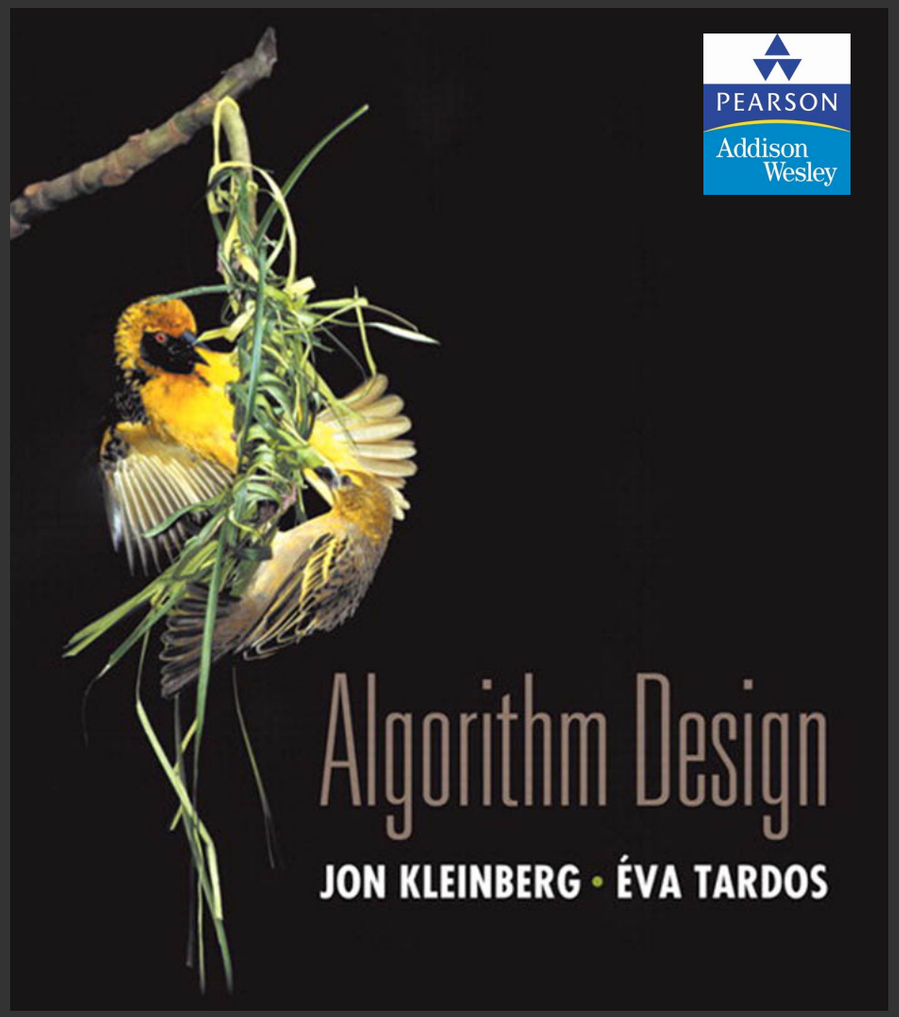


6. Programmazione Dinamica

- *schedulazione di intervalli pesati*
- *minimi quadrati a segmenti*
- *problema della bisaccia*

Traduzione e adattamento di Vincenzo Bonifaci
Original lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 09/04/2024 09:02

Paradigmi algoritmici

Avido. Processa l'input in un qualche ordine, prendendo decisioni in modo miope ed irrevocabile.

Divide et impera. Dividi un problema in sottoproblemi indipendenti; risolvi ciascun sottoproblema; combina le soluzioni dei sottoproblemi in una soluzione del problema originale.

Programmazione dinamica. Dividi un problema in sottoproblemi sovrapposti; combina le soluzioni dei sottoproblemi più piccoli in una soluzione del problema originale.

modo sofisticato di dire
che dei risultati intermedi vengono
memorizzati in una tabella
per un uso successivo

Storia della programmazione dinamica

Bellman. Pioniere dello studio sistematico della programmazione dinamica negli anni 1950.

Etimologia.

- Programmazione dinamica = pianificazione nel tempo.
- Il segretario della difesa aveva una paura patologica della matematica
- Bellman cercò un aggettivo “dinamico” per evitare conflitti.

THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Applicazioni della programmazione dinamica

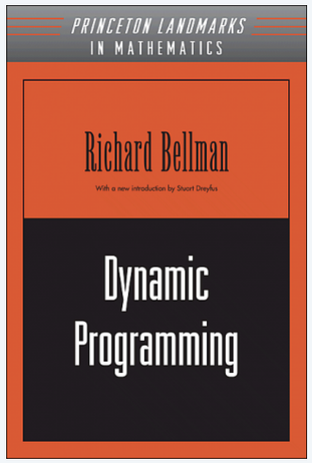
Aree applicative.

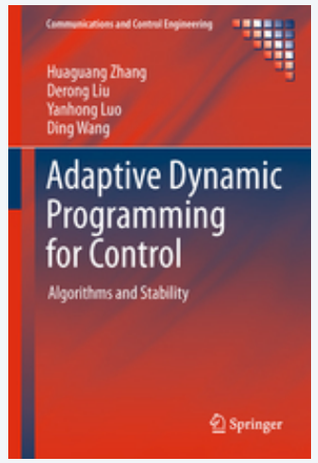
- Informatica: AI, compilatori, sistemi, grafica, teoria, ….
- Ricerca operativa.
- Teoria dell'informazione.
- Teoria del controllo.
- Bioinformatica.

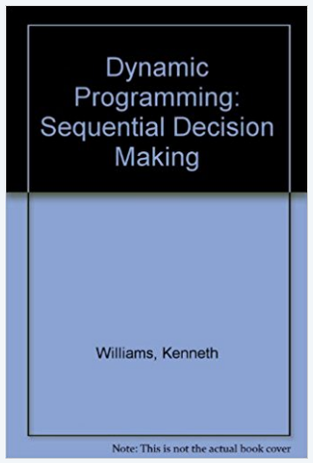
Alcuni famosi algoritmi di programmazione dinamica.

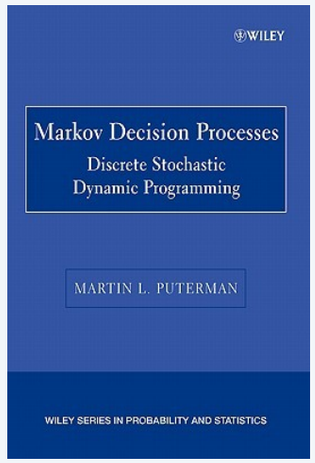
- Avidan–Shamir per il seam carving.
- diff in Unix per il confronto di due file.
- Viterbi per modelli Markoviani nascosti.
- De Boor per la valutazione di curve spline.
- Bellman–Ford–Moore per trovare cammini minimi in un grafo.
- Knuth–Plass per il "word wrapping" in $T_{\mathrm{E}}X$.
- Cocke–Kasami–Younger per il parsing di grammatiche context-free.
- Needleman–Wunsch/Smith–Waterman per l'allineamento di sequenze.

