algorithm

4.c.分治策略

本章内容

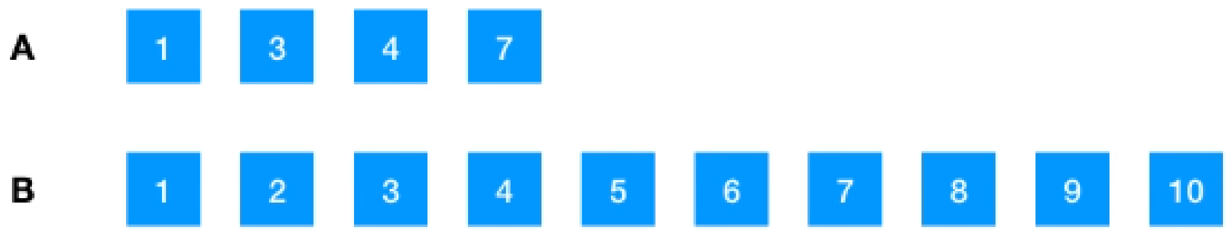
- 4.12 寻找两个有序数组的中位数
- 4.13 卷积及应用
- 4.14 卷积计算
- 4.15 快速傅立叶变换FFT算法
- 4.16 平面点集的凸包

4.12 寻找两个有序数组的中位数

- 给定两个大小为 m 和 n 的有序数组 nums1 和 nums2。
- 请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。
- 假设 nums1 和 nums2 不会同时为空。

- 示例 1:
  - nums1 = [1, 3]
  - nums2 = [2]
  - 则中位数是 2.0
- 示例 2:
  - nums1 = [1, 2]
  - nums2 = [3, 4]
  - 则中位数是 (2 + 3)/2 = 2.5

蛮力法



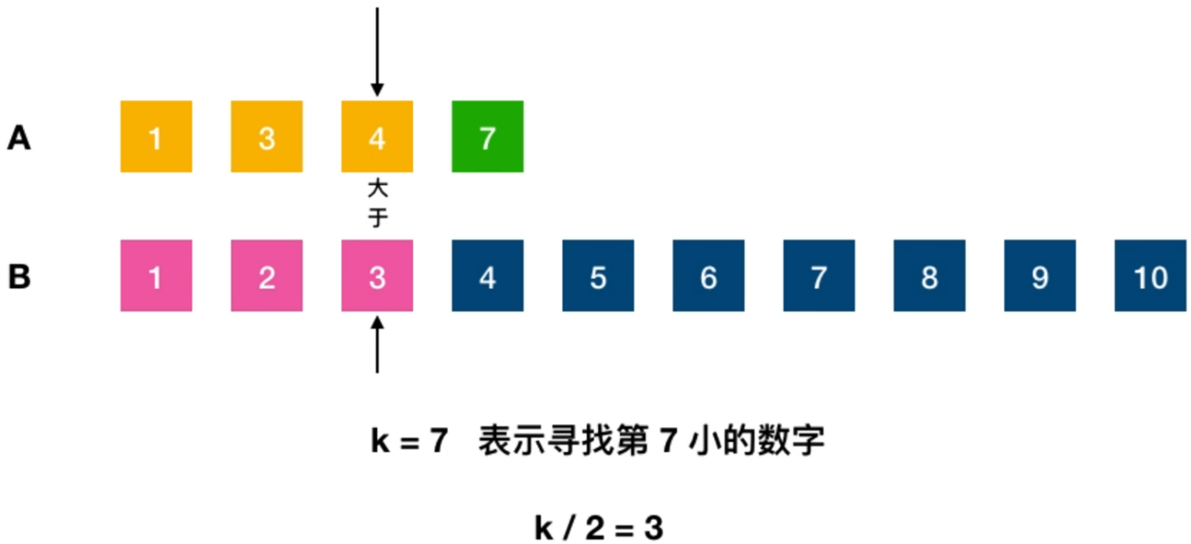
- 蛮力法1：
  - 将两个数组合并为一个数组 O(n)
  - 并对合并的数组进行排序 O(nlogn)
  - 输出排序后中间一位或中间两位之和 O(1)
  - 时间复杂度：O(nlogn)
- 蛮力法2：
  - 在两个数组小端各设置一个指针
  - 比较两个指针大小，谁小将谁右移一位，相等则任意选一个指针移动
  - 直到移动m+n/2位以后，直选或判断求和
  - 时间复杂度：O((m+n)/2)

寻找两个有序数组的中位数

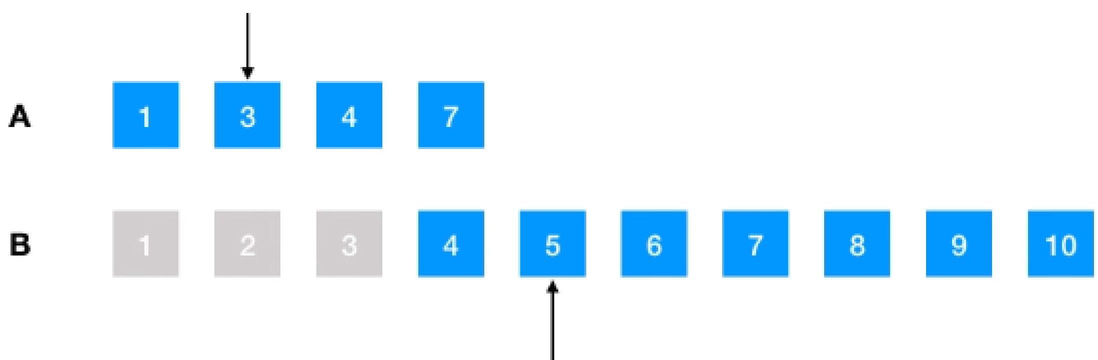
- 中位数即第(m+n)/2小的数（或之和）
- 求第k小的最佳时间复杂度是nlogn，但本题有一个前提，即两个数组均已排好序，因此可以对求最k小的问题做一个变形
- 在蛮力法2中，思路是一次次比较，一次次排除
- 是否可以一次性多排除一些数字?
- 假设我们要找第 k 小数，实际上我们可以每次比较后 排除 k/2 个数字
- 假设在下面两个排好序的数组上求中位数:
  - A = [1,3,4,7]
  - B = [1,2,3,4,5,6,7,8,9,10]
- A和B长度之和为14，因此问题等同于求第7和第8小数字之和

