# Algoritmi golosi

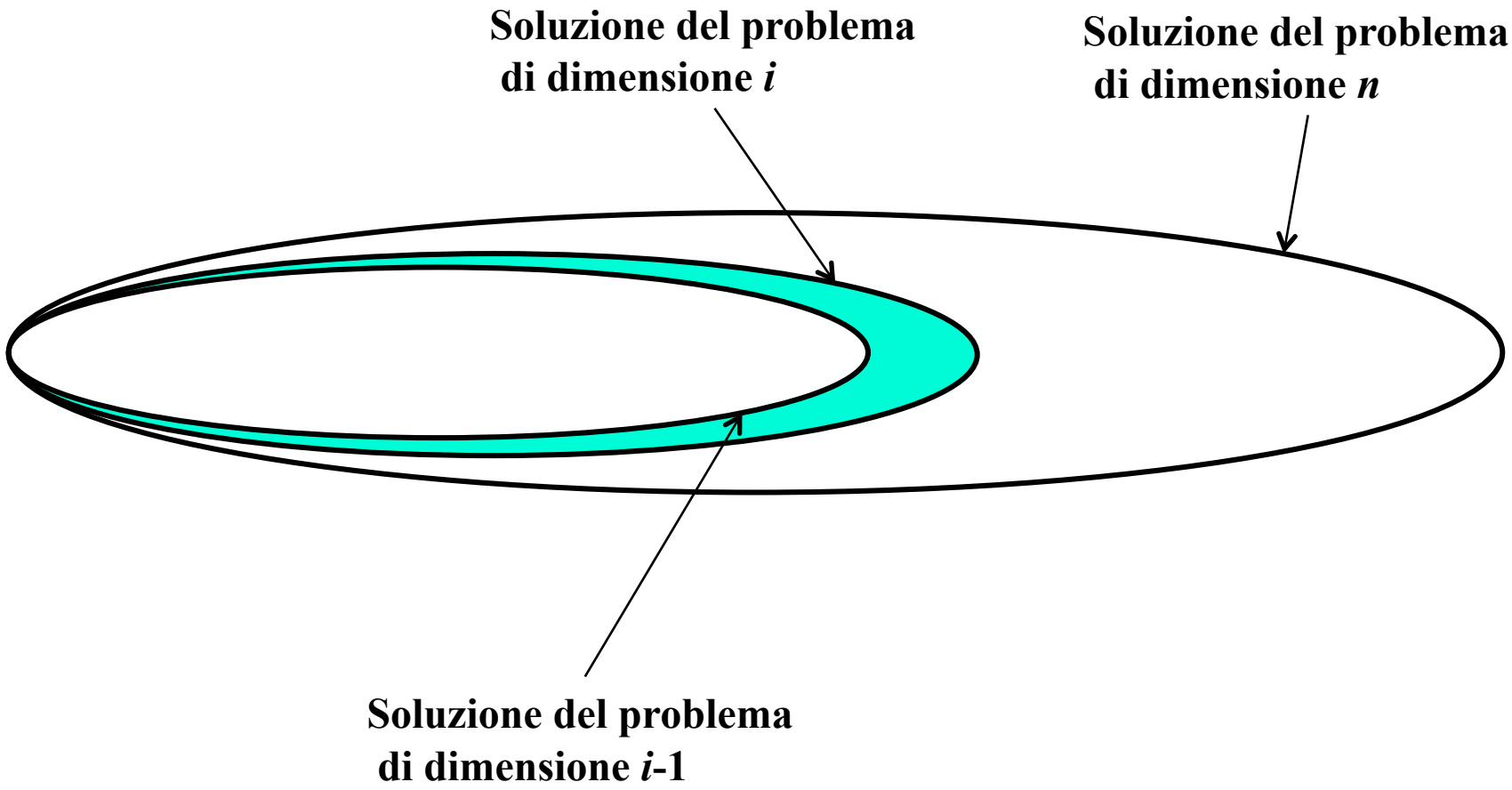
Tecniche di soluzione dei problemi viste finora:

- Metodo iterativo
- Divide et impera
- Programmazione dinamica

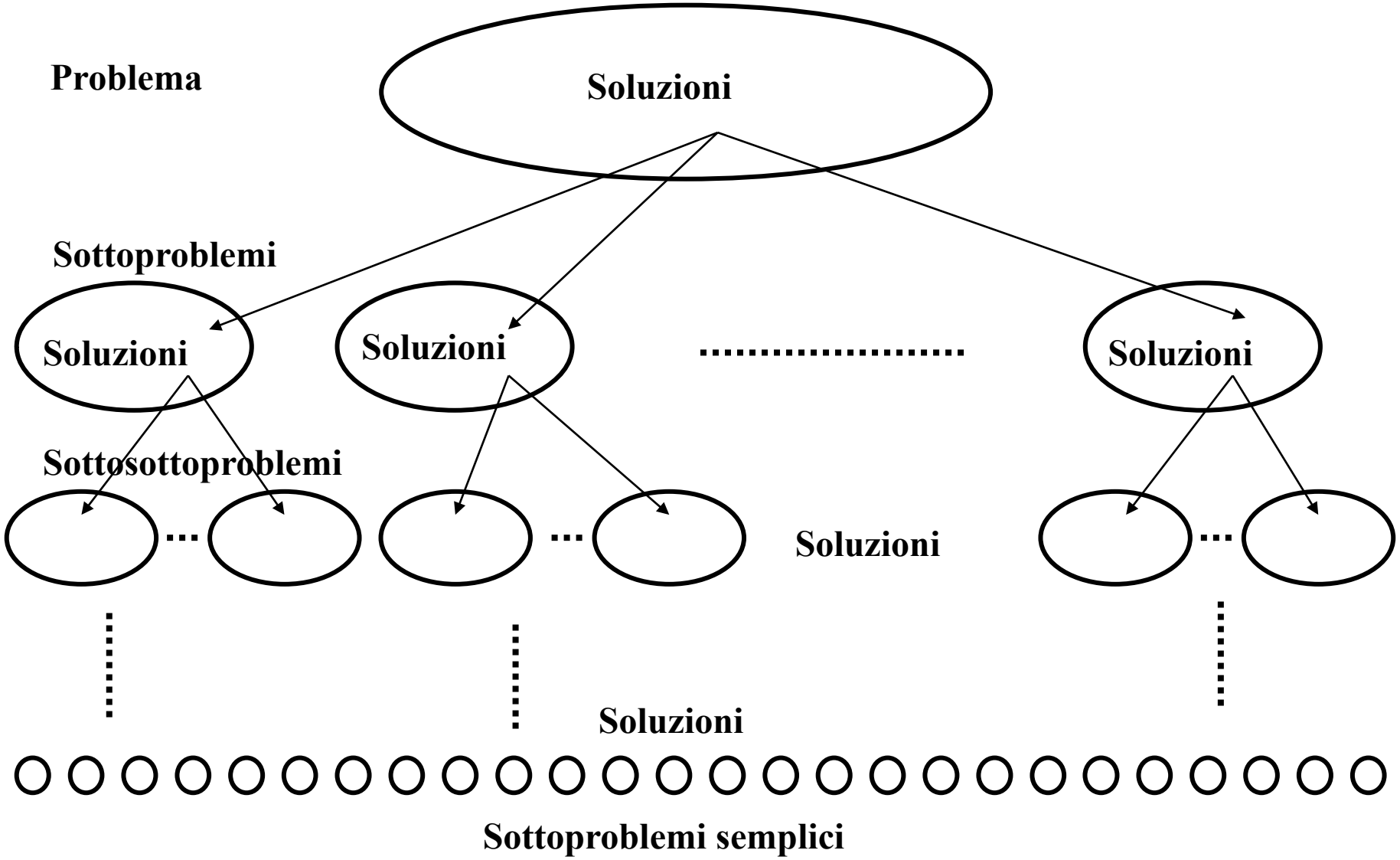
Nuova tecnica:

