

Programmazione dinamica (I parte)

Progettazione di Algoritmi a.a. 2022-23

Matricole congrue a 1

Docente: Annalisa De Bonis

1

1

Paradigmi della Progettazione degli Algoritmi

- **Greedy.** Costruisci una soluzione in modo incrementale, ottimizzando (in modo miope) un certo criterio locale.
- **Divide-and-conquer.** Suddividi il problema in sottoproblemi, risolvi ciascun sottoproblema indipendentemente e combina le soluzioni dei sottoproblemi per formare la soluzione del problema di partenza.
- **Programmazione dinamica.** Suddividi il problema in un insieme di sottoproblemi che si sovrappongono, cioè che hanno dei sottoproblemi in comune. Costruisci le soluzioni a sottoproblemi via via sempre più grandi **in modo da computare la soluzione di un dato sottoproblema un'unica volta.**
- Nel divide and conquer, se due sottoproblemi condividono uno stesso sottoproblema quest'ultimo viene risolto più volte.

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

2

2

Storia della programmazione dinamica

- **Bellman.** Negli anni '50 è stato il pioniere nello studio sistematico della programmazione dinamica.
- **Etimologia.**
 - Programmazione dinamica = pianificazione nel tempo.

3

Applicazioni della programmazione dinamica

Aree.

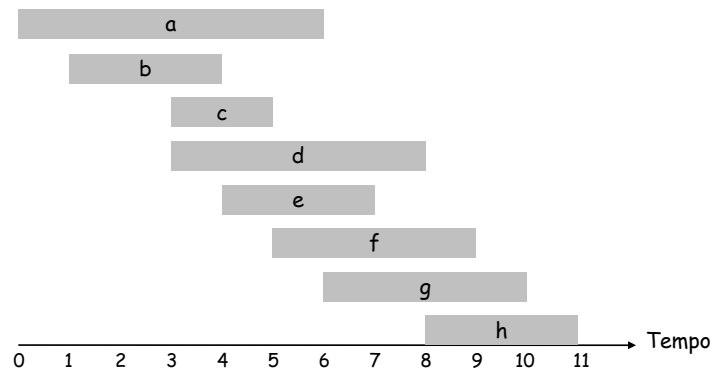
- Bioinformatica.
- Teoria dell'informazione
- Ricerca operativa
- Informatica teorica
- Computer graphics
- Sistemi di Intelligenza Artificiale

4

Interval Scheduling Pesato

Interval scheduling con pesi

- Job j : comincia al tempo s_j , finisce al tempo f_j , ha associato un valore (peso) v_j .
- Due job sono **compatibili** se non si sovrappongono
- Obiettivo: trovare il sottoinsieme di job compatibili con il massimo peso totale.



Progettazione di Algoritmi A.A. 2019-20
A. De Bonis

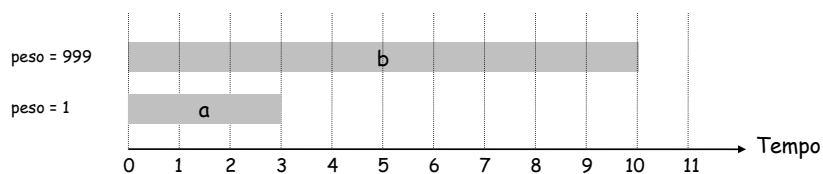
5

5

Interval scheduling senza pesi

- L'algoritmo greedy Earliest Finish Time funziona quando tutti i pesi sono uguali ad 1.
- Considera i job in ordine non decrescente dei tempi di fine
- Seleziona un job se è compatibile con quelli già selezionati

Osservazione. L'algoritmo greedy Earliest Finish Time può fallire se i pesi dei job sono valori arbitrari.



Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

6

6

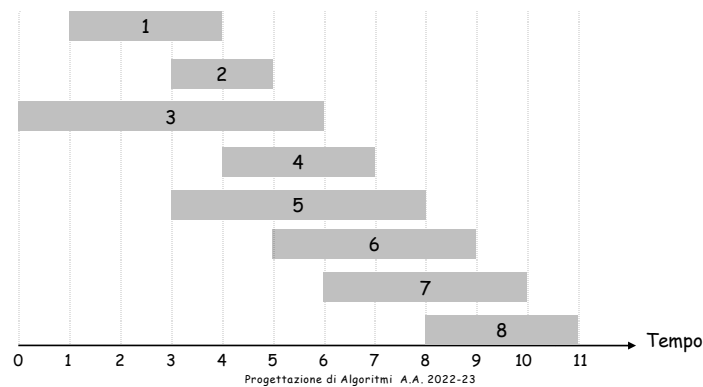
Interval Scheduling Pesato

Notazione. Etichettiamo i job in base al tempo di fine :

$$f_1 \leq f_2 \leq \dots \leq f_n.$$

Def. $p(j)$ = il più grande indice $i < j$ tale che i è compatibile con j

Ex: $p(8) = 5, p(7) = 3, p(2) = 0$.



7

Interval Scheduling Pesato: soluzione basata sulla PD

Notazione. $OPT(j)$ = valore della soluzione ottima per l'istanza del problema dell'Interval Scheduling Pesato costituita dalle j richieste con i j tempi di fine più piccoli

Si possono verificare due casi:

- **Caso 1:** La soluzione ottima per i j job con i tempi di fine più piccoli include il job j .
 - In questo caso la soluzione non può usare i job incompatibili $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$
 - Deve includere la soluzione ottima al problema dell'Interval Scheduling Pesato per i job $1, 2, \dots, p(j)$
- **Caso 2:** La soluzione ottima per i j job con i tempi di fine più piccoli non contiene il job j .
 - In questo caso la soluzione deve includere la soluzione ottima al problema dell'Interval Scheduling Pesato per i job $1, 2, \dots, j-1$

$$OPT(j) = \begin{cases} 0 & \text{if } j=0 \\ \max \{ v_j + OPT(p(j)), OPT(j-1) \} & \text{otherwise} \end{cases}$$

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

8

8

Interval Scheduling Pesato: soluzione basata sulla PD

N.B.

- Quando viene detto che la soluzione per i primi j job con il tempo di fine più piccolo deve includere la soluzione ottima per i job $1, 2, \dots, p(j)$ (nel caso 1) o quella per i job $1, 2, \dots, j-1$ (nel caso 2), NON vuol dire che la soluzione deve includere necessariamente tutti i job $1, 2, \dots, p(j)$ nel caso 1 e i job $1, 2, \dots, j-1$, nel caso 2.
- Quando si deriva la formula di ricorrenza per calcolare il valore della soluzione ottima di un problema non si deve MAI
 - fare riferimento agli algoritmi usati per calcolare il valore della soluzione ottima: sono gli algoritmi ad usare la formula, non è la formula ad essere costruita sulla base dell'algoritmo!
 - ragionare in modo iterativo nella derivazione della formula

9

Interval Scheduling Pesato: algoritmo ricorsivo inefficiente

- Inizialmente Compute-Opt viene invocato con $j=n$

```

Input:  $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$ 
Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
Compute  $p(1), p(2), \dots, p(n)$ 

Compute-Opt( $j$ ) {
  if ( $j = 0$ )
    return 0
  else
    return  $\max(v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1))$ 
}

```

10

Interval Scheduling Pesato: algoritmo ricorsivo inefficiente

L'algoritmo computa correttamente $OPT(j)$

Dim per induzione.

- **Caso base** $j=0$. Il valore restituito è correttamente 0.
- **Passo Induttivo**. Consideriamo un certo $j>0$ e supponiamo (ipotesi induttiva) che l'algoritmo produca il valore corretto di $OPT(i)$ per ogni $i<j$.

Il valore computato per j dall'algoritmo è

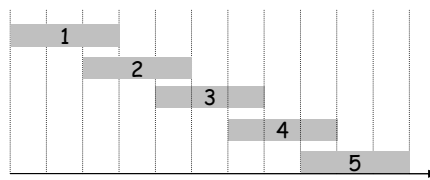
$$\text{Compute-Opt}(j) = \max(v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1))$$

- Siccome per ipotesi induttiva
- **valore computato da $\text{Compute-Opt}(p(j)) = OPT(p(j))$ e**
- **valore computato da $\text{Compute-Opt}(j-1) = OPT(j-1)$**
- allora ne consegue che
- **$\text{Compute-Opt}(j) = \max(v_j + OPT(p(j)), OPT(j-1)) = OPT(j)$**

per la relazione di ricorrenza

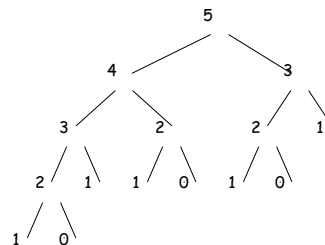
Interval Scheduling Pesato: algoritmo ricorsivo inefficiente