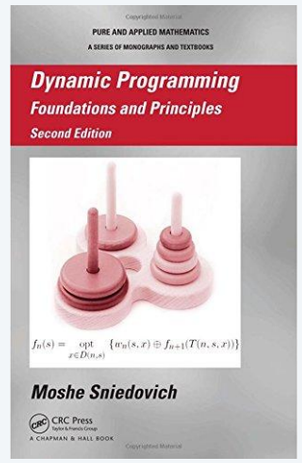
Libri sulla programmazione dinamica

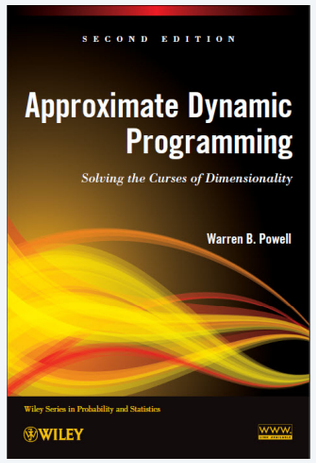


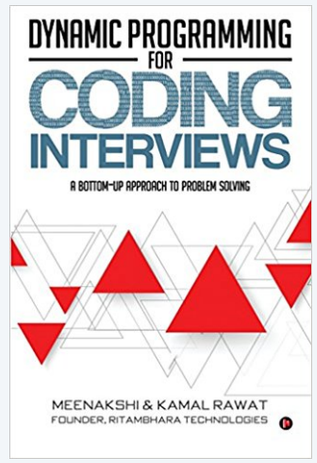


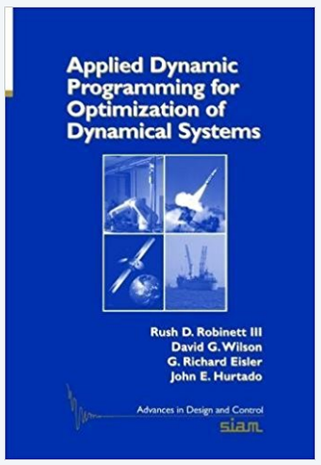


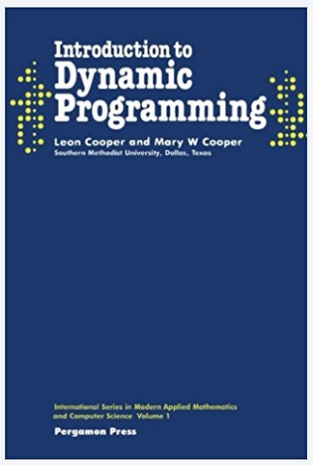


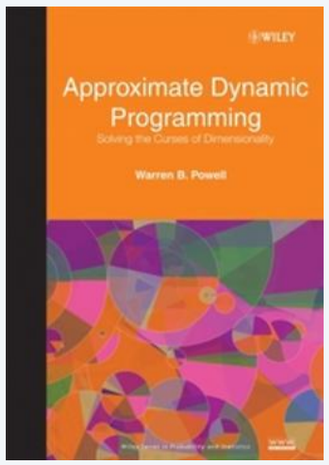


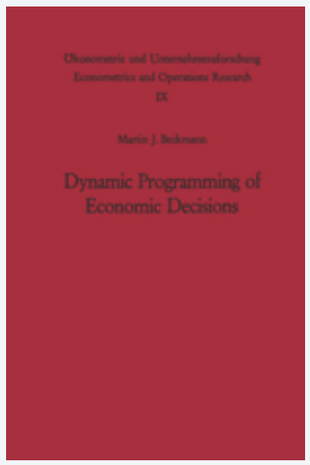


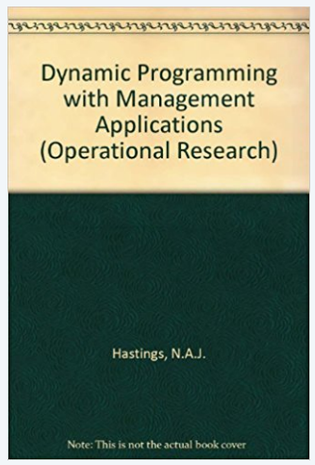


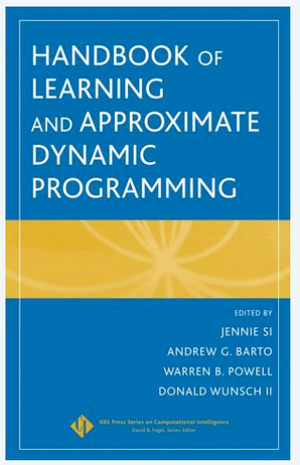


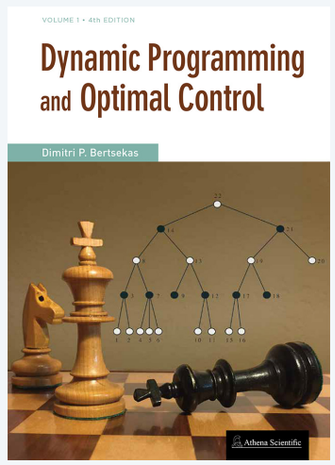


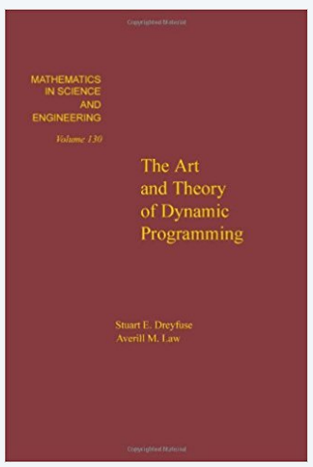


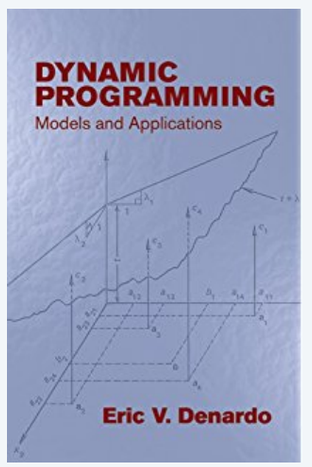


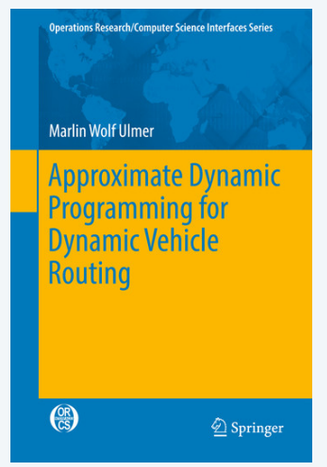


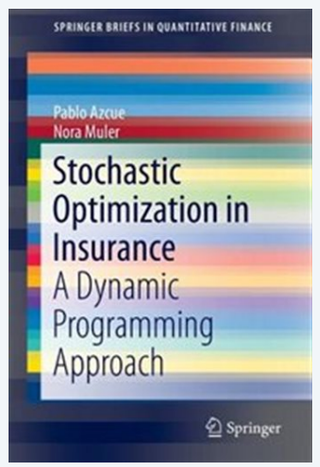


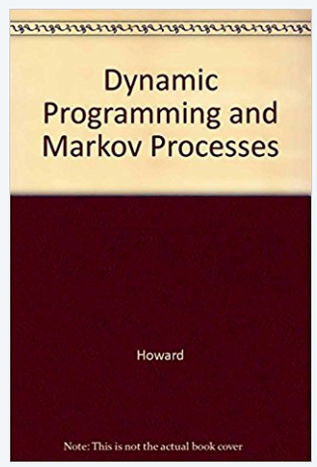


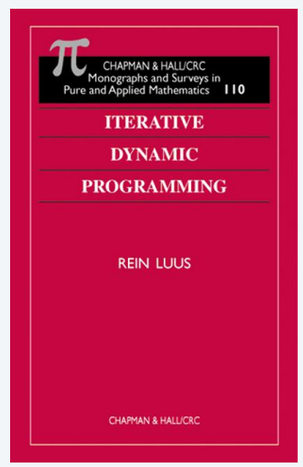


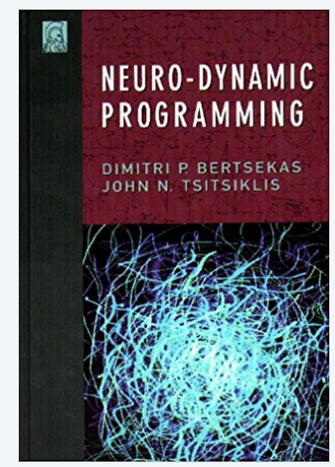


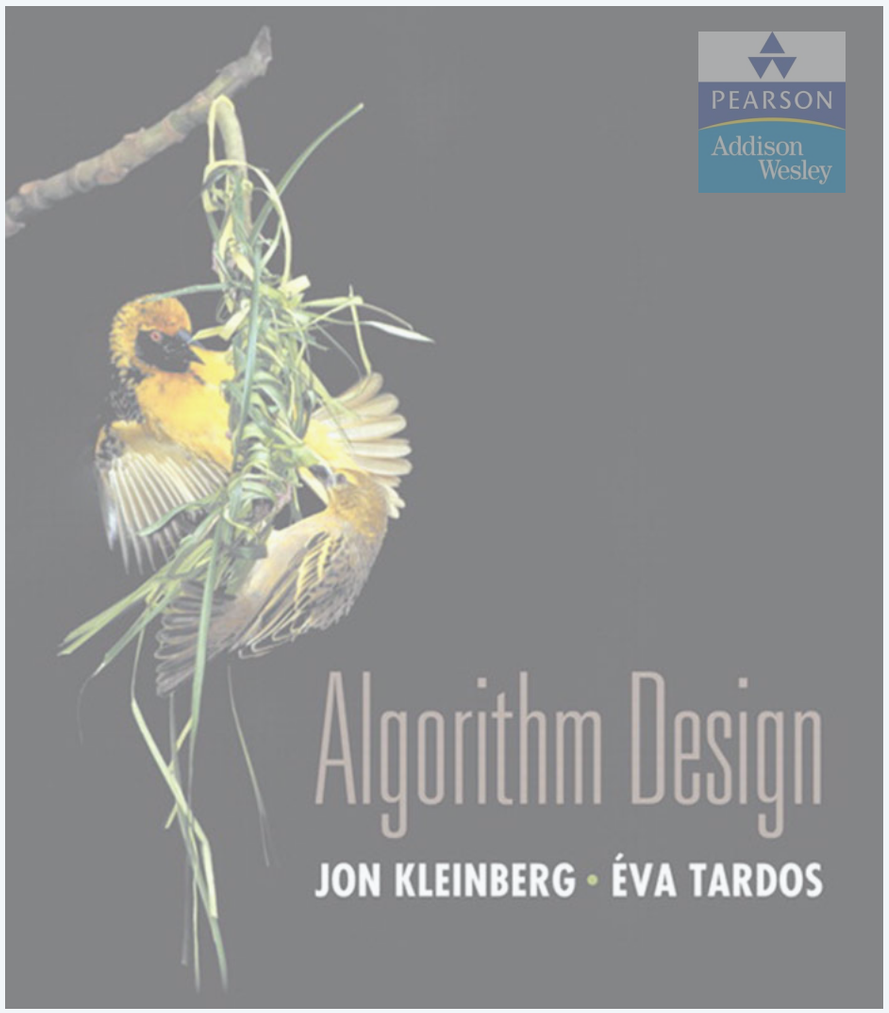












