COMP9319 Web Data Compression and Search

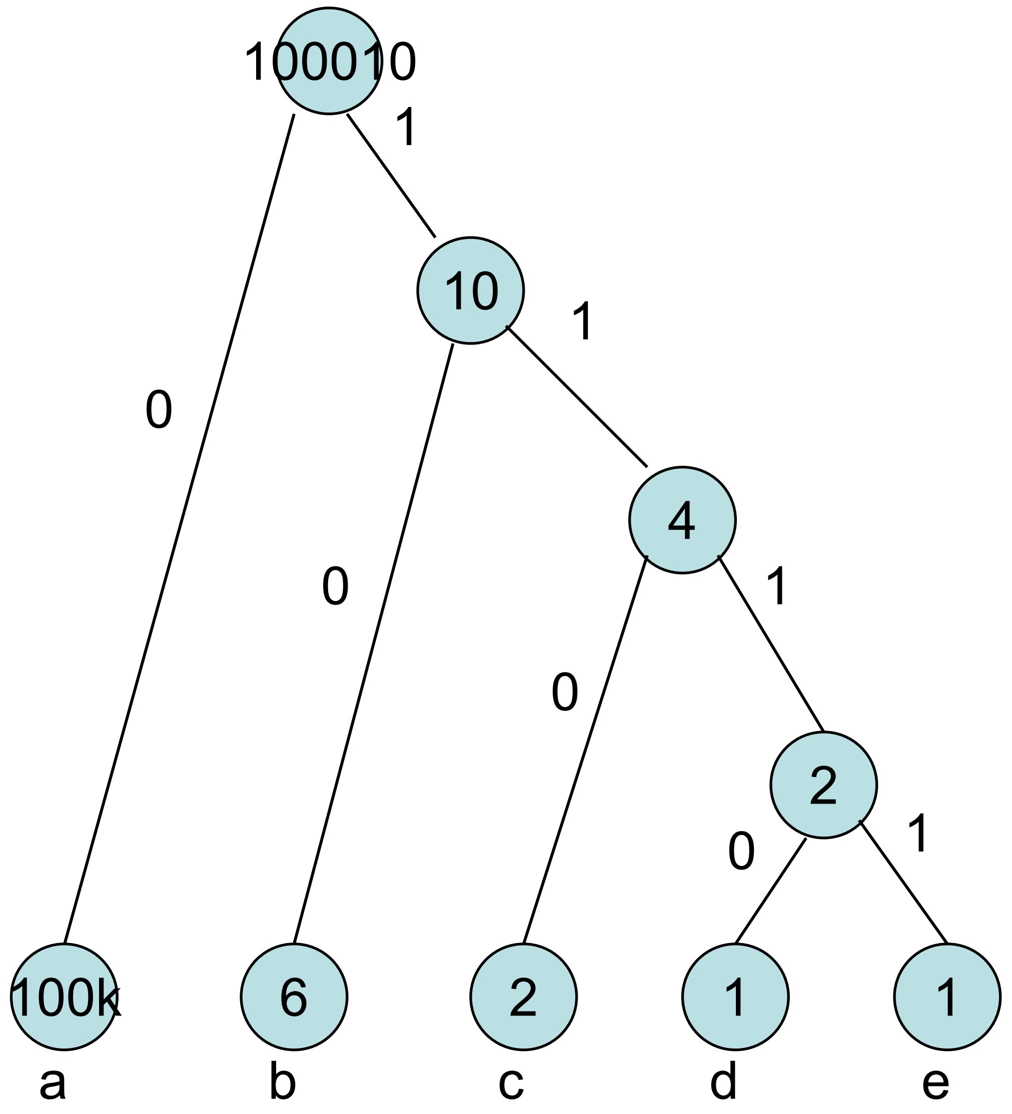
Lecture 2: Adaptive Huffman, BWT

Course schedule

- Data compression
- Search
- Data compression + Search
- Web data compression + Search
- Optional topics

Huffman coding

S	Freq	Huffman
a	100000	0
b	6	10
c	2	110
d	1	1110
e	1	1111



Huffman not optimal

H = 0.9999 log 1.0001 + 0.00006 log 16668.333 + … + 1/100010 log 100010

≈ **0.00**

L = (100000*1 + …)/100010

≈ **1**

Problems of Huffman coding

- Huffman codes have an integral # of bits.
  - E.g., log (3) = 1.585 while Huffman may need 2 bits
- Noticeable non-optimality when prob of a symbol is high.

=> Arithmetic coding

Problems of Static coding

- Need statistics & static: e.g., single pass over the data just to collect stat & stat unchanged during encoding
- To decode, the stat table need to be transmitted. Table size can be significant for small msg.

=> Adaptive compression e.g., adaptive huffman

Adaptive compression

Encoder

```
Initialize the model
Repeat for each input char
 (
   Encode char
   Update the model
 )
```

Decoder

```
Initialize the model
Repeat for each input char
 (
   Decode char
   Update the model
 )
```

Make sure both sides have the same Initialize & update model algorithms.

Adaptive Huffman Coding (dummy)

Encoder

```
Reset the stat
Repeat for each input char
 (
   Encode char
   Update the stat
   Rebuild huffman tree
 )
```

Decoder

```
Reset the stat
Repeat for each input char
 (
   Decode char
   Update the stat
   Rebuild huffman tree
 )
```

Adaptive Huffman Coding (dummy)

Encoder

```
Reset the stat
Repeat for each input char
 (
   Encode char
   Update the stat
   Rebuild huffman tree
 )
```

Decoder

```
Reset the stat
Repeat for each input char
 (
   Decode char
   Update the stat
   Rebuild huffman tree
 )
```

This works but too slow!

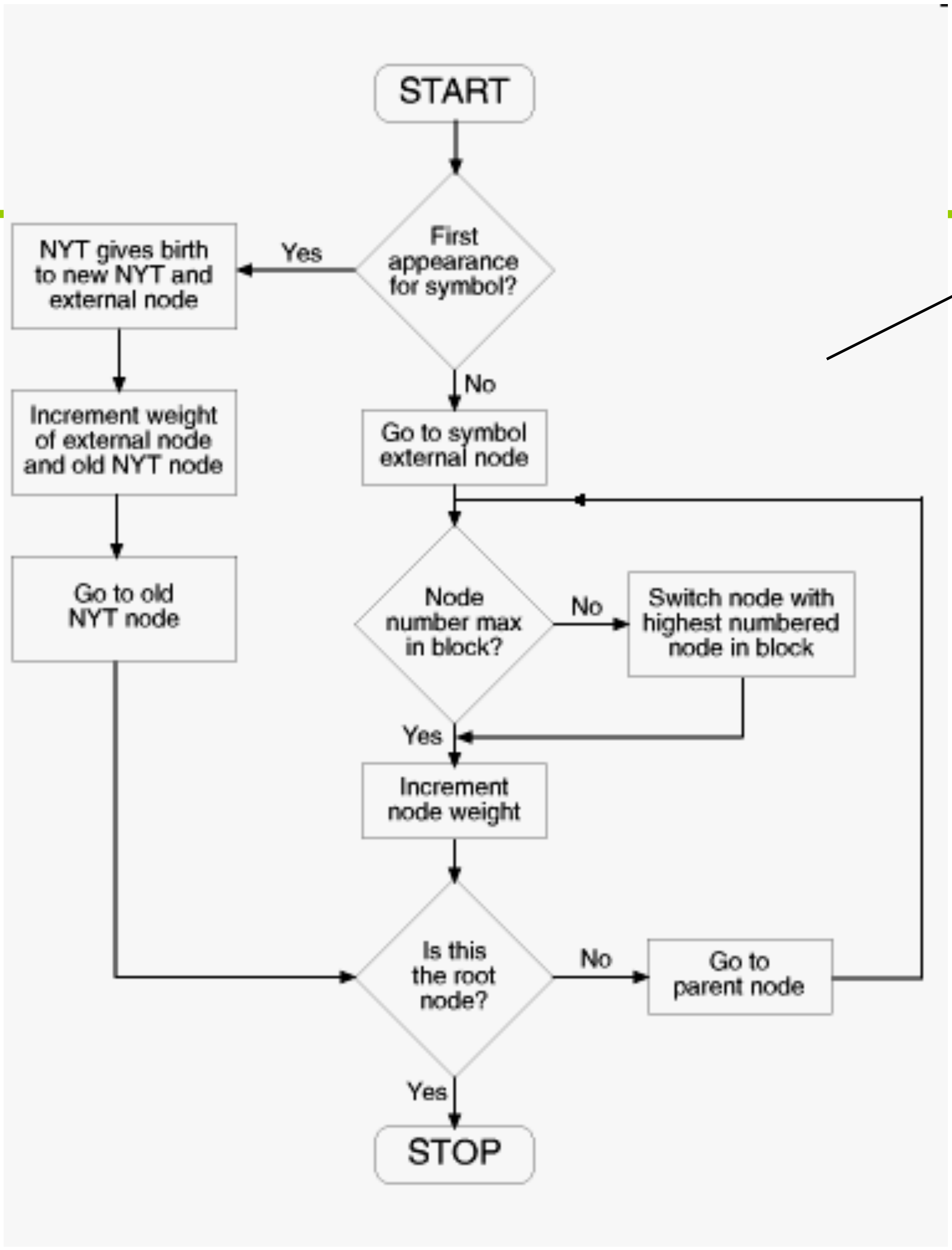
Adaptive Huffman (Algorithm outline)

1. If current symbol is NYT, add two child nodes to NYT node. One will be a new NYT node the other is a leaf node for our symbol. Increase weight for the new leaf node and the old NYT and go to step 4. If not, go to symbol's leaf node.

