# PROGRAMMAZIONE DINAMICA: LONGEST COMMON SUBSEQUENCE

[Cormen] cap. 15

Sezione 15.4

[Deme, seconda edizione] cap. 11

Sezione 11.3

# Definizioni

Data una **sequenza S: $a_1, \ldots, a_m$**

una **sottosequenza** di S è una **qualsiasi sequenza ottenuta** da S **togliendo alcuni** (o nessun) **elementi**.

La sottosequenza deve rispettare **l'ordine** degli elementi della sequenza originale.

Quindi $a_1$, $a_2$, $a_{10}$, $a_m$ è una sottosequenza di S

mentre $a_2$, $a_1$, $a_5$ non lo è, perché $a_1$ e $a_2$ non sono nel giusto ordine

**S** stessa **e la sequenza vuota** sono **sottosequenze di S**.

S: **AGC**C**G**GA**T**CG**A**GT

Sottosequenza: **AGCGTA**

# Problema

Date **due sequenze S1 ed S2**, trovare **la più lunga sequenza S3 che è sottosequenza sia di S1 che di S2**, cioè trovare una **sottosequenza comune di lunghezza massima**. Notazione:
**S3 = lcs(S1, S2)**

Se vi sono più sottosequenze comuni di lunghezza massima, trovarne una, non importa quale.

Esempi del problema:

- biologia: trovare la più lunga sottosequenza comune a due **sequenze di DNA**
- sicurezza informatica: individuare, in log costituiti da una sequenza di comandi, quelle sottosequenze comuni che indicano la presenza di un **possibile attacco al sistema**
- È alla base di **diff**, il software (Unix) per trovare le differenze tra file

# Esempio

Date le due sequenze di DNA:

**AGC**C**G**GA**T**CG**A**GT

TC**AG**TA**CGT**T**A**

una sottosequenza comune di lunghezza massima è:

**AGCGTA**

Un'altra sottosequenza comune di lunghezza massima, per le stesse due sequenze, è:

**AGTCGA**

Infatti:

**AG**CCGGA**TCGA**GT

TC**AGT**A**CG**TT**A**

# Algoritmo a forza bruta

Siano S1 e S2 le due sequenze.

**Algoritmo:**

Si **generano tutte le possibili sottosequenze** di S1 e si controlla se sono sottosequenze di S2, tenendo traccia della più lunga trovata.

Tuttavia, tale algoritmo ha un **costo** troppo elevato. Infatti, il numero di sottosequenze di S1 è **esponenziale** nella lunghezza n di S1:

$a_1$ può esserci o no (2 possibilità)

$a_2$ può esserci o no ($2^1$x2 possibilità)

…

$a_m$ può esserci o no ($2^{m-1}$x2 possibilità)

# Algoritmo a forza bruta - II

In più, **per ogni sottosequenza** di S1 ci vuole un tempo n (lunghezza di S2) per **verificare se** la prima **è sottosequenza** di S2, quindi la complessità totale è

**$O(n2^m)$**

Possiamo fare di meglio?

Sì, sfruttando la **programmazione dinamica**.

# Sottostruttura Ottima

Ragioniamo sul problema. Siano

S1: $a_1, \ldots, a_m$

S2: $b_1, \ldots, b_n$

E S3: $c_1, \ldots, c_k$ sia **S3 = lcs(S1, S2)**

E ragioniamo sugli ultimi elementi di ciascuna sequenza.

Abbiamo **2 casi** possibili:

Caso 1: $\mathbf{a_m = b_n}$

Caso 2: $\mathbf{a_m \neq b_n}$

Analizziamo i 2 casi.

# Caso 1: $a_m = b_n$

Ad esempio ‘BAC**A**’, ‘ATCB**A**’

In questo caso, è abbastanza facile capire che **$a_m$** (o equiv. $b_n$) **sarà contenuto in S3**, e **occuperà l’ultima posizione** di S3 (non ci possono essere elementi comuni dopo l’ultimo elemento di ciascuna sequenza). Quindi:

$$c_k = a_m = b_n$$

In più, poiché gli ultimi elementi delle sequenze iniziali già appartengono alla lcs (cioè li abbiamo già «accoppiati»), possiamo **non considerarli per il confronto con il resto delle sequenze**. Quindi possiamo dire che

$$S3 = lcs(S1-\{a_m\}, S2-\{b_n\}) + \{a_m\}$$

o meglio (usando i prefissi dove $S_{l-1}$ sono i primi l-1 elementi di S)

$$S3 = lcs(S1_{m-1}, S2_{n-1}) + \{a_m\}$$

# Caso 2: $a_m \neq b_n$

Ad esempio 'BACA', 'ATCBAB' o 'BACT', 'ATCBZ'

In questo caso, possiamo dire che **$a_m$, o $b_n$ (o entrambi) non saranno contenuti in S3**.

