Algoritmi greedy (parte II)

Progettazione di Algoritmi a.a. 2022-23
Matricole congrue a 1
Docente: Annalisa De Bonis

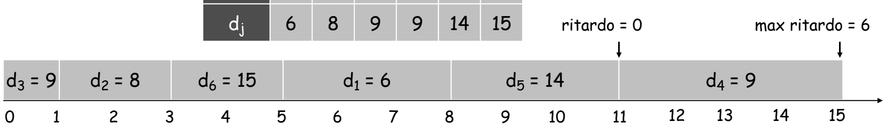
Scheduling per Minimizzare i Ritardi

Problema della minimizzazione dei ritardi.

- Una singola risorsa in grado di elaborare un unico job.
- Il job j richiede t_j unità di tempo e deve essere terminato entro il tempo d_j (scadenza).
- Considerazione: Se j comincia al tempo s_j allora finisce al tempo $f_j = s_j + t_j$.
- Def. Ritardo del job j è definito come $\ell_j = \max \{ 0, f_j - d_j \}$.
- Obiettivo: trovare uno scheduling di tutti i job che minimizzi il ritardo massimo $L = \max \ell_j$.

Esempio:

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



Scheduling per Minimizzare i Ritardi

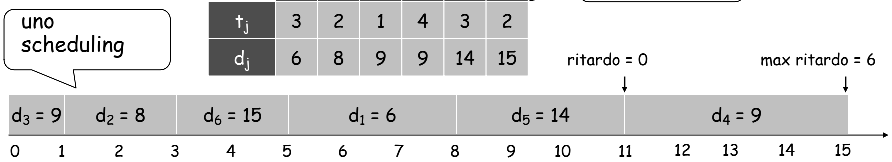
Problema della minimizzazione dei ritardi.

- Input: n job ciascuno dei quali richiede t_j unità di tempo e deve essere terminato entro il tempo d_j (scadenza).
- Obiettivo: trovare uno scheduling di tutti i job che minimizzi il ritardo massimo $L = \max \ell_j$.
  - $\ell_j = \max \{ 0, f_j - d_j \}$
- Uno scheduling assegna ad ogni job j un tempo di inizio s_j

Esempio:

input

	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: Algoritmo Greedy

Schema greedy. Considera i job in un certo ordine.

- [Shortest processing time first] Considera i job in ordine non decrescente dei tempi di elaborazione t_j.

- [Earliest deadline first] Considera i job in ordine non decrescente dei tempi entro i quali devono essere ultimati d_j.

- [Smallest slack] Considera i job in ordine non decrescente degli scarti $d_j - t_j$.

Minimizzare il ritardo: Algoritmo Greedy

- [Shortest processing time first] Considera i job in ordine non decrescente dei tempi di elaborazione t_j.

	1	2
t_j	1	10
d_j	100	10

controesempio

Viene eseguito prima il job 1. Ritardo massimo è 11-10=1. Se avessimo eseguito prima il job 2 avremmo avuto ℓ_1 = max{0, 10-10}=0 e ℓ_2 = max{0,11-100}=0 per cui il ritardo massimo sarebbe stato 0.

- [Smallest slack] Considera i job in ordine non decrescente degli scarti $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

controesempio

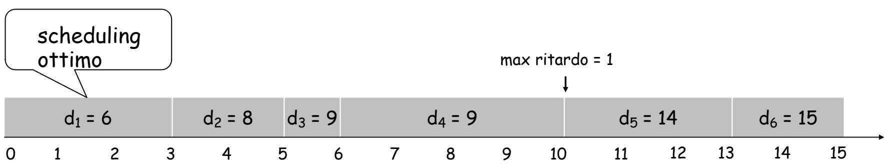
Viene eseguito prima il job 2. Ritardo massimo è 11-2=9. Se avessimo eseguito prima il job 1 il ritardo massimo sarebbe stato 11-10=1

Minimizzare il ritardo: Algoritmo Greedy

Algoritmo greedy. Earliest deadline first: Considera i job in ordine non decrescente dei tempi d_j entro i quali devono essere ultimati.

