

Analisi e Progetto di Algoritmi

Elia Ronchetti

@ulerich

2023/2024

Contents

Chapter 1

Programmazione Dinamica - DP

La programmazione dinamica (DP - Dynamic Programming) è una tecnica che (come il Divide et Impera), risolve i problemi combinando le soluzioni dei sottoproblemi.

Divide et Impera è ottimo quando i sottoproblemi da risolvere sono indipendenti, mentre DP è efficace quando i sottoproblemi non sono indipendenti e quindi hanno in comune dei sottosottoproblemi e le tecniche di risoluzione top-down risultano quindi inefficienti (chiamate ripetute). La programmazione dinamica si applica tipicamente ai **problemi di ottimizzazione**.

1.1 Problemi di ottimizzazione

Sono problemi dove ci sono molte soluzioni possibile. Ogni soluzione ha un valore e si vuole trovare una soluzione con il valore ottimo. Ci possono essere più soluzioni che raggiungono il valore ottimo.

1.2 Il processo di sviluppo

Il processo di sviluppo è diviso in 4 fasi:

- Caratterizzare la struttura di una soluzione ottima
- Definire in modo ricorsivo il valore di una soluzione ottima
- Calcolare il valore di una soluzione ottima, di solito con uno schema bottom-up (dal basso verso l'alto, risulta spesso più efficiente rispetto a top-down)

- Costruire una soluzione ottima dalle informazioni calcolate

1.3 Esempio - Fibonacci

Classico esempio è l'esecuzione di Fibonacci. Utilizzando la ricorsione pura si effettuano più volte le stesse chiamate (perchè i sotto-numeri sono gli stessi). Se invece utilizziamo la DP, con un approccio Bottom-Up ci dobbiamo chiedere, ma chi è Fibonacci di n ? è Fibonacci di $(1) + \text{Fibonacci}(2) + \dots + \text{Fibonacci}(n)$. In pratica inizio a calcolare le soluzioni dal sottoproblema più piccolo a salire, così facendo possiamo risparmiare molto tempo, al costo però di un maggiore utilizzo di spazio, dato che ho un Array che deve memorizzare i valori. Si tratta di un compromesso accettabile dato che senza usare Array il tempo di esecuzione sarebbe esponenziale.

Passaggi

A livello pratico dobbiamo:

1. Scomporre il problem in sottoproblemi di dimensione inferiore
2. Formulare la soluzione in maniera ricorsiva - Equazioni di Ricorrenza
3. Usare una strategia bottom-up (non top-down)
4. Memorizzare i risultati in una opportuna struttura dati
5. Individuare il "luogo" che contiene la soluzione del problema (nel caso di Fibonacci l'ultima cella a destra)

Knapsack Problem 0/1

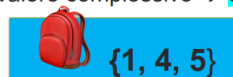
Oggetti Ad ogni oggetto viene associato un peso e un valore, quindi il problema consiste nel inserire nello zaino il massimo valore possibile senza superare il peso massimo.

$$C = 10$$

1. $v_1=1, w_1=7$
2. $v_2=3, w_2=4$
3. $v_3=1, w_3=5$
4. $v_4=1, w_4=1$
5. $v_5=1, w_5=1$



Peso complessivo → 10
Valore complessivo → 5



Peso complessivo $\rightarrow 9$
Valore complessivo $\rightarrow 3$