寻找两个有序数组的中位数

- 先求第7小，k=4，则 k/2 大的数为 4 与 3

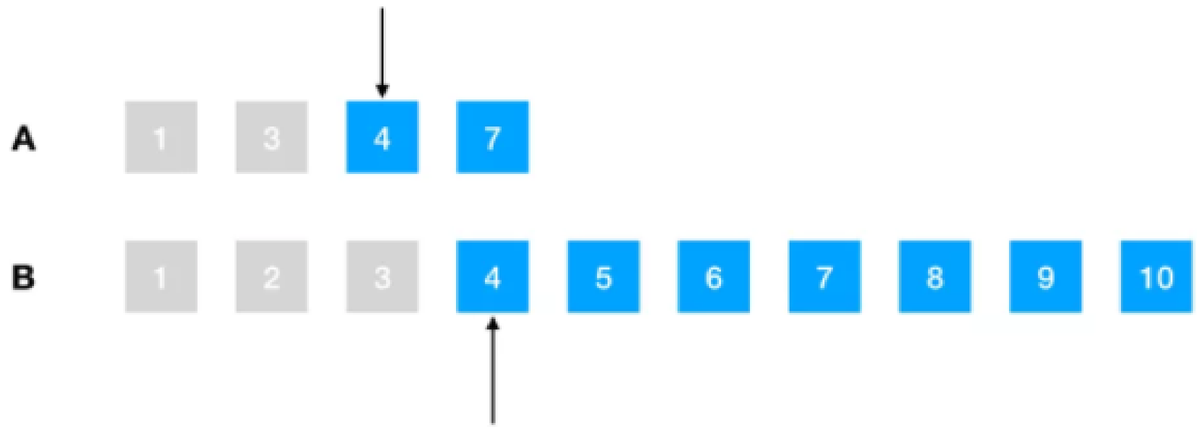


- 4>3，因此3和它以前的数字都不可能是中位数，故舍弃123，则问题变为了要查找第k=7-3=4的数字，则k/2=2

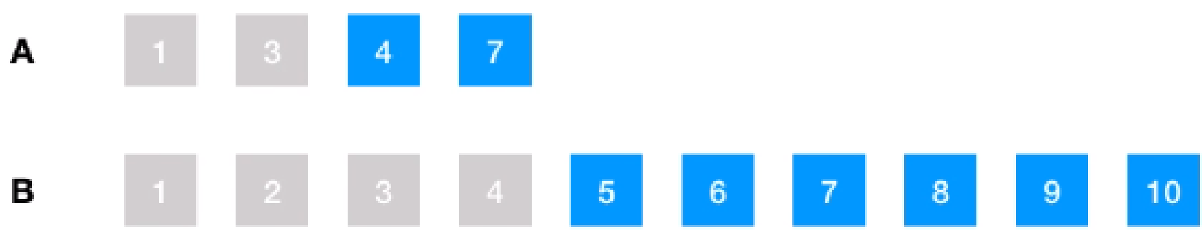


寻找两个有序数组的中位数

- 5>3，因此舍弃1和3，此时k=4-2=2，k/2=1

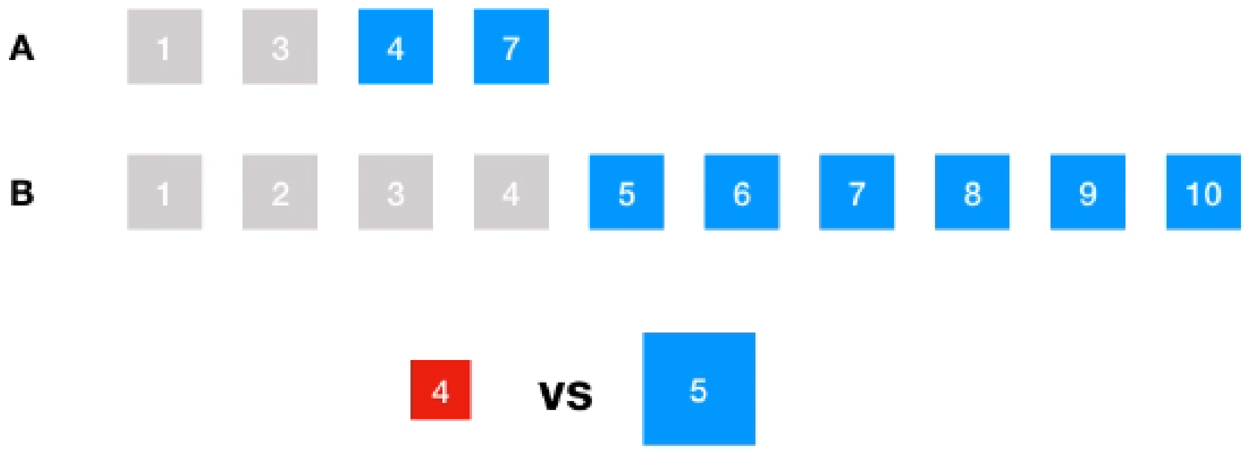


- 此时两个比较的数字均为4，这种情况下随意舍弃一个就行，假设舍弃B数组的4

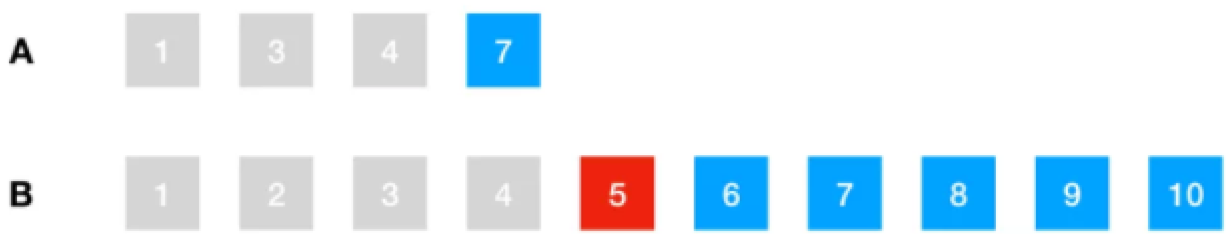


寻找两个有序数组的中位数

- A的4和B下一位的5比较，4是小的，因此第7小是4



- 将5和A剩下的7比较，得到第8小的数是5



- 所以，A 数组和 B 数组的中位数是 (4+5)/2=4.5
- 因为每次排除的数字是k/2个，所以时间复杂度为O(logn)

4.13 卷积及应用

- 向量计算
  - 给定向量 $a = (a_0, a_1, \ldots, a_{n-1})$

    $b = (b_0, b_1, \ldots, b_{n-1})$
  - 向量和 $a+b = (a_0+b_0, a_1+b_1, \ldots, a_{n-1}+b_{n-1})$
  - 内积 $a \cdot b = a_0 b_0 + a_1 b_1 + \ldots + a_{n-1} b_{n-1}$
  - 卷积 $a*b = (c_0, c_1, \ldots, c_{2n-2})$，其中

$$c_k = \sum_{\substack{i+j=k \\ i,j<n}} a_i b_j, \quad k = 0, 1, \ldots, 2n-2$$

卷积的含义

- 在下述矩阵中，每个斜线的项之和恰好是卷积中的各个分量

$$
\begin{array}{c|ccccc}
 & ab_0 & ab_1 & \cdots & ab_{n-2} & ab_{n-1} \\
\hline
a_0 b & a_0b_0 & a_0b_1 & \cdots & a_0b_{n-2} & a_0b_{n-1} \\
a_1 b & a_1b_0 & a_1b_1 & \cdots & a_1b_{n-2} & a_1b_{n-1} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
a_{n-2} b & a_{n-2}b_0 & a_{n-2}b_1 & \cdots & a_{n-2}b_{n-2} & a_{n-2}b_{n-1} \\
a_{n-1} b & a_{n-1}b_0 & a_{n-1}b_1 & \cdots & a_{n-1}b_{n-2} & a_{n-1}b_{n-1}
\end{array}
$$

