# COMP9319 Web Data Compression and Search

Adaptive Huffman (revisit),
Assignment 1 FAQ

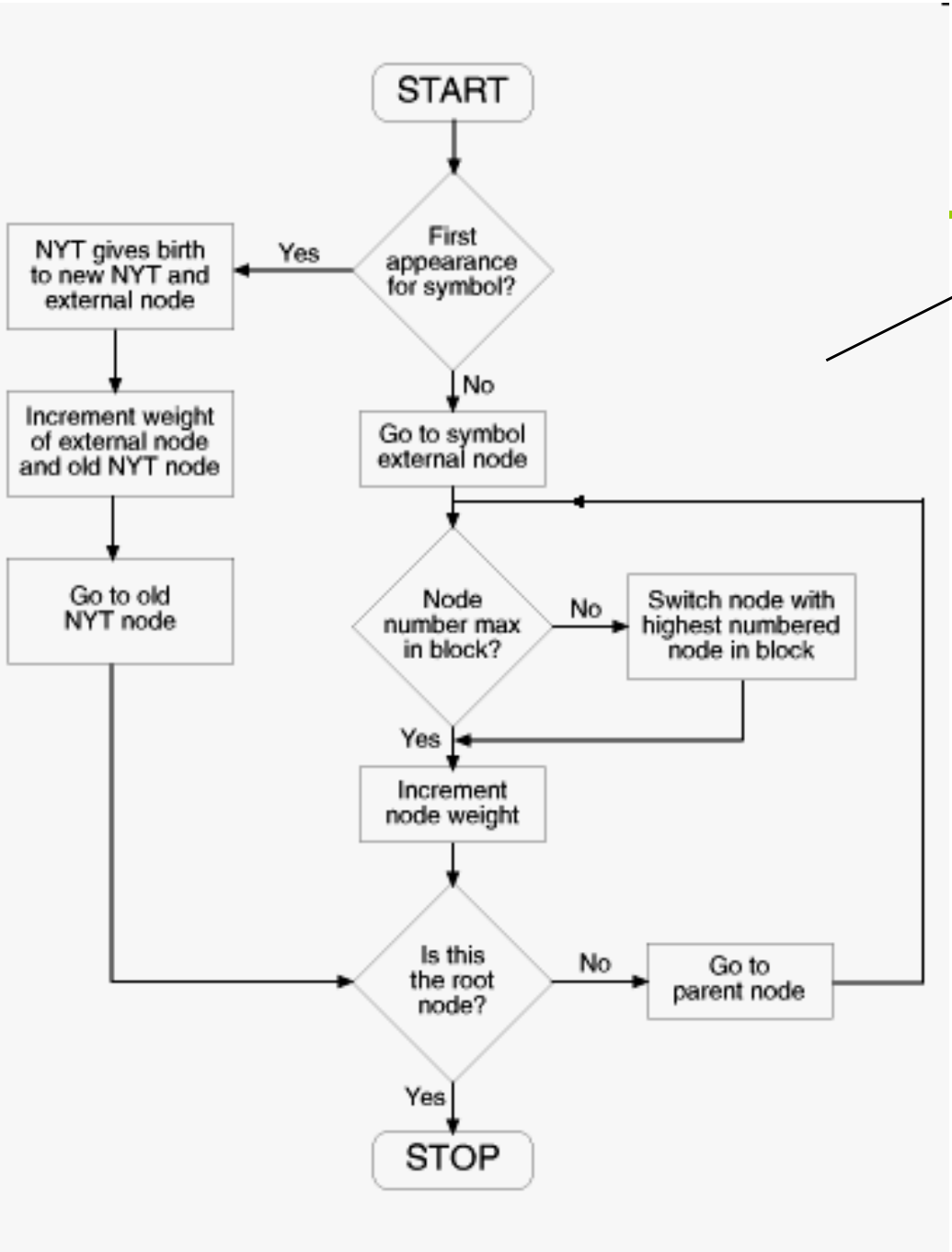
# Adaptive Huffman (Algorithm outline)

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.

