Lezione 15

Sommario della Lezione

Programmazione Dinamica

Date due sequenze di caratteri

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

Date due sequenze di caratteri

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

il **problema** consiste nel trovare la più lunga sottosequenza comune ad a e b;

Date due sequenze di caratteri

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

il **problema** consiste nel trovare la più lunga sottosequenza comune ad a e b; ovvero trovare una sottosequenza $a[i_1] \dots a[i_k]$ di a, una sottosequenza $b[j_1] \dots b[j_k]$ di b, tali che $i_1 < \dots < i_k$, $j_1 < \dots < j_k$, e

$$a[i_1] = b[j_1], \dots, a[i_k] = b[j_k]$$

Date due sequenze di caratteri

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

il **problema** consiste nel trovare la più lunga sottosequenza comune ad a e b; ovvero trovare una sottosequenza $a[i_1] \dots a[i_k]$ di a, una sottosequenza $b[j_1] \dots b[j_k]$ di b, tali che $i_1 < \dots < i_k$, $j_1 < \dots < j_k$, e

$$a[i_1] = b[j_1], \dots, a[i_k] = b[j_k]$$

con k più **grande possibile**.

Date due sequenze di caratteri

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

il **problema** consiste nel trovare la più lunga sottosequenza comune ad a e b; ovvero trovare una sottosequenza $a[i_1] \dots a[i_k]$ di a, una sottosequenza $b[j_1] \dots b[j_k]$ di b, tali che $i_1 < \dots < i_k$, $j_1 < \dots < j_k$, e

$$a[i_1] = b[j_1], \dots, a[i_k] = b[j_k]$$

con k più **grande possibile**.

Denotiamo con $LCS(a, b)$ una tale piú lunga sottosequenza comune, e con $|LCS(a, b)|$ la sua lunghezza.

Perchè ?

- Applicazioni in Biologia Molecolare

Perchè ?

- Applicazioni in Biologia Molecolare
- Applicazioni all'analisi di testi (comando `diff` in Unix)

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$a = A\ B\ C\ B\ D\ A\ B$$

$$b = B\ D\ C\ A\ B\ A$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

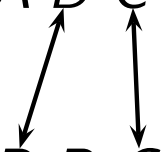
$$a = A\ B\ C\ B\ D\ A\ B$$

$$b = B\ D\ C\ A\ B\ A$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

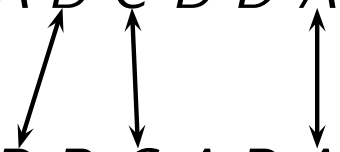
$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$a = A\ B\ C\ B\ D\ A\ B$$



$$b = B\ D\ C\ A\ B\ A$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$a = A\ B\ C\ B\ D\ A\ B$$

$$b = B\ D\ C\ A\ B\ A$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

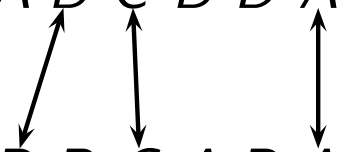
$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \quad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

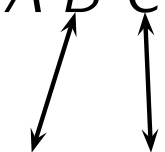
Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\!\nearrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\ \swarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\ \swarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$a = A\ B\ C\ B\ D\ A\ B$$

$$b = B\ D\ C\ A\ B\ A$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\ \swarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\ \swarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

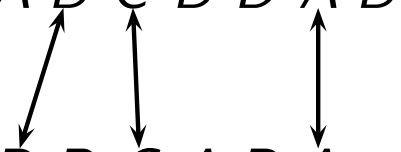
$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \qquad \swarrow\ \swarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \updownarrow\ \updownarrow\ \updownarrow\ \qquad \updownarrow \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$

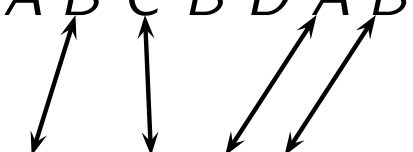
Esempi

Se $a = A\ B\ C\ B\ D\ A\ B$ e $b = B\ D\ C\ A\ B\ A$, avremmo

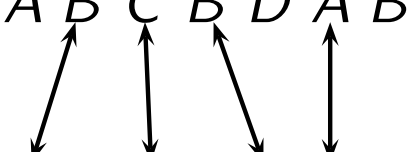
$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



$$\begin{array}{rcl} a & = & A\ B\ C\ B\ D\ A\ B \\ & & \\ b & = & B\ D\ C\ A\ B\ A \end{array}$$



Ricapitoliamo: date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

vogliamo trovare la più lunga sottosequenza $a[i_1] \dots a[i_k]$ di a e $b[j_1] \dots b[j_k]$ di b,

Ricapitoliamo: date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

vogliamo trovare la più lunga sottosequenza $a[i_1] \dots a[i_k]$ di a e $b[j_1] \dots b[j_k]$ di b, tali che $i_1 < \dots < i_k$, $j_1 < \dots < j_k$, e

$$a[i_1] = b[j_1], \dots, a[i_k] = b[j_k]$$

Ricapitoliamo: date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

vogliamo trovare la più lunga sottosequenza $a[i_1] \dots a[i_k]$ di a e $b[j_1] \dots b[j_k]$ di b, tali che $i_1 < \dots < i_k$, $j_1 < \dots < j_k$, e

$$a[i_1] = b[j_1], \dots, a[i_k] = b[j_k]$$

Primo passo: formulare in termini ricorsivi il problema.

Date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

sia $c[i, j]$ la lunghezza di una più lunga sottosequenza comune di $a[1] \dots a[i]$ e $b[1] \dots b[j]$, dove $1 \leq i \leq m$ e $1 \leq j \leq n$.

Date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

sia $c[i, j]$ la lunghezza di una più lunga sottosequenza comune di $a[1] \dots a[i]$ e $b[1] \dots b[j]$, dove $1 \leq i \leq m$ e $1 \leq j \leq n$.