- **Osservazione**. L'algoritmo ricorsivo corrisponde ad un algoritmo di forza bruta perchè ha tempo esponenziale
 - Ciò è dovuto al fatto che
 - ✓ Un gran numero di sottoproblemi sono condivisi da più sottoproblemi
 - ✓ L'algoritmo computa più volte la soluzione ad uno stesso sottoproblema.
- **Esempio**. In questo esempio il numero di chiamate ricorsive cresce come i numeri di Fibonacci.



$$P(1) = 0, p(2) = 0 \text{ e } p(j) = j-2 \text{ per ogni } j > 1$$

- $N(j)$ = numero chiamate ricorsive per j .
- Per ogni $j > 1$ si ha $N(j) = N(j-1) + N(j-2)$



Interval Scheduling Pesato: Memoization

- **Osservazione:** l'algoritmo ricorsivo precedente computa la soluzione di $n+1$ sottoproblemi soltanto $OPT(0), \dots, OPT(n)$. Il motivo dell'inefficienza dell'algoritmo è dovuto al fatto che computa la soluzione ad uno stesso problema più volte.
- **Memoization.** Consiste nell'immagazzinare le soluzioni di ciascun sottoproblema in un'area di memoria accessibile globalmente.

```

Input:  $n, s_1, \dots, s_n, f_1, \dots, f_n, v_1, \dots, v_n$ 

Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ .
Compute  $p(1), p(2), \dots, p(n)$ 

for  $j = 1$  to  $n$ 
     $M[j] = \text{empty}$  ← array globale

M-Compute-Opt( $j$ ) {
    if  $j = 0$  Return 0
    if  $M[j]$  is empty
         $M[j] = \max(v_j + \text{M-Compute-Opt}(p(j)), \text{M-Compute-Opt}(j-1))$ 
    return  $M[j]$ 
}

```

13

13

Interval Scheduling pesato: Tempo di Esecuzione

Affermazione. La versione "memoized" dell'algoritmo ha tempo di esecuzione $O(n \log n)$.

Fase di inizializzazione: $O(n \log n)$

- Ordinamento in base ai tempi di fine: $O(n \log n)$.
- Computazione dei valori $p(\cdot)$: $O(n)$ dopo aver ordinato i job (rispetto ai tempi di inizio e di fine). Siano a_1, \dots, a_n i job ordinati rispetto ai tempi di inizio e b_1, \dots, b_n i job ordinati rispetto ai tempi di fine. (si noti che il job con l' i -esimo tempo di inizio non corrisponde necessariamente a quello con l' i -esimo tempo di fine)
 - Si confronta il tempo di fine di b_1 con i tempi di inizio di a_1, a_2, a_3, \dots , fino a che non si incontra un job a_j con tempo di inizio $\geq f_1$. Si pone $p'(1)=p'(2)=\dots=p'(j-1)=0$. Si confronta il tempo di fine di b_2 con i tempi di inizio di $a_j, a_{j+1}, a_{j+2}, \dots$, fino a che non si incontra un job a_k con tempo di inizio $\geq f_2$. Si pone $p'(j)=p'(j+1)=p'(j+2)=\dots=p'(k-1)=1$. Si confronta il tempo di fine di b_3 con i tempi di inizio di $a_k, a_{k+1}, a_{k+2}, \dots$, fino a che non si incontra un job a_m con tempo di inizio $\geq f_3$. Si pone $p'(k)=p'(k+1)=p'(k+2)=\dots=p'(m-1)=2$, e così via.

Continua nel slide successiva

14

14

Interval Scheduling pesato: Tempo di Esecuzione

- Si noti che in $p'(j)$, j e' l'indice del job nella sequenza a_1, \dots, a_n .
- Per ottenere il corrispondente valore $p(r)$ basta sostituire a j l'indice del job nella sequenza b_1, \dots, b_n . L'associazione tra il job e la sua posizione nella sequenza a_1, \dots, a_n e quella nella sequenza b_1, \dots, b_n puo' essere creata quando si ordinano i job.

