Algoritmi greedy

Progettazione di Algoritmi a.a. 2019-20
Matricole congrue a 1
Docente: Annalisa De Bonis

1

Scelta greedy

Un algoritmo greedy è un algoritmo che effettua ad ogni passo la scelta che in quel momento sembra la migliore (localmente ottima) nella speranza di ottenere una soluzione globalmente ottima.

Domanda. Questo approccio porta sempre ad una soluzione ottima?
Risposta. Non sempre ma per molti problemi sì.
Esempio:
Voglio andare in auto dalla città a alla città b. La distanza tra a e b e di 1500 km. Lungo la strada ci sono n punti in cui posso fermarmi a fare rifornimento di carburante (a e b compresi). Voglio minimizzare il numero di volte in cui devo fermarmi per fare rifornimento considerando che con un pieno posso percorrere 450 km e che alla partenza il serbatoio è vuoto.
Strategia greedy: faccio il pieno in a e arrivo al più lontano punto di rifornimento che riesco a raggiungere (distanza da a ≤ 450 km). Da questo punto raggiungo il punto di rifornimento più lontano che si trova ad al più 450 km da esso, e così via fino a che non arrivo in b. Si puo` dimostrare che questa strategia fornisce la soluzione ottima.

2

Scelta greedy

- Esempio di problema per cui questo approccio non porta alla soluzione ottima
- Problema dello zaino
- Input
  - n oggetti ed uno zaino
  - L'oggetto i pesa $w_i > 0$ chili e ha valore $v_i > 0$.
  - Lo zaino puo` trasportare fino a W chili.
- Obiettivo: riempire lo zaino in modo da massimizzare il valore totale degli oggetti inseriti tenendo conto che lo zaino trasporta al più W chili
- Esempio: { 3, 4 } ha valore 40.

W = 11

Oggetto	Valore	Peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Proviamo diverse strategie greedy

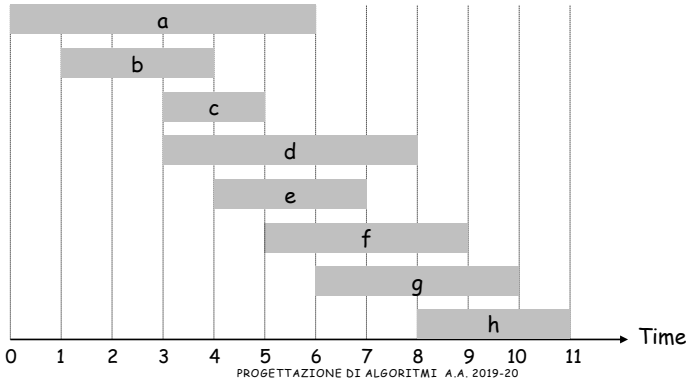
1. seleziona ad ogni passo l'oggetto con valore v_i più grande in modo che il peso totale dei pesi selezionati non superi w → {5,2,1} valore=35
2. seleziona ad ogni passo l'oggetto con peso w_i più piccolo in modo che il peso totale dei pesi selezionati non superi w → {1,2,5} valore=25
3. seleziona ad ogni passo l'oggetto con il rapporto v_i/w_i più grande in modo che il peso totale dei pesi selezionati non superi w → { 5,2,1 }, valore = 35

3

Interval Scheduling

Interval scheduling.

- Input: un insieme di attività ognuna delle quali inizia ad un certo instante s_j e finisce ad un certo istante f_j. Può essere eseguita al più un'attivita alla volta.
- Obiettivo: fare in modo che vengano svolte quante più attività è possibile.

