
    end
    if w_{i+j} ≠ x_i ∧ w_{i+j} = y_j then
        return INTERLEAVING-RIC(i, j − 1)
    end
    if w_{i+j} = x_i ∧ w_{i+j} = y_j then
        return INTERLEAVING-RIC(i − 1, j) ∨
          INTERLEAVING-RIC(i, j − 1)
    end
```

L'algoritmo sopra **calcola ricorsivamente** la soluzione al problema; tuttavia esso **ha complessità esponenziale**. Scriviamo quindi un algoritmo bottom-up che calcoli iterativamente la soluzione al problema. Questo algoritmo permetterà di risolvere il problema in tempo:

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

occupando uno spazio di $\Theta(m \cdot n)$ (ossia una matrice che contiene i vari $s_{i,j}$).

		0	1	2	3	4	5	j
		ε	M	A	M	M	A	y_j
0	ε	T	F	F	F	F	F	
1	C	T	F	F	F	F	F	
2	I	T	T	T	F	F	F	
3	A	F	T	T	T	T	T	
4	O	F	T	F	F	F	T	
i	x_i							

1	2	3	4	5	6	7	8	9	i+j
C	I	M	A	A	M	M	A	O	w_{i+j}

Algorithm 28: Determina iterativamente se W è interleaving di X e Y

Procedure `INTERLEAVING-IT`(X, Y, W)**:**

```
s[0,0] := True
for i ← 1 to m do
    if w_i = x_i then
        s[i,0] := s[i−1,0]
    else
        s[i,0] := False
    end
end
for j ← 1 to n do
    if w_j = y_j then
        s[0,j] := s[0,j−1]
    else
        s[0,j] := False
    end
end
for i ← 1 to m do
    for j ← 1 to n do
        if w_{i+j} ≠ x_i ∧ w_{i+j} ≠ y_j then
            s[i,j] := False
        end
        if w_{i+j} = x_i ∧ w_{i+j} ≠ y_j then
            s[i,j] := s[i−1,j]
        end
        if w_{i+j} ≠ x_i ∧ w_{i+j} = y_j then
            s[i,j] := s[i,j−1]
        end
        if w_{i+j} = x_i ∧ w_{i+j} = y_j then
            s[i,j] := s[i−1,j] ∨ s[i,j−1]
        end
    end
end
return s[m,n]
```

2.9 Rendere palindroma una stringa

Il problema è definito come segue:
PROBLEMA: Sia $S =< s_1, \ldots, s_n >$ una stringa di lunghezza n definita su un alfabeto Σ. Determinare il numero minimo di caratteri da aggiungere ad S per renderla palindroma. Introduciamo la funzione $f : \Sigma^* \to \mathbb{N}$, così definita:

$\forall S \in \Sigma^*, f(S) =$ numero minimo di caratteri da aggiungere ad S per renderla palindroma

Il problema può essere quindi definito in termini più formali come segue:
PROBLEMA: Sia $S =< s_1, \ldots, s_n >$ una stringa di lunghezza n defiita su un alfabeto Σ. Determinare $f(S)$.
Prima di deifnire le variabili dei sottoproblemi, è necessario capire che casi si possono presentare in base alla struttura di S:

- $S = \varepsilon$, allora S è palindroma e quindi $f(S) = 0$
- $S = a, \forall a \in \Sigma^*$, allora S è palindroma e quindi $f(S) = 0$
- $|S| \geq 2$ allora possiamo vedere S come $aS'b$ con:
  - a il primo carattere di S
  - b l'ultimo carattere di S
  - S' la sottostringa di S rimanente dopo aver tolto a e b

  Se siamo in quest'ultimo caso, per rendere S palindroma dobbiamo distinguere:

  - Se $a = b$ allora S è palindroma se lo è anche S', quindi $f(S) = f(S')$
  - Se $a \neq b$, allora per rendere S palindroma abbiamo due possibilità
    1. Aggiungere a alla fine di S e rendere palindroma $S'b$, quindi $f(S) = 1 + f(S'b)$
    2. Aggiungere b alla fine di S e rendere palindroma aS', quindi $f(S) = 1 + f(aS')$

  Di conseguenza, quando $a \neq b$ per scegliere tra i due casi, prenderemo quello di valore minimo. Ovvero:

$$f(S) = \min\{f(S'b), f(aS')\} + 1$$

Si noti che nelle situazioni illustrate in precedenza, per rendere palindroma una stringa $S = aS'b$ con $|S| \geq 2$ occorre rendere palindroma la stringa ottenuta da S o togliendo sia il primo che l'ultimo simbolo (ossia rendere palindroma S' se $a = b$) o togliendo solo il primo o solo l'ultimo simbolo. In generale, qindi, il passaggio ad un sottoproblema richiede di ridurre una stringa potendo togliere il suo primo o ultimo simbolo (o entrambi). Pertanto, $\forall (i, j), 1 \leq i \leq j \leq n$, definiamo:

$$S_{i,j} =< s_i, \ldots, s_j >$$

la sottostringa della stringa S ottenuta da S considerando tutti i simboli di S dalla posizione i alla posizione j. Si noti che $\forall (i, j), i > j$ si avrà che $S_{i,j} = \varepsilon$.

