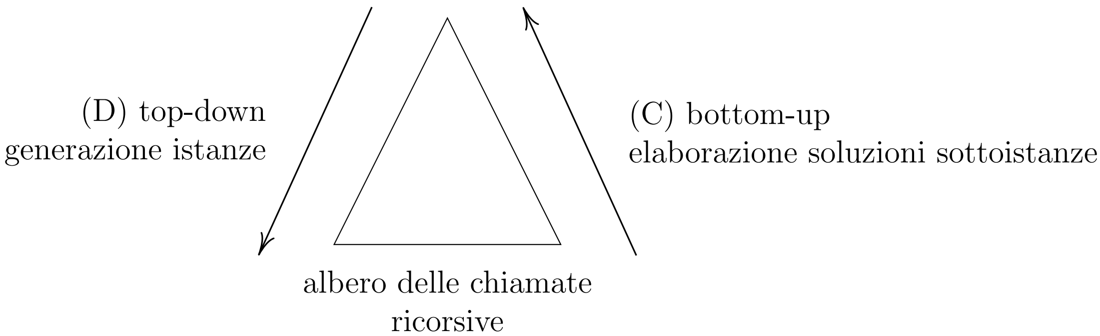
7 Programmazione Dinamica

7.1 Critica al Divide & Conquer (D&C)



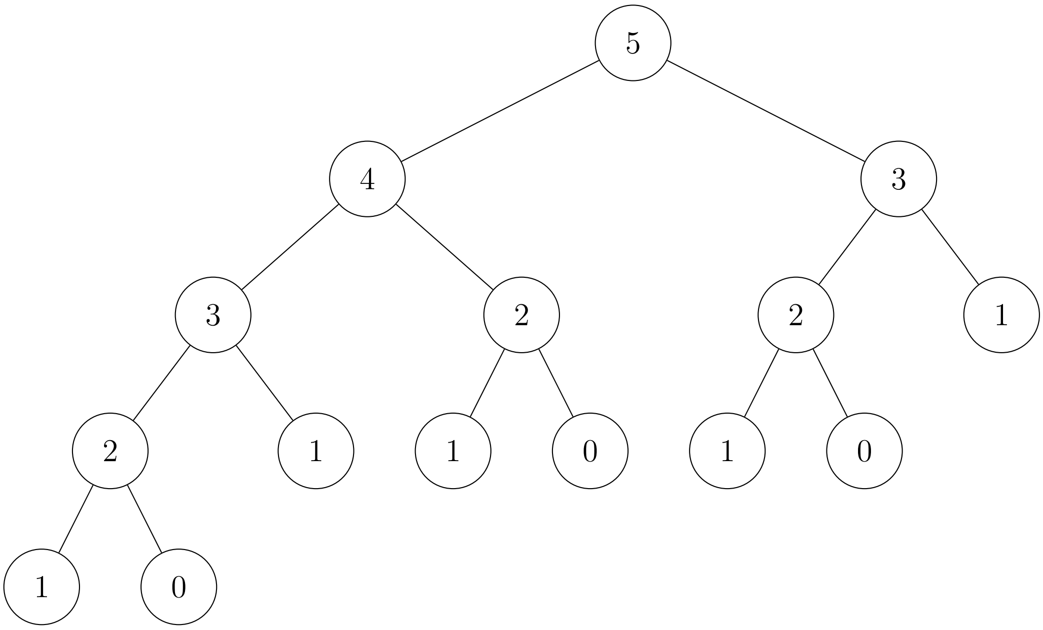
Il processo di soluzione non ha memoria, quindi le soluzioni di sottoistanze vanno ricalcolate.

Esempio Vediamo uno ”spreco” usando D&C: la sequenza di Fibonacci.

$$F(n) = \begin{cases} 1 & \text{se } n = 0, 1 \\ F(n-1) + F(n-2) & \text{se } n \geq 2 \end{cases}$$

```
Rec-Fib(n)
if (n = 0) or (n = 1)
    return 1
return Rec-Fib(n − 1) + Rec-Fib(n − 2)
```

Ad esempio, con $n = 5$



Vengono ricalcolate $F(3)$ e $F(2)$.

Complessità

$$T(n) = \begin{cases} 0 & \text{se } n = 0,1 \quad \text{(il "return" costa 0)} \\ T(n-1) + T(n-2) + 1 & \text{se } n \geq 2 \quad \text{(il "+" costa 1)} \end{cases}$$

$$\begin{aligned} T(n) &\geq T(n-1) + T(n-2) + 1 \\ &\geq 2T(n-2) + 1 \\ &\geq 2(2T(n-2-2) + 1) + 1 \\ &= 2^2 T(n-2-2) + 2 + 1 \\ &\geq 2^i T(n - 2 \cdot i) + \sum_{j=0}^{i-1} 2^j \end{aligned}$$

$$\begin{aligned} i_0 \to i = \left\lfloor \frac{n}{2} \right\rfloor & \\ \text{se n è pari:} \quad & 2^{\frac{n}{2}} T(n - 2\frac{n}{2}) = 2^{\frac{n}{2}} T(0) \\ \text{se n è dispari:} \quad & \left\lfloor \frac{n}{2} \right\rfloor = \frac{n-1}{2} \\ & 2^{\frac{n-1}{2}} T(n - 2\frac{n-1}{2}) = 2^{\frac{n-1}{2}} T(1) \end{aligned}$$

Otteniamo

$$T(n) \geq \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor - 1} 2^j = \Theta(2^{\frac{n}{2}})$$

In verità,

$$T(n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

Vediamo ora una versione iterativa:

IT-FIB(n)
if $(n = 0)$ **or** $(n = 1)$
    return 1
$F[0] = 1$
$F[1] = 1$
for $i = 2$ **to** n
    $F[i] = F[i-1] + F[i-2]$
return $F[n]$

Complessità $\Theta(n)$

La programmazione dinamica salta la fase top-down.

7.2 Memoizzazione

È un ibrido tra il D&C e la programmazione dinamica che vuole mantenere la fase top-down pur cercando di ricordare le soluzioni ai sottoproblemi.

Def Un algoritmo **memoizzato** è costituito da due subroutine distinte:

1) **routine di inizializzazione**: risolve direttamente i casi base e inizializza una struttura dati che contiene le soluzioni ai casi base e gli elementi per tutte le sottoistanze da calcolare, inizializzate ad un valore di default

2) **routine ricorsiva**: esegue il codice D&C preceduto da un test sulla struttura dati per verificare se la soluzione è già stata calcolata e memorizzata. Se sì, si ritorna, altrimenti la si calcola ricorsivamente e la si memorizza nella struttura.

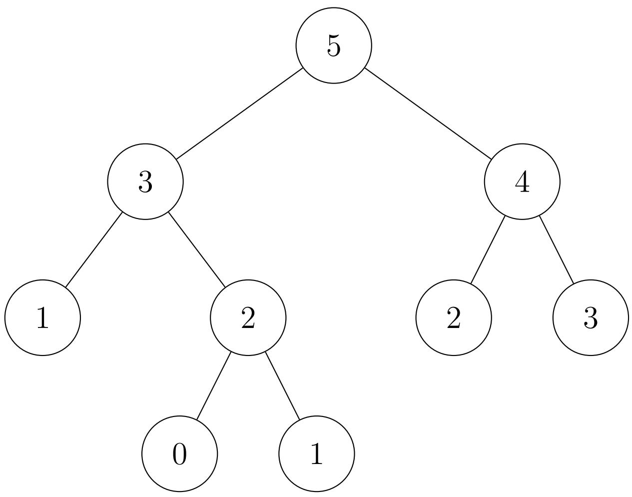
Esempio Riprendiamo l'esempio di prima sulla sequenza di Fibonacci.

```
Init-Fib(n)
if (n = 0) or (n = 1)
    return 1
F[0] = 1
F[1] = 1
for i = 2 to n
    F[i] = 0
return Rec-Fib(n)
```

Complessità $\Theta(n)$

```
Rec-Fib(i)
if F[i] = 0
    F[i] = Rec-Fib(i − 2) + Rec-Fib(i − 1)
return F[i]
```

Riprendiamo l'esempio di prima, con $n = 5$



Questa volta, $F(3)$ e $F(2)$ non vengono ricalcolate.
Abbiamo n foglie e $n - 1$ nodi interni (n parte da 0).