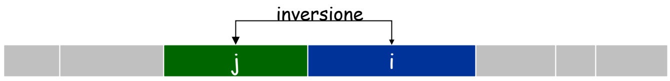
```
MinRitardo(t₁,t₂ , … ,tₙ , d₁,d₂ , … ,dₙ)
   Sort n jobs by deadline so that d₁ ≤ d₂ ≤ … ≤ dₙ

   t ← 0
   for j = 1 to n
     Assign job j to interval [t, t + tⱼ]
     sⱼ ← t, fⱼ ← t + tⱼ
     t ← t + tⱼ
   output intervals [s₁,f₁],...,[sₙ, fₙ ]
```



Minimizzare il ritardo: Inversioni

Def. Un' inversione in uno scheduling S è una coppia di job i e j tali che: $d_i < d_j$ ma j viene eseguito prima di i.



$d_i < d_j$

Ottimalità soluzione greedy

La dimostrazione dell'ottimalità si basa sulle seguenti osservazioni che andremo poi a dimostrare

1. La soluzione greedy ha le seguenti due proprietà:
   a. Nessun idle time. Non ci sono momenti in cui la risorsa non è utilizzata
   b. Nessuna inversione. Se un job j ha scadenza maggiore di quella di un job i allora viene eseguito dopo i

2. Tutte le soluzioni che hanno in comune con la soluzione greedy le caratteristiche a e b, hanno lo stesso ritardo massimo della soluzione greedy.

3. Ogni soluzione ottima può essere trasformata in un'altra soluzione ottima per cui valgono la a e la b

Si noti che la 3 implica che esiste una soluzione ottima che soddisfa la a e la b e, per la 2, questa soluzione ha lo stesso ritardo massimo della soluzione greedy che quindi è a sua volta ottima.

Dimostrazioni delle osservazioni 1. 2. e 3.

1. La soluzione greedy ha le seguenti due proprietà:

a. Nessun idle time. Non ci sono momenti in cui la risorsa non è utilizzata
b. Nessuna inversione. Se un job j ha scadenza maggiore di quella di un job i allora viene eseguito dopo i

Dim.

Il punto a discende dal fatto che ciascun job comincia nello stesso istante in cui finisce quello precedente.

Il punto b discende dal fatto che i job sono esaminati in base all'ordine non decrescente delle scadenze.

Dimostrazioni delle osservazioni 1. 2. e 3.

Prima di dimostrare il punto 2 consideriamo i seguenti fatti

Fatto I. In uno scheduling con le caratteristiche a e b i job con una stessa scadenza d sono disposti uno di seguito all'altro.

Dim.

- Consideriamo i e j con $d_i=d_j=d$ e assumiamo senza perdere di generalità (da ora in poi s.p.d.g.) che i venga eseguito prima di j .
- Supponiamo **per assurdo** che tra i e j venga eseguito il job q con $d \neq d_q$.
- Se $d < d_q$ allora la coppia j, q è un'inversione. Se $d > d_q$ allora la coppia i, q è un'inversione. Ciò contraddice la proprietà b.

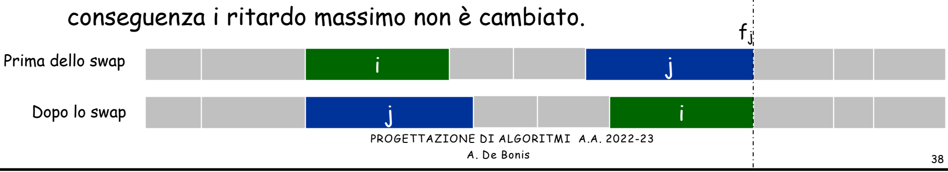
Ne consegue che tra due job con una stessa scadenza d non vengono eseguiti job con scadenza diversa da d e poiché lo scheduling non ha idle time, i job con una stessa scadenza vengono eseguiti uno di seguito all'altro.