Numeri neri= indici nella sequenza ordinata b_1, \dots, b_n
 Numeri rossi= indici nella sequenza ordinata a_1, \dots, a_n

$p'(1)=p'(2)=p'(3)=p'(4)=0$ $p(3)=p(1)=p(2)=p(5)=0$
 $p'(5)=1$ $p(4)=1$
 $p'(6)=2$ $p(6)=2$
 $p'(7)=3$ $p(7)=3$
 $p'(8)=5$ $p(8)=5$

Tempo

Continua nel slide successiva

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

15

Interval Scheduling pesato: Tempo di Esecuzione

- Il tempo per calcolare i valori $p(1), \dots, p(n)$ (una volta ottenuti i due ordinamenti a_1, \dots, a_n e b_1, \dots, b_n) e' $O(n)$ perche' dopo ogni confronto l'algoritmo passa a considerare o il prossimo job nell'ordinamento b_1, \dots, b_n (nel caso di confronto tra due job compatibili) o il prossimo job nell'ordinamento a_1, \dots, a_n (nel caso di confronto tra due job incompatibili).

Numeri neri= indici nella sequenza ordinata b_1, \dots, b_n
 Numeri rossi= indici nella sequenza ordinata a_1, \dots, a_n

$p'(1)=p'(2)=p'(3)=p'(4)=0$ $p(3)=p(1)=p(2)=p(5)=0$
 $p'(5)=1$ $p(4)=1$
 $p'(6)=2$ $p(6)=2$
 $p'(7)=3$ $p(7)=3$
 $p'(8)=5$ $p(8)=5$

Tempo

Continua nel slide successiva

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

16

Interval Scheduling pesato: Tempo di Esecuzione

Affermazione: `M-Compute-Opt (n)` richiede $O(n)$

Dim.

- `M-Compute-Opt (j)`: escludendo il tempo per le chiamate ricorsive, ciascuna invocazione prende tempo $O(1)$ e fa una delle seguenti cose
 - (i) restituisce il valore esistente di $M[j]$
 - (ii) riempie l'entrata $M[j]$ facendo due chiamate ricorsive
- Per stimare il tempo di esecuzione di `M-Compute-Opt (j)` dobbiamo stimare il numero totale di chiamate ricorsive innescate da `M-Compute-Opt (j)`
 - Abbiamo bisogno di una misura di come progredisce l'algoritmo
 - **Misura di progressione** $\Phi = \#$ numero di entrate non vuote di $M[]$.
 - inizialmente $\Phi = 0$ e durante l'esecuzione si ha sempre $\Phi \leq n$.
 - per far crescere Φ di 1 occorrono al piu' 2 chiamate ricorsive.
 - quindi per far andare Φ da 0 a j , occorrono al piu' $2j$ chiamate ricorsive per un tempo totale di $O(j)$
- Il tempo di esecuzione di `M-Compute-Opt (n)` e' quindi $O(n)$.

N.B. $O(n)$, una volta ordinati i job in base ai valori di inizio.

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

17

Memoization nei linguaggi di programmazione

- **Automatica.** Molti linguaggi di programmazione funzionale, quali il Lisp, prevedono un meccanismo per rendere automatica la memoization

```
(defun F (n)
  (if (<= n 1)
      n
      (+ (F (- n 1)) (F (- n 2)))))
```

Lisp (efficiente)

```
static int F(int n) {
  if (n <= 1) return n;
  else return F(n-1) + F(n-2);
}
```

Java (esponenziale)

```

graph TD
    F40[F(40)] --- F39[F(39)]
    F40 --- F38[F(38)]
    F39 --- F38_1[F(38)]
    F39 --- F37_1[F(37)]
    F38_1 --- F37_1_1[F(37)]
    F38_1 --- F36_1_1[F(36)]
    F37_1_1 --- F36_1_1_1[F(36)]
    F37_1_1 --- F35_1_1_1[F(35)]
    F37_1 --- F36_1_2[F(36)]
    F37_1 --- F35_1_2[F(35)]
    F36_1_2 --- F35_1_2_1[F(35)]
    F36_1_2 --- F34_1_2_1[F(34)]
    F38 --- F37_2[F(37)]
    F38 --- F36_2[F(36)]
    F37_2 --- F36_2_1[F(36)]
    F37_2 --- F35_2_1[F(35)]
    F36_2_1 --- F35_2_1_1[F(35)]
    F36_2_1 --- F34_2_1_1[F(34)]
    F36_2 --- F35_2_2[F(35)]
    F36_2 --- F34_2_2[F(34)]
    F35_2_2 --- F34_2_2_1[F(34)]
    F35_2_2 --- F33_2_2_1[F(33)]
    
```