$c_0 \quad c_1 \quad c_{n-1} \quad c_{2n-3} \quad c_{2n-2}$

计算实例

- 向 量 $a = (1, 2, 4, 3)$, $b = (4, 2, 8, 0)$
- 则 $a+b = (5, 4, 12, 3)$

  $a \cdot b = (4, 4, 32, 0)$

  $a * b = (4, 10, 28, 36, 38, 24, 0)$

	ab_0	ab_1	ab_2	ab_3
a_0b	1×4	1×2	1×8	1×0
a_1b	2×4	2×2	2×8	2×0
a_2b	4×4	4×2	4×8	4×0
a_3b	3×4	3×2	3×8	3×0

$c_2 = 4\times4 + 2\times2 + 1\times8 = 28$

卷积与多项式乘法

- **多项式乘法：** $C(x) = A(x)\,B(x)$

$$A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{m-1}x^{m-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1}$$

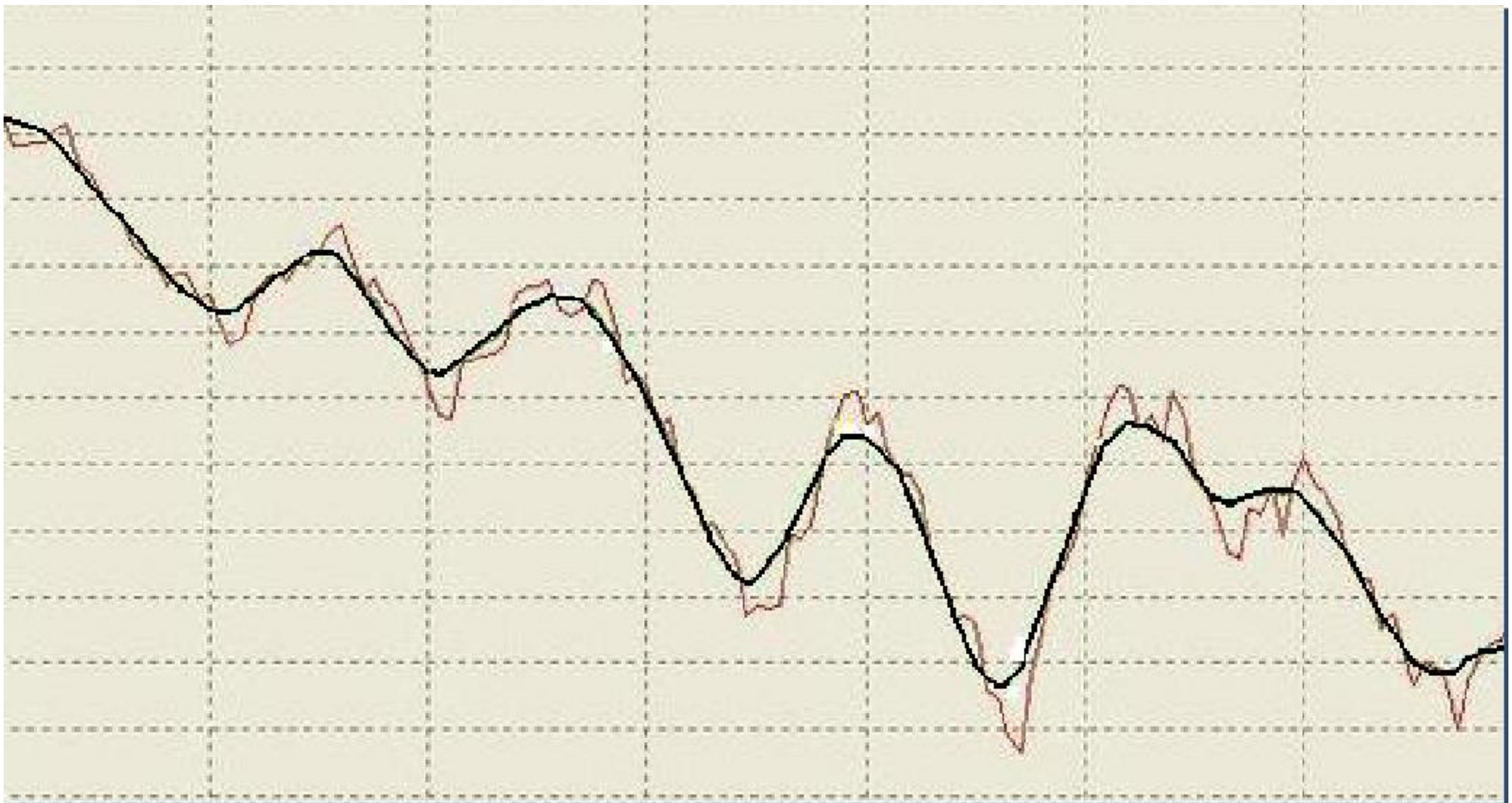
$$C(x) = a_0b_0 + (a_0b_1+a_1b_0)\,x + \ldots \quad + a_{m-1}b_{n-1}\,x^{m+n-2}$$

- 其中 x^k 的系数

$$c_k = \sum_{\substack{i+j=k \\ i\in\{0,1,\ldots,m-1\} \\ j\in\{0,1,\ldots,n-1\}}} a_i b_j, \quad k = 0, 1, \ldots, m+n-2$$

卷积应用：信号平滑处理

- 由于噪音干扰，对信号需要平滑处理

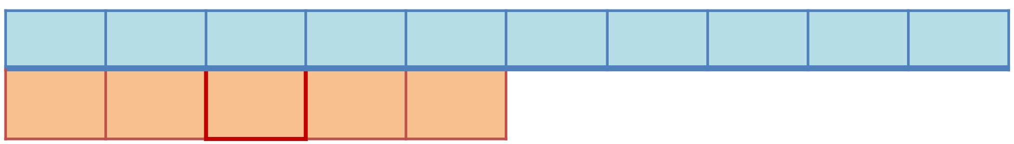


平滑处理

- 信号向量： $a=(a_0, a_1, \ldots, a_{m-1})$

  $b=(b_{2k},b_{2k-1},\ldots,b_0)=(w_{-k}, \ldots, w_k)$

$$a_i' = \sum_{s=-k}^{k} a_{i+s} b_{k-s} = \sum_{s=-k}^{k} a_{i+s} w_s$$



- 把向量 b 看作$2k+1$长度窗口在a上移 动计算 $a*b$, 得到 $(a_0', a_1', \ldots, a_{m-1}')$. 有 少数项有误差.

