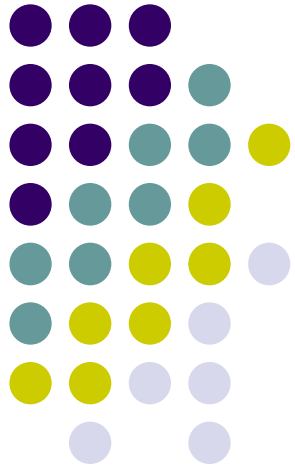
数据结构习题 第 4 章



4-1

- 二维数组A中，每个元素的长度为3个字节，行下标从0到9，列下标从0到11，则连续存放该数组至少需要多少个字节？
- 答案：10 * 12 * 3 = 360

4-2

- 二维数组A有4行8列，下标从0开始，存储A的起始地址为2000，每个元素用相邻的4个字节存储，试计算：
  - 存储整个数组一共需要多少个字节。
  - 数组A的最后一个元素的起始地址。
  - 按行存储时，A[2][4]的起始地址。
  - 按列存储时，A[3][2]的起始地址。

分析

2000 ← 4 →

A(0, 0)	A(0, 1)	A(0, 2)	A(0, 3)	A(0, 4)	A(0, 5)	A(0, 6)	A(0, 7)
A(1, 0)	A(1, 1)	A(1, 2)	A(1, 3)	A(1, 4)	A(1, 5)	A(1, 6)	A(1, 7)
A(2, 0)	A(2, 1)	A(2, 2)	A(2, 3)	A(2, 4)	A(2, 5)	A(2, 6)	A(2, 7)
A(3, 0)	A(3, 1)	A(3, 2)	A(3, 3)	A(3, 4)	A(3, 5)	A(3, 6)	A(3, 7)

参考答案

- 4*8*4=128
- 2000+(3*8+7)*4=2124
- 2000+(2*8+4)*4=2080
- 2000+(2*4+3)*4=2044

补充

- 给出如下稀疏矩阵的三元组表表示：

$$A = \begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$

参考答案

A[0]	0	0	50
A[1]	1	0	10
A[2]	1	2	20
A[3]	3	0	-30
A[4]	3	2	-60
A[5]	3	3	5

作业4-3

- 设稀疏矩阵M_{m*n}中有t个非零元素，用三元组表的方式存储. 请设计一个算法，计算矩阵M的转置矩阵N，且算法的时间复杂性为O(n+t). 注意，书中给出的算法的复杂性为O(n*t)

$$A=\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \qquad A'=\begin{pmatrix} 50 & 10 & 0 & -30 \\ 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & -60 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

A[0]	0	0	50
A[1]	1	0	10
A[2]	1	2	20
A[3]	3	0	-30
A[4]	3	2	-60
A[5]	3	3	5

B[0]	0	0	50
B[1]	0	1	10
B[2]	0	3	-30
B[3]	2	1	20
B[4]	2	3	-60
B[5]	3	3	5

算法的关键是求出A中元素在B中的位置

```
num = 0.
FOR i=0 TO Cols(A) DO
    FOR j=0 TO t DO (
    IF col(A[j])=i Then (
        row(B[num] ← i.
        col(B[num]) ← row(A[j]).
        value(B[num]) ← Value(A[j]).
        num ← num+1.
    )
)
```