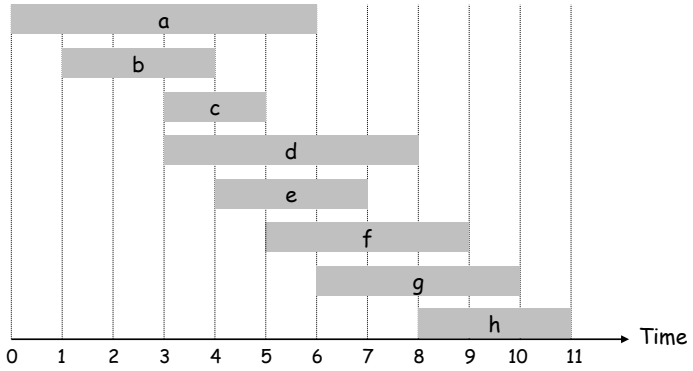


4

Interval Scheduling

Due job i e j si dicono compatibili se $f_i \le s_j$ oppure $f_j \le s_i$.

- Possiamo riformulare l'obiettivo del problema nel modo seguente.
- Obiettivo: trovare un sottoinsieme di cardinalità massima di job a due a due compatibili.



5

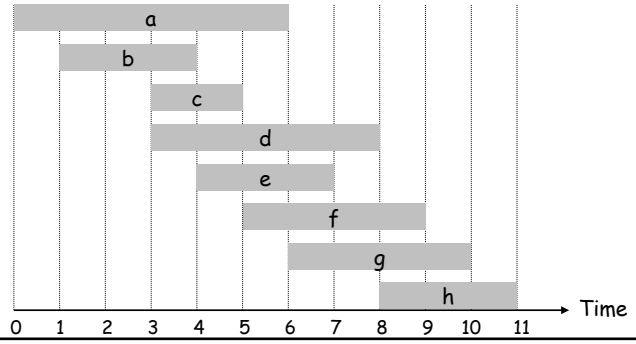
Interval Scheduling

Se all'inizio scegliamo a poi possiamo scegliere o g o h.

- Dopo aver scelto a e g oppure a e h, non possiamo scegliere nessun altro job. Totale = 2

Se all'inizio scegliamo b poi possiamo scegliere uno tra e, f, g e h.

- Se dopo b scegliamo e poi possiamo scegliere anche h. Totale = 3
- Se dopo b scegliamo f poi non possiamo scegliere nessun altro job. Totale = 2



6

Interval Scheduling: Algoritmi Greedy

Schema greedy. Considera i job in un certo ordine. Ad ogni passo viene esaminato il prossimo job nell'ordinamento e se il job è compatibile con quelli scelti nei passi precedenti allora il job viene inserito nella soluzione.

L'ordinamento dipende dal criterio che si intende ottimizzare localmente.

Diverse strategie basate su diversi tipi di ordinamento

- [Earliest start time] Considera i job in ordine crescente rispetto ai tempi di inizio s_j. Scelta greedy consiste nel provare a prendere ad ogni passo il job che inizia prima tra quelli non ancora esaminati.
- [Earliest finish time] Considera i job in ordine crescente rispetto ai tempi di fine f_j. Scelta greedy consiste nel provare a prendere ad ogni passo il job che finisce prima tra quelli non ancora esaminati.
- [Shortest interval] Considera i job in ordine crescente rispetto alle lloro durate $f_j - s_j$. Scelta greedy consiste nel provare a prendere ad ogni passo il job che dura meno tra quelli non ancora esaminati.
- [Fewest conflicts] Per ogni job, conta il numero c_j di job che sono in conflitto con lui e ordina in modo crescente rispetto al numero di conflitti. Scelta greedy consiste nel provare a prendere ad ogni passo il job che ha meno conflitti tra quelli non ancora esaminati.

7

7

Interval Scheduling: Algoritmi Greedy

La strategia "Earliest Start Time" sembra la scelta più ovvia ma…

Problemi con la strategia "Earliest Start Time". Può accadere che il job che comincia per primo finisca dopo tutti gli altri o dopo molti altri.

8

Interval Scheduling: Algoritmi Greedy

Ma se la lunghezza dei job selezionati incide sul numero di job che possono essere selezionati successivamente perché non provare con la strategia "Shortest Interval"?