实例

- 信号向量： $a = (a_0, a_1, \ldots , a_8)$
- 步长： $k = 2$
- 权值： $w = (w_{-2}, w_{-1}, w_0, w_1, w_2) = (b_4, b_3, b_2, b_1, b_0)$

$$a_i' = a_{i-2}b_4 + a_{i-1}b_3 + a_ib_2 + a_{i+1}b_1 + a_{i+2}b_0$$

- 下标之和为 $i + k$

w_{-2}	w_{-1}	w_0	w_1	w_2				
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

a_0b_0	a_0b_1	a_0b_2	a_0b_3	a_0b_4	a_2'
a_1b_0	a_1b_1	a_1b_2	a_1b_3	a_1b_4	a_3'
a_2b_0	a_2b_1	a_2b_2	a_2b_3	a_2b_4	a_4'
a_3b_0	a_3b_1	a_3b_2	a_3b_3	a_3b_4	a_5'
a_4b_0	a_4b_1	a_4b_2	a_4b_3	a_4b_4	a_6'
a_5b_0	a_5b_1	a_5b_2	a_5b_3	a_5b_4	
a_6b_0	a_6b_1	a_6b_2	a_6b_3	a_6b_4	
a_7b_0	a_7b_1	a_7b_2	a_7b_3	a_7b_4	
a_8b_0	a_8b_1	a_8b_2	a_8b_3	a_8b_4	

4.14 卷积计算

- 蛮力法
- 向量 $a=(a_0,a_1,\ldots,a_{n-1})$ 和 $b=(b_0,b_1,\ldots,b_{n-1})$

$$A(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{n-1}$$

$$B(x)=b_0+b_1x+b_2x^2+\ldots+b_{n-1}x^{n-1}$$

$$C(x)=A(x)B(x)=a_0b_0+(a_0b_1+a_1b_0)\,x+\ldots+a_{n-1}b_{n-1}x^{2n-2}$$

- $C(x)$的系数向量就是$a*b$
- 卷积 $a*b$ 计算等价于多项式相乘
- 蛮力算法的时间：$O(n^2)$

计算2n-1次多项式$C(x)$

1. 选择值 $x_0, x_1, \ldots, x_{2n-1}$,
   求出 $A(x_j)$ 和 $B(x_j)$，$j = 0, 1, \ldots, 2n-1$
1. 对每个 j,计算$C(x_j)=A(x_j)B(x_j)$
2. 利用多项式插值方法，由$C(x)$ 在 $x = x_0, x_1, \ldots, x_{2n-1}$
   的值求出多项式 $C(x)$的系数

问题:

如何选择$x_0,x_1,\ldots,x_{2n-1}$的值?

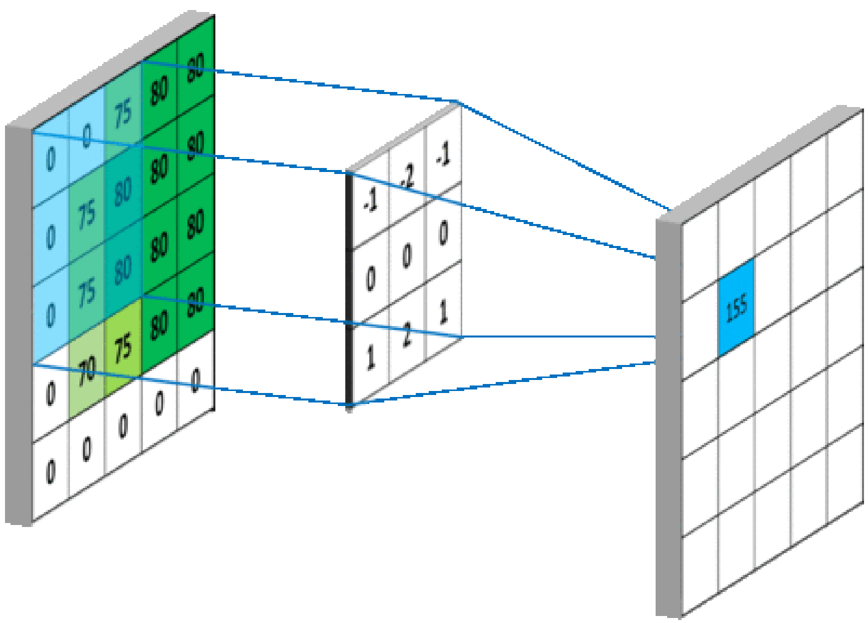
如何高效计算多项式插值的结果?

图片的高斯滤波

What We See

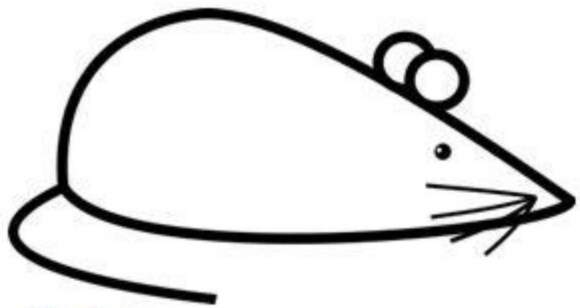
What Computers See

图片的高斯滤波

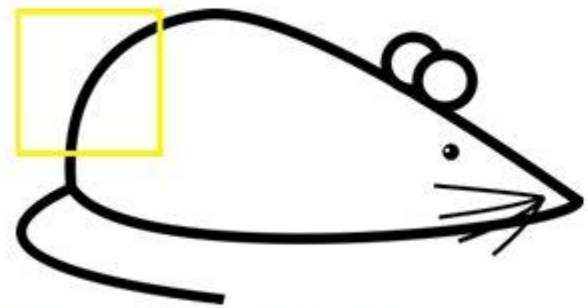


0 0 75 80 80
0 75 80 80 80
0 75 80 80 80
0 70 75 80 80
0 0 0 0 0
-1 -2 -1
0 0 0
1 2 1
155

神经元激活放电



Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

卷积核

- 卷积核Kernel也叫滤波器filter，代表图像的某种特征；也称为神经元。比如垂直边缘，水平边缘，颜色，纹理等等，这些所有神经元加起来就好比就是整张图像的特征提取器集合。
- 卷积核越深越能检测图像更高级别，更高层次，更复杂，更抽象，更泛化的特征。