Il generico sottoproblema è quindi individuato da una coppia (i, j) ed è definito come segue:
SOTTOPROBLEMA GENERICO: Data una stringa S, definita su Σ e di lunghezza n, determinare il numero di caratteri da aggiungere alla sottostringa $S_{i,j}$ per renderla palindroma.
Dato che $1 \leq i \leq n$ e $1 \leq j \leq n$ si ottengono n^2 sottoproblemi. Ad ogni sottoproblema è **associata una variabile** così definita:

$$m_{i,j} = \text{il minimo numero di caratteri da aggiungere alla sottostringa } S_{i,j} \text{ per renderla palindroma}$$

Per determinare la soluzione di un qualsiasi sottoproblema di dimensione (i, j), oltre all'input del problema, si utilizzeranno anche le soluzioni dei sottoproblemi di dimensione minore. Si noti che ognuna delle variabili associate ai sottoproblemi è una **black-box**: si può utilizzare, ma non ne si può conoscere il contenuto. Scriviamo le **equazioni di ricorrenza**:
CASO BASE: (i, j) con $i \geq j$
In questo caso abbiamo da considerare due possibilità:

- Se $i = j$ allora $S_{i,j}$ è composa da un solo carattere e quindi è palindroma, quindi

$$m_{i,j} = 0$$

- Se $i > j$, allora $S_{i,j}$ è vuota, quindi è palindroma, quindi

$$m_{i,j} = 0$$

Di conseguenza, possiamo riassumere il caso base come:

$$m_{i,j} = 0 \text{ Se } i \geq j$$

PASSO RICORSIVO: (i, j) con $i < j$
Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione (i, j) con $i < j$, ossia quando la sottostringa $S_{i,j}$ è composta da almeno due caratteri. In questo caso, si devono considerare due possibilità:

- Se $s_i = s_j$, allora $S_{i,j}$ è palindroma se lo è anche $S_{i+1,j-1}$, quindi:

$$m_{i,j} = m_{i+1,j-1}$$

- Se $s_i \neq s_j$, allora $S_{i,j}$ è palindroma se lo è anche $S_{i+1,j}$ o $S_{i,j-1}$, quindi:

$$m_{i,j} = \min\{m_{i+1,j}, m_{i,j-1}\} + 1$$

Di conseguenza, possiamo riassumere le equazioni di ricorrenza come segue:

$$m_{i,j} = \begin{cases} 0 & \text{Se } i \geq j \\ m_{i+1,j-1} & \text{Se } i < j \wedge s_i = s_j \\ 1 + \min\{m_{i+1,j}, m_{i,j-1}\} & \text{Se } i < j \wedge s_i \neq s_j \end{cases}$$

Una volta calcolati i valori $m_{0,0}, m_{0,1}, \ldots, m_{n,n}$ la soluzione del problema è in

$$m_{1,n}$$

infatti, sono interessato a trovare il numero minimo di caratteri da aggiungere per rendere l'intera S palindroma. Utilizzando le equazioni di ricorrenza, potrei scrivere un algoritmo ricorsivo per determinare la soluzione del problema, tuttavia esso avrebbe **complessità esponenziale**. Quindi, scrivo un algoritmo iterativo bottom-up che calcola la soluzione al problema:

Algorithm 29: Algoritmo iterativo che calcola il minimo numero di caratteri da aggiungere ad S per renderla palindroma

```
Procedure PAL-IT(S):
    n := LENGTH(S)
    for j ← 1 to n do
        for i ← j to n do
            m[i, j] := 0
        end
    end
    for i ← n − 1 down to 1 do
        for i ← i + 1 to n do
            if s_i = s_j then
                m[i, j] := m[i + 1, j − 1]
            else
                m[i, j] := 1 + min{m[i + 1, j], m[i, j − 1]}
            end
        end
    end
    return m[1, n]
```

L'algoritmo ha complessità

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

e occupa uno spazio pari a $\Theta(n^2)$ (ossia una matrice per contenere i vari $m_{i,j}$).