7.3 Problemi di Ottimizzazione

I = insieme delle istanze
S = insieme delle soluzioni
$\Pi \subseteq I \times S$
$\forall\ i \in I,\ S(i) = \{s \in S : (i, s) \in \Pi\}$ = insieme delle soluzioni ammissibili
funzione di costo $c\colon S \to \mathbb{R}$
Determinare, data $i \in I,\ s^* \in S(i) : c(s^*) = \min(/\max)\{c(s) : s \in S(i)\}$

Problema della raggiungibilità su un grafo orientato

$$I = \{\langle G = (V, E),\ u,\ v\rangle\ :\ V \subseteq \mathbb{N},\ V \text{ finito},\ E \subseteq V \times V,\ u, v \in V\}$$
$$S = \{\langle v_1, v_2, \dots, v_k\rangle\ :\ k \geq 1,\ v_i \in \mathbb{N} \quad \forall\ 1 \leq i \leq k\} \cup \{\varepsilon\} \qquad (\varepsilon = \text{cammino vuoto})$$
$$\Big(i = \langle G = (V, E),\ u,\ v\rangle,\ s\Big) \in \Pi \iff \begin{cases} S = \varepsilon, \exists \text{ un cammino tra } u \text{ e } v \text{ in } G \\ S = \langle v_1, v_2, \dots, v_k\rangle,\ v_1 = u,\ v_k = v, \\ \qquad (v_i, v_{i+1}) \in E \quad \forall\ 1 \leq i \leq k \end{cases}$$
$$c(\langle v_1, v_2, \dots, v_k\rangle) = k - 1$$
$$c(\varepsilon) = +\infty$$

Caratteristiche Un problema di ottimizzazione, per essere risolto con la programmazione dinamica, deve avere le seguenti caratteristiche:

- struttura ricorsiva;
- esistenza di sottoistanze ripetute;
- spazio di sottoproblemi "piccolo".

Paradigma Generale

1. Caratterizza la struttura di una soluzione ottima s^* in funzione di soluzione ottime $s_1^*, s_2^*, \dots, s_k^*$ di sottoistanze di taglia inferiore.
2. Determina una relazione di ricorrenza del tipo $c(s^*) = f(c(s_1^*), \dots, c(s_k^*))$.
3. Calcola $c(s^*)$ impostando il calcolo in maniera bottom-up (oppure memoizzando).
4. Mantiene informazioni strutturali aggiuntive che permettono di ricostruire s^*.

7.4 Problemi su Stringhe

Def Dato un alfabeto finito Σ, una **stringa**

$$X = \langle x_1, x_2, \ldots, x_m \rangle, \quad x_i \in \Sigma \quad \forall\, 1 \le i \le m$$

è una concetazione finita di simboli in Σ.

$m = |X| =$ lunghezza di X
$\Sigma^* =$ insieme di tutte le stringhe di lunghezza finita costruibili su Σ
$\varepsilon =$ stringa vuota

Data una stringa X, il **prefisso** di X è

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle, \quad 1 \le i \le m$$

Data una stringa X, il **suffisso** di X è

$$X^i = \langle x_i, x_{i+1}, \ldots, x_m \rangle, \quad 1 \le i \le m$$

Per convenzione $X_0 = X^{m+1} = \varepsilon$

Def Data una stringa X, la **sottostringa** di X è

$$X_{i \ldots j} = \langle x_i, x_{i+1}, \ldots, x_j \rangle, \quad 1 \le i \le j \le m$$

Per convenzione $X_{i \ldots j} = \varepsilon \quad$ se $i > j$

possibili sottostringhe di una stringa con m caratteri:

$$\underset{\substack{\uparrow \\ i \neq j}}{\binom{m}{2}} + \underset{\substack{\uparrow \\ i = j}}{m} + \underset{\substack{\uparrow \\ \varepsilon}}{1} = \frac{m(m+1)}{2} = \Theta(m^2)$$

Lo spazio delle sottostringhe "non è troppo grande".

Def Data una stringa

$$X = \langle x_1, x_2, \ldots, x_m \rangle \in \Sigma^*$$

e

$$Z = \langle z_1, z_2, \ldots, z_k \rangle \in \Sigma^*$$

si dice che Z è **sottosequenza** di X se $\exists$ una successione crescente di indici

$$1 \le i_1 \le i_2 \le \cdots \le i_k \le m \ : \ z_j = x_{ij} \quad \forall\, 1 \le j \le k$$

Esempio

$X = \langle a, b, c, b, b, d \rangle$
$Z_1 = \langle a, b, c \rangle = X_{1\ldots3}$
$Z_2 = \langle a, c, b \rangle \qquad i_1 = 1, \quad i_2 = 3, \quad i_3 = 4 \text{ o } 5$
$Z_3 = \langle b, b \rangle = X_{4\ldots5} \qquad i_1 = 2, \quad i_2 = 5$

possibili sottosequenze di una stringa con m caratteri:

$$\sum_{k=0}^{m} \qquad \underset{\substack{\uparrow \\ \text{stringhe lunghe } k \\ \text{prese da un insieme} \\ \text{di } m \text{ elementi}}}{\binom{m}{k}} \qquad = 2^m$$

7.5 Longest Common Subsequence (LCS)

7.5.1 Problema di Ottimizzazione

Date due stringhe X, Y determina Z tale che:

1) Z è sottosequenza di X e Y;

2) Z è la più lunga tra tutte le sottosequenze comuni.

Esempio

$X = \langle a, b, c, b, b, d \rangle$
$Y = \langle a, d, c, c, b, d \rangle$
$Z = \langle a, c, b, d \rangle$ è una LCS (in questo caso è l'unica)

$$i_1 = 1, \quad i_2 = 3, \quad i_3 = 4 \text{ o } 5, \quad i_4 = 6$$
$$j_1 = 1, \quad j_2 = 3 \text{ o } 4, \quad j_3 = 5, \quad j_4 = 6$$

Risolvo il problema:

$|X| = m$
$|Y| = n$

L'approccio "brute force" ha complessità $\Omega(2^m \cdot 2^n)$.

Devo cercare di individuare una proprietà di sottostruttura, cioè la LCS deve "nascondere" al suo interno LCS di qualche stringa più piccola di X e Y.

