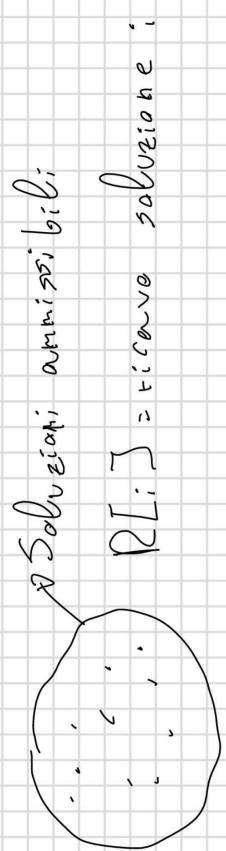


## Problemi di ottimizzazione

## Selezione di attività: esempio



Voglio trovare i  $\ell$  ricavo migliori  
tipi di problemi

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	3	5	5	8	2	1	0	3	6	8	12
$f[i]$	5	7	9	11	14	4	6	9	10	12	16

## Selezione di attività (activity selection)

## Una soluzione ottima

- ▷ Input: un insieme  $S$  di  $n$  attività
- ▷ Ogni attività  $i$  ha un tempo di inizio  $s[i]$  e un tempo di fine  $f[i]$ , ovviamente  $f[i] > s[i]$
- ▷ Output: un insieme  $A \subseteq S$  massimale di attività a due a due compatibili
- ▷ Le attività  $i$  e  $j$  sono compatibili se  $f[i] \leq s[j]$  oppure  $f[j] \leq s[i]$

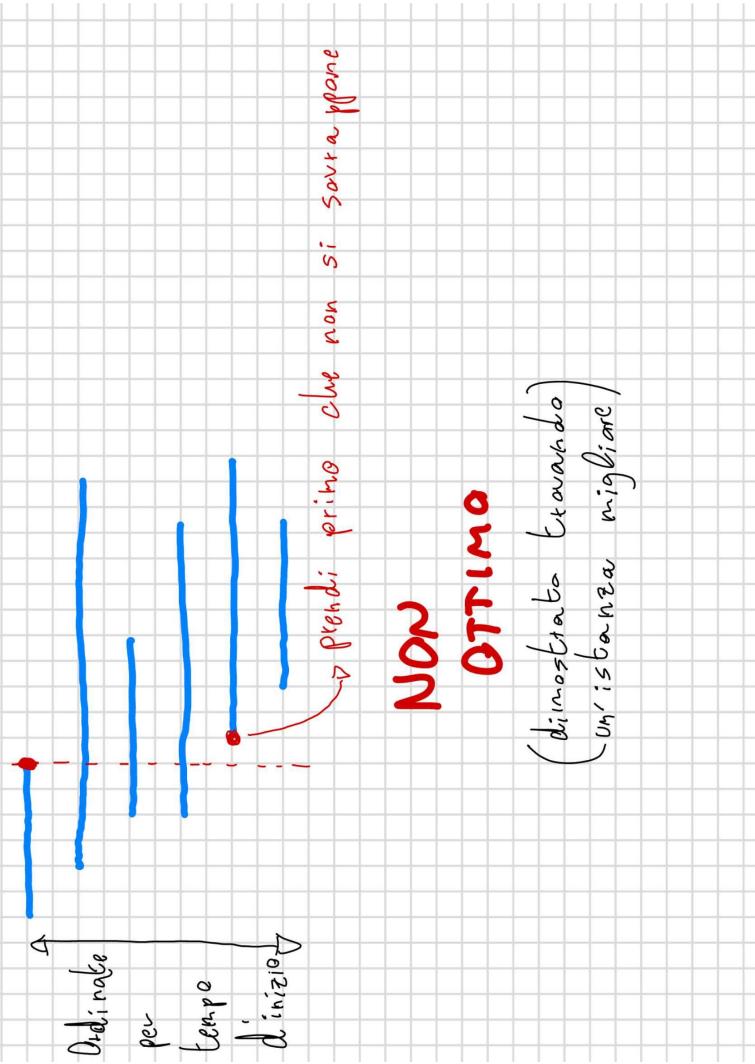
non si sovrappongono

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	3	5	5	8	2	1	0	3	6	8	12
$f[i]$	5	7	9	11	14	4	6	9	10	12	16

