Metodo greedy: Scheduling to Minimize Lateness

12 maggio 2023

Punto della situazione

Tecniche di progettazione di algoritmi:

- algoritmi esaustivi
- divide et impera;
- programmazione dinamica;
- tecnica greedy
- algoritmi esaustivi ''intelligenti''

Esempi:

1. Selezione di intervalli (versione non pesata) ([KT] par. 4.1)
2. Partizionamento di intervalli ([KT] par. 4.1)
3. Scheduling che minimizza il ritardo ([KT] par. 4.2)
4. Codici di Huffman ([KT] par. 4.8)

E altri sui grafi: MST, cammini minimi, …

Tecnica greedy

- Serve per risolvere problemi di ottimizzazione = ricerca di una soluzione ammissibile che ottimizzi (max/min) un valore associato.
- Non sempre è possibile utilizzarla per ottenere una soluzione ottimale (devo usare programmazione dinamica)
- Schema molto semplice
- Prova della correttezza… di meno

Schema algoritmo greedy

- Ordino secondo un criterio di convenienza
- Inizializzo SOL = insieme vuoto
- Considero ogni oggetto in ordine di convenienza
  - Se è compatibile con SOL lo aggiungo a SOL
- Restituisco SOL

Greedy

Hard, if not impossible, to precisely define "greedy algorithm."

Myopic greed: "at every iteration, the algorithm chooses the best morsel it can swallow, without worrying about the future“

Objective function. Does not explicitly appear in greedy algorithm!

Scheduling to Minimizing Lateness

Minimizing lateness problem.

Single resource processes one job at a time.

Job j requires t_j units of processing time and is due at time d_j (deadline).

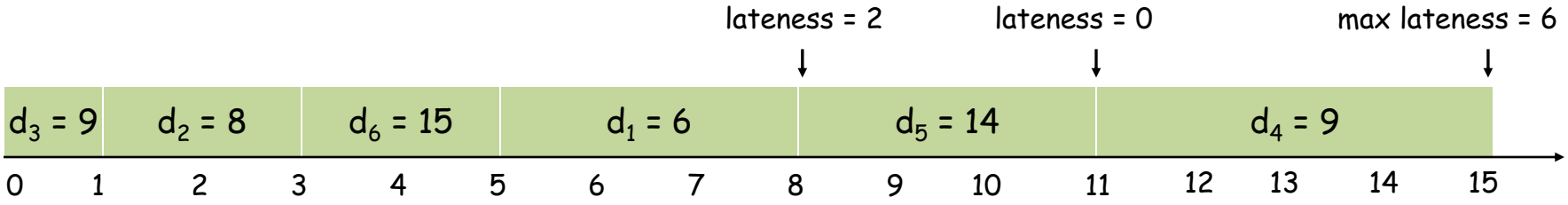
If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.

Lateness of j: $\lambda_j = \max \{ 0, f_j - d_j \}$. (Se $f_j \leq d_j$ allora il ritardo=0)

Goal: schedule all jobs to minimize maximum lateness $L = \max \lambda_j$.

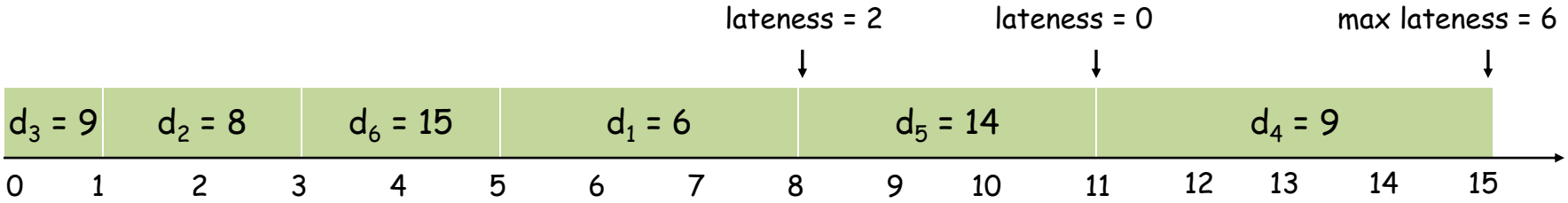
Ex:

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

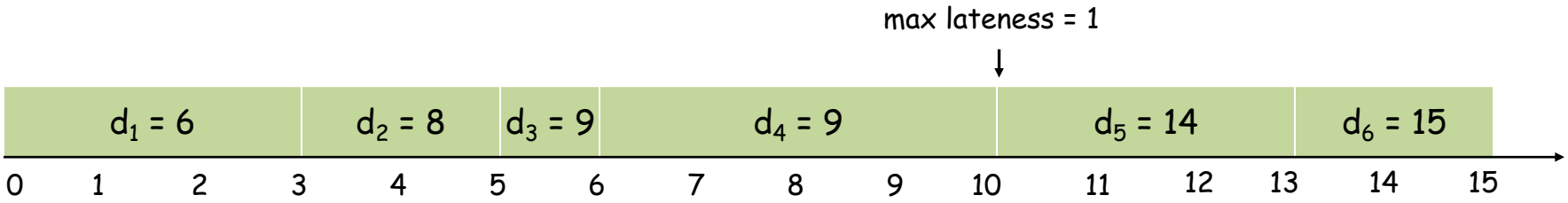


Esempio

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



- Il ritardo di questo primo scheduling è max{0, 0, 0, 2, 0, 6} = 6



- Il ritardo di questo secondo scheduling è max{0, 0, 0, 1, 0, 0} = 1

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

[Shortest processing time first] Consider jobs in ascending order of processing time t_j.

[Earliest deadline first] Consider jobs in ascending order of deadline d_j.

[Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

Quale porta ad una soluzione ottimale?

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

[Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

[Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

Minimizing Lateness: Greedy Algorithm

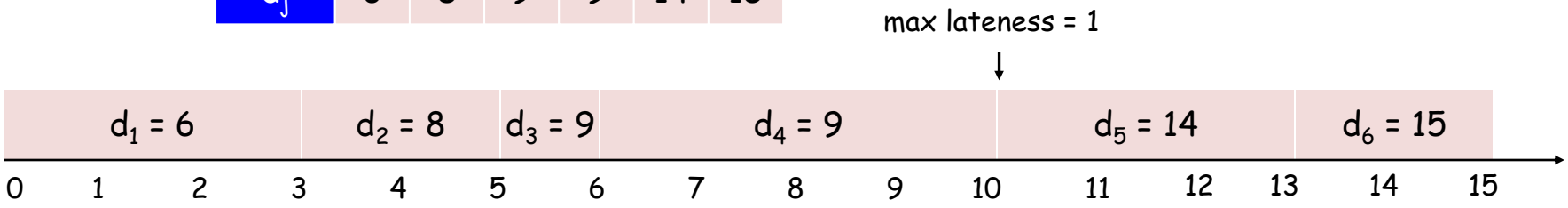
Greedy algorithm. Earliest deadline first.

```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
output intervals [s_j, f_j]
```

Esempio

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



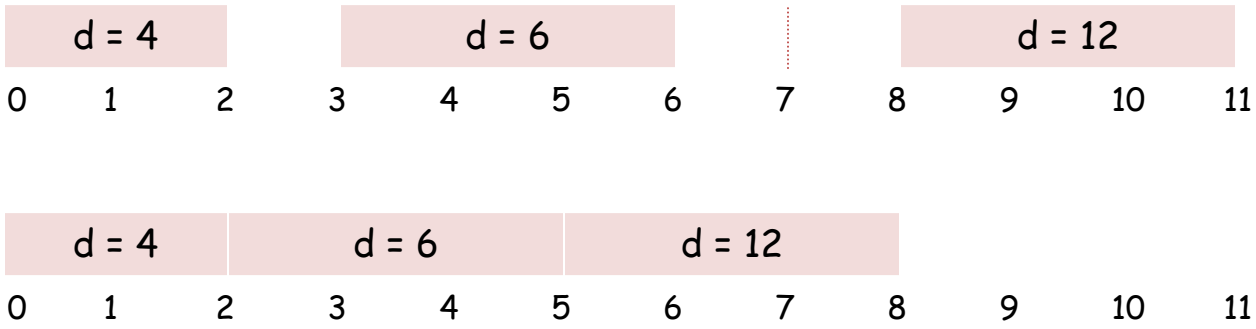
Minimizing Lateness: No Idle Time

Il nostro scopo è adesso dimostrare che:

fra tutte le possibili soluzioni ottimali c'è anche la soluzione fornita dall'algoritmo greedy (cioè lo scheduling che inserisce le attività una dopo l'altra in ordine crescente di deadline)

Observation. The greedy schedule has no idle time (tempo di inattività).

