Programmazione dinamica: problema dello zaino e della somma di sotto-insiemi.

5 aprile 2022

6.4 Knapsack Problem

Ovvero: come aiutare un ladro a fare il suo mestiere….

Problema dello zaino

Problema dello zaino.

- Abbiamo n oggetti ed uno "zaino."
- Oggetto i pesa $w_i > 0$ chilogrammi ed ha valore $v_i > 0$.
- Zaino ha una capacità di W chilogrammi.
- Obiettivo: riempire zaino per massimizzare valore totale.

Esempio: { 3, 4 } ha valore 40.

W = 11

oggetti	valore	peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Algoritmo "intuitivo": aggiungere oggetto con valore massimo vi
Esempio: {5, 2, 1} ha valore =35. Quindi l'algoritmo non è ottimale

Problema dello zaino

Problema dello zaino.

- Abbiamo n oggetti ed uno "zaino."
- Oggetto i pesa $w_i > 0$ chilogrammi ed ha valore $v_i > 0$.
- Zaino ha una capacità di W chilogrammi.
- Obiettivo: riempire zaino per massimizzare valore totale.

Esempio: { 3, 4 } ha valore 40.

W = 11

oggetti	valore	peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Algoritmo greedy: aggiungere oggetto con peso minimo (compatibile con la capienza)

Esempio: {1,2,3} ha valore 25 $\Rightarrow$ algoritmo greedy non ottimale.

Problema dello zaino

Problema dello zaino.

- Abbiamo n oggetti ed uno "zaino."
- Oggetto i pesa $w_i > 0$ chilogrammi ed ha valore $v_i > 0$.
- Zaino ha una capacità di W chilogrammi.
- Obiettivo: riempire zaino per massimizzare valore totale.

Esempio: { 3, 4 } ha valore 40.

W = 11

oggetti	valore	peso	val/peso
1	1	1	1
2	6	2	3
3	18	5	3,6
4	22	6	3,66
5	28	7	4

Algoritmo greedy: aggiungere oggetto con rapporto massimo v_i / w_i.

Esempio: { 5, 2, 1 } ha valore = 35 $\Rightarrow$ algoritmo greedy non ottimale.

Che fare?

Bisogna cambiare approccio!

Programmazione dinamica

Programmazione dinamica

Istruzioni per l'uso:

- si calcola prima il valore ottimo, poi eventualmente una soluzione ottimale
- si guarda ad una soluzione ottimale OPT e si cerca di esprimere il suo valore (ottimo) in termini di valori di soluzioni ottimali a sotto-problemi (=stesso problema su istanze più piccole). Oppure, equivalentemente, si guarda al valore ottimo di soluzioni piccole, e si cerca una regola generale per ottenere dai valori già calcolati il valore ottimo per soluzioni a problemi più grandi
- si definisce OPT(alcuni indici)= il valore ottimo di un generico sotto-problema; quali indici?
- si scrive una relazione di ricorrenza per OPT(alcuni indici)
- si calcola OPT(alcuni indici) in maniera iterativa o ricorsiva con annotazione, con l'ausilio di una tabella.
- si valuta complessità di tempo e di spazio
- si ricostruisce una soluzione ottimale dai dati inseriti nella tabella, da quello che dà la soluzione finale all'indietro

OPT(i)

Nei problemi visti finora abbiamo definito OPT(i) come il valore (ottimo) cercato per il sotto-problema «fino ad i»:

- Fibonacci: F(i)= i-esimo numero della sequenza
- Scheduling di intervalli pesati: OPT(i)=valore ottimo ottenibile dagli intervalli 1, 2, … , i
- Allineamento di sequenze X e Y: OPT(i,j)=valore ottimo di un allineamento del prefisso di X fino ad i con quello di Y fino a j.

Proviamo così anche per il problema dello zaino.

Dynamic Programming: False Start

Def. OPT(i) = max profit subset of items 1, …, i.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 }

- Case 2: OPT selects item i.
  - accepting item i does not immediately imply that we will have to reject other items
  - without knowing what other items were selected before i, we don't even know if we have enough room for i

Conclusion. Need more sub-problems!