i=0 j=0

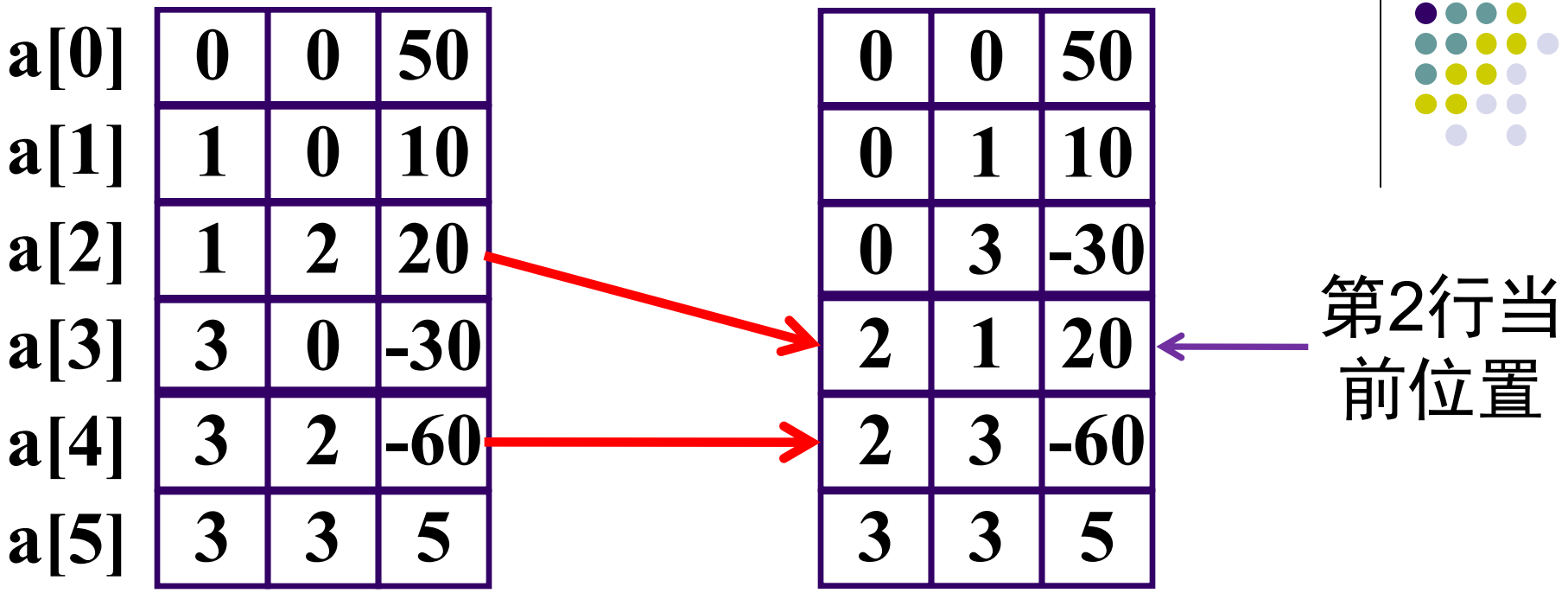
i=1 j=0

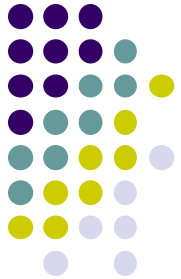
i=2 j=0

i=3 j=0

a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10
0	3	-30
2	1	20
2	3	-60
3	3	5


a[0]
0 0 50
a[1]
1 0 10
a[2]
1 2 20
a[3]
3 0 -30
a[4]
3 2 -60
a[5]
3 3 5
0 0 50
0 1 10
0 3 -30
2 1 20
2 3 -60
3 3 5
第2行当前位置



算法TRANSPOSE(A. B)

TP1[初始化]

/*声明A的转置矩阵B，使得B的行数等于A的列数，B的列数等于A的行数，B中非0元素的个数等于A中非0元素的个数*/

n←Rows(B)←Cols(A).

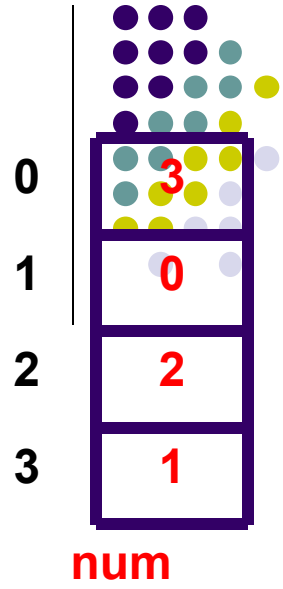
Cols (B)←Rows(A).

t←Count(B)← Count(A).

