Algoritmi e Strutture Dati (Mod. B)

Algoritmi Greedy
(parte I)

Algoritmi greedy

- **Gli algoritmi per *problemi di ottimizzazione* devono in genere operare una sequenza di scelte per arrivare alla soluzione**
- **A volte usare la *programmazione dinamica* è costoso in termini di sviluppo e di efficienza…**
- **… mentre, in alcuni casi, esistono tecniche più immediate ed efficienti**
- **Gli *algoritmi greedy* sono algoritmi basati sull'idea di *fare sempre scelte che sembrano ottime al momento della scelta*.**
- **IMPORTANTE: *Non sempre è garantita la correttezza*, ma, se lo è, sono in genere molto semplici ed efficienti.**

Problema della selezione di attività

Problema: **Siano date N attività in competizione tra loro per l'utilizzo di una certa risorsa. Trovare l'*allocazione ottimale della risorsa*, cioè il massimo sottoinsieme di attività che la possano condividere senza creare conflitti per l'utilizzo della risorsa.**

- **$S=\{1,2,\ldots,N\}$ un insieme di attività**
- **Ad ogni attività $i \in S$ sono associati**
  - s_i = tempo di inizio (attivazione)
  - f_i = tempo di fine (conclusione)
- **tali che $s_i \leq f_i$**

Problema della selezione di attività

Definizione: **Due attività *i* e *j* si dicono *compatibili* se gli intervalli $[s_i, f_i]$ e $[s_j, f_j]$ sono *disgiunti*, cioè non si sovrappongono. In altre parole se vale**

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

Problema (riformulato): **Dato l'insieme di attività $S=\{1,2,\ldots,N\}$, trovare il *massimo sottoinsieme di attività tra loro compatibili*. Assumiamo che le attività siano *ordinate per tempo di terminazione*, cioè:**

$$f_j \leq f_i \qquad \text{se } j < i$$

Problema della selezione di attività

Problema (riformulato): **Dato l'insieme di attività *S*={1,2,…,*N*}, (*ordinate per tempo di terminazione*) trovare il *massimo sottoinsieme di attività tra loro compatibili*.**

1. **Partiamo con un insieme *A* di attività inizialmente vuoto e *j=1***
2. **Inseriamo in *A* l'attività *j* col minimo tempo di terminazione**
3. **Fra le attività rimanenti *compatibili* con *j***
   - **selezionare l'attività *i* col minore tempo di terminazione**
   - **aggiungere l'attività selezionata *i* ad *A***
4. **Se esistono altre attività compatibili con *i*, torniamo al passo 3 altrimenti terminamo**

Problema della selezione di attività

1. **Assumiamo che le attività siano ordinate in modo crescente rispetto al tempo di fine.**

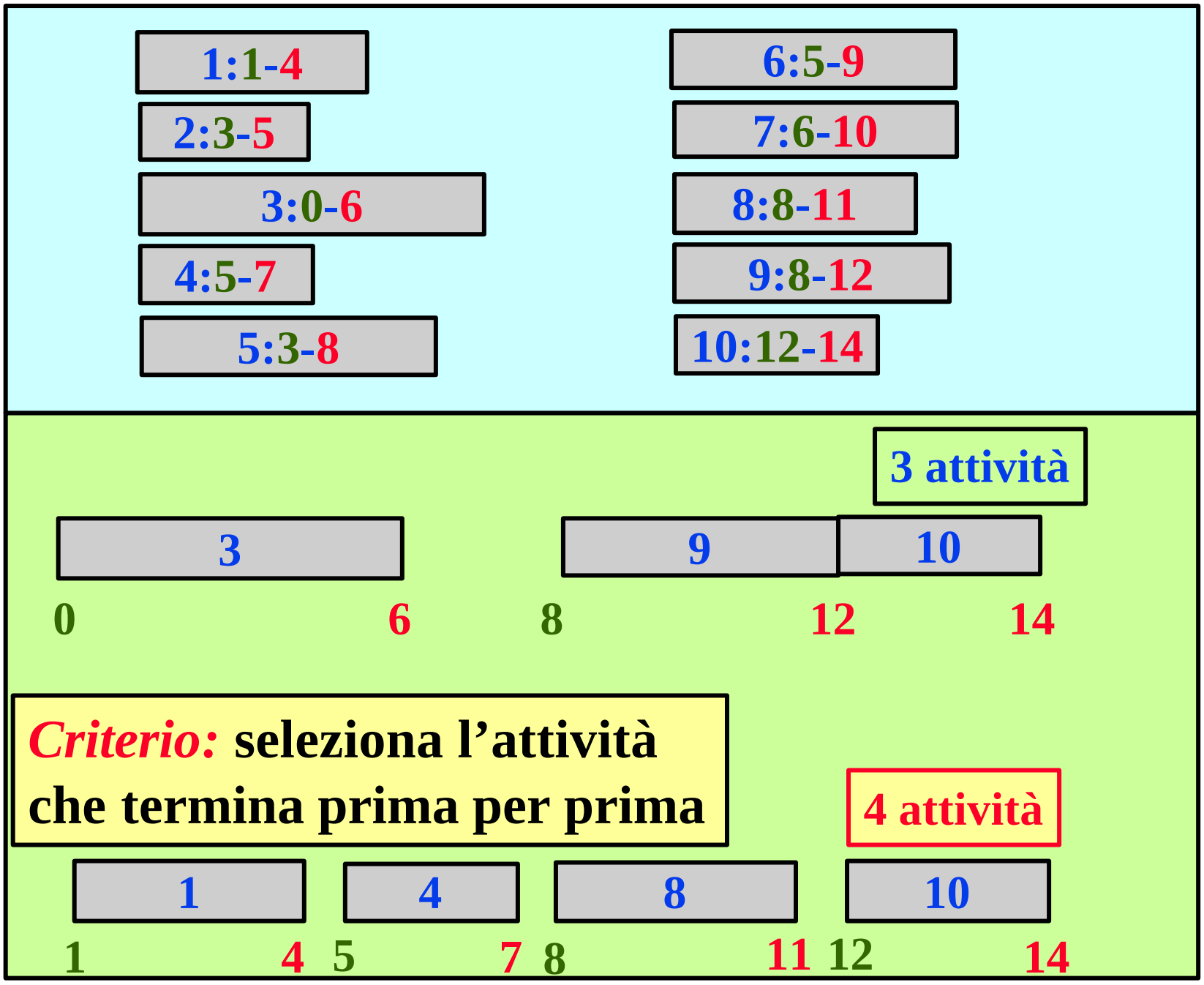
   $f_1 \leq f_2 \leq \ldots \leq f_N$ (***possiamo applicare Algoritmo di Ordinamento***!)