- Algoritmi golosi

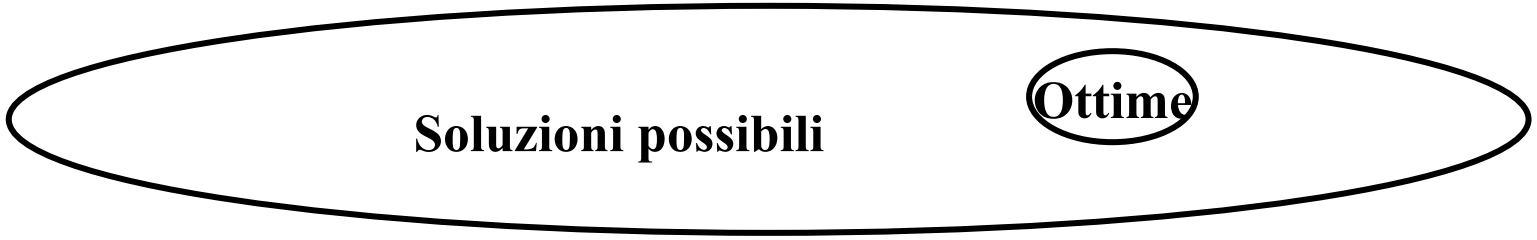
# Metodo iterativo



# Divide et impera



In un problema di ottimizzazione abbiamo un insieme generalmente molto grande di soluzioni e dobbiamo scegliere tra di esse una soluzione che sia ottima in qualche senso (costo minimo, valore massimo, lunghezza minima, ecc.)



Possiamo risolvere un problema di questo tipo con una ***enumerazione esaustiva***

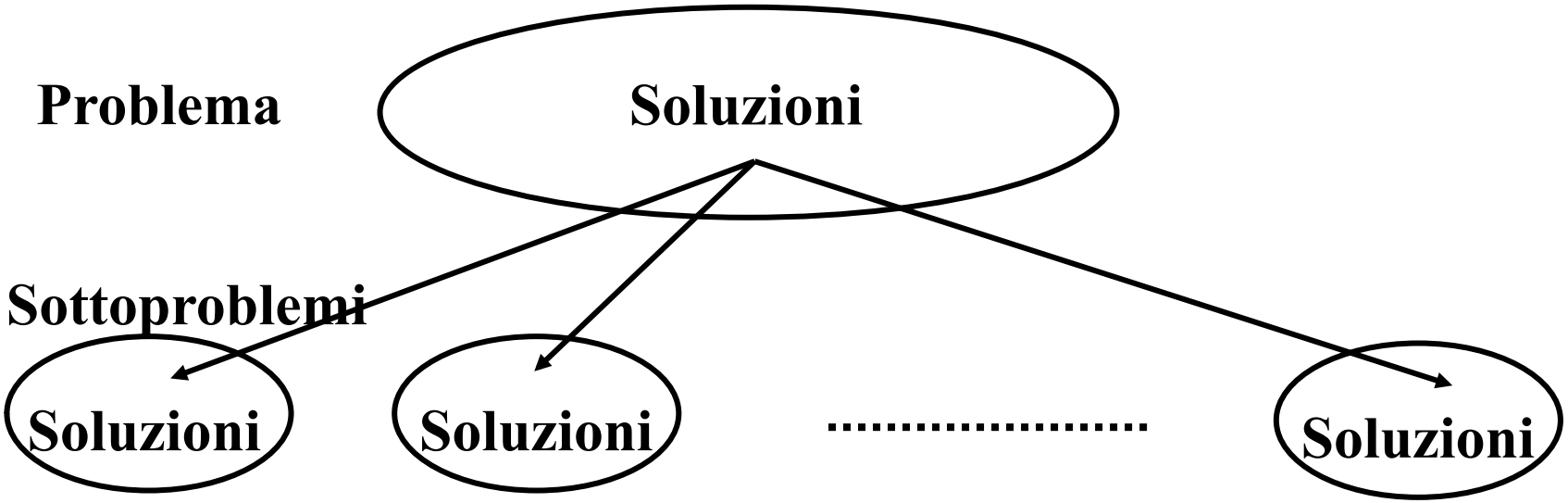
- si generano tutte le soluzioni possibili,

- si calcola il costo di ciascuna di esse

- e infine se ne seleziona una di ottima.

Purtroppo l'insieme di soluzioni è generalmente molto grande (spesso esponenziale nella dimensione dell'input) per cui una enumerazione esaustiva richiede tempo esponenziale.

Molto spesso le soluzioni di un problema di ottimizzazione si possono costruire estendendo o combinando tra loro soluzioni di sottoproblemi.