TP2.1[定义数组num存储A中每列非零元素个数]

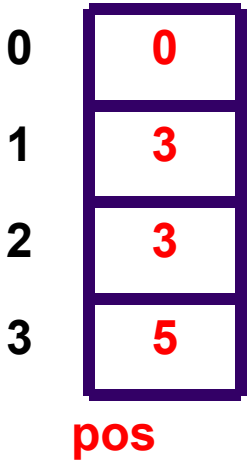
```
FOR i ← 0 TO n-1 DO
    num[i]←0.
FOR i ← 0 TO t-1 DO (
    j ←col(A [i]).
    num[j]←num[j]+1.
)
```

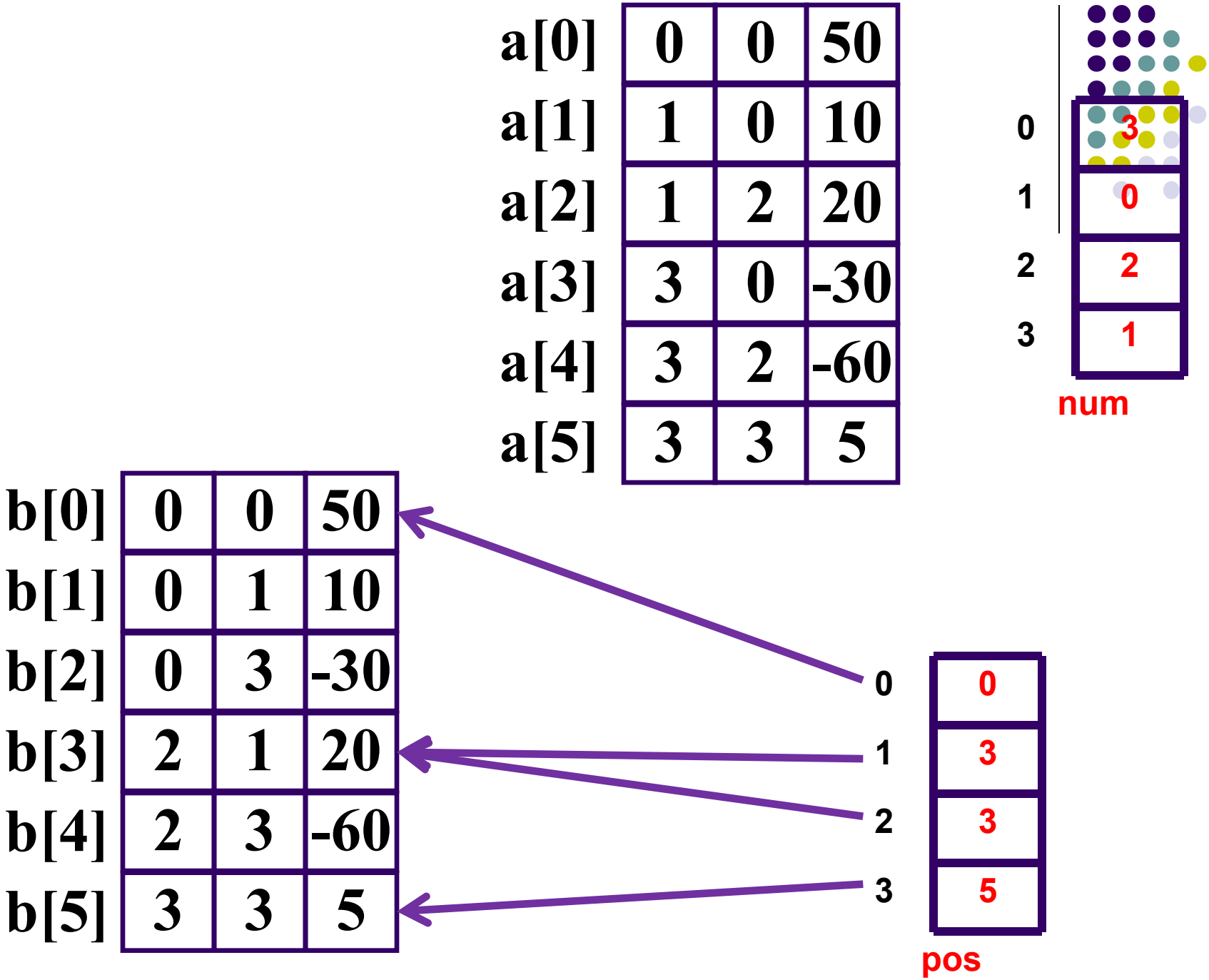
a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5



