Esempi di Algoritmi Numerici

Salvatore Filippone
salvatore.filippone@uniroma2.it

Polinomi

Si consideri un polinomio di grado $n-1$ rappresentato attraverso i suoi coefficienti a_j, ossia

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

Con questa rappresentazione si può calcolare la valutazione del polinomio in un punto dato x utilizzando le potenze di x

Algorithm 1: powerPoly(float[] A, float[] x, int n) : float[]

float $y \leftarrow 0$;
for $j \leftarrow 0$ *to* $n-1$ **do**
  $y \leftarrow y + A[j] * x^j$;
return

Quale è il costo di questo algoritmo?

Una alternativa un po' più efficiente per il calcolo delle potenze di x è la seguente

Algorithm 2: powerPoly1(float[] A, float[] x, int n) : float[]

float $y \leftarrow 0$;
float $px \leftarrow 1$;
for $j \leftarrow 0$ *to* $n - 1$ **do**
    $y \leftarrow y + A[j] * px$;
    $px \leftarrow px * x$;
return

La soluzione più efficiente è data dallo schema di Horner

Algorithm 3: Horner(float[] A, float[] x, int n) : float[]

float $y \leftarrow A[n-1]$;
for $j \leftarrow n-2$ *to* 0 **do**
  y $\leftarrow y \cdot x + A[j]$;
return y

Consideriamo ora il problema di calcolare i coefficienti del prodotto di due polinomi $W = P \cdot Q$. Il coefficiente del prodotto di due qualunque monomi $p_k x^k \cdot q_j x^j$ va a contribuire al coefficiente del monomio $s_{k+j} x^{k+j}$ di grado $k + j$, per tutti i valori possibili di k e j tra 0 ed $n - 1$; pertanto vale

$$W(x) = \sum_{j=0}^{2n-2} w_j x^j, \qquad \text{con} \qquad w_j = \sum_{k=0}^{j} p_k q_{j-k},$$

che sono esattamente tutti i termini che contribuiscono al coefficiente w_j.
L'ultima espressione si chiama *prodotto di convoluzione*; è evidente che nella formulazione appena vista, il calcolo dei coefficienti di W costa $O(n^2)$ operazioni aritmetiche.

Consideriamo ora il problema di calcolare i coefficienti del prodotto di due polinomi $W = P \cdot Q$. Il coefficiente del prodotto di due qualunque monomi $p_k x^k \cdot q_j x^j$ va a contribuire al coefficiente del monomio $s_{k+j} x^{k+j}$ di grado $k + j$, per tutti i valori possibili di k e j tra 0 ed $n - 1$; pertanto vale

$$W(x) = \sum_{j=0}^{2n-2} w_j x^j, \qquad \text{con} \qquad w_j = \sum_{k=0}^{j} p_k q_{j-k},$$

che sono esattamente tutti i termini che contribuiscono al coefficiente w_j.
L'ultima espressione si chiama *prodotto di convoluzione*; è evidente che nella formulazione appena vista, il calcolo dei coefficienti di W costa $O(n^2)$ operazioni aritmetiche.

Si può fare di meglio?

La trasformata di Fourier

La trasformata di Fourier discreta (DFT) di una sequenza di coefficienti

$$a = (a_0, a_1, \ldots, a_{n-1}),$$

è definita da

$$A = (A_0, A_1, \ldots, A_{n-1}),$$

dove

$$A_k = \sum_{j=0}^{n-1} a_j e^{-\frac{2\pi ikj}{n}} = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

dove

$$\omega_n^k = e^{-\frac{2\pi k}{n}}, \qquad \omega_n = e^{-\frac{2\pi}{n}}$$

è una radice n-esima complessa dell'unità, ossia un numero complesso tale che

$$(\omega_n^k)^n = 1.$$

Proprietà delle radici n-esime dell'unità

1. Se $d > 0$ allora
$$\omega_{dn}^{dk} = \omega_n^k;$$
2. Se $n > 0$ è pari allora
$$\omega_n^{n/2} = \omega_2 = -1;$$
3. Se $n > 0$ è pari, allora i quadrati delle radici n-esime sono le $n/2$ radici $(n/2)$-esime dell'unità;
4. Se $n > 0$ e $k > 0$ non divisibile per n, allora
$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Consideriamo di nuovo l'espressione della DFT (per n potenza di 2)

