FFT

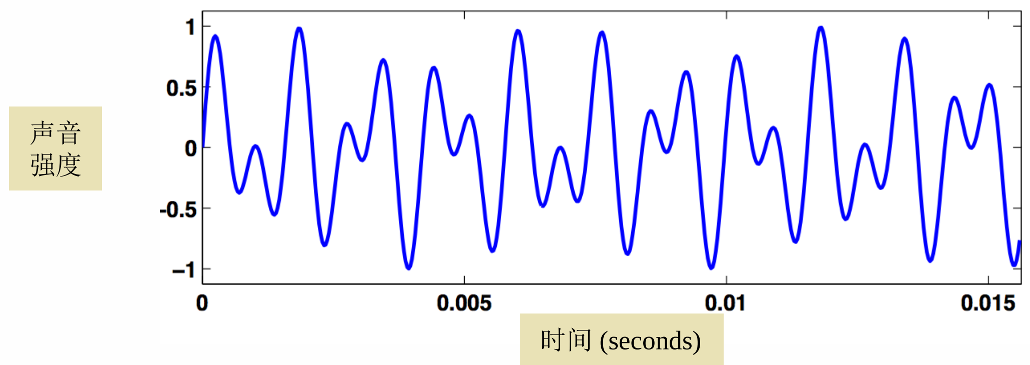
授课人：长沙市一中 周祖松

FFT的产生与发展

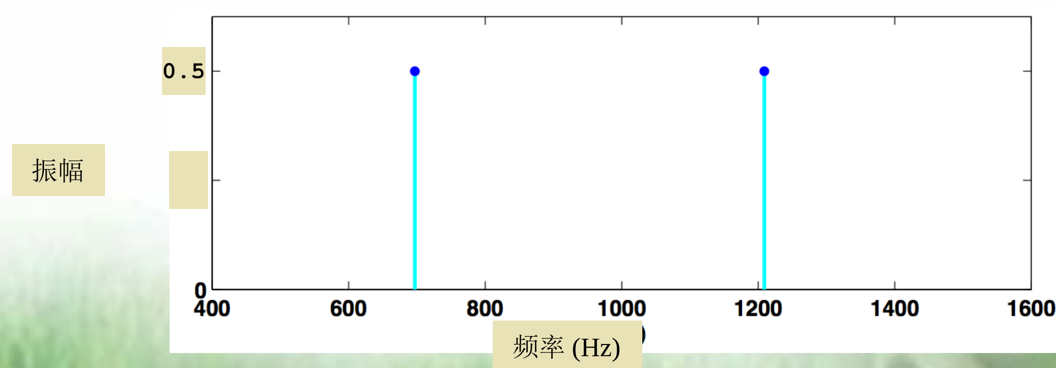
- Fourier（1807）任何周期性信号都可以用一系列正弦函数来表示时域和频域
- Gauss (1805, 1866). 行星运动周期分析(天文计算)
- Runge-König (1924).理论基础。
- Danielson-Lanczos (1942). 设计出高效算法(X-射线技术)
- Cooley库里-Tukey图基 (1965). 发表论文《机器计算傅立叶级数的一种算法》. FFT开始普及和大规模应用.

时域与频域

- 信号
- 时域



- 频域.