Ovviamente

$$c[i, 0] = 0 = c[0, j], \quad \forall 0 \leq i \leq m,\ 0 \leq j \leq n.$$

Date

$$a = a[1] \dots a[m], \quad b = b[1] \dots b[n]$$

sia $c[i, j]$ la lunghezza di una più lunga sottosequenza comune di $a[1] \dots a[i]$ e $b[1] \dots b[j]$, dove $1 \leq i \leq m$ e $1 \leq j \leq n$.

Ovviamente

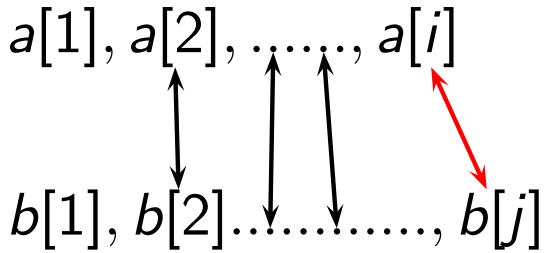
$$c[i, 0] = 0 = c[0, j], \quad \forall 0 \leq i \leq m,\ 0 \leq j \leq n.$$

Deriviamo una equazione di ricorrenza per $c[i, j]$, che esprima $c[i, j]$ in termini di $c[p, t]$, dove p e t sono "più piccoli" di i e j.

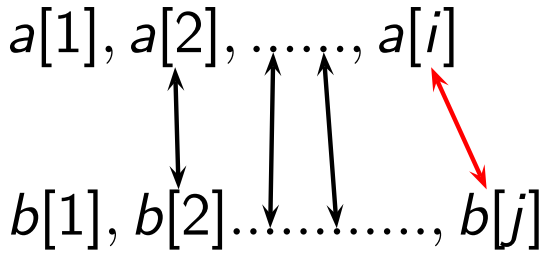
Caso 1: $a[i] \neq b[j]$.

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)

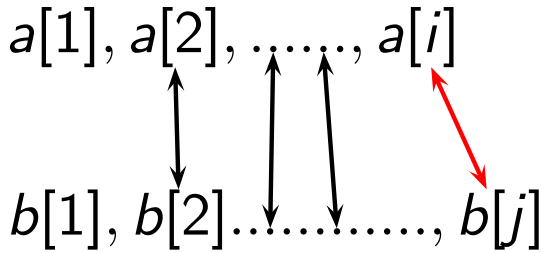


Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)



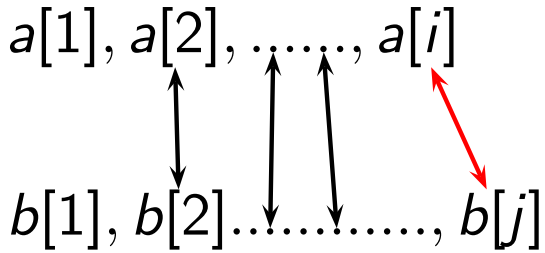
Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)



Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$ **oppure non** contiene $b[j]$.

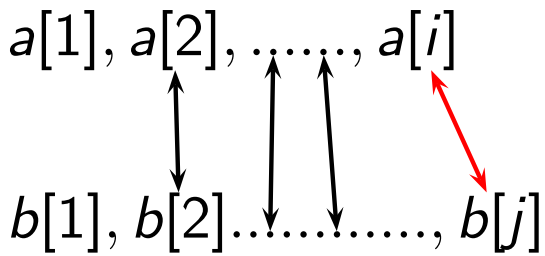
Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)



Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$ **oppure non** contiene $b[j]$.
In altre parole, una tale più lunga sottosequenza comune o è una più lunga sottosequenza comune di

$$a[1] \ldots a[i-1], \quad \text{e} \quad b[1] \ldots b[j]$$

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)

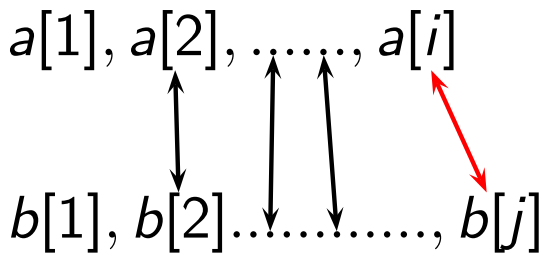


Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$ **oppure non** contiene $b[j]$.
In altre parole, una tale più lunga sottosequenza comune o è una più lunga sottosequenza comune di

$$a[1] \ldots a[i-1], \quad \text{e} \quad b[1] \ldots b[j] \qquad \text{oppure di}$$

$$a[1] \ldots a[i], \quad \text{e} \quad b[1] \ldots b[j-1].$$

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)



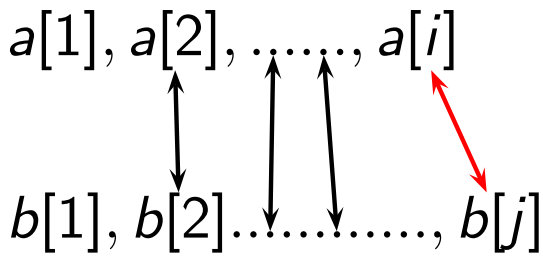
Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$ **oppure non** contiene $b[j]$.
In altre parole, una tale più lunga sottosequenza comune o è una più lunga sottosequenza comune di

$$a[1] \ldots a[i-1], \quad \text{e} \quad b[1] \ldots b[j] \qquad \text{oppure di}$$

$$a[1] \ldots a[i], \quad \text{e} \quad b[1] \ldots b[j-1].$$

E qual è delle due?

Caso 1: $a[i] \neq b[j]$. In tal caso, **non** è possibile che $a[i]$ e $b[j]$ siano **entrambi** presenti nella più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ (altrimenti la sottosequenza scelta non sarebbe nemmeno **comune** ad $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$!)



