Analisi e progetto di algoritmi

DaveRhapsody

1 Ottobre 2020

Indice

1 Introduzione **3**

2 Programmazione dinamica **4**

2.1 Definizione 4

2.2 Tecniche 4

2.3 Il problema funzioni ricorsive 4

2.3.1 Fibonacci iterativo 5

Capitolo 1

Introduzione

Riprendendo tutto quanto da Algoritmi 1, quella del primo anno, teniamoci a mente le basi tipo calcolo computazionale, i tempi, Θ, Ω, O, nLog(n) etc., ed in particolare tutto ciò che riguarda la ricorsione, quindi cos'è una funzione ricorsiva, come funziona, e tutto ciò che ci sta attorno.

Capitolo 2

Programmazione dinamica

2.1 Definizione

La programmazione dinamica è una tecnica di programmazione che si pone come obbiettivo quello di ottimizzare i tempi di esecuzione di un algoritmo ma utilizzando dello spazio in memoria per tenere conto delle informazioni parziali.

Per informazione parziale si intende: un risultato di un calcolo parziale che servirà solo alla fine per il risultato.

2.2 Tecniche

Esistono varie tecniche per ottenere una programmazione dinamica, la principale vista è la tecnica "golosa", ovvero si osserva no le possibili strade, e senza porre troppe seghe mentali si sceglie quella che sembra la migliore.

Si prenda per esempio **l'albero di copertura**, vale a dire un albero che connette tutti i nodi del grafo, posto come g(N,A) non orientato e connesso! Bisogna trovare la soluzione avente costo minimo, più o meno come in ricerca operativa, stesso concetto.

Le funzioni ricorsive: saranno ciccine carine e pucciose perchè con 3 righe di codice ci si può sintetizzare pure il mondo intero, però dal punto di vista della computazione sono un macigno, pesantissime, sono una perdita di tempo enorme per nulla. La domanda è: come mai?

2.3 Il problema funzioni ricorsive

Per capire meglio si consideri la funzione del fibonacci. Molto semplice: fib(n):

$$\begin{cases} n \; se \; n \leq 1 \\ fib(n-1) + fib(n-2) \; se \; n > 1 \end{cases}$$

Ora, ipotizziamo di avere il fibonacci di 4 da calcolare.
fib 4

fib 3 + fib 2
fib 2 + fib 1 + fib 1 + fib 0
fib 1 + fib 0 + fib 1 + fib 1 + fib 0
Com'è possibile notare, più volte si effettua il calcolo di fib 2, ovvero, più volte si ricalcola una cosa che in realtà si ha già!

A questo punto potremmo memorizzare il risultato parziale, no? Magari in un array che funzionerà tipo:

Arriva fib(2), guardo nell'array se c'è già, oh, c'è! Bene! Prenderò da qua!

Ora, ci sarebbe un problema, hai da scandirti un array bello grande se il valore è 1500 per esempio. C'è dello spazio che si spreca per questo.. Inoltre altra cosa, la scansione, la comparazione di ogni elemento dell'array ha un costo, quindi non è detto che si accorcino così tanto i tempi.

Manteniamoci sul semplice concetto del **riutilizzare il risultato di ciò che hai già calcolato, senza ricalcolarlo di nuovo**. proviamo a ipotizzare il calcolo dei tempi del fibonacci:

$$T(n) = \begin{cases} c \; se \; n = 0, 1, 2 \\ T(n-1) + T(n-2) + c \; se \; n > 2 \end{cases}$$

Da qui si ipotizzi un tempo standard per $T(n) = r^n$, verrà fuori un'equazione del tipo: $r^n - r^{n-1} - r^{n-2} = c$, in cui c la consideriamo momentaneamente a 0, riscritta meglio diventerà: $r^{n-1}(r^2 - r - 1) = 0$ quindi

$$T(n) = k_1 \cdot (\frac{1+\sqrt{5}}{2})^{n-1} + k_2 \cdot (\frac{1+\sqrt{5}}{2})^{n-1} + c \rightarrow \Theta(a)^n \; con \; a > 1$$

Bene, ricorsivo fa schifo, e quindi non si può scrivere iterativo? Sì

2.3.1 Fibonacci iterativo

fib(int n, A[n]):
A[1] = 1, A[2] = 2
for i = 3 to n:
A[i] = A[i-1] + A[i-2]
return A[n]

Il for viene eseguito **n** volte, esattamente come l'assegnamento, quindi fondamentalmente t(N) = $\Theta(n)$, ma allo stesso tempo viene occupato dello spazio s(n) = $\Theta(n)$

Teorema: Data la matrice $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ si consideri $A^{n-1} = \begin{pmatrix} f(n) & f(n-1) \\ f(n-1) & f(n-2) \end{pmatrix}$ Ecco, il valore che assumerà A[1][1] sarà proprio il fibonacci di n, rendendolo un codice:

$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
M = potenza (A, n-1)
return M[0][0]

Praticamente si va a fare la potenza applicata alla matrice e si fa la return del valore nella locazione 0 0. Ora, anche la potenza richiede della potenza di calcolo, se ricorsiva, quindi ridefiniamo ricorsivamente la potenza in modo più ottimizzato:

$$potenza(A,k) = \begin{cases} A \ se \ k = 1 \\ potenza(A, \frac{k}{2}) * potenza(A, \frac{k}{2}) \ se \ k \ pari \\ A * potenza(A, \frac{k}{2}) * potenza(A, \frac{k}{2}) \ se \ k \ dispari \end{cases}$$

Dal punto di vista del codice diventerebbe:

$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
B = potenza (A, $\frac{n}{2}$)
return B*B

In pratica non solo si dimezzano i tempi di carico, ma uno dei due rami dell'albero ricorsivo non viene nemmeno calcolato! C'è già!

Capitolo 3

Tipi di problema

3.1 Problema decisionale

Un problema decisionale è un problema la cui risposta è un sì od un no. In altri termini quando chiedi a una di mettersi con te e ti fa "boh non lo sssso, ceh perchè esco da un'altra storia ggnegnegne", ecco, quello non è un decisionale.

Qualsiasi funzione booleana è una soluzione di un problema decisionale. Come lo era per prolog, per intenderci.

3.2 Problema di ricerca

Prima di tutto ti dice se un determinato valore c'è, è presente, e successivamente ti dice anche dove si trova. Non basta dire se c'è o no la soluzione, qua si dice anche dove si trova, considerando che dev'essere una sola. Il tipo di risposta sarà: **(x, S)** $\in \pi$

3.3 Problema di ottimo

Il problema di ottimo si rifà ai concetti di **ricerca operativa** ossia, si vuole trovare, tra tante soluzioni accettabili, quella migliore, basandosi su una serie di vincoli che specifichiamo.

Si dice problema di **ottimo** poichè la soluzione è ottima, la migliore tra quelle accettabili.

Come si risolvono questi problemi? L'ideale sarebbe trovare un algoritmo esatto, ma ci sono altri modi, tipo trovare tutte le possibili soluzioni, scartare quelle che non rispettano i vincoli, e fare un confronto delle idonee.. Insomma, vien fuori magari per un problema corto 3 ore di tempo per trovare la soluzione. Sicuro è efficace, ma non efficiente.