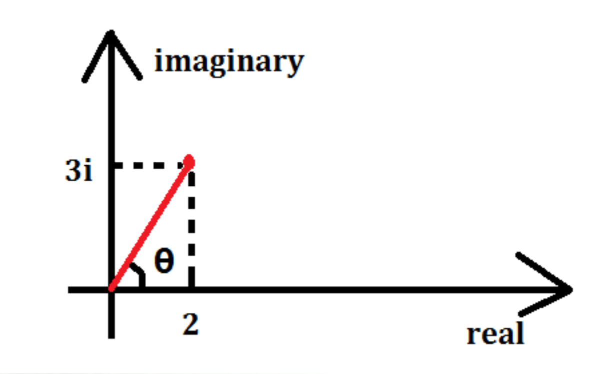


预备知识

- **复数**

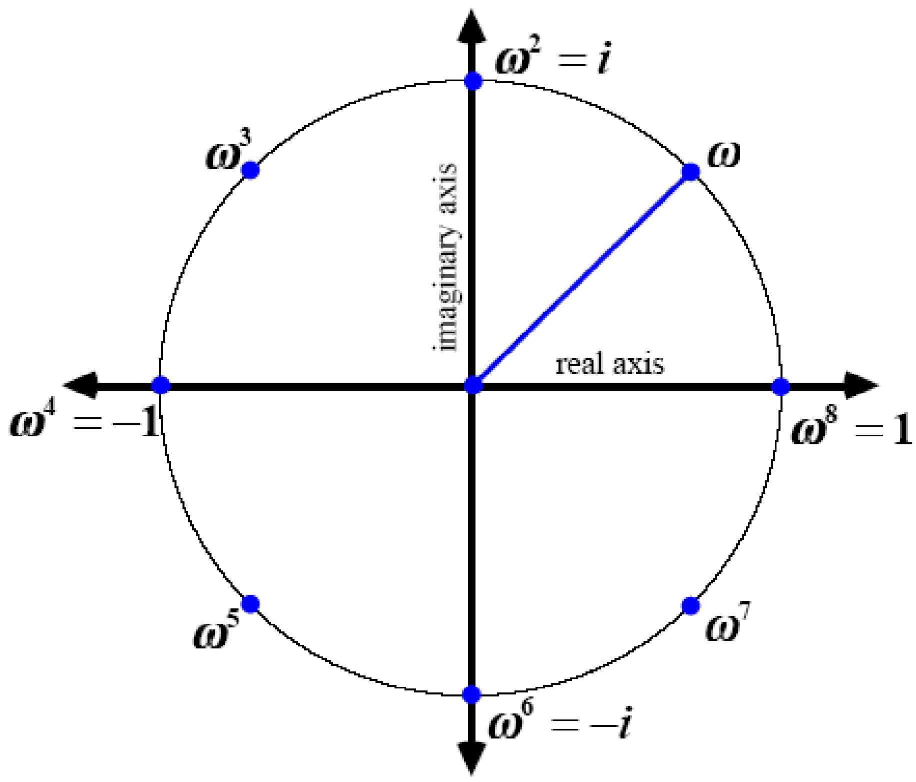
$z=a+bi$

$Z=r*\cos(\theta)+r*\sin(\theta)i$



单位根

- 一个数X的N次方等于1,则称X是N的单位根 $\omega^n = 1$.
- 有N个单位根: $\omega^0, \omega^1, \ldots, \omega^{n-1}$
- $\omega = \cos(2\pi/n) + i\sin(2\pi/n)$
- $\omega^x * \omega^x = \omega^{2x}$
- $\omega^x * \omega^y = \omega^{x+y}$
- $\omega^{k+n/2} = -\omega^k$



预备知识

多项式:系数表示法

- 多项式:系数表示法:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

- 加法: 时间$O(n)$.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

- 乘法: 暴力时间复杂度$O(n^2)$

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

- $A(x) = 6x^3 + 7x^2 - 10x + 9$ $\quad (9, -10, 7, 6)$
- $B(x) = -2x^3 + 4x - 5$ $\quad (-5, 4, 0, -2)$

$$
\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -30x^3 & -35x^2 & +50x & -\ 45 \\
 & & 24x^4 & +28x^3 & -40x^2 & +36x & \\
-12x^6 & -14x^5 & +20x^4 & -18x^3 & & & \\
\hline
-12x^6 & -14x^5 & +44x^4 & -20x^3 & -75x^2 & +86x & -\ 45
\end{array}
$$

HDU 1402A * B Problem Plus

- 问题:
- 计算 A * B.
- 输入两个长度少于50000整数A和B。

- 输出两个整数的积 A * B。
- 样例

输入	输出
1 2 1000 2	2 2000

A*B问题

$$
\begin{array}{r}
\longleftarrow n \longrightarrow \\
1124 \\
\times \quad 8317 \\
\hline
7868 \\
1124 \\
3372 \\
+\ 8992 \\
\hline
9348308 \\
\longleftarrow 2n \longrightarrow
\end{array}
\qquad \updownarrow n
$$