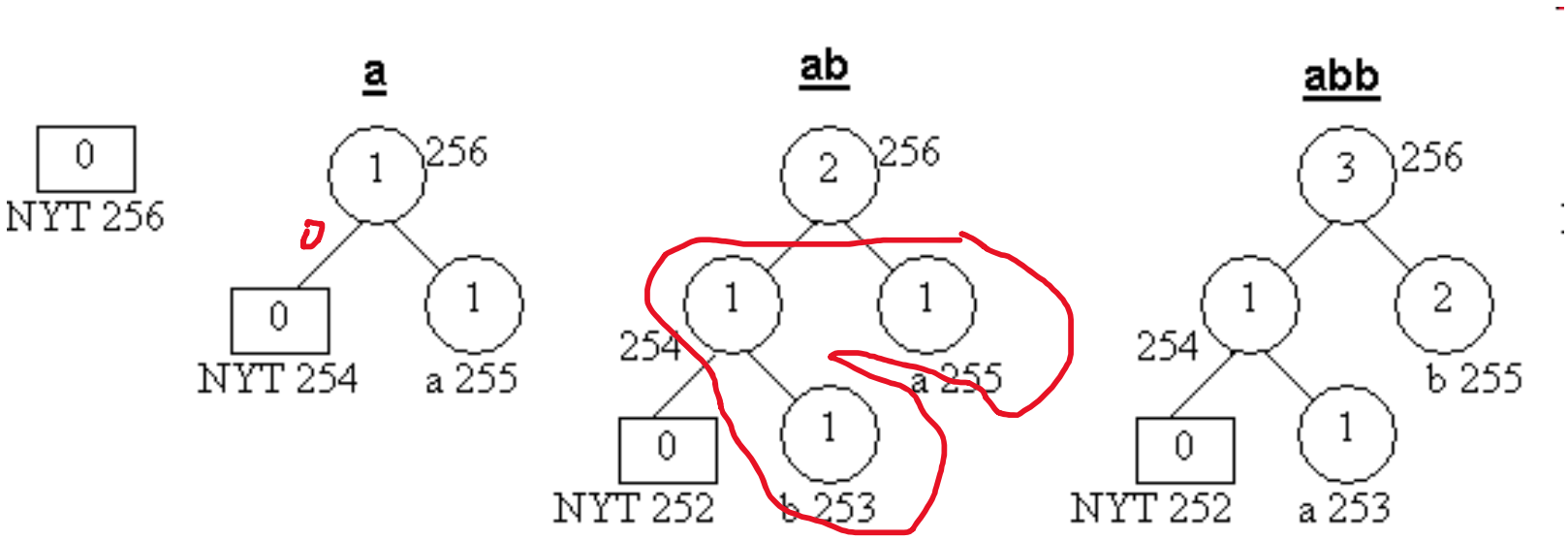



The update procedure from Introduction to Data Compression by by Sayood Khalid

# Adaptive Huffman

abbbbba: 