TP2.2[定义数组pos存储A中每列第一个非零元素在B的三元组表中的位置]

```
pos[0] ←0
FOR i ← 1 TO  n-1 DO (
    pos[i] ←pos[i-1]+num[i-1]
)
```




a[0] 0 0 50
a[1] 1 0 10
a[2] 1 2 20
a[3] 3 0 -30
a[4] 3 2 -60
a[5] 3 3 5
0
1
2
3
3
0
2
1
num
b[0] 0 0 50
b[1] 0 1 10
b[2] 0 3 -30
b[3] 2 1 20
b[4] 2 3 -60
b[5] 3 3 5
0
1
2
3
0
3
3
5
pos

TP3[处理三元组表]

FOR i ← 0 TO t-1 **DO** (

p←col(A[i]).

k←pos[p].

col(B[k]) ← row(A[i]).

row(B[k]) ←col(A[i]).

val(B[k]) ← val(A[i]).

pos[p]← pos[p]+1.

).

0
3
3
5

pos

a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50

TP3[处理三元组表]

FOR i ← 0 TO t-1 **DO** (

p←col(A[i]).

k←pos[p].

col(B[k]) ← row(A[i]).

row(B[k]) ←col(A[i]).

val(B[k]) ← val(A[i]).

pos[p]← pos[p]+1.

).

1
3
3
5

pos

a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10

TP3[处理三元组表]

```
FOR i ← 0 TO t-1 DO (
    p←col(A[i]).
    k←pos[p].
    col( B[k]) ← row(A[i]).
    row(B[k]) ←col(A[i]).
    val(B[k])  ← val(A[i]).
    pos[p]← pos[p]+1.
).
```

2
3
3
5

pos

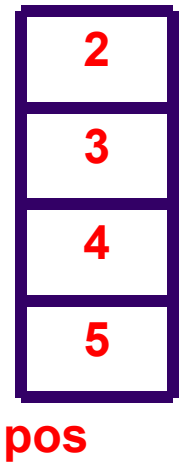
a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10

2	1	20

TP3[处理三元组表]

```
FOR i ← 0 TO t-1 DO (
    p←col(A[i]).
    k←pos[p].
    col( B[k]) ← row(A[i]).
    row(B[k]) ←col(A[i]).
    val(B[k])  ← val(A[i]).
    pos[p]← pos[p]+1.
).
```



a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10
0	3	-30
2	1	20

TP3[处理三元组表]

```
FOR i ← 0 TO t-1 DO (
    p←col(A[i]).
    k←pos[p].
    col( B[k]) ← row(A[i]).
    row(B[k]) ←col(A[i]).
    val(B[k])  ← val(A[i]).
    pos[p]← pos[p]+1.
).
```

3
3
4
5

pos

a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10
0	3	-30
2	1	20
2	3	-60

TP3[处理三元组表]

```
FOR i ← 0 TO t-1 DO (
    p←col(A[i]).
    k←pos[p].
    col( B[k]) ← row(A[i]).
    row(B[k]) ←col(A[i]).
    val(B[k])  ← val(A[i]).
    pos[p]← pos[p]+1.
).
```

pos
3
3
5
5

a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10
0	3	-30
2	1	20
2	3	-60
3	3	5

TP3[处理三元组表]

```
FOR i ← 0 TO t-1 DO (
    p←col(A[i]).
    k←pos[p].
    col( B[k]) ← row(A[i]).
    row(B[k]) ←col(A[i]).
    val(B[k])  ← val(A[i]).
    pos[p]← pos[p]+1.
).
```

pos
3
3
5
6

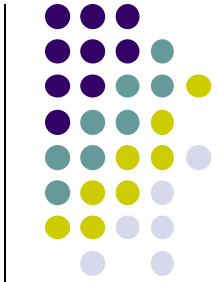
a[0]	0	0	50
a[1]	1	0	10
a[2]	1	2	20
a[3]	3	0	-30
a[4]	3	2	-60
a[5]	3	3	5