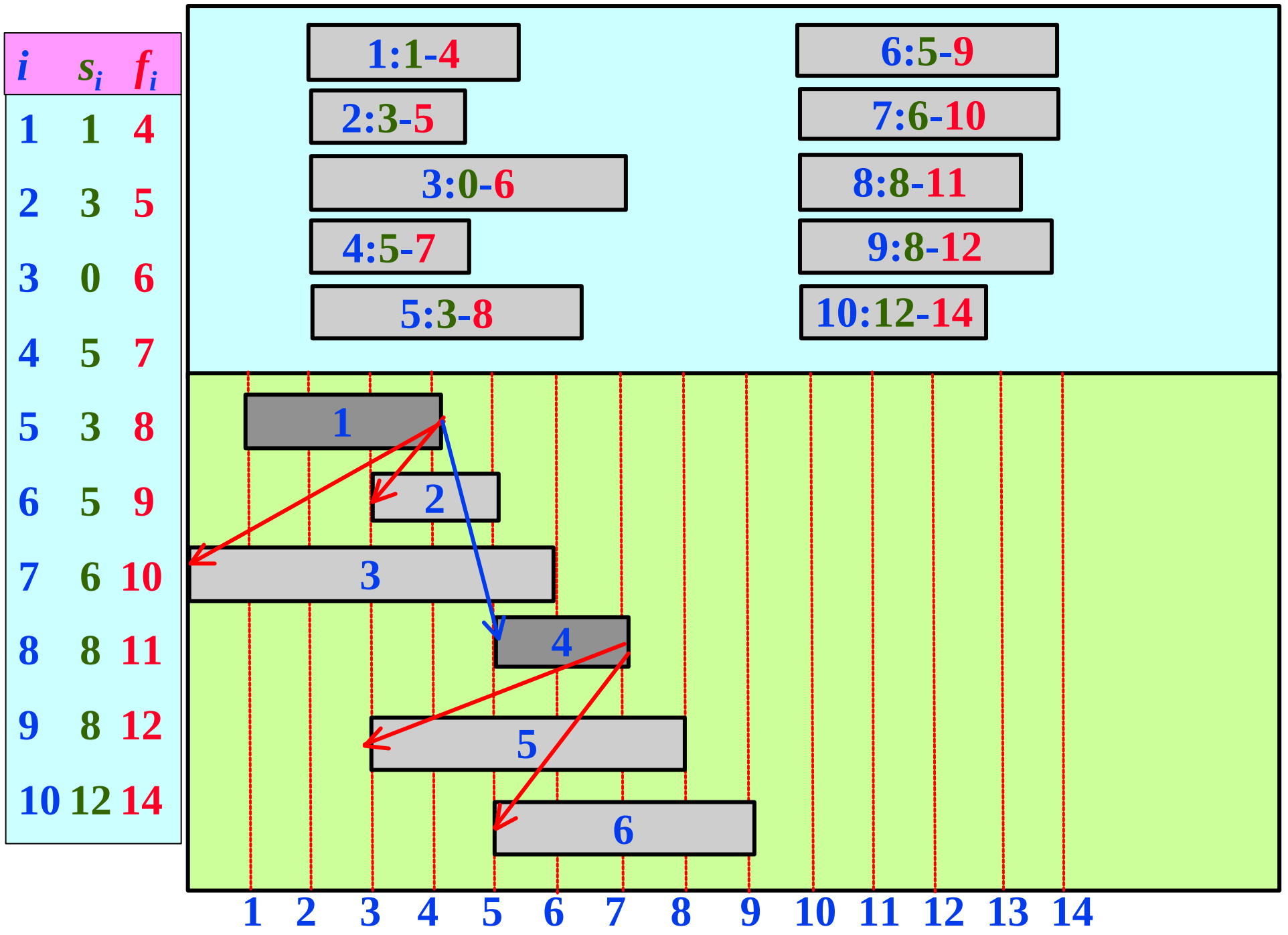
2. **Siano $S=[s_1,\ldots,s_N]$ e $F=[f_1,\ldots,f_N]$ i vettori contenenti i tempi di inizio (non ordinati) e di fine (ordinati)**

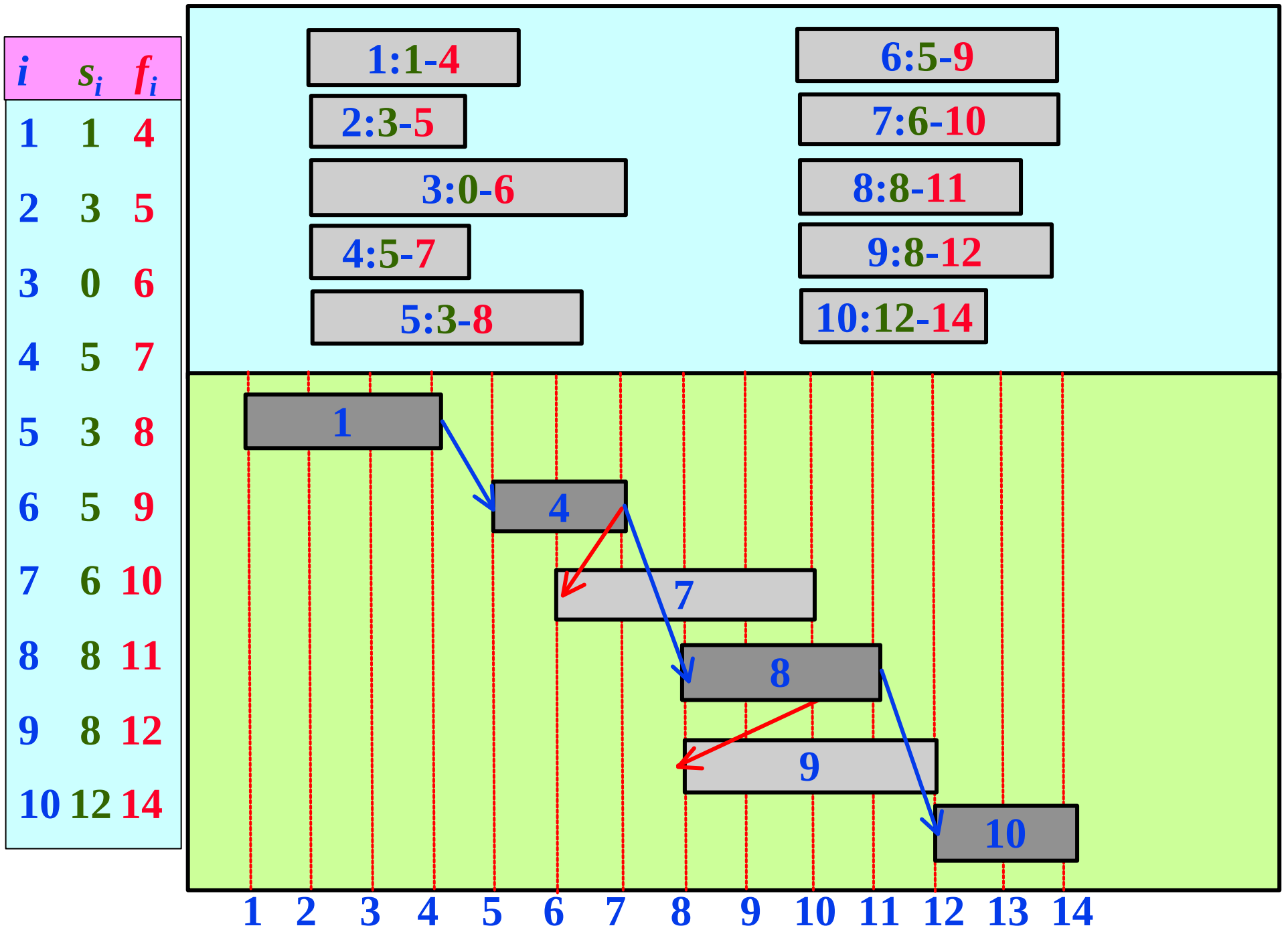
3.
```
Greedy-Activity-Selection(S,F: array)
 A = {1}
 j = 1
 for i = 2 to length[S]
   do if S[i] ³ F[j]
         then A = A ∪ {i}
              j = i
```

