7.1.4 Hierarchycal Bitmap–based RLE Encoding

The most effective approach for the RLE encoding is the hierarchycal bitmap encoding. The basic idea is to create a bitmap where is one bit reserved for each character in a message. We set a corresponding bit to `1` if a character belongs to a run, `0` otherwise. Note that the first character of a run has the first bit set to `0` to let the decoder know what character is repeated. This is illustrated on bottom of table 7.1. We can reconstruct the original message from a knowledge of B and all characters

hierarchy B_1 (1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hierarchy B_2 (0)	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
hierarchy B_3 (1)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	0	0

bitmap B	0	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0
message M	a	b	a	a	a	a	a	a	c	d	a	a	a	a	c	a

Table 7.1: Illustration of hierarchycal RLE encoding.

where bits are set to `0`. The size of B can be significantly decreased by hierarchycal encoding as there are many of bit runs present in it. In upper part of table 7.1 is shown the hierarchycal decomposition, which is done in the way that the original B is downsampled by a factor of two to B_3, so that a bit in B_3 assigned to two bits in B is set to `1` when both the B bits are set to `1`, zero otherwise. This is done because if we know B_3, we need to save only bits from B that have their respective bit in B_3 set to zero. It is likely that zero bits will prevail in B_3 after this transformation, so that we can use the similar decomposition of B_3 to B_2 but for zeros in B_3. We can go on subsampling the higher hierarchies up to the decomposition to lower hierarchies still decreases the total bits for representing B. In our case we stop after B_2. Then the output of coder is a bit sequence from the lowest hierarchy up to B_3 and all characters for which B_3=`0`:

$$M = \text{“}\texttt{abaaaaaacdaaaaca}\text{”} \quad \Longrightarrow \quad M_{\text{RLE}} = \text{“}\texttt{(10101110) abaacdaaca}\text{”} \qquad (7.3)$$

Note that the information about runs in M is encoded only to 8–bits, what is usually a size of one character within the message. The bits written to output are shown in

brackets in 7.3. The results achieved by this method depends highly on basic heuristic what decides whether it is better to suppress `0`'s or `1`'s from a subsequent hierarchy. Also different decimation by factor of 4 or 8 can be used in the decomposition in order to improve compression.

7.2 Burrows–Wheeler Transform

This method based on block–sorting was first invented by D.J. Wheeler in 1983, but was first published in 1994 [1]. The principle of the Burrows–Wheeler transform (BWT) is that a block of data to be transformed is sorted lexicographically to put similar contexts present within a data block together to be better encoded by entropy coders. Thus BWT is not a compression method itself but leads to better performance for further applied compression. The essential property of BWT is that the data can be reconstructed without any losses to its original form.

7.2.1 Forward Burrows–Wheeler Transform

To demonstrate the forward BWT itself we use encoding of a message $M =$"`abaca`". First, one has to do a number of string rotations that equals the total characters within the message (table 7.2a). The I_f is a position of the first character of the rotated

a) rotations of the input message

I_f	rotated M
0	`abaca`
1	`bacaa`
2	`acaab`
3	`caaba`
4	`aabac`

b) sorted M and BWT output

I	I_f	rotated, sorted M	output M_{BWT}
0	4	`aabac`	`c`
1	0	`abaca`	`a`
2	2	`acaab`	`b`
3	1	`bacaa`	`a`
4	3	`caaba`	`a`

Table 7.2: Illustration of the forward Burrows–Wheeler transform.

message in the original string M. Then the rotated strings are lexicographically sorted (table 7.2a) in the way that we keep an information of the original index I