$$S_{ott} = \{a_2, a_4, a_6, a_{11}\} \leftarrow \text{Non unica}$$

Soluzione ottimale: max numero di attività

## Esempio



Attività ordinate in base ai tempi di inizio  
attività crescenti

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	0	1	2	3	3	5	5	6	8	8	12
$f[i]$	6	4	14	5	9	7	9	10	11	12	16

## Selezione di attività: primo metodo

S-GREEDY-ACTIVITY-SELECTOR( $s, f$ )

```

1  ( $s, f$ ) = SORT( $s, f$ ) by  $s$ 
2   $n = s.length$ 
3   $A = \{a_1\}$ 
4   $k = 1$ 
5  for  $m = 2$  to  $n$ 
6    if  $s[m] \geq f[k]$ 
7       $A = A \cup \{a_m\}$ 
8     $k = m$ 
9  return  $A$ 

```

## Esempio

Attività ordinate in base ai tempi di inizio  
attività crescenti

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	0	1	2	3	3	5	5	6	8	8	12
$f[i]$	6	4	14	5	9	7	9	10	11	12	16

$S = \{a_1, a_9, a_{11}\}$

## Selezione di attività: secondo metodo (ottimo)

### Esempio

F-GREEDY-ACTIVITY-SELECTOR( $s, f$ )

```

1  ( $s, f$ ) = SORT( $s, f$ ) by  $f$ 
2   $n = s.length$ 
3   $A = \{a_1\}$ 
4   $k = 1$ 
5  for  $m = 2$  to  $n$ 
6    if  $s[m] \geq f[k]$  non sovrappone
7       $A = A \cup \{a_m\}$ 
8     $k = m$ 
9  return  $A$ 
```

Attività ordinate in base ai tempi di fine attività

crescenti

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	1	3	0	5	3	5	6	8	2	12	2
$f[i]$	4	5	6	7	9	9	10	11	12	14	16

$$S = \{a_1, a_4, a_8, a_{11}\}$$

### Esempio

Algoritmo greedy generico

Attività ordinate in base ai tempi di fine attività  
crescenti

ALGORITMO-GREEDY-GENERICO( $P$ )

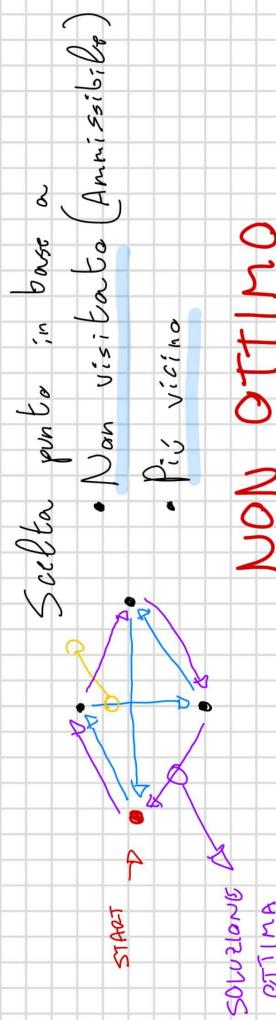
```

1   $S = \emptyset$ 
2  repeat
3     $s = \text{SCELTA-GREEDY-SAFE}(S, P)$ 
4     $S = S \cup \{s\}$ 
5  until  $S$  è completa
6  return  $S$ 
```

$i$	1	2	3	4	5	6	7	8	9	10	11
$s[i]$	1	3	0	5	3	5	6	8	8	2	12
$f[i]$	4	5	6	7	9	9	10	11	12	14	16