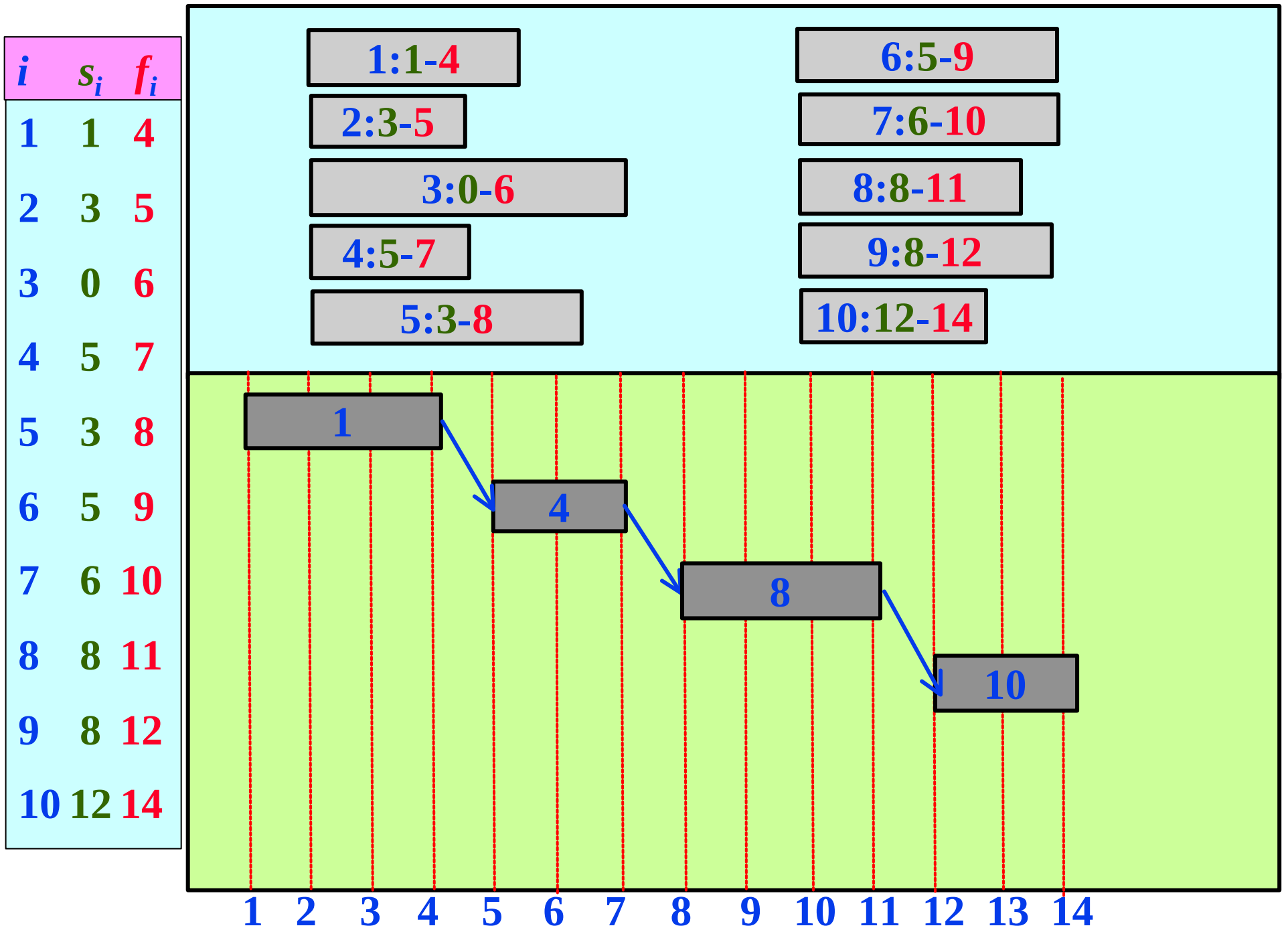
4. **Al termine l'inseme A contiene la *soluzione greedy* al problema costruito in tempo lineare $\Theta(n)$ [$\Theta(n \log n)$]**

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




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


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


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

Selezione di attività: correttezza

Teorema: **L'algoritmo `Greedy-Activity-Selection` produce soluzioni di dimensione massima per il problema della selezione delle attività.**

Dimostrazione: **Per assunzione dell'algoritmo, le N attività sono *ordinate* per tempo di fine, quindi la prima attività è quella che termina per prima.**

Dimostriamo innanzi tutto che *esiste una soluzione ottima* che inizia con la *prima attività*.

Sia A Í S una qualsiasi soluzione ottima.

Ordiniamola per tempi di fine crescenti.

Supponiamo che $A[1] = S[k]$ e k ¹ 1 (cioè la prima attività di A non è quella con tempo di terminazione minore in S)

Quindi vale $f_1 < f_k$.

Selezione di attività: correttezza

Teorema: **L'algoritmo `Greedy-Activity-Selection` produce soluzioni di dimensione massima per il problema della selezione delle attività.**

Dimostrazione: **Dimostriamo innanzi tutto che *esiste una soluzione ottima* che inizia con la *prima attività*.**

$A \subseteq S$ sia una qualsiasi soluzione ottima (ordinata per tempi di fine) con $A[1] = S[k]$ e $k \neq 1$. Quindi vale $f_1 < f_k$.

Possiamo quindi costruire un'altra soluzione $B = A - \{k\} + \{1\}$

***B* è una soluzione perchè vale $f_1 < f_k$ e quindi nessun vincolo di compatibilità è violato.**

***B* è anche ottima perchè ha lo stesso numero di attività della soluzione ottima *A*.**

***B è quindi una sol. ottima che inizia con la scelta greedy*.**

Selezione di attività: correttezza

Teorema: **L'algoritmo `Greedy-Activity-Selection` produce soluzioni di dimensione massima per il problema della selezione delle attività.**

Dimostrazione: **Quindi *esiste una soluzione ottima che inizia con la prima attività (passo base proprietà della scelta greedy)*.**

Dopo la scelta greedy, il problema si riduce al sottoproblema di trovare la massima sequenza di *attività compatibili* con la prima.

Il sottoproblema è quindi di cercare di una sequenza massima di attività tra $S' = \{i \in S : s_i \geq f_1\}$ (vedi `if` in codice)

Se A è una soluzione ottima, la soluzione ottima del sottoproblema S' è allora $A' = A - \{1\}$. *Dimostriamolo!*

Se A' non fosse ottima, esisterebbe un B' maggiore di A' ...

Selezione di attività: correttezza

Teorema: **L'algoritmo `Greedy-Activity-Selection` produce soluzioni di dimensione massima per il problema della selezione delle attività.**

Dimostrazione: **Quindi *esiste una soluzione ottima che inizia con la prima attività (passo base proprietà della scelta greedy)*.**

Il sottoproblema è quindi di cercare di una sequenza massima di attività tra $S' = \{i \in S : s_i \geq f_1\}$ (vedi `if` in codice)

Se A è una soluzione ottima, la soluzione ottima del sottoproblema S' è $A' = A - \{1\}$.

Se A' non fosse ottima, esisterebbe un B' maggiore di A' per lo stesso sottoproblema $S' = \{i \in S : s_i \geq f_1\}$.

Ma aggiungendo l'attività 1 a B', otterremo una soluzione B migliore di A (*contraddizione*).

Selezione di attività: correttezza

Teorema: **L'algoritmo `Greedy-Activity-Selection` produce soluzioni di dimensione massima per il problema della selezione delle attività.**

Dimostrazione:

Quindi *esiste una soluzione ottima che inizia con la prima attività (passo base proprietà della scelta greedy)*...

... e dopo la prima scelta greedy, rimaniamo col sotto-problema (analogo) di cercare di una *sequenza massima* di attività in un insieme $S' = \{i \in S : s_i \geq f_1\}$.

Per induzione si può allora dimostrare facilmente che data una qualsiasi soluzione ottima ad un (sotto-)problema, essa contiene al suo interno le soluzioni ottime dei suoi sottoproblemi (*proprietà della sottostruttura ottima*).

L'induzione completa anche la *proprietà della scelta greedy*.

Selezione di attività: esercizio

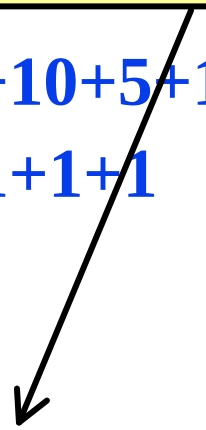
Esercizio: **mostrare che le seguenti due strategie greedy non garantiscono la soluzione ottima:**

1. **selezionare sempre l'attività con durata minore compatibile con le precedenti**
2. **selezionare sempre l'attività che, tra quelle rimanenti, si accavalla col minor numero di attività**

Problema del cambio di denaro

- **Input**
  - **Un numero intero positivo *n***
- **Output**
  - **Il più piccolo numero intero di banconote per cambiare *n* mila lire usando pezzi da 20 mila, 10 mila, 5 mila, e mille lire.**
- **Esempi**
  - ***n* = 58 (mila), 7 banconote: 20+20+10+5+1+1+1**
  - ***n* = 18 (mila), 5 banconote: 10+5+1+1+1**
- **Algoritmo**
  - **Dispensa una banconota alla volta**
  - **Ad ogni passo, utilizza la banconota *più grande* che *non superi la cifra rimanente*.**