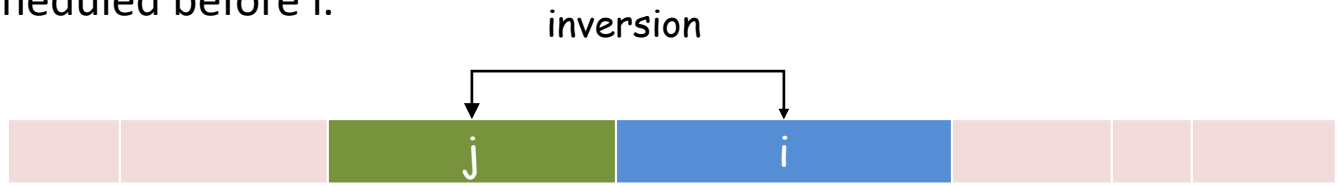
Observation. There exists an optimal schedule with no idle time.



Minimizing Lateness: Inversions

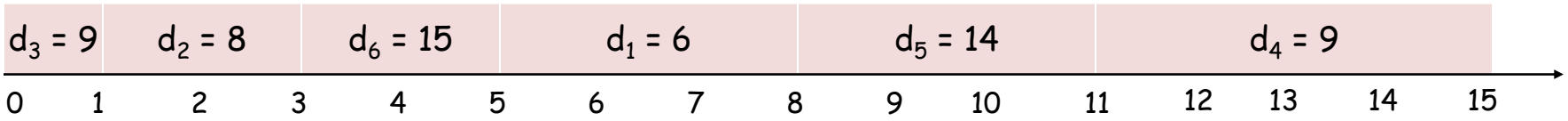
Recall: jobs sorted in ascending order of due dates d_i.

Def. An inversion in schedule S is a pair of jobs i and j such that:
i < j but j scheduled before i.



Esempio:

3 -2, 3-1, 2-1, 6-1, 6-5, 6-4, 5-4



Observation. Greedy schedule has no inversions.

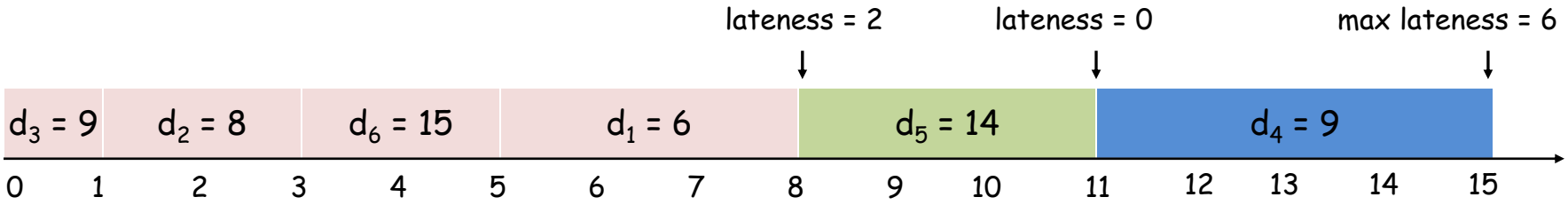
Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively (in rosso nell'esempio sopra).

Altrimenti: la deadline del primo ≤ deadline del secondo ≤ deadline del terzo, etc: nessuna inversione!