$X = \langle b, c, f, a \rangle$
$Y = \langle c, f, d, a \rangle$
$Z = LCS(X, Y) = \langle Z', a \rangle \qquad \text{con } Z' = LCS(X_3, Y_3)$

$X = \langle X', a \rangle$
$Y = \langle Y', b \rangle$
Z o non termina con a, o non termina con b
$Z = LCS(X', Y)$ o $LCS(X, Y')$
$S = \{LCS(X_i, Y_j) : 0 \leq i \leq m,\ 0 \leq j \leq n\}, \quad |S| = (m\underset{\overset{\uparrow}{\varepsilon}}{+1})(n\underset{\overset{\uparrow}{\varepsilon}}{+1})$

7.5.2 Proprietà di Sottostruttura Ottima

Dati i prefissi

$X_i = \langle x_1, x_2, \dots, x_i \rangle$
$Y_i = \langle y_1, y_2, \dots, y_j \rangle$
Sia $Z = \langle z_1, z_2, \dots, z_k \rangle = LCS(X_i, Yj)$

0. caso base: o $i = 0$ o $j = 0$
   $\Rightarrow Z = \varepsilon$

1. $i, k > 0$
   se $x_i = y_j$ allora
   (a) $z_k = x_i (= y_j)$
   (b) $Z_{k-1} = LCS(X_{i-1}, Y_{j-1})$

2. $i, j > 0$
   se $x_i \neq y_j$ allora
   Z è la stringa di lunghezza massima tra $LCS(X_i, Y_{j-1})$ e $LCS(X_{i-1}, Y_j)$

Dimostrazione

0. banale

1. $x_i = y_j$

   $Z = LCS(X_i, Y_j) = \langle z_1, z_2, \dots, z_k \rangle = \langle x_{i_1}, x_{i_2}, \dots, x_{i_k} \rangle = \langle y_{j_1}, y_{j_2}, \dots, y_{j_k} \rangle$
   $1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq i, \qquad 1 \leq j_1 \leq j_2 \leq \cdots \leq j_k \leq j$

   (a) Ragioniamo per assurdo

   $z_k = x_{i_k} = y_{j_k}$
   $z_k \neq (x_i = y_j)$
   $\Rightarrow i_k < i, \quad j_k < j$
   $Z' = \langle Z, x_i \rangle$
   $1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq i_{k+1} = i, \qquad 1 \leq j_1 \leq j_2 \leq \cdots \leq j_k \leq j_{k+1} = j$

   (b) Devo dimostrare che

   $Z_{k-1} = LCS(X_{i-1}, Y_{j-1})$
   $Z_{k-1} = \langle x_{i_1}, x_{i_2}, \dots, x_{i_{k-1}} \rangle = \langle y_{j_1}, y_{j_2}, \dots, y_{j_{k.1}} \rangle$
   $i_{k-1} \leq i - 1 < i$
   $Z_{k-1} = CS(X_{i-1}, Y_{j-1})$

Ora dimostro che

$Z_{k-1} = LCS(X_{i-1}, Y_{j-1})$

Suppongo per assurdo che

$Z_{k-1} \neq LCS(X_{i-1}, Y_{j-1})$
$\Rightarrow \exists\ Z'$ con $|Z'| \geq k$
$\Rightarrow$ creo $Z'' = \langle \underset{\substack{\uparrow \\ \geq k}}{Z'}\ ,\ \underset{\substack{\uparrow \\ 1}}{x_i}(= y_j)\rangle \Rightarrow\ \geq k+1$

2. (come esercizio)

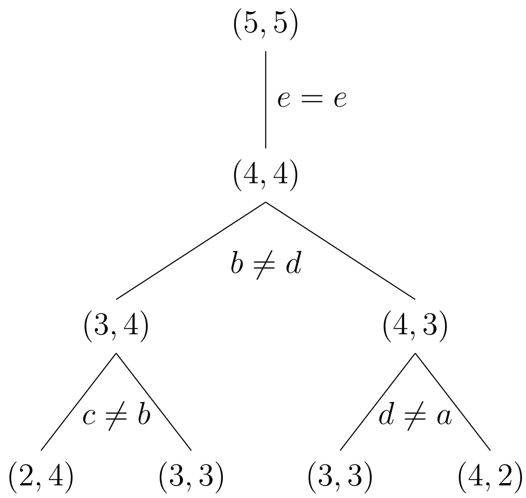
Il D&C "non funziona" perchè ci sono molti sottoproblemi ripetuti.

Esempio

$X = \langle a, b, c, d, e\rangle$
$Y = \langle b, c, a, b, e\rangle$

Trova $LCS(X, Y)$

Albero delle chiamate:



L'istanza $(3, 3)$ è ripetuta.

Complessità Strategia Ricorsiva Modello di costo: confronto tra caratteri.

$$T(n, m) = \begin{cases} 0 & \text{se } n = 0 \text{ o } m = 0 \\ T(n-1, m) + T(n, m-1) + 1 & \text{se } n, m > 0 \end{cases}$$

Si dimostra che

$$T(n, m) = \Theta\left(\binom{m}{n}\right)$$
$$\binom{m}{2} \geq \binom{m}{2}^n$$

Caso $m = 2n$

$$\binom{m}{2}^n = 2^n$$

7.5.3 Ricorrenza sui Costi

La scrittura della ricorrenza sui costi è il secondo passo per costruire un algoritmo di programmazione dinamica.

Definisco

$l(i,j) = |LCS(X_i, Y_j)|$

$$l(i,j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 & \text{(caso 0)} \\ l(i-1, j-1) + 1 & \text{se } i,j > 0 \text{ e } x_i = x_j & \text{(caso 1)} \\ \max\{l(i, j-1), l(i-1, j)\} & \text{se } i,j > 0 \text{ e } x_i \neq x_j & \text{(caso 2)} \end{cases}$$

Alla fine ci interessa calcolare $l(m,n)$.

Per calcolare $l(i,j)$ mi possono servire tre valori:

$$L = \begin{bmatrix} & & \\ (i{-}1, j{-}1) & & (i{-}1, j) \\ & \nwarrow & \uparrow \\ (i, j{-}1) & \leftarrow & (i,j) \\ & & \end{bmatrix}$$

Scansione "row-major": riempio la tabella per righe, da sinistra a destra.

Informazione addizionale per costruire la sequenza (vera e propria):

$$b(i,j) = \begin{cases} '\nwarrow' & \text{se } x_i = y_j \\ '\leftarrow' & \text{se } x_i \neq x_j \text{ e } max = LCS(i, j-1) \\ '\uparrow' & \text{se } x_i \neq y_j \text{ e } max = LCS(i-1, j) \end{cases}$$

Pseudocodice

```
LCS(X, Y)
m = X.[length]
n = Y.length
for i = 0 to m
      L[i, 0] = 0
for j = 0 to n
      L[0, j] = 0
for i = 1 to m
      for j = 1 to n
            if x_i = y_j
                  L[i, j] = L[i − 1, j − 1] + 1
                  B[i, j] = '↖'
            else if L[i − 1, j] ≥ L[i, j − 1]
                  L[i, j] = L[i − 1, j]
                  B[i, j] = '↑'
            else
                  L[i, j] = L[i, j − 1]
                  B[i, j] = '←'
return (L[m, n], B)
```

Complessità

$T(m, n) = \Theta(m \cdot n)$
Caso $m = n \Rightarrow T(m, n) = \Theta(n^2)$

Procedura per stampare la LCS:

```
PRINT-LCS(B, X, i, j)
if i = 0 or j = 0
      return ε
if B[i, j] = '↖'
      PRINT-LCS(B, X, i − 1, j − 1)
      PRINT(x_i)
else if B[i, j] = '←'
      PRINT-LCS(B, X, i, j − 1)
else // B[i, j] = '↑'
      PRINT-LCS(B, X, i − 1, j)
```

Complessità $\Theta(m) = \Theta(|LCS|)$

Esercizio

$X = \langle b, d, c, d \rangle$
$Y = \langle a, b, c, b, d \rangle$

Restituisci $LCS(X, Y)$ e $|LCS(X, Y)|$

$$L = \begin{array}{c c} & \begin{array}{cccccc} & a & b & c & b & d \end{array} \\ \begin{array}{c} \\ b \\ d \\ c \\ d \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & \mathbf{2} & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & \mathbf{3} \end{bmatrix} \end{array} \qquad B = \begin{array}{c c} & \begin{array}{ccccc} a & b & c & b & d \end{array} \\ \begin{array}{c} \\ b \\ d \\ c \\ d \end{array} & \begin{bmatrix} & & & & \\ \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \\ \uparrow & \uparrow & \nwarrow & \leftarrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \end{bmatrix} \end{array}$$