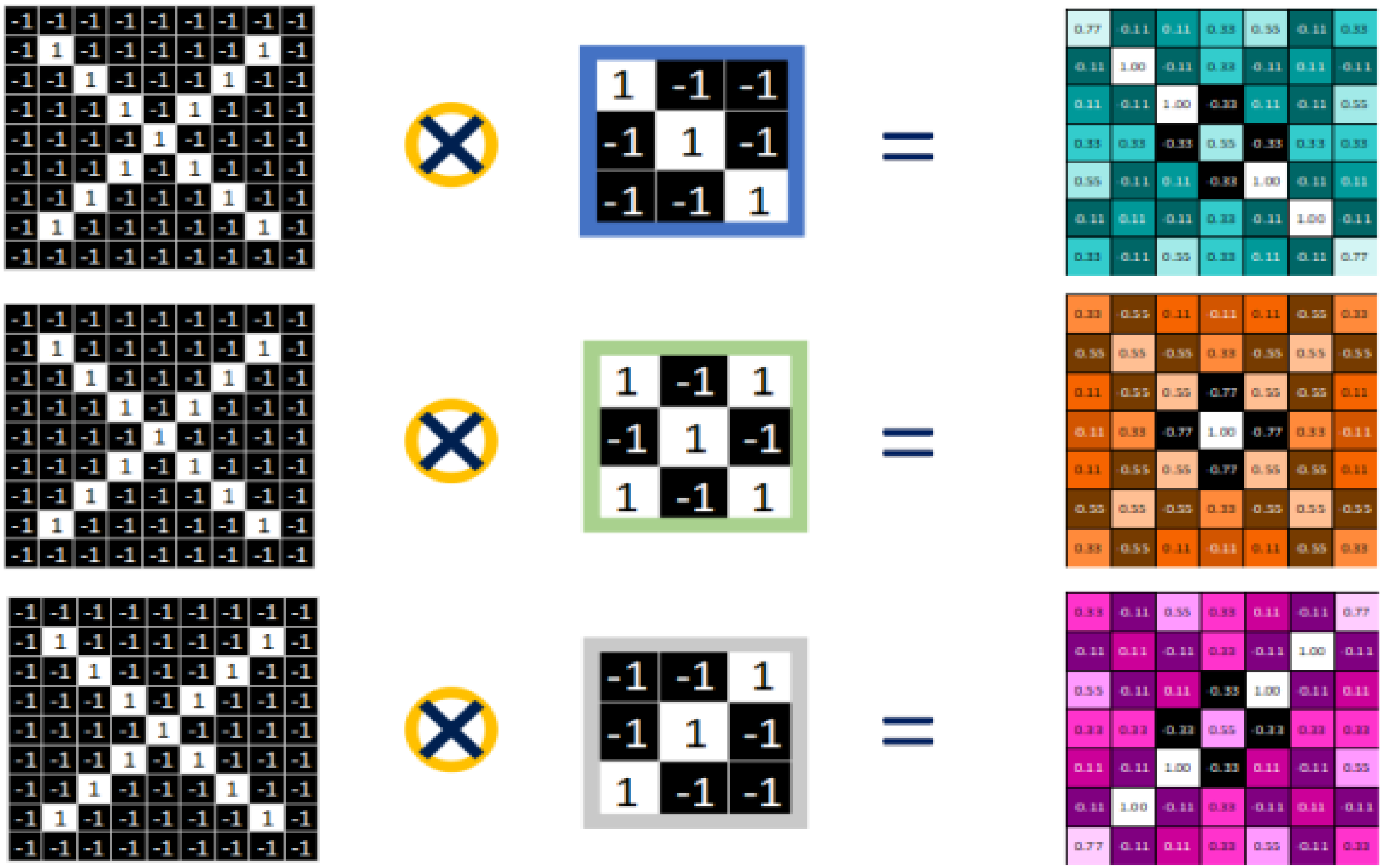
1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

