

## Appunti Analisi e Progetto di Algoritmi

### Domande e Risposte

#### Capitolo 1 - Nozioni introduttive

##### 1) Cosa si intende per alfabeto?

Un **alfabeto** è un insieme finito e non vuoto di simboli che vengono identificati mediante la lettera maiuscola  $\Sigma$  (sigma).

Esempio:

→  $\Sigma = \{0, 1\}$ , l'alfabeto binario;

→  $\Sigma = \{a, b, \dots, z\}$ , l'alfabeto di tutte le lettere dell'alfabeto;

→  $\Sigma = \{0, 1, \dots, 9\}$ , l'alfabeto delle cifre da 0 a 9.

##### 2) Cosa si intende per stringa?

Una **stringa** è una sequenza finita di simboli scelti da un alfabeto.

Esempio: Sia l'alfabeto  $\Sigma = \{0, 1\}$  binario, le possibili stringhe sono:

→  $w_1 = 10110$

→  $w_2 = 1101$

→  $w_3 = 100101$

→  $w_1 = \langle 1, 0, 1, 1, 0 \rangle$

→  $w_2 = \langle 1, 1, 0, 1 \rangle$

→  $w_3 = \langle 1, 0, 0, 1, 0, 1 \rangle$

Uso le ultime lettere  $x, y, z$  dell'alfabeto  $\Sigma = \{a, b, \dots, x, y, z\}$  per indicare una generica sequenza (o stringa).

##### Esempio 1:

$X = \langle x_1, \dots, x_n \rangle$

$x_1$  = primo simbolo di  $X$

$x_n$  = ultimo simbolo di  $X$

##### Esempio 2:

$X = \langle G, A, T, T, O \rangle$

$G = x_1$

$A = x_2$

$T = x_3$

$T = x_4$

$O = x_5$

##### Esempio 3:

Con  $n = 6$ , ho che

$Y = \langle y_1, y_2, y_3, y_4, y_5, y_6 \rangle$

#### Capitolo 2 - LCS (Longest Common Subsequence)

##### 3) Cosa si intende per sequenza?

Si definisce **sequenza** una successione di elementi topologicamente ordinati presi da un insieme  $\Sigma$  (esempio:  $X = \langle 2, 4, 10, 5, 9, 11 \rangle$ )

#### 4) Cosa si intende per prefisso di lunghezza k?

Si definisce **prefisso di lunghezza k** di una sequenza i primi k elementi della sequenza.

##### Esempi

$$X_3 = \langle 2, 4, 10 \rangle$$

$$X_6 = \langle 2, 4, 10, 5, 9, 11 \rangle = X$$

$$X_0 = \langle \rangle$$

#### 5) Cosa si intende per sottosequenza comune di due sequenze X e Y?

Si definisce **sottosequenza comune di due sequenze X e Y**, la sottosequenza sia di X che di Y.

##### Esempio:

$$X = \langle 1, 13, 5, 3, 1, 12, 8, 11, 6, 10, 10 \rangle$$

$$Y = \langle 1, 5, 5, 2, 3, 1, 12, 8, 8, 10 \rangle$$

$\langle 5, 3, 1, 8, 10 \rangle$  è sottosequenza comune di X e Y

$\langle 1, 5, 3, 1, 12, 8, 10 \rangle$  è sottosequenza comune di X e Y

#### 6) LCS di due sequenze: teorema (sottostruttura ottima)

Avendo il seguente problema: date due sequenze X di lunghezza m e Y di lunghezza n, trovare la più lunga sottosequenza comune Z di X e Y.

Gli elementi di un problema di ottimizzazione sono:

→ **soluzioni ammissibili**: tutte le sottosequenze comuni

→ **funzione obiettivo**: lunghezza

→ **output**: soluzione ammissibile con la massima lunghezza (LCS)

Date due sequenze:

$$X = \langle x_1, x_2, \dots, x_i, \dots, x_{m-1}, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_j, \dots, y_{n-1}, y_n \rangle$$

$$\text{LCS}(X_m, Y_n) \rightarrow Z = \langle z_1, z_2, \dots, z_{k-1}, z_k \rangle$$

##### Sottostruttura ottima

a - se  $x_m = y_n$ , allora  $z_k = x_m$  &  $\langle z_1, z_2, \dots, z_{k-1} \rangle = \text{LCS}(X_{m-1}, Y_{n-1})$

b - se  $x_m \neq y_n$ , allora se  $z_k \neq x_m$  allora  $Z = \text{LCS}(X_{m-1}, Y_n)$ , altrimenti se  $z_k \neq y_n$  allora  $Z = \text{LCS}(X_m, Y_{n-1})$

##### Equazione di ricorrenza

$$X_i = \langle x_1, x_2, \dots, x_i \rangle$$

$$Y_j = \langle y_1, y_2, \dots, y_j \rangle$$

$$x_i = y_j \Rightarrow \text{LCS}(X_i, Y_j) = \text{LCS}(X_{i-1}, Y_{j-1}) + \langle x_i \rangle$$

$$x_i \neq y_j \Rightarrow \text{LCS}(X_i, Y_j) = \max\{\text{LCS}(X_{i-1}, Y_j), \text{LCS}(X_i, Y_{j-1})\}$$

## 7) Equazione di ricorrenza LCS

$$X = \langle x_1, x_2, \dots, x_i, \dots, x_{m-1}, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_j, \dots, y_{n-1}, y_n \rangle$$

$$\text{LCS}(X_m, Y_n) \rightarrow Z = \langle z_1, z_2, \dots, z_{k-1}, z_k \rangle$$

$$(i = 0 \vee j = 0) \rightarrow \text{LCS}(X_i, Y_j) = \langle \rangle$$

$$(i > 0 \wedge j > 0) \rightarrow \begin{aligned} &1. x_i = y_j \rightarrow \text{LCS}(X_i, Y_j) = \text{LCS}(X_{i-1}, Y_{j-1}) + \langle x_i \rangle \\ &2. \text{LCS}(X_i, Y_j) = \max\{\text{LCS}(X_{i-1}, Y_j), \text{LCS}(X_i, Y_{j-1})\} \end{aligned}$$

## Scrittura di LCS in Python

```
def lcs(x, y):
    length_x = len(x)
    length_y = len(y)

    if not x or not y:
        return ""
    else:
        if x[length_x - 1] == y[length_y - 1]:
            return lcs(x[0:length_x-1], y[0:length_y-1]) + x[length_x - 1]
        else:
            return max_string(lcs(x[0:length_x-1], y), lcs(x, y[0:length_y-1]))

def max_string(x, y):
    if len(x) >= len(y):
        return x
    else:
        return y

x = input("Enter first string: ")
y = input("Enter second string: ")

print(lcs(x, y))
```

## Output

```
PS C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\3_Anno\Attivita_Didattica\PrimoSemestre\Analisi_Progetto_Algoritmi\Teoria\Esempi> python lcs.py
Enter first string: Cascata
Enter second string: Casetta
Casta
PS C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\3_Anno\Attivita_Didattica\PrimoSemestre\Analisi_Progetto_Algoritmi\Teoria\Esempi> python lcs.py
Enter first string: saloon
Enter second string: lontra
lon
PS C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\3_Anno\Attivita_Didattica\PrimoSemestre\Analisi_Progetto_Algoritmi\Teoria\Esempi> python lcs.py
Enter first string: maiale
Enter second string: magica
maia
PS C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\3_Anno\Attivita_Didattica\PrimoSemestre\Analisi_Progetto_Algoritmi\Teoria\Esempi> python lcs.py
Enter first string: maiale
Enter second string: magic
mai
```

## Ottimizzazione

$$c_{ij} = |\text{LCS}(X_i, Y_j)|$$

if  $i = 0 \vee j = 0$  then //Caso base  
 $c_{ij} = 0$

if  $i > 0 \wedge j > 0$  then //Passo ricorsivo  
 if  $x_i = y_j$  then  
 $c_{ij} = c_{i-1,j-1} + 1$  //  $(m+1) \cdot (n+1)$  coefficienti  
 else  
 $c_{ij} = \max\{c_{i-1,j}, c_{i,j-1}\}$

### 8) Come è possibile calcolare il valore ottimo di LCS?

Il **valore ottimo** può essere calcolato in due modi:

→ algoritmo ricorsivo (top-down)

→ algoritmo di **PD** (bottom-up)