$LCS(X, Y) = \langle b, c, d \rangle \qquad |LCS(X, Y)| = 3$

Pseudocodice Memoizzato

```
INIT-LCS(X, Y)
m = X.length
n = Y.length
if (m = 0) or (n = 0)
    return 0
for i = 0 to m
    L[i, 0] = 0
for j = 0 to n
    L[0, j] = 0
for i = 1 to m
    for i = 1 to n
        L[i, j] = −1
return R-LCS(X, Y, m, n)
```

```
R-LCS(X, Y, i, j)
if L[i, j] = −1
    if x_i = y_j
        L[i, j] = R-LCS(X, Y, i − 1, j − 1)
    else if R-LCS(X, Y, i − 1, j) ≥ R-LCS(X, Y, i, j − 1)
        L[i, j] = L[i − 1, j]
    else
        L[i, j] = L[i, j − 1]
return L[i, j]
```

Complessità $O(m \cdot n)$

Osservazione Se $x_i = y_j$ sempre, invoco `R-LCS(X,Y,i-1,j-1)` ma non invoco mai `R-LCS(X,Y,i-1,j)` o `R-LCS(X,Y,i,j-1)`.

Ad esempio

$X = \langle a, a, b, b, c\rangle$
$Y = \langle b, b, c\rangle$

Albero delle chiamate:

$$\begin{array}{c}(5,3)\\ \Big| \; c = c\\ (4,2)\\ \Big| \; b = b\\ (3,1)\\ \Big| \; b = b\\ (2,0)\end{array}$$

In generale, se Y è suffisso di $n \leq m$ caratteri di X, la complessità di `R-LCS` nel caso migliore è:

$$T_{R-LCS}(m, n) = n$$

Inoltre,

$$\Omega_{LCS}(m + n) \approxeq \Omega(n)$$
$$O_{LCS,R-LCS}(m \cdot n) \approxeq O(n^2)$$

Spazio

$$S_{LCS}(m, n) = \Theta(m, n)$$

Tuttavia, posso migliorarlo a

$$\Theta(2n) = \Theta(n)$$

poichè mi bastano due righe della tabella in memoria ad ogni istante, quindi due vettori lunghi n.

Inoltre, se $m \ll n$, posso fare un'ulteriore ottimizzazione utilizzando la tecnica "column-major", cioè scansione per colonne, con due vettori lunghi m.

7.6 Longest Increasing Subsequence (LIS)

Def Dato un alfabeto Σ totalmente ordinato ($\forall\ a, b \in \Sigma \quad a < b$ o $a = b$ o $a > b$) e dato $X = \langle x_1, x_2, \ldots, x_n \rangle$, si dice che $Z = \langle z_1, z_2, \ldots, z_k \rangle$ è **sottosequenza crescente** di X ($Z = IS(X)$).

7.6.1 Problema di Ottimizzazione

Determinare la **più lunga sottosequenza crescente** di X ($Z = LIS(X)$)

Esempio

$X = \langle 5, 2, 4, 3, 7, 8 \rangle$
$Z = LIS(X) = \langle 2, 3, 7, 8 \rangle$
$Z' = LIS(X) = \langle 2, 4, 7, 8 \rangle$

Tentativo Data X, calcolo $LIS(X_i)\ \forall\ 0 \leq i \leq n$
$Z = \langle z_1, z_2, \ldots, z_k \rangle = LIS(X_i)$

- caso fortunato: $z_k < X_{i+1}$
- caso sfortunato: $z_k \geq X_{i+1}$

Def $Z = \overline{LIS}(X_i)$ è la più lunga tra le $IS(X_i)$ con $Z = \langle z_1, z_2, \ldots, z_k \rangle = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$ con $i_k = i$.

Esempio

$X = \langle 8, 2, 5, 1, 3 \rangle$
$LIS(X_4) = \langle 2, 5 \rangle$
$\overline{LIS}(X_4) = \langle 1 \rangle$

In generale, LIS e $\overline{LIS}$ sono problemi molto diversi.

Osservazione $|LIS(X)| = \max_{1 \leq i \leq n} \{ |\overline{LIS}(X_i)| \}$

Dimostrazione

- $|LIS(X)| \leq \max_{1\leq i\leq n}\{|\overline{LIS}(X_i)|\}$

  $Z = LIS(X) = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k}\rangle$
  $Z = \overline{LIS}(X_{i_k})$
  $\Rightarrow |Z| \leq \max_{1\leq i\leq n}\{|\overline{LIS}(X_i)|\}$

- $|LIS(X)| \geq \max_{1\leq i\leq n}\{|\overline{LIS}(X_i)|\}$
  Si dimostra analogamente al punto precedente.

7.6.2 Proprietà di Sottostruttura Ottima

1. caso base: $\overline{LIS}(X_1) = \langle x_1\rangle$ $(= LIS(X_1))$
2. $i > 1$
   (a) $\forall\ j : 1 \leq j \leq i \quad x_j \geq x_i$

   $\overline{LIS}(X_i) = \langle x_i\rangle$ $(= LIS(X_i))$

   (b) $\exists\ \bar{j} : 1 \leq n\bar{j} \leq i, \quad x_{\bar{j}} < x_i$

   $|\overline{LIS}(X_i)| \geq 2$
   $\overline{LIS}(X_i) = \langle z_1, z_2, \ldots, z_k\rangle = \langle Z_{k-1}, x_i\rangle$ con $Z_{k-1} : |Z_{k-1}| = \max_{1\leq j<i}\{\overline{LIS}(X_j) : x_j < x_i\}$

Dimostrazione

1. banale
2. $i > 1$
   (a) $\forall\ j < i \quad x_j < x_i$

   $\langle x_i\rangle = IS(X_i)$ e chiaramente non può essere $|Z| > 1$

   Suppongo per assurdo che

   $|Z| = |\overline{LIS}(X_i)| > 1$
   $\Rightarrow Z = \langle z_1, z_2, \ldots, z_k\rangle, \quad k > 1$
   $Z = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k}\rangle$
   $i_k = i$
   $\Rightarrow i_{k-1} < i_k = i$

Assurdo perchè allora avrei

$x_{i_{k-1}} \geq x_{i_k}$

che contraddice l'ipotesi che Z è una sequenza crescente!

(b) Si dimostra analogamente al sottocaso precedente.