## Esempio: Commessa-Viaggiatore

Trovare  $\ell$  circuito più corto dai punti



## Soluzione dell'algoritmo greedy generico

- ▷ Se al passo 3 dell'algoritmo greedy generico si effettua una scelta ammissibile safe, l'algoritmo produrrà una soluzione ammissibile (primo metodo)
- ▷ Se al passo 3 dell'algoritmo greedy generico si effettua una scelta ottima safe, l'algoritmo produrrà una soluzione ottima (secondo metodo)

## Scelta safe

- ▷  $S_{am}$  insieme delle soluzioni ammissibili
- ▷  $S_{ott}$  insieme delle soluzioni ottime
- ▷  $S_{ott} \subseteq S_{am}$
- ▷ Sia  $S$  un sottoinsieme di una soluzione ottima,  $s$  è una **scelta ottima safe** per  $S$  se  $S \cup \{s\}$  è sottoinsieme di almeno una soluzione ottima
- ▷ Sia  $S$  un sottoinsieme di una soluzione ammissibile,  $s$  è una **scelta ammissibile safe** per  $S$  se  $S \cup \{s\}$  è sottoinsieme di almeno una soluzione ammissibile

## Greedy-Activity-Selector: correttezza

- ▷ Siano  $S_k = \{a_1, \dots, a_k\}$  le prime  $k$  attività scelte dall'algoritmo. Si noti che  $k$  può anche essere uguale a 0
- ▷ Sia  $S_{ott} = \{a_1, \dots, a_k, a_{k+1}, \dots, a_m\}$  una soluzione ottima che contiene  $S_k$ , assumiamo senza perdere di generalità che  $a_{k+1}$  termini prima di  $a_{k+2}, \dots, a_m$ . Questo implica anche che  $a_{k+1}$  termini prima dell'inizio di  $a_{k+2}, \dots, a_m$
- ▷ L<sub>d</sub> ammissibile  $\Rightarrow$  non si sara pronti
- ▷ Sia  $s$  la scelta greedy fatta dall'algoritmo, supponiamo che  $S_k \cup \{s\}$  non sia contenuta in alcuna soluzione ottima.

## Greedy-Activity-Selector: correttezza

## Proprietà della sottostruttura ottima

- ▷ Sia  $S'_{ott}$  la soluzione ottenuta da  $S_{ott}$  rinnuovendo  $a_{k+1}$  e aggiungendo  $s$ 
  - ▷  $S'_{ott}$  è ammibile in quanto  $s$  inizia dopo la fine di  $a_k$  e termina prima dell'inizio di  $a_{k+2}$  (per come è stata scelta)  $\therefore$ 
    - ▷  $S'_{ott}$  inoltre contiene  $s$  e ha la stessa cardinalità di  $S_{ott}$  quindi è ottima! Assurdo!
    - ▷ Possiamo quindi concludere che l'algoritmo Greedy-Activity-Selector produca una soluzione ottima

## Selezione di attività: costo computazionale

## Selezione di attività: proprietà della sottostruttura ottima

- ▷ Sia  $S_{ott} = \{a_1, a_2, \dots, a_{k-1}, a_k, a_{k+1}, \dots, a_m\}$  una soluzione ottima (ordinata in base ai tempi) per l'insieme  $S$  di attività
  - ▷ Sia  $S_L$  l'insieme delle attività in  $S$  che terminano prima dell'inizio di  $a_k$  e  $S_R$  l'insieme delle attività in  $S$  che iniziano dopo la fine di  $a_k$ .
  - ▷  $S - (S_L \cup S_R)$  è l'insieme delle attività incompatibili con  $a_k$
  - ▷  $S^L_{ott} = \{a_1, a_2, \dots, a_{k-1}\}$  è ottima per  $S_L$
  - ▷  $S^R_{ott} = \{a_{k+1}, \dots, a_m\}$  è ottima per  $S_R$
  - ▷  $S^L_{ott} \cup S^R_{ott} \cup \{a_k\}$  è ottima per  $S$