Criterio di scelta greedy



Un altro problema del cambio di denaro

- **Input**
  - **Un intero positivo *n***
- **Output**
  - **Il più piccolo numero di banconote per cambiare *n* dollari usando banconote da 12, 8, e 1 dollari.**
- **Esempio**
  - ***n* = 31**
  - **9 banconote: 12 + 12 + 1 + 1 + 1 + 1 + 1 + 1 + 1**
  - **6 banconote : 12 + 8 + 8 + 1 + 1 + 1**

Il criterio greedy non garantisce ottimalità

Un altro problema del cambio di denaro

- **Input**
  - **Un intero positivo *n***
- **Output**
  - **Il più piccolo numero di banconote per cambiare *n* dollari usando banconote da 50, 25, 10 e 1 dollari.**
- **Esempio**
  - ***n* = 55**
  - **6 banconote: 50 + 1 + 1 + 1 + 1 + 1**
  - **4 banconote : 25 + 10 + 10 + 10**

Il criterio greedy non garantisce ottimalità

Algoritmo Greedy

- **Costituito da una *sequenza* di *scelte*.**
- **Ad ogni *punto di decisione*, sceglie quella che "*sembra*" essere la *scelta migliore* in quel *momento*.**
- **Procede in maniera "*top-down*"**
  - **Prende una decisione greedy dopo l'altra**
  - ***Iterativamente* riduce il problema ad uno di entità (complessità) minore**

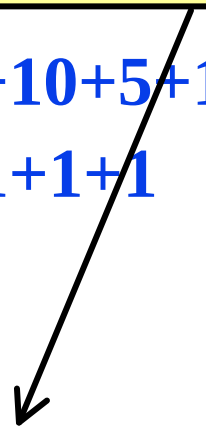
Ingredienti per garantire l'ottimalità

- **Ottimo per alcuni problemi ma non per tutti**
- **Propertà della *scelta greedy***
  - **Una *soluzione globalmente ottima* può esser ottenuta effettuando, in sequenza, delle *scelte localmente ottime* (*greedy*).**
- **Propertà della *sottostruttura ottima***
  - **Una *soluzione ottima* al problema *contiene* le *soluzioni ottime* dei *sottoproblemi***
  - **Simile al caso della *programmazione dinamica***

Problema del cambio di denaro

- **Input**
  - **Un numero intero positivo *n***
- **Output**
  - **Il più piccolo numero intero di banconote per cambiare *n* mila lire usando pezzi da 20 mila, 10 mila, 5 mila, e mille lire.**
- **Esempi**
  - ***n* = 58 (mila), 7 banconote: 20+20+10+5+1+1+1**
  - ***n* = 18 (mila), 5 banconote: 10+5+1+1+1**
- **Algoritmo**
  - **Dispensa una banconota alla volta**
  - **Ad ogni passo, utilizza la banconota *più grande* che *non superi la cifra rimanente*.**

Criterio di scelta greedy



Problema del cambio di denaro

Teorema: **Il problema del cambio di denaro precedente soddisfa sia la *proprietà della sottostruttura ottima* che la *proprietà della scelta greedy*.**

Dimostrazione (cenni): **Se $b_1,\ldots,b_k$ è una soluzione ottima al problema di cambiare n mila lire, $b_2,\ldots,b_k$ deve essere una soluzione ottima al problema di cambiare $n - b_1 v_1$ mila lire (la banconota 1 vale v_1 mila lire).**

La seconda parte (*scelta greedy*) si basa sul fatto che non è possibile che in una soluzione ottima non compaia la *scelta greedy*.

Problema del cambio di denaro

Teorema: **Il problema del cambio di denaro precedente soddisfa sia la *proprietà della sottostruttura ottima* che la *proprietà della scelta greedy*.**

Dimostrazione (cenni): **Assumiamo $b_1,\ldots,b_k$ sia una soluzione ottima, che la banconota h sia la più grande non superiore all'importo n e che h non compaia nella soluzione.**

Analizzando i casi possibili, si nota che se h non supera n, esisterà sempre nella soluzione un insieme di almeno due biglietti di tagia inferiore ad h la cui somma sia proprio h (*tutti i tagli sono infatti divisibili per qualsiasi dei tagli minori*).

Problema del cambio di denaro

Teorema: **Il problema del cambio di denaro precedente soddisfa sia la *proprietà della sottostruttura ottima* che la *proprietà della scelta greedy*.**

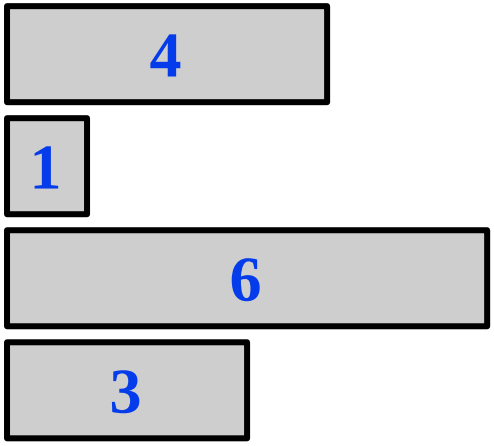
Dimostrazione (cenni): **Assumiamo $b_1,\ldots,b_k$ sia una soluzione ottima, che la banconota h sia la più grande non superiore all'importo n e che h non compaia nella soluzione.**

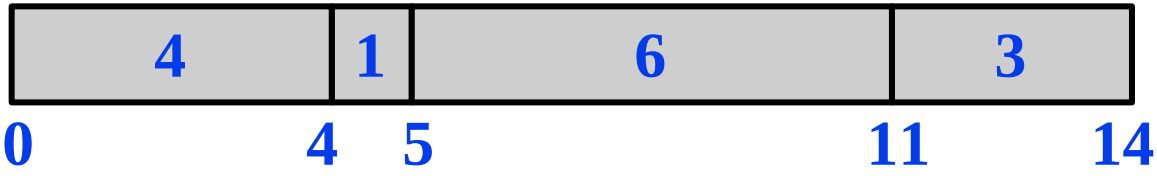
Analizzando i casi possibili, si nota che se h non supera n, esisterà sempre nella soluzione un insieme di almeno due biglietti di tagia inferiore ad h la cui somma sia proprio h (*tutti i tagli sono infatti divisibili per qualsiasi dei tagli minori*).

Quindi *sostituendo* nella soluzione il *biglietto di taglia maggiore* si otterrebbe una *soluzione migliore* di $b_1,\ldots,b_k$, il che è una *contraddizione*.

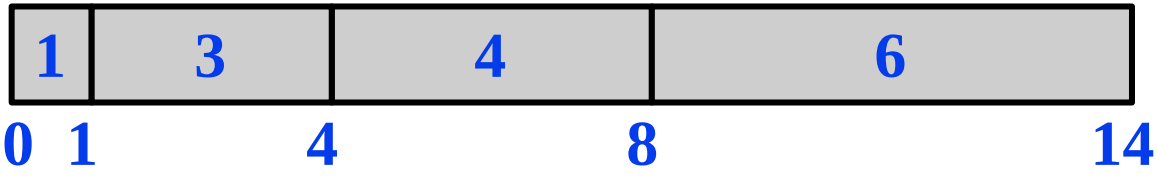
Un semplice problema di Scheduling

- ***n*** **job e 1 macchina**
- **Ogni job ha un tempo di esecuzione**
- **Eseguire i job sulla macchina**
- **Minimizzare tempo di completamento totale**