7.6.3 Ricorrenza sui Costi

Definisco

$l(i) = \left|\overline{LIS}(X_i)\right|$

$$l(i) = \begin{cases} 1 & \text{se } i = 1 \\ 1 + \max\limits_{1 \leq j < i}\{l(j) : x_j < x_i\} & \text{se } i > 1 \end{cases}$$

Per costruire la sottosequenza

$$\overline{LIS}(X_i) = \begin{cases} \langle x_i \rangle & & (1) \\ \langle \overline{LIS}(X_{\overline{j}}), x_i \rangle & \text{con } 1 \leq \overline{j} < i & (2) \end{cases}$$

l'informazione addizionale è:

- $prev(i) = \begin{cases} 0 & \text{nel caso (1)} \\ \overline{j} & \text{nel caso (2)} \end{cases}$
- $len = \max\limits_{1 \leq i \leq n}\{l(i)\}$
- $end = i \quad \text{se } \overline{LIS}(X_i) = LIS(X)$
  (mantiene l'indice dell'ultimo carattere della LIS)

Pseudocodice bottom-up

```
LIS(X)
L[1] = 1
len = 1
end = 1
prev[1] = 0
for i = 2 to n
    L[i] = 1
    prev[i] = 0
    for j = 1 to i - 1
        if x_j < x_i
            if L[i] < 1 + L[j]
                L[i] = 1 + L[j]
                prev[i] = j
    if len < L[i]
        len = L[i]
        end = i
return (len, prev, end)
```

Procedura per stampare la LIS:

```
R-PRINT(X, prev, i)
if prev[i] ≠ 0
    R-PRINT(X, prev, prev[i])
PRINT(x_i)
```

Complessità $\sum_{i=2}^{n}\sum_{j=1}^{i-1} 1 = \sum_{i=2}^{n}(i-1) = \frac{n(n-1)}{2} = \Theta(n^2)$

7.7 Completamento a Palindromo (CP)

Def Una stringa $Z = \langle z_1, z_2, \ldots, z_m \rangle$ è **palindroma** se $z_{1+h} = z_{m-h} \quad \forall\, 0 \leq h \leq m-1$.

Problema Data $X = \langle x_1, x_2, \ldots, x_n \rangle$, un **complemento palindromo** di X è una stringa $Z = CP(X) = \langle z_1, z_2, \ldots, z_m \rangle$ con $m \geq n$ tale che:

1) Z è palindroma;

2) X è sottosequenza di Z

(cioè, Z è un palindromo che contiene X come sottosequenza).

Esempio

$X = \langle a, c, b \rangle$
$Z = \langle a, c, b, b, c, a \rangle$
$Z' = \langle a, b, c, c, b, a \rangle$

Osservazione $|X| = n \leq |Z| \leq 2n = 2|X|$

7.7.1 Problema di Ottimizzazione

Determinare $Z = CP(X)$ di lunghezza minima.
Spazio dei sottoproblemi: $X_{i\ldots j}$ (cioè lo spazio delle sottostringhe di X).
Determinare un algoritmo bottom-up quadratico nella lunghezza di X.
$\forall\, i, j : 1 \leq i \leq j \leq n$, determinare il minimo $Z = CP(X_{i\ldots j})$.

7.7.2 Proprietà di Sottostruttura Ottima

Casi base:

1. $i = j$ (1 carattere)

   $X = \langle x_i \rangle$
   $CP(X_{i\ldots j}) = X_{i\ldots j}$
   $|CP(X_{i\ldots j})| = |X_{i\ldots j}|$

2. $j = i + 1$ (2 caratteri)

   $X_{i\ldots j} = \langle x_i, x_{i+1} \rangle$

(a) $x_i = X_{i+1}$

$CP(X_{i...i+1}) = X_{i...i+1}$

$|CP(X_{i...i+1})| = 2$

(b) $x_i \neq x_{i+1}$

Un possibile $CP(X_{i...i+1})$ è $\langle x_i, x_{i+1}, x_i \rangle$

$|CP(X_{i...i+1})| = 3$

Caso generale:

$X_{i...j} = \langle x_i, x_{i+1}, \ldots, x_j \rangle$

(a) $x_i = x_j$

$Z = CP(X_{i...j})$

$z_1 = z_k = x_i \ (= x_j)$

$Z_{2...k-1} = CP(X_{i+1...j-1})$

(b) $x_i \neq x_j$ Ci sono due possibili soluzioni:

i. $z_1 = z_k = x_i$ e $Z_{2...k-1} = CP(X_{i+1...j})$

ii. $z_1 = z_k = x_j$ e $Z_{2...k-1} = CP(X_{i...j-1})$

Dimostrazione

(b).i. Suppongo per assurdo che

$z_1 = z_k \neq x_i$

$\Rightarrow X_{i...j}$ è sottosequenza di

$Z_{2...k-1}$ che è palindroma e più corta di 2 rispetto a Z

Assurdo perchè Z è la più corta!

(b).ii. Si dimostra analogamente al sottocaso precedente.

7.7.3 Ricorrenza sulle Lunghezze

Definisco

$l(i,j) = |CP(X_{i...j})|$

$$l(i,j) = \begin{cases} 1 & \text{se } i = j \\ 2 & \text{se } j = i+1 \text{ e } x_i = x_j \\ 3 & \text{se } j = i+1 \text{ e } x_i \neq x_j \\ 2 + l(i+1, j-1) & \text{se } j > i+1 \text{ e } x_i = x_j \\ 2 + \min\{l(i+1,j), l(i,j-1)\} & \text{se } j > i+1 \text{ e } x_i \neq x_j \end{cases}$$

Per calcolare $l(i,j)$ mi possono servire tre valori:

$$L = \begin{bmatrix} & & \\ (i,j{-}1) & \leftarrow & (i,j) \\ & \swarrow & \downarrow \\ (i{+}1,j{-}1) & & (i{+}1,j) \\ & & \end{bmatrix}$$

Riempio la tabella per diagonali, dall'alto verso il basso e da sinistra verso destra.

SPC(X)

```
n = X.length
for i = 1 to n − 1
    L[i, j] = 1
    if x_i = x_{i+1}
        L[i, i + 1] = 2
    else
        L[i, i + 1] = 3
L[n, n] = 1
for l = 3 to n // scansione diagonale con l indice della diagonale
    for i = 1 to n − l + 1
        j = i + l − 1
        if x_i = x_j
            L[i, j] = 2 + L[i + 1, j − 1]
        else
            L[i, j] = 2 + min{L[i + 1, j], L[i, j − 1]}
return L[1, n]
```

Complessità

$$\sum_{i=1}^{n-1} 1 + \sum_{l=3}^{n} \sum_{i=1}^{n-l+1} 1 = \sum_{i=1}^{n-1} 1 + \sum_{l=3}^{n} (n-l+1) = \sum_{i=1}^{n-1} 1 + \sum_{i=2}^{n-1} (n-i) =$$

$$= \sum_{i=1}^{n-1} (n-i) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \Theta(n^2)$$

8 Algoritmi Greedy

Un **algoritmo greedy** ("incauto", "irruento") è un tipo di algoritmo per problemi di ottimizzazione che cerca di risolvere il problema in modo diretto, scegliendo la soluzione al sottoproblema più piccolo che al momento sembra la migliore.

Caratteristiche:

- semplice, con complessità di solito linerare;
- difficoltà di analisi;
- limitato campo di applicazione.

8.1 Critica alla Programmazione Dinamica

La programmazione dinamica è "prudente", nel senso che risolve <u>tutti</u> i sottoproblemi di una certa taglia.

8.2 Problema di Selezione di Attività Compatibili

Abbiamo:

- risorsa condivisa (e.g. aula);
- insieme di attività $S = \{a_i : 1 \leq i \leq n\}$

  $a_i = [s_i, f_i), \quad 0 \leq s_i \leq f_i \quad (s_i = \text{tempo di inizio}, f_i = \text{tempo di fine})$

Def Diciamo che a_i e a_j sono **compatibili** sse

$$[s_i, f_i) \cap [s_j, f_j) = \emptyset$$

Equivalentemente

$$f_i \leq s_j \text{ oppure } f_j \leq s_i$$

Esempio

Aula Lum250

$a_1 = [10.30, 12.00)$

$a_2 = [11.30, 13.00)$

$a_3 = [12.30, 14.00)$

a_1 e a_2 non sono compatibili

a_1 e a_3 sono compatibili

a_2 e a_3 non sono compatibili

8.2.1 Problema di Ottimizzazione

Determinare un sottoinsieme di massima cardinalità di attività mutuamente compatibili (cioè compatibili a coppie).