**Algoritmo ricorsivo:** procedura

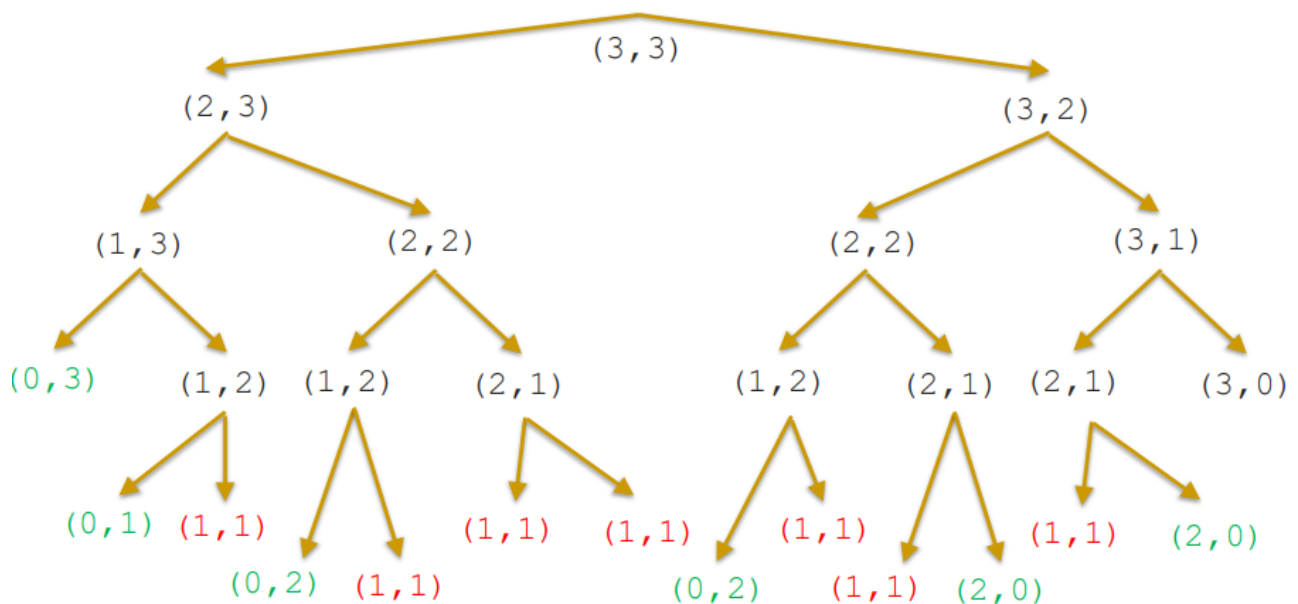
```

int ottimo_ricorsivo(i, j)
    if i = 0 ∨ j = 0 then
        return 0
    else
        if xi = yj then
            ci-1,j-1 ← ottimo_ricorsivo(i-1, j-1)
            return ci-1,j-1 + 1
        else
            ci-1,j ← ottimo_ricorsivo(i-1, j)
            ci,j-1 ← ottimo_ricorsivo(i, j-1)
            return max{ci-1,j, ci,j-1}
  
```

Tempi di esecuzione

Caso migliore:  $T(n) = \Omega(n)$

Caso peggiore: Albero della ricorsione nel caso peggiore per  $|X| = |Y| = 3$



$T(n) = O(2^{\min(m,n)})$

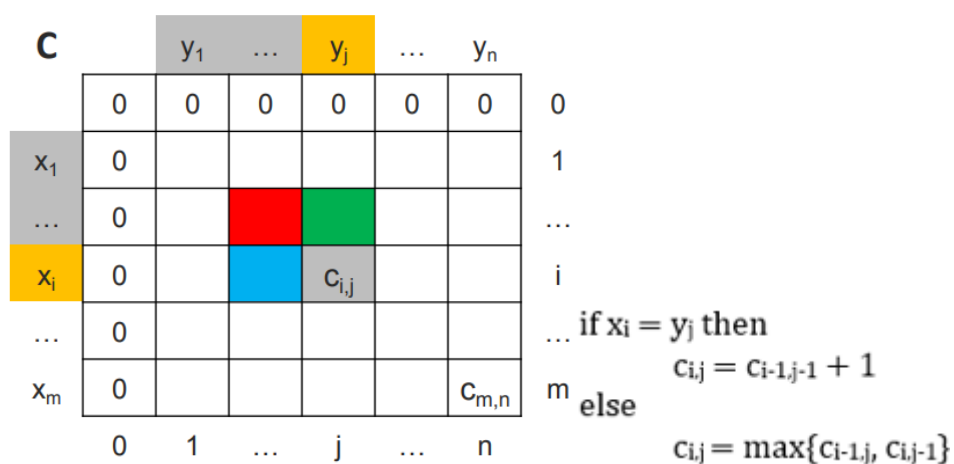
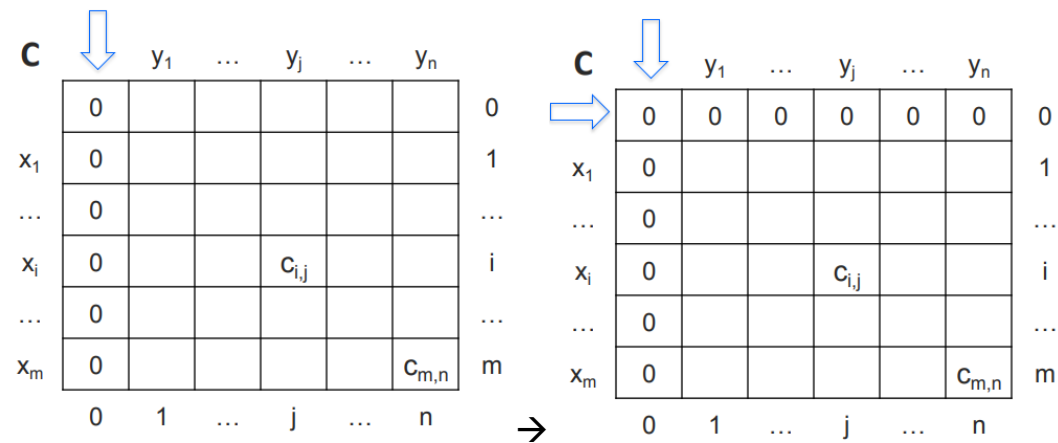
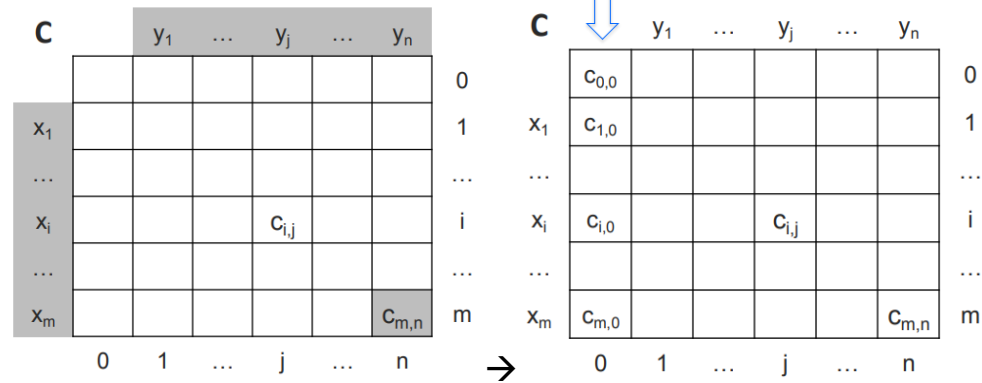
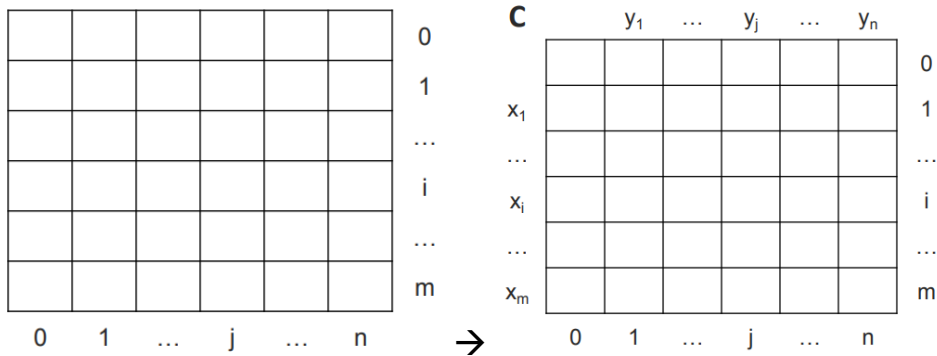
**Algoritmo iterativo di PD (iterativo)**

I passi sono:

→ si costruisce una matrice  $C$  di  $m+1$  righe e  $n+1$  colonne;

→ si riempie  $C$  in modo tale che  $C[i, j] = c_{i,j}$

→ valore ottimo =  $C[m, n]$



## Pseudocodice

```
int ottimo_PD(X, Y)
  for i from 0 to m do
    C[i, 0] ← 0
  for j from 0 to n do
    C[0, j] ← 0
  for i from 1 to m do
    for j from 1 to n do
      if  $x_i = y_j$  then
        C[i, j] = C[i-1, j-1] + 1
      else
        C[i, j] = max(C[i-1, j], C[i, j-1])
  return C[m, n]
```

## 9) Stampa LCS ricorsiva

**Problema:** data la matrice  $C$  e gli indici  $i$  e  $j$  stampare gli elementi della sequenza ricavata mediante LCS.