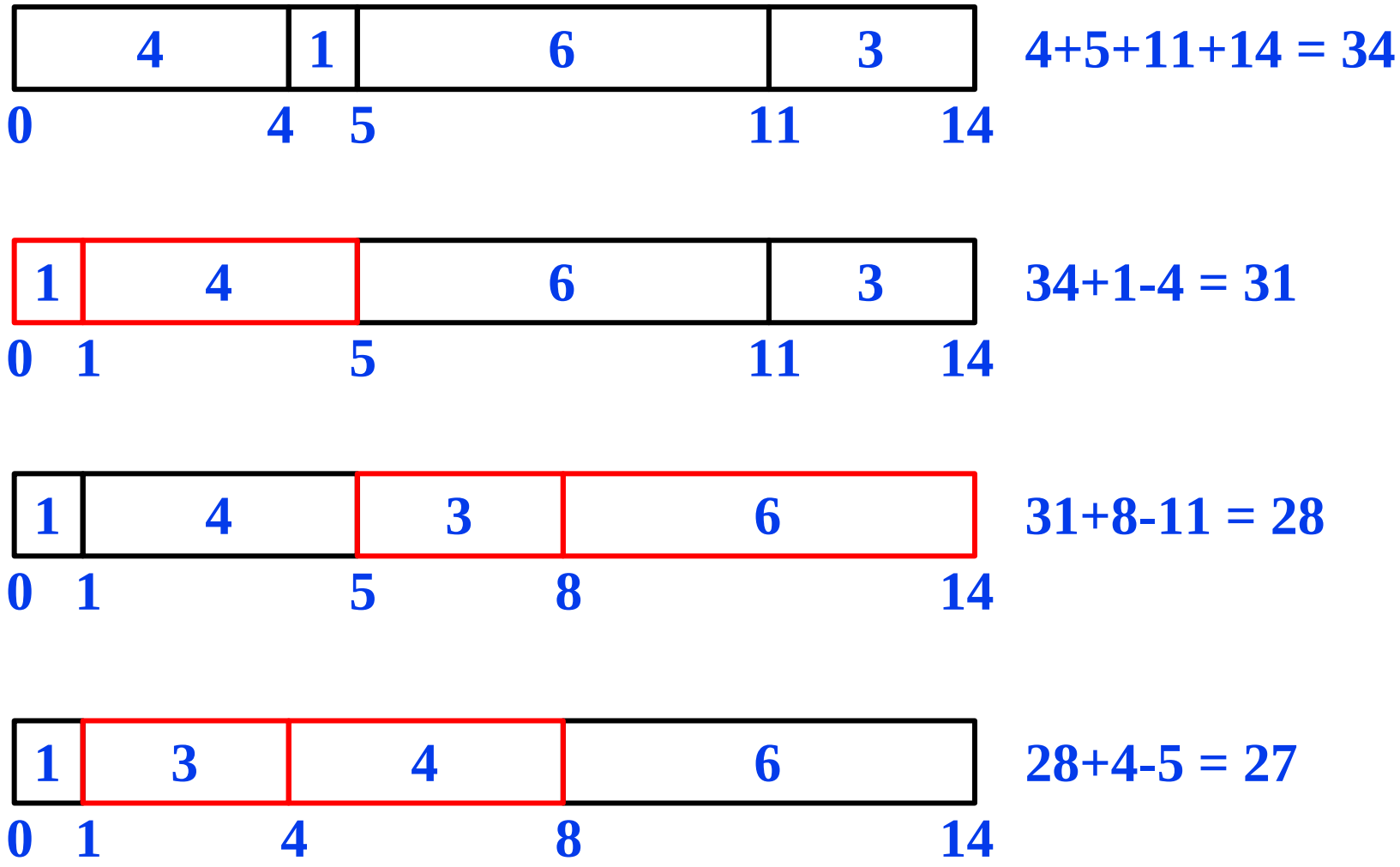


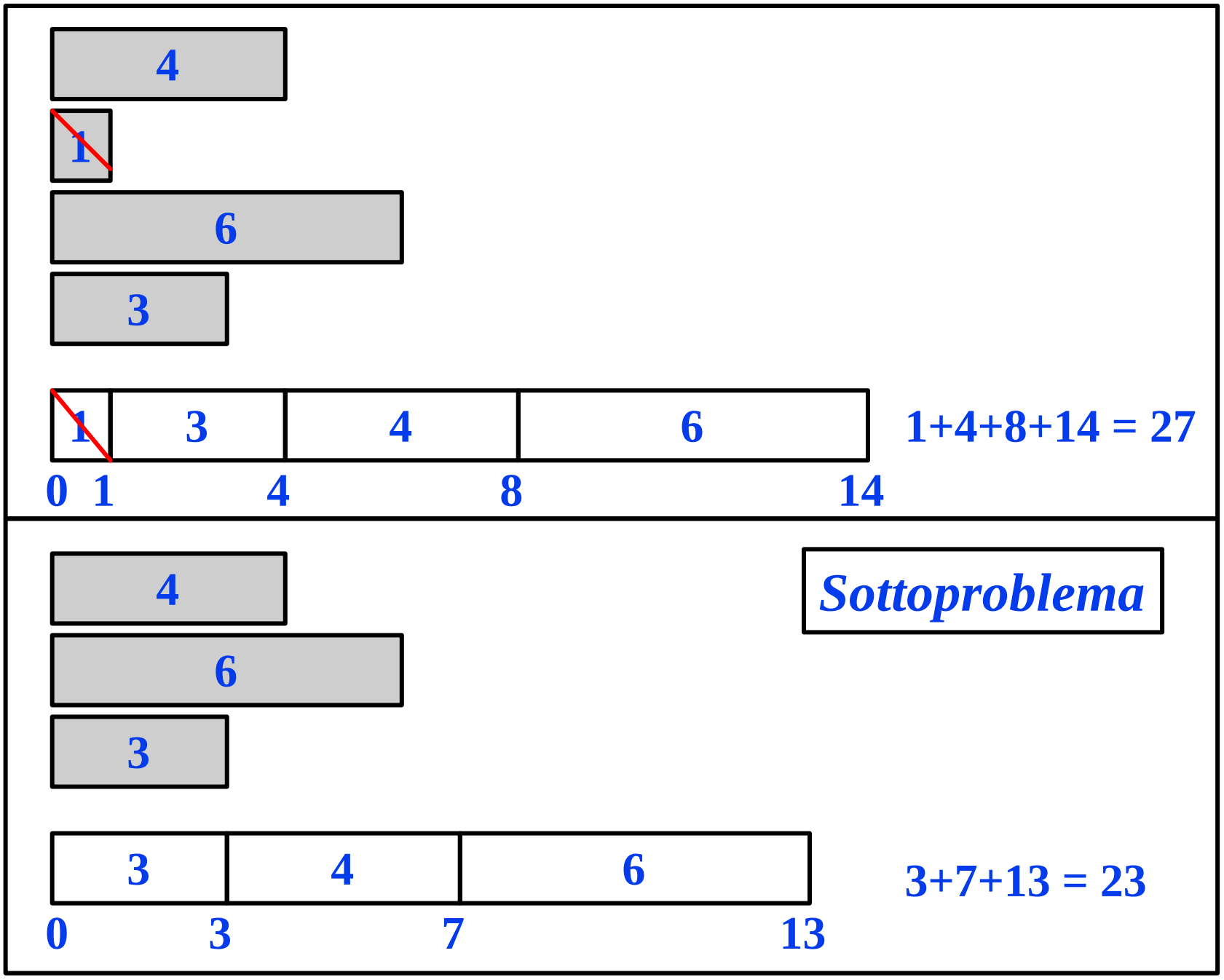
4+5+11+14 = 34



1+4+8+14 = 27

Criterio: **esegui il job più piccolo per primo**


4
1
6
3
0
4
5
11
14
4+5+11+14 = 34
1
4
6
3
0
1
5
11
14
34+1-4 = 31
1
4
3
6
0
1
5
8
14
31+8-11 = 28
1
3
4
6
0
1
4
8
14
28+4-5 = 27


4
1
6
3
1
3
4
6
1+4+8+14 = 27
0
1
4
8
14
4
6
3
Sottoproblema
3
4
6
3+7+13 = 23
0
3
7
13