Segue che una più lunga sottosequenza comune di $a[1] \ldots a[i]$ e $b[1] \ldots b[j]$ o **non** contiene $a[i]$ **oppure non** contiene $b[j]$.
In altre parole, una tale più lunga sottosequenza comune o è una più lunga sottosequenza comune di

$$a[1] \ldots a[i-1], \quad \text{e} \quad b[1] \ldots b[j] \qquad \text{oppure di}$$

$$a[1] \ldots a[i], \quad \text{e} \quad b[1] \ldots b[j-1].$$

E qual è delle due? Non lo sappiamo a priori, ma visto che ne cerchiamo una più lunga, sicuramente varrà

$$c[i,j] = \max\{c[i-1,j], c[i,j-1]\}$$

Caso 2: $a[i] = b[j]$.

Caso 2: $a[i] = b[j]$. Sia $d[1] \ldots d[k]$ una più lunga sottosequenza comune di

$$a[1] \ldots a[i], \quad \text{e} \quad b[1] \ldots b[j],$$

Caso 2: $a[i] = b[j]$. Sia $d[1] \ldots d[k]$ una più lunga sottosequenza comune di

$$a[1] \ldots a[i], \quad \text{e} \quad b[1] \ldots b[j],$$

e quindi vale anche $c[i, j] = k$.

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$,

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$, allora appendendo alla fine di α il simbolo comune $a[i]$, otterremo una sottosequenza **comune** di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j]$$

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$, allora appendendo alla fine di α il simbolo comune $a[i]$, otterremo una sottosequenza **comune** di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j]$$

di lunghezza $\geq k + 1$,

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$, allora appendendo alla fine di α il simbolo comune $a[i]$, otterremo una sottosequenza **comune** di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j]$$

di lunghezza $\geq k+1$, contraddicendo il fatto che $c[i, j] = k$.

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$, allora appendendo alla fine di α il simbolo comune $a[i]$, otterremo una sottosequenza **comune** di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j]$$

di lunghezza $\geq k+1$, contraddicendo il fatto che $c[i, j] = k$. Abbiamo quindi provato che $c[i-1, j-1] = k-1$, ovvero

Caso 2: $a[i] = b[j]$. Sia $d[1] \dots d[k]$ una più lunga sottosequenza comune di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j],$$

e quindi vale anche $c[i, j] = k$. Allora affermiamo che $d[1], \dots, d[k-1] = \beta$ è una più lunga sottosequenza comune di

$$a[1] \dots a[i-1] \quad \text{e} \quad b[1] \dots b[j-1].$$

Infatti, se per assurdo esistesse una sequenza comune α di $a[1] \dots a[i-1]$ e $b[1] \dots b[j-1]$di lunghezza $\geq k$, allora appendendo alla fine di α il simbolo comune $a[i]$, otterremo una sottosequenza **comune** di

$$a[1] \dots a[i], \quad \text{e} \quad b[1] \dots b[j]$$

di lunghezza $\geq k+1$, contraddicendo il fatto che $c[i, j] = k$. Abbiamo quindi provato che $c[i-1, j-1] = k-1$, ovvero

$$c[i, j] = c[i-1, j-1] + 1.$$

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ \\ \end{cases}$$

I valori $c[i, j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1, j-1] + 1 & \text{se } i, j > 0 \text{ e } a[i] = b[j] \end{cases}$$

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

Un algoritmo di PD per il calcolo dei $c[i,j]$ basato sulle tecnica della memoization può essere il seguente:

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
```

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

Un algoritmo di PD per il calcolo dei $c[i,j]$ basato sulle tecnica della memoization può essere il seguente:

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
```

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

Un algoritmo di PD per il calcolo dei $c[i,j]$ basato sulle tecnica della memoization può essere il seguente:

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.         IF (a[i]==b[j]) {
5.         c(i,j)= Mem-Rec(i-1,j-1)+1
```

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

Un algoritmo di PD per il calcolo dei $c[i,j]$ basato sulle tecnica della memoization può essere il seguente:

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.         IF (a[i]==b[j]) {
5.         c(i,j)= Mem-Rec(i-1,j-1)+1
6.         } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
          }
      }
```

I valori $c[i,j]$ sono definiti dalla equazione di ricorrenza

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \text{ o se } j = 0, \\ c[i-1,j-1]+1 & \text{se } i,j > 0 \text{ e } a[i] = b[j] \\ \max\{c[i-1,j], c[i,j-1])\} & \text{se } i,j > 0 \text{ e } a[i] \neq b[j]. \end{cases}$$

Un algoritmo di PD per il calcolo dei $c[i,j]$ basato sulle tecnica della memoization può essere il seguente:

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.         IF (a[i]==b[j]) {
5.         c(i,j)= Mem-Rec(i-1,j-1)+1
6.         } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
          }
       }
7. RETURN (c(i,j))
```

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
5.        c(i,j)= Mem-Rec(i-1,j-1)+1
6.        } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
        }
     }
7. RETURN (c(i,j))
```

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
5.        c(i,j)= Mem-Rec(i-1,j-1)+1
6.        } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
          }
      }
7. RETURN (c(i,j))
```

A noi interessa il valore ritornato da `Mem-Rec(m,n)`.

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
5.        c(i,j)= Mem-Rec(i-1,j-1)+1
6.        } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
          }
      }
7. RETURN (c(i,j))
```

A noi interessa il valore ritornato da `Mem-Rec(m,n)`. Ogni entrata della tabella `c(m,n)` viene calcolata esattamente una sola volta,