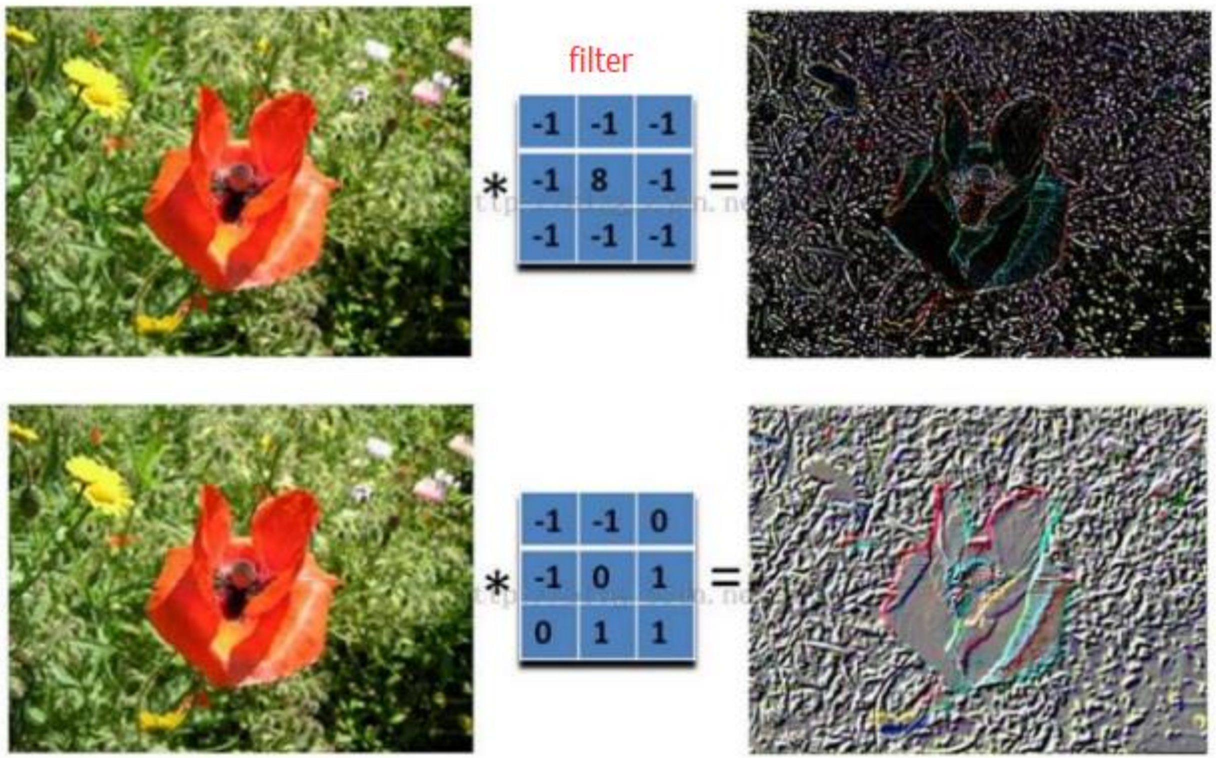
-1	-1	1
-1	1	-1
1	-1	-1

图像卷积运算

- 相似则输出一个明显变大的值，否则输出极小值。



图像卷积运算



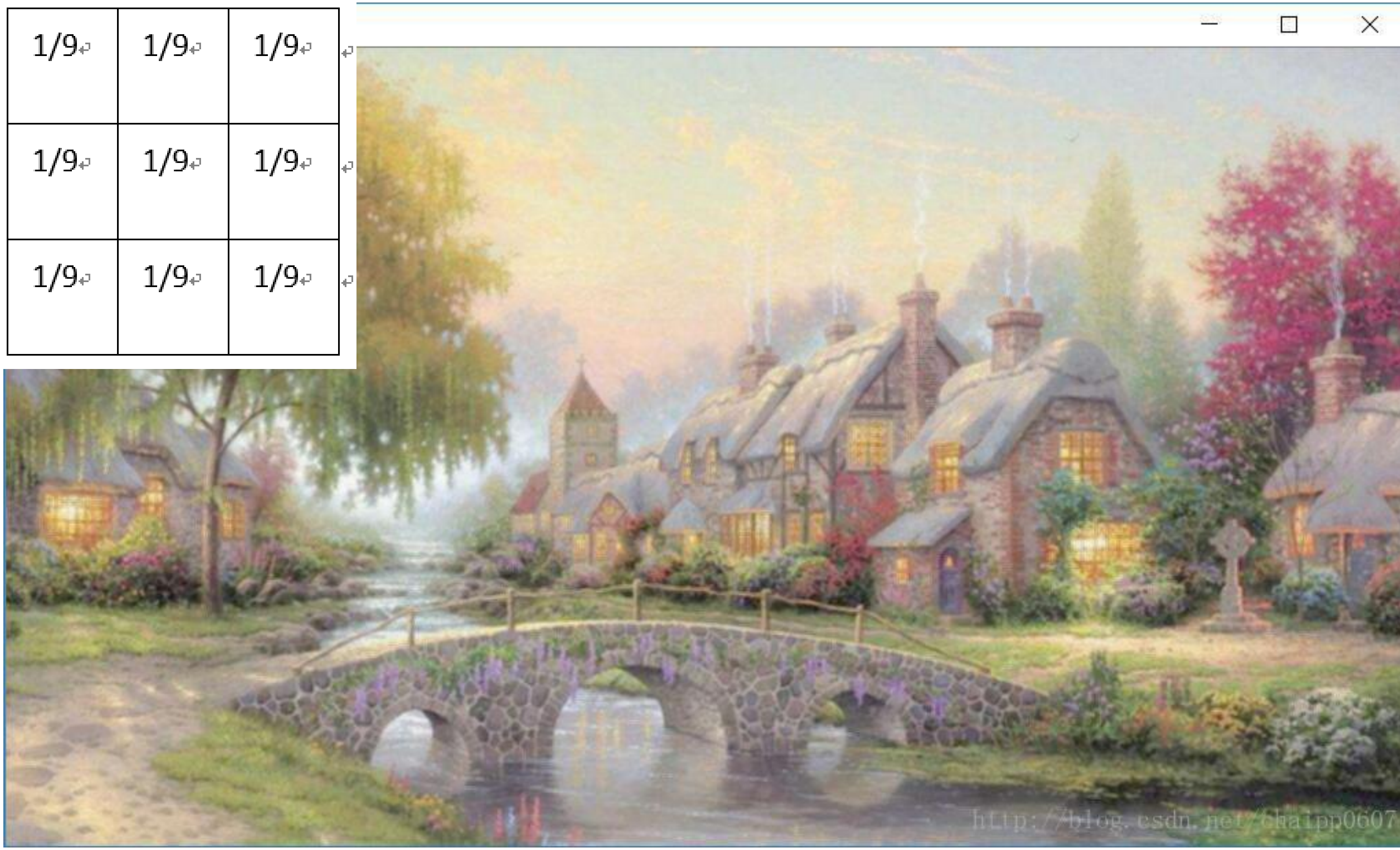
原图

不作处理

0	0	0
0	1	0
0	0	0

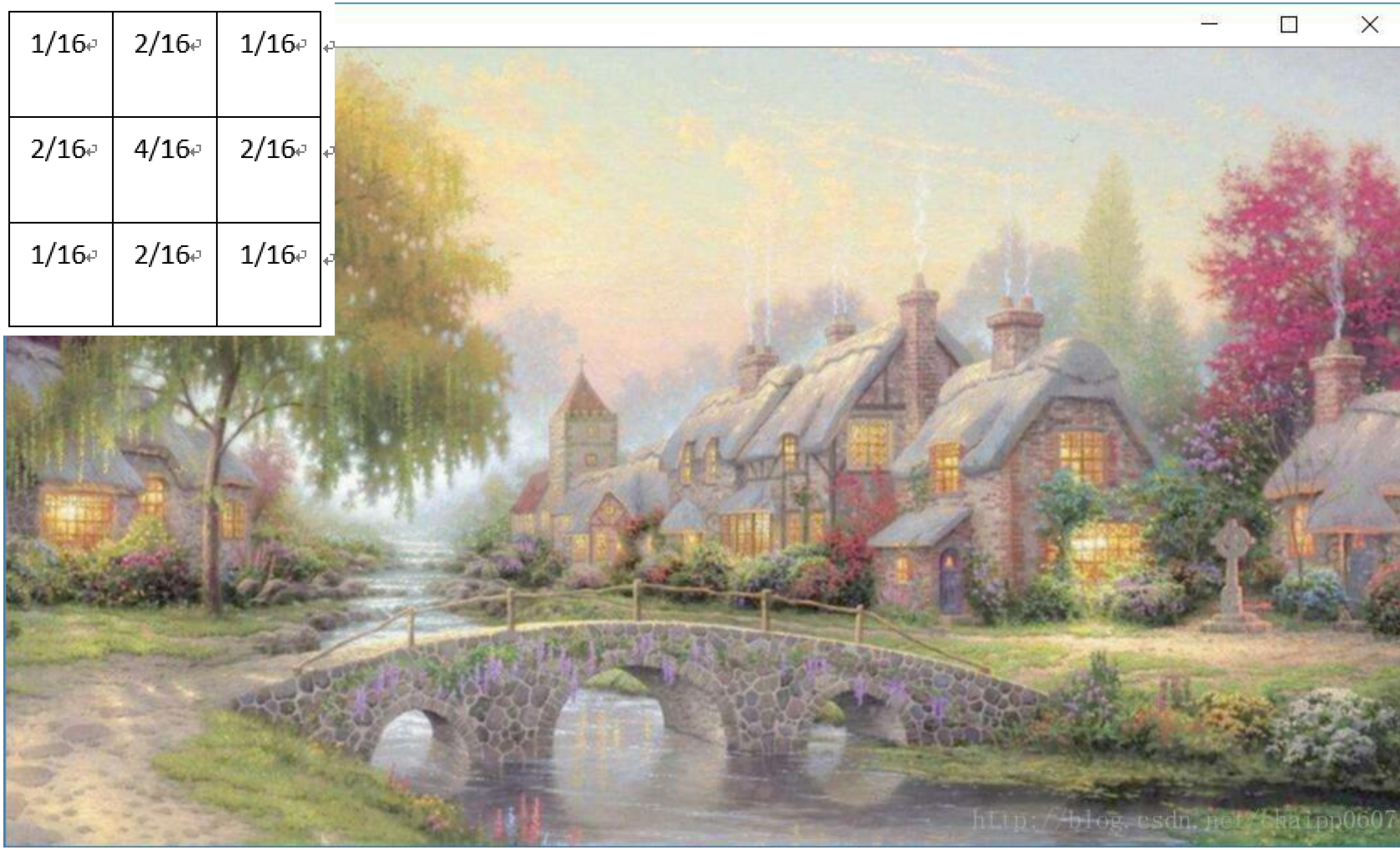
均值滤波

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



高斯平滑

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

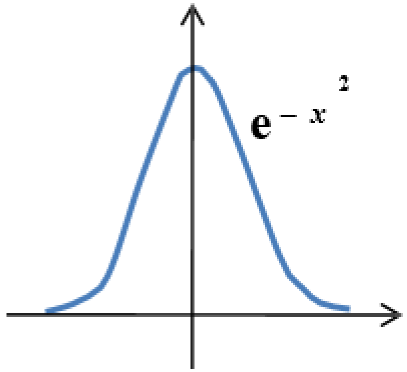


高斯滤波的权值函数

- 高斯滤波的权值函数为

$$w_s = \frac{1}{z} e^{-s^2}, \quad s = 0, \pm 1, \ldots, \pm k$$

$$w = (w_{-k}, \ldots, w_{-1}, w_0, w_1, \ldots, w_k)$$



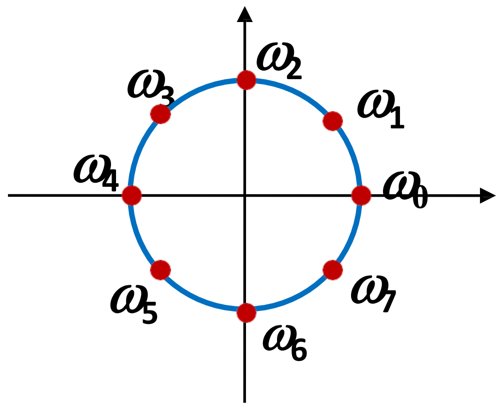
- 其中 z 用于归一化处理，使所有的权值之和为1
- 处理结果

$$a_i' = \sum_{s=-k}^{k} a_{i+s} w_s$$

$2n$个数的选择：1的$2n$次根

$$\omega_j = e^{\frac{2\pi j}{2n} i} = e^{\frac{\pi j}{n} i} = \cos\frac{\pi j}{n} + i\sin\frac{\pi j}{n}$$

$j = 0,1,\ldots,2n-1$， $i = -1$



n=4 的实例

$$\omega_0 = 1, \qquad \omega_1 = e^{\frac{\pi}{4} i} = \sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_2 = e^{\frac{\pi}{2} i} = i, \quad \omega_3 = e^{\frac{3\pi}{4} i} = -\sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_4 = e^{\pi i} = -1, \ \omega_5 = e^{\frac{5\pi}{4} i} = -\sqrt{2}/2 - \sqrt{2}/2 \cdot i,$$

$$\omega_6 = e^{\frac{3\pi}{2} i} = -i, \ \omega_7 = e^{\frac{7\pi}{4} i} = \sqrt{2}/2 - \sqrt{2/2} \cdot i$$

快速傅立叶变换FFT

1. 对 $x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$, 分别计算 $A(x)$，$B(x)$
2. 利用步1的结果对每个 $x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$，计算 $C(x)$，得到 $C(1)=d_0, C(\omega_1)=d_1, \ldots, C(\omega_{2n-1})=d_{2n-1}$
3. 构造多项式 $D(x) = d_0+d_1x+d_2x^2+\ldots+d_{2n-1}x^{2n-1}$
4. 对$x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$, 计算$D(x)$, $D(1), D(\omega_1), \ldots, D(\omega_{2n-1})$