**Input:** matrice  $C$  e indici  $i$  e  $j$  di una cella di  $C$

**Output:**  $LCS(X_i, Y_j)$

## Pseudocodice

```
function stampa_LCS(C, i, j)
  if  $i > 0$  and  $j > 0$  then
    if  $x_i = y_j$  then
      stampa_LCS(C, i-1, j-1)
      print  $x_i$ 
    else
      if  $C[i, j] = C[i-1, j]$  then
        stampa_LCS(C, i-1, j)
      else
        stampa_LCS(C, i, j-1)
```

## 10) Qual è lo scopo principale della programmazione dinamica?

La **programmazione dinamica** è una tecnica di ottimizzazione che viene utilizzata per risolvere problemi complessi dividendo il problema in sottoproblemi più semplici e risolvendo questi ultimi una sola volta, memorizzando i risultati in una struttura dati (solitamente array monodimensionale/multidimensionale). Utilizza la strategia di bottom up, ossia i sottoproblemi vengono risolti uno dopo l'altro a partire da quelli più piccoli fino ad arrivare al più grande.

## 11) Dimostrare la proprietà della sottostruttura ottima nel problema del calcolo di una LCS

Date le due sequenze

$$X = \langle x_1, x_2, \dots, x_i, \dots, x_{m-1}, x_m \rangle$$

$$Y = \langle y_1, y_2, \dots, y_j, \dots, y_{n-1}, y_n \rangle$$

LCS di X e Y è:

$$\text{LCS}(X_m, Y_n) \rightarrow Z = \langle z_1, z_2, \dots, z_{k-1}, z_k \rangle$$

La dimostrazione della proprietà della sottostruttura ottima si basa su un'analisi di come si costruisce la LCS considerando l'ultimo carattere delle due stringhe.

**Caso 1:**  $x_i = y_j$

Se l'ultimo carattere delle due stringhe X e Y è lo stesso, cioè  $x_i = y_j$ , allora possiamo affermare che tale carattere farà parte della LCS e il problema si riduce al calcolo della LCS delle sottostringhe che escludono l'ultimo carattere, ovvero  $X[1, \dots, i-1]$  e  $Y[1, \dots, j-1]$ .

$$\text{LCS}(X[1, \dots, i], Y[1, \dots, j]) = \text{LCS}(X[1, \dots, i-1], Y[1, \dots, j-1]) + 1$$

Ciò rappresenta la soluzione ottima per  $X[1, \dots, i]$ ,  $Y[1, \dots, j]$  a partire dalla soluzione ottima per i sottoproblemi  $X[1, \dots, i-1]$ ,  $Y[1, \dots, j-1]$ , dimostrando che la sottostruttura ottima è preservata.

**Caso 2:**  $x_i \neq y_j$

Se invece l'ultimo carattere delle due stringhe è diverso, cioè  $x_i \neq y_j$ , allora l'ultimo carattere di X o di Y non può far parte della LCS. In tal caso, la LCS sarà la soluzione ottima di uno dei due sottoproblemi seguenti:

→ la LCS di  $X[1, \dots, i-1]$  e  $Y[1, \dots, j]$  (escludendo l'ultimo carattere di X);

→ la LCS di  $X[1, \dots, i]$  e  $Y[1, \dots, j-1]$  (escludendo l'ultimo carattere di Y).

Di conseguenza, la lunghezza della LCS sarà la lunghezza massima tra questi due casi:

$$\text{LCS}(X[1, \dots, i], Y[1, \dots, j]) = \max\{\text{LCS}(X[1, \dots, i-1], Y[1, \dots, j]), \text{LCS}(X[1, \dots, i], Y[1, \dots, j-1])\}$$

## Capitolo 3 - LIS (Longest Increasing Subsequence)

### 12) Cosa si intende per Increasing Subsequence?

Si definisce **Increasing Subsequence** la sequenza

$$Z = \langle z_1, z_2, \dots, z_{k-1}, z_k \rangle$$

tale che  $z_i < z_{i+1}$  per ogni indice i da 1 a k - 1.

**Esempi di sequenze crescenti:**

→  $\langle 2, 4, 10 \rangle$

→  $\langle 2, 4, 7, 13, 21 \rangle$

→  $\langle 2 \rangle$

### 13) Cosa si intende per Longest Increasing Subsequence (LIS)

Data

$$X = \langle x_1, x_2, \dots, x_i, \dots, x_{m-1}, x_m \rangle$$

la lunga sottosequenza di  $X$  che sia crescente è  $Z = \text{LIS}(X)$

**Esempio:**

$$X = \langle 14, 2, 4, 2, 7, 0, 13, 21, 11 \rangle$$

$$\text{LIS}(X) = \langle 2, 4, 7, 13, 21 \rangle$$

### 14) Problema: Longest Increasing Subsequence (LIS)

**P:** Data una sequenza  $X = \langle x_1, x_2, \dots, x_m \rangle$ , trovare la più lunga sottosequenza crescente  $Z = \text{LIS}(X)$ . **P** è un problema di ottimizzazione di **massimo**, dove:

→ (m): dimensione del problema

→ **Soluzioni possibili:** tutte le sottosequenze crescenti di  $X$

→ **Funzione obiettivo:** lunghezza

→  $|Z|$  è il valore ottimo del problema

→  $Z$  è una soluzione ottimale

### 15) Sottoproblemi e variabili associate

Il **sottoproblema** di dimensione (i) è definito come segue:

**P:** Data una sequenza  $X$  di  $m$  numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze crescenti di  $X$ .

Dato che  $0 \leq i \leq m$ , si ottengono  $m + 1$  sottoproblemi e ad ogni sottoproblema di PBR è associata una **variabile**. Considerato il sottoproblema di dimensione (i), la variabile ad esso associata è  $c_i$  ed è così definita:

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

### 16) LIS: Teorema e dimostrazione della proprietà della sottostruttura ottima

**Teorema:** 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  $Z^i$  una tra le più lunghe sottosequenze crescenti di  $X_i$  e che termina con  $x_i$ . Allora vale che  $Z^i = Z^*|_{x_i}$ , con  $Z^* \in W_i$  e  $|Z^*| = \max_{W \in W_i} \{|W|\}$  dove  $W_i$  è l'insieme di tutte le sottosequenze crescenti di  $X_j$  che finiscono con  $x_j$  e a cui è possibile concatenare  $x_i$ , ovvero

$$W_i = \bigcup_{1 \leq j < i: x_j < x_i} \{W \text{ sottosequenza crescente di } X_j \text{ che termina con } x_j\}.$$

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

→  $Z^i = Z^*|_{x_i}$

→  $|Z^i| > |Z^*|$

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  $z' < x_i$ , poichè è stato possibile



concatenare  $x_i$  a  $Z'$ . Inoltre, sia  $h < i$  il più grande indice tale che  $x_h = z'$ . Di conseguenza, per come è stato definito  $W_i$ , si ottiene che  $Z' \in W_i$ . Infatti  $Z'$  è una sottosequenza crescente di  $X_h$ , la quale termina con  $x_h < x_i$ . Ciò però porta ad una contraddizione: infatti dal punto 2 vale che  $|Z'| > |Z^*|$ , ma ciò è in contraddizione con l'ipotesi che  $|Z^*| = \max_{W \in W_i} \{|W|\}$ .

### 17) Equazione di ricorrenza

Un'equazione di ricorrenza è composta da:

→ **caso base**: definisce i casi più semplici che possono essere subito risolti senza ricorrere alle soluzioni dei sottoproblemi più piccoli e lo si ha per un qualunque sottoproblema di dimensione ( $i$ ) con  $i = 0 \vee i = 1$ , ossia quando il prefisso considerato è la sequenza vuota oppure è una sequenza composta da un singolo elemento

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

→ **passo ricorsivo**: lo si ha per un qualunque sottoproblema di dimensione ( $i$ ) tale che  $i > 1$ , ossia quando si considera un prefisso della sequenza  $X$  in input di almeno due elementi e i dati disponibili per calcolare  $c_i$  sono: l'input  $X$  ed in particolare l'elemento  $x_i$  e tutte le variabili  $\{c_0, \dots, c_{i-1}\}$ .