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
5.        c(i,j)= Mem-Rec(i-1,j-1)+1
6.        } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
          }
      }
7. RETURN (c(i,j))
```

A noi interessa il valore ritornato da `Mem-Rec(m,n)`. Ogni entrata della tabella `c(m,n)` viene calcolata esattamente una sola volta, e per il suo calcolo l'algoritmo impiega tempo costante,

```
Mem-Rec(i,j) %fà uso di una tabella c(i,j)
1. IF((i==0)||(j==0)) {
2.    RETURN 0
3.    } ELSE IF (c(i,j) non è definito) {
4.        IF (a[i]==b[j]) {
5.        c(i,j)= Mem-Rec(i-1,j-1)+1
6.        } ELSE {c(i,j)= max(Mem-Rec(i-1,j), Mem-Rec(i,j-1))
         }
      }
7. RETURN (c(i,j))
```

A noi interessa il valore ritornato da `Mem-Rec(m,n)`. Ogni entrata della tabella `c(m,n)` viene calcolata esattamente una sola volta, e per il suo calcolo l'algoritmo impiega tempo costante, quindi l'algoritmo `Mem-Rec(m,n)` impiega tempo $O(mn)$.

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST,

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

		G	V	C	E	K	S	T
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3
G	5	0	1	2	2	3	3	3	3

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3
G	5	0	1	2	2	3	3	3	3
T	6	0	1	2	2	3	3	3	4

Supponiamo che le due sequenze siano a=GDVEGTA e b =GVCEKST, la tabella $c[\cdot,\cdot]$ risultante sarebbe la seguente:

			G	V	C	E	K	S	T
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
G	1	0	1	1	1	1	1	1	1
D	2	0	1	1	1	1	1	1	1
V	3	0	1	2	2	2	2	2	2
E	4	0	1	2	2	3	3	3	3
G	5	0	1	2	2	3	3	3	3
T	6	0	1	2	2	3	3	3	4
A	7	0	1	2	2	3	3	3	4

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b.

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

Possiamo usare la tabella $c[\cdot, \cdot]$ prima calcolata, ragionando nel modo seguente:

- Se $c[m, n] = 0$ allora ritorniamo la stringa vuota,

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

Possiamo usare la tabella $c[\cdot, \cdot]$ prima calcolata, ragionando nel modo seguente:

- Se $c[m, n] = 0$ allora ritorniamo la stringa vuota,
- altrimenti, se $c[m, n] = c[m-1, n]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m-1]$ e $b[1] \dots b[n]$,

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

Possiamo usare la tabella $c[\cdot, \cdot]$ prima calcolata, ragionando nel modo seguente:

- Se $c[m, n] = 0$ allora ritorniamo la stringa vuota,
- altrimenti, se $c[m, n] = c[m-1, n]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m-1]$ e $b[1] \dots b[n]$,
- se $c[m, n] = c[m, n-1]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m]$ e $b[1] \dots b[n-1]$,

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

Possiamo usare la tabella $c[\cdot, \cdot]$ prima calcolata, ragionando nel modo seguente:

- Se $c[m, n] = 0$ allora ritorniamo la stringa vuota,
- altrimenti, se $c[m, n] = c[m-1, n]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m-1]$ e $b[1] \dots b[n]$,
- se $c[m, n] = c[m, n-1]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m]$ e $b[1] \dots b[n-1]$,
- Se entrambe le condizioni di sopra sono false,

L'algoritmo appena visto calcola $c[m, n] = |LCS(a, b)|$ ovvero la **lunghezza** della più lunga sottosequenza comune ad a e b. Come calcolare la più lunga sottosequenza comune ad a e b?

Possiamo usare la tabella $c[\cdot, \cdot]$ prima calcolata, ragionando nel modo seguente:

- Se $c[m, n] = 0$ allora ritorniamo la stringa vuota,
- altrimenti, se $c[m, n] = c[m-1, n]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m-1]$ e $b[1] \dots b[n]$,
- se $c[m, n] = c[m, n-1]$, allora calcoliamo (ricorsivamente) la LCS tra $a[1] \dots a[m]$ e $b[1] \dots b[n-1]$,
- Se entrambe le condizioni di sopra sono false, allora vuol dire che $a[m] = b[n]$, di conseguenza l'algoritmo stampa $a[m]$ e ricorre su $a[1] \dots a[m-1]$ e $b[1] \dots b[n-1]$

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.       IF(c[m,n]==c[m-1,n]) {
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.      IF(c[m,n]==c[m-1,n]) {
5.        LCS_print(a(m-1),b,c)
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.       IF(c[m,n]==c[m-1,n]) {
5.          LCS_print(a(m-1),b,c)
6.       } ELSE {IF (c[m,n]=c[m,n-1]) {
7.          LCS_print(a,b(n-1),c)
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.       IF(c[m,n]==c[m-1,n]) {
5.          LCS_print(a(m-1),b,c)
6.       } ELSE {IF (c[m,n]=c[m,n-1]) {
7.          LCS_print(a,b(n-1),c)
8.       } ELSE {
9.        LCS_print(a(m-1),b(n-1),c)
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.       IF(c[m,n]==c[m-1,n]) {
5.          LCS_print(a(m-1),b,c)
6.       } ELSE {IF (c[m,n]=c[m,n-1]) {
7.          LCS_print(a,b(n-1),c)
8.       } ELSE {
9.        LCS_print(a(m-1),b(n-1),c)
10.       print(a[m])
          }
      }
```

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.      IF(c[m,n]==c[m-1,n]) {
5.        LCS_print(a(m-1),b,c)
6.      } ELSE {IF (c[m,n]=c[m,n-1]) {
7.        LCS_print(a,b(n-1),c)
8.      } ELSE {
9.       LCS_print(a(m-1),b(n-1),c)
10.      print(a[m])
         }
     }
```

L' algoritmo `LCS_print(a,b,c)` ad ogni chiamata ricorsiva decrementa n, oppure decrementa m, o addirittura entrambi.