S:	C	A	S	A

	1	2	3	4	j
1	0	1	2	1	
2	0	0	1	0	
3	0	0	0	1	
4	0	0	0	0	
i					

2.10 Knapsack

Sia $C > 0$ la capacità di uno zaino e sia $X = \{1, \ldots, n\}$ un insieme di n oggetti, con $n \in \mathbb{N}$. Ad ogni oggetto $i \in X$ sono associate le seguenti informazioni (v_i, w_i) dove:

- v_i indica il **valore** dell'oggetto
- w_i indica il **peso** dell'oggetto

Introduciamo ora la funzione:

$$V : \mathcal{P}(X) \to \mathbb{R}^+$$

che associa ad ogni sottoinsieme di oggetti $A \subseteq X$ il suo valore complessivo. Formalmente, $\forall A \subseteq X$:

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

Introduciamo inoltre la funzione

$$W : \mathcal{P}(X) \to \mathbb{R}^+$$

che associa ad ogni sottonsieme di oggetti $A \subseteq X$ il suo ingombro complessivo. Formalemente, $\forall A \subseteq X$:

$$W(A) = \begin{cases} \sum_{i \in A} w_i & \text{Se } A \neq \emptyset \\ 0 & \text{Se } A = \emptyset \end{cases}$$

Il problema è quindi così definito:
PROBLEMA: Dato un insieme $X = \{1, \ldots, n\}$ di oggetti, trovare un sottoinsieme di ingombro complessivo $\leq C$ e di massimo valore complessivo.
Il problema può essere ridefinito come segue:
ISTANZA: $X = \{1, \ldots, n\}$ con $(v_i, w_i), \forall i \in X$
SOLUZIONE: $S \subseteq X$ tale che:

$$W(S) \leq C \wedge V(S) = \max_{\substack{A \subseteq X \\ W(A) \leq C}} \{V(A)\}$$

Come per il problema WIS, possiamo risolvere questo problema senza definire un problema ridotto. Il problema contiene diversi sottoproblemi, ognuno dei quali non ha come input l'insieme $\{1, \ldots, n\}$ con $(v_i, w_i), \forall i \in \{1, \ldots, n\}$ ma un suo sottoinsieme:

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

$$W(S_n) \leq C \wedge V(S_n) = \max_{\substack{A \subseteq X_n \\ W(A) \leq C}} \{V(A)\}$$

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

$$W(S_{n-1}) \leq C \wedge V(S_{n-1}) = \max_{\substack{A \subseteq X_{n-1} \\ W(A) \leq C}} \{V(A)\}$$

3. ...

4. Dato $X_i = \{1, \ldots, i\}$, si vuole trovare $S_i \subseteq X_i$ tale che:

$$W(S_i) \leq C \wedge V(S_i) = \max_{\substack{A \subseteq X_i \\ W(A) \leq C}} \{V(A)\}$$

5. ...

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

$$W(S_0) \leq C \wedge V(S_0) = \max_{\substack{A \subseteq X_0 \\ W(A) \leq C}} \{V(A)\}$$

Quindi il generico sottoproblema è identificato da una variabile i, ed è così definito:
SOTTOPROBLEMA GENERICO: Dato un insieme $X_i = \{1, \ldots, i\}$ di oggetti, trovare un suo sottoinsieme di ingombro complessivo $\leq C$ e di massimo valore complessivo.
Ossia:
ISTANZA GEN. SOTTOPROB.: $X_i = \{1, \ldots, i\}$ con $(v_j, w_j), \forall j \in X_i$
SOLUZIONE GEN. SOTTOPROB.: $S_i \subseteq X_i$ tale che:

$$W(S_i) \leq C \wedge V(S_i) = \max_{\substack{A \subseteq X_i \\ W(A) \leq C}} \{V(A)\}$$

Dato che $0 \leq i \leq n$ si ottengono $(n+1)$ sottoproblemi, ad ognuno dei quali è associata una **coppia di variabili** (OPT_i, S_i) così definita:

$OPT_i = V(S_i)$, ossia il valore di un sottoninseme di X_i di ingombro complessivo $\leq C$ e di massimo valore complessivo
$S_i =$ un sottoinsieme di X_i di ingombro complessivo $\leq C$ e di massivo valore complessivo, pari a OPT_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. Si noti che ognuna delle variabili associate ad un sottoproblema è da considerarsi come una **black-box**: si può utilizzare ma non è possibile conoscerne il contenuto.