Il passo ricorsivo è quindi scrivibile come:

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

$i$	0	1	2	3	4	5	6	7	8	9
$x_i$	$\epsilon$	2	4	7	6	11	13	21	14	1
	0	1	2	3	3	4	5	6	6	1

### 18) Algoritmo ricorsivo

```
function LISRic(i):
    if i = 1 then
        return 1
    else
        max := 0
        for h ← 1 to i - 1 do
            if  $x_h < x_i$  then
                S ← LISRic(h)
                if S > max then
                    max ← S
        return 1 + max
```

### 19) Implementazione bottom up

Con la tecnica bottom-up tutti i valori vengono calcolati in modo tale da risolvere ogni sottoproblema una volta sola. Questo permette di risolvere il problema in  $O(m^2)$  occupando  $\Theta(m)$  memoria.

**Procedura**

```

function LIS(X):
    c[1] ← 1
    max ← c[1]
    for i ← 2 to m do
        temp ← 0
        for h ← 1 to i - 1 do
            if (xh < xi) ∧ (c[h] > temp) then
                temp ← c[h]
        c[i] ← 1 + temp
        if c[i] > max then
            max ← c[i]
    return max

```

**Capitolo 4 - LICS (Longest Increasing Common Subsequence)****20) Problema: Longest Increasing Common Subsequence (LICS)**

**P1:** Date due sequenze  $X$  e  $Y$ , rispettivamente di  $m$  e  $n$  numeri interi, si determini una tra le più lunghe sottosequenze crescenti comuni a  $X$  e  $Y$ .

**Esempio:**  $X = \langle 2, 4, 7, 11, 21, 14, 1 \rangle$ ,  $Y = \langle 2, 7, 4, 23, 21, 14, 1, 8 \rangle$

$Z = \langle 2, 4, 21 \rangle = \text{LICS}(X, Y)$

**P2:** Date due sequenze  $X$  e  $Y$ , rispettivamente di  $m$  e  $n$  numeri interi, si determini la lunghezza tra le più lunghe sottosequenze crescenti comuni a  $X$  e  $Y$ .

**Esempio:**

$X = \langle 2, 4, 7, 11, 21, 14, 1 \rangle$

$Y = \langle 2, 7, 4, 23, 21, 14, 1, 8 \rangle$

$|Z| = 3$

**21) Sottoproblemi e variabili associate**

Il sottoproblema di dimensione  $(i, j)$  è definito come segue:

"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  $Y_j$ ".

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). Considerato il sottoproblema di dimensione  $(i, j)$ , la variabile ad esso associata è  $c_{i,j}$  ed è così definita:

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

**22) Teorema e dimostrazione della proprietà della sottostruttura ottima**

**Teorema:** 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 di  $X_i$  e  $Y_j$  tale che termini con  $x_i = y_j$ . Allora vale che  $Z^{i,j} = Z^*|_{x_i}$ , con  $Z^* \in W_{i,j}$  e  $|Z^*| = \max_{W \in W_{i,j}}\{|W|\}$  dove  $W_{i,j}$  è l'insieme di tutte le sottosequenze crescenti comuni di  $X_h$  e  $Y_k$  che finiscono con  $x_h = y_k$  e a cui è possibile concatenare  $x_i$  (o  $y_i$ ), ovvero

$$W_{i,j} = \bigcup_{1 \leq h < i, 1 \leq k < j: x_h = y_k < x_i = y_j} \{W \text{ sottosequenza comune crescente di } X_h \text{ e } Y_k \text{ che termina con } x_h = y_k\}.$$

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

$$\rightarrow Z^{i,j} = Z^*|_{x_i}$$

$$\rightarrow |Z'| > |Z^*|$$

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  $z' < x_i = y_j$ , poichè è stato possibile concatenare  $x_i$  (o  $y_j$ ) a  $Z'$ . Inoltre, siano  $r < i$  e  $s < j$  i più grande indici tale che  $x_r = y_s = z'$ . Di conseguenza, per come è stato definito  $W_{i,j}$ , si ottiene che  $Z' \in W_{i,j}$ . 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  $|Z^*| = \max_{W \in W_{i,j}}\{|W|\}$ .

### 23) Equazione di ricorrenza

Un'equazione di ricorrenza è composta da:

→ **caso base:** definisce i casi più semplici che possono essere subito risolti senza ricorrere alle soluzioni dei sottoproblemi più piccoli e lo si ha per un qualunque sottoproblema di dimensione  $(i, j)$  con  $i = 0 \vee j = 0$  ma anche  $x_i \neq y_j$  ossia quando i due prefissi considerati terminano con due elementi diversi

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

→ **passo ricorsivo:** lo si ha per un qualunque sottoproblema di dimensione  $(i, j)$  tale che  $x_i = y_j$ , ossia quando i due prefissi  $X_i$  e  $Y_j$  è uguale alla lunghezza della più lunga sottosequenza crescente comune calcolata per un sottoproblema di dimensione minore di  $x_i = y_j$  aumentata di uno. Il passo ricorsivo è quindi scrivibile come:

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

		1	2	3	4	5	6	7	8	j
		2	7	4	23	21	14	1	8	y <sub>j</sub>
1	2	1	0	0	0	0	0	0	0	0
2	4	0	0	2	0	0	0	0	0	0
3	7	0	2	0	0	0	0	0	0	0
4	11	0	0	0	0	0	0	0	0	0
5	21	0	0	0	0	3	0	0	0	0
6	14	0	0	0	0	0	3	0	0	0
7	1	0	0	0	0	0	0	1	0	0
i	x <sub>i</sub>									

## 24) Algoritmo ricorsivo

```

function LICSRic(i, j):
    if i = 0 ∨ j = 0 ∨ xi ≠ yj then
        return 0
    else
        max ← 0
        for h ← 1 to i - 1 do
            for k ← 1 to j - 1 do
                if xh < xi then
                    S ← LGCSRic(h, k)
                    if S > max then
                        max ← S
        return 1 + max

```

## 25) Implementazione bottom up

Con la tecnica bottom-up tutti i valori vengono calcolati in modo tale da risolvere ogni sottoproblema una volta sola. Questo permette di risolvere il problema in  $O(m^2 * n^2)$  occupando  $\Theta(m * n)$ .

Procedura

```

function LICs(X, Y):
    max ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            if xi ≠ yj then
                c[i, j] ← 0
            else
                temp ← 0
                for h ← 1 to i - 1 do
                    for k ← 1 to j - 1 do
                        if (xh < xi) ∧ (c[h, k] > temp) then
                            temp ← c[h, k]
                c[i, j] ← 1 + temp
                if c[i, j] > max then
                    max ← c[i, j]
    return max

```

## Capitolo 5 - Longest Common Subsequence (LCS) con al più K elementi rossi

### 26) Input e Output del Problema

**Input:**  $X = \langle x_1, x_2, x_3, \dots, x_m \rangle$ ,  $Y = \langle y_1, y_2, y_3, \dots, y_n \rangle$ :  $|X| = m \wedge |Y| = n \wedge K \in \mathbb{Z}^+$ .

Sia ora la funzione

$col: N \rightarrow C \subseteq \Sigma^*$

$C = \{\text{'Rosso'}, \text{'Blu'}, \text{'Nero'}\}$

**Output:**  $|LCS_k(X, Y)| = |\langle z_1, z_2, z_3, \dots, z_k \rangle|$  con al più  $k$  elementi rossi

**27) Definizione dei sottoproblemi e variabili associate**

Trovare la LCS dei prefissi  $X_i$  e  $Y_j$  che ha al più  $k$  elementi rossi  $\rightarrow \text{LCS}_k(X_i, Y_j)$

$i \in \{0, 1, 2, \dots, m\}$

$j \in \{0, 1, 2, \dots, n\}$

$k \in \{0, 1, 2, \dots, K\}$

**Numero di sottoproblemi:**  $(m + 1) * (n + 1) * (K + 1)$

**Coefficiente del sottoproblema di dimensione  $(i, j, k)$ :**  $c_{i,j,k}$  è la lunghezza della LCS dei prefissi  $X_i$  e  $Y_j$  che ha al più  $k$  elementi rossi.