Dimostrazioni delle osservazioni 1. 2. e 3.

Fatto II. Se in uno scheduling con le caratteristiche a e b scambiamo due job con la stessa scadenza, il ritardo massimo non cambia.

Dim.

- Consideriamo due job i e j con d_i=d_j e supponiamo s.p.d.g. che i preceda j in S.
- Per il fatto I, tra i e j vengono eseguiti solo job con la stessa scadenza di i e j. Ovviamente il ritardo di j è maggiore del ritardo di i e dei ritardi di tutti i job eseguiti tra i e j perché j finisce dopo tutti questi job e ha la loro stessa scadenza.
- Se scambiamo i con j in S otteniamo che il ritardo di j non può essere aumentato mentre quello di i è diventato uguale a quello che aveva prima j in quanto i finisce nello stesso istante in cui finiva prima j e la scadenza di i è la stessa di j. Il ritardo dei job compresi tra i e j potrebbe essere aumentato ma non può superare il ritardo che aveva prima j. Di conseguenza i ritardo massimo non è cambiato.



Dimostrazioni delle osservazioni 1. 2. e 3.

2. Tutte le soluzioni che hanno in comune con la soluzione greedy le caratteristiche a e b, hanno lo stesso ritardo massimo della soluzione greedy

Dim.

- Dimostriamo che dati due scheduling S ed S' di n job entrambi aventi le caratteristiche a e b, S può essere trasformato in S' senza che il suo ritardo massimo risulti modificato.
- Osserviamo che S ed S' possono differire solo per il modo in cui sono disposti tra di loro job con la stessa scadenza altrimenti o S o S' conterrebbero un'inversione.
- Di conseguenza S può essere trasformato in S' scambiando tra di loro di posto coppie di job con la stessa scadenza.
- Per il fatto II, scambiando coppie di job con la stessa scadenza il ritardo max non cambia. Di conseguenza possiamo trasformare S in S' senza che cambi il ritardo max. In altre parole S ed S' hanno lo stesso ritardo max.
- Prendendo S uguale ad un qualsiasi scheduling con le caratteristiche a e b ed S' uguale allo scheduling greedy si ottiene la tesi.

Una considerazione sull'osservazione 2

- Tutti gli scheduling che hanno le proprieta` a e b assegnano ai job intervalli consecutivi (per la a) che hanno tempi di inizio che crescono al crescere delle scadenze (per la b).
- A prescindere dall'algoritmo che ha generato lo scheduling, uno scheduling siffatto differira` da quello restituito dall'algoritmo nella slide 32 solo per come sono disposti tra di loro i job con la stessa scadenza e avra` , per il fatto II, lo stesso ritardo massimo.
- Notiamo inoltre che il modo in cui sono disposti i job con la stessa deadline nello scheduling restituito dall'algoritmo nella slide 32 dipende da come l'algoritmo di ordinamento ordina tra di loro questi job.

40

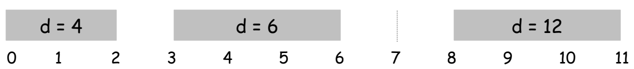
40

Dimostrazioni delle osservazioni 1. 2. e 3.

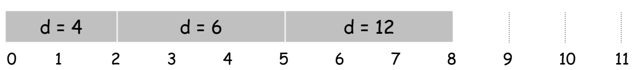
Prima di dimostrare il punto 3 consideriamo i seguenti fatti.

Fatto III. Una soluzione ottima può essere trasformata in una soluzione ottima con nessun tempo di inattività (idle time).
Dim. Se tra il momento in cui finisce l'elaborazione di un job e quello in cui inizia il successivo vi è un gap, basta shiftare all'indietro l'inizio del job successivo in modo che cominci non appena finisce il precedente. Ovviamenti i ritardi dei job non aumentano dopo ogni shift

Esempio: Soluzione ottima con idle time



Esempio: Soluzione ottima con nessun idle time



41

Dimostrazioni delle osservazioni 1. 2. e 3.