输入： 两个***n*位的整数*x* 和*y***

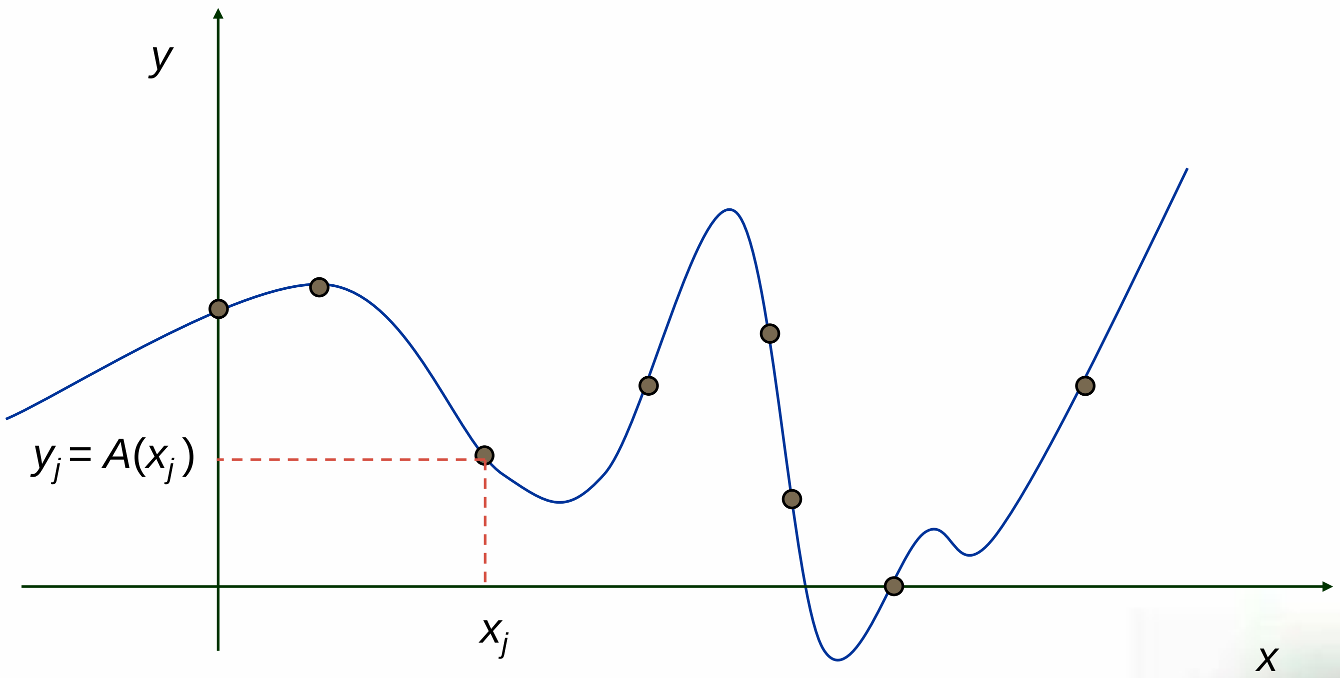
输出： ***z* = *x* * *y***

标准算法时间复杂度： $O(n^2)$

有更优的算法吗?

跳出思维的局限

多项式： 点值表示



多项式： 点值表示

- 多项式. [点值表示]

$$A(x):\ (x_0, y_0), \ldots, (x_{n\text{-}1}, y_{n-1})$$

$$B(x):\ (x_0, z_0), \ldots, (x_{n\text{-}1}, z_{n-1})$$

- 加法: $O(n)$

$$A(x)+B(x):\ (x_0, y_0+z_0), \ldots, (x_{n\text{-}1}, y_{n-1}+z_{n-1})$$

- 乘法 (卷积). $O(n)$, 需要取2n-1 个点

$$A(x)\times B(x):\ (x_0, y_0\times z_0), \ldots, (x_{2n\text{-}1}, y_{2n-1}\times z_{2n-1})$$

- $A(x) = x^3 - 2x + 1$
- $B(x) = x^3 + x^2 + 1$
- $x_k = (-3, -2, -1, 0, 1, 2, 3)$ *We need 7 coefficients!*
- A: $\{(-3, -17), (-2, -3), (-1, 1), (0, 1), (1, 0), (2, 5), (3, 22)\}$
- B: $\{(-3, -20), (-2, -3), (-1, 2), (0, 1), (1, 3), (2, 13), (3, 37)\}$
- C: $\{(-3, 340), (-2, 9), (-1, 2), (0, 1), (1, 0), (2, 65), (3, 814)\}$

多项式乘法算法:

- Input.
  - 两个系数表示的n次多项式 A 和 B
- Output.
  - 它们的 积 C=A*B
- 选点.
  - 选取2*N-1个点 $x_0 , ..., x_{2n-2}$
- 计算
  - 计算 $A(x_i), B(x_i)$
- 相乘.
  - $C_i = A(x_i)B(x_i)$
- 插值
  - 还原系数表示法 C

$O(n^2)$

$O(n)$

$O(n^2)$

多项式乘法算法:

系数表示

$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$

FFT $O(n \log n)$

$A(x_0) \ldots A(x_{2n-2})$
$B(x_0) \ldots B(x_{2n-2})$

点值乘法

$O(n)$

$C(x_0) \ldots C(x_{2n-2})$

点值表示

Inverse FFT $O(n \log n)$

$c_0, c_1, \ldots, c_{2n-2}$

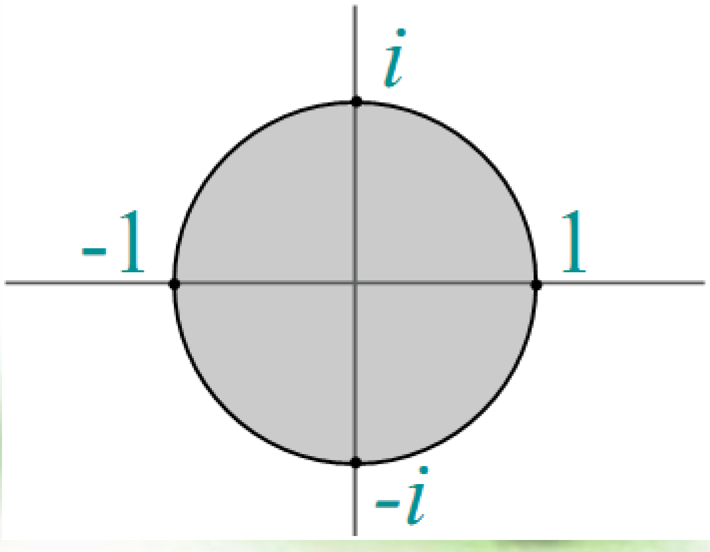
系数表示法转为点值表示法

- 系数 $\Rightarrow$ 点值. 对于给定的多项式 $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, 从中取$n$ 个不同的点 $x_0, \ldots, x_{n-1}$.

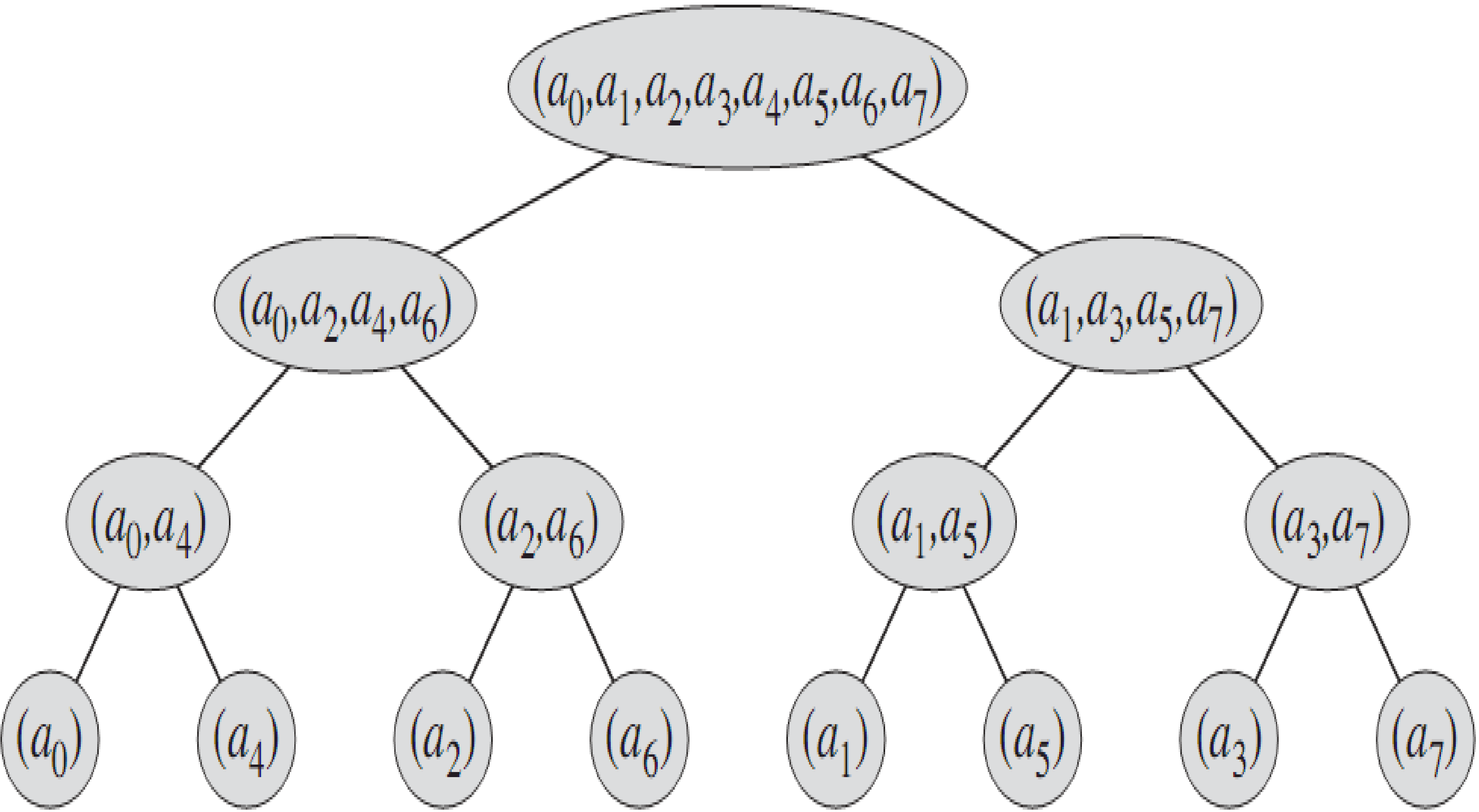
- **按指数奇偶分治**
  - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
  - $A_{even}(x) = a_0 + a_2x^1 + a_4x^2 + a_6x^3.$
  - $A_{odd}(x) = a_1 + a_3x^1 + a_5x^2 + a_7x^3.$
  - $A(x) = A_{even}(x^2) + x\,A_{odd}(x^2).$
  - $A(-x) = A_{even}(x^2) - x\,A_{odd}(x^2).$