```
LCS_print(a,b,c)    % prende in input le sequenze a,b e la matrice
c[,]
1. IF(c[m,n]==0) {
2.    RETURN ()
3.    } ELSE {
4.      IF(c[m,n]==c[m-1,n]) {
5.        LCS_print(a(m-1),b,c)
6.      } ELSE {IF (c[m,n]=c[m,n-1]) {
7.        LCS_print(a,b(n-1),c)
8.      } ELSE {
9.       LCS_print(a(m-1),b(n-1),c)
10.      print(a[m])
         }
     }
```

L' algoritmo `LCS_print(a,b,c)` ad ogni chiamata ricorsiva decrementa n, oppure decrementa m, o addirittura entrambi. Di conseguenza, termina in tempo $O(n + m)$

Altro Problema: Distanza di Edit

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri,

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri,

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri `albero` in `labbro`.

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

albero

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$\texttt{albero} \xrightarrow{\text{Cancella a}}$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero}$$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero} \xrightarrow{\text{Inserisci a}}$$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero} \xrightarrow{\text{Inserisci a}} \texttt{labero}$$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero} \xrightarrow{\text{Inserisci a}} \texttt{labero} \xrightarrow{\text{Sostituisci e}}$$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero} \xrightarrow{\text{Inserisci a}} \texttt{labero} \xrightarrow{\text{Sostituisci e}} \texttt{labbro}$$

Altro Problema: Distanza di Edit

Date due sequenze di lettere $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, la distanza di edit tra s e t (denotata con $\texttt{dist}(s, t)$) è il **minimo** numero di inserzioni di caratteri, cancellazioni di caratteri, o sostituzioni di caratteri necessarie per trasformare s in t.

Es.: vogliamo trasformare la sequenza di caratteri albero in labbro. Potremmo procedere nel seguente modo:

$$\texttt{albero} \xrightarrow{\text{Cancella a}} \texttt{lbero} \xrightarrow{\text{Inserisci a}} \texttt{labero} \xrightarrow{\text{Sostituisci e}} \texttt{labbro}$$

Si può far vedere che non si può far meglio, quindi in questo caso $\texttt{dist}(\texttt{albero}, \texttt{labbro}) = 3$

Applicazioni

- Correzione automatica di parole

Applicazioni

- Correzione automatica di parole
- Linguistica (distanza linguistica)

Applicazioni

- Correzione automatica di parole
- Linguistica (distanza linguistica)
- Correzione di errori in OCR

Applicazioni

- Correzione automatica di parole
- Linguistica (distanza linguistica)
- Correzione di errori in OCR
- Biologia Molecolare

Applicazioni

- Correzione automatica di parole
- Linguistica (distanza linguistica)
- Correzione di errori in OCR
- Biologia Molecolare
- ....

Vogliamo risolvere il problema di calcolare la distanza di Edit $\texttt{dist}(s, t)$ tra due sequenze $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$ utilizzando la tecnica Programmazione Dinamica

Vogliamo risolvere il problema di calcolare la distanza di Edit $\texttt{dist}(s, t)$ tra due sequenze $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$ utilizzando la tecnica Programmazione Dinamica

Primo passo: formulare il problema in termini ricorsivi.

Vogliamo risolvere il problema di calcolare la distanza di Edit $\texttt{dist}(s, t)$ tra due sequenze $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$ utilizzando la tecnica Programmazione Dinamica

Primo passo: formulare il problema in termini ricorsivi.

E chi sono i sottoproblemi del problema di calcolare $\texttt{dist}(s, t)$?

Vogliamo risolvere il problema di calcolare la distanza di Edit $\texttt{dist}(s, t)$ tra due sequenze $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$ utilizzando la tecnica Programmazione Dinamica

Primo passo: formulare il problema in termini ricorsivi.

E chi sono i sottoproblemi del problema di calcolare $\texttt{dist}(s, t)$?

Essi corrispondono naturalmente al calcolo delle distanze di edit tra *sottosequenze* di s e t di lunghezze inferiori rispetto a quelle di s e t.

Date $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, sia $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$, la distanza di edit tra le **sottosequenze** $s[1] \dots s[i]$, e $t[1] \dots t[j]$.

Date $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, sia $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$, la distanza di edit tra le **sottosequenze** $s[1] \dots s[i]$, e $t[1] \dots t[j]$.

La Programmazione Dinamica ci suggerisce di determinare una equazione di ricorrenza per le quantità $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$.

Date $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, sia $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$, la distanza di edit tra le **sottosequenze** $s[1] \dots s[i]$, e $t[1] \dots t[j]$.

La Programmazione Dinamica ci suggerisce di determinare una equazione di ricorrenza per le quantità $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$.

Vogliamo calcolare il minimo numero di operazioni $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$ per trasformare $s[1] \dots s[i]$ in $t[1] \dots t[j]$.

Date $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, sia $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$, la distanza di edit tra le **sottosequenze** $s[1] \dots s[i]$, e $t[1] \dots t[j]$.

La Programmazione Dinamica ci suggerisce di determinare una equazione di ricorrenza per le quantità $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$.

Vogliamo calcolare il minimo numero di operazioni $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$ per trasformare $s[1] \dots s[i]$ in $t[1] \dots t[j]$.

Iniziamo dalla fine: come può l'ultima lettera di $s[1] \dots s[i]$, ovvero $s[i]$, essere trasformata in $t[j]$, l'ultima lettera di $t[1] \dots t[j]$?

Date $s = s[1] \dots s[m]$ e $t = t[1] \dots [n]$, sia $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$, la distanza di edit tra le **sottosequenze** $s[1] \dots s[i]$, e $t[1] \dots t[j]$.

La Programmazione Dinamica ci suggerisce di determinare una equazione di ricorrenza per le quantità $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$, per $1 \leq i < m$ e $1 \leq j < n$.

Vogliamo calcolare il minimo numero di operazioni $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j])$ per trasformare $s[1] \dots s[i]$ in $t[1] \dots t[j]$.

Iniziamo dalla fine: come può l'ultima lettera di $s[1] \dots s[i]$, ovvero $s[i]$, essere trasformata in $t[j]$, l'ultima lettera di $t[1] \dots t[j]$?

Possiamo usare una delle tre seguenti operazioni:

- **Sostituire** $s[i]$ con $t[j]$.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$,

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a

$$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \dots s[i-1]$ in $t[1] \dots t[j]$.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \dots s[i-1]$ in $t[1] \dots t[j]$.