Fatto IV. Se uno scheduling privo di idle time ha un'inversione allora esso ha una coppia di job invertiti che cominciano uno dopo l'altro.

Dim.

- Consideriamo tutte le coppie di job i e j tali che $d_i < d_j$ e j viene eseguito prima di i nello scheduling. Supponiamo per assurdo tutte le coppie di questo tipo siano separate da un job.
- Tra tutte le coppie siffatte prendiamo quella più vicina nello scheduling. Deve esistere un job k≠i eseguito subito dopo j che non forma un'inversione né con i né con j altrimenti i e j non formerebbero l'inversione più vicina.
- Deve quindi essere $d_j \leq d_k$ e $d_k \leq d_i$. Le due diseguaglianze implicano $d_j \leq d_i$ il che contraddice il fatto che la coppia i, j sia un'inversione.

Abbiamo dimostrato che se uno scheduling contiene inversioni allora deve esistere una coppia di job invertiti tra i quali non viene eseguito nessun altro job. Siccome lo scheduling considerato non contiene idle time allora questi due job invertiti devono cominciare uno dopo l'altro

Dimostrazioni delle osservazioni 1. 2. e 3.

Fatto V. Scambiare due job adiacenti invertiti i e j riduce il numero totale di inversioni di uno e non fa aumentare il ritardo massimo.

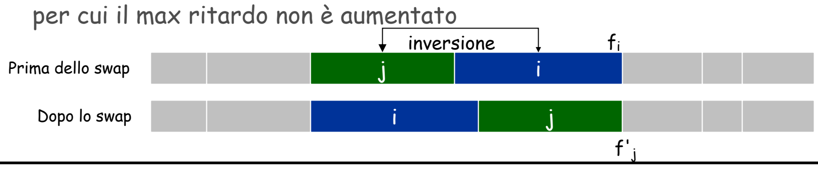
Dim. Supponiamo $d_i < d_j$ e che j precede i nello scheduling.

Siano $\ell_1, \ldots, \ell_n$ i ritardi degli n job e siano $\ell'_1, \ldots, \ell'_n$ i ritardi degli n job dopo aver scambiato i e j di posto. Si ha che

- $\ell'_k = \ell_k$ per tutti i $k \neq i, j$
- $\ell'_i < \ell_i$ perchè viene anticipata la sua esecuzione.
- Vediamo se il ritardo di j è aumentato al punto da far aumentare il ritardo max. Ci basta considerare il caso in cui $\ell'_j > 0$ altrimenti vuol dire che il ritardo di j non è aumentato. Si ha quindi

$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \text{(per la definizione di ritardo)} \\
&= f_i - d_j && \text{(dopo lo swap, j finisce al tempo } f_i\text{)} \\
&< f_i - d_i && \text{(in quanto } d_i < d_j\text{)} \\
&\leq \ell_i && \text{(per la definizione di ritardo)}
\end{aligned}
$$

per cui il max ritardo non è aumentato



Dimostrazioni delle osservazioni 1. 2. e 3.

- 3. Ogni soluzione ottima S può essere trasformata in un'altra soluzione ottima per cui valgono la a e la b

- Dim.
- Il fatto III implica che la soluzione ottima S può essere trasformata in una soluzione ottima S' per cui non ci sono idle time.
- Il fatto IV implica che se la soluzione ottima S' contiene inversioni allora S' contiene una coppia di job **adiacenti** invertiti.
  - Il fatto V implica che se scambiamo le posizioni di questi due job invertiti adiacenti il ritardo massimo non cambia per cui otteniamo ancora una soluzione ottima con un numero inferiore di inversioni.
- Quindi se S' contiene inversioni, possiamo scambiare di posto coppie di job adiacenti invertiti fino a che non ci sono più inversioni e la soluzione S'' così ottenuta sarà a sua volta ottima.