Consideriamo il caso in cui sia $a_m$ a non essere contenuta in S3. Possiamo «toglierla» dal confronto, e dire che

**S3 = lcs(S1-{$a_m$}, S2)**

Il caso in cui sia $b_n$ a non essere incluso è (speculare e) dato da

**S3 = lcs(S1, S2-{$b_n$})**

(il caso in cui entrambi non siano compresi può essere ottenuto da un passaggio ulteriore di questo ragionamento, quindi non lo consideriamo per ora).

Ma come faccio a sapere **quale delle due versioni di S3 usare?**

Semplice. Sto cercando la **Longest**CS, quindi

**S3 è la più lunga tra lcs(S1-{$a_m$}, S2) e lcs(S1, S2-{$b_n$})**

# Formula ricorsiva per LCS

$$LCS(S1, S2) = \begin{cases} \emptyset & se\ m = 0\ o\ n = 0 \\ LCS(S1_{m-1}, S2_{n-1}) + a_m & se\ a_m = b_n \\ \max\big(LCS(S1_{m-1}, S2_n), LCS(S1_m, S2_{n-1})\big) & altrimenti \end{cases}$$

# Sottoproblemi

Quindi, il problema della lcs gode della proprietà della **sottostruttura ottima**.

Abbiamo visto come, **per calcolare lcs(S1, S2)**, ci basti conoscere le soluzioni dei sottoproblemi

**lcs($S1_{m-1}$, $S2_{n-1}$), lcs($S1_{m-1}$, S2), lcs(S1, $S2_{n-1}$)** e $a_m$ e $b_n$.

Possiamo quindi applicare un algoritmo di **programmazione dinamica** per risolvere il problema della lcs.

Per risolvere il problema, dobbiamo ancora definire

- strutture di **input** e **output** del problema
- una struttura per la **memoizzazione** e
- un **algoritmo** per **popolarla** (basato sulla sottostruttura ottimale)

# Rappresentare le sequenze

Prima di tutto definiamo un modo per rappresentare efficientemente le sequenze e la sottosequenza comune.

È facile immaginare che **S1 ed S2 vengano rappresentate tramite vettori** (ad es. vettori di char)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | A | G | C | C | G | G | A | T | C | G | A | G | T |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| b | T | C | A | G | T | A | C | G | T | T | A |

**Nota:** essendo vettori, **gli indici che prima erano 1...n diventano 0..n-1**. Non vi confondete.

# Definiamo la struttura per la memoizzazione

Dobbiamo memoizzare la soluzione di parecchi sottoproblemi. Praticamente tutti quelli che si ottengono dalla **combinazione di tutti i prefissi** di S1 (o **del vettore a**) **con tutti i prefissi di** S2 (**vettore b**).

Usiamo una **matrice**, chiamandola **LCS**, di **m+1 righe e n+1 colonne**.

La casella **LCS[i, j] contiene la più lunga sottosequenza comune dei due segmenti iniziali** di lunghezze rispettive i e j, cioè **a[0 .. i-1]** e **b[0 .. j-1]**, cioè:

**LCS[i, j] = lcs(a[0 .. i-1], b[0 .. j-1])**

Perché il +1 in righe e colonne?

