# 4. ALGORITMI GREEDY

- cambia-monete
- scheduling a minimo il ritardo

## Il problema del cambia-monete

Scopo. Dati i tagli disponibili: 1c, 2c, 5c, 10c, 20c, 50c, 1€, 2€ progettare un algoritmo che data una certa somma la cambi usando il minimo numero di monete.

Ex. 34cent.

Algoritmo del cassiere. Ad ogni iterazione, aggiungi una moneta del valore più alto che non supera la somma rimanente da pagare.

Ex. €2.89.



## Algoritmo del cassiere

Ad ogni iterazione, aggiungi una moneta del valore più alto possibile che non supera la somma rimanente da pagare.

```
ALGORITMO-DEL-CASSIERE (x, c1, c2, ..., cn)

ORDINA gli n tagli di moneta c1 < c2 < ... < cn
S ← φ        ⟵ inseieme di monete selezionate
WHILE x > 0
   k ← n
   WHILE k > 0
      IF ck ≤ x
         x ← x − ck ;     S ← S ∪ { k }
         BREAK WHILE
   IF k = 0 RETURN "nessuna soluzione"
END WHILE
RETURN S
```

Domanda. L'algoritmo del cassiere fornisce sempre la soluzione ottima?



## Proprietà di una soluzione ottima

**Tagli disponibili**: 1c, 2c, 5c, 10c, 20c, 50c, 1€, 2€

Prop. 1 Il numero di monete da 1c è ≤ 1.
Dim. Se ci fossero 2 monete da 1c potremmo sostituirle con una da 2c.

Prop. 2 Numero di monete da 2c ≤ 2.
Prop. 3 Number di monete da 5c ≤ 1.
Prop. 4 Number di monete da 10c ≤ 1.
Prop. 5 Number di monete da 20c ≤ 2.
Prop. 6 Number di monete da 50c ≤ 1.
Prop. 7 Number di monete da 1€ ≤ 1.

Prop. 9 Numero di 1c + numero di 2c ≤ 2.
Dim.
- due 2c e un 1c possono essere sostituiti da un 5c;

Prop. 10 Numero di 10c + number di 20c ≤ 2.

## Analisi dell'algoritmo del cassiere (greedy)

Teorema. L'algoritmo è ottimo per le monete: 1, 2, 5, 10, 20, 50, 1€, 2€.

Dim. [per induzione su x (la somma da cambiare)]

- Consideriamo la soluzione ottima Opt per cambiare una somma $x$ tale che $c_k \le x < c_{k+1}$ : Greedy sceglie prima la moneta di valore $c_k$.
- Se nella soluzione ottima Opt non ci fosse la moneta di taglio $c_k$
  - Opt dovrebbe contenere monete $c_1, \ldots, c_{k-1}$ che sommano ad $x \ge c_k$
  - dalla tabella si vede che tale collezione di monete non può esistere
- Quindi Opt=$\{c_k\} \cup$ soluzione per $x - c_k$ cents, che, per induzione, è quella ottenuta dall'algoritmo del cassiere. ▪

| k | $c_k$ | ogni soluzione ottima soddisfa | max valore con monete $c_1, \ldots, c_{k-1}$ in Opt |
|---|---|---|---|
| 1 | 1 | $\#1c \le 1$ | - |
| 2 | 2 | $\#2c \le 2$ | 1 |
| 3 | 5 | $\#1c + \#2c \le 2$ | 2 + 2 = 4 |
| 4 | 10 | $\#5c \le 1$ | 4 + 5 = 9 |
| 5 | 20 | $\#10c \le 1$ | 9 + 10 = 19 |
| 6 | 50 | $\#10c + \#20c \le 2$ | 9+20+20 = 49 |

## L'algoritmo del cassiere per altri tagli di moneta

Domanda. L'algoritmo del cassiere è ottimo per ogni insieme di monete?

Risposta. No. Si consideri il caso dei francobolli:
10c, 70c, 85c, 90c, 1€, 1.4€, 1.5€, 2€, 2.20€, 2.80€, 3€.

- Algoritmo del cassiere: 1.80€ = 1.5€ + 10c + 10c + 10c.
- Ottimo: 1.80€ = 90c + 90c.


Importante. Infatti potrebbe addirittura portare a non dare una soluzione, dove ne esiste una. Esempio, con i tagli dei francobolli senza quello da 10c: Per la somma 2.70€

- Cashier's algorithm: 2.70€ = 2.20€ + ???. Ottimo: 2.70€ = 2€ + 70c.



## Cambia-monete in Bioinformatica

Mass Decomposition Problem. Abbiamo un frammento di DNA di massa pari a 2650±1Da e nessun'altra informazione. Come possiamo provare ad identficarlo?