Dynamic Programming: Adding a New Variable

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 } using weight limit w
- Case 2: OPT selects item i.
  - new weight limit = $w - w_i$
  - OPT selects best of { 1, 2, …, i-1 } using this new weight limit

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

Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

```
Input: n, w_1,…,w_n, v_1,…,v_n,W

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 0 to W
      if (w_i > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], v_i + M[i-1, w-w_i ]}

return M[n, W]
```

Knapsack Algorithm

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	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: { 4, 3 }
value = 22 + 18 = 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

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

Knapsack Problem: Running Time

Running time. $\Theta(n\ W)$.

- Not polynomial in input size (=n)!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]

Knapsack approximation algorithm. There exists a polynomial algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]

Somma di sottoinsiemi

Problema:

Abbiamo un'unica macchina che può eseguire dei compiti ed abbiamo n richieste. Possiamo utilizzare la macchina dal tempo 0 al tempo T. Ogni compito i richiede tempo d_i per essere eseguito.
Il nostro obiettivo è di mantenere la risorsa il più possibile impegnata fino all'orario T.

Che problema vi ricorda?

Scheduling di intervalli?

Formalizziamolo:

Input: $S = \{1, 2, \ldots, n\}$; ad ogni $i=1,\ldots,n$, è associato d_i; T.

Output: $S' \subseteq S$ tale che $\sum_{i \in S'} d_i \leq T$ e $\sum_{i \in S'} d_i$ sia massima.

E' un caso particolare del problema dello zaino in cui $v_i = w_i = d_i$
Noto come problema della somma di sottoinsiemi.

Appello del 10 novembre 2017

Quesito 1 (23 punti) (*Zaino*)

a) Definire il problema computazionale dello zaino (0-1), specificando i dati in ingresso e quelli in uscita.

b) Definire la funzione OPT(i, x) studiata per risolvere il problema e scrivere la relativa relazione di ricorrenza. E' necessario giustificare la risposta.

c) Eseguire l'algoritmo studiato sui seguenti dati: {1, 2, 3}, $w_1 = 2$, $w_2 = 3$, $w_3 = 1$; $v_1 = 4$, $v_2 = 2$, $v_3 = 5$ e $W=5$. E' necessario mostrare e commentare i passi salienti dell'esecuzione.

d) Mostrare come ottenere un insieme di oggetti ottimale per i dati del punto c), a partire dai valori ottimi calcolati al punto c).

Esercizi: varianti al problema dello zaino

Problema dello zaino: Esercizio 1

Si descriva ed analizzi un algoritmo per la seguente variazione del problema dello zaino: Dati n oggetti di peso $w_1, w_2, \ldots, w_n$ e valore $v_1, v_2, \ldots, v_n$ ed uno zaino di capacità W (tutti gli input sono >0), trovare il massimo valore di un sottoinsieme degli oggetti il cui peso totale è al massimo W, con la condizione che ogni oggetto può essere preso anche più di una volta.

(La variazione rispetto al problema del testo, consiste nel superamento del vincolo che ogni oggetto poteva essere preso al massimo una sola volta.)

Problema dello zaino: Esercizio 2

Si descriva ed analizzi un algoritmo per la seguente variazione del problema dello zaino: Dati n oggetti di peso $w_1,w_2,\ldots,w_n$ e valore $v_1,v_2,\ldots,v_n$ ed uno zaino di capacità W (tutti gli input sono >0), trovare il massimo valore di un sottoinsieme degli oggetti il cui peso totale è al massimo W, con la condizione che ogni oggetto può essere preso al massimo 2 volte.

(La variazione rispetto al problema del testo, consiste nel superamento del vincolo che ogni oggetto poteva essere preso al massimo una sola volta.)

