Capitolo 3

POLINOMI ED FFT

3.1 Polinomi

Un polinomio nella variabile x su un campo algebrico F è una funzione $A(x)$ che può essere rappresentata come:

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

dove a_j sono i coefficienti ed $n-1$ è il grado di un polinomio (supponiamo che a_{n-1} sia diverso da zero). I coefficienti sono estratti dal campo F, tipicamente l'insieme **C** dei numeri complessi.

La *somma di due polinomi* (per comodità supponiamo che siano dello stesso grado $n-1$) è la funzione $C(x)$ tale che $C(x) = A(x) + B(x)$ per tutti gli x del campo sottostante. $C(x)$ è definita come segue:

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

$$B(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$C(x) = A(x) + B(x) = \sum_{j=0}^{n-1} c_j x^j$$

dove $c_j = a_j + b_j$.

La *moltiplicazione di due polinomi* (per comodità supponiamo che siano dello stesso grado $n-1$) è la funzione $C(x)$ tale che $C(x) = A(x)B(x)$ per tutti gli x del campo sottostante. $C(x)$ è definita come segue:

$$C(x) = A(x)B(x) = \sum_{j=0}^{2n-2} c_j x^j$$

dove:

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

Si noti che il grado del prodotto è dato dalla somma dei gradi dei due polinomi.

3.2 Rappresentazione dei polinomi

3.2.1 Rappresentazione con coefficienti

La rappresentazione con coefficienti di un polinomio

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

è un vettore di coefficienti:

$$(a_0, a_1, ..., a_{n-1})$$

Nelle equazioni matriciali il vettore sarà considerato un vettore colonna.

La rappresentazione con coefficienti è conveniente per **valutare un polinomio**, cioè per trovare il valore $A(x_0)$ a partire da un certo x_0.

La valutazione richiede tempo $\Theta(n)$ se si usa la regola di Horner:

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + ... + x_0(a_{n-2} + x_0(a_{n-1}))...))$$

L'addizione di due polinomi rappresentati con coefficienti richiede banalmente tempo $\Theta(n)$.

Purtroppo, la **moltiplicazione di due polinomi** rappresentati con coefficienti richiede tempo $\Theta(n^2)$ se si considera il metodo descritto in precedenza (ogni coefficiente a deve essere moltiplicato per ogni coefficiente b).

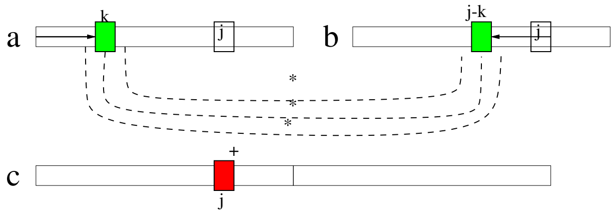

Figura 3.1: La convoluzione di due vettori.

Il vettore di coefficienti c che rappresenta il prodotto di due polinomi è anche detto **convoluzione** dei vettori in ingresso a e b, denotata come

$$c = a \otimes b$$

.

3.2.2 Rappresentazione punto-valore

La rappresentazione punto-valore consiste in n “esempi” della mappa eseguita dalla funzione (coppie ingresso-uscita).

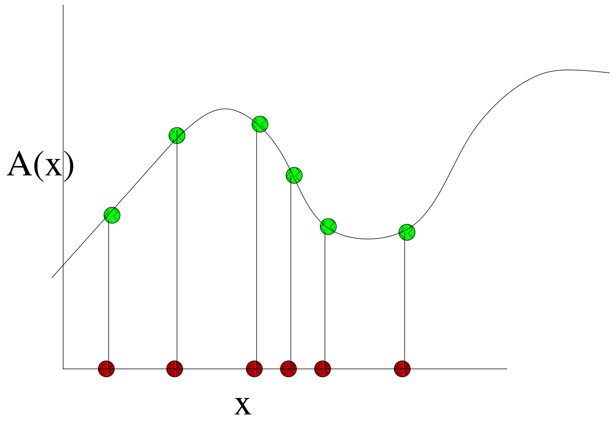

Figura 3.2: La rappresentazione punto-valore dei polinomi.