- **选两个点为:±1.**
  - $A(1) = A_{even}(1) + 1\,A_{odd}(1).$
  - $A(-1) = A_{even}(1) - 1\,A_{odd}(1).$

- **选四个点为:±1, ± i.**



Fast Fourier Transform

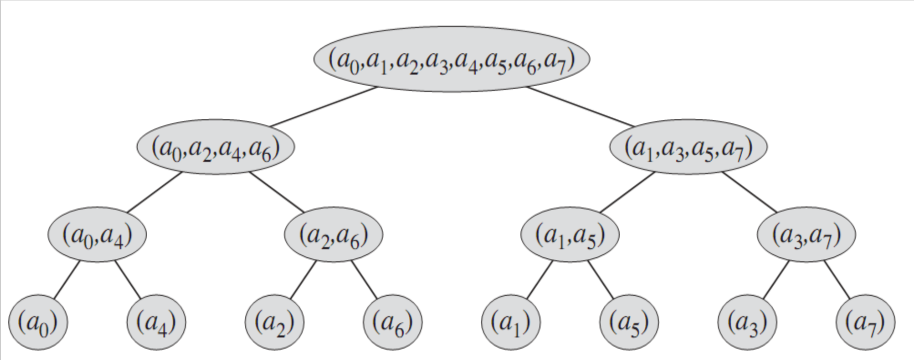


```
fft(n, a_0,a_1,…,a_{n-1}) {
   if (n == 1) return a_0

   (e_0,e_1,…,e_{n/2-1}) ← FFT(n/2, a_0,a_2,a_4,…,a_{n-2})
   (d_0,d_1,…,d_{n/2-1}) ← FFT(n/2, a_1,a_3,a_5,…,a_{n-1})

   for k = 0 to n/2 - 1 {
      ω^k      ← cos(2*pai*k/n)+sin(2pai*k/n)i
      y_k      ← e_k + ω^k d_k
      y_{k+n/2} ← e_k - ω^k d_k
   }

   return (y_0,y_1,…,y_{n-1})
}
```



```
void FFT(int complex a[] ,int len)
{   if(len==1) return ;
    int mid=len/2;
    complex buf[len];
    for(int i=0;i<=mid; i++)  buf[i]=a[2*i],buf[i+mid]=a[2*i+1];
    for(int i=0;i<=len; i++)  a[i]=buf[i];
    fft(a,mid);   fft(a+mid,mid);
   complex wn=complex(cos(2*PI/h),sin(2*PI/h)),w=complex(1,0);
    for(int i=0;i<=mid; i++)
    {   buf[i]=a[i]+w*a[i+mid];
        buf[i+mid]=a[i]-w*a[i+mid];
        w = w*wn;
     }
     for(int i=0;i<=len; i++)  a[i]=buf[i];
 }
```

点值－系数

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & & \omega^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

IFFT

$$G_n = \frac{1}{n}\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & & \omega^{-(n-1)(n-1)} \end{pmatrix}$$

$$\omega^{-1} = e^{-2\pi i/n}$$

快速傅里叶逆变换(IFFT)

$(y0,y1,y2,\ldots,yn-1)$为$(a0,a1,a2,\ldots,an-1)$的点值表示）

设有另一个向量$(c0,c1,c2,\ldots,cn-1)$满足

$$c_k = \sum_{i=0}^{n-1} y_i (\omega_n^{-k})^i$$

即多项式$B(x)=y_0+y_1x+y_2x^2+\ldots+y_{n-1}x^{n-1}$在$\omega^0,\omega^{-1},\omega^{-2},\ldots,\omega^{-(n-1)}$处的点值表示