Passo 1. Quali combinazioni di basi esistono con tale massa?

Risposta. Esistono solo due possibili combinazioni:

- 8 guanine: 2632.42Da + 18.01Da (una molecola d'acqua)
- 7 citosine e 2 timine: 2631.42 Da + 18.01 Da (acqua)

DNA Mass Decomposizione Problem = Problema del Cambiamonete

- Pesi molecolari dei nucleotidi = tagli di monete disponibili
- molecola da identificare = somma da cambiare



## 4. Algoritmi Greedy

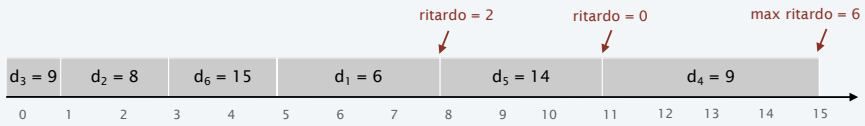
- cambia-monete
- scheduling a minimo ritardo

## Scheduling che minimizza il ritardo

Il problema del minimo ritardo.

- Un computer esegue un programma per volta.
- Job $j$ richiede $t(j)$ unità di tempo e deve essere completato entro $d(j)$.
- Se $j$ comincia l'esecuzione al tempo $s(j)$, terminerà al tempo $f(j) = s(j) + t(j)$.
- Ritardo: $\ell_j = \max \{ 0, f(j) - d(j) \}$.
- Scopo: ordine di esecuzione dei jobs che minimizza il massimo ritardo $L = \max_j \ell_j$.

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $t_j$ | 3 | 2 | 1 | 4 | 3 | 2 |
| $d_j$ | 6 | 8 | 9 | 9 | 14 | 15 |






## Minimizzare il ritardo: possibili scelte greedy

Lo schema Greedy. Ordina i job seguendo un criterio apparentemente ottimale.

- [In ordine di lunghezza] Ordina i jobs in ordine di tempo di esecuzione crescente $t_j$.
- [In ordine di scadenza] Ordina i job in ordine crescente di scadenza $d(j)$.
- [In ordine di tempo rimanente] Ordina i job in ordine crescente della differenza tra la deadline e la lunghezza del job $d_j - t_j$.



## Minimizzare il ritardo: scelte greedy

Schema Greedy. Ordina i job seguendo un criterio (apparentemente) ottimale.

- [In ordine di lunghezza] Ordina i jobs in ordine di tempo di esecuzione crescente $t_j$.

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 100 | 10 |

controesempio



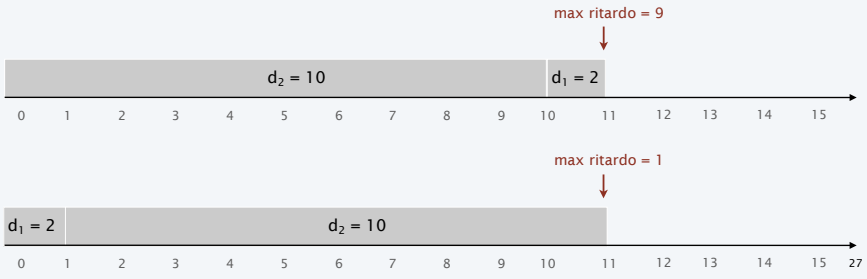

## Minimizzare il ritardo: scelte greedy

Schema Greedy. Ordina i job seguendo un criterio (apparentemente) ottimale.

- [In ordine di tempo rimanente] Ordina i job in ordine crescente della differenza tra la deadline e la lunghezza del job $d_j - t_j$.

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 2 | 10 |

controesempio



## Minimizzare il ritardo: in ordine di deadline

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

ORDINA $n$ job in modo che $d_1 \le d_2 \le \ldots \le d_n$.

$t \leftarrow 0$

FOR $j = 1$ TO $n$

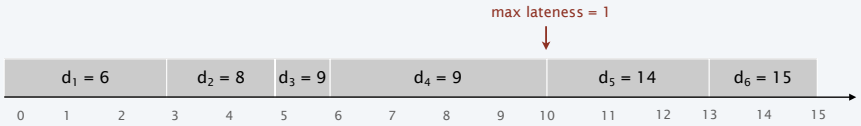
  Assegna job $j$ all'intervallo $[t, t + t_j]$.

  $s_j \leftarrow t$ ; $f_j \leftarrow t + t_j$

  $t \leftarrow t + t_j$

RETURN intervalli $[s_1, f_1]$, $[s_2, f_2]$, ..., $[s_n, f_n]$.