  Ciò richiede $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$ operazioni.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \dots s[i-1]$ in $t[1] \dots t[j]$.
  Ciò richiede $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$$

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \dots s[i-1]$ in $t[1] \dots t[j]$.
  Ciò richiede $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$$
- **Inserire** $t[j]$ alla fine di $s[1] \dots s[i]$.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \ldots s[i-1]$ in $t[1] \ldots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \ldots s[i-1]$ in $t[1] \ldots t[j]$.
  Ciò richiede $\texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j])$$
- **Inserire** $t[j]$ alla fine di $s[1] \ldots s[i]$.
  Ciò ci lascia poi con il problema di trasformare $s[1] \ldots s[i]$ in $t[1] \ldots t[j-1]$, che richiede un numero di operazioni pari a $\texttt{dist}(s[1] \ldots s[i], t[1] \ldots t[j-1])$.

- **Sostituire** $s[i]$ con $t[j]$.
  Ciò ci lascia poi con il problema di trasformare la sottosequenza $s[1] \dots s[i-1]$ in $t[1] \dots t[j-1]$, che richiederà ulteriori $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j-1])$$
- **Cancellare** $s[i]$ e poi trasformare $s[1] \dots s[i-1]$ in $t[1] \dots t[j]$.
  Ciò richiede $\texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$ operazioni. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i-1], t[1] \dots t[j])$$
- **Inserire** $t[j]$ alla fine di $s[1] \dots s[i]$.
  Ciò ci lascia poi con il problema di trasformare $s[1] \dots s[i]$ in $t[1] \dots t[j-1]$, che richiede un numero di operazioni pari a $\texttt{dist}(s[1] \dots s[i], t[1] \dots t[j-1])$. In totale, useremo un numero di operazioni pari a
  $$1 + \texttt{dist}(s[1] \dots s[i], t[1] \dots t[j-1])$$

Vi è un caso speciale da considerare. Se $s[i] = t[j]$, allora nel primo passo precedentemente considerato non occorre trasformare $s[i]$ in $t[j]$, e quindi basteranno $\texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j-1])$ operazioni per trasformare $s[1] \ldots s[i]$ in $t[1] \ldots t[j]$.

Vi è un caso speciale da considerare. Se $s[i] = t[j]$, allora nel primo passo precedentemente considerato non occorre trasformare $s[i]$ in $t[j]$, e quindi basteranno $\texttt{dist}(s[1] \ldots s[i-1], t[1] \ldots t[j-1])$ operazioni per trasformare $s[1] \ldots s[i]$ in $t[1] \ldots t[j]$.

Per trattare questo caso, definiamo la quantità $\texttt{diff}(x, y) = 1$ se $x \neq y$, 0 altrimenti.

Abbiamo quindi l' equazione di ricorrenza per $\texttt{dist}(s[1] \ldots s[i], t[1] \ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

Abbiamo quindi l' equazione di ricorrenza per $\texttt{dist}(s[1] \ldots s[i], t[1] \ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\texttt{dist}(s[1] \ldots s[i], t[1] \ldots t[j])$$
$$= \min$$

Abbiamo quindi l' equazione di ricorrenza per
$\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$$
$$= \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]),$$

Abbiamo quindi l' equazione di ricorrenza per $\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\begin{aligned}
&\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j]) \\
&\qquad = \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]), \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1,
\end{aligned}$$

Abbiamo quindi l' equazione di ricorrenza per
$\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\begin{aligned}
&\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j]) \\
&\qquad = \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]), \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1, \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j-1]) + 1\}
\end{aligned}$$

Abbiamo quindi l' equazione di ricorrenza per
$\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\begin{aligned}
&\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j]) \\
&\qquad = \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]), \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1, \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j-1]) + 1\}
\end{aligned}$$

Per i casi base $i = 0 = j$, effettueremo la ovvia assunzione che
$s[0] = t[0] =$*sequenza vuota*,

Abbiamo quindi l' equazione di ricorrenza per
$\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])$ che cercavamo, $\forall\, i, j \geq 1$:

$$\begin{aligned}
&\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j]) \\
&\qquad = \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]), \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1, \\
&\qquad\qquad \texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j-1]) + 1\}
\end{aligned}$$

Per i casi base $i = 0 = j$, effettueremo la ovvia assunzione che
$s[0] = t[0] =$*sequenza vuota*, e che quindi

$$\texttt{dist}(s[0], t[1]\ldots t[j]) = j, \texttt{dist}(s[1]\ldots s[i], t[0]) = i, \quad \text{e } \forall\, i, j \geq 1.$$

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s, t)   %Fà uso di una matrice dist[.,.]
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i          % (inizializzazione prima colonna)
   }
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i          % (inizializzazione prima colonna)
   }
3.   FOR (j= 0, j<n+1, j=j+1) {
4.     dist[0,j]= j          % (inizializzazione prima riga)
   }
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i              % (inizializzazione prima colonna)
   }
3.   FOR (j= 0, j<n+1, j=j+1) {
4.     dist[0,j]= j              % (inizializzazione prima riga)
   }
5.   FOR (i=1, i<m+1, i=i+1) {
6.     FOR (j= 1, j<n+1, j=j+1) {
7.       IF (s[i]==t[j]) {          % (calcolo di dist[i,j] )
8.           dist[i,j]=min(dist[i-1,j-1], dist[i-1,j]+1,
                           dist[i,j-1]+1)
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i              % (inizializzazione prima colonna)
   }
3.   FOR (j= 0, j<n+1, j=j+1) {
4.     dist[0,j]= j              % (inizializzazione prima riga)
   }
5.   FOR (i=1, i<m+1, i=i+1) {
6.     FOR (j= 1, j<n+1, j=j+1) {
7.       IF (s[i]==t[j]) {              % (calcolo di dist[i,j] )
8.           dist[i,j]=min(dist[i-1,j-1], dist[i-1,j]+1,
                           dist[i,j-1]+1)
9.         } ELSE { dist[i,j]= min(dist[i-1,j-1]+1,
                                   dist[i-1,j]+1,
                                   dist[i,j-1]+1)
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i          % (inizializzazione prima colonna)
   }
3.   FOR (j= 0, j<n+1, j=j+1) {
4.     dist[0,j]= j          % (inizializzazione prima riga)
   }
5.   FOR (i=1, i<m+1, i=i+1) {
6.     FOR (j= 1, j<n+1, j=j+1) {
7.       IF (s[i]==t[j]) {          % (calcolo di dist[i,j] )
8.           dist[i,j]=min(dist[i-1,j-1], dist[i-1,j]+1,
                           dist[i,j-1]+1)
9.         } ELSE { dist[i,j]= min(dist[i-1,j-1]+1,
                                   dist[i-1,j]+1,
                                   dist[i,j-1]+1)
       }
     }
10. RETURN dist[m,n]
```