可得：$a_k=c_k/n$

$$
\begin{aligned}
c_k &= \sum_{i=0}^{n-1} y_i(\omega_n^{-k})^i \\
&= \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1} a_j(\omega_n^i)^j)(\omega_n^{-k})^i \\
&= \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1} a_j(\omega_n^j)^i)(\omega_n^{-k})^i \\
&= \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1} a_j(\omega_n^j)^i(\omega_n^{-k})^i) \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_j(\omega_n^{j-k})^i \\
&= \sum_{j=0}^{n-1} a_j(\sum_{i=0}^{n-1}(\omega_n^{j-k})^i)
\end{aligned}
$$

考虑一个式子

$$S(\omega_n^k)=1+\omega_n^k+(\omega_n^k)^2+\cdots+(\omega_n^k)^{n-1}$$

当 $k\neq 0$ 时，两边同时乘上 ω_n^k 得

$$\omega_n^k S(\omega_n^k)=\omega_n^k+(\omega_n^k)^2+(\omega_n^k)^3+\cdots+(\omega_n^k)^n$$

两式相减，整理后得

$$\begin{aligned}\omega_n^k S(\omega_n^k)-S(\omega_n^k)&=(\omega_n^k)^n-1\\ S(\omega_n^k)&=\frac{(\omega_n^k)^n-1}{\omega_n^k-1}\end{aligned}$$

分子为零，分母不为零，所以

$$S(\omega_n^k)=0$$

当 $k=0$ 时，显然 $S(\omega_n^k)=n$。

继续考虑上式

$$\begin{aligned} c_k &= \sum_{j=0}^{n-1} a_j \left(\sum_{i=0}^{n-1} (\omega_n^{j-k})^i\right) \\ &= \sum_{j=0}^{n-1} a_j S(\omega_n^{j-k}) \end{aligned}$$

当 $j = k$ 时，$S(\omega_n^{j-k}) = n$，否则 $S(\omega_n^{j-k}) = 0$，即

$$\begin{aligned} c_i &= n a_i \\ a_i &= \frac{1}{n} c_i \end{aligned}$$

```
ifft(n, a_0,a_1,…,a_{n-1}) {
    if (n == 1) return a_0

    (e_0,e_1,…,e_{n/2-1}) ← FFT(n/2, a_0,a_2,a_4,…,a_{n-2})
    (d_0,d_1,…,d_{n/2-1}) ← FFT(n/2, a_1,a_3,a_5,…,a_{n-1})

    for k = 0 to n/2 - 1 {
        ω^k ← cos(2*pai*k/n)-sin(2pai*k/n)i
        y_k     ← (e_k + ω^k d_k) / n
        y_{k+n/2} ← (e_k - ω^k d_k) / n
    }

    return (y_0,y_1,…,y_{n-1})
}
```

```
void IFFT(int complex a[] ,int len)
{   if(len==1) return ;
    int mid=len/2;
    complex buf[len];
    for(int i=0;i<=mid; i++)  buf[i]=a[2*i],buf[i+mid]=a[2*i+1];
    for(int i=0;i<=len; i++)  a[i]=buf[i];
    fft(a,mid);   fft(a+mid,mid);
   complex wn=complex(cos(2*PI/h),-sin(2*PI/h)),w=complex(1,0);
    for(int i=0;i<=mid; i++)
    {   buf[i]=a[i]+w*a[i+mid];
        buf[i+mid]=a[i]-w*a[i+mid];
        w = w*wn;
     }
     for(int i=0;i<=len; i++)  a[i]=buf[i];
 }
```

Running time.