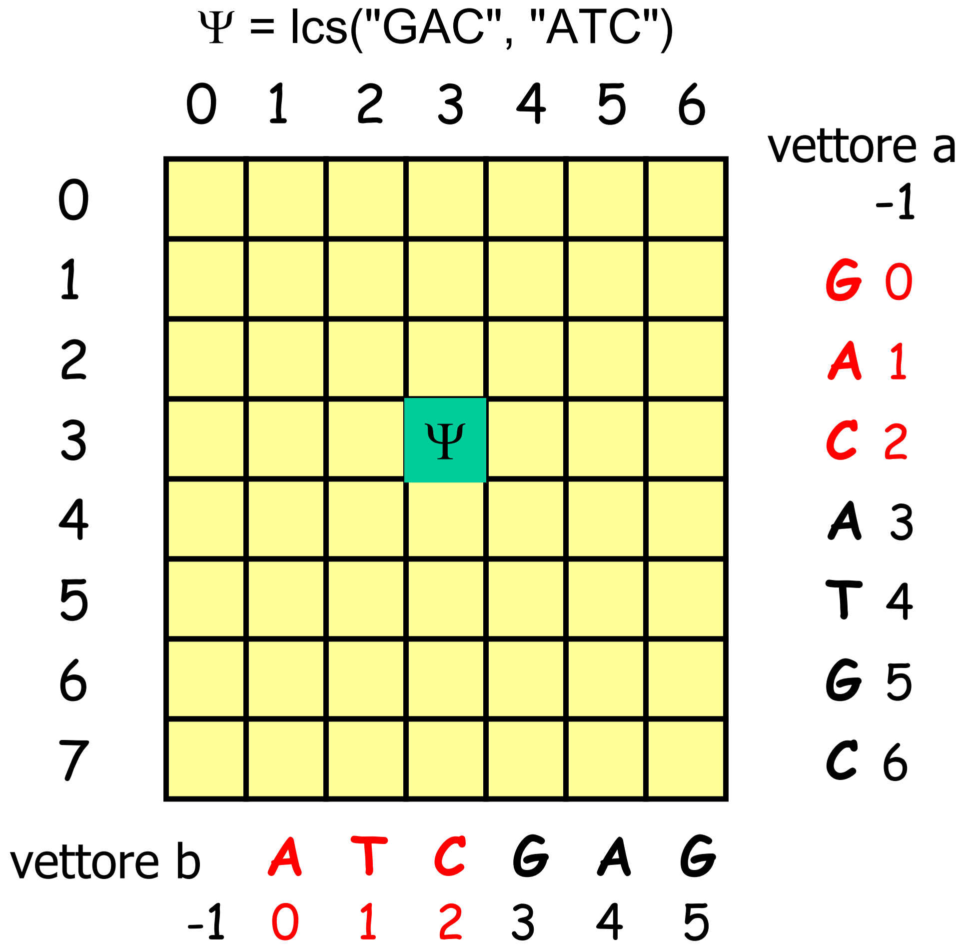
La **casella 0-esima** contiene la **più lunga sottosequenza dei prefissi vuoti** (con i e/o j = 0).

# Esempio: lcs("BACATBA", "ATCBAB")

$\Psi$ = lcs("BAC", "ATCB")

| vett. b | | -1 | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|---|---|
| vett. a | | | A | T | C | B | A | B | |
| -1 | | | | | | | | | 0 |
| 0 | B | | | | | | | | 1 |
| 1 | A | | | | | | | | 2 |
| 2 | C | | | | | $\Psi$ | | | 3 |
| 3 | A | | | | | | | | 4 |
| 4 | T | | | | | | | | 5 |
| 5 | B | | | | | | | | 6 |
| 6 | A | | | | | | | | 7 |
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | matrice LCS |

# Esempio: lcs('GACATGC', 'ATCGAG') (disegno alternativo)



# Inizializziamo la matrice LCS

Partiamo dai casi base. Se una delle due (sotto)sequenze è **vuota**, la **lcs** delle due **è** ovviamente **vuota**. Quindi

**LCS[0, j] = []** **per $0 \leq j \leq n$**

**LCS[i, 0] = []** **per $0 \leq i \leq m$**

# Esempio: lcs("BACATBA", "ATCBAB")

| vett. a \ vett. b | | -1 | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|---|---|
| | | | A | T | C | B | A | B | |
| -1 | | -- | -- | -- | -- | -- | -- | -- | 0 |
| 0 | B | -- | | | | | | | 1 |
| 1 | A | -- | | | | | | | 2 |
| 2 | C | -- | | | | | | | 3 |
| 3 | A | -- | | | | | | | 4 |
| 4 | T | -- | | | | | | | 5 |
| 5 | B | -- | | | | | | | 6 |
| 6 | A | -- | | | | | | | 7 |
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | matrice LCS |

# Esempio: lcs('GACATGC', 'ATCGAG')

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | vettore a |
|---|---|---|---|---|---|---|---|---|
| 0 | - | - | - | - | - | - | - | -1 |
| 1 | - | | | | | | | G 0 |
| 2 | - | | | | | | | A 1 |
| 3 | - | | | | | | | C 2 |
| 4 | - | | | | | | | A 3 |
| 5 | - | | | | | | | T 4 |
| 6 | - | | | | | | | G 5 |
| 7 | - | | | | | | | C 6 |
| vettore b | -1 | A 0 | T 1 | C 2 | G 3 | A 4 | G 5 | |

# Popolamento della matrice LCS

Supponendo di aver valorizzato **LCS[i, j-1], LCS[i-1, j] e LCS[i-1, j-1]** dobbiamo **valorizzare LCS[i, j]**.