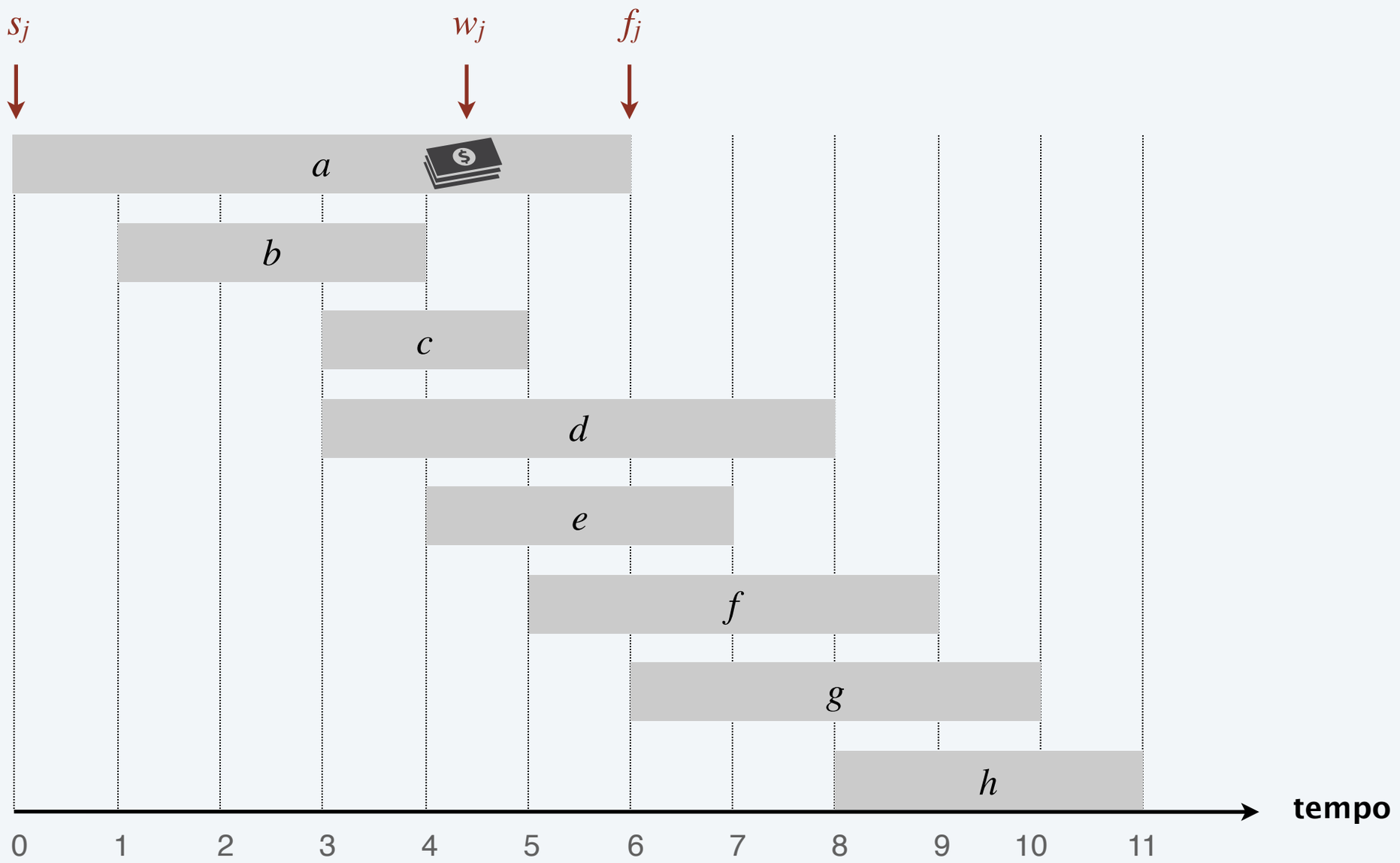
SECTIONS 6.1–6.2

6. PROGRAMMAZIONE DINAMICA

- *schedulazione di intervalli pesati*
- *segmented least squares*
- *knapsack problem*
- *RNA secondary structure*

Schedulazione di intervalli pesati

- Lavoro j inizia a s_j, finisce a f_j, ed ha peso $w_j > 0$.
- Due lavori sono compatibili se non si sovrappongono.
- Scopo: trovare un sottoinsieme a peso massimo di lavori mutuamente compatibili.



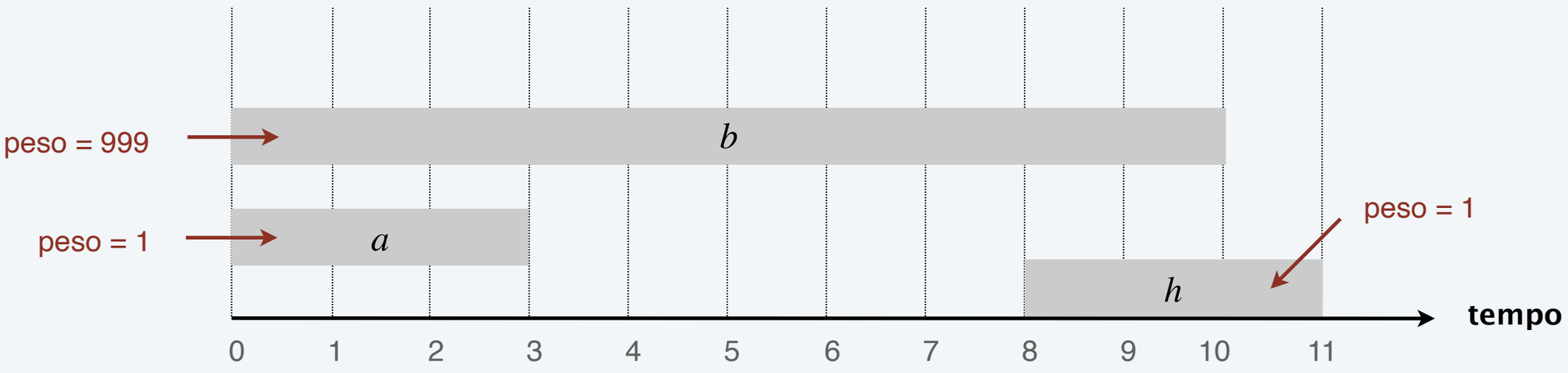
Algoritmo Earliest-finish-time-first

Earliest finish-time first.

- Considera i lavori in ordine crescente di tempo di fine.
- Accetta il lavoro se è compatibile con i lavori già selezionati.

Ricordiamo. L'algoritmo avido è corretto se tutti i pesi sono pari ad 1.

Osservazione. L'algoritmo avido fallisce gravemente per la versione pesata.