2. If this node does not have the highest number in a block, swap it with the node having the highest number

3. Increase weight for current node

4. If this is not the root node go to parent node then go to step 2. If this is the root, end.


START
First appearance for symbol?
Yes
NYT gives birth to new NYT and external node
Increment weight of external node and old NYT node
Go to old NYT node
No
Go to symbol external node
Node number max in block?
No
Switch node with highest numbered node in block
Yes
Increment node weight
Is this the root node?
No
Go to parent node
Yes
STOP

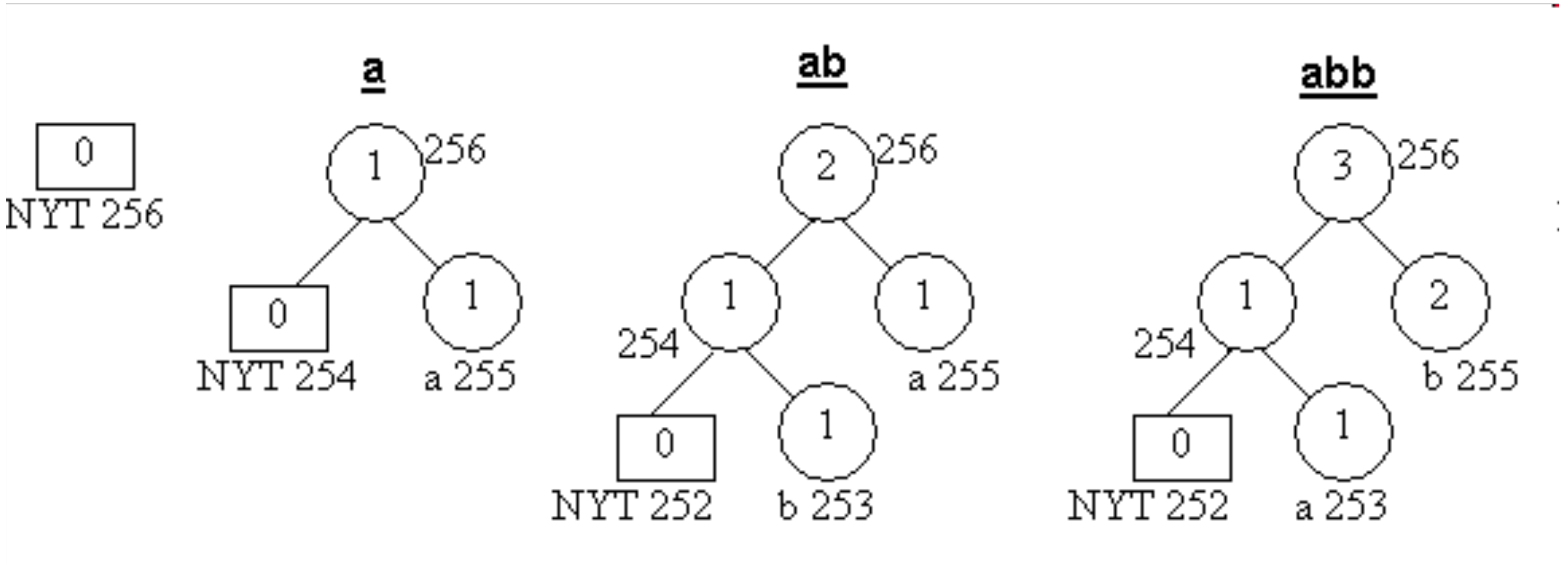
The update procedure from Introduction to Data Compression by by Sayood Khalid

Also, Wikipedia provides a good summary, example and explanation (i.e., http://en.wikipedia.org/wiki/Adaptive_Huffman_coding)