Dalla definizione della sottostruttura ottimale, abbiamo che

$a_m = b_n \Rightarrow lcs(S1, S2) = lcs(S1_{m-1}, S2_{n-1}) + a_m$

che diventa

**if a[i-1] == b[j-1] then LCS[i, j] <- LCS[i-1, j-1] + a[i-1]**

e

$a_m \neq b_n \Rightarrow lcs(S1, S2) = max\_lunghezza(lcs(S1_{m-1}, S2), lcs(S1, S2_{n-1}))$

che diventa

**if a[i-1] != b[j-1] then LCS[i, j] <- max_lunghezza(LCS[i-1, j], LCS[i, j-1])**

# Algoritmo lcs – programmazione dinamica

```
lcs (a, b, m, n)
LCS <- nuova matrice di dimensioni m+1 x n+1
for i = 0..m LCS[i, 0] <- [] //inizializzazione casi base
for j = 0..n LCS[0, j] <- [] //inizializzazione casi base
for i = 1..m
  for j = 1..n
    if a[i-1] == b[j-1] then LCS[i, j] <- LCS[i-1, j-1] + a[i-1]
    else LCS[i, j] <- max_lunghezza(LCS[i-1, j], LCS[i, j-1])
return LCS[m, n]
```

# Ottimizzazioni

Notiamo che dobbiamo fare spesso un **controllo sulla lunghezza** delle lcs memoizzate (max_lunghezza…).

Per ottimizzare tale operazione, definiamo una nuova **matrice L** che mantenga in L[i,j] **la lunghezza della lcs memoizzata in LCS[i,j]**.

# Algoritmo lcs – matrice L

```
lcs (a, b, m, n)
LCS <- nuova matrice di dimensioni m+1 x n+1
L <- nuova matrice di dimensioni m+1 x n+1
for i = 0..m LCS[i, 0] <- []; L[i, 0] <- 0
for j = 0..n LCS[0, j] <- []; L[0, j] <- 0
for i = 1..m
  for j = 1..n
    if a[i-1] == b[j-1] then
      LCS[i, j] <- LCS[i-1, j-1] + a[i-1]
      L[i, j] <- L[i-1, j-1] + 1
    else if L[i-1,j] > L[i,j-1]
      LCS[i, j] <- LCS[i-1, j]
      L[i,j] <- L[i-1, j]
    else
       LCS[i, j] <- LCS[i, j-1]
       L[i, j] <- L[i, j-1]
return LCS[m, n]
```

# Ottimizzazioni - II

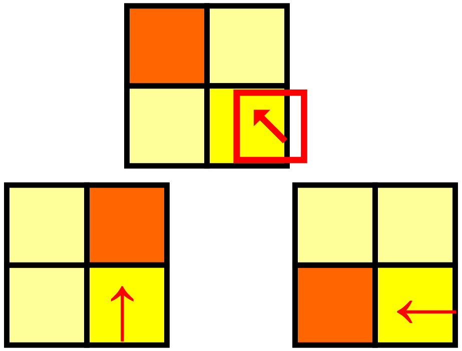
Notiamo che dobbiamo fare spesso un **controllo sulla lunghezza** delle lcs memoizzate (max_lunghezza…).

Per ottimizzare tale operazione, definiamo una nuova **matrice L** che mantenga in L[i,j] **la lunghezza della lcs memoizzata in LCS[i,j]**.

Ogni volta, LCS[i,j] è uguale a

**LCS[i-1,j-1]** + **a[i-1]** o

**LCS[i-1,j]** o **LCS[i,j-1]**



Possiamo quindi usare semplicemente **le frecce** (o meglio, una loro **codifica**) per indicare come stiamo costruendo LCS[i,j], senza dover ogni volta copiare tutto l'elemento precedente

# Algoritmo – Frecce

```
lcs (a, b, m, n)
LCS <- nuova matrice di dimensioni m+1 x n+1
L <- nuova matrice di dimensioni m+1 x n+1
for i = 0..m LCS[i, 0] <- [], L[i, 0] <- 0
for j = 0..n LCS[0, j] <- [], L[0, j] <- 0
for i = 1..m
  for j = 1..n
    if a[i-1] == b[j-1] then
      LCS[i, j] <- ↖
      L[i, j] <- L[i-1, j-1] + 1
    else if L[i-1,j] > L[i,j-1]
      LCS[i, j] <- ↑
      L[i,j] <- L[i-1, j]
    else
      LCS[i, j] <- ←
      L[i, j] <- L[i, j-1]
return LCS[m, n]
```

**ATTENZIONE:** prima ogni casella della matrice LCS conteneva una sequenza di caratteri, ora contiene una sola freccia

# Ricostruzione della soluzione

In questa maniera, LCS[i,j] contiene solo una freccia. Come ricostruisco la lcs?