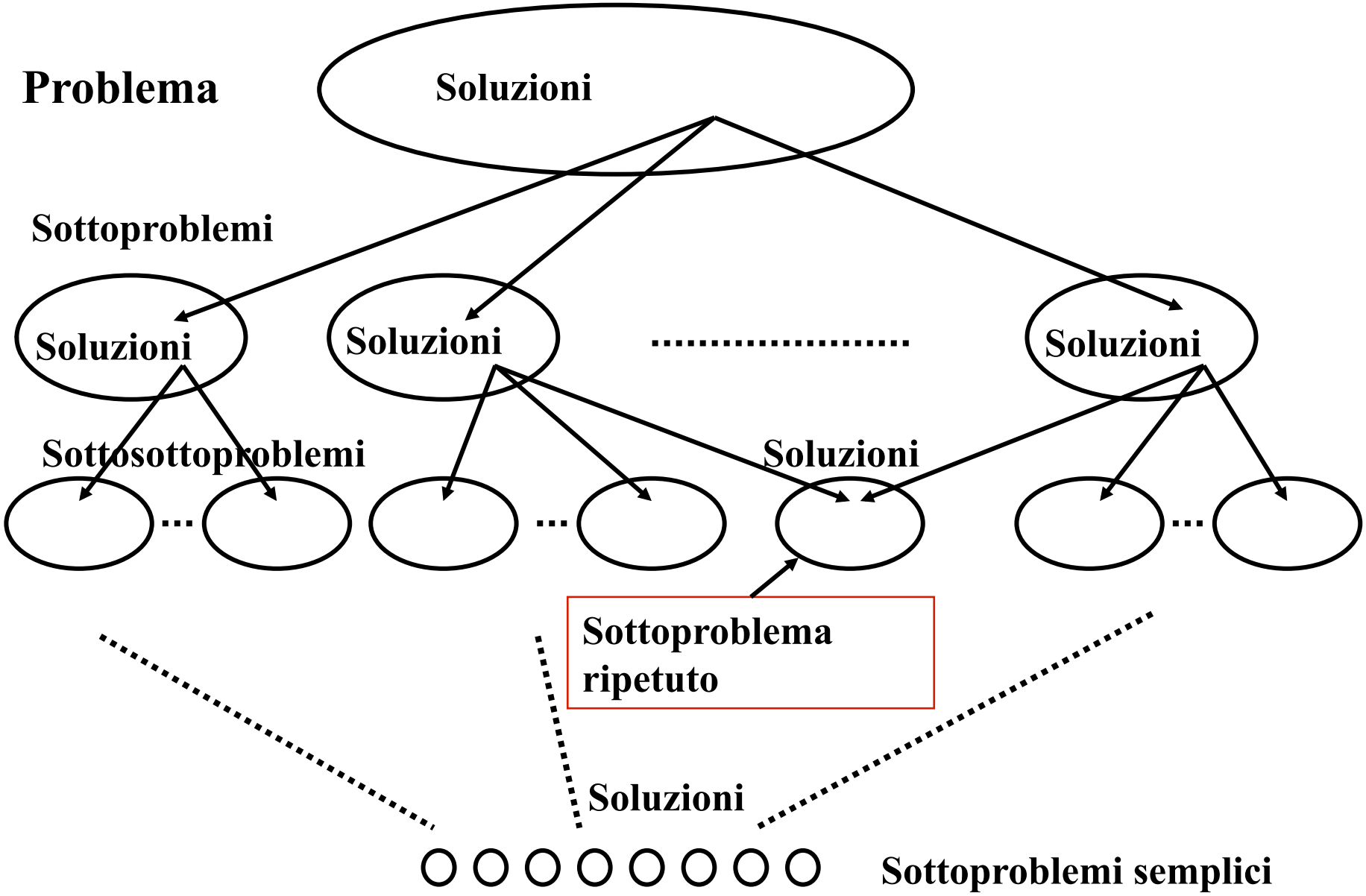

Esempio: Problema Torino-Trieste.

Sottoproblemi: Torino-Asti, Asti-Trieste; Torino -Novara, Novara-Trieste, ecc.

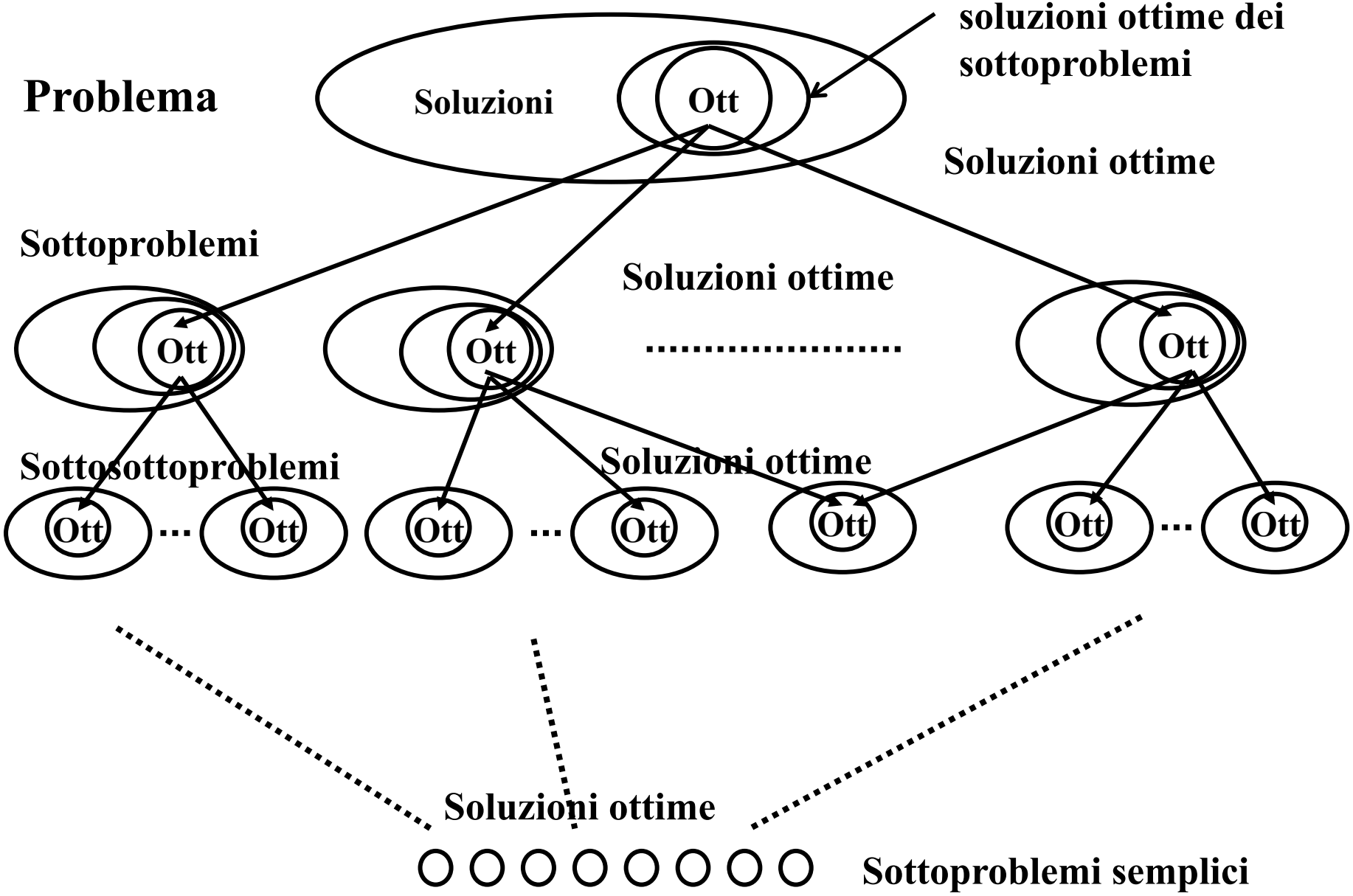
Abbiamo visto che perché la ***programmazione dinamica*** sia vantaggiosa rispetto all'enumerazione esaustiva bisogna che siano soddisfatte due condizioni:

1. Esistenza di sottoproblemi ripetuti.
2. Sottostruttura ottima.

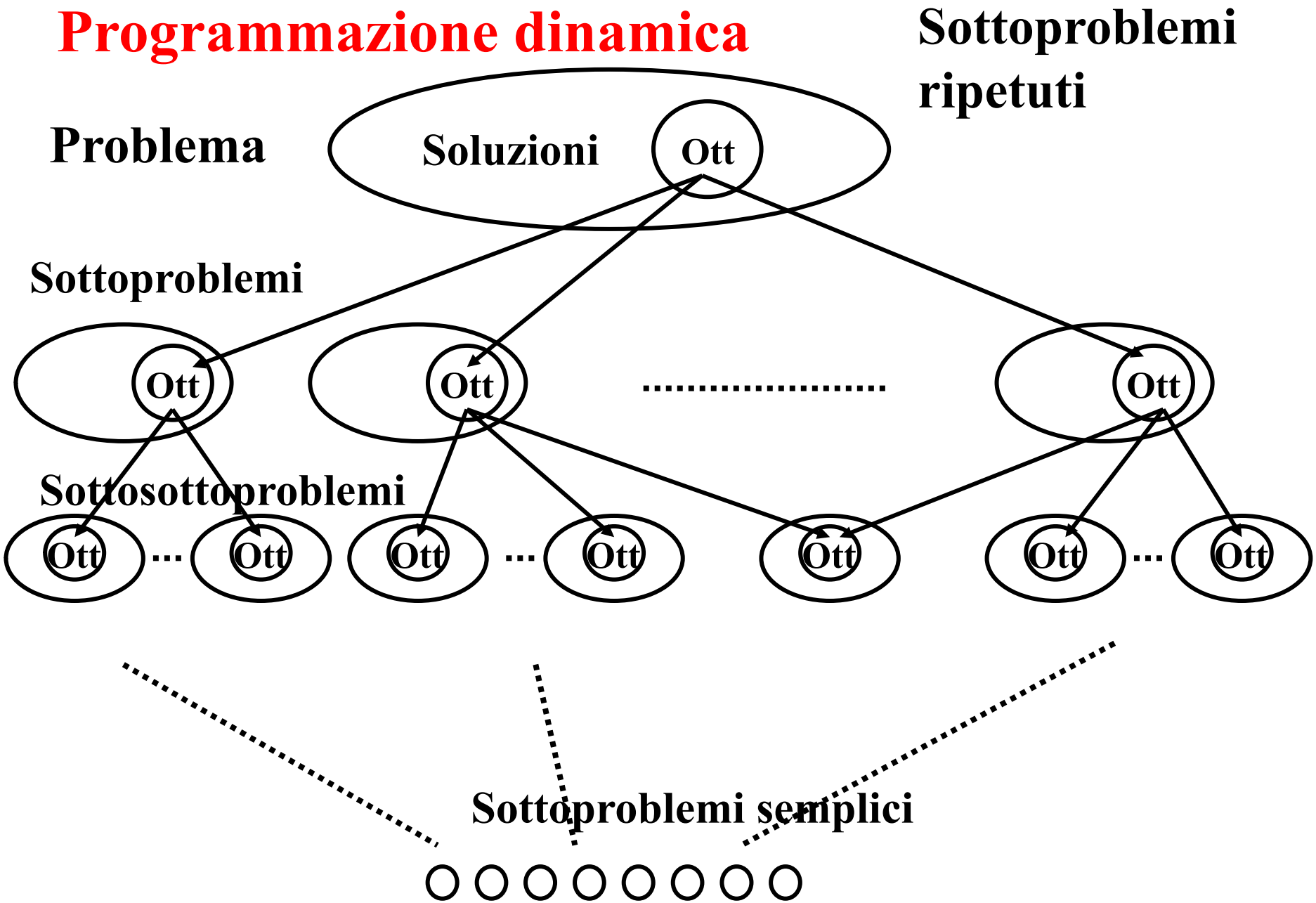
# Sottoproblemi ripetuti



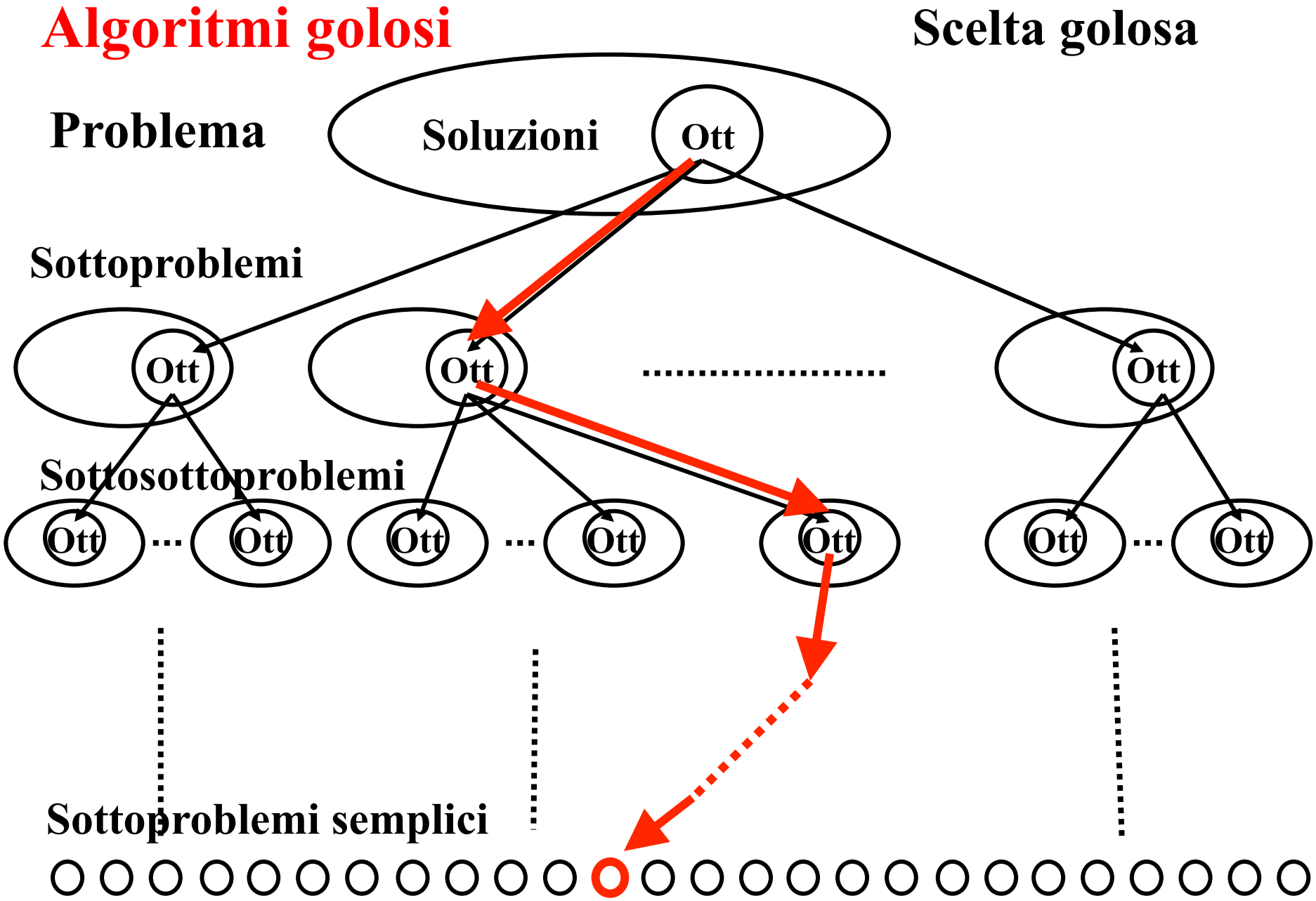
# Sottostruttura ottima



# Programmazione dinamica



# Algoritmi golosi



1) ogni volta si fa la scelta che sembra migliore localmente.