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

18

Interval Scheduling Pesato: Trovare una soluzione

- **Domanda.** Gli algoritmi di programmazione dinamica computano il valore ottimo. E se volessimo trovare la soluzione ottima e non solo il suo valore?
- **Risposta.** Facciamo del post-processing (computazione a posteriori).

```

Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
  if (j = 0)
    output nothing
  else if (vj + M[p(j)] > M[j-1])
    Find-Solution(p(j))
  print j
  else
    Find-Solution(j-1)
}

```

Progettazione di Algoritmi A.A. 2022-23
A. De Bonis

19

19

Interval Scheduling Pesato: Bottom-Up

Programmazione dinamica bottom-up

Per capire il comportamento dell'algoritmo di programmazione dinamica e' di aiuto formulare una versione iterativa dell'algoritmo.

```

Input: n, s1, ..., sn, f1, ..., fn, v1, ..., vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), ..., p(n)

Iterative-Compute-Opt {
  M[0] = 0
  for j = 1 to n
    M[j] = max(vj + M[p(j)], M[j-1])
}

```

Correttezza: Con l'induzione su j si puo' dimostrare che ogni entrata $M[j]$ contiene il valore $OPT(j)$

Tempo di esecuzione: n iterazioni del for, ognuna della quali richiede tempo $O(1)$ → tempo totale $O(n)$

20

20

Esercizio

- Si forniscano i valori $p(j)$ per la seguente istanza del problema dell'Interval Scheduling Pesato.

$$s_1=7 \quad f_1=10 \quad v_1=5$$

$$s_2=10 \quad f_2=14 \quad v_2=13$$

$$s_3=8 \quad f_3=11 \quad v_3=2$$

$$s_4=6 \quad f_4=9 \quad v_4=4$$

- Si fornisca poi l'array M dei valori $M[j]$ calcolati dall'algoritmo di programmazione dinamica per Interval Scheduling Pesato. Alla fine si fornisca il valore della soluzione ottima e si contrassegnino con un cerchio le entrate su cui viene invocato l'algoritmo che stampa la soluzione ottima.

Attenzione: gli indici j di $p(j)$ e $M[j]$ non corrispondono necessariamente agli indici j dei valori input s_j , f_j e v_j .

21

Esercizio

Soluzione

$$s_1=7 \quad f_1=10 \quad v_1=5$$

$$s_2=10 \quad f_2=14 \quad v_2=13$$

$$s_3=8 \quad f_3=11 \quad v_3=2$$

$$s_4=6 \quad f_4=9 \quad v_4=4$$

Riordiniamo i job in base ai tempi di fine (indico con \underline{s} , \underline{f} , \underline{v} i parametri indicizzati con la posizione del job nell'ordinamento).

$$\underline{s}_1=6 \quad \underline{f}_1=9 \quad \underline{v}_1=4 \quad (\text{ex } 4) \quad p(1)=0 \quad M[1]=\max\{M[0], v_1+M[0]\}=v_1+M[0]=4$$

$$\underline{s}_2=7 \quad \underline{f}_2=10 \quad \underline{v}_2=5 \quad (\text{ex } 1) \quad p(2)=0 \quad M[2]=\max\{M[1], v_2+M[0]\}=\max\{4, 5+0\}=5$$

$$\underline{s}_3=8 \quad \underline{f}_3=11 \quad \underline{v}_3=2 \quad (\text{ex } 3) \quad p(3)=0 \quad M[3]=\max\{M[2], v_3+M[0]\}=\max\{5, 2+0\}=5$$

$$\underline{s}_4=10 \quad \underline{f}_4=14 \quad \underline{v}_4=13 \quad (\text{ex } 2) \quad p(4)=2 \quad M[4]=\max\{M[3], v_4+M[2]\}=\max\{5, 18\}=18$$

M	0	4	5	5	18	stampa nell'ordine
	0	1	2	3	4	2 4

22