Ulteriore assunzione:

$$0 < f_1 \leq f_2 \leq \cdots \leq f_n$$

Voglio applicare la programmazione dinamica:

determinare i sottoproblemi

$$S_{ij} = \{a_k : f_i \leq s_k < f_k \leq s_j\}$$

Osservazione

1. $i > j \Rightarrow S_{ij} = \emptyset$

   perchè $f_i \leq s_k < f_k \leq s_j$
   ma $f_i \geq f_j$

2. S_{ij} non contiene tutte le attività di indice k con $i < k < j$

3. $|S_{ij}| \leq j - 1 - i, \quad 1 \leq i \leq n$

   Definisco due attività fittizie:

   $a_0 \to f_0 = 0$
   $a_{n+1} \to s_{n+1} = +\infty$
   $S = S_{0\,n+1}$

8.2.2 Proprietà di Sottostruttura Ottima

Sia A^*_{ij} un sottoinsieme di attività compatibili di S_{ij} di cardinalità massima.

Caso base: $S_{ij} = \emptyset \Rightarrow A_{ij} = \emptyset$

Caso generale: $S_{ij} \neq \emptyset$

1. $A^*_{ij} \neq \emptyset$
2. se $a_k \in A^*_{ij}$ allora $A^*_{ij} = A^*_{ik} \cup A^*_{kj}$
   dove A^*_{ik} e A^*_{kj} sono soluzioni ottime per S_{ik} e S_{kj}

Dimostrazione

Caso base: banale

Caso generale: $S_{ij} \neq \emptyset$

1. Un'attività è sempre compatibile con sè stessa $\Rightarrow \left|A^*_{ij}\right| \geq 1$
2. Dimostreremo che una qualsiasi attività $\in A^*_{ij}$ e $\neq a_k$ si trova o in S_{ik} o in S_{kj}

   Ho a_k, una qualsiasi attività $\in A^*_{ij}$ deve essere compatibile con a_k

   Prendiamo $a_l \in A^*_{ij},$ con $l \neq k$
   Deve valere
   $f_i \leq s_l < f_l \leq s_k \Rightarrow a_l \in S_{ik}$
   oppure
   $s_k < f_k \leq s_l < f_l \leq s_j \Rightarrow a_l \in S_{kj}$
   Quindi $A^*_{ij} = \overline{A}_{ik} \cup \{a_k\} \cup \overline{A}_{kj}$
   dove $\overline{A}_{ij}$ = tutte le attività compatibili in S_{ij}

   Ora dimostriamo che gli $\overline{A}$ sono anche A^*

   Per assurdo suppongo che $\overline{A}_{ik}$ non sia l'unico insieme di attività compatibili massimale in S_{ik}

   $$A^*_{ij} = \underset{\substack{\uparrow \\ A^*_{ik}}}{\cancel{\overline{A}_{ik}}} \cup \{a_k\} \cup \overline{A} \ \rightarrow \ \text{soluzione con cardinalità } > \left|A^*_{ij}\right|$$

8.2.3 Ricorrenza sui Costi

Definisco

$$c(i,j) = \left|A^*_{ij}\right|$$

$$c(i,j) = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ 1 + \max\limits_{a_k \in S_{ij}} \{c(i,k) + c(k,j)\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

L'algoritmo bottom-up ha complessità $\Theta(n^3)$

Abbiamo visto che $A^*_{ij} = A^*_{ik} \cup \{a_k\} \cup A^*_{kj}$ ma non so chi è k.
Se conoscessi k potrei scrivere $1 + c(i,k) + c(k,j)$.

Teorema Sia a_m l'attività tale che

$$f_m = \min\{f_k : a_k \in S_{ij}\}$$

Se $S_{ij} \neq \emptyset$ allora

1. $\exists\ A^*_{ij}$ soluzione ottima tale che $a_m \in A^*_{ij}$
2. il sottoproblema $S_{im} = \emptyset$

Dimostrazione

1. Si consideri una soluzione ottima $\overline{A}_{ij}$ a S_{ij}

   Se $a_k = a_m \Rightarrow$ ho finito

   Altrimenti, $a_k \neq a_m \Rightarrow f_m \leq f_k \Rightarrow$ posso togliere a_k da $\overline{A}_{ij}$
2. banale

Strategia

1) Scegliere a_m
2) Risolvere S_{mj} (perchè $A^*_{ij} = \{a_m\} \cup A^*_{mj}$)

A noi interessa risolvere $S_{0\,n+1}$

Osservazione Devo risolvere solo problemi del tipo $S_{m\,n+1}$
$\Rightarrow$ lo spazio dei sottoproblemi è linerare (perchè il 2° indice dei sottoproblemi è fisso).

Codice ad alto livello

OPT($S_{i\,n+1}$)
if $S_{i\,n+1} \neq \emptyset$
    $a_m = f_m = \min\{f_k : a_k \in S_{i\,n+1}\}$ // molto grande
else
    return $\emptyset$

Pseudocodice

```
REC-SEL(S, f, i)
n = S.length
m = i + 1
while (m ≤ n) and (s_m < f_i)
      m = m + 1
if m ≤ n
      return {a_m} ∪ REC-SEL(S, f, m)
else
      return ∅
```

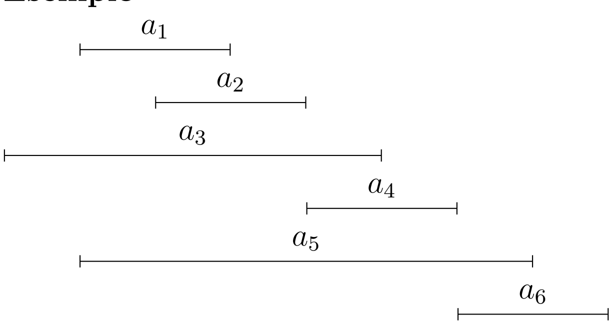
Complessità Modello di costo: confronti tra elementi s e f.

La complessità è $\Theta(n)$ perchè ogni attività viene coinvolta in un confronto una sola volta.

Versione iterativa:

```
GREEDY-SEL(S, f)
n = S.length
A = {a_1}
last = 1 // indice dell'ultima attività selezionata
for m = 2 to n
      if s_m ≥ f_last
            A = A ∪ {a_m}
            last = m
return A
```

Esempio



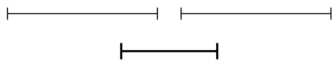
$A = a_1$	$A = a_1, a_4$	$A = a_1, a_4, a_6$
$last = 1$	$last = 4$	$last = 6$

8.2.4 Scelte Greedy Alternative

Oltre alla scelta greedy vista precedentemente, ne esistono altre:

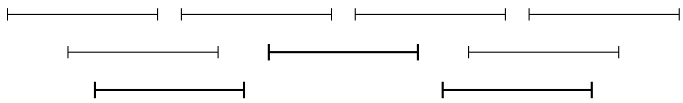
- Scegli l'attività di durata inferiore → non è ottima.

  Controesempio:

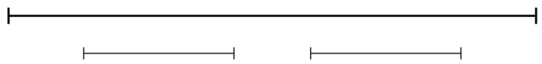
- Scegli l'attività col minor numero di sovrapposizioni → non è ottima.

  Controesempio:

- Scegli l'attività che inizia per prima → non è ottima.

  Controesempio:

- Scegli l'attività che inizia per ultima → è ottima.

8.3 Compressione dei Dati: Codici di Huffman

La compressione può essere di due tipi:

- lossy, cioè con perdita di informazione (e.g. immagine, video);
- lossless, cioè senza perdita di informazione (e.g. testo).