Un semplice problema di Scheduling

Teorema: **Il problema dello scheduling soddisfa la *sottostruttura ottima*.**

Teorema: **Il problema dello scheduling soddisfa la *scelta greedy*.**

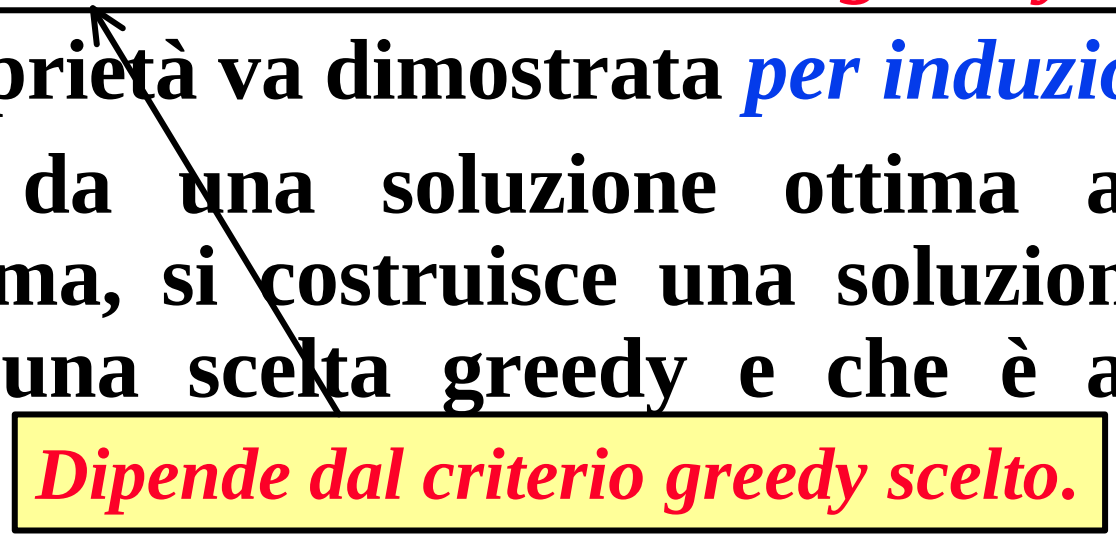
Dimostrare i due teoremi per esercizio!

Vantaggi e Svantaggi

- **Svantaggi**
  - **Fornisce una soluzione subottima: *Cambio di denaro***
  - **Non riesca a fornire una soluzione: *Instradamento con collegamenti bloccati***
- **Vantaggi**
  - **Facile da sviluppare e realizzare**
  - **Basso tempo di esecuzione**
  - **Spesso i problemi che si vogliono risolvere con questo metodo sono "*difficili*". È improbabile riuscire a trovare una soluzione ottima in ogni caso.**

Proprietà della scelta greedy

- **Perché sia applicabile la tecnica greedy, il problema deve esibire la *proprietà della scelta greedy*: *per ogni sottoproblema, esiste una soluzione ottima che inizia con la scelta greedy*.**
- **Questa proprietà va dimostrata *per induzione*.**
- **A partire da una soluzione ottima ad un sottoproblema, si costruisce una soluzione che inizia con una scelta greedy e che è ancora ottima.**
- **La scelta riduce il problema ad un sottoproblema analogo più piccolo**
- **L'induzione ci permette di completare la prova.**



Proprietà della sottostruttura ottima

- **Perché sia applicabile la tecnica greedy, il problema deve esibire la *proprietà della sottostruttura ottima***
- **Ovviamente anche questa proprietà va dimostrata come per la *programmazione dinamica*.**
- **La tecnica per dimostrarla è essenzialmente la stessa.**

Problema dello Zaino frazionario

- **Ci sono *n* oggetti a disposizione.**
- **L'oggetto *i*-esimo pesa p_i chili e vale c_i mila lire.**
- **Vogliamo caricare uno zaino con un carico di oggetti in modo da massimizzare il valore complessivo del carico ma di non superare un peso fissato di *W* chili.**
- **Si possono anche inserire *porzioni di oggetti* di meno di p_i chili.**

£60.000 | 10 | £6.000 per chilo

£125.000 | 25 | £5 .000 per chilo

£120.000 | 30 | £4.000 per chilo

55

Scegli per primo l’oggetto che ha il più alto valore

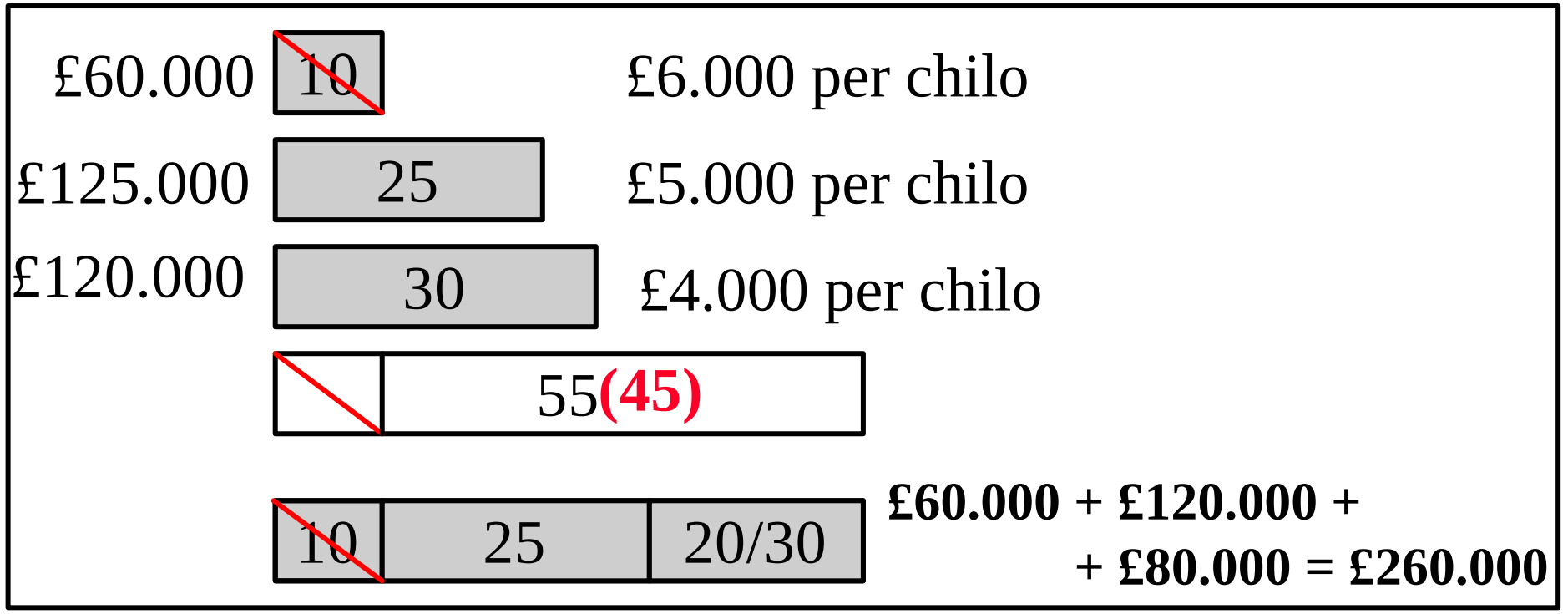
25 | 30 | £125.000 + £120.000 = £245.000