可以证明：

$$\begin{aligned} D(1) &= 2nc_0 \\ D(\omega_1) &= 2nc_{2n-1} \\ &\ldots \\ D(\omega_{2n-1}) &= 2nc_1 \end{aligned} \quad \Rightarrow \quad \begin{aligned} c_0 &= D(1)/2n \\ c_{2n-1} &= D(\omega_1)/2n \\ &\ldots \\ c_1 &= D(\omega_{2n-1})/2n \end{aligned}$$

知道了$D(x)$的值，就能求$C(x)$的系数

算法的关键

- 令 $x = 1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$,
- 步1 对 $2n$个x值分别求值多项式$A(x)$, $B(x)$
- 步2 做 $2n$次乘法
- 步3 对$2n$个x值求值多项式 $D(x)$
- 关键：一个对所有的 x 快速多项式求值算法

小结

- 卷积的定义
  - 卷积与多项式乘法的关系
  - 卷积的应用——信号平滑处理
- 卷积计算
  - 蛮力算法$O(n^2)$
  - 快速傅立叶变换FFT算法
  - 确定x的取值：1 的 $2n$ 次根
  - 关键步骤：多项式对x求值

4.15 快速傅立叶变换：FFT算法

- 多项式求值算法
- 给定多项式：

$$A(x)=a_0+a_1x+\ldots+a_{n-1}x^{n-1}$$

- 设 x 为1 的 $2n$ 次方根，对所有的 x 计算 $A(x)$ 的值.
- 算法1：对每个 x 做下述运算：
- 依次计算每个项 $a_i x^i$， $i=1, \ldots, n-1$ 对 $a_i x^i$ $(i=0,1,\ldots,n-1)$求和.
- 蛮力算法的时间复杂度

$$T_1(n) = O(n^3)$$

改进的求值算法

- 算法2： 依次对 每个 x 做下述计算

$$A_1(x)=a_{n-1}$$

$$A_2(x)=a_{n-2}+xA_1(x)=a_{n-2}+a_{n-1}x$$

$$A_3(x)=a_{n-3}+xA_2(x)=a_{n-3}+a_{n-2}x+a_{n-1}x^2$$

$$\cdots$$

$$A_n(x)=a_0+x\,A_{n-1}(x)=A(x)$$

- 时间复杂度 $T_2(n)=O(n^2)$

偶系数与奇系数多项式

$n=4$

$A(x)=a_0+a_1x+a_2x^2+a_3x^3$

$A_{\text{even}}(x) = a_0+a_2x$

$A_{\text{odd}}(x) = a_1+a_3x$

$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2) = a_0+a_2x^2 + x(a_1+a_3x^2)$

分治多项式求值算法

- 一般公式（n为偶数）

$$A(x)= a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{n-1}$$

$$A_{\text{even}}(x)=a_0+a_2x+a_4x^2+\ldots+a_{n-2}x^{(n-2)/2}$$

$$A_{\text{odd}}(x)=a_1+a_3x+a_5x^2+\ldots+a_{n-1}x^{(n-2)/2}$$

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)$$

- x^2 也是1 的 $2n$ 次根
- 偶次数与奇次数多项式计算作为 $n/2$ 规模的子问题，然后利用子问题的解 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$ 得到 $A(x)$