Esempio File di caratteri con frequenze associate:

	a	b	c	d	e	f
frequenza (%)	45	13	12	16	9	5

La codifica ASCII usa 8 bit per carattere.
File con 100K caratteri → 800 Kbit

Definisco una codifica che usa 3 bit per carattere:

	a	b	c	d	e	f
codeword	000	001	010	011	100	101

File con 100K caratteri → 300 Kbit (risparmio più del 50%)
+ spazio per una tabella di conversione/decodifica

0	'a'
1	'b'
2	'c'
3	'd'
4	'e'
5	'f'

Nella posizione i si trova il carattere la cui codifica è i stesso (in binario).

Def

C = alfabeto dei simboli presenti nel file
funzione di encoding $e\colon C \to 0,1^*$

Proprietà che deve avere e:

- invertibile → iniettiva: se $a, b \in C, a \neq b \Rightarrow e(a) \neq e(b)$;
- ammettere algoritmi efficienti: e, e^{-1}.

Una funzione di encoding che associa ad ogni carattere di C una stringa di 0 e 1 della stessa lunghezza si chiama **fixed length encoding**.

Finora ho ignorato la frequenza dei caratteri.

Idea Associare a caratteri più frequenti codeword più corte e, di conseguenza, a caratteri meno frequenti codeword più lunghe.

	a	b	c	d	e	f
frequenza (%)	45	13	12	16	9	5
codeword	0	00	01	1	100	101

Problema $e(aa) = e(b)$
Non sono come decodificare in modo univoco perchè esistono delle codeword che sono prefissi di altre codeword.

La codifica che sto cercando deve:

- avere lunghezza variabile della codeword;
- affinchè si possa avere l'invertibilità, il codice deve essere libero da prefissi (**codice prefisso**[1]), cioè $\nexists\ a, b \in C : e(a)$ è un prefisso di $e(b)$.

Soluzione

	a	b	c	d	e	f
frequenza (%)	45	13	12	16	9	5
codeword	0	101	100	111	1101	1100

Questo è un codice a lunghezza variabile libero da prefissi ed è ottimo tra tutti i codici che associano una codeword ad ogni singolo carattere.

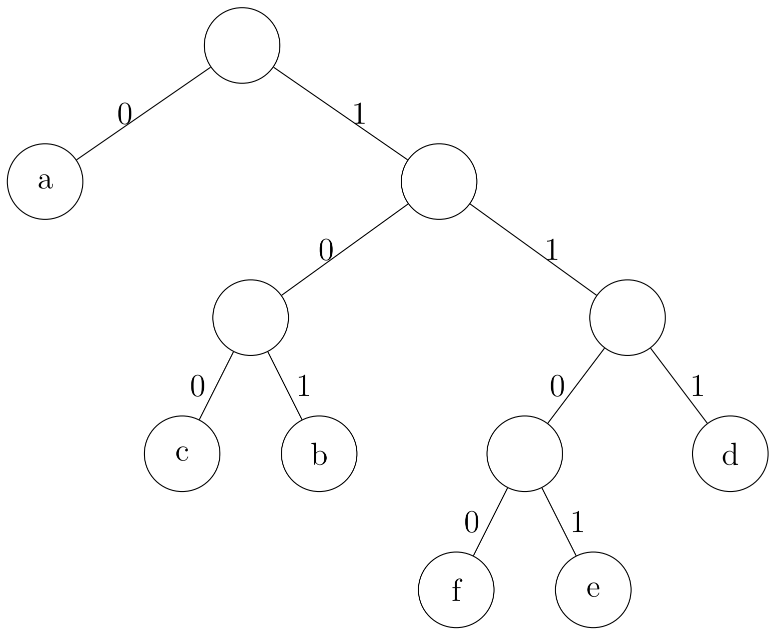
Spazio occupato da questo encoding:

$$100\text{K}\left(\frac{45}{100}\cdot 1+\frac{13}{100}\cdot 3+\frac{12}{100}\cdot 3+\frac{16}{100}\cdot 3+\frac{9}{100}\cdot 4+\frac{5}{100}\cdot 4\right) = 224 \text{ Kbit}$$

300 Kbit $\rightarrow$ risparmio $\approx 25\%$

Serve memorizzare dell'informazione aggiuntiva per la decodifica:

[1] Attenzione: con il termine "codice prefisso" si intende un codice senza prefissi.



Stringa codificata: 110001001101
Stringa decodificata: face

Un qualsiasi codice binario può essere rappresentato in modo compatto con un albero binario.

Un codice prefisso è associato a un **albero di codifica** T in cui i caratteri da codificare appaiono tutti alle foglie.

$\forall\ c \in C$ ho $f(c)$ frequenza

Definisco

$$d_T(c) = \text{profondità di c in T}$$

$d_T(\text{'a'}) = 1$
$d_T(\text{'b'}) = d_T(\text{'c'}) = d_T(\text{'d'}) = 3$
$d_T(\text{'e'}) = d_T(\text{'f'}) = 4$

La profondità corrisponde alla lunghezza della codeword associata.

Funzione di costo:

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

La dimensione del file compresso è

$$|F_c| = \frac{B(T) \cdot |F|}{100}$$

dove

$$|F| = \text{taglia del file compresso (in \# di caratteri)}$$

$B(T)$, a meno di una costante, è il fattore di compressione $\min B(T)$.

Proprietà Un albero ottimo è sempre pieno (cioè i nodi interni hanno due figli).

Spazio di problemi: spazio di alberi pieni con n foglie ($n = |C|$)

Idea Prendo due nodi con frequenza minore, gli unisco in un nodo che avrà come frequenza la somma delle due, ripeto $n-1$ volte.
In questo modo, i nodi con la frequenza maggiore verranno aggiunti per ultimi alla cima dell'albero, quindi avranno una profondità bassa, ovvero la lunghezza dalla codeword corta, proprio come volevo.

Q: coda di priorità (Heap) di nodi con chiave $f(z)$

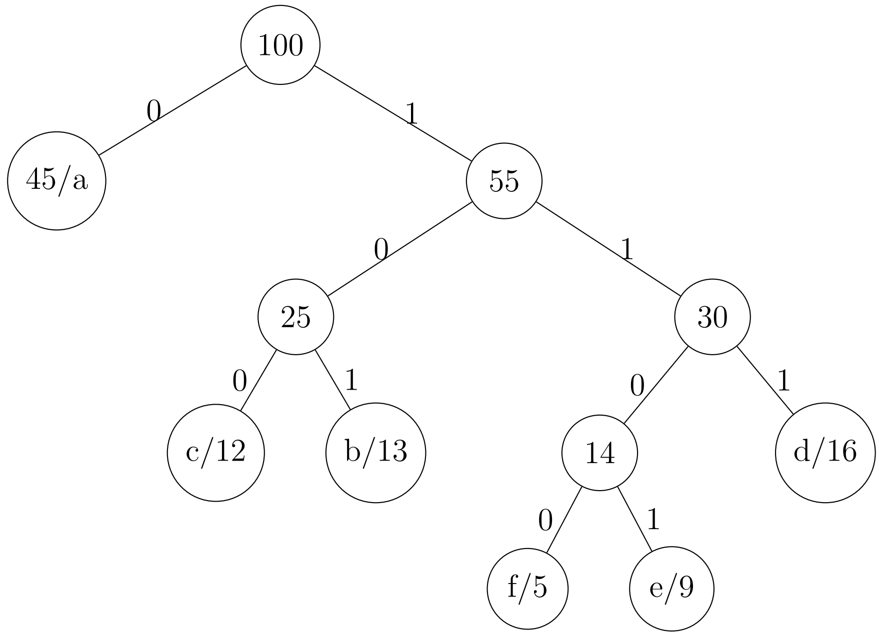
Attributi di un nodo:
- $z.left$
- $z.right$
- $z.f$