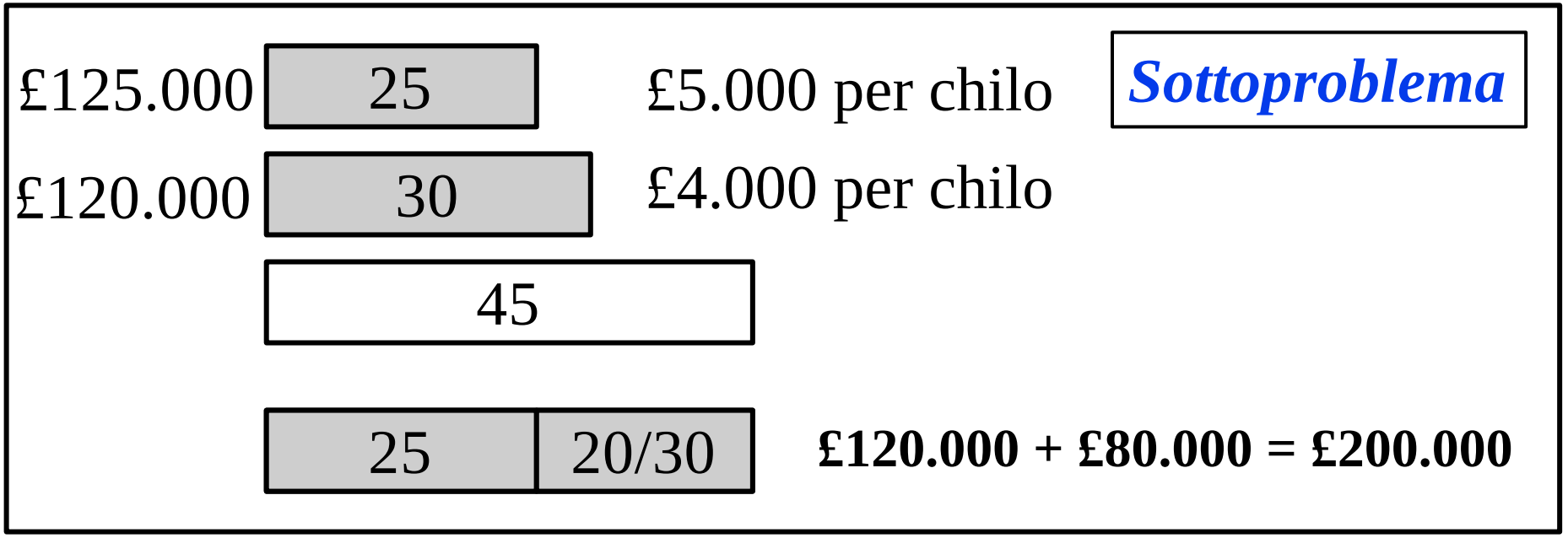
Scegli per primo l’oggetto che ha il magior peso

30 | 25 | £120.000 + £125.000 = £245.000

Scegli per primo l’oggetto che ha il più alto valore per chilo

10 | 25 | 20/30 | £60.000 + £120.000 + £80.000 = £260.000


£60.000
10
£6.000 per chilo
£125.000
25
£5.000 per chilo
£120.000
30
£4.000 per chilo
55(45)
10
25
20/30
£60.000 + £120.000 +
+ £80.000 = £260.000


£125.000
25
£5.000 per chilo
Sottoproblema
£120.000
30
£4.000 per chilo
45
25
20/30
£120.000 + £80.000 = £200.000

Sottostruttura ottima

Teorema: **Il problema dello zaino frazionario soddisfa la *sottostruttura ottima*.**

***Dimostrazione*:**

- **Sia $S = \cup w_i$ la soluzione ottima al problema W con n oggetti di peso massimo p_i**
- **S associa ad ogni oggetto i un peso $0 \leq w_i \leq p_i$**
- **Il costo di W sarà quindi $\sum w_i \; c_i/p_i$ e $W = \sum w_i$**
- **Se rimuoviamo la quantità w dell'oggetto h da S otteniamo una sol. S' per il sottoproblema $W-w$ con n-1 oggetti più $p_h - w$**
- ***S' deve essere ottima*! Perché?**

Scelta greedy

Teorema: **Il problema dello zaino frazionario soddisfa la *scelta greedy*.**

***Dimostrazione (cenni)*:**

1. **Possiamo dimostrare che: *nella soluzione ottima deve comparire la quantità massima dell'oggetto con il maggior rapporto costo/peso* (c_i / p_i) (*scelta greedy*).**
2. **Successamente si può dimostrare che: *la scelta greedy può sempre essere fatta per prima*.**

Scelta greedy

Teorema: **Il problema dello zaino frazionario soddisfa la *scelta greedy*.**

***Dimostrazione (cenni)*:**

1. ***Nella soluzione ottima deve comparire la quantità massima dell'oggetto con il maggior rapporto costo/peso*** **(c_i /p_i) (*scelta greedy*).**

Siano c_h e p_h valore e peso disponibile dell'oggetto col massimo rapporto c_i/p_i e w_i il peso dell'i-esimo oggetto nella soluzione.

$$C = \Sigma_{i=1..n} w_i \bullet c_i/p_i$$

Se w_j ^{1} 0 e avanza p'_h ^{3} w_j dell'oggetto h allora sostituendolo nella soluzione per j ci da una soluzione migliore, infatti

$$w_j \bullet c_j/p_j \pounds w_j \bullet c_h/p_h$$

Scelta greedy

Teorema: **Il problema dello zaino frazionario soddisfa la *scelta greedy*.**

Dimostrazione (cenni):

2. la scelta greedy può sempre essere fatta per prima

***Caso 1*: $W \leq p_h$ allora $w_h = W$ e $w_i = 0$ per gli altri.**

***Caso 2*: Se $W > p_h$ e w_i è la prima scelta e per la dimostrazione precedente tutto *h* deve comparire.**

$$C = w_i \bullet c_i/p_i + p_h \bullet c_h/p_h + \sum_{i^1 1,h} w_i \bullet c_i/p_i$$

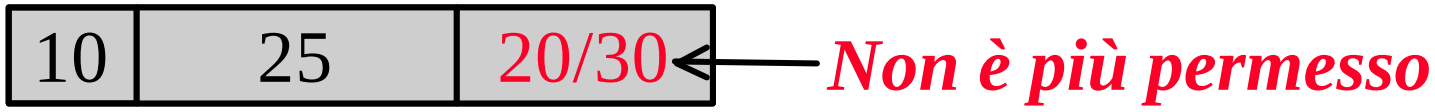
Se scambiamo *i* con *h*, otteniamo una nuova soluzione con valore

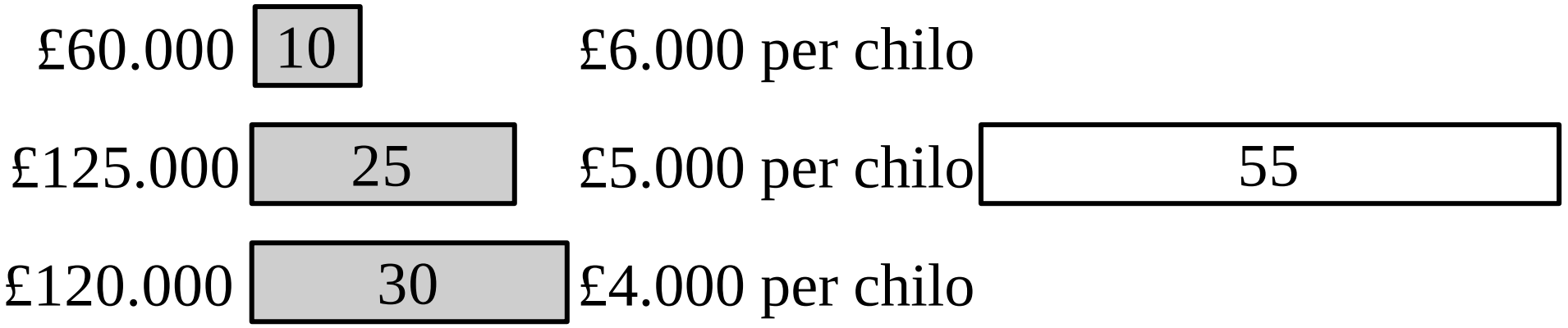
$$C' = p_h \bullet c_h/p_h + w_i \bullet c_i/p_i + \sum_{i^1 1,h} w_i \bullet c_i/p_i$$

Ma se *C* è ottima, lo è anche *C'*, poiché *C* = *C'*

Problema dello Zaino 0-1

- **Identica formulazione del problema dello zaino eccettuato che**
  - **un oggetto deve essere abbandonato tutto**
  - **oppure preso tutto**





Scegli per primo l'oggetto che ha il più alto valore per chilo

10	25	20/30

£60.000 + £120.000 + £80.000 = £260.000

10	25	

£60.000 + £125.000 = £185.000

10	30	

£60.000 + £120.000 = £180.000

30	25

£120.000 + £125.000 = £245.000

Sottostruttura ottima

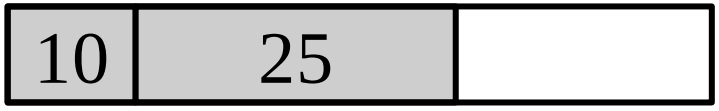
Teorema: **Il problema dello zaino 0-1 soddisfa la *sottostruttura ottima*.**

***Dimostrazione*:**

- **Sia $S \acute{I} I$ la soluzione ottima al problema W, I è l'insieme complessivo degli n oggetti, $i \hat{I} I$ ha peso p_i**
- **S è cioè un sottoinsieme degli oggetti a disposizione.**
- **Il *valore* di W sarà quindi $S_{i \hat{I} S} c_i$ e $W = S_{i \hat{I} S} p_i$**
- **Se rimuoviamo l'oggetto h da S otteniamo una sol. S' per il sottoproblema $W - p_h$ con n-1 oggetti (h escluso).**
- ***S' deve essere ottima*! Perché?**

Scelta greedy

Il problema dello zaino 0-1 *non* soddisfa la proprietà di *scelta greedy*.