01100001011000100110001001100010011000100110001001100001

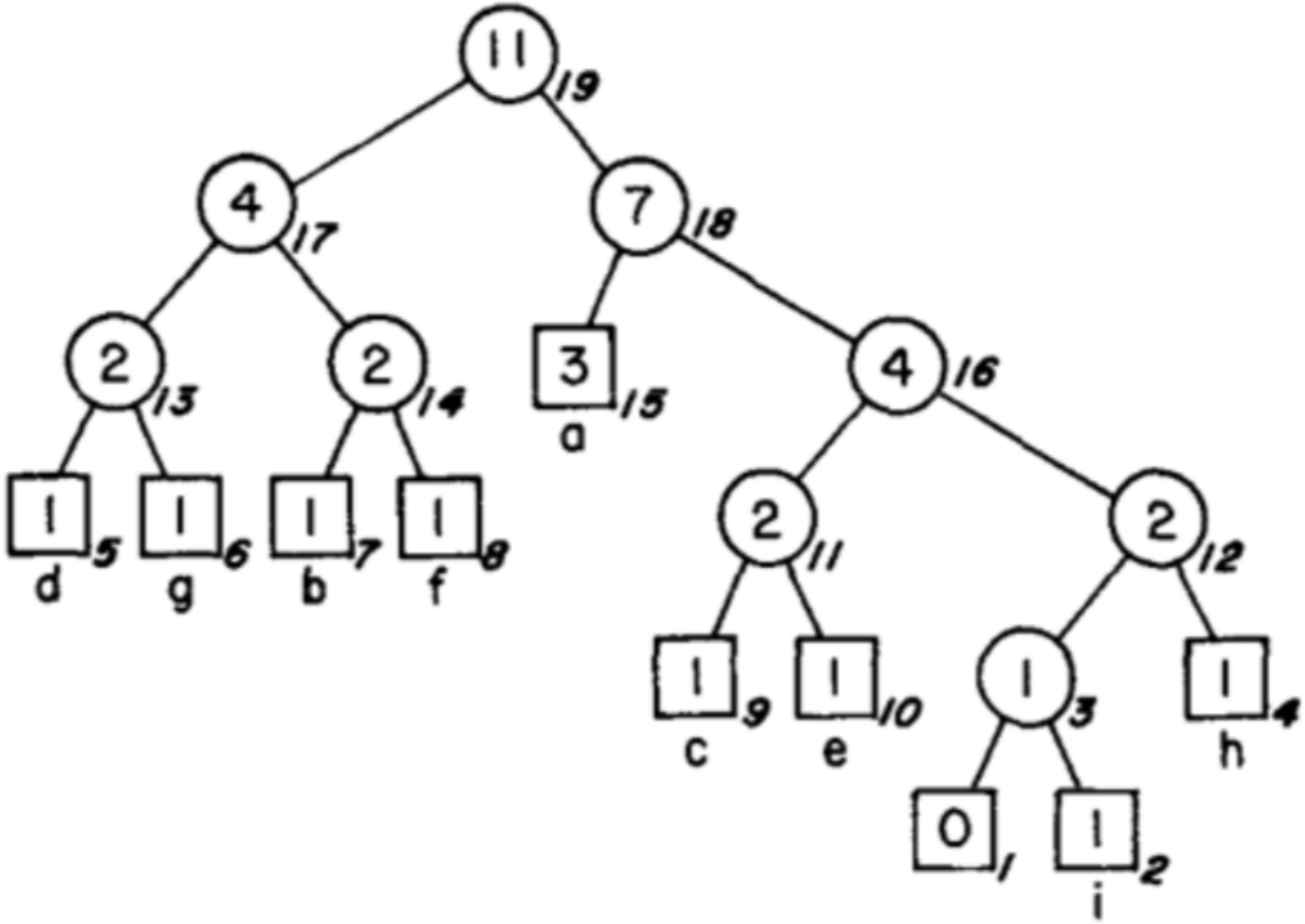
abbbbba: 0110000100110001000111101



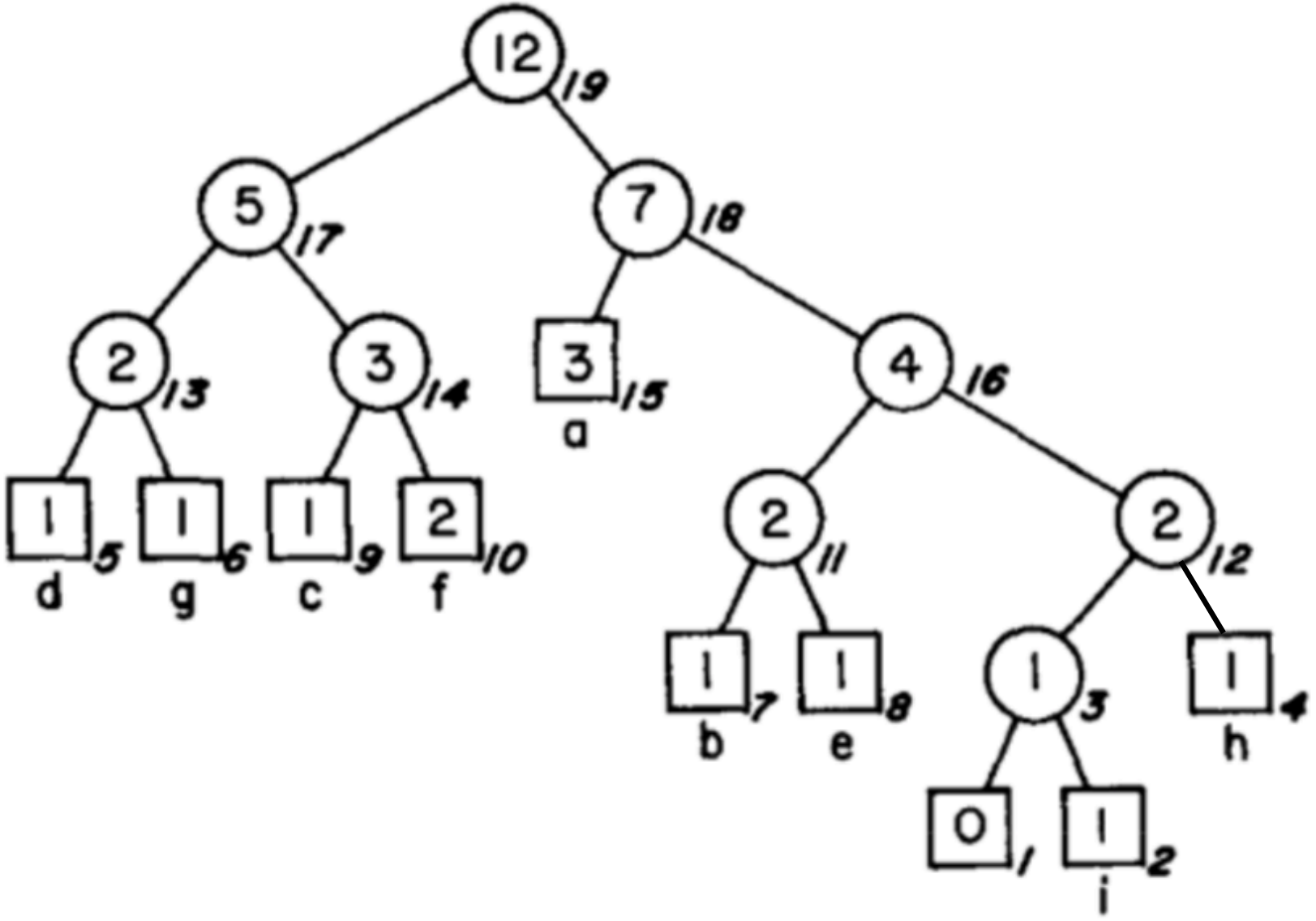

