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

19 aprile 2023

19 + 7 = ?

Calendario

19 aprile: Il problema dello zaino (PD)

20 aprile: Struttura secondaria dell'RNA (PD)

21 aprile: Esercitazione

26 aprile ore 11-13: Esercitazione

ore 16, aule P3+P4: Prima prova intercorso

Prima prova inter-corso

- Sono previste 2 prove intercorso per gli studenti che stanno seguendo il corso. Si può sostenere la **seconda** prova intercorso solo se si è **superata** la **prima**. In alcuni casi sarà possibile (o obbligatoria) una prova orale.
- La **prima prova** intercorso è fissata per mercoledì **26 aprile** ore 16 in aula P3, salvo avviso contrario. Per accedere alla prova occorrerà prenotarsi su **Esse3** entro il 21 aprile. Entro tale data è anche possibile cancellare la propria prenotazione, in caso si sia cambiata idea.
- Si noti che, anche se qualche studente **fuori corso** si sarà registrato su Esse3, la sua prenotazione verrà **ignorata**. Gli studenti del **terzo anno** possono prenotarsi su Esse3, ma **con riserva**. Una volte chiuse le iscrizioni, a secondo del numero di prenotati, verrà deciso se tali studenti saranno ammessi oppure no.
- La seconda prova si terrà venerdì **8 giugno** alle ore 15, in concomitanza col pre-appello.

Programma Prima Prova

1. Introduzione al concetto di algoritmo e all'analisi asintotica. ([KT], parr. 2.1, 2.2, 2.4).
2. La tecnica Divide et Impera e relativi esempi di applicazione: Mergesort, Quicksort, Moltiplicazione veloce di matrici. Derivazione relazioni di ricorrenza e loro soluzione. ([KT], parr. 5.1, 5.2, 5.5, [DPV] par. 2.5 e altri testi consigliati).
3. La tecnica della Programmazione Dinamica e relativi esempi di applicazione: Calcolo di numeri di Fibonacci, Problemi di ottimizzazione: Scheduling di risorse, Somma di sottoinsiemi, Zaino intero, Problemi su stringhe: allineamento di sequenze e struttura secondaria dell'RNA. ([KT], parr. 6.1, 6.2, 6.4, 6.5, 6.6).

Prima prova inter-corso

Il compito sarà composto da domande a risposta multipla e domande aperte, come specificato durante le prossime esercitazioni.

Per ogni informazione, consultare la sezione Prove intercorso sulla pagina del corso su https://elearning.informatica.unisa.it/.

6.4 Knapsack Problem

Ovvero: come aiutare un ladro a derubare una villa…

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

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

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, …, n}; ad ogni i=1,…,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).