$$A_k = \sum_{j=0}^{n-1} a_j e^{-\frac{2\pi i k j}{n}},$$

e separiamo gli indici pari $j = 2l$ dagli indici dispari $j = 2l + 1$; è evidente che

$$A_k = \sum_{l=0}^{n/2-1} a_{2l} e^{-\frac{2\pi i k 2l}{n}} + \sum_{l=0}^{n/2-1} a_{2l+1} e^{-\frac{2\pi i k (2l+1)}{n}}$$

che si può anche scrivere come

$$A_k = \sum_{l=0}^{n/2-1} a_{2l} e^{-\frac{2\pi i k l}{n/2}} + e^{-\frac{2\pi i k}{n}} \sum_{l=0}^{n/2-1} a_{2l+1} e^{-\frac{2\pi i k l}{n/2}}.$$

Abbiamo quindi ottenuto che

$$A_k = A_{\text{even}} + e^{-\frac{2\pi ik}{n}} A_{\text{odd}}$$

FFT

Una DFT di lunghezza n si decompone nella somma di due DFT di lunghezza $n/2$

Siamo nelle condizioni del master theorem, quindi

Il costo di una FFT di dimensione $n = 2^p$ è $O(n \log(n))$

La trasformazione inversa DFT^{-1} si ottiene con lo stesso algoritmo[1] utilizzando

$$\bar{\omega}_n^k = e^{\frac{2\pi ik}{n}}$$

ed ha evidentemente lo stesso costo.

[1] A meno di un fattore $1/n$; si può anche usare un fattore $1/\sqrt{n}$ sia nella DFT che nella inversa

Infine, se indichiamo con

$$a \otimes b$$

il prodotto di convoluzione di due sequenze di lunghezza n, si ha il seguente

Teorema (Calcolo del prodotto di convoluzione)

$$a \otimes b = DFT_{2n}^{-1}(DFT_{2n}(a) \cdot DFT_{2n}(b)),$$

dove $DFT_{2n}(W)$ è la trasformata della sequenza W estesa con zeri fino a dimensione $2n$.

Ne segue che

Teorema (Costo del prodotto di polinomi)

Il prodotto di due polinomi si può calcolare in un tempo

$$\Theta(n \log(n))$$

La trasformata di Fourier

La trasformata di Fourier si usa in moltissime applicazioni; tutta l'elaborazione dei segnali e delle immagini è fondata sulla FFT.
La traduzione in codice efficiente dell'algoritmo ha diversi aspetti interessanti, che tuttavia non approfondiremo ulteriormente in questo corso.

La ricerca degli zeri di funzioni

Si consideri il problema della ricerca degli zeri di una funzione $\{x|F(x) = 0\}$. Per una funzione generica questo problema non ammette una formula chiusa di soluzione, quindi a stretto rigore non può esistere un algoritmo per la sua soluzione; tuttavia si può utilizzare un metodo che *approssimi* la soluzione stessa.

Il metodo di bisezione

Se una funzione è continua su un intervallo chiuso e limitato, allora assume tutti i valori compresi tra minimo e massimo. Quindi se abbiamo un intervallo $[a, b]$ e F assume segni diversi in a e b, allora esiste $x \in [a, b]$ tale che $F(x) = 0$. Restringendo l'intervallo si può approssimare x

Algorithm 4: float Bisection(float a, float b, float tol) :

```
float mid;
while (b − a) > tol do
    mid ← (a + b)/2;
    if sign(F(mid)) == sign(F(a)) then
        a ← mid;
    else
        b ← mid;
return mid
```