L'algoritmo di Programmazione Dinamica (nella versione iterativa) per il calcolo di $\texttt{dist}(s,t) = \texttt{dist}(s[1]\ldots s[m], t[1]\ldots t[n])$ è

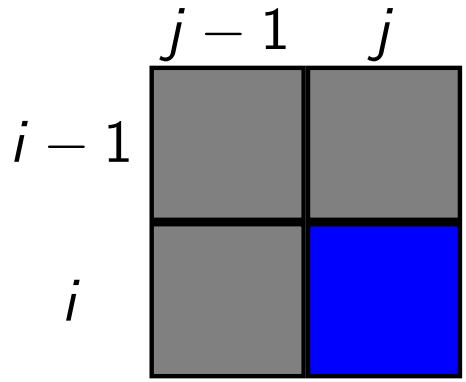
```
Edit_distanza(s,t)   %Fà uso di una matrice dist[.,.]
1. FOR (i=0, i<m+1, i=i+1) {
2.     dist[i,0]= i              % (inizializzazione prima colonna)
   }
3.   FOR (j= 0, j<n+1, j=j+1) {
4.     dist[0,j]= j              % (inizializzazione prima riga)
   }
5.   FOR (i=1, i<m+1, i=i+1) {
6.     FOR (j= 1, j<n+1, j=j+1) {
7.       IF (s[i]==t[j]) {            % (calcolo di dist[i,j] )
8.           dist[i,j]=min(dist[i-1,j-1], dist[i-1,j]+1,
                           dist[i,j-1]+1)
9.         } ELSE { dist[i,j]= min(dist[i-1,j-1]+1,
                                   dist[i-1,j]+1,
                                   dist[i,j-1]+1)
       }
     }
10. RETURN dist[m,n]
```

Complessità $O(nm)$

Vediamo un esempio di applicazione dell'algoritmo appena visto.
Sia a =`presto`, b =`peseta`.

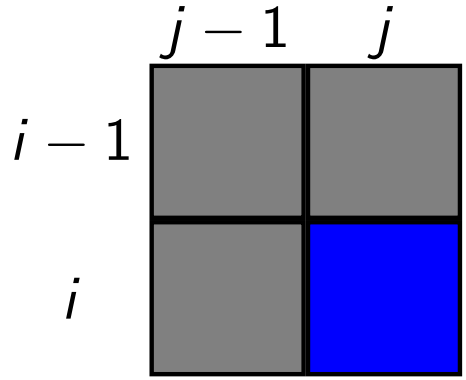
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.



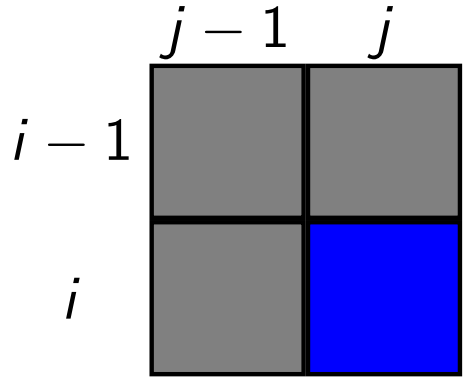
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6



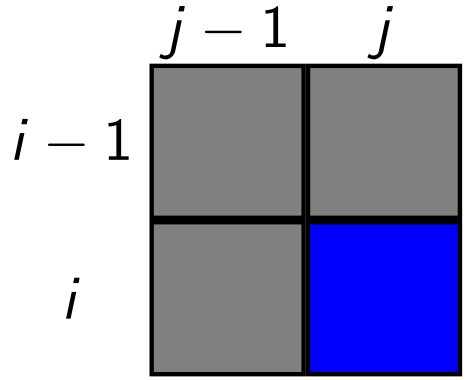
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p							
2	r							
3	e							
4	s							
5	t							
6	o							