**28) Equazione di ricorrenza: caso base**

Se  $i = 0 \vee j = 0 \rightarrow c_{i,j,k} = 0$ , t.c.  $k \in \mathbb{Z}^+$

**29) Equazione di ricorrenza: passo ricorsivo**

Se  $i > 0 \wedge j > 0$ , allora

se  $x_i \neq y_j \rightarrow c_{i,j,k} = \max\{c_{i-1,j,k}, c_{i,j-1,k}\}$

se  $x_i = y_j \wedge \text{col}(x_i) \neq \text{'Rosso'} \rightarrow c_{i,j,k} = c_{i-1,j-1,k} + 1$

se  $x_i = y_j \wedge \text{col}(x_i) = \text{'Rosso'} \wedge k > 0 \rightarrow c_{i,j,k} = c_{i-1,j-1,k-1} + 1$

se  $x_i = y_j \wedge \text{col}(x_i) = \text{'Rosso'} \wedge k = 0 \rightarrow c_{i,j,k} = c_{i-1,j-1,k-1}$

**30) Soluzione del problema**

La soluzione di tale problema è il coefficiente  $c_{m,n,k}$

**31) Algoritmo Bottom Up per il calcolo del valore ottimo**

```
function LCS_at_most_k_red(X, Y, K)
  m := X.length
  n := Y.length

  for k := 0 to K do
    for i := 0 to m do
      ck[i, 0] := 0
    for k := 0 to K do
      for j := 0 to n do
        ck[0, j] := 0
    for k := 0 to K do
      for i := 1 to m do
        for j := 1 to n do
          if xi ≠ yj then
            ck[i, j] := max{ck[i - 1, j], ck[i, j - 1]}
          else
            if col(xi) ≠ 'Rosso' then
              ck[i, j] := ck[i - 1, j - 1] + 1
            else
              if k > 0 then
                ck[i, j] := ck-1[i - 1, j - 1] + 1
              if k = 0 then
                ck[i, j] := ck-1[i - 1, j - 1]

  return ck[m, n]
```

### 32) Ricostruzione dell'algoritmo

```

function LCS_most_k_red(i, j, k)
    if i > 0 ∧ j > 0 then
        if xi ≠ yj then
            return max{LCS_most_k_red(i - 1, j, k), LCS_most_k_red(i, j - 1, k)}
        else
            if col(xi) ≠ 'Rosso' then
                LCS_most_k_red(i - 1, j - 1, k)
                print(xi)
            else
                if k > 0 then
                    LCS_most_k_red(i - 1, j - 1, k - 1)
                    print(xi)
                if k = 0 then
                    LCS_most_k_red(i - 1, j - 1, k - 1)

```

Capitolo 6 - Longest Common Subsequence (LCS) con ingombro minore o uguale di K

### 33) Input e Output del Problema

**Input:**  $X = \langle x_1, x_2, x_3, \dots, x_m \rangle$ ,  $Y = \langle y_1, y_2, y_3, \dots, y_n \rangle$ :  $|X| = m \wedge |Y| = n \wedge K \in \mathbb{Z}^+$ .

Sia ora la funzione

$w: \Sigma \rightarrow \mathbb{N}$

**Output:**  $|LCS_k(X, Y)| = |\langle z_1, z_2, z_3, \dots, z_k \rangle|$  con ingombro al più K

### 34) Definizione dei sottoproblemi e variabili associate

Trovare la **LCS** dei prefissi  $X_i$  e  $Y_j$  con ingombro al più K  $\rightarrow LCS_k(X_i, Y_j)$

$i \in \{0, 1, 2, \dots, m\}$

$j \in \{0, 1, 2, \dots, n\}$

$k \in \{0, 1, 2, \dots, K\}$

**Numero di sottoproblemi:**  $(m + 1) * (n + 1) * (K + 1)$

**Coefficiente del sottoproblema di dimensione (i, j, k):**  $c_{i,j,k}$  è la lunghezza della LCS dei prefissi  $X_i$  e  $Y_j$  con ingombro minore.

### 35) Equazione di ricorrenza: caso base

Se  $i = 0 \vee j = 0 \vee k = 0 \rightarrow c_{i,j,k} = 0$ , t.c.  $k \in \mathbb{Z}^+$

### 36) Equazione di ricorrenza: passo ricorsivo

Se  $i > 0 \wedge j > 0$ , allora

se  $x_i \neq y_j \rightarrow c_{i,j,k} = \max\{c_{i-1,j,k}, c_{i,j-1,k}\}$

se  $x_i = y_j \wedge w(x_i) \leq k \rightarrow c_{i,j,k} = \max\{c_{i-1,j-1,k-w(x_i)} + 1, c_{i-1,j-1,k-1}\}$

se  $x_i = y_j \wedge w(x_i) > k \wedge k > 0 \rightarrow c_{i,j,k} = c_{i-1,j-1,k}$

### 37) Soluzione del problema

La soluzione di tale problema è il coefficiente  $c_{m,n,K}$

### 38) Algoritmo Bottom Up

```
function LCS_ingombro_most_k(X, Y, K)
  m := X.length
  n := Y.length

  for k := 0 to K do
    for i := 0 to m do
      ck[i, 0] := 0
  for k := 0 to K do
    for j := 0 to n do
      ck[0, j] := 0
  for k := 0 to K do
    for i := 1 to m do
      for j := 1 to n do
        if xi ≠ yj then
          ck[i, j] := max{ck[i - 1, j], ck[i, j - 1]}
        else
          if w(xi) ≤ k then
            ck[i, j] := max{ck-w(xi)[i - 1, j - 1] + 1, ck[i - 1, j - 1]}
          else
            ck[i, j] := ck[i - 1, j - 1]

  return ck[m, n]
```

### 39) Ricostruzione dell'algoritmo

```
function LCS_ing_most_k(i, j, k)
  if i > 0 ∧ j > 0 then
    if xi ≠ yj then
      return max{LCS_ing_most_k(i - 1, j, k), LCS_ing_most_k(i, j - 1, k)}
    else
      if w(xi) ≤ k then
        return max{LCS_ing_most_k(i - 1, j, k - w(xi)),
                    LCS_ing_most_k(i - 1, j - 1, k - 1)}
        print(xi)
      else
        return LCS_ing_most_k(i - 1, j - 1, k)
```

Capitolo 7 - Longest Alternative Common Subsequence (LACS) con colori alternanti

### 40) Input e Output del Problema

**Input:**  $X = \langle x_1, x_2, x_3, \dots, x_m \rangle$ ,  $Y = \langle y_1, y_2, y_3, \dots, y_n \rangle$ :  $|X| = m \wedge |Y| = n$

**Sia ora:**

$\text{col}: N \rightarrow C = \{R, N, B\}$

**Output:**  $|LCS_k(X, Y)| = |\langle z_1, z_2, z_3, \dots, z_k \rangle|$  con colori alternanti.

Considerare che

$$\phi(x) = \begin{cases} \text{rosso} & x < 5 \\ \text{blu} & 5 \leq x \leq 10 \\ \text{verde} & x > 10 \end{cases}$$

#### 41) Definizione dei sottoproblemi e variabili associate

Trovare la **LCS** dei prefissi  $X_i$  e  $Y_j$  con colori alternanti associati.

$i \in \{0, 1, 2, \dots, m\}$

$j \in \{0, 1, 2, \dots, n\}$

**Numero di sottoproblemi:**  $(m + 1) * (n + 1)$

**Coefficiente del sottoproblema di dimensione  $(i, j, k)$ :**  $c_{i,j}$  è la lunghezza della LCS dei prefissi  $X_i$  e  $Y_j$  con colori alternanti associati.

#### 42) Equazione di ricorrenza: caso base

Se  $i = 0 \vee j = 0 \rightarrow c_{i,j} = 0$ .