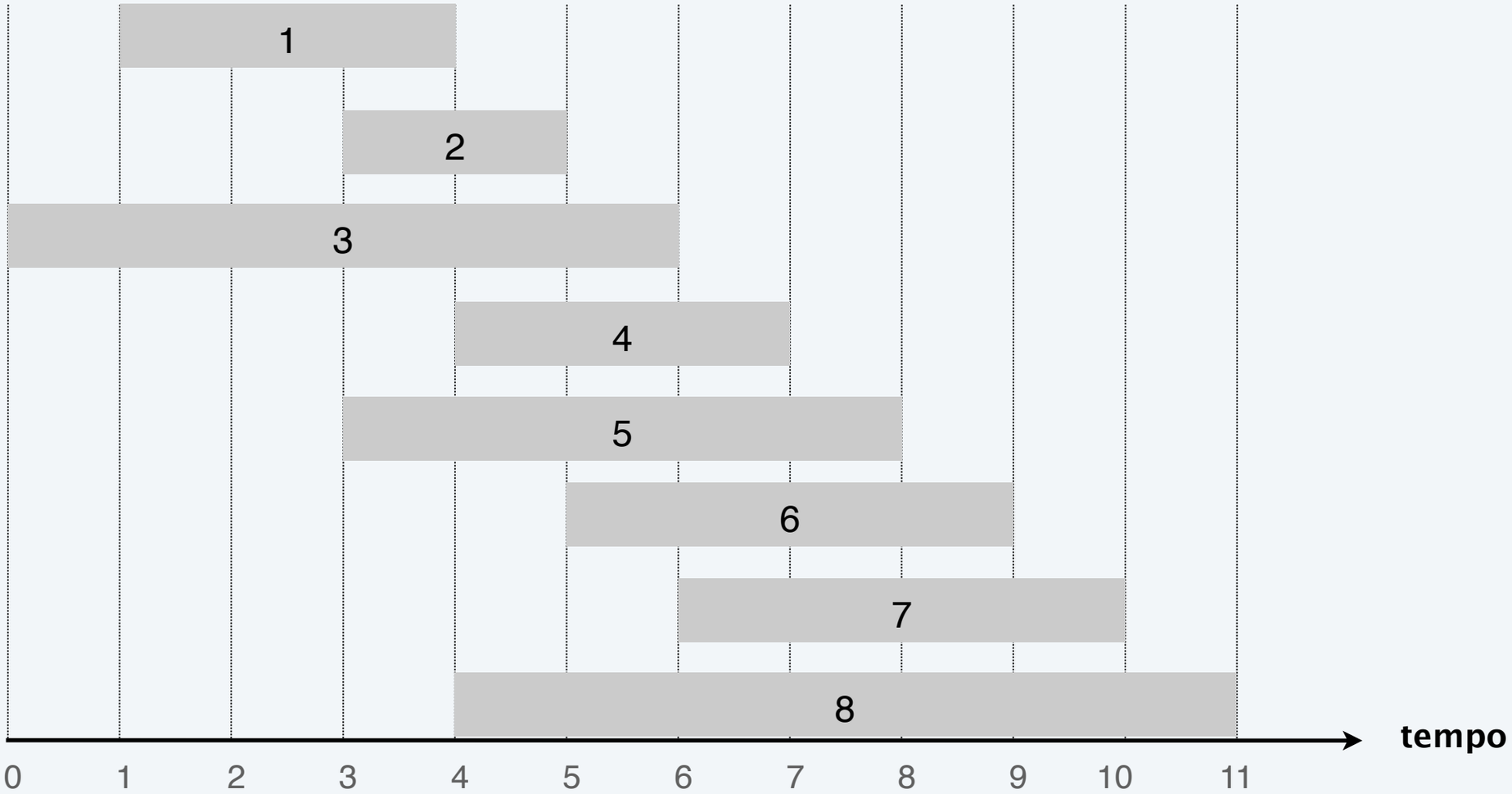
Schedulazione di intervalli pesati

Convenzione. I lavori sono in ordine crescente di tempo di fine: $f_1 \leq f_2 \leq \ldots \leq f_n$.

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

Es. $p(8) = 1, p(7) = 3, p(2) = 0.$

i è l'intervallo più a destra che finisce prima che j inizi



Programmazione dinamica: scelta binaria

Def. $OPT(j)$ = peso massimo di un sottoinsieme di lavori mutuamente compatibili per il problema consistente solo dei lavori $1, 2, ..., j$.

Scopo. $OPT(n)$ = peso massimo di un sottoinsieme di lavori compatibili.

Caso 1. $OPT(j)$ non seleziona il lavoro j.

- Deve essere una soluzione ottima al problema consistente dei lavori residui $1, 2, ..., j-1$.

Caso 2. $OPT(j)$ seleziona il lavoro j.

- Guadagna un profitto w_j.
- Non può usare i lavori incompatibili $\{ p(j)+1, p(j)+2, ..., j-1 \}$.
- Deve includere una soluzione ottima al problema consistente dei lavori compatibili residui $1, 2, ..., p(j)$.

proprietà di sottostruttura ottima (dimostrabili con argomentazioni basate su scambi)

Equazione di Bellman.

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

Schedulazione di intervalli pesati: forza bruta

BRUTE-FORCE $(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Ordina i lavori per tempo di fine in modo che $f_1 \leq f_2 \leq \ldots \leq f_n$.

Calcola $p[1], p[2], \ldots, p[n]$ tramite ricerca binaria.

RETURN COMPUTE-OPT(n).

COMPUTE-OPT(j)

IF $(j = 0)$

  RETURN 0.

ELSE

  RETURN $\max \{$COMPUTE-OPT$(j-1)$, w_j + COMPUTE-OPT$(p[j])$ $\}$.

Programmazione dinamica: quiz 1

Qual è il tempo di esecuzione di COMPUTE-OPT(n) nel caso peggiore?

A. $\Theta(n \log n)$

B. $\Theta(n^2)$

C. $\Theta(1.618^n)$

D. $\Theta(2^n)$

COMPUTE-OPT(j)

IF ($j = 0$)

  RETURN 0.

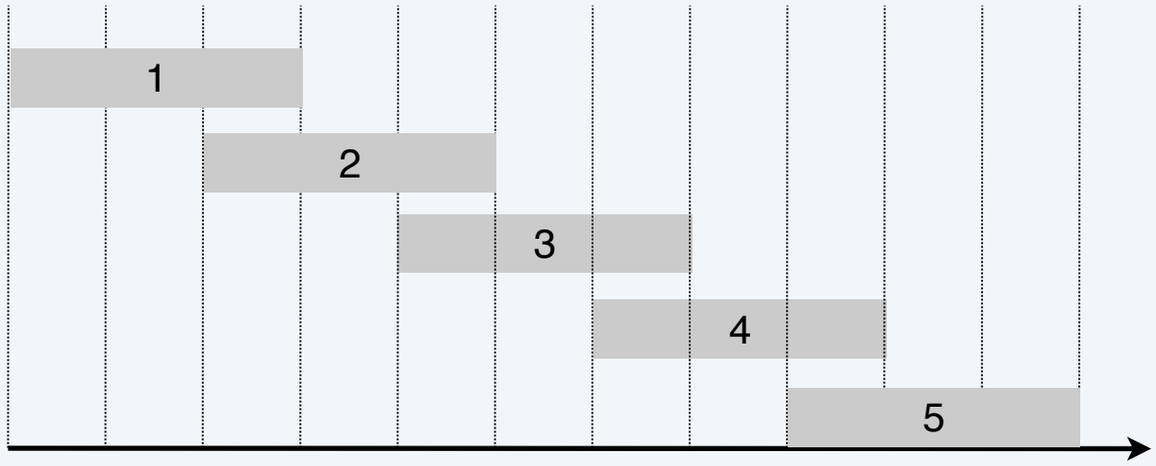
ELSE

  RETURN max {COMPUTE-OPT($j-1$), w_j + COMPUTE-OPT($p[j]$) }.

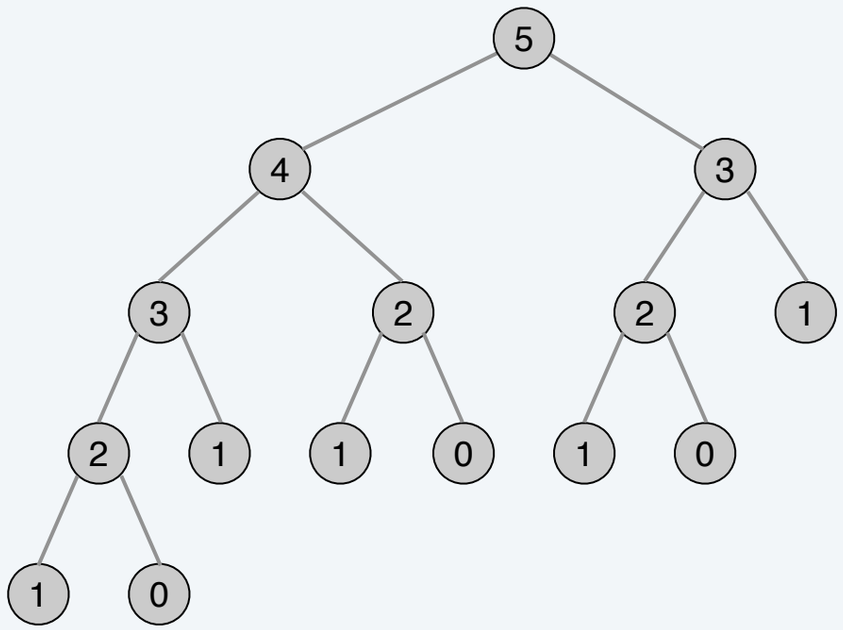
Schedulazione di intervalli pesati: forza bruta

Osservazione. L'algoritmo ricorsivo è incredibilmente lento a causa dei problemi sovrapposti $\Rightarrow$ algoritmo tempo-esponenziale.