Adaptive Huffman

abbbbba: 01100001011000100110001001100010011000100110001001100001

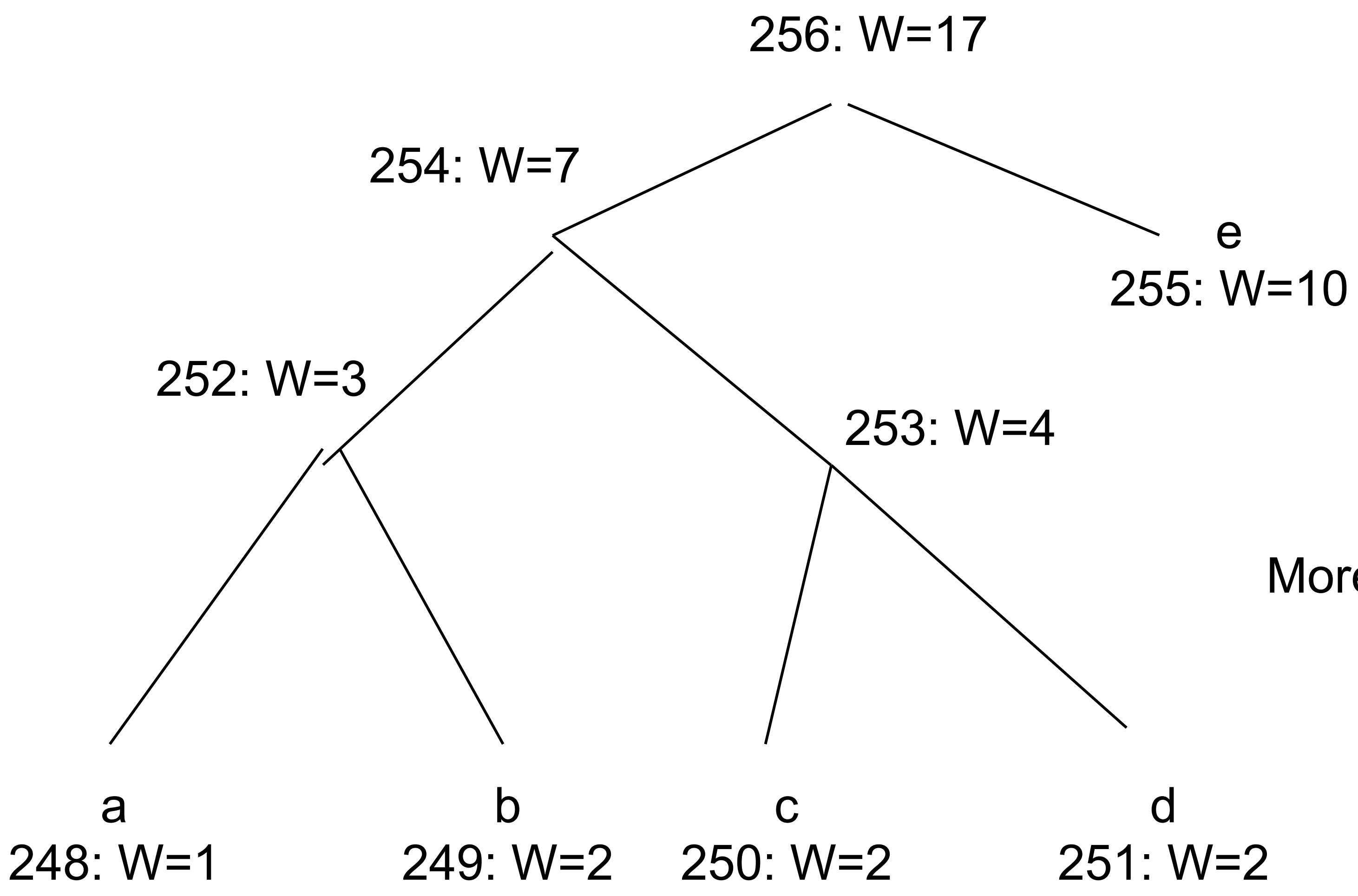
abbbbba: 011000010011000100111101



a: 01100001
b: 01100010

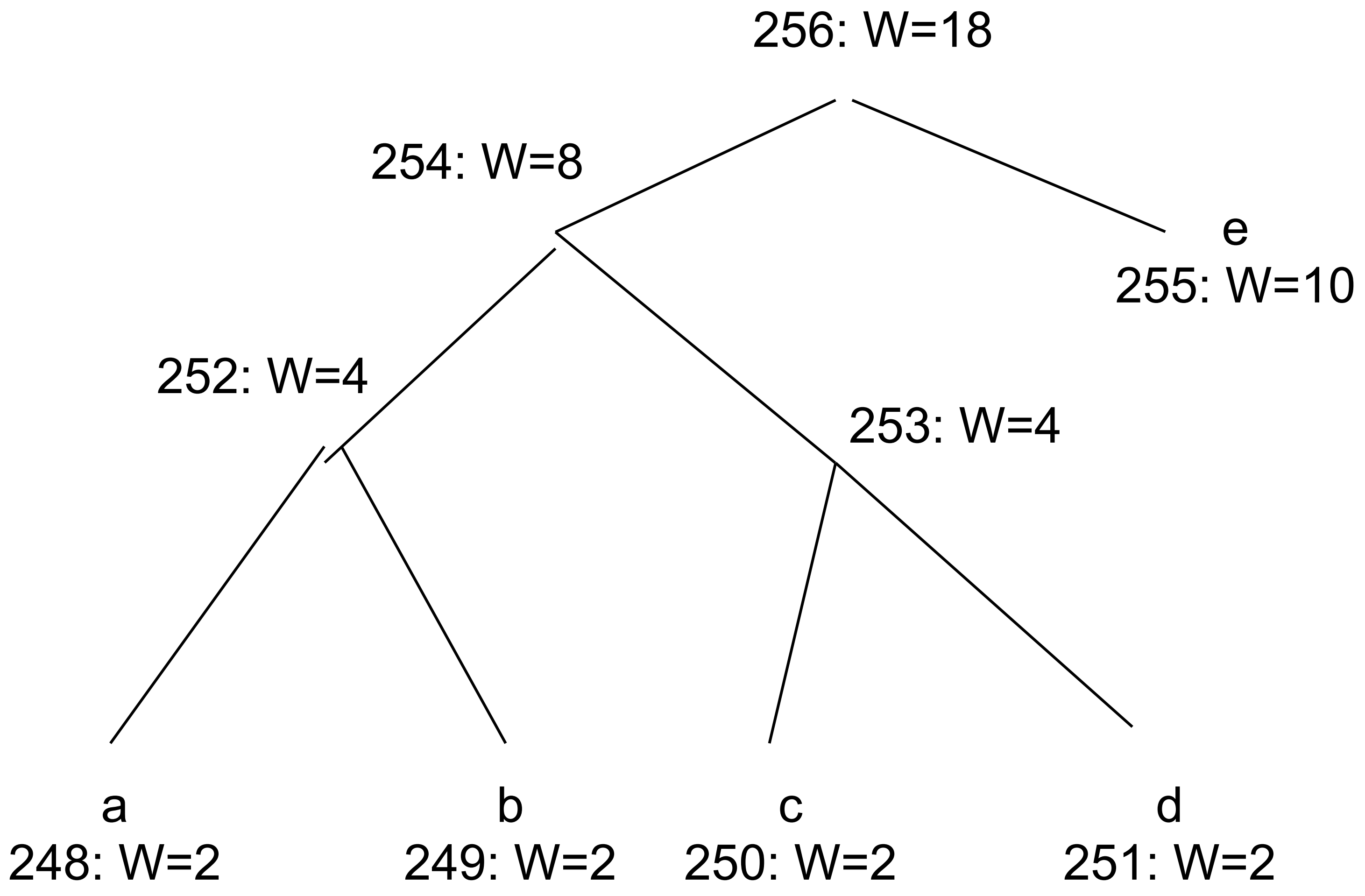
Modified from Wikipedia

More example

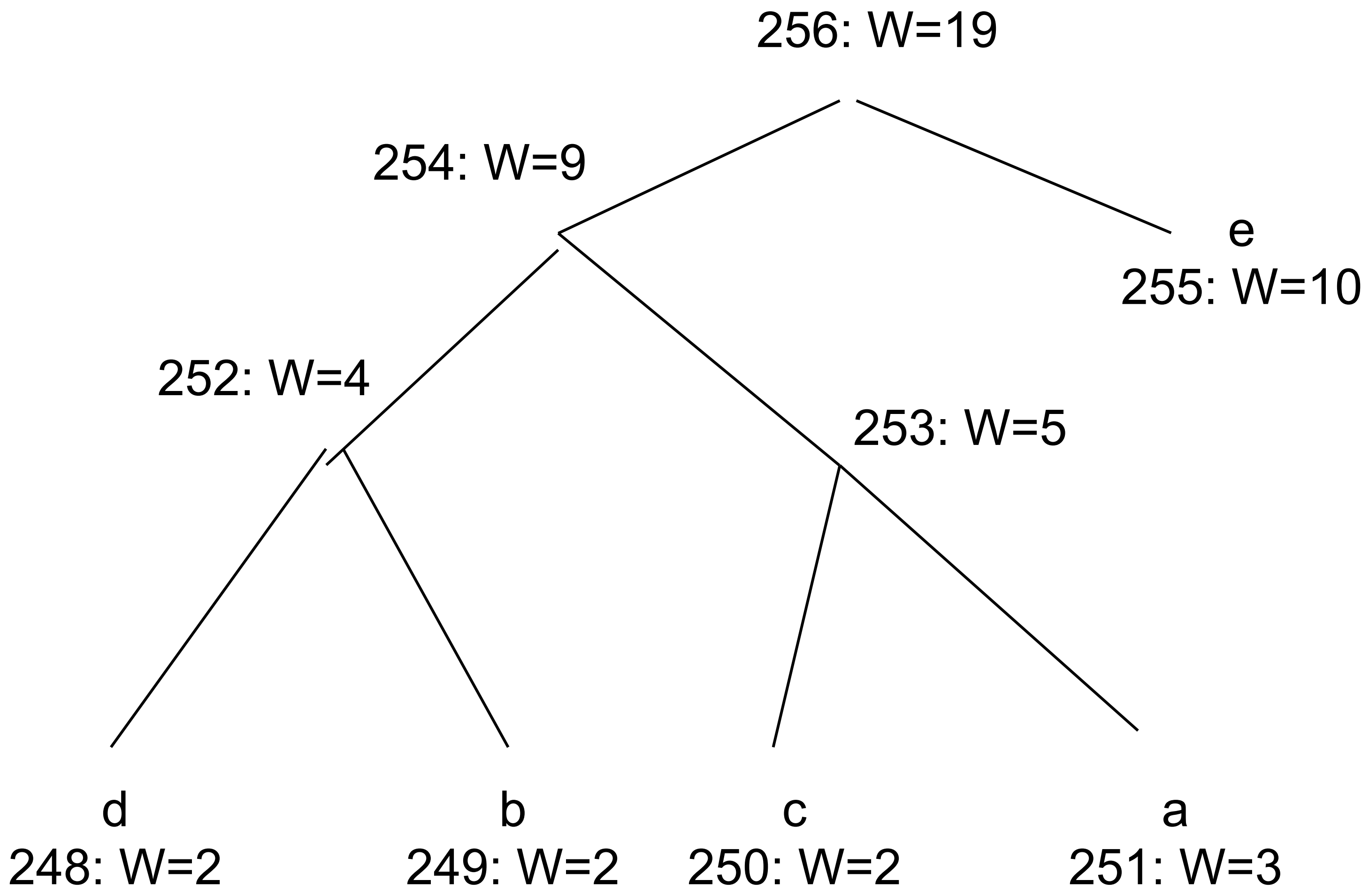


More aaaa…. coming

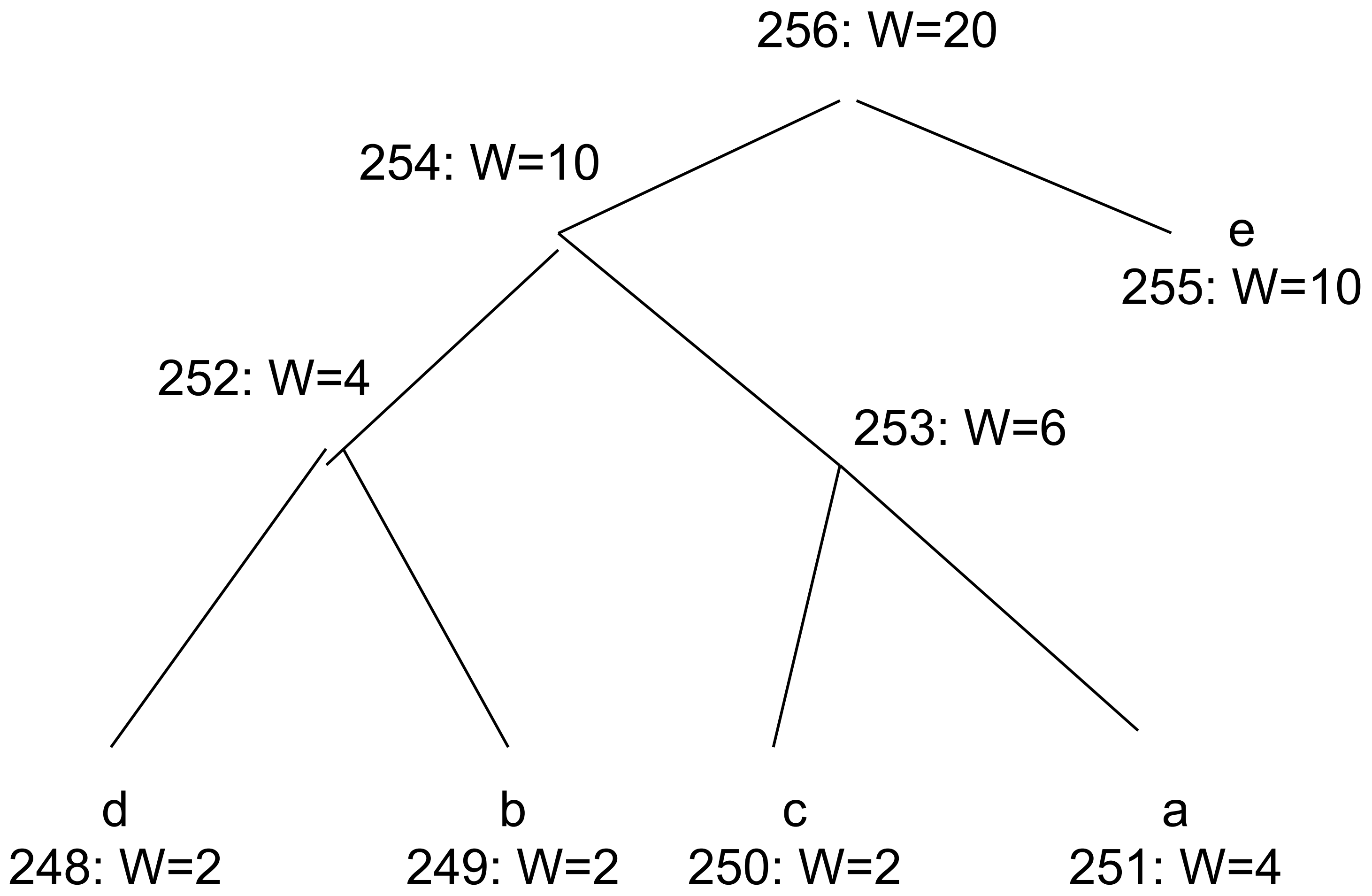