#### 43) Equazione di ricorrenza: passo ricorsivo

Se  $i > 0 \wedge j > 0$ , allora

se  $x_i \neq y_j \rightarrow c_{i,j} = \max\{c_{i-1,j}, c_{i,j-1}\}$

se  $x_i = y_j \rightarrow c_{i,j} = 1 + \max\{c_{h,k} \mid 1 \leq h < i, 1 \leq k < j, \phi(x_i) = \phi(x_h)\}$

#### 44) Soluzione del problema

La soluzione di tale problema è  $\max\{c_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$

#### 45) Algoritmo Bottom Up

```
function LACS(X, Y)
    max := 0
    for i := 1 to m do
        for j := 1 to n do
            if  $x_i \neq y_j$  then
                 $c[i, j] := 0$ 
            else
                temp := 0
                for h := 1 to  $i-1$  do
                    for k := 1 to  $j-1$  do
                        if  $(\phi(x_h) = \phi(x_i)) \wedge (c[h, k] > temp)$  then
                             $temp \leftarrow c[h, k]$ 
                 $c[i, j] := 1 + temp$ 
            if  $c[i, j] > max$  then
                 $max := c[i, j]$ 
    return max
```



#### 46) Algoritmo di Stampa

```

function LACSric(i, j)
    if i = 0 ∨ j = 0 ∨ xi ≠ yj then
        return 0
    else
        max ← 0
        for h ← 1 to i-1 do
            for k ← 1 to j-1 do
                if φ(xh) = φ(xi) then
                    S ← LACSric(h, k)
                if S > max then
                    max ← S
        return 1+max

```

### Capitolo 8 - Weighted Interval Scheduling

#### 47) Input e Output del Problema

**Input:**  $A = \{1, \dots, n\}$  con  $([s_i, e_i], v_i), \forall i \in \{1, \dots, n\}$ , dove

$s_i \rightarrow$  indica il tempo di inizio dell'attività;

$e_i \rightarrow$  tale che  $e_i \leq s_i$ , indica il tempo di fine dell'attività (si noti che  $e_i$  non appartiene all'intervallo che specifica la durata dell'attività);

$v_i \rightarrow$  indica il valore dell'attività

Due attività vengono definite compatibili se non si sovrappongono:

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

La funzione

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

determina se l'insieme  $A$  contiene tutte attività compatibili

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

La funzione

$$V : \mathcal{P}(\{1, \dots, n\}) \rightarrow \mathbb{R}_+$$

determina il valore complessivo del sottoinsieme  $S$  di attività  $A$ .

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

**Output:** Determinare  $S \subseteq \{1, \dots, n\}$  tale che

$$\text{COMP}(S) = \text{True} \wedge V(S) = \max_{A \subseteq \{1, \dots, n\}: \text{COMP}(A) = \text{True}} \{V(A)\}$$

#### 48) Definizione dei sottoproblemi e variabili associate

In questo caso il sottoproblema  $i$ -esimo viene associato al sottoinsieme  $i$ -esimo tale che

$$\text{COMP}(S) = \text{True} \wedge V(S) = \max_{A \subseteq \{1, \dots, n\}: \text{COMP}(A) = \text{True}} \{V(A)\}$$

La variabile associata è la coppia  $(\text{OPT}_i, S_i)$  tale che:

$\text{OPT}_i = V(S_i)$ , ossia il valore di un sottoinsieme di  $\{1, \dots, i\}$  che contiene attività mutualmente compatibili di peso massimo  $S_i$

Numero di sottoproblemi:  $(m + 1)$

49) Equazione di ricorrenza: caso base

Se  $i = 0 \rightarrow S_i = \emptyset \wedge OPT_i = 0$

50) Equazione di ricorrenza: passo ricorsivo

Se  $i > 0$ , allora si definisce

$$\psi(i) = \max\{j \mid j < i \wedge \text{comp}(\{i, j\}) = \text{true}\}$$

assumendo che  $\max\{\emptyset\} = 0$ .

Per determinare il passo ricorsivo si distinguono i due casi:

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

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

$\rightarrow i \in S_i$ : 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 compatibile con l'attività  $i$ .

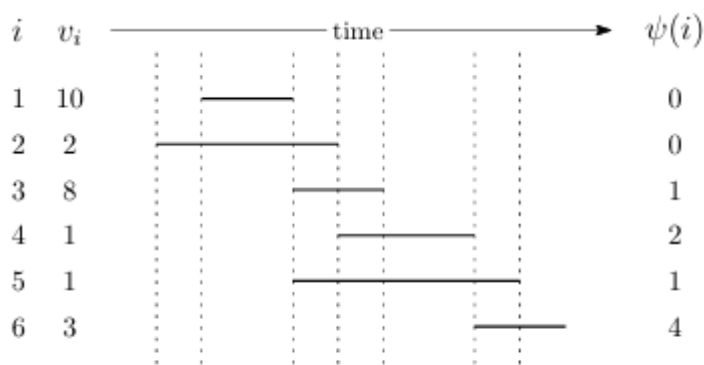
Risulta dunque possibile considerare la soluzione del sottoproblema di dimensione  $\psi(i)$  e aggiungere a questa l'attività  $i$ . Si ottiene dunque:

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

Risulta quindi

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

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



OPT	0	10	2	18	18	18	21
	0	1	2	3	4	5	6

$$S = \{1, 3, 6\}$$

51) Soluzione del problema

La soluzione di tale problema è la coppia  $(OPT_n, S_n)$

## 52) Algoritmo Bottom Up

```

function WIS(n)
  OPT[0] ← 0
  S ← ∅

  for i ← 1 to n do
    V1 ← OPT[i-1]
    V2 ← OPT[ψ(i)] + vi
    OPT[i] ← max{V1, V2}
    if V1 ≥ V2 then
      S[i] ← S[i-1]
    else
      S[i] ← S[ψ(i)] ∪ {i}

  return (OPT[n], S[n])

```

## Capitolo 9 - Hateville

## 54) Input e Output del Problema

**Input:**  $X = \{1, \dots, n\} \wedge d_i, \forall i \in \{1, \dots, n\}$

**Output:**  $S \subseteq \{1, \dots, n\} : COMP(S) = true \wedge D(S) = \max\{D(A), A \subseteq X_n : COMP(A)\}$

## 55) Definizione dei sottoproblemi e variabili associate

Dato un insieme  $X = \{1, \dots, i\}$  rappresentante i primi  $i$  abitanti di Hateville, trovare un suo sottoinsieme, che rispetti la compatibilità, che massimizzi il denaro raccolto.

**Input:**  $\{1, \dots, i\} \wedge d_j, \forall j \in \{1, \dots, j\}$

**Output:**  $S_i \subseteq \{1, \dots, i\} : COMP(S_i) = true \wedge D(S_i) = \max\{D(A), A \subseteq X_n : COMP(A)\}$

## 56) Equazione di ricorrenza: caso base

Se  $i = 0$ , allora significa che non ci sono abitanti in Hateville e quindi non c'è nessuno che può partecipare alla colletta. Se  $i = 1$ , allora significa che c'è un solo abitante in Hateville e quindi non è necessario fare nessuna scelta per evitare di inserire i vicini.

Il caso base è:

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

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

## 57) Equazione di ricorrenza: passo ricorsivo

Il **passo ricorsivo** si ha per un qualunque sottoproblema di dimensione ( $i$ ) con

$i > 1$ , ossia quando ci sono almeno due abitanti in **Hateville**.

Si ha quindi:

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

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

### 58) Soluzione del problema

La soluzione del problema é  $(OPT_n, S_n)$ .

### 59) Algoritmo Bottom Up

```
function HatevilleRic(i)
  if i = 0 then
    return (0, ∅)
  else
    (V1, S1) := HatevilleRic(i - 1)
    (V2, S2) := HatevilleRic(i - 2)
    V2 := V2 + di
    S2 := V2 ∪ {i}
    if V1 ≥ V2 then
      return (V1, S1)
    else
      return (V2, S2)
```

### 60) Ricostruzione dell'algoritmo

```
function HatevilleRic(n)
  OPT[0] := 0
  S[0] := 0
  OPT[1] := d1
  S[1] := {1}
  for i := 2 to n do
    V1 := OPT[i - 1]
    V2 := OPT[i - 2] + di
    if V1 ≥ V2 then
      S[i] := S[i - 1]
      OPT[i] := V1
    else
      S[i] := S[i - 2] ∪ {i}
      OPT[i] := V2
  return (OPT[n], S[n])
```

## Capitolo 10 - Edit Distance

### 61) Definizione del problema

Date due sequenze  $X = \langle x_1, \dots, x_m \rangle$  e  $Y = \langle y_1, \dots, y_n \rangle$ , rispettivamente di lunghezza  $m$  ed  $n$  definite su un alfabeto  $\Sigma$ , si determini il minimo numero di operazioni elementari che permette di trasformare  $X$  in  $Y$ .

### 62) Definizione dei sottoproblemi e variabili associate

Date due sequenze  $X = \langle x_1, \dots, x_m \rangle$  e  $Y = \langle y_1, \dots, y_n \rangle$ , rispettivamente di lunghezza  $m$  ed  $n$  definite su un alfabeto  $\Sigma$ , si determini il minimo numero di operazioni elementari che permette di trasformare  $X_i$  in  $Y_j$ . L'equazione  $\delta_{i,j}$  è il numero minimo di operazioni elementari che permette di trasformare  $X_i$  in  $Y_j$ .