Questa strategia va bene per l'input della slide precedente ma…

Problemi con la strategia "Shortest Interval". Può accadere che un job che dura meno di altri si sovrapponga a due job che non si sovrappongono tra di loro. Se questo accade invece di selezionare due job ne selezioniamo uno solo.

9

Interval Scheduling: Algoritmi Greedy

Visto che il problema sono i conflitti, perché non scegliamo i job che confliggono con il minor numero di job?

Questa strategia va bene per l'input nella slide precedente ma…

Problemi con la strategia "Fewest Conflicts". Può accadere che un job che genera meno conflitti di altri si sovrapponga a due job che non si sovrappongono tra di loro. Se applichiamo questa strategia all'esempio in questa slide, invece di selezionare 4 job ne selezioniamo solo 3.

10

Interval Scheduling: Algoritmo Greedy Ottimo

L'algoritmo greedy che ottiene la soluzione ottima è quello che usa la stategia "Earliest Finish Time".

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
f ← 0
A ← ϕ
for j = 1 to n {
   if (sj ≥ f)
      A ← A ∪ {j}
      f ← fj
}
return A
```

Analisi tempo di esecuzione. $O(n \log n)$.

- Costo ordinamento $O(n \log n)$
- Costo for $O(n)$: mantenendo traccia del tempo di fine f dell'ultimo job selezionato, possiamo capire se il job j è compatibile con A verificando che $s_j \geq f$

11

11

Interval Scheduling: Ottimalità soluzione greedy

Teorema. L'algoritmo greedy basato sulla strategia "Earliest Finish Time" è ottimo.

Dim.

- Denotiamo con $i_1, i_2, \ldots i_k$ l'insieme di job selezionati dall'algoritmo greedy nell'ordine in cui sono selezionati, cioè in ordine non decrescente rispetto ai tempi di fine.
- Denotiamo con $j_1, j_2, \ldots j_m$ l'insieme di job nella soluzione ottima, disposti in ordine non decrescente rispetto ai tempi di fine.

1. Dimostriamo prima che l'esecuzione dei job $i_1, i_2, \ldots i_k$ termina non più tardi di quella dei job $j_1, j_2, \ldots j_k$
2. Usiamo il punto 1 per dimostrare che non è possibile che k sia minore di m e che quindi la soluzione greedy è ottima.

- NB: Supponiamo di ordinare i job in input in base ai tempi di fine e siano x e y due job t.c. x che precede y nell'ordinamento. I job x e y sono compatibili ←→ il tempo di fine di x è ≤ del tempo di inizio di y.
  Dim. Per definizione di job compatibili x e y sono compatibili ←→ x termina non più tardi dell'inizio di y oppure y termina non più tardi dell'inizio di x.
  Siccome x precede y nell'ordinamento allora solo il primo dei due casi è possibile.

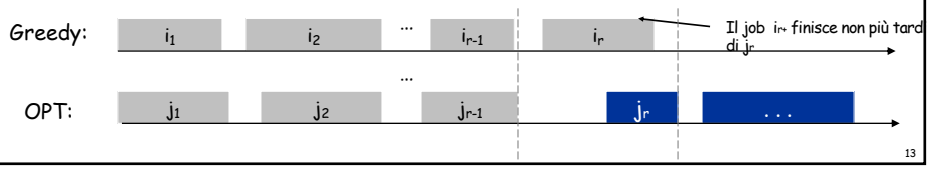
12

12

Interval Scheduling: Ottimalità soluzione greedy

Dimostriamo il punto 1.:

- Dimostriamo per induzione che per ogni indice r, $1 \le r \le k$, si ha che il tempo di fine di i_r è **non** più grande di quello di j_r.
- **Base.** r=1 : Banalmente vero perchè la prima scelta greedy seleziona la richiesta con il minimo tempo di fine.
- **Passo induttivo. Ipotesi induttiva:** per $r-1 \ge 1$, il tempo di fine di i_{r-1} è **non** più grande di quello di j_{r-1}.