Precisamente, si tratta di n **paia punto-valore**

$$\{(x_0, y_0), (x_1, y_1), (x_{n-1}, y_{n-1})\}$$

tali che tutti i valori x_k sono distinti e $y_k = A(x_k)$. La rappresentazione non è univoca.

Il calcolo di una rappresentazione punto-valore richiede n valutazioni del polinomio, quindi $\Theta(n^2)$ in totale se si usa la regola di Horner (si vedrà che con una scelta oculata di particolari valori di x_k il calcolo può essere accelerato fino a $\Theta(n \log n)$).

L'operazione inversa della valutazione (ricostruire i coefficienti di un polinomio) da n coppie punto-valore, si chiama **interpolazione**.

Data una rappresentazione punto-valore, il polinomio interpolante è unico. Si tratta di risolvere un sistema di equazioni lineari: n equazioni in n incognite[1].

Il sistema è risolvibile in tempo $O(n^3)$, usando la decomposizione LU della matrice. Una tecnica più veloce è basata sulla formula di Lagrange, che richiede $\Theta(n^2)$ per calcolare tutti i coefficienti.

La rappresentazione punto-valore è comoda per eseguire somme e **moltiplicazioni** di polinomi (purchè polinomi diversi siano rappresentati scegliendo gli stessi valori $x_0, x_1, ..., x_{n-1}$). Per rappresentare il prodotto servono $2n$ coppie punto-valore, è quindi richiesta un'estensione delle rappresentazioni punto-valore di A e di B. Ad esempio, la rappresentazione estesa di A è:

$$\{(x_0, y_0), (x_1, y_1), (x_{2n-1}, y_{2n-1})\}$$

La moltiplicazione di polinomi nella rappresentazione punto-valore richiede *meno* della moltiplicazione nella rappresentazione con coefficienti, si passa da $\Theta(n^2)$ a $\Theta(n)$

3.2.3 Moltiplicazione veloce di polinomi nella rappresentazione con coefficienti

È possibile sfruttare la moltiplicazione "veloce" nella rappresentazione punto-valore per accelerare la moltiplicazione nella rappresentazione con coefficienti?

Sì, purchè si scelgano furbamente i punti di valutazione. Se si usano le **radici complesse dell'unità** è possibile produrre la rappresentazione punto-valore a partire dalla rappresentazione con coefficienti in tempo $\Theta(n \log n)$. Il passaggio si chiama Trasformata Discreta di Fourier (DFT). L'operazione inversa (inverse - DFT) riporta la rappresentazione nella forma con coefficienti.

La **Trasformata Veloce di Fourier** (Fast Fourier Transform – FFT) esegue la trasformazione diretta ed inversa in tempo $\Theta(n \log n)$. La strategia è illustrata in Fig. 3.3.

Per semplicità assumeremo che n sia una potenza di due e che, quando serve, si aggiungano dei coefficienti uguali a zero.

La strategia per la moltiplicazione veloce è la seguente:

[1]Si tratta della generalizzazione del noto principio per il quale per due punti nel piano passa una ed una sola retta – purchè ovviamente i punti siano distinti.

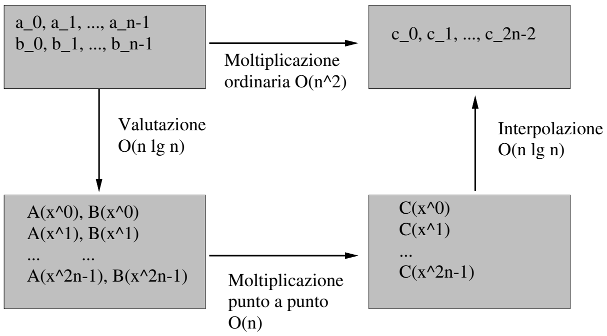

Figura 3.3: Come moltiplicare velocemente i polinomi.

1. Aumenta il grado di $A(x)$ e $B(x)$ portando entrambi a grado $2n-1$ (aggiungendo coefficienti uguali a zero).
2. Valuta i due polinomi usando le $2n$ radici dell'unità (FFT).
3. Moltiplica punto a punto
4. Crea la rappresentazione con coefficienti di $C(x)$ applicando ancora FFT per eseguire l'inversa della DFT.