### 63) Equazione di ricorrenza: caso base

Il caso base si ha per un qualunque sottoproblema di dimensione  $(i, j)$  con  $i = 0 \wedge j = 0$ , ossia quando uno dei due prefissi considerati è la sequenza vuota.

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

### 64) Equazione di ricorrenza: passo ricorsivo

Il passo ricorsivo si ha per un qualunque sottoproblema di dimensione  $(i, j)$  con  $i > 0 \wedge j > 0$ , ossia quando entrambi i prefissi considerati non sono vuoti.

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

### 65) Soluzione del problema

La soluzione del problema è  $\delta_{m,n}$ .

### 66) Algoritmo Ricorsivo

```

procedure EDRIC(i, j)
  if i = 0 ∨ j = 0 then
    if i = 0 then
      return j
    else
      return i
  else
    if xi = yj then
      return EDRIC(i - 1, j - 1)
    else
      ins ← 1 + EDRIC(i, j - 1)
      del ← 1 + EDRIC(i - 1, j)
      rep ← 1 + EDRIC(i - 1, j - 1)
      return MIN(ins, del, rep)

```

## 67) Algoritmo Bottom Up

```

procedure ED(X, Y)
  for i ← 0 to m do
     $\delta[i, 0] \leftarrow i$ 
  for j ← 1 to n do
     $\delta[0, j] \leftarrow j$ 
  for i ← 1 to m do
    for j ← 1 to n do
      if  $x_i = y_j$  then
         $\delta[i, j] \leftarrow \delta[i - 1, j - 1]$ 
      else
         $\delta[i, j] \leftarrow 1 + \text{MIN}(\delta[i, j - 1], \delta[i - 1, j], \delta[i - 1, j - 1])$ 
  return  $\delta[m, n]$ 

```

## Capitolo 11 - Interleaving

## 68) Definizione del problema

Date tre sequenze, definite su un alfabeto  $\Sigma$ :

$X = \langle x_1, \dots, x_m \rangle$ , tale che  $|X| = m$

$Y = \langle y_1, \dots, y_n \rangle$ , tale che  $|Y| = n$

$W = \langle w_1, \dots, w_{m+n} \rangle$ , tale che  $|W| = m + n$

stabilire se  $W$  è un **interleaving** di  $X$  e  $Y$ , ovvero se  $X$  e  $Y$  si possono trovare come due sottosequenze disgiunte in  $W$ .

## 69) Definire il coefficiente che risolve i vari sottoproblemi

Il coefficiente  $s_{i,j}$  determina se  $W_{i+j}$  è **interleaving** dei prefissi  $X_i$  e  $Y_j$ .

## 70) Equazione di ricorrenza: caso base

Se  $i = 0 \vee j = 0$ , ossia uno dei prefissi è vuoto, allora risulta necessario considerare i seguenti casi:

→ se  $i = 0 \wedge j = 0$ , allora si avrebbe che  $W_{i+j}$  rappresenta la sequenza vuota e quindi vale che  $s_{i,j} = \text{True}$ ;

→ se  $i = 0 \wedge j > 0 \wedge w_j = y_j$ , allora  $w_j$  è **interleaving** solo se lo è anche  $W_{i,j-1}$  e quindi  $s_{i,j} = s_{i,j-1}$ ;

→ se  $i = 0 \wedge j > 0 \wedge w_j \neq y_j$ , allora  $w_j$  non può essere **interleaving** e quindi  $s_{i,j} = \text{False}$ ;

→ se  $i > 0 \wedge j = 0 \wedge w_i \neq x_i$ , allora  $w_i$  non può essere **interleaving** e quindi  $s_{i,j} = \text{False}$ ;

→ se  $i > 0 \wedge j = 0 \wedge w_i = x_i$ , allora  $w_i$  è **interleaving** solo se lo è anche  $W_{i-1,j}$  e quindi  $s_{i,j} = s_{i-1,j}$ ;

Riassumendo si avrebbe:

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

### 71) Equazione di ricorrenza: passo ricorsivo

Se  $i > 0 \wedge j > 0$ , ossia nessuno dei due prefissi è vuoto, allora risulta necessario considerare i seguenti casi:

→ se  $w_{i+j} \neq x_i \wedge w_{i+j} \neq y_j$ , allora  $w_{i+j}$  non può essere **interleaving** e quindi  $s_{i,j} = False$ ;

→ se  $w_{i+j} = x_i \wedge w_{i+j} \neq y_j$ , allora  $w_{i+j}$  è **interleaving** solo se lo è anche  $W_{i-1,j}$  e quindi  $s_{i,j} = s_{i-1,j}$ ;

→ se  $w_{i+j} \neq x_i \wedge w_{i+j} = y_j$ , allora  $w_{i+j}$  è **interleaving** solo se lo è anche  $W_{i,j-1}$  e quindi  $s_{i,j} = s_{i,j-1}$ ;

→ se  $w_{i+j} = x_i \wedge w_{i+j} = y_j$ , allora  $w_{i+j}$  è **interleaving** solo se lo è  $W_{i-1,j}$  e  $W_{i,j-1}$  e quindi  $s_{i,j} = s_{i-1,j} \vee s_{i,j-1}$

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

### 72) Soluzione del problema

La soluzione del problema è  $s_{m,n}$ .

## Capitolo 12 - Knapsack

### 73) Input e Output del Problema

**Input:**  $X = \{1, \dots, n\}: (v_i, w_i), \forall i \in \{1, \dots, n\}$

**Output:**  $S \subseteq \{1, \dots, n\}: W(S) \leq C \wedge V(S) = \max_{A \subseteq \{1, \dots, n\}: W(A) \leq C} \{V(A)\}$

### 74) Definizione dei sottoproblemi e variabili associate

Dato un insieme  $\{1, \dots, i\}$  di oggetti, trovare un suo sottoinsieme di ingombro complessivo  $\leq C$  e di massimo valore complessivo.

Si ha quindi:

**Input:**  $X_i = \{1, \dots, i\}: (v_j, w_j), \forall j \in \{1, \dots, i\}$

**Output:**  $S \subseteq \{1, \dots, i\}: W(S_{i,c}) \leq c \wedge V(S_{i,c}) = \max_{A \subseteq \{1, \dots, i\}: W(A) \leq c} \{V(A)\}$

Dato che  $0 \leq i \leq n$  si ottengono  $(n + 1)$  sottoproblemi, ad ognuno dei quali è associata una coppia di variabili. Si ha quindi  $(OPT_i, S_i)$ , dove:

$OPT_{i,c} = V(S_{i,c})$ , ossia il valore di un sottoinsieme di  $\{1, \dots, i\}$  di ingombro complessivo  $\leq c$  e di massimo valore complessivo.

### 75) Equazione di ricorrenza: caso base

Se  $i = 0 \vee c = 0$ , allora vale

$$OPT_{i,c} = 0 \wedge S_{i,c} = \emptyset \text{ se } i = 0 \vee c = 0$$

### 76) Equazione di ricorrenza: passo ricorsivo

Quando  $i > 0 \wedge c > 0$ , allora risulta necessario considerare i seguenti casi:

→  $w_i > c$ : se l'oggetto ha ingombro maggiore della capacità, allora l'oggetto non può appartenere alla soluzione e quindi si va a considerare la soluzione del sottoproblema di dimensione minore  $i - 1, c$ . In questo caso

$$OPT_{i,c} = OPT_{i-1,c} \wedge S_{i,c} = S_{i-1,c} \text{ se } w_i > c$$

→  $w_i \leq c$ : allora risulta necessario tenere in considerazione due casi:

a.  $i \notin S_i$ : se l'oggetto  $i$  non appartiene alla soluzione allora risulta necessario considerare l'insieme  $\{1, \dots, i - 1\}$  e quindi:

$$OPT_{i,c} = OPT_{i-1,c} \wedge S_{i,c} = S_{i-1,c} \text{ se } i \notin S_i$$

b.  $i \in S_i$ : se l'oggetto  $i$  appartiene alla soluzione allora vale:

$$OPT_{i,c} = OPT_{i-1,c-w_i} + v_i \wedge S_{i,c} = S_{i-1,c-w_i} \cup \{i\} \text{ se } i \in S_i$$

Riassumendo si ha:

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

e

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

### 77) Soluzione del problema

La soluzione del problema è  $(OPT_{n,c}, S_{n,c})$ .



