通过先前的介绍可知，实现非精确匹配的前提是计算 Occ(a,i)。
若考虑 D(i) 的需求，还需要计算 Occ'(a,i)。
若将 Occ(a,i) 和 Occ(a,i) 同时置于内存中以便查寻，则需要大量的内存空间，大规模的序列比对来说是难以实现的。
此时就需要将 Occ 压缩

下面将以实际例子说明压缩过程

BWT-String = CTAAT GC G GGATATAC

		A	C	G	T	
C	0	0	1	0	0	header
T	1	0	1	0	1	
A	2	1	1	0	1	header
A	3	2	1	0	1	block0
T	4	2	1	0	2	
G	5	2	1	1	2	
C	6	2	2	1	2	
G	7	2	2	2	2	
G	8	2	2	3	2	header
G	9	2	2	4	2	
A	10	3	2	4	2	header
T	11	3	2	4	3	block1
A	12	4	2	4	3	
T	13	4	2	4	4	
A	14	5	2	4	4	
C	15	5	3	4	4	

将整个 Occ 分为两个 block，即 block0 和 block1。
每个 block 分别对应一个 code。
每个 code 分为两部分，即 head 和 body

block	head: A	C	G	T	body
0	0	1	0	0	T A A T G C G
1	2	2	3	2	G A T A T A C

① 每个 block 的第一个 entry，作为 head
② 每个 block 对应的 BWT-string 除去第一个符号作为 body

考虑一下数据压缩的长度.

|BWT-string| = 16　则 O(a,i)max = 16　需要 5 bit

编码前需要 5 × 16 × 4 = 320 bits

BWT-string 总共有4种字符　需要 2 bit

编码后需要 (4×5+7×2)×2 = 68 bits

68 / 320 = 0.2185

解码过程

例如要获 Occ(A,6) 的值，i=6 属于 block 0

block	A	C	G	T	body
0	0	1	0	0	T A A T G C G

Number of "A"s in header = 0　Number of "A"s in body = 2

Occ(A,6) = "A"s in header + "A"s in body = 2

考虑到解码过程需要遍历字符串，为了减少解码过程中的遍历长度，取每个 block 中间的 entry 作为 head

block	head A	C	G	T	body
0	2	1	0	1	T A A T G C G
1	3	2	4	3	G A T A T A C

① 每个 block 的第四个 entry，作为 head

② 每个 block 对应的 BWT-string 除去第一个符号作为 body

解码过程

例如要获 Occ(A,6) 的值，i=6 属于 block 0，并且位于 head 之后

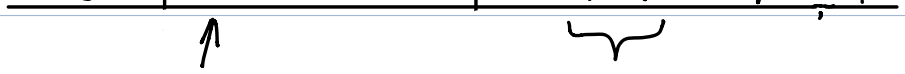
block	A	C	G	T	body
0	2	1	0	1	T A A T G C G

Number of "A"s in header = 2 Number of "A"s in body = 0

Occ(A,6) = "A"s in header + "A"s in body = 2

例如要获 Occ(A,1) 的值，i=1 属于 block 0，并且位于 head 之前

block	A	C	G	T	body
0	2	1	0	1	T A A T G C G



Number of "A"s in header = 2 Number of "A"s in body = 2

Occ(A,6) = "A"s in header - "A"s in body = 0