Es. Il numero di chiamate ricorsive per una famiglia di instanze "a strati" cresce come la sequenza di Fibonacci.



p(1) = 0, p(j) = j-2



albero della ricorsione

Schedulazione di intervalli pesati: memoizzazione [memoization]

Programmazione dinamica top-down (memoization).

- Memorizziamo il valore del sottoproblema j in $M[j]$.
- Usiamo $M[j]$ per non dover risolvere il sottoproblema j più di una volta.

TOP-DOWN$(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Ordina i lavori per tempo di fine in modo che $f_1 \leq f_2 \leq \ldots \leq f_n$.

Calcola $p[1], p[2], \ldots, p[n]$ tramite ricerca binaria.

$M[0] \leftarrow 0$. ← array globale

RETURN M-COMPUTE-OPT(n).

M-COMPUTE-OPT(j)

IF ($M[j]$ non è stato inizializzato)

$M[j] \leftarrow \max \{$ M-COMPUTE-OPT $(j-1),\ w_j$ + M-COMPUTE-OPT$(p[j])$ $\}$.

RETURN $M[j]$.

Schedulazione di intervalli pesati: tempo di esecuzione

Prop. La versione memoizzata dell'algoritmo prende tempo $O(n \log n)$.

Dim.

- Ordina per tempi di fine: $O(n \log n)$ tramite mergesort.
- Calcola $p[j]$ per ogni j : $O(n \log n)$ tramite ricerca binaria.

- M-COMPUTE-OPT(j): ogni invocazione prende tempo $O(1)$ e:
  - (1) restituisce un valore già inizializzato $M[j]$, oppure
  - (2) inizializza $M[j]$ ed effettua due chiamate ricorsive

- Misura di progresso $\Phi = \#$ di elementi inizializzati in $M[1..n]$.
  - inizialmente $\Phi = 0$; si ha sempre $\Phi \leq n$.
  - (2) incrementa Φ di $1 \Rightarrow \leq 2n$ chiamate ricorsive totali.

- Il tempo complessivo di esecuzione di M-COMPUTE-OPT(n) è $O(n)$. ▪

Those who cannot remember the past are condemned to repeat it.

- Dynamic Programming

Schedulazione di intervalli pesati: costruire la soluzione

D. L'algoritmo di PD calcola il valore ottimo. Come troviamo la soluzione?

R. Effettuiamo una seconda passata chiamando FIND-SOLUTION(n).

```
FIND-SOLUTION(j)

IF (j = 0)
    RETURN ∅.
ELSE IF (w_j + M[p[j]] > M[j-1])
    RETURN { j } ∪ FIND-SOLUTION(p[j]).
ELSE
    RETURN FIND-SOLUTION(j-1).
```

$$M[j] = \max \{ M[j-1],\ w_j + M[p[j]] \}.$$

Analisi. # di chiamate ricorsive $\le n \Rightarrow O(n)$.

Schedulazione di intervalli pesati: soluzione PD bottom-up

Programmazione dinamica bottom-up. Dipaniamo la ricorsione.

BOTTOM-UP($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Ordina i lavori per tempo di fine in modo che $f_1 \leq f_2 \leq \ldots \leq f_n$.

Calcola $p[1], p[2], \ldots, p[n]$.

$M[0] \leftarrow 0$.

valori precedentemente calcolati

FOR $j = 1$ TO n

  $M[j] \leftarrow \max \{ M[j-1],\ w_j + M[p[j]] \}$.

Tempo di esecuzione. La versione bottom-up prende tempo $O(n \log n)$.

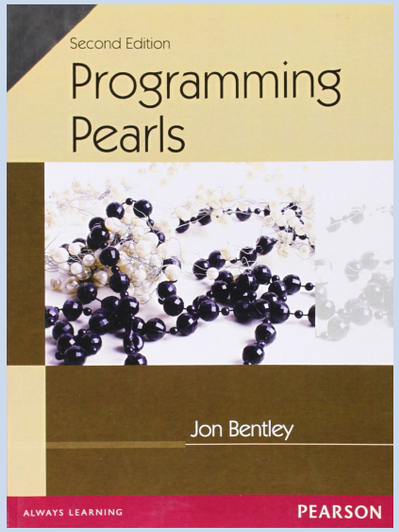
Problema del sottoarray massimo

Scopo. Dato un array x di n interi (positivi o negativi), trovare un sottoarray contiguo la cui somma sia massima.

12	5	−1	31	−61	59	26	−53	58	97	−93	−23	84	−15	6

187

Applicazioni. Visione artificiale, data mining, analisi di sequenze genomiche, ….



Algoritmo di Kadane

Def. $OPT(i)$ = somma massima di un sottoarray di x il cui estremo destro ha indice i.

Scopo. Calcolare $\max_i OPT(i)$

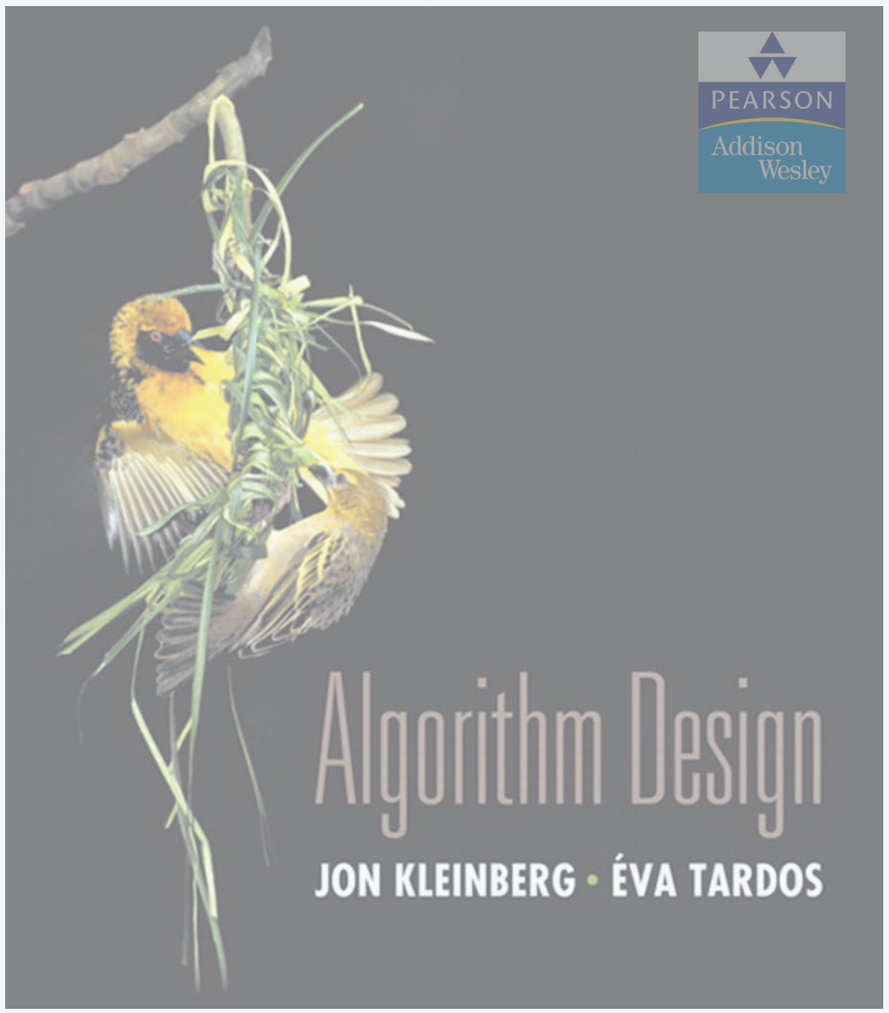
Equazione di Bellman.

$$OPT(i) = \begin{cases} x_1 & \text{if } i = 1 \\ \max\{\, x_i,\ x_i + OPT(i-1) \,\} & \text{if } i > 1 \end{cases}$$

prendi solo l'elemento i

prendi l'elemento i assieme al miglior sottoarray che termina all'indice $i-1$

Tempo di esecuzione. $O(n)$.