# Greedy-Activity-Selector ricorsivo

## Esempio

```
RECURSIVE-ACTIVITY-SELECTOR( $S$ )
1    $s =$  attività che termina per prima in  $S$ 
2    $S' = S -$  attività incompatibili con  $s$ 
3   return  $\{s\} \cup$  RECURSIVE-ACTIVITY-SELECTOR( $S'$ )
```

- ▷  $\Sigma = \{a, b, c, d, e, f\}$
- ▷  $|s| = 100.000$
- ▷ Frequenze (in migliaia):  $a : 45, b : 13, c : 12,$   
 $d : 16, e : 9, f : 5$

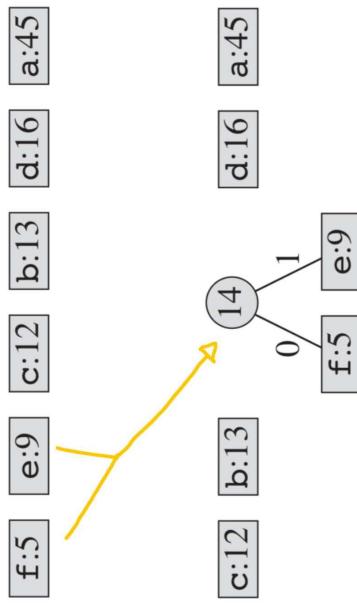
## Compressione dati

Codici: di Huffman

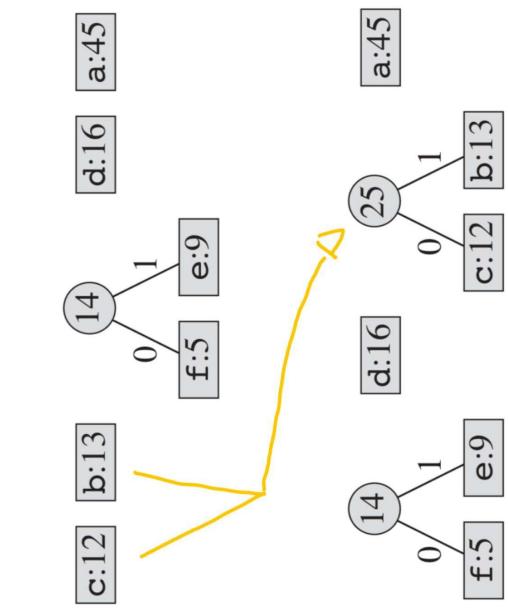
- ▷ Sia  $s$  una stringa di caratteri su un certo alfabeto  $\Sigma$
- ▷ Algoritmo  $C$  di compressione e  $D$  di decompressione
- ▷  $C(s) = s'$  e  $D(s') = s$
- ▷ Obiettivo: minimizzare  $|s'|$