2) in questo modo per alcuni problemi si ottiene una soluzione globalmente ottima.

# Problema della scelta delle attività

$n$ ***attività*** $a_1,\ldots,a_n$ usano la stessa risorsa (es: lezioni da tenere in una stessa aula).

Ogni attività $a_i$ ha un ***tempo di inizio*** $s_i$ ed un ***tempo di fine*** $f_i$ con $s_i < f_i$.

$a_i$ occupa la risorsa nell'intervallo di tempo $[s_i, f_i)$.

$a_i$ ed $a_j$ sono ***compatibili*** se $[s_i, f_i)$ ed $[s_j, f_j)$ sono disgiunti.

**Problema:** scegliere il massimo numero di attività compatibili.

# Storiella Golosa

## Personaggi:

**Pinocchio**

**L'algoritmo goloso**

**Il grillo parlante**

**Controlla Pinocchio**

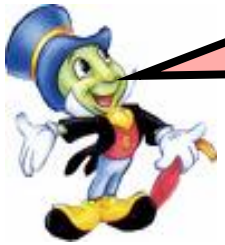
**La fata turchina**

**Conosce il futuro**

**Pinocchio arriva nella Città dei Balocchi e può scegliere i divertimenti che preferisce**

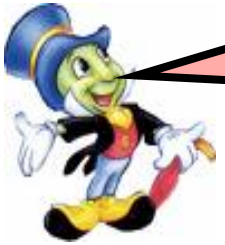
**Ogni divertimento ha un'orario di inizio ed una durata**

**Perciò comincio scegliendo il divertimento che inizia per primo!! Così non perdo tempo.**



**Attenzione Pinocchio!!! Se fai così non è detto che tu possa scegliere il maggior numero di divertimenti**

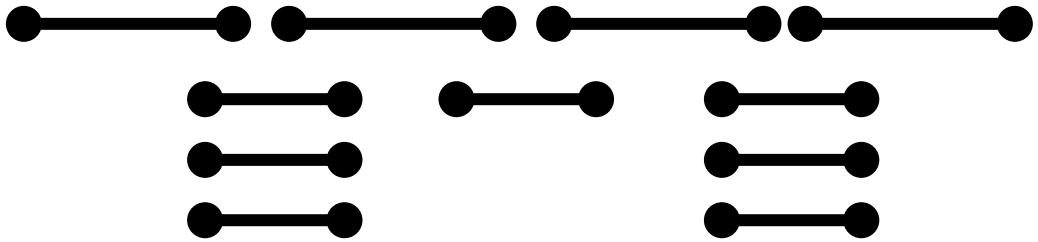
**Allora scelgo il divertimento che dura di meno!!**
**Così mi rimane più tempo per gli altri.**



**Attenzione Pinocchio!!! Anche così non è detto che tu possa scegliere il maggior numero di divertimenti**

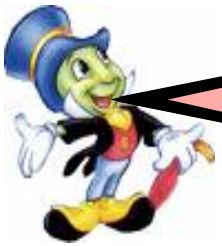
**Uffa!! Ma sei proprio un rompiscatole!!**
**Ora riprovo e se non ti va ancora bene ti schiaccio con il martello.**

**Attenzione Pinocchio!!! Anche così non è detto che tu possa scegliere il maggior numero di divertimenti**



**Io conosco una soluzione ottima ma non la mostro a nessuno.**

**Scelgo il divertimento "*D*" che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.**



**Insegnerò alla fatina come modificare la sua soluzione ottima in modo che contenga il divertimento "*D*".**

**Io conosco una soluzione ottima che contiene "*D*".**

**Ho scelto il divertimento "*D*" che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.**

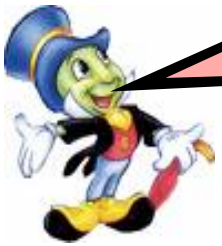
# Primo caso:

**Ora so che la fatina conosce una soluzione ottima che contiene il divertimento "*D*".**

**Io conosco una soluzione ottima ma non la mostro a nessuno.**

**Ho scelto il divertimento "*D*" che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.**

# Secondo caso:



**Cara fatina, se la tua soluzione non contiene il divertimento "*D*" metti "*D*" al posto del primo divertimento.**

$D_1$ $D_2$ ..... $D_m$

$D$ $D_2$ ..... $D_m$

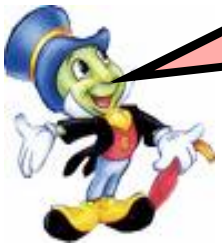
**Io conosco una nuova soluzione ottima che contiene "*D*".**

**Ho scelto il divertimento "*D*" che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.**

**Ora so che la fatina conosce una soluzione ottima che contiene il divertimento "*D*".**

**Io conosco una soluzione ottima che contiene tutti i divertimenti scelti finora da Pinocchio.**

**Ho finito tutti i divertimenti scelti finora. Ora scelgo il divertimento "*D*" che termina per primo tra quelli non ancora iniziati.**



**Insegnerò alla fatina come modificare la sua soluzione ottima in modo che contenga anche "*D*".**

**Io conosco una soluzione ottima che contiene i divertimenti scelti finora compreso il divertimento "*D*".**

**Ho finito tutti i divertimenti scelti finora ed ora ho scelto quel divertimento "*D*" che terminerà per primo tra quelli non ancora iniziati.**

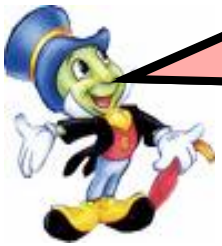
# Primo caso:

**Cara fatina, se la tua soluzione contiene il divertimento "*D*" lasciala invariata.**

**Io conosco una soluzione ottima che contiene tutti i divertimenti scelti finora da Pinocchio.**

**Ho finito tutti i divertimenti scelti finora ed ora ho scelto quel divertimento "*D*" che terminerà per primo tra quelli non ancora iniziati.**

# Secondo caso:



**Cara fatina, se la tua soluzione non contiene il divertimento "*D*" mettilo al posto del primo divertimento che nella tua soluzione segue quelli già scelti da Pinocchio.**