a: 01100001
b: 01100010

From an old Wikipedia page

# Adaptive Huffman (FGK)



# Adaptive Huffman (FGK): when f is inserted



# 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 $w$.

# Adaptive Huffman (Vitter'87)

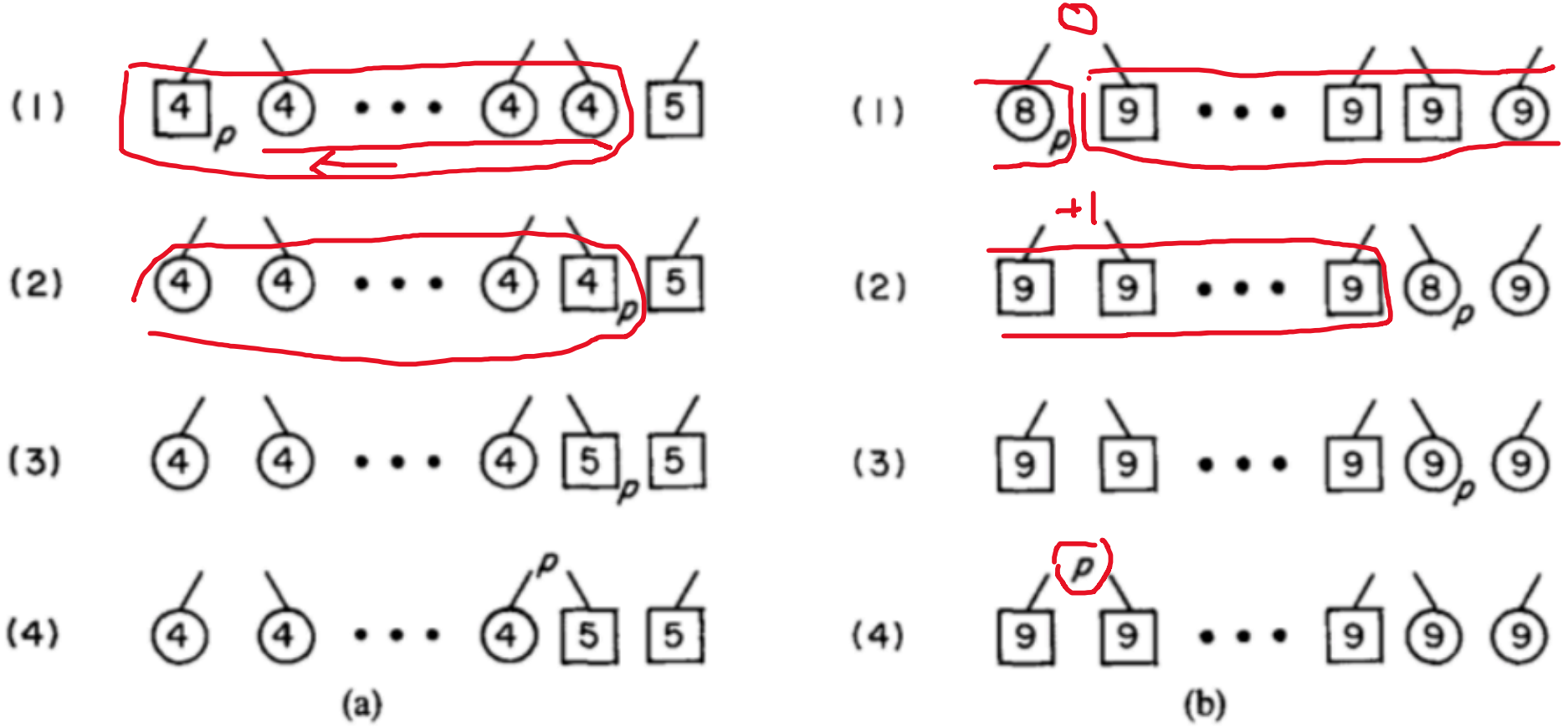
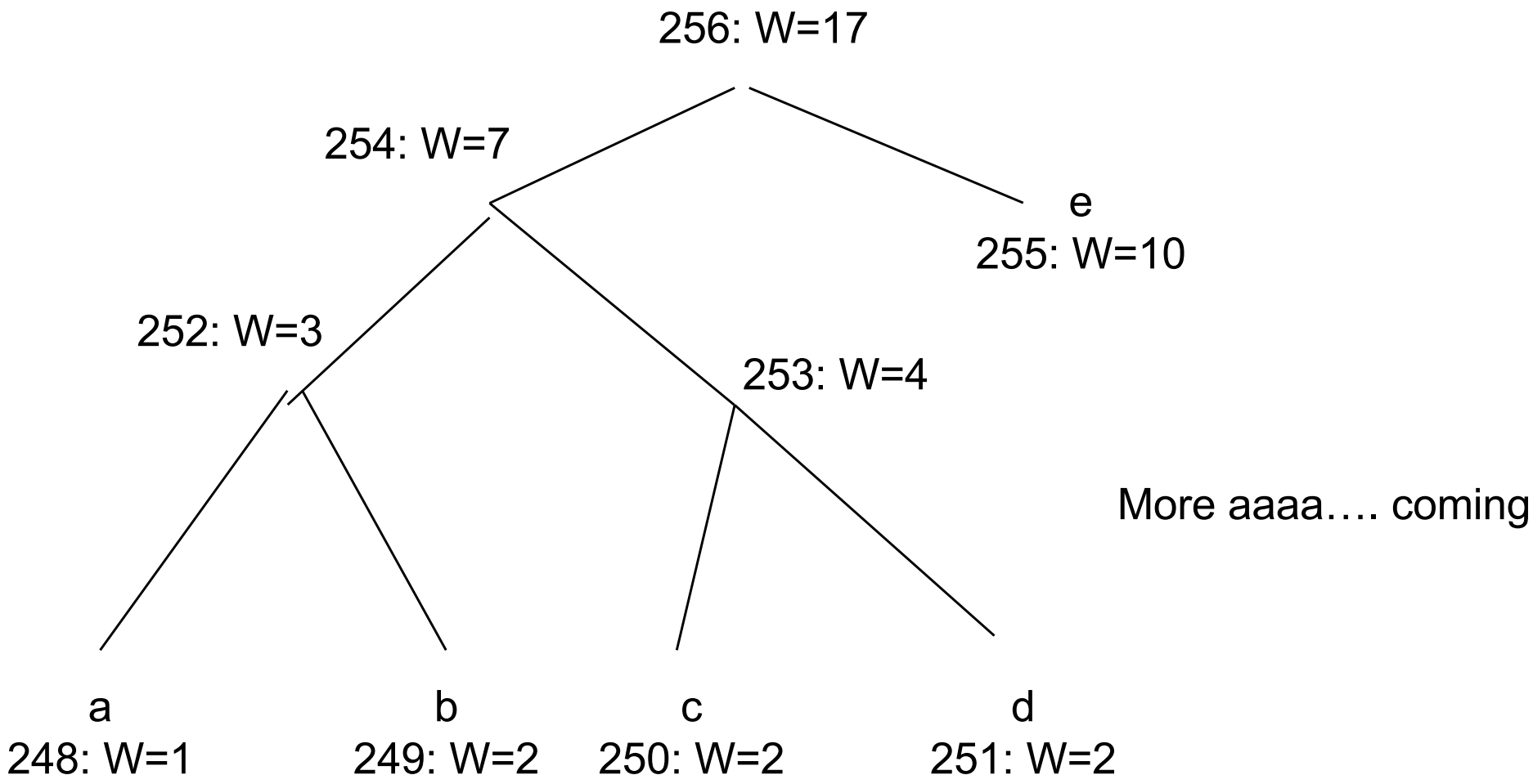