SECTION 6.3

6. PROGRAMMAZIONE DINAMICA

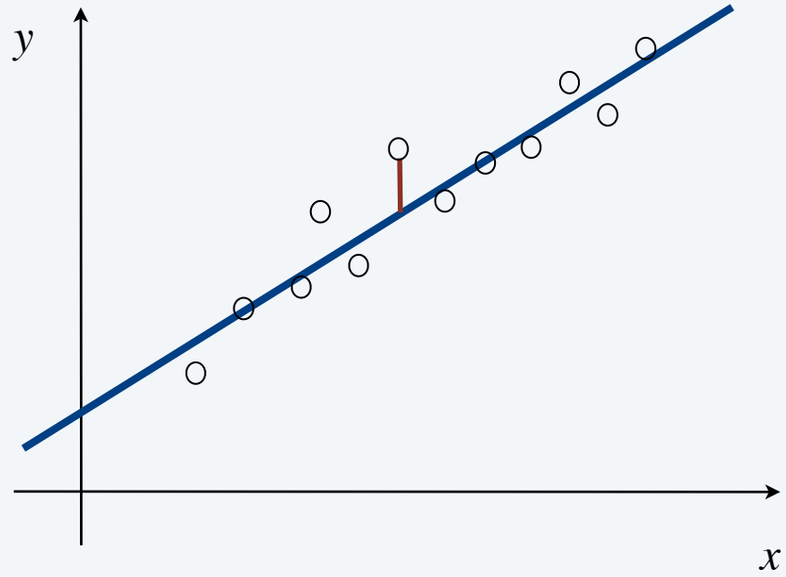
- *weighted interval scheduling*
- *minimi quadrati a segmenti*
- *knapsack problem*
- *RNA secondary structure*

Problema dei minimi quadrati

Minimi quadrati. Problema fondamentale in statistica.

- Dati n punti nel piano: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.
- Trovare una retta $y = ax + b$ che minimizza la somma dei quadrati degli errori [Sum of Squared Errors, SSE]:

$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



Soluzione. Analisi matematica $\Rightarrow$ l'errore minimo si ottiene quando

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

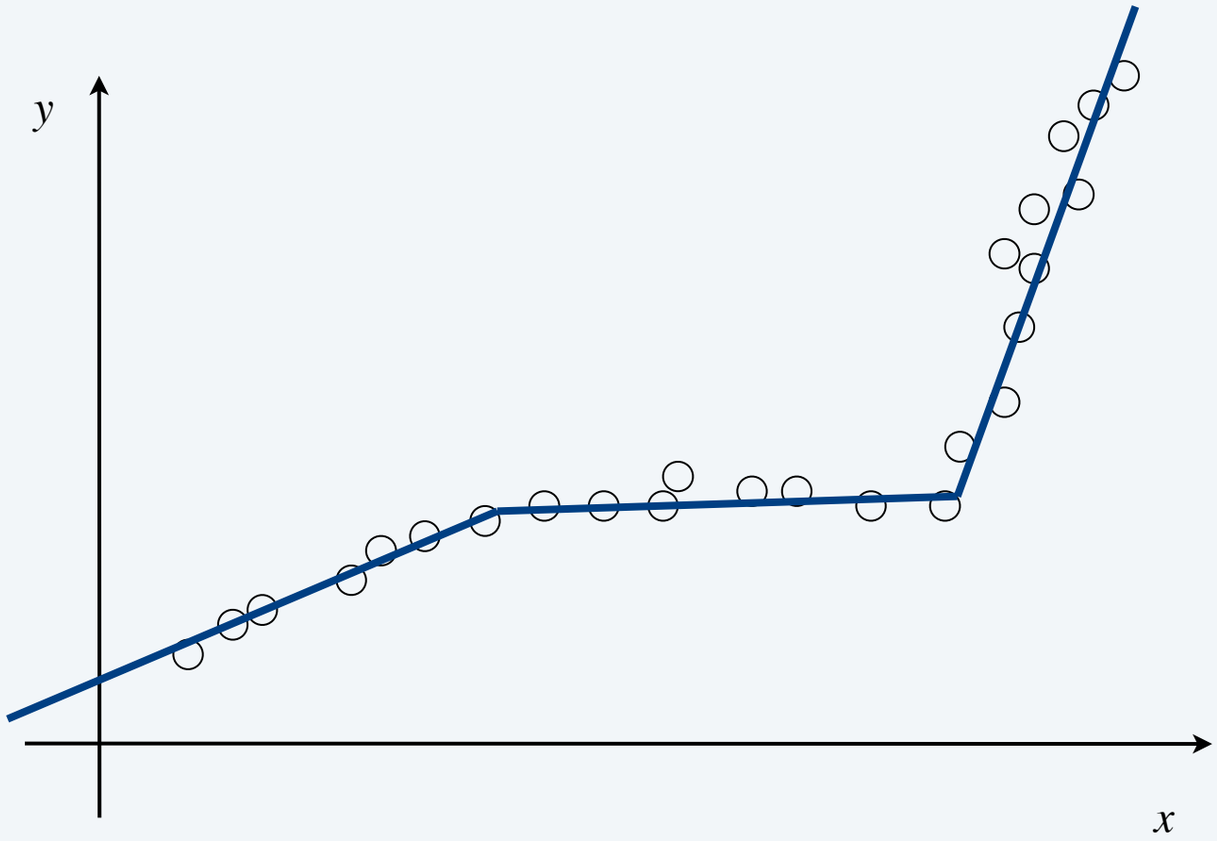
Problema dei minimi quadrati a segmenti

Minimi quadrati a segmenti.

- I punti giacciono all'incirca su una sequenza di segmenti di rette.
- Dati n punti nel piano: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ con $x_1 < x_2 < \ldots < x_n$, trovare una partizione P dei punti in segmenti che minimizzi $f(P)$.

Q. Qual è una scelta di $f(P)$ che bilancia accuratezza e parsimonia?

bontà del fit — numero di segmenti



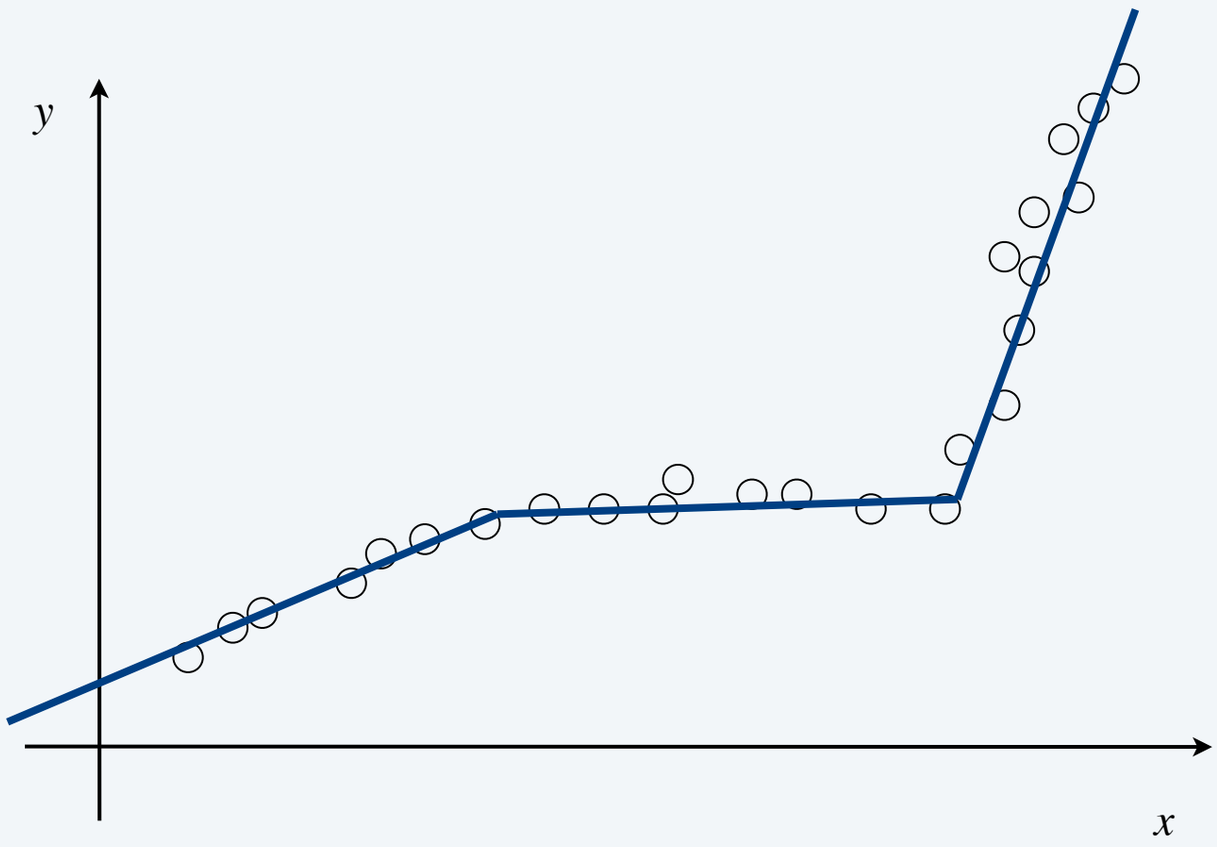
Problema dei minimi quadrati a segmenti

Minimi quadrati a segmenti.