**ora attuale**

$D_1$ ..... $D_k$ $D_{k+1}$ $D_{k+2}$ ..... $D_m$

$D_1$ ..... $D_k$ $D$ $D_{k+2}$ ..... $D_m$

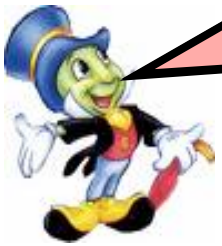
**Io conosco una nuova soluzione ottima che contiene i divertimenti scelti finora da Pinocchio compreso "*D*".**

**Ho finito tutti i divertimenti scelti finora ed ora ho scelto quel divertimento "*D*" che terminerà per primo tra quelli non ancora iniziati.**

**So che la fatina conosce una soluzione ottima che contiene tutti i divertimenti scelti finora da Pinocchio compreso "*D*".**

**Io conosco una soluzione ottima che contiene tutti i divertimenti scelti finora da Pinocchio.**

**Ho finito tutti i divertimenti scelti finora ma tutti gli altri sono già iniziati.**



**Quindi la soluzione ottima della fatina non contiene altri divertimenti e quelli scelti finora da Pinocchio sono una soluzione ottima.**

# Strategie golose:

**Scegliere l'attività che inizia per prima**

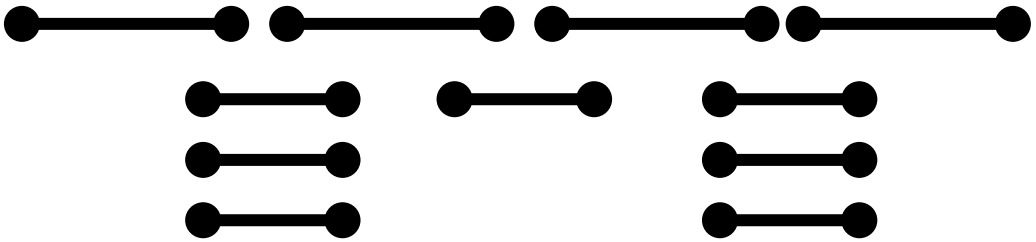
*Non funziona*

**Scegliere l'attività che dura meno tempo**

*Non funziona*

**Scegliere l'attività incompatibile con il minor numero di altre attività**

*Non funziona*



Strategia che funziona:
Scegliere l'attività che termina per prima.

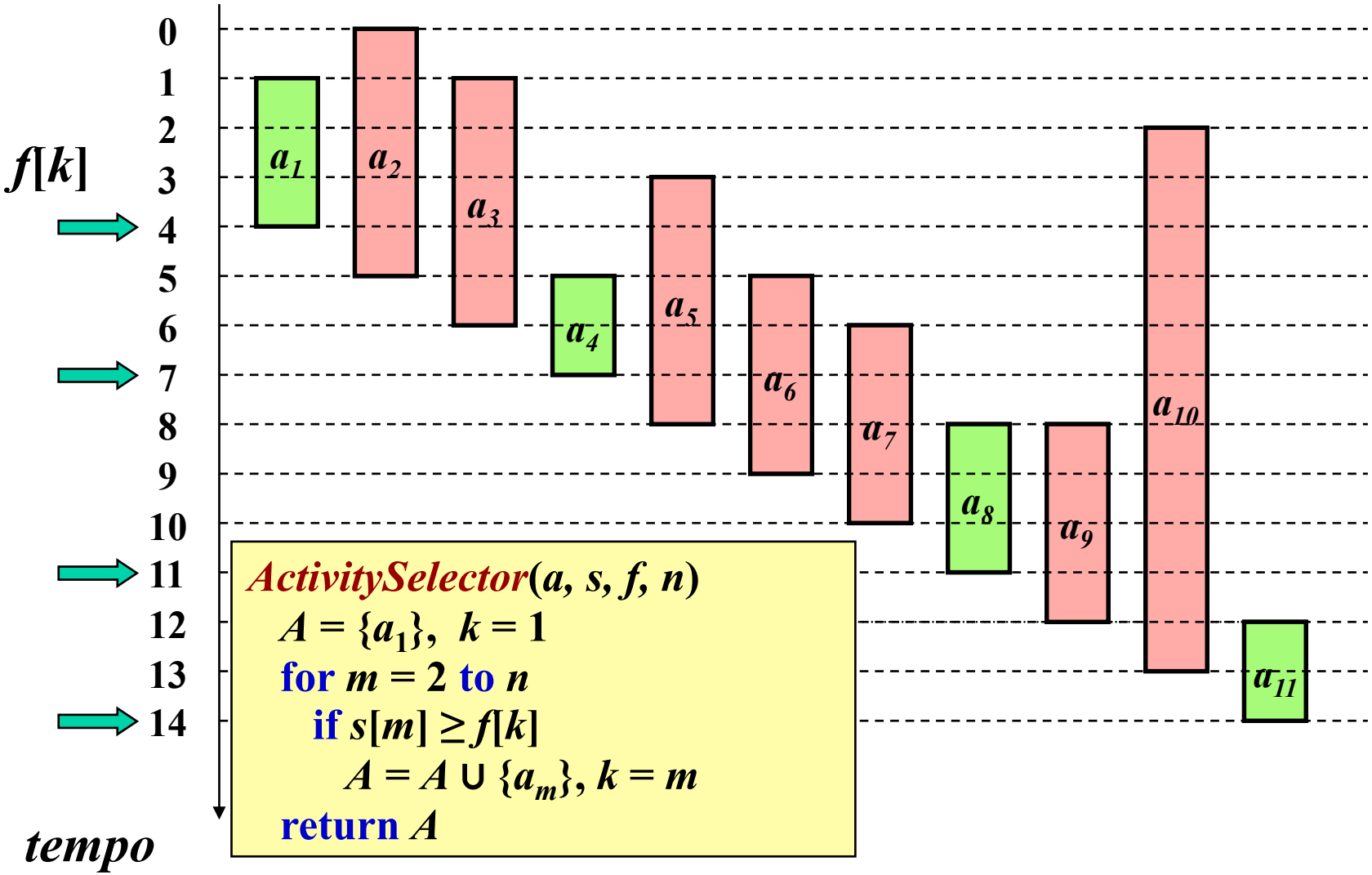
***ActivitySelector*(*Att*)**
***AttScelte* = Ø, *AttComp* = *Att***
**while** ***AttComp* ≠ Ø**
" in ***AttComp*** scegli l'attività '***a***' che termina per prima, aggiungi '***a***' a ***AttScelte*** e togli da ***AttComp*** tutte le attività incompatibili con '***a***'"
**return** ***AttScelte***

Per implementarla supponiamo le attività $a_1,\ldots,a_n$ ordinate per tempo di fine non decrescente $f_1 \leq \ldots \leq f_n$

Altrimenti possiamo ordinarle in tempo $O(n \log n)$

```
ActivitySelector(a, s, f, n)   // f_1 ≤ ... ≤ f_n
  A = {a_1}, k = 1
  for m = 2 to n
    if s[m] ≥ f[k]
      A = A ∪ {a_m}, k = m
  return A
```