More example



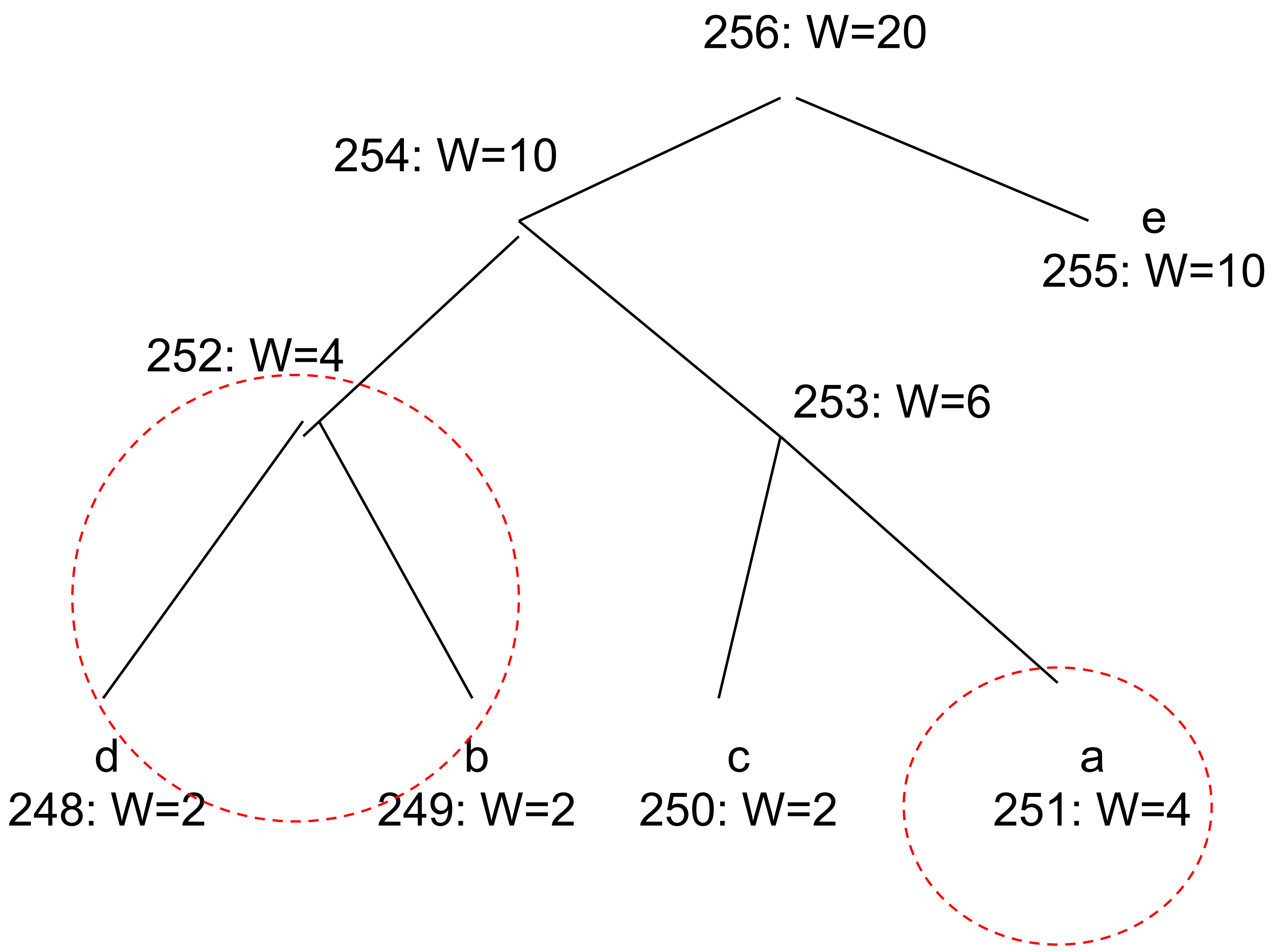
More example



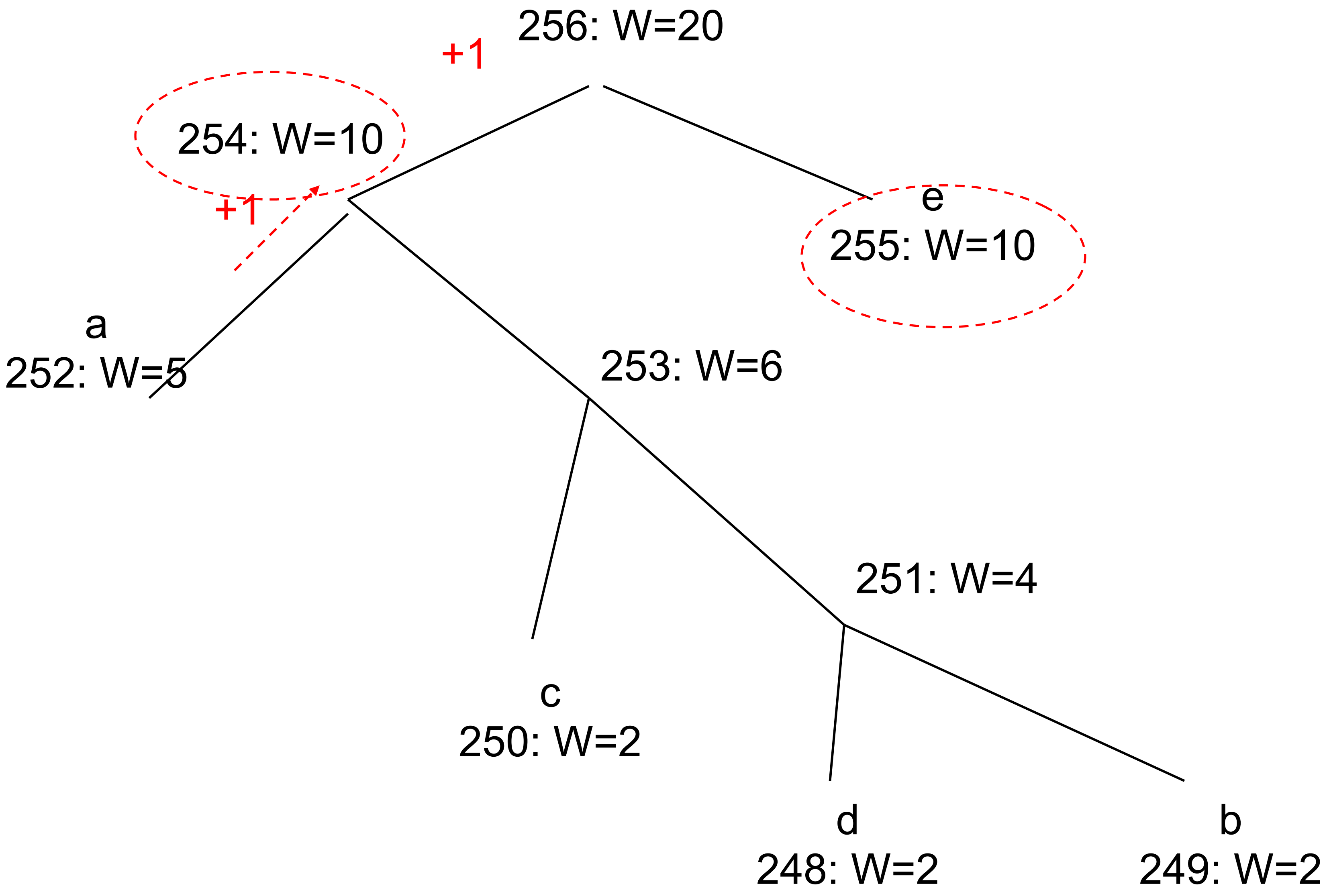
More example



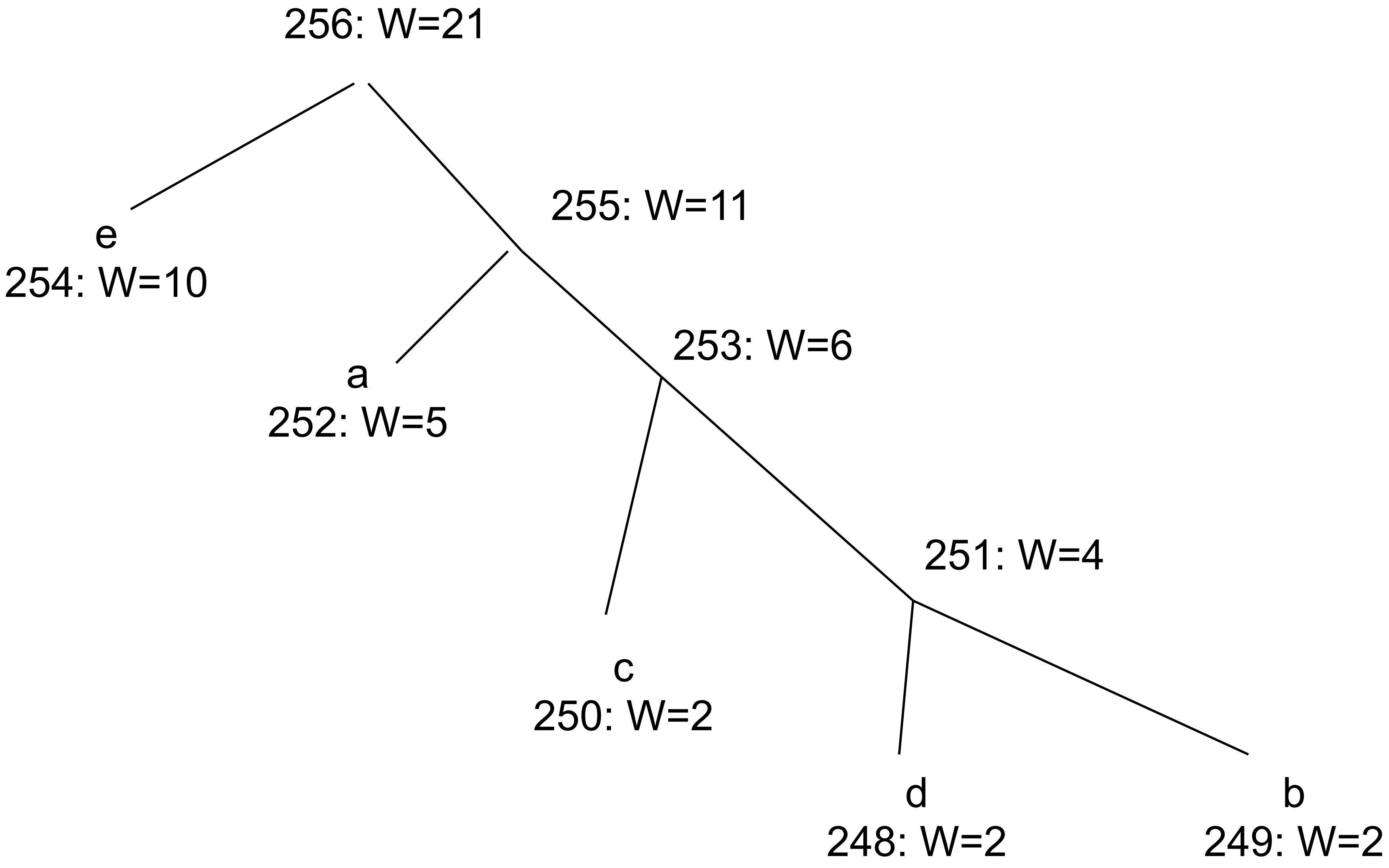
More example



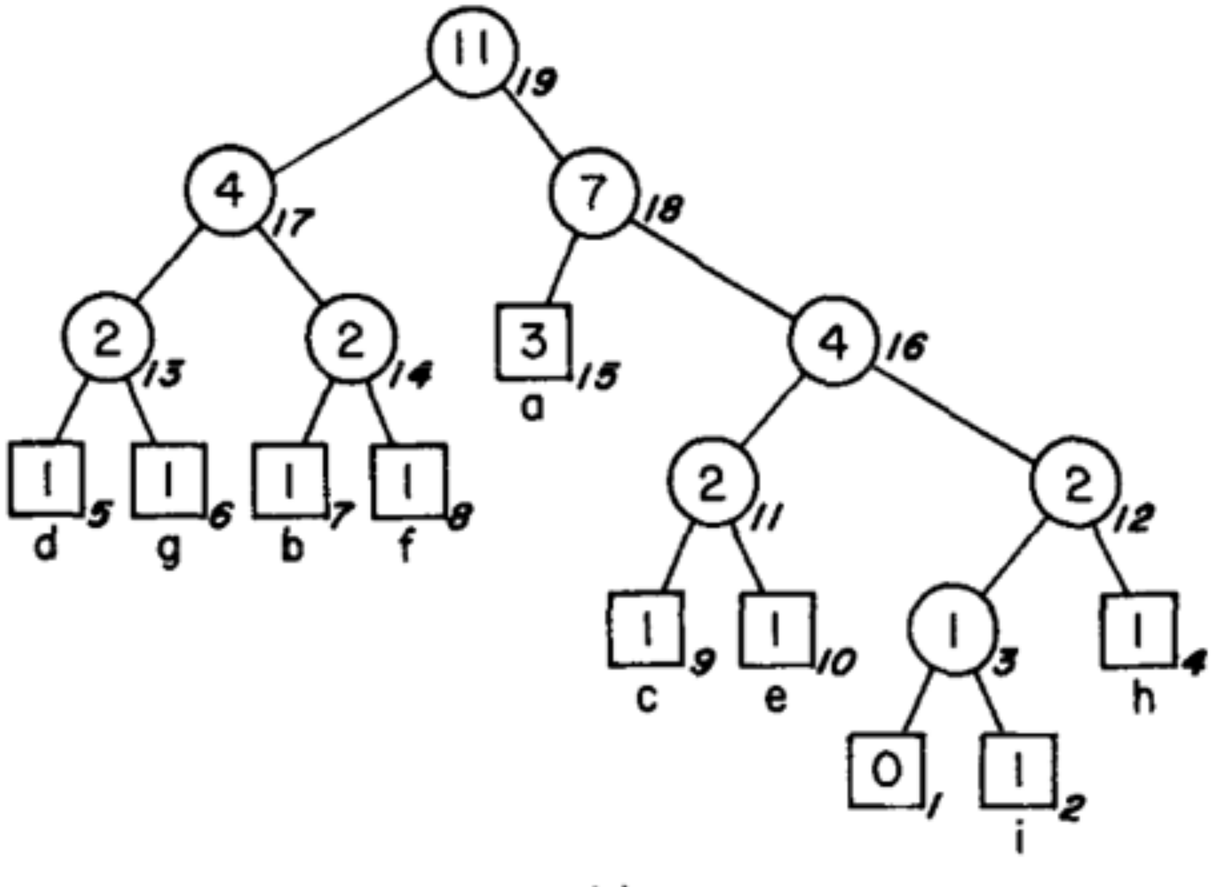
More example



More example

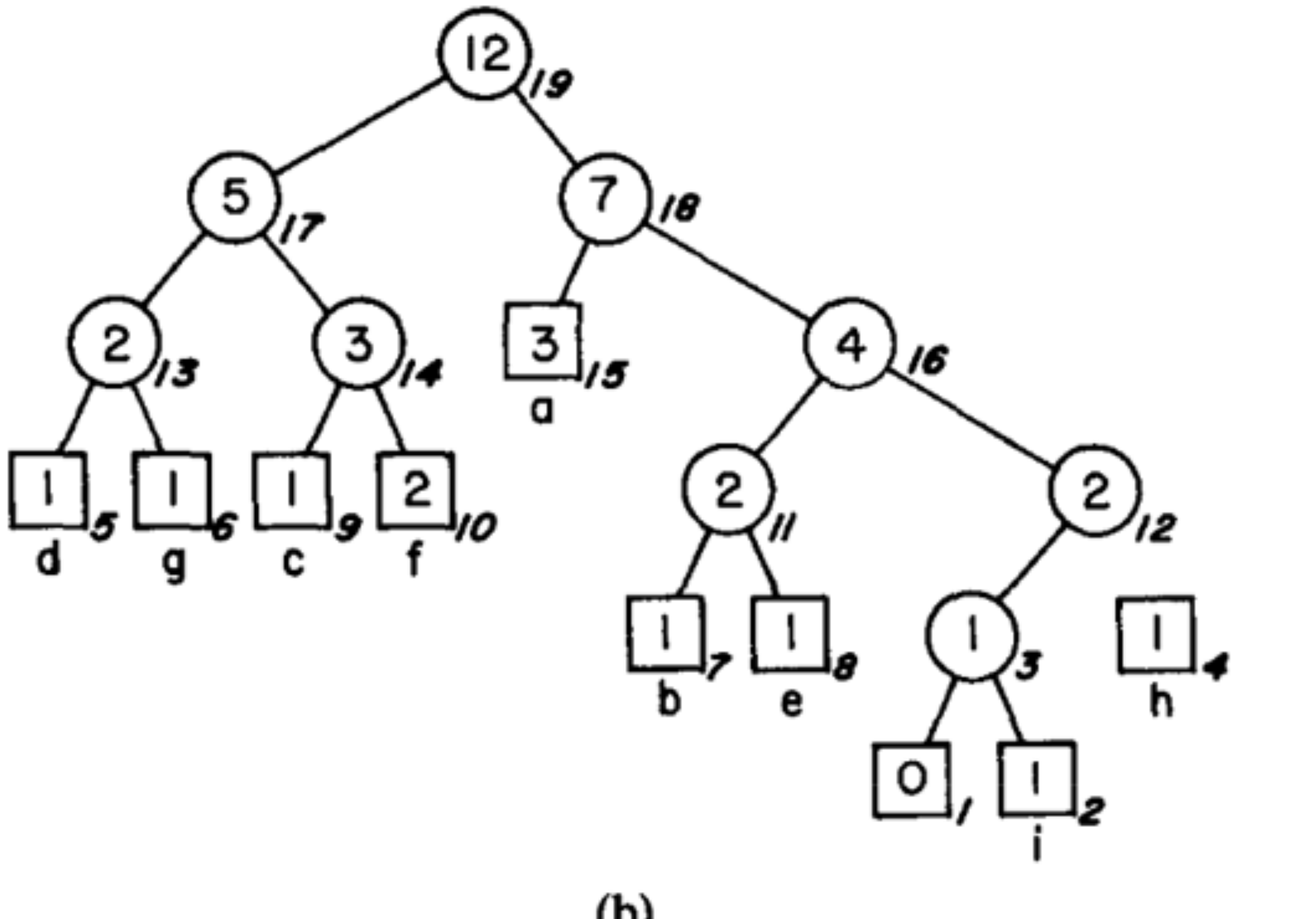


Adaptive Huffman (FGK)



(a)