Purtroppo, però, il problema così definito **non è risolvibile**: i sottoproblemi così definiti non ci permettono di risolvere il problema. Date solamente le variabili $\{(OPT_0, S_0), \ldots, (OPT_{i-1},$ e il sottoinsieme X_i, non c'è alcun modo per **poter comprendere se l'elemento i possa essere aggiunto a qualche sottoinsieme di oggetti relativo ai sottoproblemi di dimensione minore a** i, poiché non sappiamo l'ingombro complessivo di questi sottoinsiemi e quindi non possiamo sapere se, aggiungendo l'oggetti i, si superi la capacità dello C dello zaino. Risulta pertanto necessario **ridefinire i sottoproblemi**: è necessario suddividere il problema in $(n+1) \cdot (C+1)$ sottoproblemi. Il sottoproblema generico sarà quindi individuato da una coppia di valori (i, c) con $0 \leq i \leq n$ e $0 \leq c \leq C$ e sarà definito come segue
GEN. SOTTOPROB. RIDEFINITO: Dato un insieme X_i di oggetti, trovare un suo sottoinsieme di ingombro complessivo $\leq c$ e di massimo valore complessivo.
Ovvero:
ISTANZA GEN. SOTTOPROB. RID.: $X_i = \{1, \ldots, i\}$ con $(v_j, w_j), \forall j \in X_i$ e

$c \in \{0, \ldots, C\}$
SOLUZIONE GEN. SOTTOPROB. RID.: $S_i \subseteq X_i$ tale che:

$$W(S_{i,c}) \leq c \wedge V(S_{i,c}) = \max_{\substack{A \subseteq X_i \\ W(A) \leq c}} \{V(A)\}$$

Ad ogni sottoproblema di dimensione (i, c) è associata **una coppia di variabili** così definita:

$OPT_{i,c} = V(S_{i,c})$, ossia il valore di un sottoninseme di X_i di ingombro complessivo $\leq c$ e di massimo valore complessivo

$S_{i,c}$ = un sottoinsieme di X_i di ingombro complessivo $\leq c$ e di massivo valore complessivo, pari a $OPT_{i,c}$

Avendo ridefinito il problema, possiamo andare a scrivere le **equazioni di ricorrenza**:
CASO BASE: (i, c) con $i = 0 \vee c = 0$
Il caso base si ha per un qualunque sottoproblema di dimensione (i, c) con $i = 0 \vee c = 0$. In questo caso, è facile ottenere la soluzione del sottoproblema. Infatti, se $i = 0$, non abbiamo oggetti mentre se $c = 0$ si richiede che l'insieme di oggetti abbia ingombro complessivo pari a 0. In entrambi questi casi, si ha che:

$$OPT_{i,c} = 0 \; e \; S_{i,c} = 0 \text{ Se } i = 0 \vee c = 0$$

PASSO RICORSIVO: (i, c) con $i > 0 \wedge c > 0$
Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione (i, c) con $i > 0 \wedge c > 0$, ossia quando si va a considerare un insieme che contiene almeno un ogetto e si richiede un ingombro complessivo minore di un certo valore diverso da 0. I dati disponibili per calcolare la soluzione di un sottoproblema sono: l'input X_i (in particolare l'oggetto i) e tutte le soluzioni ai problemi di dimensione minore. È necessario considerare **due casi**:

- Se $w_i > c$, cioè se l'oggetto i ha peso maggiore della capacità c, allora l'oggetto **non può appartenere alla soluzione** e quindi si va a considerare la soluzione del sottoproblema di dimensione minore; cioè:

$$OPT_{i,c} = OPT_{i-1,c} \; e \; S_{i,c} = S_{i-1,c} \text{ Se } w_i > c$$

- Se $w_i \leq c$, cioè se l'oggetto i ha peso minore delal capacità c, allora l'oggetto potrebbe appartenere alla soluzione. Ci si può quindi trovare in due sottocasi:
  - Se $i \notin S_i$, allora è sufficiente considerare l'insieme di oggetti X_{i-1} e si può dunque considerare la soluzione del sottoproblema di dimensione $(i - 1, c)$, ossia:
  $$OPT_{i,c} = OPT_{i-1,c} \; e \; S_{i,c} = S_{i-1,c}$$
  - Se $i \in S_i$ allora è necessario determinare la soluzione del sottoproblema di dimensione $(i - 1, c - w_i)$ e aggiungere a questo l'oggetto i. Quindi:
  $$OPT_{i,c} = OPT_{i-1,c-w_i} + v_i \; e \; S_{i,c} = S_{i-1,c-w_i} \cup \{i\}$$

Siccome non possiamo sapere a priori se $i \in S_i$, allora scegliamo l'opzione che ci restituisce il valore massimo. Dunque:

$$OPT_{i,c} = \max\{OPT_{i-1,c}, OPT_{i-1,c-w_i} + v_i\}$$

e

$$S_{i,c} = \begin{cases} S_{i-1,c} & \text{Se } OPT_{i-1,c} \geq OPT_{i-1,c-w_i} + v_i \\ S_{i-1,c-w_i} \cup \{i\} & \text{Altrimenti} \end{cases}$$

Dunque, il passo ricorsivo può essere riscritto come:

$$OPT_{i,c} = \begin{cases} OPT_{i-1,c} & \text{Se } w_i > c \\ \max\{OPT_{i-1,c}, OPT_{i-1,c-w_i} + v_i\} & \text{Altrimenti} \end{cases}$$

$$S_{i,c} = \begin{cases} S_{i-1,c} & \text{Se } w_i > c \\ \begin{cases} S_{i-1,c} & \text{Se } OPT_{i-1,c} \geq OPT_{i-1,c-w_i} + v_i \\ S_{i-1,c-w_i} \cup \{i\} & \text{Altrimenti} \end{cases} & \text{Altrimenti} \end{cases}$$

Enunciamo quindi la **proprietà della sottostruttura ottima**:

Teorema 2.10.1 (PSO Knapsack) *Sia $i > 0$ e $c > 0$. Siano $S_{0,0}, \ldots, S_{i-1,c-1}$ le soluzione dei sottoproblemi di dimensione $(0,0), \ldots, (i-1, c-1)$.di Knapsack. Allora vale che:*

$$S_{i,c} = \begin{cases} S_{i-1,c} & \textit{Se } i \notin S_{i,c} \\ S_{i-1,c-w_i} \cup \{i\} & \textit{Se } i \in S_i \end{cases}$$

Dimostrazione 2.10.1 *Distinguiamo i due casi:*

- *Se $i \notin S_i$ allora devo dimostrare che $S_{i,c} = S_{i-1,c}$. Per assurdo, assumiamo che $S_{i-1,c}$ non sia soluzione del problema di dimensione (i,c), allora $S_{i-1,c} \neq S_{i,c}$. Deve allora esistere $S'_{i-1,c} \neq S_{i-1,c}$ che è soluzione del problema (i,c). Quindi*

$$V(S'_{i-1,c}) = \max_{\substack{A \subseteq X_i \\ W(A) \leq c}} \{V(A)\}$$

*Quindi $V(S'_{i-1,c}) > V(S_{i-1,c})$ e $W(S'_{i-1,c}) < c$. Poiché $i \notin S_{i,c}$, allora $i \notin S'_{i-1,c}$, quindi $S'_{i-1,c} \subseteq X_{i-1}$, ma ciò non è possibile, poiché allora $S_{i-1,c}$ non sarebbe più la soluzione del sottoproblema di dimensione $(i-1,c)$; quindi **assurdo**.*

- *Se $i \in S_{i,c}$, allora devo dimostrare che $S_{i,c} = S_{i-1,c-w_i} \cup \{i\}$. Per assurdo, supponiamo che $S_{i-1,c-w_i} \cup \{i\}$ non è soluzione del problema di dimensioni (i,c). Allora deve esistere $S'_{i-1,c-w_i} \neq S_{i-1,c-w_i} \cup \{i\}$ tale che $S'_{i-1,c-w_i}$ è soluzione del sottoproblema (i,c). Allora $V(S'_{i-1,c-w_i}) > V(S_{i-1,c-w_i} \cup \{i\})$.*
  Siccome $i \in S_{i,c}$ allora $i \in S'_{i-1,c-w_i}$, quindi $S'_{i-1,c-w_i} = S''_{i-1,c-w_i} \cup \{i\}$. Allora:

$$V(S'_{i-1,c-w_i}) > V(S_{i-1,c-w_i} \cup \{i\})$$
$$V(S''_{i-1,c-w_i} \cup \{i\}) > V(S_{i-1,c-w_i} \cup \{i\})$$
$$V(S''_{i-1,c-w_i}) + v_i > V(S_{i-1,c-w_i}) + v_i$$
$$V(S''_{i-1,c-w_i}) > V(S_{i-1,c-w_i})$$

*Tuttavia ciò non può essere, poiché allora $S_{i-1,c-w_i}$ non sarebbe più la soluzione del sottoproblema $i(i-1, c-w_i)$, quindi **assurdo**.*

Una volta calcolati i valori $OPT_{0,0}, \ldots, OPT_{n,C}$, è possibile determinare **la soluzione** al problema andando a considerare:

$$(OPT_{n,C}, S_{n,C})$$

Seguendo le equazioni di ricorrenza, potremmo scrivere un algoritmo ricorsivo che calcola la soluzione al problema. Tuttavia, esso avrebbe costo computazionale **esponenziale**. Quindi, scriviamo un algoritmo iterativo bottom-up che calcola la soluzione del problema:

Algorithm 30: Algoritmo iterativo che calcola la soluzione a Knapsack

```
Procedure KNAPSACK-IT(n, C):
    for i ← 0 to n do
        OPT[i, 0] := 0
        S[i, 0] := ∅
    end
    for c ← 0 to C do
        OPT[0, c] := 0
        S[0, c] := ∅
    end
    for i ← 1 to n do
        for c ← 1 to C do
            if w_i > c_i then
                OPT[i, c] := OPT[i − 1, c]
                S[i, c] := S[i − 1, c]
            else
                V_1 := OPT[i − 1, c]
                V_2 := OPT[i − 1, c − w_i] + v_i
                OPT[i, c] := MAX(V_1, V_2) if V_1 > V_2 then
                    S[i, c] := S[i − 1, c]
                else
                    S[i, c] := S[i − 1, c − w_i] ∪ {i}
                end
            end
        end
    end
    return (OPT[n, C], S[n, C])
```

L'algoritmo ha complessità computazionale pari a:

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

occupando però $\Theta(n \cdot C + n^2 \cdot C)$ spazio in memoria (ossia una matrice che contiene i vari $OPT_{i,c}$ e una matrice che contiene i vari $S_{i,c}$, ognuno dei quali può contenere sino a n elementi).

Osservazione 2.10.1 *Questo algoritmo* ***non è polinomiale*** *nell'input ma* ***pseudopolinomiale***

Presentiamo l'algoritmo di ricostruzione:

Algorithm 31: Algoritmo di ricostruzione della soluzione di Knapsack

```
Procedure PRINT-KNAPSACK(i, c):
    if i > 0 ∧ c > 0 then
        if w_i > c then
            PRINT-KNAPSACK(i − 1, c)
        else
            if OPT[i − 1, c − w_i] + v_i ≥ OPT[i − 1, c] then
                PRINT-KNAPSACK(i − 1, c − w_i)
                print(i)
            else
                PRINT-KNAPSACK(i − 1, c)
            end
        end
    end
```

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Tabella 1: Problema dello zaino - Input.

i \ c	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	28	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

Tabella 2: Problema dello zaino: matrice delle soluzioni $\texttt{OPT}_{i,c}$ ottenuta considerando l'input presentato in Tabella 1 e una capacità dello zaino pari a $C = 11$.

2.10.1 Knapsack con oggetti colorati

Presentiamo, in maniera meno formale, una variante del problema Knapsack.
PROBLEMA: Dato un insieme $X = \{1, \ldots, n\}$ di oggetti, trovare un sottoinsieme di ingombro complessivo $\leq C$ e di massimo valore complessivo e che contiene al massimo R oggetti di colore rosso.
ISTANZA: Si ha che:

- $X_n = \{1, \ldots, n\}$ con $n \in \mathbb{N}$
- $\forall i \in X_n$ si ha la seguente tripla di valori:
  - $v_i \in \mathbb{N}$, valore di i
  - $w_i \in \mathbb{N}$, peso di i

– $col(i) \in \{Red, Blue\}$, colore di i

- $C \in \mathbb{N}$, capacità dello zaiono
- $R \in \mathbb{R}$, numero massimo di oggetti rossi concessi

SOLUZIONE: $S \subseteq X_n$ tale che:

$$W(S) \leq C \wedge V(S) = \max_{\substack{A \subseteq X_n \\ W(A) \leq C \\ |\{j \in A | col(i) = Red\}| \leq R}} \{V(A)\}$$

Il generico sottoproblema è quindi definito dalla tripla (i, c, r) ed è così definito:
SOTTOPROBLEMA GENERICO: Dato un insieme $X_i = \{1, \ldots, i\}$ di oggetti, trovare un sottoinsieme di ingombro complessivo $\leq c$ e di massimo valore complessivo e che contiene al massimo r oggetti di colore rosso.