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 0 | 1 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



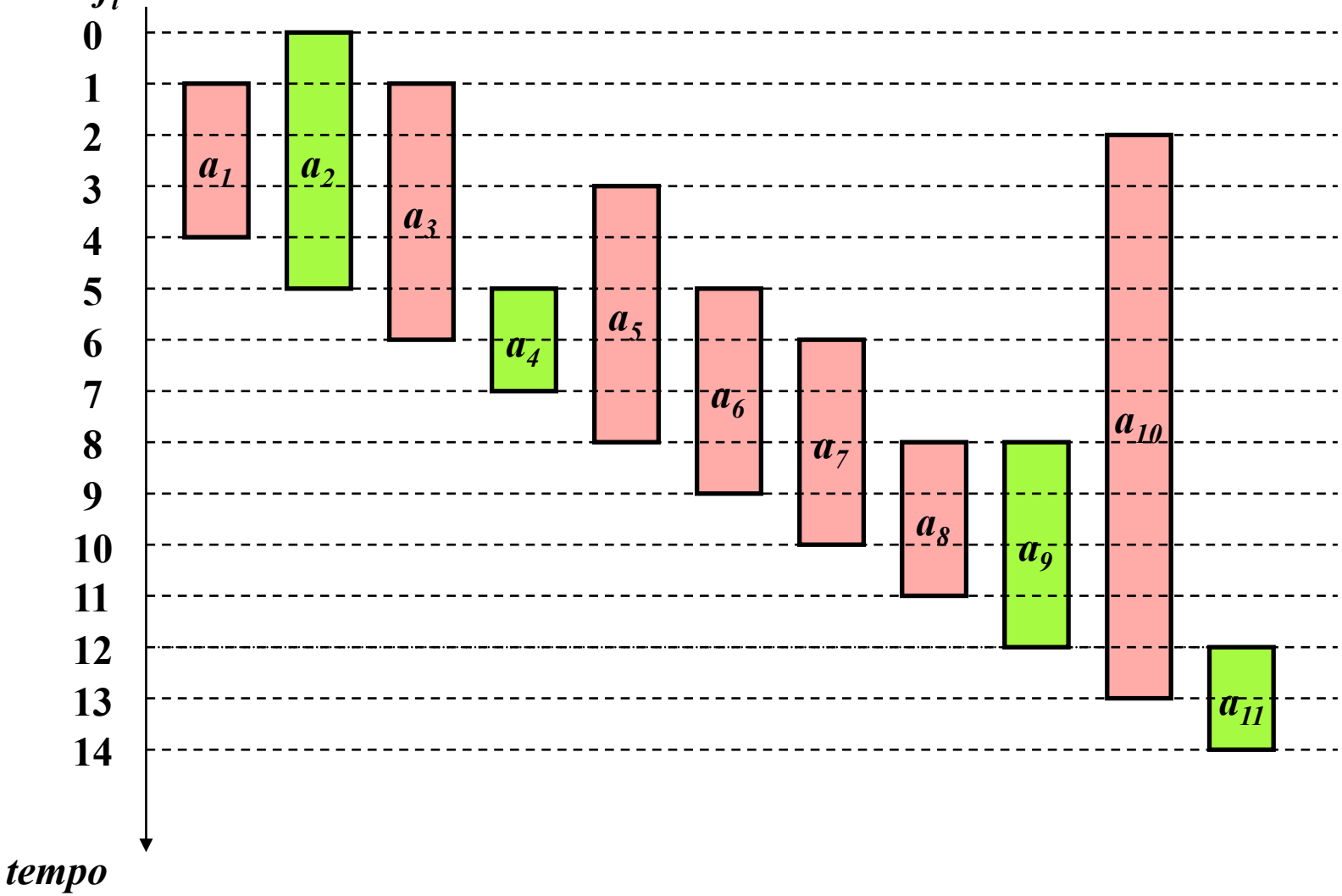

```
ActivitySelector(a, s, f, n)
  A = {a1},  k = 1
  for m = 2 to n
    if s[m] ≥ f[k]
      A = A ∪ {am}, k = m
  return A
```

La soluzione trovata contiene quattro attività
Due domande:

1) La soluzione trovata con l'algoritmo goloso è l'unica possibile che contiene quattro attività?

2) La soluzione trovata con l'algoritmo goloso è ottima o esistono anche soluzioni con più di quattro attività?

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 0 | 1 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



Cerchiamo di rispondere alla seconda domanda

2) La soluzione trovata con l'algoritmo goloso è ottima o esistono anche soluzioni con più di quattro attività?

```
ActivitySelector(a, s, f, n)   // f_1 ≤ ... ≤ f_n
  A = {a_1},  k = 1
  for m = 2 to n
    if s[m] ≥ f[k]
      A = A ∪ {a_m}, k = m
  return A
```

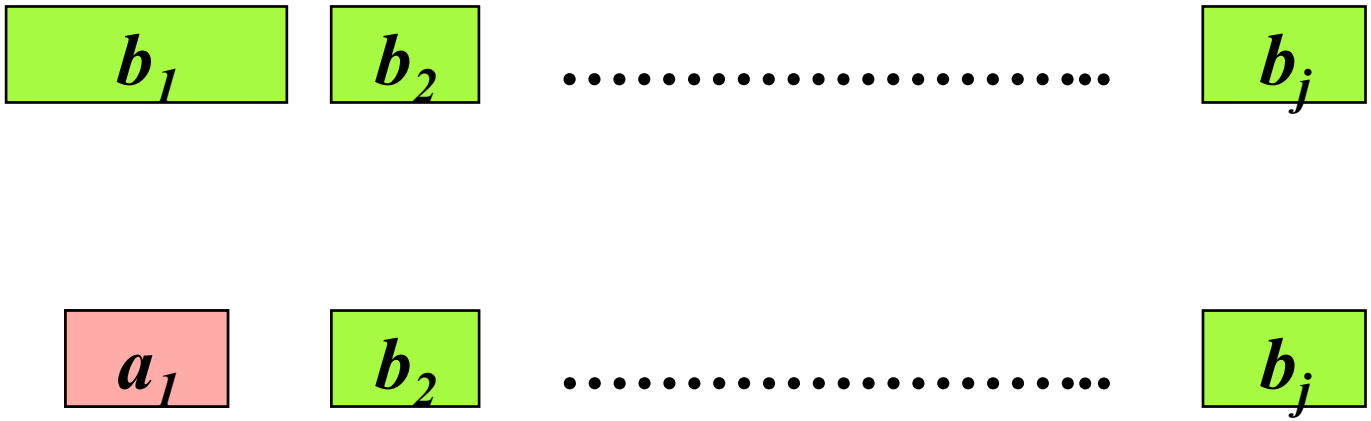
L'algoritmo comincia con scegliere la prima attività $\boldsymbol{a_1}$ (quella con tempo di fine minimo)

Siamo sicuri che questa scelta non possa compromettere il risultato?

In altre parole: esiste sempre una soluzione ottima che contiene $\boldsymbol{a_1}$?