Adaptive Huffman (FGK): when f is inserted

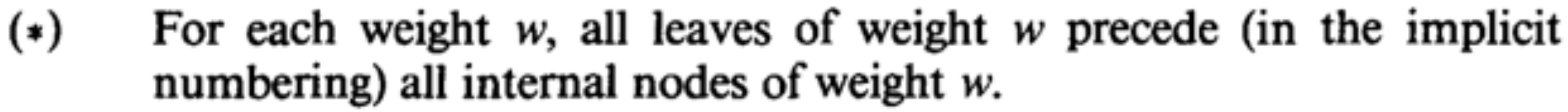


(b)

Adaptive Huffman (FGK vs Vitter)

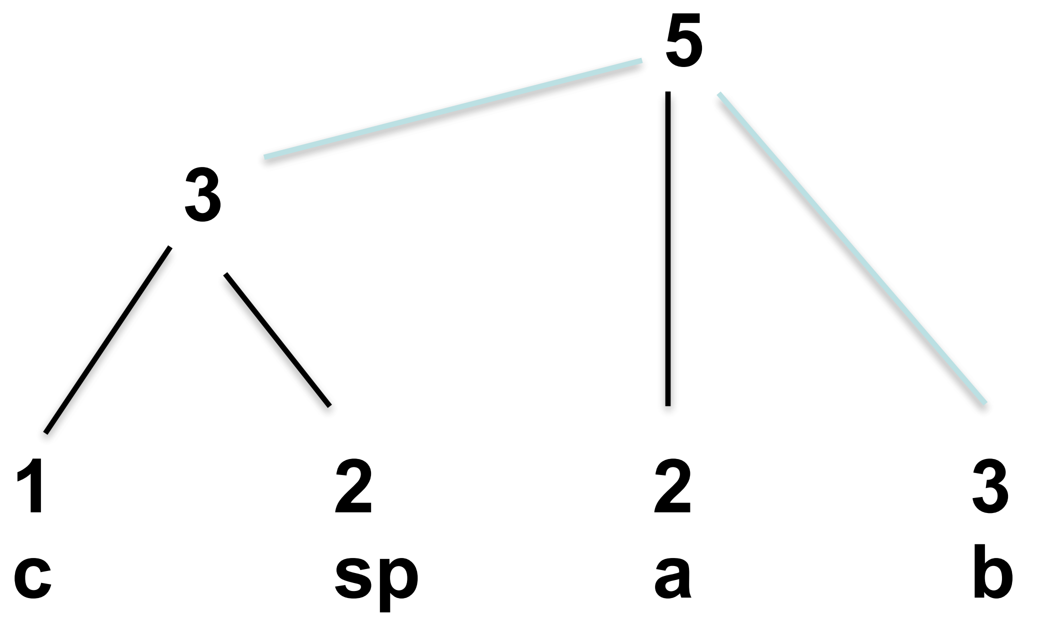
1.
FGK: (Explicit) node numbering
Vitter: Implicit numbering

2.
Vitter's Invariant:


(*) For each weight *w*, all leaves of weight *w* precede (in the implicit numbering) all internal nodes of weight *w*.