Esercizio taglio tubo

- Abbiamo un tubo metallico di lunghezza L. Da questo tubo vogliamo ottenere al piu` n segmenti più corti, aventi rispettivamente lunghezze lung[1], lung[2], …, lung[n]. Il tubo viene segato sempre a partire da una delle estremità, quindi ogni taglio riduce la sua lunghezza della misura asportata. Descrivere un algoritmo greedy per determinare il numero massimo di segmenti che è possibile ottenere.
- Input: valori positivi L , lung[1], lung[2], …, lung[n].
- Obiettivo: selezionare il piu` grande sottoinsieme S di {1,2,…,n} in modo che la somma dei valori lung[i] per i in S non superi L.

Soluzione.

- Ordiniamo lung[1], lung[2], …, lung[n] in ordine non decrescente. Siano lung'[1], lung'[2], …, lung'[n] le lunghezze cosi` ordinate. Tagliamo prima un segmento di lunghezza lung'[1], poi quello di lunghezza lung'[2] e cosi` via fino a che ad un certo punto non riusciamo ad ottenere altri segmenti.
- Possiamo dimostrare che questa strategia greedy è ottima con la tecnica dello scambio. Siano $g_1,\dots,g_p$ i segmenti tagliati da greedy e $o_1,\dots,o_q$ quelli della soluzione ottima. Entrambe le sequenze sono ordinate in base all'ordine non decrescente delle lunghezze dei segmenti. Supponiamo che fino ad un certo $j\geq 0$ si abbia $g_1=o_1,\dots,g_j= o_j$. Facciamo vedere che se rimpiazziamo o_{j+1} con g_{j+1} in $o_1,\dots,o_q$ otteniamo ancora una soluzione ottima.

continua

Esercizio taglio tubo

- Se si ha gia` g_{j+1}= o_{j+1} allora la dimostrazione è conclusa. In caso contrario, mostriamo che se rimpiazziamo o_{j+1} con g_{j+1} in $o_1,\ldots,o_q$, otteniamo ancora una soluzione al problema senza ridurrre il numero di segmenti della soluzione. Notiamo prima di tutto che g_{j+1} non è uno dei $o_{j+1},\ldots,o_q$ per cui puo` essere messo al posto di $o_{j+1}$ nella soluzione ottima. Infatti, per come funziona greedy, $g_{j+1}$ è la lunghezza del segmento piu` corto tra quelli di lunghezza maggiore o uguale di $g_1,\ldots,g_j$. Ne consegue che se g_{j+1} è diverso da o_{j+1} allora g_{j+1} puo` essere solo minore di o_{j+1} e di conseguenza è minore (diverso) anche dai successivi $o_{j+2},\ldots,o_q$. Se rimpiazziamo o_{j+1} con g_{j+1}, la parte che resta da tagliare del tubo è maggiore rispetto a quella che si aveva prima della sostituzione per cui anche le scelte successive $o_{j+2},\ldots,o_q$ saranno possibili.

- Abbiamo dimostrato che per ogni j≥0, un algoritmo ottimo che fa le stesse prime j scelte di greedy puo` essere trasformato in un algoritmo ottimo che fa le stesse j+1 scelte di greedy.
- A partire da j=0 e fino ad arrivare a j=p possiamo quindi rimpiazzare man mano ogni scelta o_j con g_j in modo da ottenere dopo ogni scambio una soluzione ancora ottima (cioe` di q segmenti). Da cio ` si deduce che il numero q di segmenti della soluzione ottima non puo` essere maggiore del numero di segmenti p della soluzione greedy perche' se cosi` fosse l'algoritmo greedy si fermerebbe dopo aver tagliato p segmenti mentre l'algoritmo ottimo tagliarebbe almeno un altro segmento. Cio` è impossibile in quanto dopo aver tagliato il p-esimo segmento, gli algoritmi hanno a disposizione una porzione di tubo della stessa lunghezza.

46