Riprendiamo l'esempio

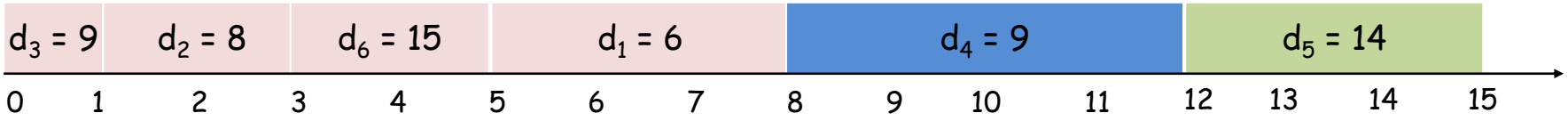
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

Inversioni: 3 -2, 3-1, 2-1, 6-1, 6-5, 6-4, 5-4



- Il ritardo di questo scheduling è max{0, 0, 0, 2, **0**, **6**} = 6

- Scambiamo 5 con 4:

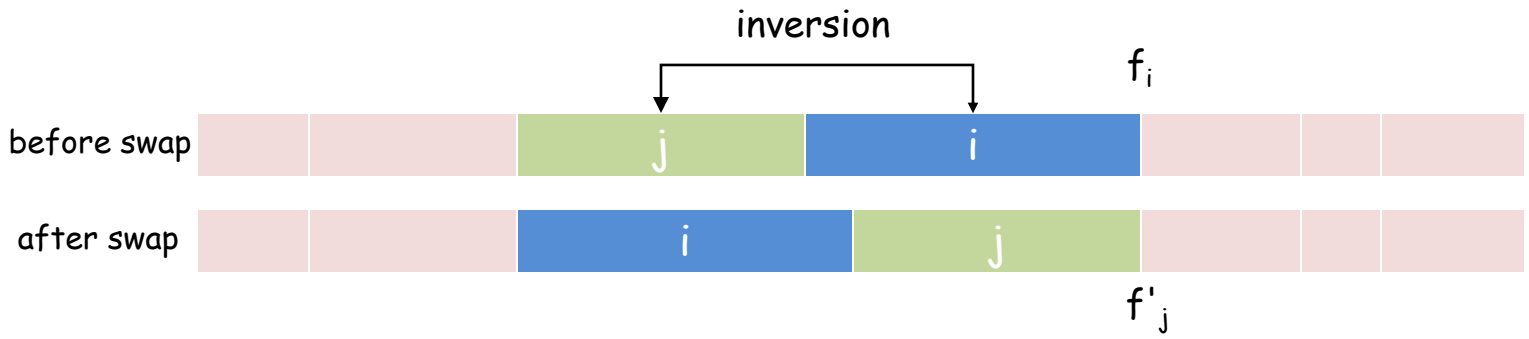


Adesso le inversioni sono: 3 -2, 3-1, 2-1, 6-1, 6-5, 6-4: una in meno!

- Il ritardo dopo lo scambio è max{0, 0, 0, 2, **3**, **1**} = 3 < 6

Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that:
i < j but j scheduled before i.



Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let λ be the lateness before the swap, and let λ' be it afterwards.

$\lambda'_k = \lambda_k$ for all $k \neq i, j$

$\lambda'_i \leq \lambda_i$

If job j is late:

$$\begin{aligned} \lambda'_j &= f'_j - d_j && \text{(definition)} \\ &= f_i - d_j && (j \text{ finishes at time } f_i) \\ &\leq f_i - d_i && (i < j) \\ &\leq \lambda_i && \text{(definition)} \end{aligned}$$

Quindi: nel calcolo del massimo fra i ritardi non troviamo valori maggiori

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

Can assume S* has no idle time.

If S* has no inversions, then S = S*.

If S* has an inversion, let i-j be an adjacent inversion.

swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions

this contradicts definition of S* ▪

Greedy Analysis Strategies

(Strategie per provare la correttezza di algoritmi greedy)

• **Greedy algorithm stays ahead**. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's. (esempio: scheduling di intervalli)

• **Structural**. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound. (esempio: partizionamento di intervalli)

• **Exchange argument**. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality. (esempio: scheduling to minimize lateness)

Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex: 34¢.

La moneta da 1 ci deve essere sempre

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex: $2.89.

Coin-Changing: Analysis of Greedy Algorithm

Observation. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

Counterexample. 140¢.