£60.000 + £125.000 = £185.000

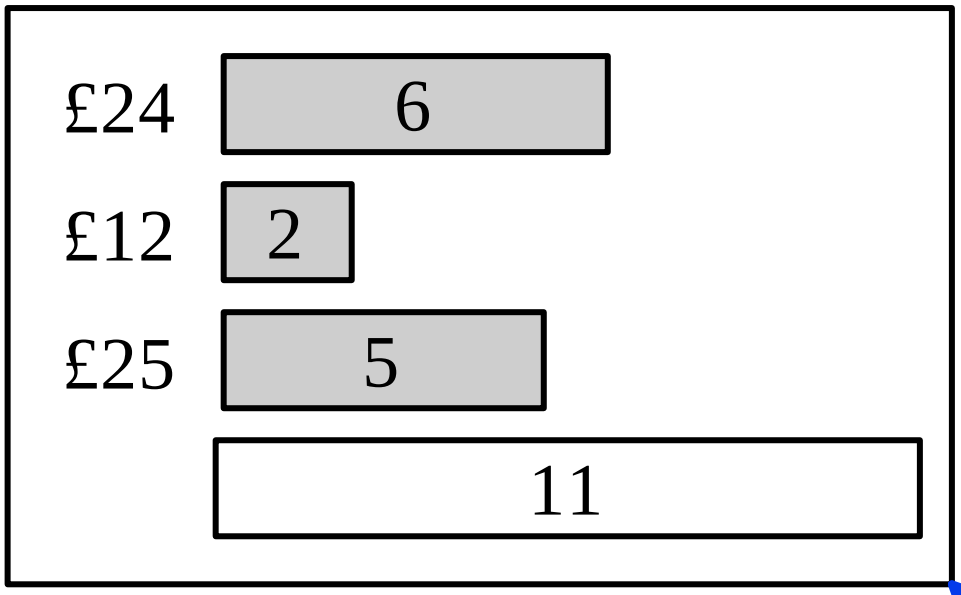
£120.000 + £125.000 = £245.000

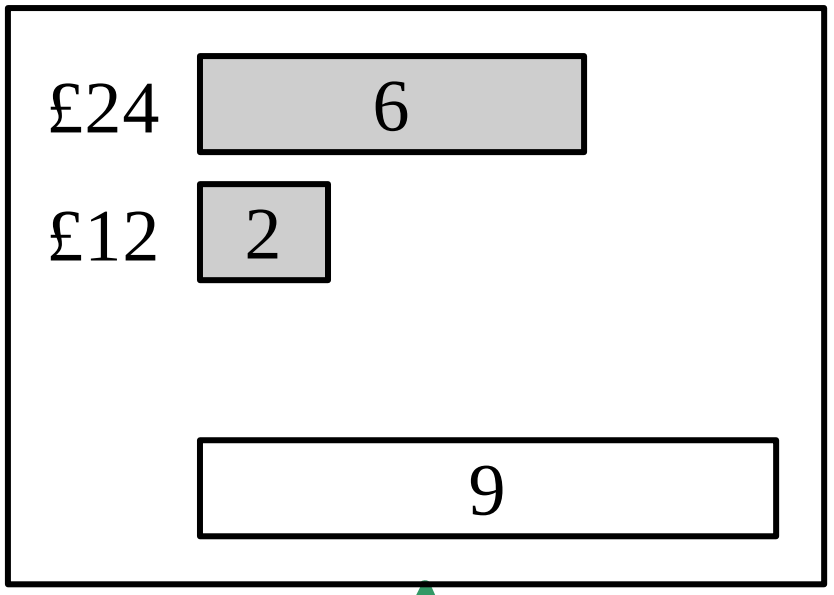
Algoritmo per lo Zaino 0-1

- **Dobbiamo usare la *programmazione dinamica***
- ***v*(*i*,*w*) = massimo *valore* dello zaino con *peso al massimo w* e con *solo* gli *oggetti* 1,…,*i disponibili*.**

Vogliamo calcolare *v*(*n*,*W*)

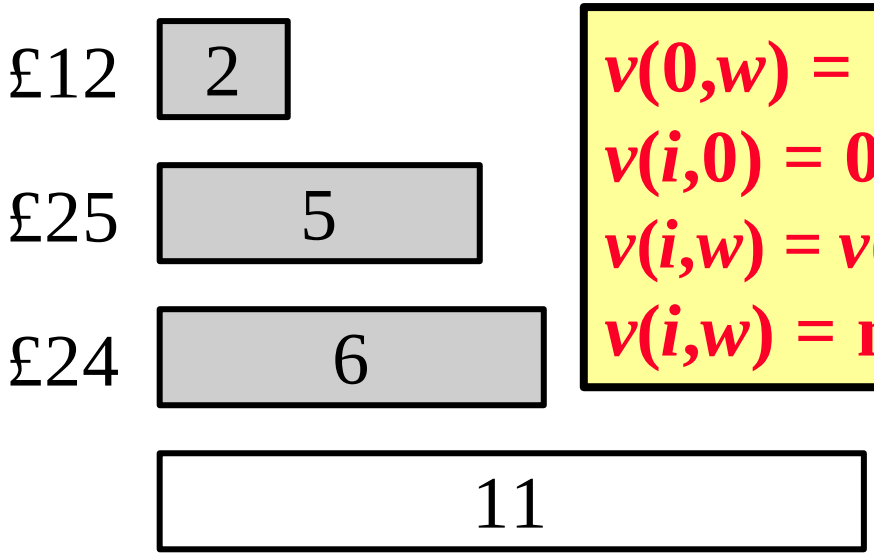
- ***v*(0,*w*) = 0 per ogni *w***
- ***v*(*i*,0) = 0 per ogni *i***
- ***v*(*i*,*w*) = *v*(*i*-1,*w*) per ogni *i* e per *w* < p_i**
- ***v*(*i*,*w*) = max{ *v*(*i*-1,*w*), *v*(*i*-1,*w*-p_i)+c_i } altrimenti**





$$v(i,w) = \max\{\, v(i-1,w),\ v(i-1,w-p_i)+c_i \,\}$$

i \ *w*	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	24	24	24	24	24	24
2	0	0	12	12	12	12	24	24	36	36	36	36
3	0	0	12	12	12	25	25	37	37	37	37	49



$v(0,w) = 0$ per ogni w

$v(i,0) = 0$ per ogni i

$v(i,w) = v(i\text{-}1,w)$ per ogni i e per $w < p_i$

$v(i,w) = \max\{ v(i\text{-}1,w), v(i\text{-}1,w\text{-}p_i)+c_i \}$

i \ *w*	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	12	12	12	12	12	12	12	12	12	12
2	0	0	12	12	12	25	25	37	37	37	37	37
3	0	0	12	12	12	25	25	37	37	37	37	49

Greedy e Prog. Dinamica

- **Il problema mostra *sottostruttura ottima***
- **Problema dello Zaino fazionario**
  - **La tecnica Greedy funziona**
  - **Usare programmazione dinamica è inutilmente complicato (il metodo è sovradimensionato)**
- **Problema dello Zaino 0-1**
  - **La tecnica Greedy non funziona**
  - **È necessaria la programmazione dinamica**

Esercizi

- **Esercizio 17.2-3**
- **Esercizio 17.2-4**