Problema dello zaino: Esercizio 3

Si descriva ed analizzi un algoritmo per la seguente variazione del problema dello zaino: Dati n oggetti di peso $w_1,w_2,\ldots,w_n$ e valore $v_1,v_2,\ldots,v_n$ ed uno zaino di capacità W (tutti gli input sono >0), trovare il massimo valore di un sottoinsieme degli oggetti il cui peso totale è al massimo W, con la condizione che non possono essere presi due oggetti con indici consecutivi (ovvero gli oggetti i-esimo ed (i+1)-esimo, per i=1,2,…,n-1).

Appello del 4 aprile 2018

Quesito 2 (24 punti) (*Programmazione dinamica*)

Si supponga che la soluzione ad un certo problema (a noi ignoto) sia data, per un certo intero n positivo, dal massimo fra i valori OPT(n,R) e OPT(n,B) definiti ricorsivamente come segue (R sta per Rosso e B sta per Blu):

$$OPT(1, R) = 2$$

$$OPT(1, B) = 1$$

$$OPT(i, R) = OPT(i - 1, B) + 1, \text{ se } i > 1$$

$$OPT(i, B) = \max \{OPT(i, R) - 1, OPT(i - 1, R)\}, \text{ se } i > 1$$

a) Calcolare i valori di OPT(i, R) e OPT(i, B) per ogni i=1, 2, …, 5, organizzandoli in una tabella.

b) Scrivere lo pseudocodice di un algoritmo **ricorsivo** per il calcolo della soluzione al problema.

c) Scrivere lo pseudocodice di un algoritmo di **programmazione dinamica** per il calcolo della soluzione al problema. Analizzarne la complessità di tempo e di spazio, giustificando la risposta.

Appello del 21 marzo 2019

Quesito 1 (26 punti) (*Giornata di seminari*)

Vi state occupando di organizzare una giornata di seminari nell'Aula Magna della vostra università. Avete avuto la disponibilità di vari relatori a tenere un loro intervento; ognuno ha specificato da che ora a che ora si terrebbe il suo seminario. Purtroppo, non riuscite ad organizzare la giornata in modo da inserire tutti i relatori. Dovete perciò scegliere alcuni fra i relatori in modo che i loro seminari possano essere svolti nell'aula senza sovrapposizioni di orario. Volete inoltre fare in modo che sia massimo il **tempo** totale di utilizzo effettivo dell'aula.

a) Definire il problema computazionale, specificandone i dati in ingresso e in uscita.

b) Indicare se si tratta di un problema studiato e, se sì, quale.

c) Si consideri un algoritmo *greedy* basato sul criterio di scelta del seminario che utilizza l'aula per più tempo. Mostrare, con un contro-esempio, che tale algoritmo non sempre porta ad una soluzione ottimale.

d) Progettare un algoritmo di **programmazione dinamica** che risolve il problema e valutarne la complessità. Si potrà ottenere il massimo della votazione solo se l'algoritmo è descritto tramite pseudo-codice ed è discussa la sua correttezza.

Prima prova intercorso 2017/18

1) 1 ☐

Qual è il tempo di esecuzione del seguente frammento di pseudocodice?

```
x=0
for i=1 to n-1
     for j=1 to logn
          x=i+j
return x
```

A. Θ (n)
B. Θ (n log n)
C. Θ (n^2)
D. Nessuna delle risposte precedenti

2) 2 ☐

Qual è la corretta successione delle funzioni seguenti affinché compaiano da sinistra a destra in ordine crescente di crescita asintotica: n^2, $n(\log n)^3$, $n\sqrt{n}$?

A. $n\ (\log n)^3$, $n\sqrt{n}$, n^2
B. n^2, $n\ (\log n)^3$, $n\sqrt{n}$
C. $n\ (\log n)^3$, n^2 , $n\sqrt{n}$
D. Nessuna delle risposte precedenti.