- I punti giacciono all'incirca su una sequenza di segmenti di rette.
- Dati n punti nel piano: $(x_1, y_1), (x_2, y_2)$, …, (x_n, y_n) con $x_1 < x_2 < \ldots < x_n$, trovarne una partizione P in segmenti che minimizzi $f(P)$.

Scopo. Minimizzare $f(P) = E + c\,L$ per qualche costante $c > 0$, dove

- E = somma delle somme dei quadrati degli errori in ciascun segmento.
- L = numero di segmenti.



Programmazione dinamica: scelta multipla

Notazione.

- $OPT(j)$ = costo minimo per i punti $p_1, p_2, \ldots, p_j$.
- e_{ij} = SSE per i punti $p_i, p_{i+1}, \ldots, p_j$.

Per calcolare $OPT(j)$:

- L'ultimo segmento usa i punti $p_i, p_{i+1}, \ldots, p_j$ per qualche $i \le j$.
- Costo = $e_{ij} + c + OPT(i-1)$. ← proprietà di sottostruttura ottima (argomentazione basata su scambi)

Equazione di Bellman.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{1 \le i \le j} \{ e_{ij} + c + OPT(i-1) \} & \text{if } j > 0 \end{cases}$$

Algoritmo per i minimi quadrati a segmenti

SEGMENTED-LEAST-SQUARES$(n, p_1, \ldots, p_n, c)$

FOR j = 1 TO n

  FOR i = 1 TO j

    Calcola le SSE e_{ij} per i punti $p_i, p_{i+1}, \ldots, p_j$.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n

  $M[j] \leftarrow \min_{1 \le i \le j} \{ e_{ij} + c + M[i-1] \}$.

valori già calcolati in precedenza

RETURN $M[n]$.

Analisi dell'algoritmo per i minimi quadrati a segmenti

Teorema. [Bellman 1961] L'algoritmo di PD risolve il problema dei minimi quadrati a segmenti in tempo $O(n^3)$ e spazio $O(n^2)$.

Dim.

- Collo di bottiglia = calcolare le SSE e_{ij} per ogni i e j.

$$a_{ij} = \frac{n\sum_k x_k y_k - (\sum_k x_k)(\sum_k y_k)}{n\sum_k x_k^2 - (\sum_k x_k)^2}, \quad b_{ij} = \frac{\sum_k y_k - a_{ij}\sum_k x_k}{n}$$

- $O(n)$ per calcolare ciascuna e_{ij}. ∎

Analisi dell'algoritmo per i minimi quadrati a segmenti

Teorema. [Bellman 1961] L'algoritmo di PD risolve il problema dei minimi quadrati a segmenti in tempo $O(n^3)$ e spazio $O(n^2)$.

Dim.

- Collo di bottiglia = calcolare le SSE e_{ij} per ogni i e j.

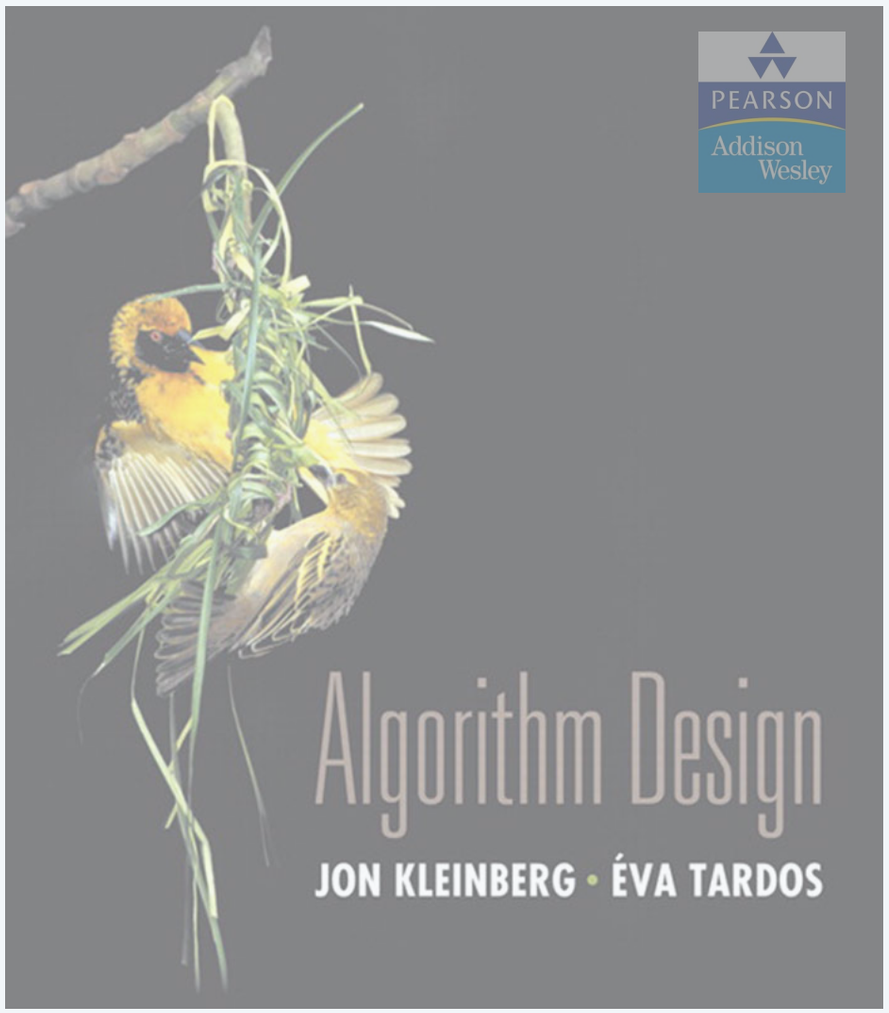
$$a_{ij} = \frac{n\sum_k x_k y_k - (\sum_k x_k)(\sum_k y_k)}{n\sum_k x_k^2 - (\sum_k x_k)^2}, \quad b_{ij} = \frac{\sum_k y_k - a_{ij}\sum_k x_k}{n}$$

- $O(n)$ per calcolare ciascuna e_{ij}. ∎

Nota. Il tempo può essere ridotto a $O(n^2)$:

- Per ogni i : precalcola le somme parziali $\sum_{k=1}^{i} x_k, \quad \sum_{k=1}^{i} y_k, \quad \sum_{k=1}^{i} x_k^2, \quad \sum_{k=1}^{i} x_k y_k$

- Usando le somme parziali, ciascuna e_{ij} è calcolabile in tempo $O(1)$.



SECTION 6.4

6. PROGRAMMAZIONE DINAMICA

- *weighted interval scheduling*
- *segmented least squares*
- *problema della bisaccia*
- *RNA secondary structure*

Problema della bisaccia [Knapsack]

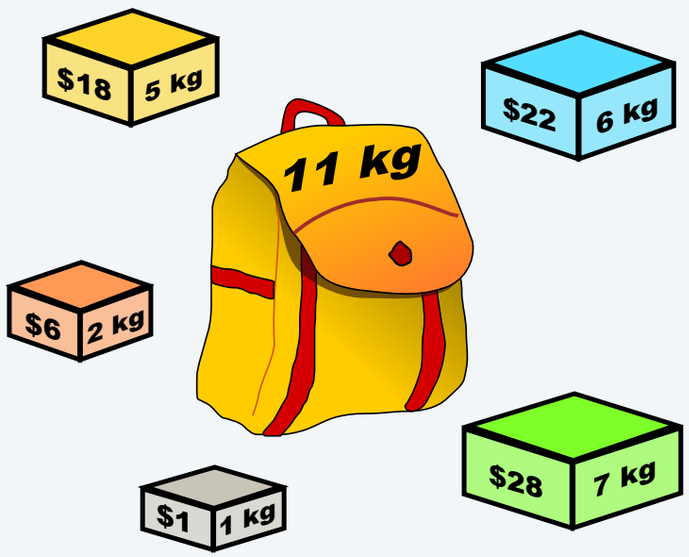
Scopo. Preparare una bisaccia massimizzando il valore degli oggetti scelti.

- Ci sono n oggetti: oggetto i fornisce valore $v_i > 0$ e pesa $w_i > 0$.
- Valore di un sottoinsieme di oggetti = somma dei valori individuali.
- La bisaccia ha un limite di peso W.