分治求值算法设计

- 算法 3：

1. 计算 1 的所有的 $2n$ 次根

   $1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$

2. 分别计算 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$

3. 利用步2 的结果计算 $A(x)$

$$A(x) = A_{\text{even}}(x^2) + x\,A_{\text{odd}}(x^2)$$

- 注意：x^2不需要重新计算，所有根在单位圆间隔分布，隔一取一即可

分治求值算法分析

$$\boldsymbol{T(n) = T_1(n) + f(n)}$$

$f(n)=O(n)$ 是步 1 计算$2n$次根的时间

递归过程 $T_1(n) = 2T_1(n/2) + g(n)$

$T_1(1) = O(1)$,

$g(n) = O(n)$ 是对所有$2n$次根在步3组合解的时间

$T_1(n) = O(n\log n)$

$T(n) = O(n\log n)+O(n)=O(n\log n)$

FFT算法伪码

1. 求值$A(\omega_j)$和$B(\omega_j)$，$j=0,1,\ldots,2n-1$
2. 计算 $C(\omega_j)$，$j=0, 1, \ldots, 2n-1$
3. 构造多项式 $D(x)=C(\omega_0)+C(\omega_1)x+\ldots+C(\omega_{2n-1})x^{2n-1}$
4. 计算所有的 $D(\omega_j)$，$j=0,1,\ldots,2n-1$
5. 利用下式计算$C(x)$的系数 c_j，

   $D(\omega_j) = 2nc_{2n-j}$

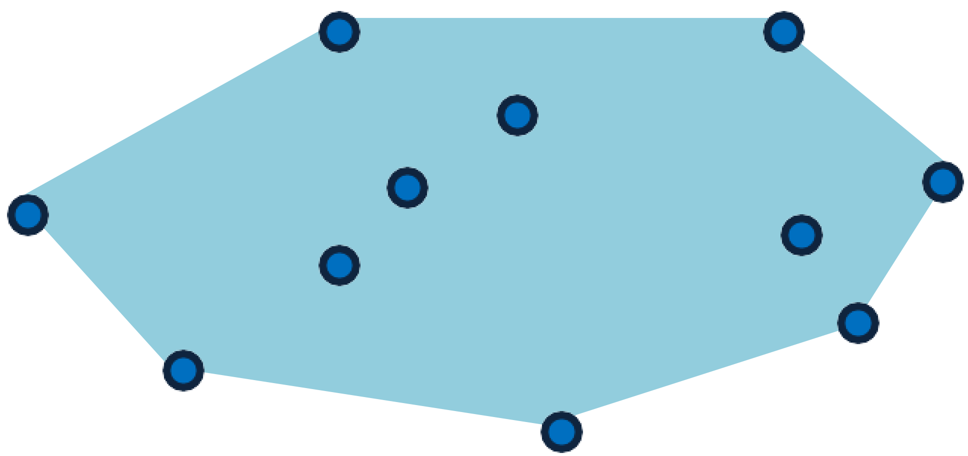
   $j = 0, 1, \ldots, 2n-1$

FFT算法分析

- 步1: 求值 $A(\omega_j)$ 和 $B(\omega_j)$ $\quad$ $\boldsymbol{O}$ **($\boldsymbol{n}$log$\boldsymbol{n}$)**
- 步2: 计算所有的 $C(\omega_j)$ $\quad$ $\boldsymbol{O}$ **($\boldsymbol{n}$)**
- 步3:
- 步4: 计算所有的 $D(\omega_j)$ $\quad$ $\boldsymbol{O}$ **($\boldsymbol{n}$log$\boldsymbol{n}$)**
- 步5: 计算所有的 c_j $\quad$ $\boldsymbol{O}$ **($\boldsymbol{n}$)**
- 算法时间为 $\quad$ $\boldsymbol{O}$ **($\boldsymbol{n}$log$\boldsymbol{n}$)**

4.16 平面点集的凸包

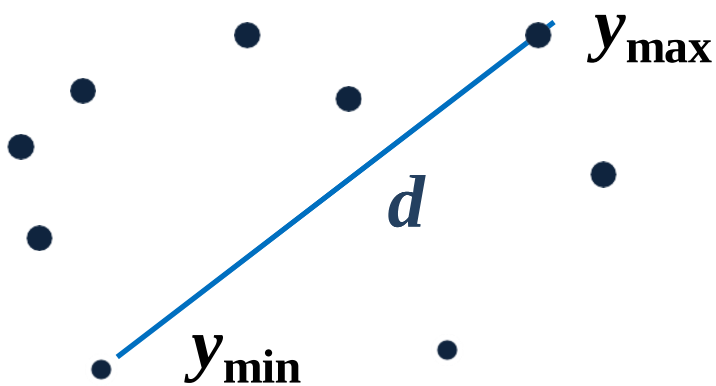
- 问题（平面点集的凸包）
- 给定大量离散点的集合Q，求一个最小的凸多边形，使得Q中的点在该多边形内或者边上



- 应用背景
- 图形处理中用于形状识别：字形识别、碰撞检测等

分治算法

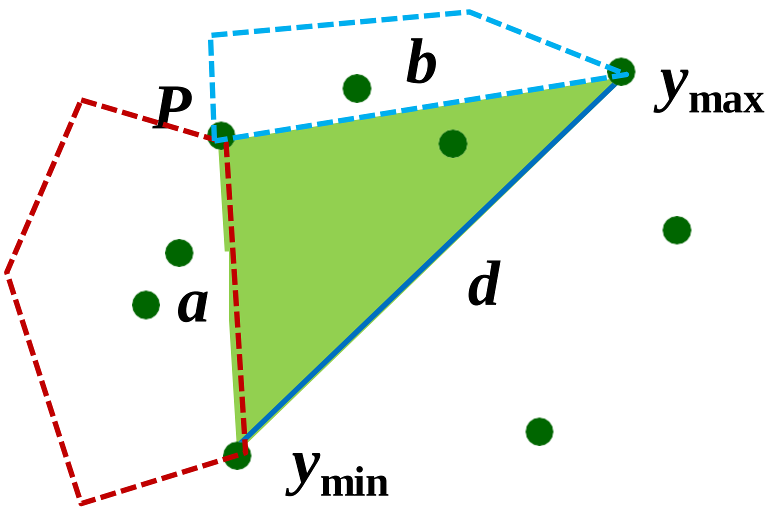
1. 以连接最大纵坐标点 $y_{\max}$ 和最小纵坐标点 $y_{\min}$ 的线段 $d=\{y_{\max},y_{\min}\}$划分L 为左点集 L_{left} 和右点集 L_{right}



2. Deal (L_{left})； Deal (L_{right})

Deal (L_{left})

- 考虑 L_{left}： 确定距d 最远的点P
  - 在三角形内的点，删除
  - a 外的点与 a 构成 L_{left} 的子问题
  - b 外的点与 b 构成 L_{left} 的子问题



伪码

Deal (L_{left})

1. 以 d 和距离 d 最远点 P 构成三角形，P加入凸包，另外两条边分别记作 a 和 b
2. 检查 L_{left} 中其他点是否在三角形内；在则从 L中删除；否则根据在 a 或 b 边的外侧划分在两个子问题中
3. Deal (a)
4. Deal (b)

算法分析

- 初始用d 划分　$O(n)$
- Deal 递归调用　$W(n)$
  - 找凸包顶点 P　$O(n)$
  - 根据点的位置划分子问题 $O(n)$

  $W(n) = W(n\text{-}1) + O(n)$

  $W(3) = O(1)$
- 最坏情况为$O(n^2)$
- $T(n) = O(n) + W(n) = O(n^2)$
- Graham扫描算法 $O(n\log n)$