Seguo le frecce. **Parto da LCS[m,n]**.

In ogni posizione [i,j] c'è una freccia.

Se la freccia in LCS[i,j] è **diagonale**, la lettera **a[i-1]** sarà uguale a **b[j-1]** e quella lettera **farà parte di lcs(a,b)**. Poi mi sposto nella casella indicata dalla freccia.

Se la freccia è **verticale o orizzontale**, non aggiungo lettere a lcs(a,b) e mi sposto nella casella indicata dalla freccia.

**Nota:** notate però che in questo modo **ricostruiamo la lcs al contrario** (dall'ultima lettera alla prima). Per «sopperire» a questo problema possiamo usare una funzione **ricorsiva** che stampa la lettera corrispondente dopo che le altre chiamate hanno stampato (abbiamo già visto questo metodo) oppure inserire le lettere in un vettore/lista **partendo dal fondo**.

# Esempio: lcs('GACATGC', 'ATCGAG') = 'ACAG'

matrice LCS

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | vettore a |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | -1 |
| 1 | | ↑ | ↑ | ↑ | ↖ | ← | ↖ | G 0 |
| 2 | | ↖ | ← | ← | ↑ | ↖ | ↑ | A 1 |
| 3 | | ↑ | ↑ | ↖ | ← | ↑ | ↑ | C 2 |
| 4 | | ↖ | ↑ | ↑ | ↑ | ↖ | ← | A 3 |
| 5 | | ↑ | ↖ | ↑ | ↑ | ↑ | ↑ | T 4 |
| 6 | | ↑ | ↑ | ↑ | ↖ | ↑ | ↖ | G 5 |
| 7 | | ↖ | ↑ | ↑ | ↑ | ↖ | ↑ | C 6 |
| vettore b | | A | T | C | G | A | G | |
| | -1 | 0 | 1 | 2 | 3 | 4 | 5 | |

matrice L

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| 3 | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| 6 | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| 7 | 0 | 1 | 2 | 2 | 3 | 4 | 4 |
| | | A | T | C | G | A | G |

# Complessità

Costruzione matrice: **$\Theta(mn)$**

Popolamento: **$\Theta(mn)$ passi**, in cui ciascun passo ha complessità **$O(1)$**

Ricostruzione soluzione: **$O(m+n)$**

**TOT: $\Theta(mn)$** (molto meglio di $O(n2^m)$ dell'algoritmo a forza bruta)

**COMPLESSITA' SPAZIALE: $\Theta(mn)$**

# Esempio

Ora facciamo un esempio di esecuzione dell'algoritmo.

Si noti che di seguito, rappresenteremo **LCS ed L nella stessa matrice** (per questioni di spazio).

In particolare, in ogni cella ci sarà un **numero**, che è il **contenuto di L**

Poi ci sarà una **freccia**, che è il **contenuto di LCS**

Per migliorare la leggibilità, ci sarà anche una **stringa**, che sarebbe il **contenuto di LCS senza l'ottimizzazione**. Ovviamente, nell'implementazione ottimizzata reale, tale stringa **non è memorizzata da nessuna parte**.

Per semplicità, **alcuni passi** sono stati **omessi**.

# Esempio: lcs("BACATBA", "ATCBAB")

| vett. b | | -1 | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|---|---|
| vett. a | | | A | T | C | B | A | B | |
| -1 | | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | 0 |
| 0 | B | -- 0 | | | | | | | 1 |
| 1 | A | -- 0 | | | | | | | 2 |
| 2 | C | -- 0 | | | | | | | 3 |
| 3 | A | -- 0 | | | | | | | 4 |
| 4 | T | -- 0 | | | | | | | 5 |
| 5 | B | -- 0 | | | | | | | 6 |
| 6 | A | -- 0 | | | | | | | 7 |
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | matrice LCS |