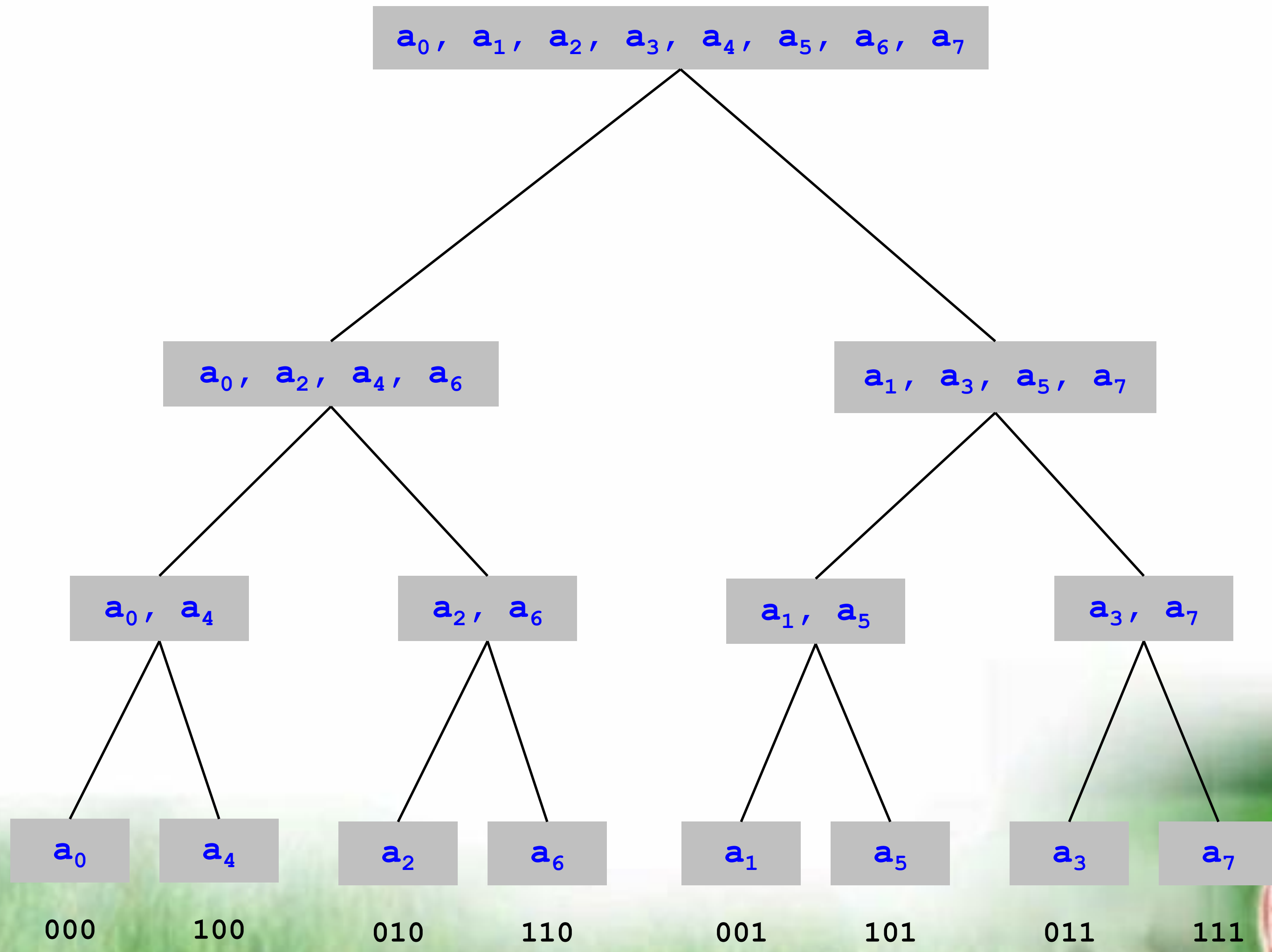
$$T(n) = 2T(n/2) + \Theta(n) \;\Rightarrow\; T(n) = \Theta(n \log n)$$

$$\begin{tikzcd}[column sep=huge] a_0, a_1, \ldots, a_{n\text{-}1} \arrow[r, shift left, "O(n \log n)"] & (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1}) \arrow[l, shift left, "{\color{red}O(n \log n)}"] \end{tikzcd}$$