È facile dimostrare che il ciclo completo richiede $\Theta(n \log n)$ una volta che si dimostra che la FFT richiede tempo $\Theta(n \log n)$.

3.3 La DFT e la FFT

Vediamo ora come eseguire la valutazione e l'interpolazione in tempo $\Theta(n \log n)$ usando le radici complesse dell'unità.

3.3.1 Radici complesse dell'unità

Una **radice complessa n-esima dell'unità** è un numero complesso ω tale che:

$$\omega^n = 1$$

Ci sono n radici complesse n-esime dell'unità: $e^{\frac{2\pi i k}{n}}$, per $k = 0, 1, ..., n-1$

Le n radici complesse sono rappresentabili in un piano, usando la relazione:

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

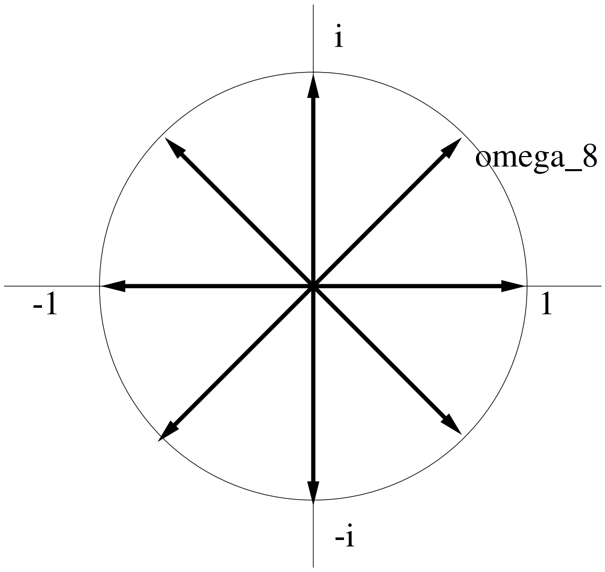

Figura 3.4: Radici complesse dell'unità: il caso $n = 8$.

Il valore: $\omega_n = e^{\frac{2\pi i}{n}}$ è detto **radice principale n-esima dell'unità**. Le altre radici complesse n-esime sono potenze di questa.

Le n radici complesse n-esime dell'unità: $\omega_n^0, \omega_n^1, ..., \omega_n^{n-1}$ sono un gruppo sotto la moltiplicazione.

È facile controllare le seguenti proprietà:

- **Lemma della cancellazione**: per ogni intero $n \geq 0, k \geq 0$ e $d > 0$

$$\omega_{dn}^{dk} = \omega_n^k$$

- per ogni $n > 0$:

$$\omega_n^{n/2} = \omega_2 = -1$$

- **Lemma del dimezzamento**: se $n > 0$ è pari, i quadrati delle n radici complesse n-esime dell'unità sono le $n/2$ radici $(n/2)$-esime dell'unità.

- **Lemma della somma**: per ogni $n \geq 1$ ed intero non-negativo k non divisibile per n:

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$$

Si usa l'equazione:

$$\sum_{j=0}^{n-1}(\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1}$$

Il denominatore è diverso da zero perchè k non è divisibile per n per le ipotesi fatte [2].

3.3.2 La DFT

Si vuole valutare il polinomio:

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

in corrispondenza delle n radici complesse n-esime dell'unità: $\omega_n^0, \omega_n^1, ..., \omega_n^{n-1}$ Assumiamo n potenza di due, sai A dato nella rappresentazione con coefficienti: $a = (a_0, a_1, ..., a_{n-1})$.

Definiamo come y_k i singoli risultati della valutazione:

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$$

Il vettore $y = (y_0, y_1, ..., y_{n-1})$ è la **Trasformata discreta di Fourier** (DFT) del vettore di coefficienti $a = (a_0, a_1, ..., a_{n-1})$. Scriveremo $y = DFT(a)$

3.3.3 La FFT

La **Trasformata Veloce di Fourier (FFT)** è un algoritmo efficiente, con tempo $\Theta(n \log n)$, per il calcolo della DFT che sfrutta le proprietà della radici dell'unità.