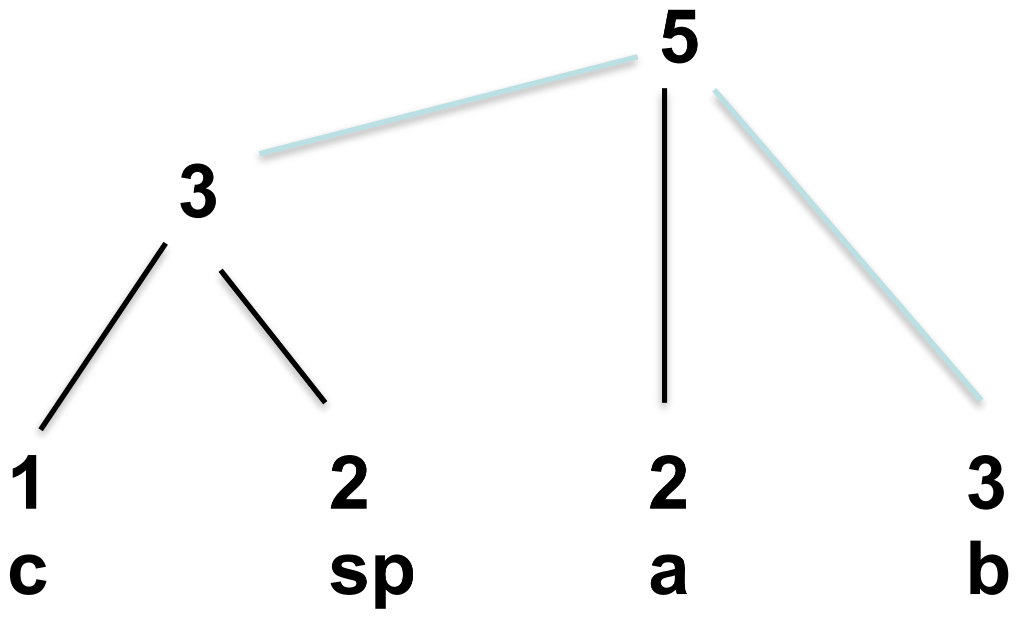
aa bbb c (Huffman)

a	2
b	3
c	1
sp	2



aa bbb c (Huffman)

a	2
b	3
c	1
sp	2

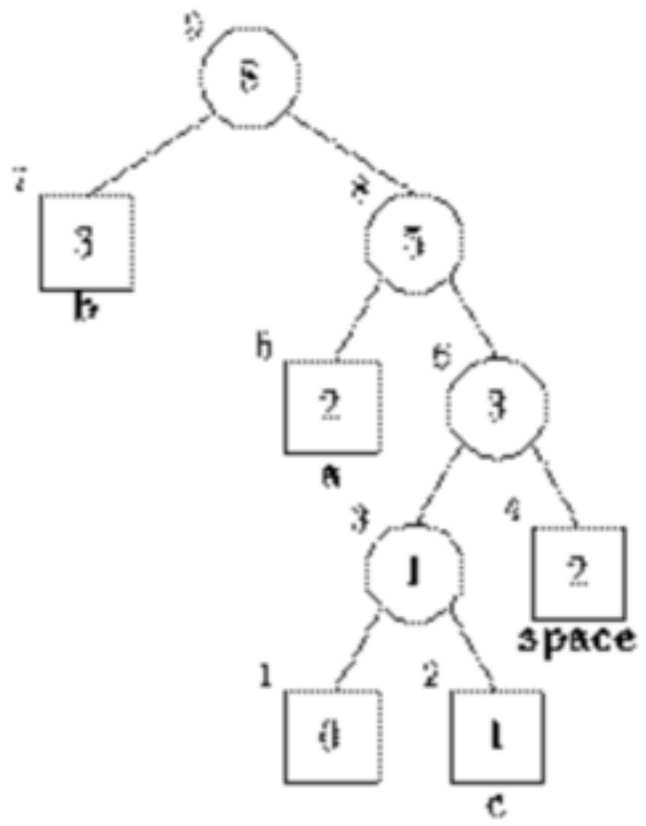


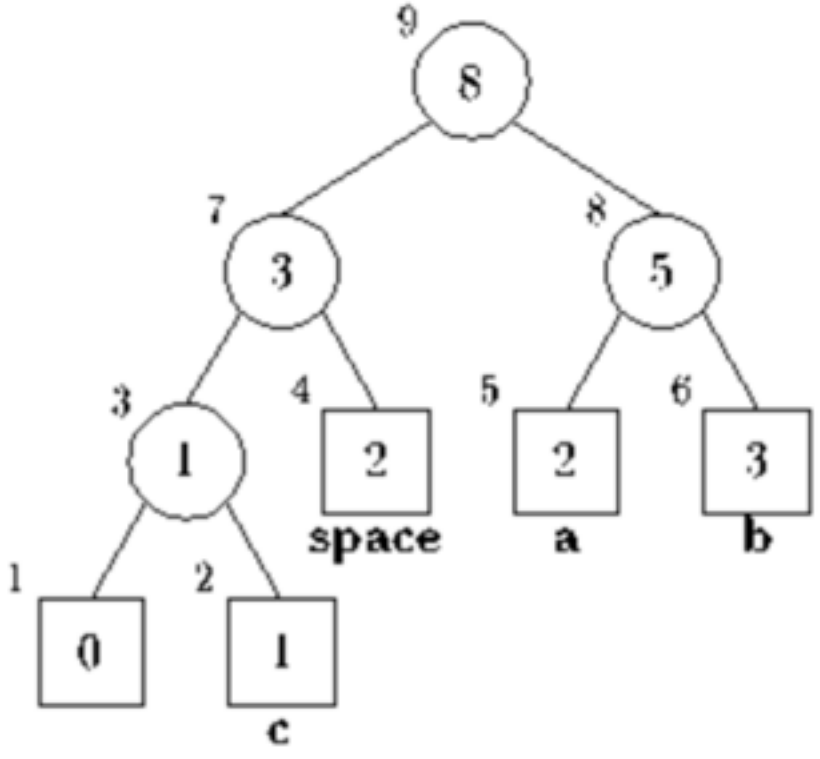
a	01
b	1
c	000
sp	001

Total 16bits

a	10
b	11
c	00
sp	01

Adaptive Huffman (Vitter's Invariant)

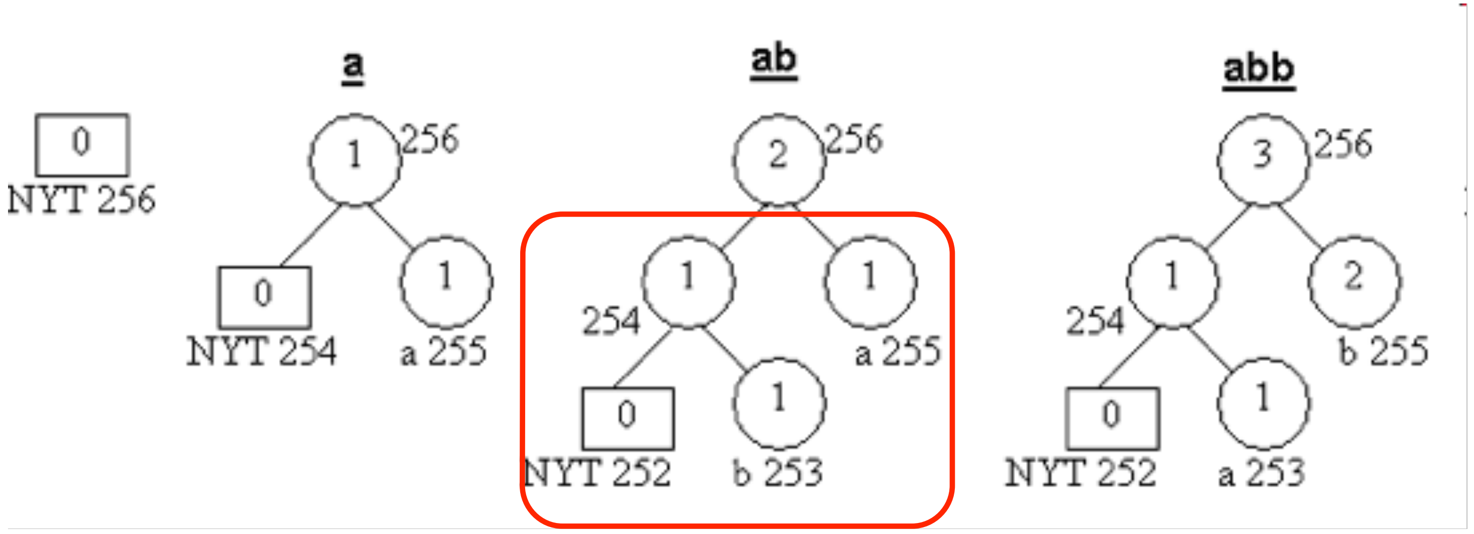




Adaptive Huffman (Vitter 1987)