Greedy: 100, 34, 1, 1, 1, 1, 1, 1.

Optimal: 70, 70.

Cambio monete: Esercizio

Conio euro: 1, 2, 5, 10, 20, 50, 100, 200.

Greedy è ottimale?

Coin-Changing: Greedy Algorithm

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value: c_1 < c_2 < … < c_n.

S ← ϕ                                   (coins selected)
while (x ≠ 0) {
   let k be largest integer such that c_k ≤ x
   if (k = 0)
      return "no solution found"
   x ← x - c_k
   S ← S ∪ {k}
}
return S
```

Q. Is cashier's algorithm optimal?

Cambio di monete (1, 5, 10)

Quesito 3 (*Cambio monete greedy*)
Si consideri il problema di determinare il minimo numero di monete da 1, 5, 10 centesimi necessarie per scambiare un ammontare di denaro D, supponendo di poter utilizzare un numero illimitato di monete dello stesso taglio.
a) Descrivere ed analizzare un algoritmo *greedy* che risolve tale problema
b) Dimostrare la correttezza dell'algoritmo proposto

Lampioni (Greedy)

Una società elettrica vuole piazzare dei lampioni lungo un viale rettilineo dove sono situate delle villette, in modo da illuminare tutte le villette. Ogni lampione riesce ad illuminare tutta la zona compresa nel raggio di 500 metri. Percorrendo il viale si incontrano, in ordine, le villette $v_1, v_2, \dots, v_n$. Per ogni $i = 1, \dots, n-1$, sia $d_{i,i+1}$ la distanza della villetta v_i da v_{i+1}. Immaginiamo che le villette siano «puntiformi».

a) **Descrivere** ed **analizzare** un algoritmo che presi in input i valori $d_{i,i+1}$ restituisca il numero minimo di lampioni necessari per illuminare tutte le villette.
b) **Argomentare** la correttezza dell'algoritmo.

Dal file sulla piattaforma (Esercizi tecnica greedy_070517.pdf):

Quesito 4 (*Zaino semplice*)

a) Definire formalmente il problema dello zaino 0-1.
b) Descrivere un algoritmo *greedy* che risolva il problema dello zaino 0-1, nelle ipotesi che tutti gli oggetti abbiamo lo stesso valore. Fornirne un esempio.
c) Dimostrare che l'algoritmo descritto risolve correttamente il problema.

Quesito 5 (*Ricoprimento numeri*)

Si consideri il problema di calcolare, dato un insieme di numeri reali, il più piccolo insieme di intervalli chiusi di lunghezza unitaria la cui unione contenga (ovvero ricopra) tutti i numeri dati.
Per esempio: se l'insieme è {0.8, 0.9, 1.65, 2, 2.789, 3.5, 4.1}, allora gli insiemi di intervalli {[0.7, 1.7], [2, 3], [3.1, 4.1]} e {[0,1], [1.5, 2.5], [2.7, 3.7], [4, 5]} sono due insiemi di intervalli come richiesti; il primo insieme ha 3 intervalli, il secondo 4.

a) Descrivere un algoritmo goloso (*greedy*) che risolva il problema.
b) Giustificarne la correttezza.

Dalla seconda prova intercorso 2021

Quesito 3 (36 punti)

Un processore deve eseguire **n** processi a partire dall'ora **0**. Ogni processo ha una sua **durata** prevista. Il problema è quello di pianificare l'esecuzione di **tutti** i processi in modo da **minimizzare** il **tempo medio di attesa** (definito come la media aritmetica dei tempi di attesa), dove il **tempo di attesa** di ogni processo è pari al tempo in cui sarà stabilito l'inizio della sua esecuzione.

a) Definire il **problema** computazionale (specificandone i dati in ingresso e in uscita). Si tratta di un problema studiato?

b) Descrivere un **algoritmo *greedy*** per risolvere il problema. Si potrà avere il massimo della votazione se l'algoritmo è descritto tramite **pseudocodice** ed è **commentato** per chiarirne il funzionamento.

c) Dimostrare la **correttezza** dell'algoritmo proposto.