La FFT usa una strategia di *divide et impera*, definendo prima due sottoproblemi ottenuti considerando i coefficienti pari e dispari di $A(x)$. I due sottoproblemi sono dati dal calcolo dei due polinomi di grado $n/2-1$:

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + ... + a_{n-2} x^{n/2-1}$$

[2]Dal punto di vista "visivo": trovo il baricentro di punti distribuiti in modo uniforme attorno all'origine: esso risulta zero.

$$A^{[1]}(x) = a_1 + a_3x + a_5x^2 + ... + a_{n-1}x^{n/2-1}$$

è immediato controllare che:

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \tag{3.1}$$

Concludendo, il problema di valutare $A(x)$ sulle n radici complesse n-esime dell'unità: $\omega_n^0, \omega_n^1, ..., \omega_n^{n-1}$ è riducibile all'esecuzione dei dei due passi:

1. valutare i due polinomi di grado $n/2 - 1$: $A^{[0]}(x)$ ed $A^{[1]}(x)$ in corrispondenza dei punti:

$$(\omega_n^0)^2, (\omega_n^1)^2, ..., (\omega_n^{n-1})^2 \tag{3.2}$$

2. combinare i risultati secondo l'equazione 3.1

Ora, per il lemma del dimezzamento, i valori diversi nella lista 3.2 sono in realtà le $n/2$ radici $n/2$-esime dell'unità.

I sottoproblemi hanno quindi la stessa forma del problema originale, ma la dimensione è dimezzata.: abbiamo decomposto il calcolo di una DFT_n su n elementi al calcolo di due $DFT_{n/2}$ su $n/2$ elementi.

La decomposizione è alla base dell'algoritmo ricorsivo FFT, che calcola la DFT di un vettore ad n elementi $a = (a_0, a_1, ..., a_{n-1})$, dove n è una potenza di due.

```
#include "Complex.H"
#include <iostream.h>
#include <math.h>

double pi = 2.0 * asin(1.0);

int RecursiveFFT(complex *a, int n, complex *result)
{
if(n == 1) {
result[0] = a[0];
return 1;
}

complex omega_n(cos(2*pi / n), sin(2*pi / n));

complex omega(1.0);

complex *a0 =  new complex [n/2];
complex *a1 =  new complex [n/2];
complex *y0 =  new complex [n/2];
complex *y1 =  new complex [n/2];


for(int i=0; i < n/2; i++){
a0[i] = a[2*i];
a1[i] = a[2*i+1];
}

RecursiveFFT(a0, n/2, y0);
RecursiveFFT(a1, n/2, y1);

for(int k=0; k< n/2; k++){
result[k] = y0[k] + omega * y1[k];
result[k + (n/2)] = y0[k] - omega * y1[k];
omega = omega * omega_n;
}

delete [] a0;
delete [] a1;
delete [] y0;
delete [] y1;

return 1;
}
```

La procedura `RecursiveFFT` lavora in questo modo. La base della ricorsione è immediata:

$$y_0 = a_0 \omega_1^0 = a_0$$

Per l'aggiornamento di ω, quando viene eseguito il ciclo iterativo per la k-esima iterazione $\omega = \omega_n^k$.

Il calcolo ricorsivo per $k = 0, 1, ..., n/2 - 1$ avviene usando il lemma della cancellazione:

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_{n/2}^k) y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_{n/2}^k)$$

Per comprendere l'ultima linea si noti che, per $k = 0, 1, ..., n/2 - 1$:

$$\begin{aligned} y_{k+(n/2)} &= y_k^{[0]} - \omega_n^k y_k^{[1]} && (3.3) \\ &= y_k^{[0]} + \omega_n^{k+(n/2)} y_k^{[1]} && (3.4) \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k}) && (3.5) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k+n}) && (3.6) \\ &= A(\omega_n^{k+(n/2)}) && (3.7) \\ & && (3.8) \end{aligned}$$

3.4 Realizzazioni efficienti della FFT