**78) Algoritmo Bottom Up**

```

procedure KP( $n, C$ )
  for  $i \leftarrow 0$  to  $n$  do
     $OPT[i, 0] \leftarrow 0$ 
     $S[i, 0] \leftarrow \emptyset$ 
  for  $c \leftarrow 0$  to  $C$  do
     $OPT[0, c] \leftarrow 0$ 
     $S[0, c] \leftarrow \emptyset$ 
  for  $i \leftarrow 1$  to  $n$  do
    for  $c \leftarrow 1$  to  $C$  do
      if  $w_i > c$  then
         $OPT[i, c] \leftarrow OPT[i - 1, c]$ 
         $S[i, c] \leftarrow S[i - 1, c]$ 
      else
         $V_1 \leftarrow OPT[i - 1, c]$ 
         $V_2 \leftarrow OPT[i - 1, c - w_i] + v_i$ 
         $OPT[i, c] \leftarrow \text{MAX}(V_1, V_2)$ 
        if  $V_1 \geq V_2$  then
           $S[i, c] \leftarrow S[i - 1, c]$ 
        else
           $S[i, c] \leftarrow S[i - 1, c - w_i] \cup \{i\}$ 
  return ( $OPT[n, C], S[n, C]$ )

```

**79) Algoritmo Ricorsivo**

```

procedure KPRIC( $i, c$ )
  if  $i = 0 \vee c = 0$  then
    return ( $0, \emptyset$ )
  else
    if  $w_i > c$  then
      return KPRIC( $i - 1, c$ )
    else
      ( $V_1, S_1$ )  $\leftarrow$  KPRIC( $i - 1, c$ )
      ( $V_2, S_2$ )  $\leftarrow$  KPRIC( $i - 1, c - w_i$ )
       $V_2 \leftarrow V_2 + v_i$ 
      if  $V_1 \geq V_2$  then
        return ( $V_1, S_1$ )
      else
        return ( $V_2, S_2 \cup \{i\}$ )

```

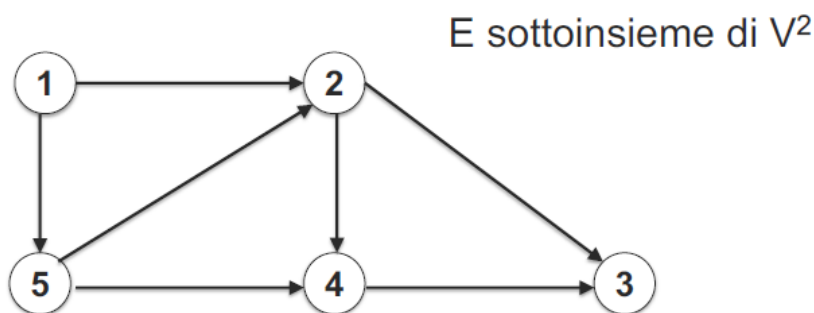
**Capitolo 13 - Grafi: Floyd - Warshall****80) Input e Output del Problema**

**Input:** Si ha un grafo  $G = (V, E, W)$  (senza cappi) orientato e pesato tale che:

$\rightarrow V = \{v_1, v_2, \dots, v_n\}$

$\rightarrow E \subseteq V^2$

$\rightarrow W: E \rightarrow R^+: w(i, j) = w_{ij}$



$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1,2), (1,5), (2,3), (2,4), (4,3), (5,2), (5,4)\}$$

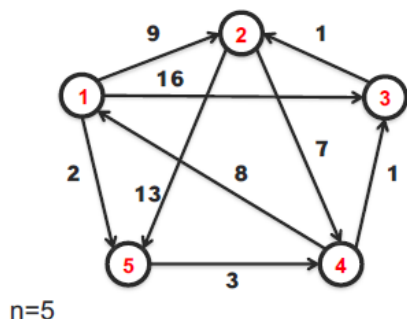
**Output:** Per ogni coppia di vertici  $i$  e  $j$ , trovare il cammino di peso minimo (cammino minimo) che parte da  $i$  e finisce in  $j$ .

### 81) Definizione dei sottoproblemi e variabili associate

Il coefficiente  $d_{i,j}^k$  determina il peso del cammino minimo dal vertice  $i$  al vertice  $j$ , con vertici intermedi  $\in \{1, \dots, k\}$  se  $k > 0$  senza vertici intermedi se  $k = 0$ .

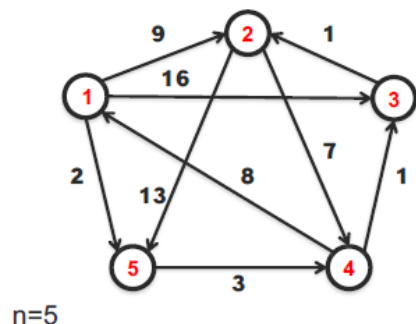
Vale quindi  $k \in \{0, 1, \dots, n\}, i \in \{1, \dots, n\}, j \in \{1, \dots, n\}$

Qui si mostra la **matrice  $W$**  dei pesi



W	1	2	3	4	5
1	0	9	16	$\infty$	2
2	$\infty$	0	$\infty$	7	13
3	$\infty$	1	0	$\infty$	$\infty$
4	8	$\infty$	1	0	$\infty$
5	$\infty$	$\infty$	$\infty$	3	0

Qui si mostra la **matrice  $\pi$**  dei predecessori



$\Pi$	1	2	3	4	5
1	NIL	3	4	5	1
2	4	NIL	4	2	2
3	4	3	NIL	2	2
4	4	3	4	NIL	1
5	4	3	4	5	NIL

### 82) Equazione di ricorrenza: caso base

Se  $k = 0$ , allora:

$\rightarrow d_{i,j}^0 = 0$ , se  $i = j$ ;

→  $d_{i,j}^0 = w_{ij}$ , se  $i \neq j \wedge (i,j) \in E$

→  $d_{i,j}^0 = \infty$ , se  $i \neq j \wedge (i,j) \notin E$

Riassumendo

$$d_{i,j}^0 = \begin{cases} 0 \\ w_{ij} \\ \infty \end{cases}$$

Per quanto riguarda i predecessori valgono le seguenti condizioni:

→  $\pi_{i,j}^0 = NIL$ , se  $i = j$ ;

→  $\pi_{i,j}^0 = 0$ , se  $i \neq j \wedge (i,j) \in E$

→  $\pi_{i,j}^0 = NIL$ , se  $i \neq j \wedge (i,j) \notin E$

Riassumendo

$$\pi_{i,j}^0 = \begin{cases} NIL \\ 0 \\ NIL \end{cases}$$

### 83) Equazione di ricorrenza: passo ricorsivo

Se  $k > 0$ , allora vale che:

→ se  $k \notin p$ , allora

$$d_{i,j}^k = d_{i,j}^{k-1} \text{ e } \pi_{i,j}^k = \pi_{i,j}^{k-1}$$

→ se  $k \in p$ , allora

$$d_{i,j}^k = d_{i,k}^{k-1} + d_{k,j}^{k-1} \text{ e } \pi_{i,j}^k = \pi_{k,j}^{k-1}$$

Riassumendo

$$d_{i,j}^k = \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\}$$

$$\pi_{i,j}^k = \begin{cases} \pi_{i,j}^{k-1} \\ \pi_{k,j}^{k-1} \end{cases}$$

### 84) Soluzione del problema

Le soluzioni del problema sono  $d_{i,j}^n$  e  $\pi_{i,j}^n$

## 85) Algoritmo Bottom Up

**Procedura calcola\_valori\_ottimi\_FW(V,E,W)** $D^0 \leftarrow W$  $\Pi^0 \leftarrow (n \times n)$  matrix of NIL values**for**  $i \leftarrow 1$  to  $n$  **do**    **for**  $j \leftarrow 1$  to  $n$  **do**        **if**  $i \neq j$  **and**  $w_{ij} \neq \infty$  **then**             $\Pi^0[i,j] \leftarrow i$ **for**  $k \leftarrow 1$  to  $n$  **do**    **for**  $i \leftarrow 1$  to  $n$  **do**        **for**  $j \leftarrow 1$  to  $n$  **do**             $D^k[i,j] \leftarrow D^{k-1}[i,j]$              $\Pi^k[i,j] \leftarrow \Pi^{k-1}[i,j]$             **if**  $i \neq k$  **and**  $j \neq k$  **then**                **if**  $D^k[i,j] > D^{k-1}[i,k] + D^{k-1}[k,j]$  **then**                     $D^k[i,j] \leftarrow D^{k-1}[i,k] + D^{k-1}[k,j]$                      $\Pi^k[i,j] \leftarrow \Pi^{k-1}[k,j]$