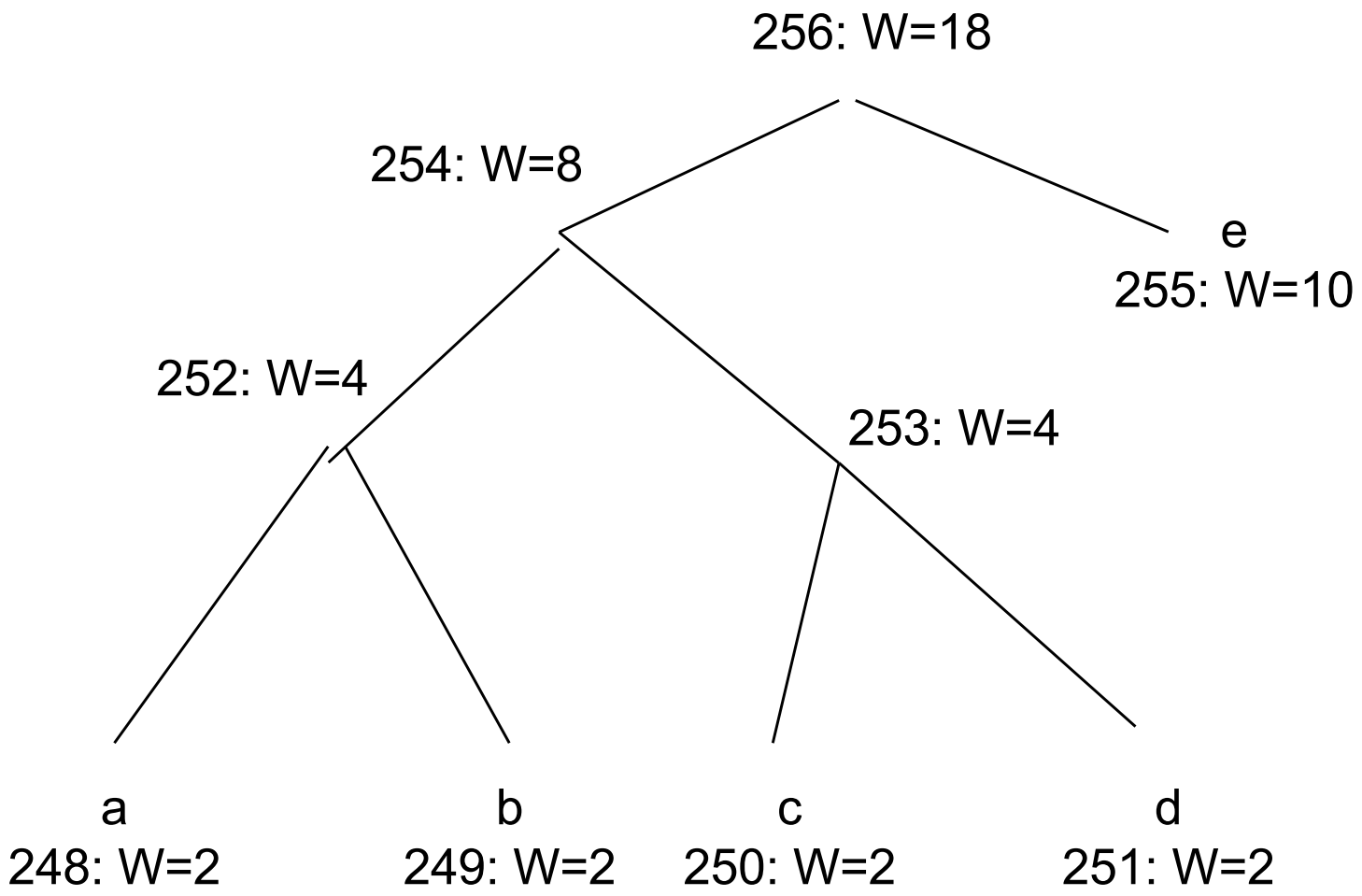

FIG. 6. Algorithm Λ's *SlideAndIncrement* operation. All the nodes in a given block shift to the left one spot to make room for node $p$, which slides over the block to the right. (a) Node $p$ is a leaf of weight 4. The internal nodes of weight 4 shift to the left. (b) Node $p$ is an internal node of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