# lcs("B", "ATCB") = "B", di lunghezza 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | -- 0 | -- 0 | -- 0 | B 1 | | |
| A | -- 0 | | | | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("B", "ATCBA") = lcs("B", "ATCB") , lunghezza 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | |
| A | -- 0 | | | | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("B", "ATCBAB") = "B", lunghezza 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | | | | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("B<u>A</u>", "<u>A</u>") = "A", lunghezza 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | | | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("B<u>A</u>", "<u>A</u>T") = lcs("B<u>A</u>", "<u>A</u>") = "A", lunghezza 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | ↖ **A** 1 | ← 1 | | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BA", "ATC") = lcs("BA", "AT") = ... = "A", lungh. 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BA", "ATCB") = lcs("BA", "ATC") = ... = "A", lunghezza 1

lcs("BA", "ATCB") = lcs("B", "ATCB") = "B", lunghezza 1

è indifferente, scegliamo la seconda

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | ↖ **A** 1 | ← 1 | ← 1 | ↑ 1 | | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BA", "ATCBA") = lcs("B", "ATCB") + A = "BA", lunghezza 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BA", "ATCBAB") = lcs("BA", "ATCBA") = ... = "BA", lun. 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | -- 0 | -- 0 | -- 0 | B 1 | 1 | B 1 |
| A | -- 0 | A 1 | 1 | 1 | 1 | BA 2 | 2 |
| C | -- 0 | | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "A") = lcs("BA", "A") = "A", lun. 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | ↖ **A** 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "AT") = lcs("BA", "AT") =... = "A", lun. 1

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "ATC") = lcs("BA", "AT") + C = ... = "AC", lun. 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | ↖ **A** 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "ATCB") = lcs("BAC", "ATC") = "AC", lun. 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | -- 0 | -- 0 | -- 0 | **B** 1 | 1 | **B** 1 |
| A | -- 0 | **A** 1 | 1 | 1 | 1 | BA 2 | 2 |
| C | -- 0 | 1 | 1 | AC 2 | 2 | | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "ATCBA") = lcs("BA", "ATCBA") = "BA", lun. 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BAC", "ATCBAB") = lcs("BA", "ATCBAB") = "BA", lun. 2

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ **B** 1 | ← 1 | ↖ **B** 1 |
| A | -- 0 | ↖ **A** 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | ↑ 2 |
| A | -- 0 | | | | | | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

# andiamo avanti di alcuni passi

lcs("BACA", "ATCBA") = lcs("BAC", "ATCB") + A = "ACA", lun. 3

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | ↑ 2 |
| A | -- 0 | ↖ A 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ ACA 3 | |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BACA", "ATCBAB") = lcs("BACA", "ATCBA") = "ACA" lun. 3

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | ↑ 2 |
| A | -- 0 | ↖ A 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ ACA 3 | ← 3 |
| T | -- 0 | | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

lcs("BACAT", "A") = lcs("BACA", "A") = "A", lun. 3

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | ↑ 2 |
| A | -- 0 | ↖ A 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ ACA 3 | ← 3 |
| T | -- 0 | ↑ 1 | | | | | |
| B | -- 0 | | | | | | |
| A | -- 0 | | | | | | |

# Alla fine:

lcs("BACATBA", "ATCBAB") = ... = ACAB, lun. 4

| | | A | T | C | B | A | B |
|---|---|---|---|---|---|---|---|
| | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 | -- 0 |
| B | -- 0 | ← -- 0 | ← -- 0 | ← -- 0 | ↖ B 1 | ← 1 | ↖ B 1 |
| A | -- 0 | ↖ A 1 | ← 1 | ← 1 | ↑ 1 | ↖ BA 2 | ← 2 |
| C | -- 0 | ↑ 1 | ↑ 1 | ↖ AC 2 | ← 2 | ↑ 2 | ↑ 2 |
| A | -- 0 | ↖ A 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ ACA 3 | ← 3 |
| T | -- 0 | ↑ 1 | ↖ AT 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 |
| B | -- 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ ACB 3 | ↑ 3 | ↖ 4 ACAB |
| A | -- 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 3 | ↖ 4 ACBA | ↑ 4 |

# Cosa devo aver capito fino ad ora

- Problema della Longest Common Subsequence
- Sottostruttura Ottimale e sottoproblemi di lcs
- Memoizzazione delle soluzioni dei sottoproblemi
- Algoritmo di programmazione dinamica per lcs

# …se non ho capito qualcosa

- Alzo la mano e chiedo
- Ripasso sul libro
- Chiedo aiuto sul forum
- Chiedo o mando una mail al docente