abbbbba: 01100001011000100110001001100010011000100110001001100001

abbbbba: 0110000100110001**01**11101



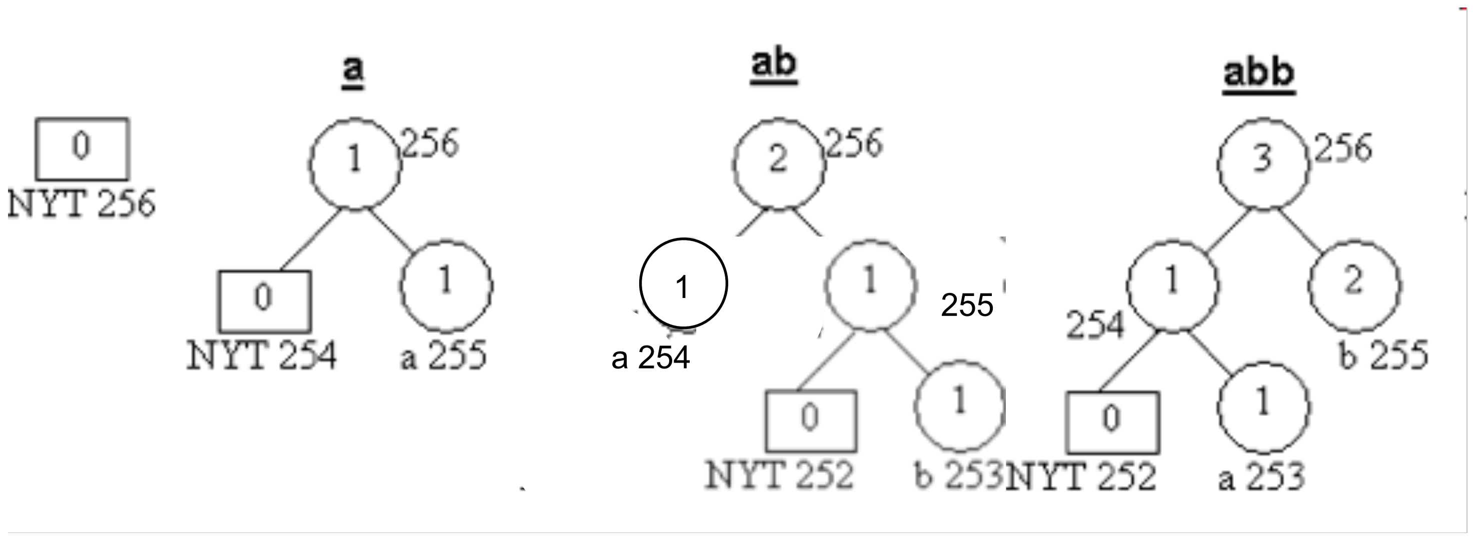
a: 01100001
b: 01100010

Modified from Wikipedia

Adaptive Huffman (Vitter 1987)

abbbbba: 01100001011000100110001001100010011000100110001001100001

abbbbba: 0110000100110001011111101



a: 01100001
b: 01100010

You can correct the Wikipedia article

Modified from Wikipedia

Adaptive Huffman (Vitter'87)

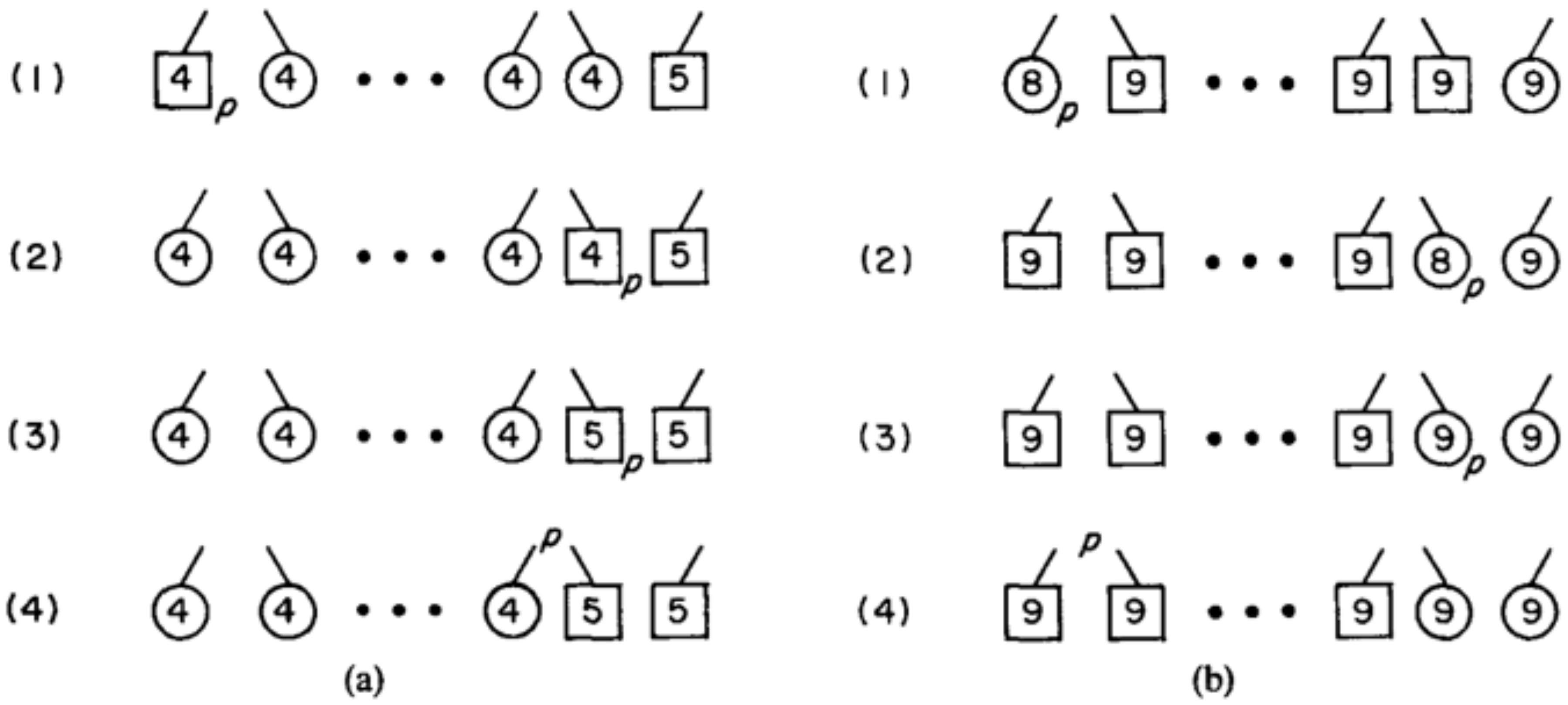


FIG. 6. Algorithm Λ's *SlideAndIncrement* operation. All the nodes in a given block shift to the left one spot to make room for node *p*, which slides over the block to the right. (a) Node *p* is a leaf of weight 4. The internal nodes of weight 4 shift to the left. (b) Node *p* is an internal node of weight 8. The leaves of weight 9 shift to the left.

Adaptive Huffman

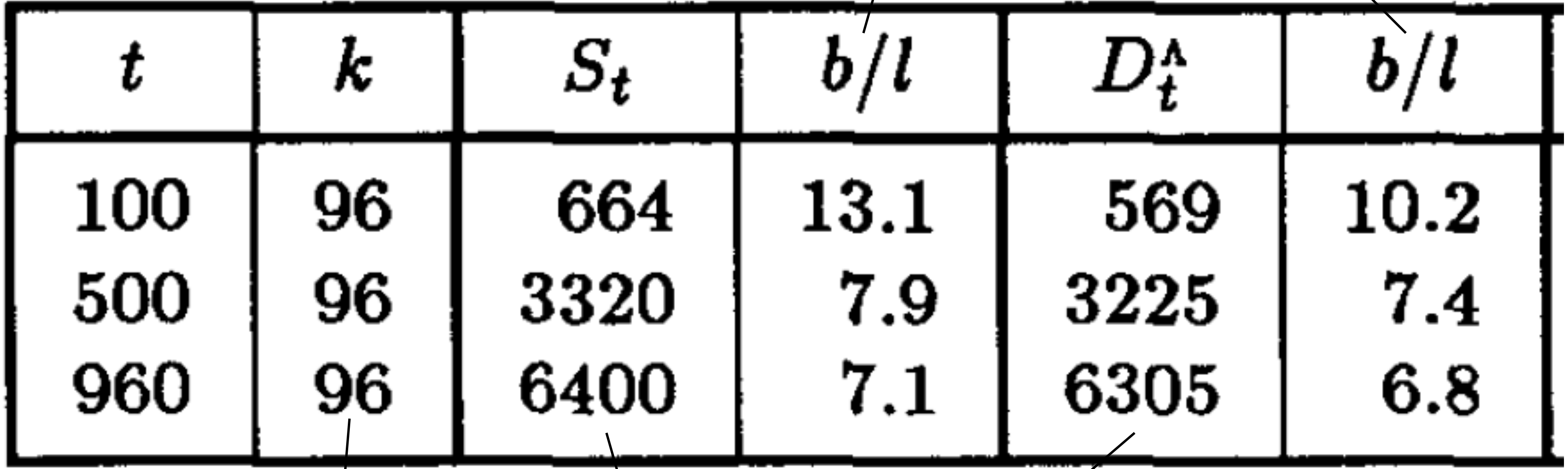
- Question: Adaptive Huffman vs Static Huffman

Compared with Static Huffman

- Dynamic and can offer better compression (cf. Vitter’s experiments next)
  - i.e., the tree can be smaller (hence shorter the code) before the whole bitstream is received.
- Works when prior stat is unavailable
- Saves symbol table overhead (cf. Vitter’s expt next)

Vitter's experiments

Include overheads such as symbol tables / leaf node code etc.