coefficient representation

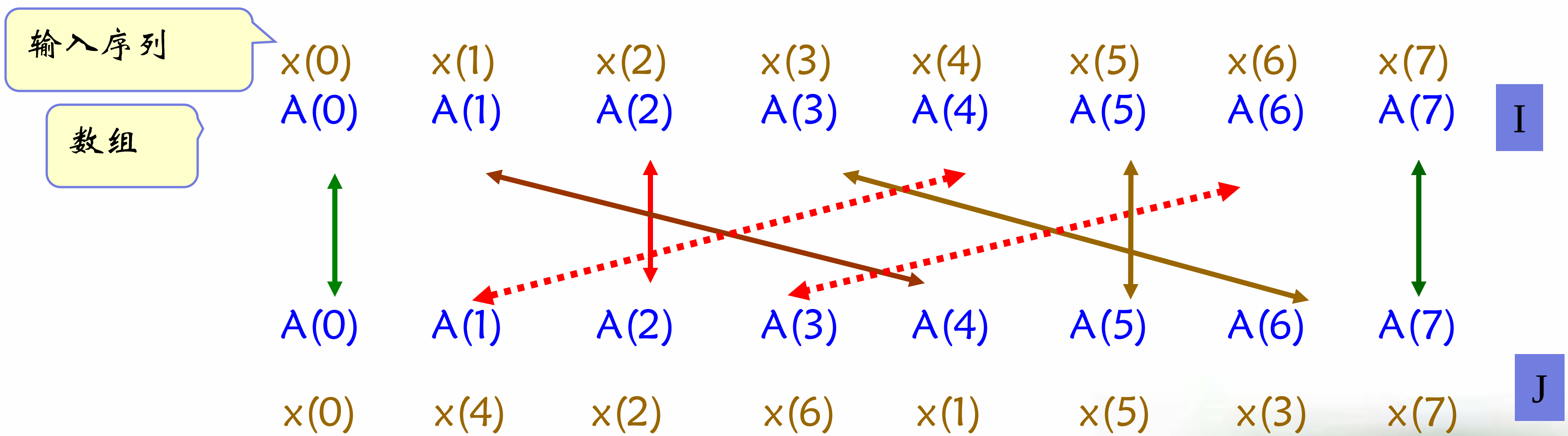
point-value representation

程序优化



a0, a1, a2, a3, a4, a5, a6, a7
a0, a2, a4, a6
a1, a3, a5, a7
a0, a4
a2, a6
a1, a5
a3, a7
a0
a4
a2
a6
a1
a5
a3
a7
000
100
010
110
001
101
011
111

FFT算法



分析上图N=8点倒序规律，顺序数**I**与倒序数**J** 存在关系：

- I = J时，不交换；
- I < J时，交换存储器内容。
- I > J时，不交换，直接更新序数值；

```
const double PI = acos(-1.0);
struct complex{//复数结构体
    double r,i;
    complex(double _r = 0.0,double _i = 0.0) {     r = _r; i = _i;   }
    complex operator +(const complex &b)   {     return complex(r+b.r,i+b.i);   }
    complex operator -(const complex &b)   {     return complex(r-b.r,i-b.i);   }
    complex operator *(const complex &b)   {     return complex(r*b.r-i*b.i,r*b.i+i*b.r);  }
};

//雷德算法
void rev(complex y[],int len)  //进行FFT和IFFT前的反转变换
{    int i,j,k;                //位置i和（i二进制反转后位置）互换
    for(i = 1, j = len/2;i < len-1; i++)
    {   if(i < j)swap(y[i],y[j]);
        k = len/2;
        while( j >= k)      {   j -= k;         k /= 2;      }
        if(j < k) j += k;
    }
}
```

```
void fft(complex y[],int len,int on)   // on==1时是DFT，on==-1时是IDFT
{   rev(y,len);   //len必须为2^k形式
    for(int h = 2; h <= len; h <<= 1)
    {   complex wn(cos(2*PI/h),sin(on*2*PI/h)); //单位复根e^(2*PI/m)用欧拉公式展开
        for(int j = 0;j < len;j+=h)
        {   complex w(1,0); //旋转因子
            for(int k = j;k < j+h/2;k++)
            {   complex u = y[k];               complex t = w*y[k+h/2];
                y[k] = u+t;       y[k+h/2] = u-t; //合并操作
                 w = w*wn; //更新旋转因子
             }
        }
    }
}
```

```
const int MAXN = 200010;                    int sum[MAXN];
complex x1[MAXN],x2[MAXN];            char str1[MAXN/2],str2[MAXN/2];
int main()
{     int len1 = strlen(str1);       int len2 = strlen(str2);       int len = 1;
      while(len < len1*2 || len < len2*2)len<<=1;    // len取2的幂
      for(int i = 0;i < len1;i++)            x1[i] = complex(str1[len1-1-i]-'0',0);
      for(int i = len1;i < len;i++)        x1[i] = complex(0,0);
      for(int i = 0;i < len2;i++)            x2[i] = complex(str2[len2-1-i]-'0',0);
      for(int i = len2;i < len;i++)        x2[i] = complex(0,0);
      fft(x1,len,1);        fft(x2,len,1); //求DFT
      for(int i = 0;i < len;i++)            x1[i] = x1[i]*x2[i];
      fft(x1,len,-1); //求IDFT
      for(int i = 0;i < len;i++)               x1[i].r /= len;
      for(int i = 0;i < len;i++)           sum[i] = (int)(x1[i].r+0.5);
      for(int i = 0;i < len;i++)       {          sum[i+1]+=sum[i]/10;          sum[i]%=10;       }
      len = len1+len2-1;
      while(sum[len] <= 0 && len > 0)len--;
      for(int i = len;i >= 0;i--)           printf("%c",sum[i]+'0');
     return 0;
}
```

快速数论变换（NTT）

下回分解

练习题

- BZOJ2179: FFT快速傅立叶
- BZOJ2194: 快速傅立叶之二
- BZOJ4827 [*Hnoi2017*]*礼*

Thanks !