- Il job j_r è compatibile con j_{r-1} e quindi $f(j_{r-1}) \le s(j_r)$ e siccome per ipotesi induttiva $f(i_{r-1}) \le f(j_{r-1})$ → $f(i_{r-1}) \le s(j_r)$ → j_r è compatibile con i_{r-1}.
- I tempi di fine di $i_1, i_2, \ldots i_{r-2}$ sono non più grandi di quello di i_{r-1} → $f(i_p) \le s(j_r)$ per ogni $1 \le p \le r-1$ → j_r è compatibile con $i_1, i_2, \ldots i_{r-2}\ i_{r-1}$.
- All'r-esimo passo l'algoritmo greedy sceglie i_r → i_r finisce non più tardi di tutti i job compatibili con $i_1, i_2, \ldots i_{r-1}$ e quindi anche non più tardi di j_r.



13

13

Interval Scheduling: Ottimalità soluzione greedy

- Abbiamo dimostrato che, per ogni indice $r \le k$, il tempo di fine di i_r è **non** più grande di quello di j_r.
- Di conseguenza il tempo di fine di i_k è non più grande di quello di j_k.
- Dal momento che $i_1, i_2, \ldots i_k$ sono ordinati in base ai tempi di fine allora si ha che anche i tempi di fine di $i_1, i_2, \ldots i_{k-1}$ sono **non** maggiori di quello di j_k.
- Quindi l'esecuzione della sequenza di job $i_1, i_2, \ldots i_k$ termina non più tardi di j_k e quindi possiamo affermare che l'esecuzione della sequenza di job $i_1, i_2, \ldots i_k$ termina non più tardi di quella dei job $j_1, j_2, \ldots j_k$

Continua nella prossima slide

14

14

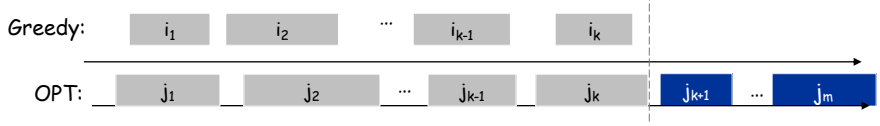
Interval Scheduling: Ottimalità soluzione greedy

Dimostriamo il punto 2.

•Supponiamo per assurdo che la soluzione greedy non sia ottima e quindi che k < m. Quindi la sequenza $j_1, j_2, \dots j_m$ conterrà il job j_{k+1}

• Per il punto 1, l'esecuzione di $i_1, i_2, \dots i_k$ termina **non** più tardi dell'esecuzione di $j_1, j_2, \dots j_k$. e si ha quindi che $i_1, i_2, \dots i_k$ sono compatibili con j_{k+1}

• L'algoritmo greedy, dopo aver inserito $i_1, i_2, \dots i_k$ nella soluzione, passa ad esaminare i job con tempo di fine maggiore o uguale a quelli di $i_1, i_2, \dots i_k$. Tra questi job vi è j_{k+1} che è compatibile con $i_1, i_2, \dots i_k$ per cui è impossibile che l'algoritmo greedy inserisca nella soluzione solo k job. Siamo giunti ad una contraddizione e quindi è impossibile che k < m.



15

15

Provare l'ottimalità della soluzione greedy

Come abbiamo provato l'ottimalità della soluzione greedy?

- Abbiamo prima di tutto dimostrato che la soluzione greedy "sta sempre avanti" a quella ottima.
  - Cosa vuol dire "sta sempre avanti"?
  - L'idea alla base della strategia greedy Earliest Finish Time è la seguente: quando si usa la risorsa è bene liberarla il prima possibile perchè ciò massimizza il tempo a disposizione per eseguire le restanti richieste
  - In questa ottica, una soluzione per il problema dell'interval scheduling "sta sempre avanti" ad un'altra se ad ogni passo seleziona una richiesta che termina non più tardi della corrispondente richiesta della soluzione ottima.
- Abbiamo usato il fatto che la soluzione greedy "sta sempre avanti" a quella ottima per provare che la soluzione greedy non può contenere un numero di richieste inferiore a quello della soluzione ottima

16

16