## Esempio

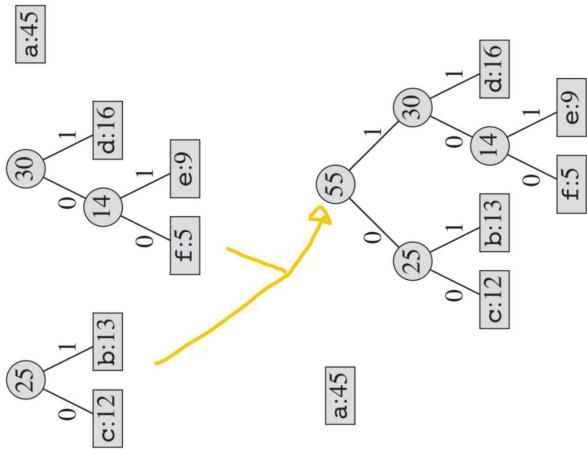
### FREQUENZA



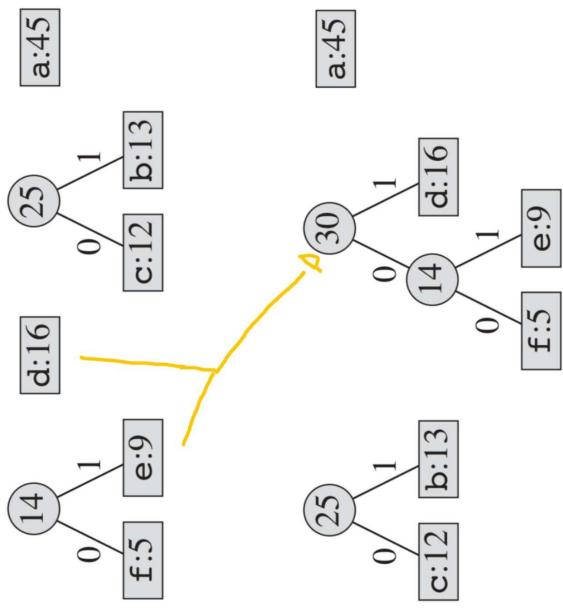
## Esempio



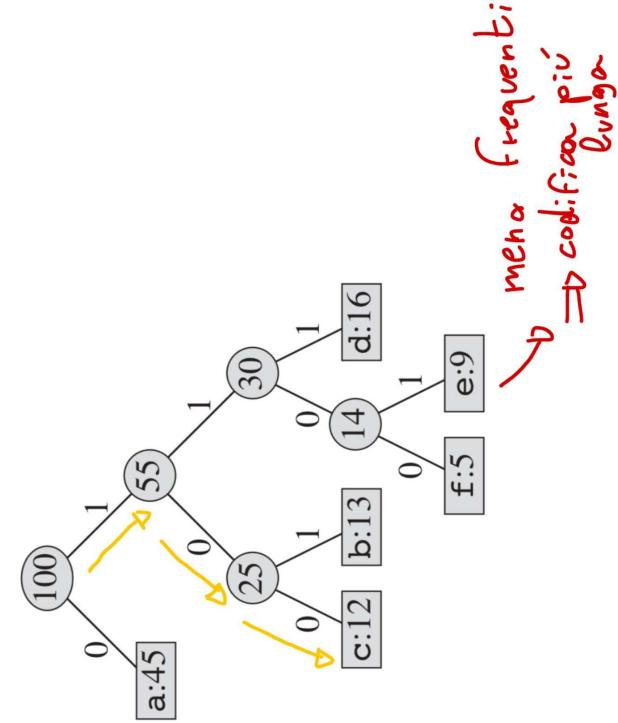
## Esempio



## Esempio



## Esempio



$C = 100$

## Codifica

## Huffman code

HUFFMAN( *frequenze*)

simbolo	Huffman	standard
a	0	000
b	101	001
c	100	010
d	111	011
e	1101	100
f	1100	101

*prefissi unici  $\Rightarrow$  lunghezza parola diversa*

0 10 1100  
a b c

Codifica: risultati (in migliaia di bit)

```
1   n = |C|  
2   Q = C  
3   for i = 1 to n - 1  
4       allocate a new node z  
5       z.left = x = EXTRACT-MIN(Q)  
6       z.right = y = EXTRACT-MIN(Q) - 2  
7       z.freq = x.freq + y.freq  
8       INSERT(Q, z)  
9   return EXTRACT-MIN(Q) // root of the tree
```

+ 1

Esercizio

Dimostrare che l'algoritmo di Huffman è ottimo ovvero fornisce la codifica più corta

Esercizio

Calcolare il costo computazionale dell'algoritmo di Huffman

Esercizio

Dimostrare che la codifica con l'algoritmo di Huffman ammette una decodifica univoca

Risultato della codifica con Huffman

$$100 * 3 = 300$$

Risultato della codifica con Huffman

$$45 * 1 + 13 * 3 + 12 * 3 + 16 * 3 + 9 * 4 + 5 * 4 = 224$$

Risparmio: 34%