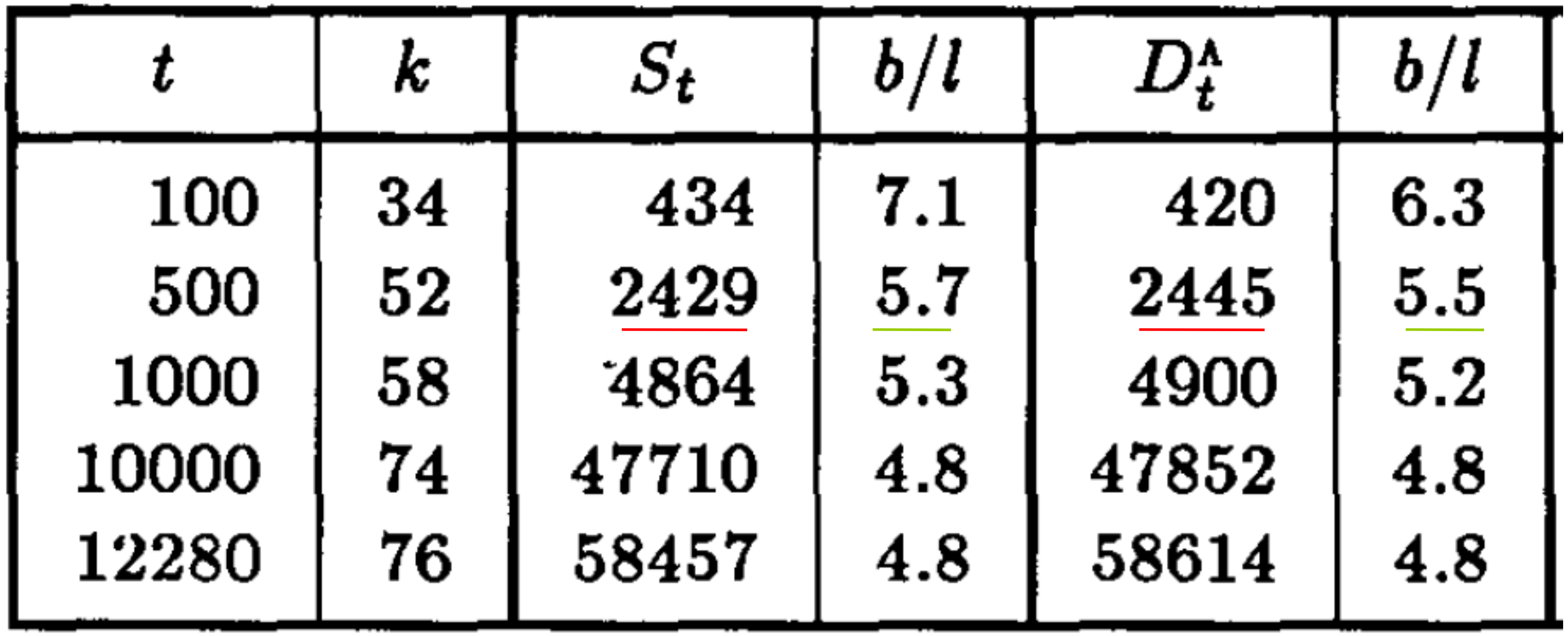
t	k	S_t	b/l	$D_t^{\wedge}$	b/l
100	96	664	13.1	569	10.2
500	96	3320	7.9	3225	7.4
960	96	6400	7.1	6305	6.8

Exclude overheads such as symbol tables / leaf node code etc.

95 ASCII chars + <end-of-line>

From Vitter's paper. You know where it is. ☺

More experiments



t	k	S_t	b/l	$D_t^{\wedge}$	b/l
100	34	434	7.1	420	6.3
500	52	2429	5.7	2445	5.5
1000	58	4864	5.3	4900	5.2
10000	74	47710	4.8	47852	4.8
12280	76	58457	4.8	58614	4.8

Next… BWT

BWT: Burrows–Wheeler Transform

It is a “transform”, not a compression;

but it usually helps compression (esp. text compression).

Excerpted from Wikipedia

Recall from Lecture 1's RLE and BWT example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbbba$

aab4ccac3rcba10b5a$

A simple example

Input:

#BANANAS

All rotations

```
#BANANAS
S#BANANA
AS#BANAN
NAS#BANA
ANAS#BAN
NANAS#BA
ANANAS#B
BANANAS#
```

Sort the rows

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Output

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Exercise: you can try the example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbbba$

Now the inverse…

Input:

```
S
B
N
N
#
A
A
A
```

First add

```
S
B
N
N
#
A
A
A
```

Then sort

```
#
A
A
A
B
N
N
S
```

Add again

```
S#
BA
NA
NA
#B
AN
AN
AS
```

Then sort

```
#B
AN
AN
AS
BA
NA
NA
S#
```

Then add

```
S#B
BAN
NAN
NAS
#BA
ANA
ANA
AS#
```

Then sort

```
#BA
ANA
ANA
AS#
BAN
NAN
NAS
S#B
```

Then add

```
S#BA
BANA
NANA
NAS#
#BAN
ANAN
ANAS
AS#B
```

Then sort

```
#BAN
ANAN
ANAS
AS#B
BANA
NANA
NAS#
S#BA
```

Then add

```
S#BAN
BANAN
NANAS
NAS#B
#BANA
ANANA
ANAS#
AS#BA
```

Then sort

```
#BANA
ANANA
ANAS#
AS#BA
BANAN
NANAS
NAS#B
S#BAN
```

Then add

```
S#BANA
BANANA
NANAS#
NAS#BA
#BANAN
ANANAS
ANAS#B
AS#BAN
```

Then sort

```
#BANAN
ANANAS
ANAS#B
AS#BAN
BANANA
NANAS#
NAS#BA
S#BANA
```

Then add

```
S#BANAN
BANANAS
NANAS#B
NAS#BAN
#BANANA
ANANAS#
ANAS#BA
AS#BANA
```

Then sort

```
#BANANA
ANANAS#
ANAS#BA
AS#BANA
BANANAS
NANAS#B
NAS#BAN
S#BANAN
```

Then add

```
S#BANANA
BANANAS#
NANAS#BA
NAS#BANA
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
```

Then sort (?)

```
#BANANAS
ANANAS#B
ANAS#BAN
AS#BANAN
BANANAS#
NANAS#BA
NAS#BANA
S#BANANA
```

Implementation

- Do we need to represent the table in the encoder?
- No, a single pointer for each row is needed.

BWT(S)

```
**function** BWT (string s)
  create a table, rows are all possible
       rotations of s
  sort rows alphabetically
  return (last column of the table)
```

InverseBWT(S)

```
function inverseBWT (string s)
  create empty table

  repeat length(s) times
      insert s as a column of table before first
              column of the table   // first insert creates
              first column
      sort rows of the table alphabetically
  return (row that ends with the 'EOF' character)
```

Move to Front (MTF)

- Reduce entropy based on local frequency correlation
- Usually used for BWT before an entropy-encoding step
- Author and detail:
  - Original paper at cs9319/papers
  - http://www.arturocampos.com/ac_mtf.html

Example: abaabacad

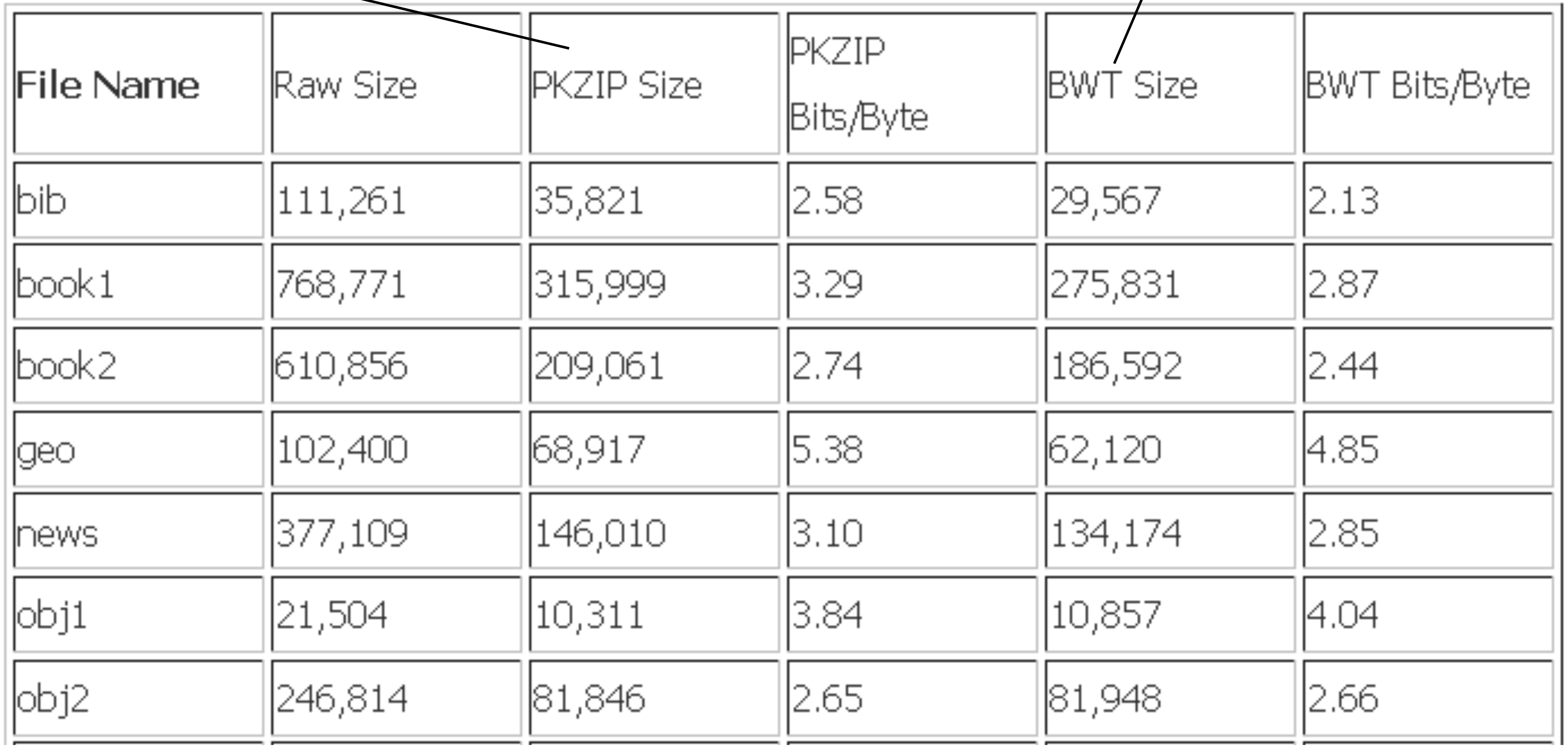
Symbol	Code	List
a	0	abcde…..
b	1	bacde…..
a	1	abcde…..
a	0	abcde…..
b	1	bacde…..
a	1	abcde…..
c	2	cabde…..
a	1	acbde…..
d	3	dacbe…..

To transform a general file, the list has 256 ASCII symbols.

BWT compressor vs ZIP

ZIP (i.e., LZW based)

BWT+RLE+MTF+AC



File Name	Raw Size	PKZIP Size	PKZIP Bits/Byte	BWT Size	BWT Bits/Byte
bib	111,261	35,821	2.58	29,567	2.13
book1	768,771	315,999	3.29	275,831	2.87
book2	610,856	209,061	2.74	186,592	2.44
geo	102,400	68,917	5.38	62,120	4.85
news	377,109	146,010	3.10	134,174	2.85
obj1	21,504	10,311	3.84	10,857	4.04
obj2	246,814	81,846	2.65	81,948	2.66

From http://marknelson.us/1996/09/01/bwt/