<u>La risposta è affermativa.</u>

Sia $b_1,\ldots,b_j$ una qualsiasi soluzione ottima (ne esiste certamente almeno una) che supponiamo ordinata per tempo di fine



$k$ viene posto ad **1** ed aggiornato ad $m$ ogni volta che si sceglie una nuova attività $a_m$

```
ActivitySelector(a, s, f, n)   // f_1 ≤ ... ≤ f_n
  A = {a_1},  k = 1
  for m = 2 to n
    if s[m] ≥ f[k]
      A = A ∪ {a_m}, k = m
  return A
```

Siccome le attività sono ordinate per tempo di fine non decrescente, $f[k]$ è il massimo tempo finale delle attività selezionate precedentemente.

Con il test:

**if** $s[m] \geq f[k]$
$A = A \cup \{a_m\}, k = m$

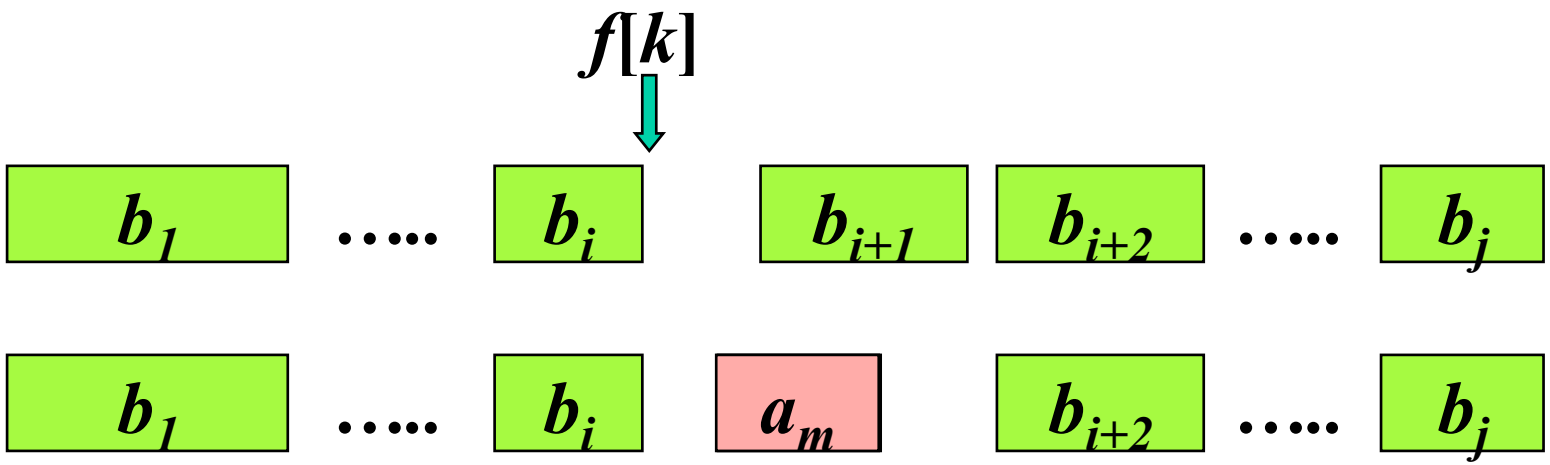
l'algoritmo seleziona la prima attività $a_m$ il cui tempo di inizio $s[m]$ è maggiore o uguale di $f[k]$

Siamo sicuri che questa scelta non comprometta il risultato?

In altre parole: esiste sempre una soluzione ottima che contiene $a_m$ e le attività finora scelte?

La risposta è ancora affermativa.

Assumiamo che esista una soluzione ottima $b_1,\ldots,b_i,b_{i+1},\ldots,b_j$ che estende le attività $b_1,\ldots,b_i$ finora scelte e supponiamo $b_1,\ldots,b_i$ e $b_{i+1},\ldots,b_j$ ordinate per tempo di fine



Sappiamo quindi che durante tutta l'esecuzione dell'algoritmo esiste sempre una soluzione ottima contenente le attività $\boldsymbol{b_1,\ldots,b_i}$ scelte fino a quel momento

Quando l'algoritmo termina non ci sono altre attività compatibili con $\boldsymbol{b_1,\ldots,b_i}$ e quindi le attività $\boldsymbol{b_1,\ldots,b_i}$ sono una soluzione ottima

L'algoritmo è ***goloso*** perchè ad ogni passo, tra tutte le attività compatibili con quelle già scelte, sceglie quella che termina prima.

Questa scelta è localmente ottima (golosa) perché è quella che lascia più tempo a disposizione per le successive attività.

***Esercizio* 1.** Problema dello "zaino" frazionario: Dati $n$ tipi di merce $M_1,\ldots,M_n$ in quantità rispettive $q_1,\ldots,q_n$ e con costi unitari $c_1,\ldots,c_n$ si vuole riempire uno zaino di capacità $Q$ in modo che il contenuto abbia costo massimo. Mostrare che il seguente algoritmo risolve il problema:

```
RiempiZaino(q, c, n, Q)    // c1 ≥ c2 ≥ ... ≥ cn
  Spazio = Q
  for i = 1 to n
    if Spazio ≥ q[i]
          z[i] = q[i], Spazio = Spazio – z[i]
    else z[i] = Spazio, Spazio = 0
  return z
```

***Esercizio 2.*** Problema dello "zaino" **0-1**:

Sono dati $n$ tipi di oggetti $O_1,\ldots,O_n$ in numero illimitato. Un oggetto di tipo $O_i$ occupa un volume $v_i$ e costa $c_i$.

Si vuole riempire uno zaino di capacità $Q$ in modo che il contenuto abbia costo massimo. Mostrare che il seguente algoritmo ***non*** risolve il problema.

```
RiempiZaino(v, c, n, Q)  // c1/v1 ≥ c2/v2 ≥ ... ≥ cn/vn
  Spazio = Q
  for i = 1 to n
      z[i] = ⌊Spazio/v[i] ⌋
      Spazio = Spazio – z[i]v[i]
  return z
```

## ***Esercizio 3***

Siano $\boldsymbol{a_1,\ldots,a_n}$ attività didattiche aventi tempi di inizio $\boldsymbol{s_1,\ldots,s_n}$ e tempi di fine $\boldsymbol{f_1,\ldots,f_n}$ e supponiamo di avere un insieme sufficiente di aule in cui svolgerle.

Trovare un algoritmo per programmare tutte le attività usando il minimo numero possibile di aule.

## ***Esercizio 4***

Siano $\boldsymbol{a_1,\ldots,a_n}$ attività didattiche aventi tempi di inizio $\boldsymbol{s_1,\ldots,s_n}$ e tempi di fine $\boldsymbol{f_1,\ldots,f_n}$ e abbiamo a disposizione $\boldsymbol{m}$ aule $\boldsymbol{A_1,\ldots, A_m}$

Trovare un algoritmo goloso per programmare il massimo numero di attività nelle $\boldsymbol{m}$ aule.