Pseudocodice

```
HUFFMAN(C, f)
n = |C|
Q = ∅
for each c ∈ C // inizializzazione
    z = NEW-NODE() // crea un nuovo nodo
    z.f = f(c)
    z.left = NIL
    z.right = NIL
    INSERT(Q, z) // Θ(log n)
for i = 1 to n − 1
    x = EXTRACT-MIN(Q)
    y = EXTRACT-MIN(Q)
    z = NEW-NODE()
    z.left = x
    z.right = y
    z.f = x.f + y.f
    INSERT(Q, z)
```

Complessità $\Theta\left(\sum_{i=1}^{n-1} \log(n-i)\right) = \Theta(n \log n)$

Esempio Riprendiamo l'esempio iniziale e applichiamo l'algoritmo.



	a	b	c	d	e	f
codeword	0	101	100	111	1101	1100

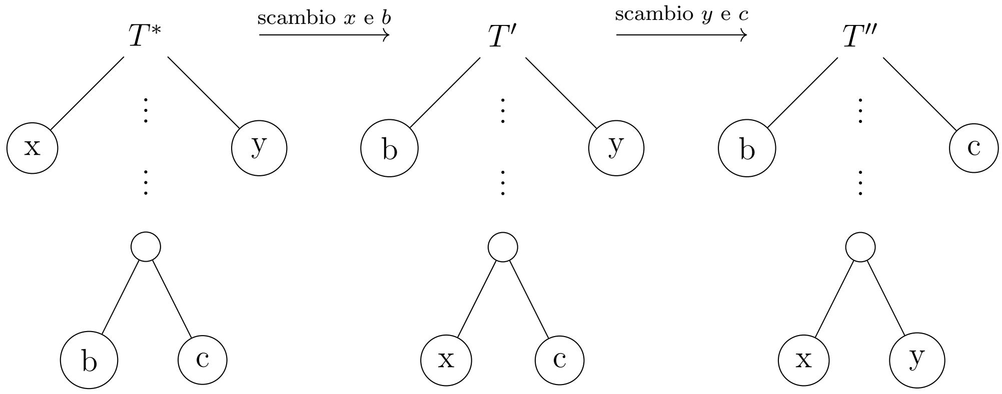
$$B(T) = \left(\frac{45}{100} \cdot 1 + \frac{13}{100} \cdot 3 + \frac{12}{100} \cdot 3 + \frac{16}{100} \cdot 3 + \frac{9}{100} \cdot 4 + \frac{5}{100} \cdot 4\right) = 224$$

8.3.1 Proprietà di Scelta Greedy

Sia C un alfabeto e siano x e y i caratteri in C di frequenza minore.
Allora esiste un codice prefisso ottimo T in cui x e y sono foglie attaccate alo stesso padre (sorelle).

Dimostrazione
Sia T^* una soluzione ottima arbitraria $\Rightarrow B(T^*)$ è minima.
Non so dove siano x e y in T^*
Siano b e c le foglie sorelle di profondità massima in T^*

1. $d_{T^*}(b) = d_{T^*}(c) \geq d_{T^*}(x), d_{T^*}(y)$
2. $f(x) \leq f(b), \quad f(y) \leq f(c)$



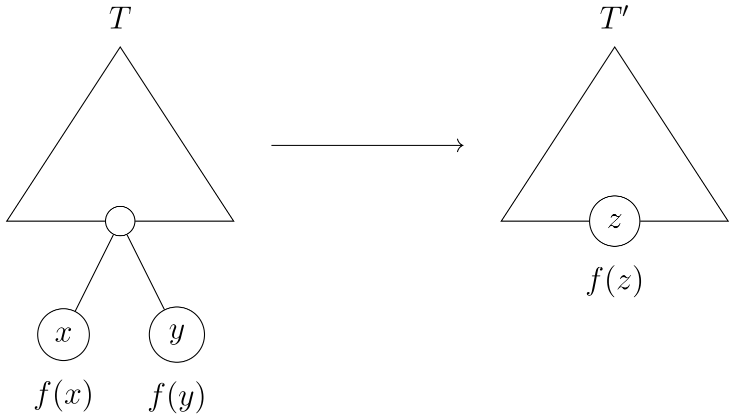
$T^* \to T'$
$B(T^*) \geq B(T')$
$B(T^*) - B(T') \geq 0$

$$\begin{aligned}
B(T^*) - B(T') &= \sum_{c \in C} f(c)\, d_{T^*}(c) - \sum_{c \in C} f(c)\, d_{T'}(c) \\
&= f(b)\, d_{T^*}(b) + f(x)\, d_{T^*}(x) - f(b)\, \underbrace{d_{T'}(b)}_{d_{T^*}(b)} - f(x)\, \underbrace{d_{T'}(x)}_{d_{T^*}(x)} \\
&= f(b)\, d_{T^*}(b) + f(x)\, d_{T^*}(x) - f(b)\, d_{T^*}(b) - f(x)\, d_{T^*}(x) \\
&= f(b)\big(d_{T^*}(b) - d_{T^*}(x)\big) - f(x)\big(d_{T^*}(b) - d_{T^*}(x)\big) \\
&= \Big(\underbrace{f(b) - f(x)}_{\geq 0 \text{ perchè } f(x) \leq f(b)}\Big)\Big(\underbrace{d_{T^*}(b) - d_{T^*}(x)}_{\geq 0 \text{ per il punto 1}}\Big) \geq 0
\end{aligned}$$

$T' \to T''$ si dimostra analogamente.

8.3.2 Proprietà di Sottostruttura Ottima

Sia T un codice prefisso ottimo che contiene la scelta greedy (sui caratteri x e y).
Sia z un nuovo carattere (cioè $\notin C$) con $f(z) = f(x) + f(y)$.
Allora il codice prefisso $T' = T \setminus \{x, y\}$ è ottimo per $C \setminus \{x, y\} \cup \{z\}$



T ha n foglie.
T' ha $n-1$ foglie.
Cerco una relazione tra $B(T)$ e $B(T')$.

Osservazione

1. $\forall\ c \in C \setminus \{x, y\},\ c \neq z : d_{T'}(c) = d_T(c)$
2. $d_{T'}(z) = d_T(x) - 1$

Dimostrazione

$$
\begin{aligned}
B(T) &= \sum_{c \in C} f(c)\, d_T(c) && \big(d_T(x) = d_T(y)\big) \\
&= \sum_{c \in C \setminus \{x,y\}} f(c)\, d_T(c) + \big(f(x) + f(y)\big) d_T(x) \\
&= \sum_{c \in C \setminus \{x,y\}} f(c)\, d_T(c) + \big(\underbrace{f(x) + f(y)}_{f(z)}\big)\big(\underbrace{d_T(x) - 1}_{d_{T'}(z)}\big) + \big(f(x) + f(y)\big) \\
&= \sum_{c \in C \setminus \{x,y\} \cup \{z\}} f(c)\, d_{T'}(c) + f(x) + f(y) \\
&= B(T') + f(x) + f(y)
\end{aligned}
$$

Suppongo per assurdo che T non sia il codice prefisso ottimo per il sottoproblema $C \setminus \{x, y\} \cup \{z\}$.
Allora $\exists\ T''$ di costo strettamente inferiore.

$$B(T) = \underbrace{B(T'')}_{<B(T')} + f(x) + f(y)$$

Assurdo!