Es. Il sottoinsieme $\{ 1, 2, 5 \}$ ha valore \$35 (e peso 10).

Es. Il sottoinsieme $\{ 3, 4 \}$ ha valore \$40 (e peso 11).

Assunzione. Tutti i valori ed i pesi sono interi.



i	v_i	w_i
1	US\$1	1 kg
2	US\$6	2 kg
3	US\$18	5 kg
4	US\$22	6 kg
5	US\$28	7 kg

istanza del problema
(limite di peso W = 11)

pesi e valori sono interi positivi arbitrari

Programmazione dinamica: quiz 2

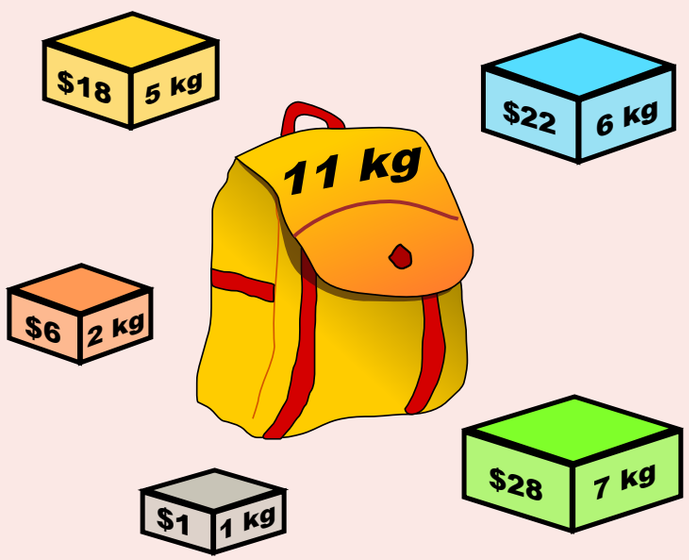
Quale di questi algoritmi risolve il problema della bisaccia?

A. Avido per valore: scegli di volta in volta l'oggetto col v_i massimo.

B. Avido per peso: scegli di volta in volta l'oggetto col w_i minimo.

C. Avido per rapporto: scegli di volta in volta l'oggetto con v_i/w_i massimo.

D. Nessuno dei precedenti.



i	v_i	w_i
1	US$1	1 kg
2	US$6	2 kg
3	US$18	5 kg
4	US$22	6 kg
5	US$28	7 kg

istanza del problema
(limite di peso W = 11)

Programmazione dinamica: quiz 3

Come definire i sottoproblemi nel problema della bisaccia?

A. $OPT(w)$ = valore ottimo di un'istanza con limite di peso w.

B. $OPT(i)$ = valore ottimo di un'istanza con oggetti $1, \ldots, i$.

C. $OPT(i, w)$ = valore ottimo di un'istanza con oggetti $1, \ldots, i$ e limite di peso w.

D. Qualunque delle precedenti.

Programmazione dinamica: falsa partenza

Def. $OPT(i)$ = valore ottimo di un'istanza con elementi $1, \ldots, i$.

Obiettivo. $OPT(n)$.

Caso 1. $OPT(i)$ non seleziona l'oggetto i.

- OPT seleziona il meglio di $\{ 1, 2, \ldots, i-1 \}$.

proprietà di sottostruttura ottima
(dimostrabile ragionando su scambi)

Caso 2. $OPT(i)$ seleziona l'oggetto i.

- Selezionare i non fornisce direttamente informazioni su se e quali altri oggetti debbano essere scartati.
- Senza sapere quali altri oggetti sono stati selezionati prima di i, non sappiamo nemmeno se abbiamo spazio per i.

Conclusione. Servono più sottoproblemi!

Programmazione dinamica: due variabili

Def. $OPT(i, w)$ = valore ottimo di un'istanza con oggetti $1, \ldots, i$, soggetta a limite di peso w.

Scopo. Calcolare $OPT(n, W)$.

Caso 1. $OPT(i, w)$ non seleziona l'oggetto i.

magari perché $w_i > w$

- $OPT(i, w)$ seleziona il meglio di $\{ 1, 2, \ldots, i-1 \}$ con limite di peso w.

Caso 2. $OPT(i, w)$ seleziona l'oggetto i.

proprietà di sottostruttura ottima (dimostrabili con argomentazioni basate su scambi)

- Guadagna il valore v_i.
- Nuovo limite di peso = $w - w_i$.
- $OPT(i, w)$ seleziona il meglio di $\{ 1, 2, \ldots, i-1 \}$ col nuovo limite di peso.

Equazione di Bellman.

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\, OPT(i-1,w),\ v_i + OPT(i-1, w-w_i) \,\} & \text{otherwise} \end{cases}$$

Problema della bisaccia: programmazione dinamica bottom-up

KNAPSACK$(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

FOR $w = 0$ TO W

  $M[0, w] \leftarrow 0$.

FOR $i = 1$ TO n

  FOR $w = 0$ TO W

    IF $(w_i > w)$ $M[i, w] \leftarrow M[i-1, w]$.

    ELSE $M[i, w] \leftarrow \max \{ M[i-1, w],\ v_i + M[i-1, w - w_i] \}$.

RETURN $M[n, W]$.

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{\, OPT(i-1, w),\ \ v_i + OPT(i-1, w - w_i)\,\} & \text{otherwise} \end{cases}$$

Problema della bisaccia: programmazione dinamica bottom-up - demo

i	v_i	w_i
1	US$1	1 kg
2	US$6	2 kg
3	US$18	5 kg
4	US$22	6 kg
5	US$28	7 kg

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i\} & \text{otherwise} \end{cases}$$

limite di peso w

sottoinsieme di oggetti 1, …, i	0	1	2	3	4	5	6	7	8	9	10	11
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT(i, w) = valore ottimo di un'istanza con oggetti 1, …, i, soggetta a limite di peso w

Problema della bisaccia: tempo di esecuzione

Teorema. L'algoritmo di PD risolve il problema della bisaccia con n oggetti e peso massimo W in tempo $\Theta(n\,W)$ e spazio $\Theta(n\,W)$.

Dim.

- Prende tempo $O(1)$ per elemento della tabella.
- Ci sono $\Theta(n\,W)$ elementi in tabella.
- Dopo il calcolo dei valori ottimi, possiamo ricostruire la soluzione a ritroso: $OPT(i, w)$ seleziona l'oggetto i sse $M\,[i, w] > M\,[i-1, w]$. ▪

i pesi sono interi tra 1 e W

Note.

- L'algoritmo sfrutta in modo critico l'assunzione che i pesi siano interi.
- L'assunzione che i valori siano interi non è stata sfruttata.

Programmazione dinamica: quiz 4

Esiste un algoritmo tempo-polinomiale per il problema della bisaccia?

A. Sì, perché l'algoritmo di PD prende tempo $\Theta(n\,W)$.

B. No, perché $\Theta(n\,W)$ non è una funzione polinomiale della taglia d'input.

C. No, perché il problema è **NP**-arduo.

D. Non si sa.

Resto in monete

Problema. Date n valori di monete $\{ c_1, c_2, \ldots, c_n \}$ ed un valore bersaglio V, trovare il minimo numero di monete che totalizzino V (o restituire "impossibile").

Ricordiamo. L'algoritmo avido del cassiere è ottimo per valori di monete USA, ma non per qualunque insieme di valori di monete.

Es. { 1, 10, 21, 34, 70, 100, 350, 1295, 1500 }.

Ottimo. 140¢ = 70 + 70.

RESTO IN MONETE

Def. $OPT(v)$ = minimo numero di monete che totalizzano v.

Scopo. Calcolare $OPT(V)$.

Scelta multipla. Per calcolare $OPT(v)$,

- Scegli un tipo di moneta c_i per qualche i.
- Scegli il minimo numero di monete che totalizzano $v - c_i$.

proprietà di sottostruttura ottima

Equazione di Bellman.

$$OPT(v) = \begin{cases} \infty & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ \min\limits_{1 \le i \le n} \{\, 1 + OPT(v - c_i) \,\} & \text{otherwise} \end{cases}$$

Tempo di esecuzione. $O(n\,V)$.

Riepilogo sulla programmazione dinamica

Schema.

di norma, il numero di sottoproblemi è polinomiale

- Definire una collezione di sottoproblemi.
- La soluzione al problema originale è calcolabile dai sottoproblemi.
- Un ordine naturale sui sottoproblemi dal “più piccolo” al “più grande” che permette di determinare la soluzione ad un problema grazie alla soluzione di sottoproblemi più piccoli.

Tecniche.

- Scelta binaria: schedulazione di intervalli pesati.
- Scelta multipla: minimi quadrati a segmenti.
- Aggiunta di una variabile: problema della bisaccia.

Programmazione dinamica top-down vs. bottom-up.