0	0	50
0	1	10
0	3	-30
2	1	20
2	3	-60
3	3	5

4-4

- 试编写一个模式匹配算法，匹配过程为：先匹配模式的首尾字符，若匹配成功，调用成员函数Substr（取子串）来检查模式的首尾之间的字符是否与目标的相应字符相匹配，若匹配不成功；则进行下一次匹配。

参考答案



```
int indexOf(string s, string p)
{
    if ( p==“” ) return 0;
    if ( s==“” ) return -1;
    int slen=s.size(), plen=p.size();
    for ( int i = 0; slen-i >= plen; i++) {
        if (s[i]!=p[0] || s[i+plen-1]!=p[plen-1]) continue;
        string ss=s.substr(i+1,plen-1);
        string pp=p.substr(1,plen-1);
        if (ss==pp) return i;
    }
    return -1;
}
```

4-5

- 设模式P="abcabcacabca",请给出该模式的失败函数。

4-6

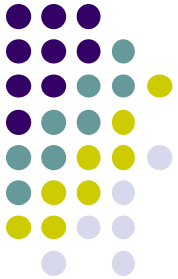
- 已知主串s="abcaabbabcabaacbacba"，模式串pat="abcabaa"，写出模式串的f值，并由此画出KMP算法匹配的全过程。

参考答案

- f值的计算：pat=“abcabaa”

j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0

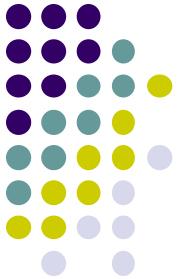
- KMP算法的匹配过程
- s=“abcaabbabcabaacbacba”
- pat=“abcabaa”



s=4; p=4
s=4; p=f(3)+1=1

a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
a	b	c	a	b	a	a													

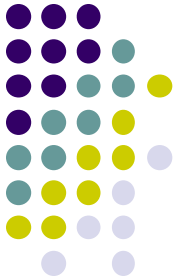
j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0



s=4; p=1
s=4; p=f(0)+1=0

a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
			a	b	c	a	b	a	a										

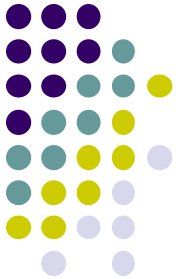
j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0



s=6; p=2
s=6; p=f(1)+1=0

a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
				a	b	c	a	b	a	a									

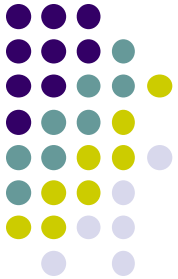
j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0



s=6; p=0
s=7; p=0

a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
						a	b	c	a	b	a	a							

j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0



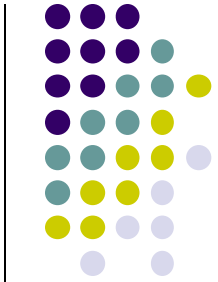
s=14; p=7

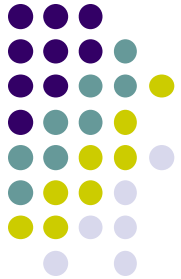
a	b	c	a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
							a	b	c	a	b	a	a						

返回s-m=7

j	0	1	2	3	4	5	6
f(j)	-1	-1	-1	0	1	0	0

- abcaabbabcabaacbacba
- abcabaa
- s=4 p=4
- abcaabbabcabaacbacba
- abcabaa
- s=4 p=f(3)+1=1
- abcaabbabcabaacbacba
- abcabaa
- s=4 p=f(0)+1=0 s=6 p=2
- abcaabbabcabaacbacba
- abcabaa
- s=6 p=f(1)+1=0 s=7
- abcaabbabcabaacbacba
- abcabaa





THE END