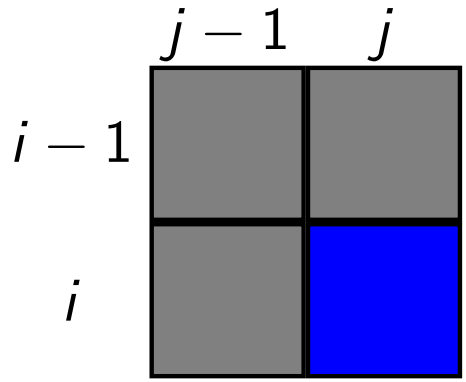
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r							
3	e							
4	s							
5	t							
6	o							



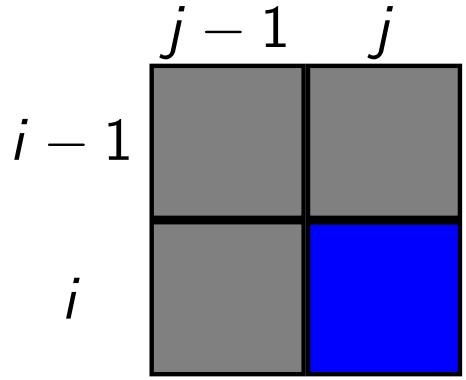
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r	2	1	1	2	3	4	5
3	e							
4	s							
5	t							
6	o							



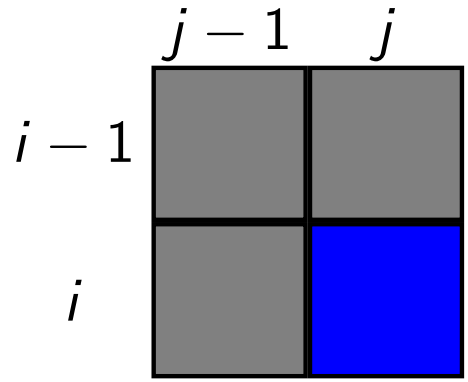
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r	2	1	1	2	3	4	5
3	e	3	2	1	2	2	3	4
4	s							
5	t							
6	o							



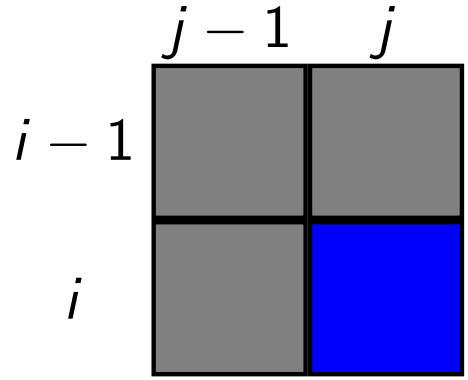
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r	2	1	1	2	3	4	5
3	e	3	2	1	2	2	3	4
4	s	4	3	2	1	2	3	4
5	t							
6	o							



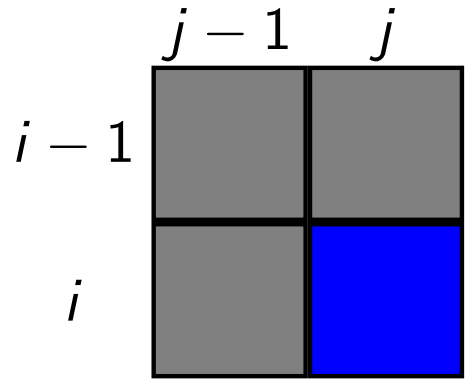
Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r	2	1	1	2	3	4	5
3	e	3	2	1	2	2	3	4
4	s	4	3	2	1	2	3	4
5	t	5	4	3	2	2	2	3
6	o							



Vediamo un esempio di applicazione dell'algoritmo appena visto. Sia a =`presto`, b =`peseta`. Il quadrato colorato ricorda che ogni nuova entrata della matrice (in blu) viene calcolato sulla base dei valori contenuti nelle tre entrate in grigio, precedentemente calcolate.

		0	1	2	3	4	5	6
			p	e	s	e	t	a
0		0	1	2	3	4	5	6
1	p	1	0	1	2	3	4	5
2	r	2	1	1	2	3	4	5
3	e	3	2	1	2	2	3	4
4	s	4	3	2	1	2	3	4
5	t	5	4	3	2	2	2	3
6	o	6	5	4	3	3	3	3



Esercizio: Data

$$\begin{aligned}&\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j])\\ &\qquad = \min\{\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]),\\ &\qquad\qquad \texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1,\\ &\qquad\qquad \texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j-1]) + 1\}\end{aligned}$$

e

Esercizio: Data

$$\begin{aligned}&\texttt{dist}(s[1]\dots s[i], t[1]\dots t[j])\\ &\qquad = \min\{\texttt{dist}(s[1]\dots s[i-1], t[1]\dots t[j-1]) + \texttt{diff}(s[i], t[j]),\\ &\qquad\qquad \texttt{dist}(s[1]\dots s[i-1], t[1]\dots t[j]) + 1,\\ &\qquad\qquad \texttt{dist}(s[1]\dots s[i], t[1]\dots t[j-1]) + 1\}\end{aligned}$$

e

$$\texttt{dist}(s[0], t[1]\dots t[j]) = j, \texttt{dist}(s[1]\dots s[i], t[0]) = i, \quad \text{e} \ \forall\, i, j \geq 1$$

Esercizio: Data

$$\begin{aligned}\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j]) \\ = \min\{&\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j-1]) + \texttt{diff}(s[i], t[j]), \\ &\texttt{dist}(s[1]\ldots s[i-1], t[1]\ldots t[j]) + 1, \\ &\texttt{dist}(s[1]\ldots s[i], t[1]\ldots t[j-1]) + 1\}\end{aligned}$$

e

$$\texttt{dist}(s[0], t[1]\ldots t[j]) = j, \texttt{dist}(s[1]\ldots s[i], t[0]) = i, \quad \text{e} \ \forall\, i, j \geq 1$$

scrivere l'algoritmo di PD che fà uso della memoization.