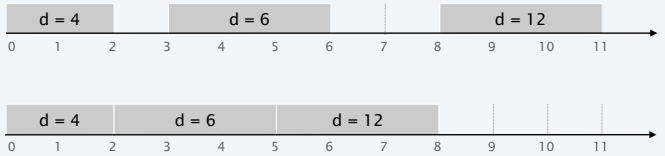
| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $t_j$ | 3 | 2 | 1 | 4 | 3 | 2 |
| $d_j$ | 6 | 8 | 9 | 9 | 14 | 15 |






## Minimizzare il ritardo: nessun tempo morto

Osservazione 1. Esiste un'ordine di esecuzione ottimo senza tempi morti.



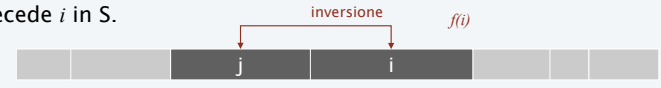

Osservazione 2. La soluzione greedy "in ordine di scadenza" non ha tempi morti.



## Minimizzare il ritardo: inversioni

Def. Dato un ordine di esecuzione $S$, un'inversione è una coppia di job $i$ e $j$ tali che: (i) greedy schedula prima $i$ e poi $j$ (quindi $d(i) \le d(j)$ )

(ii) ma $j$ precede $i$ in S.




[ assumiamo che i job siano numerati con $d_1 \le d_2 \le \ldots \le d_n$ ]

Osservazione 3. La soluzione greedy "in ordine di scadenza" non ha inversioni.

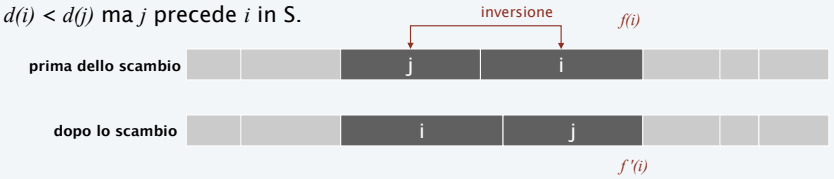
Osservazione 4. Se un ordine di esecuzione (senza tempi morti) presenta un'inversione, allora ha un'inversione costituita da una coppia di job consecutivi.



## Minimizzare il ritardo: inversioni

Def. Dato un ordine di esecuzione $S$, un'inversione è una coppia di job $i$ e $j$ tali che:

$d(i) < d(j)$ ma $j$ precede $i$ in S.




Lemma. Scambiando due job adiacenti che formavano un'inversione, riduce il numero di inversioni di uno e non aumenta il max ritardo.

Dim. Sia $\ell$ il ritardo prima dello scambio, e sia $\ell'$ il ritardo dopo lo scambio.

- $\ell'_k = \ell_k$ per ogni $k \ne i, j$.
- $\ell'_i \le \ell_i$.
- se $j$ è in ritardo, $\ell'_j = f'(j) - d(j)$ (per definizione)
  $= f(i) - d(j)$ ($j$ ora finisce al tempo $f(i)$)
  $\le f(i) - d(i)$ (perché $i$ e $j$ formano un'inversione)
  $\le \ell_i$. (per definizione)

## Minimizzare il ritardo: analisi di greedy

Teorema. L'ordine $S$ fornito dall'algoritmo greedy "per ordine di scadenza" è ottimo.

Dim. [per assurdo]

Prendiamo una soluzione ottima $S^*$ che ha il minimo numero di inversioni, e vediamo che succede.

- Possiamo assumere che $S^*$ non abbia tempi morti.
- Se $S^*$ non ha inversioni allora $S = S^*$
- Se $S^*$ ha un'inversione, sia $i$–$j$ una coppia adiacente di job in inversione.
- Scambiando $i$ e $j$
  - non incrementiamo il max ritardo
  - riduciamo di uno il numero delle inversioni
- Questo contraddice la definizione di $S^*$
  - il nuovo ordine ha lo stesso ritardo massimo: è una soluzione ottima
  - ha un'inversione in meno rispetto ad $S^*$, che doveva essere una soluzione ottima con il minimo numero di inversioni ▪



## Strategie per l'analisi di algoritmi Greedy

Greedy è più avanti. mostriamo che dopo ogni scelta dell'algoritmo greedy, la sua soluzione è almeno tanto buona quanto quella di un qualsiasi altro algoritmo .

Analisi strutturale. definiamo un bound "strutturale" garantendo che ogni possibile soluzione deve avere almeno un certo valore. Quindi mostriamo che greedy soddisfa sempre quel bound.

Argomento basato sullo scambio. Gradualmente transformiamo una qualsiasi soluzione in quella trovata da greedy, mostrando che ogni passo di tale trasformazione non peggiora la qualità della soluzione.

Altri algoritmi greedy. Gale-Shapley (visto), Dijkstra (cammini minimi in un grafo), Huffman (codici e compressione dati), …