Ad ogni sottoproblema è associata una **coppia di variabili** $(OPT_{i,c,r}, S_{i,c,r})$ così definite:

$OPT_{i,c,r} = V(S_{i,c,r})$, ossia il valore di un sottoninseme di X_i di ingombro complessivo $\leq c$ di massimo valore complessivo e con al massimo r simboli rossi
$S_{i,c,r}$ = un sottoinsieme di X_i di ingombro complessivo $\leq c$ di massivo valore complessivo, pari a $OPT_{i,c}$, e con al massimo r simboli rossi

Si hanno quindi $(n+1) \cdot (C+1) \cdot (R+1)$ sottoproblemi diversi. Scriviamo le **equazioni di ricorrenza**:
CASO BASE: (i, c, r) con $i = 0 \vee c = 0$ e $r \in \{0, \ldots, R\}$
Si hanno i seguenti casi:

- Se $i = 0$ allora $\forall c \in \{0, \ldots, C\}$ e $\forall r \in \{0, \ldots, R\}$ si ha che:

$$OPT_{i,c,r} = 0 \; e \; S_{i,c,r} = \emptyset$$

- Se $c = 0$ allora $\forall i \in \{0, \ldots, i\}$ e $\forall r \in \{0, \ldots, R\}$ si ha che:

$$OPT_{i,c,r} = 0 \; e \; S_{i,c,r} = \emptyset$$

PASSO RICORSIVO: (i, c, r) con $i > 0 \wedge c > 0$ e $r \in \{0, \ldots, R\}$
Dobbiamo distinguere i seguenti casi:

1. Se $w_i > c$ allora certamente $i \notin S_{i,c,r}$, allora:

$$S_{i,c,r} = S_{i-1,c,r}$$

e

$$OPT_{i,c,r} = OPT_{i-1,c,r}$$

2. Se $w_i \leq c$ allora dobbiamo distinguere altri tre sottocasi:

(a) Se $col(i) = Red \wedge r > 0$ allora:

$$S_{i,c,r} = \begin{cases} S_{i-1,c-w_i,r-1} \cup \{i\} & \text{Se } OPT_{i-1,c-w_i,r-1} + v_i \geq OPT_{i-1,c,r} \\ S_{i-1,c,r} & \text{Altrimenti} \end{cases}$$

e:

$$OPT_{i,c,r} = \max\{OPT_{i-1,c-w_i,r-1} + v_i, OPT_{i-1,c,r}\}$$

(b) Se $col(i) = Red \wedge r = 0$ allora certamente $i \notin S_{i,c,r}$, quindi

$$S_{i,c,r} = S_{i-1,c,r}$$

e

$$OPT_{i,c,r} = OPT_{i-1,c,r}$$

(c) Se $col(i) \neq Red$, allora $\forall r \in \{0, \ldots, R\}$ si ha che:

$$S_{i,c,r} = \begin{cases} S_{i-1,c-w_i,r} \cup \{i\} & \text{Se } OPT_{i-1,c-w_i,r} + v_i \geq OPT_{i-1,c,r} \\ S_{i-1,c,r} & \text{Altrimenti} \end{cases}$$

e

$$OPT_{i,c,r} = \max\{OPT_{i-1,c-w_i,r} + v_i, OPT_{i-1,c,r}\}$$

Algorithm 32: Algoritmo di ricostruzione della soluzione di Knapsack con R simboli rossi

```
Procedure PRINT-KNAPSACK-RED(i, c, r):
    if i > 0 ∧ c > 0 then
        if w_i > c then
            PRINT-KNAPSACK(i − 1, c, r)
        else
            if col(x_i) ≠ Red then
                if OPT[i − 1, c − w_i, r] + v_i ≥ OPT[i − 1, c, r] then
                    PRINT-KNAPSACK(i − 1, c − w_i, r)
                    print(i)
                else
                    PRINT-KNAPSACK(i − 1, c, r)
                end
            else
                if r > 0 then
                    if OPT[i − 1, c − w_i, r − 1] + v_i ≥ OPT[i − 1, c, r] then
                        PRINT-KNAPSACK(i − 1, c − w_i, r − 1)
                        print(i)
                    else
                        PRINT-KNAPSACK(i − 1, c, r)
                    end
                else
                    PRINT-KNAPSACK(i − 1, c, r)
                end
            end
        end
    end
```

Una volta calcoalte tutte le variabili associate ai sottoproblemi, è possibile trovare **la soluzione** considerando:

$$(OPT_{n,C,R}, S_{n,C,R})$$

2.10.2 Subset-Sum

Presentiamo in maneira informale il seguente problema:
ISTANZA: $X = \{x_1, \ldots, x_n\}$ con $x_i \in \mathbb{N}^+$ e $K \in \mathbb{N}^+$
PROBLEMA: Stabilire se esiste un sottoinsieme $Y \subseteq X$ i cui elementi danno somma K, ossia:

$$\sum_{y \in Y} y = K$$

SOLUZIONE: $True \Leftrightarrow \exists Y \subseteq X \; t.c. \; \sum_{y \in Y} y = K$
Il generico sottoproblema è individuato da una **coppia** (i, k) con $0 \leq i \leq n$ e $0 \leq k \leq K$ ed è così definito:
SOTTOPROBLEMA GENERICO:Stabilire se esiste un sottoinsieme $Y_{i,k} \subseteq X_i = \{x_1, \ldots, x_i\}$ i cui elementi danno somma k, ossia:

$$\sum_{y \in Y_{i,k}} y = k$$

Ad ogni sottoproblema di dimensioni (i, k) è assocaita **una variabile** così definita:

$$e_{i,k} = True \Leftrightarrow \exists Y_{i,k} \subseteq X_i \; t.c. \; \sum_{y \in Y_{i,k}} y = k$$

Scriviamo le **equazioni di ricorrenza**:
CASO BASE: (i, k) con $i = 0 \vee k = 0$
Dobbiamo distinguere 3 casi:

- Se $i = 0 \wedge k = 0$ allora $e_{i,k} = True$ poiché $Y_{i,k} = \emptyset$
- Se $i = 0 \wedge k > 0$ allora $e_{i,k} = False$ poiché $\nexists Y_{i,k}$
- Se $i > 0 \wedge k = 0$ allora $e_{i,k} = True$ poiché $Y_{i,k} = \emptyset$

PASSO RICORSIVO: (i, k) con $i > 0 \wedge k > 0$
Dobbiamo distinguere due casi:

1. Se $x_i > k$, allora certamente $Y_{i,k}$ non contiene x_i, quindi $e_{i,k} = e_{i-1,k}$ e $Y_{i,k} = Y_{i-1,k}$
2. Se $x_i \leq k$, potrebbe verificarsi uno dei seguente sottocasi:
   - Se $x_i \in Y_{i,k}$ allora $e_{i,k} = e_{i-1,k-x_i}$ (se $Y_{i,k}$ esiste, allora $Y_{i,k} = Y_{i-1,k-x_i} \cup \{x_i\}$)
   - Se $x_i \notin Y_{i,k}$ allora $e_{i,k} = e_{i-1,k}$ (Se $Y_{i,k}$ esiste allora $Y_{i,k} = Y_{i-1,k}$)

   Non sapendo a priori quale tra i due casi si verifica, avremo che:

$$e_{i,k} = e_{i-1,k-x_i} \vee e_{i-1,k}$$

La **soluzione al problema** può essere trovata considerando $e_{n,K}$. Scriviamo l'algoritmo bottom-up:

Algorithm 33: Algoritmo iterativo che trova la soluzione al problema Subset-Sum

```
Procedure SUBSETSUM-IT(X, K):
    n := LENGTH(X)
    for k ← 0 to K do
        if k = 0 then
            e[0, k] := True
        end
        e[0, k] := False
    end
    for i ← 1 to n do
        e[i, 0] := True
    end
    for i ← 1 to n do
        for k ← 1 to K do
            if x_i > k then
                e[i, k] := e[i − 1, k]
            else
                e[i, k] := e[i − 1, k − x_i] ∨ e[i − 1, k]
            end
        end
    end
    return e[n, K]
```

Algorithm 34: Algoritmo di ricostruzione della soluzione di Subset-Sum

```
Procedure PRINT-SUBSETSUM(i, k):
    if e[i, k] = False then
        print("non esiste alcun sottoinsieme Y_{i,k}")
    else
        if i > 0 ∧ k > 0 then
            if x_i > k then
                PRINT-SUBSETSUM(i − 1, k)
            else
                if e[i − 1, k − x_i] = True then
                    PRINT-SUBSETSUM(i − 1, k − x_i)
                    print(x_i)
                else
                    PRINT-SUBSETSUM(i − 1, k)
                end
            end
        end
    end
```

2.10.3 LCS con ingombri

Presentiamo, in maniera informale, il seguente problema: sia $w : \Sigma \to \mathbb{N}^+$ una funzione che da un peso ad ogni simbolo dell'alfabeto Σ e siano X e Y due sequenze, di lunghezza rispettivamente m ed n, costruire su tale alfabeto. Sia infine C l'ingombro complessivo permesso. Il problema è quindi così definito:
PROBLEMA: Calcolare la lunghezza di una più lunga sottosequenza comune di X e Y la quale ha ingombro complessivo $\leq C$