Prima prova intercorso 2017/18

3) 3 □

Quale relazione di ricorrenza soddisfa il tempo di esecuzione T(n) del seguente algoritmo ricorsivo, supponendo che l'esecuzione di `qualcosa` richieda tempo costante?

```
PROVA(n)
if n=1 then do qualcosa
       else
           return PROVA(n/2)+n-1
```

A. $T(n) = T(n/2) + \Theta(n)$ con $T(1) = \Theta(1)$

B. $T(n) = T(n-1) + \Theta(n)$ con $T(1) = \Theta(1)$

C. $T(n) = T(n-1) + \Theta(1)$ con $T(1) = \Theta(1)$

D. Nessuna delle risposte precedenti

Prima prova intercorso 2017/18

4) 4 ☐

Una chiamata alla procedura `PARTITION` (come studiata) su A[1..5] = [3, 9, 8, 1, 2] restituisce

A. 2 B. 3 C. 5 D. Nessuna delle risposte precedenti

5) 5 ☐

Quali delle seguenti affermazioni è vera?

Il tempo di esecuzione dell'algoritmo `MERGE` per la fusione di due array ordinati di n elementi è

A. O (n)
B. Θ(n log n)
C. O(log n)
D. Nessuna delle risposte precedenti

Prima prova intercorso 2017/18

Quesito 1 (16 punti)

Si risolva la seguente relazione di ricorrenza. Si potrà avere il massimo del punteggio se si mostrano **due diversi** modi di risolverla.

$$T(n) = 2\,T(n/2) + \Theta(1) \text{ con } T(1)=\Theta(1)$$

Prima prova intercorso 2017/18

Quesito 2 (18 punti)

a) Descrivere un algoritmo efficiente basato sul paradigma **divide et impera** che dato un vettore **ordinato** A[1..n] di interi strettamente positivi (cioè per ogni $1 \leq i \leq n$, $A[i] \geq 1$), restituisca il numero di occorrenze di 1 nel vettore A. Commentare il funzionamento dell'algoritmo.

b) Sia T(n) il tempo di esecuzione dell'algoritmo proposto al punto precedente. Scrivere la relazione di ricorrenza soddisfatta da T(n). Non è necessario mostrarne la soluzione.

Nota: Può essere utile sapere che esiste un algoritmo che risolve il problema in tempo O(log n).

Prima prova intercorso 2017/18

Quesito 3 (16 punti)

Si consideri il problema dello *scheduling* di intervalli pesato.

Ricostruire uno *scheduling* ottimale per il problema dato su un insieme di intervalli $S=\{1, 2, \dots, 6\}$ ordinato secondo il tempo di fine crescente degli intervalli (cioè $f_1 \leq f_2 \leq \dots \leq f_6$), sapendo che i pesi degli intervalli sono rispettivamente $w_1=12$, $w_2=2$, $w_3=8$, $w_4=9$, $w_5=3$, $w_6=10$, che i valori della funzione p sono $p(1)=0$, $p(2)=0$, $p(3)=2$, $p(4)=0$, $p(5)=1$, $p(6)=4$ e l'array M calcolato dall'algoritmo di programmazione dinamica studiato è $M[0 .. 6] = [0, 12, 12, 20, 20, 20, 30]$. E' necessario giustificare la risposta.

Si noti che non occorre conoscere i valori dei tempi di inizio e di fine degli intervalli per ricostruire la soluzione.

Prima prova intercorso 2017/18

Prima prova intercorso 2017/18

Quesito 4 (18 punti)

La soluzione ad un problema (a noi ignoto) è data da OPT(m,n) dove OPT(i,j) per i=0,1, …, m e j=0,1, …, n soddisfa la seguente relazione di ricorrenza dove $v_1, v_2, \ldots, v_m$, sono dei valori dati.

OPT(i, 0) = 0 per ogni i = 0,1, …, m
OPT(0, j) = j per ogni j=1, …, n
OPT(i,j) = max { v_i + OPT(i−1, j), OPT(i, j−1) } altrimenti.

a) Scrivere lo pseudocodice di un algoritmo di **programmazione dinamica** per il calcolo di OPT(m,n) ed analizzarne la complessità di tempo e di spazio, giustificando la risposta.
b) Indicare quali sono i punti salienti dell'algoritmo descritto al punto precedente che lo rendono un algoritmo di programmazione dinamica.