# More example on Vitter's

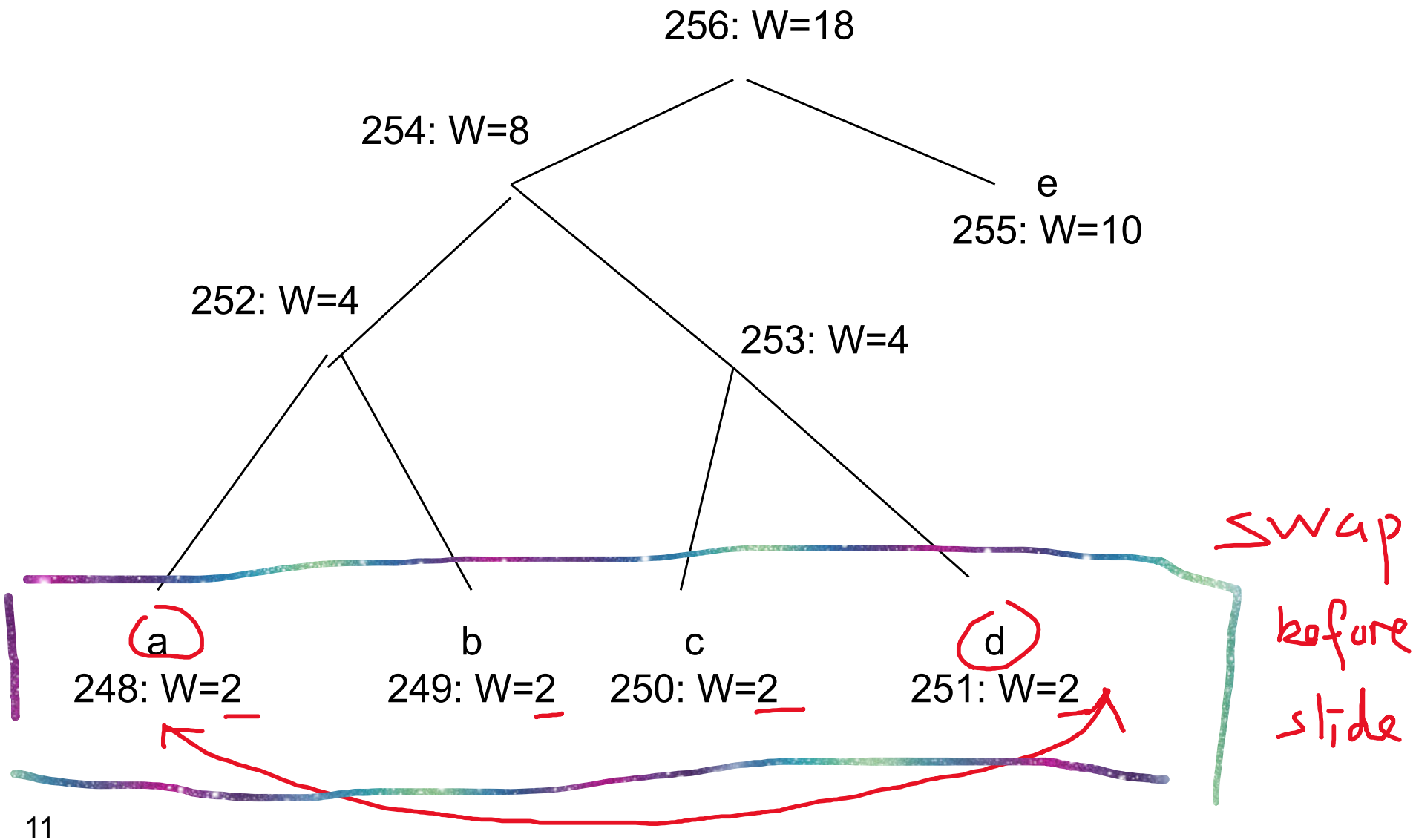



More aaaa…. coming

# More example



# More example



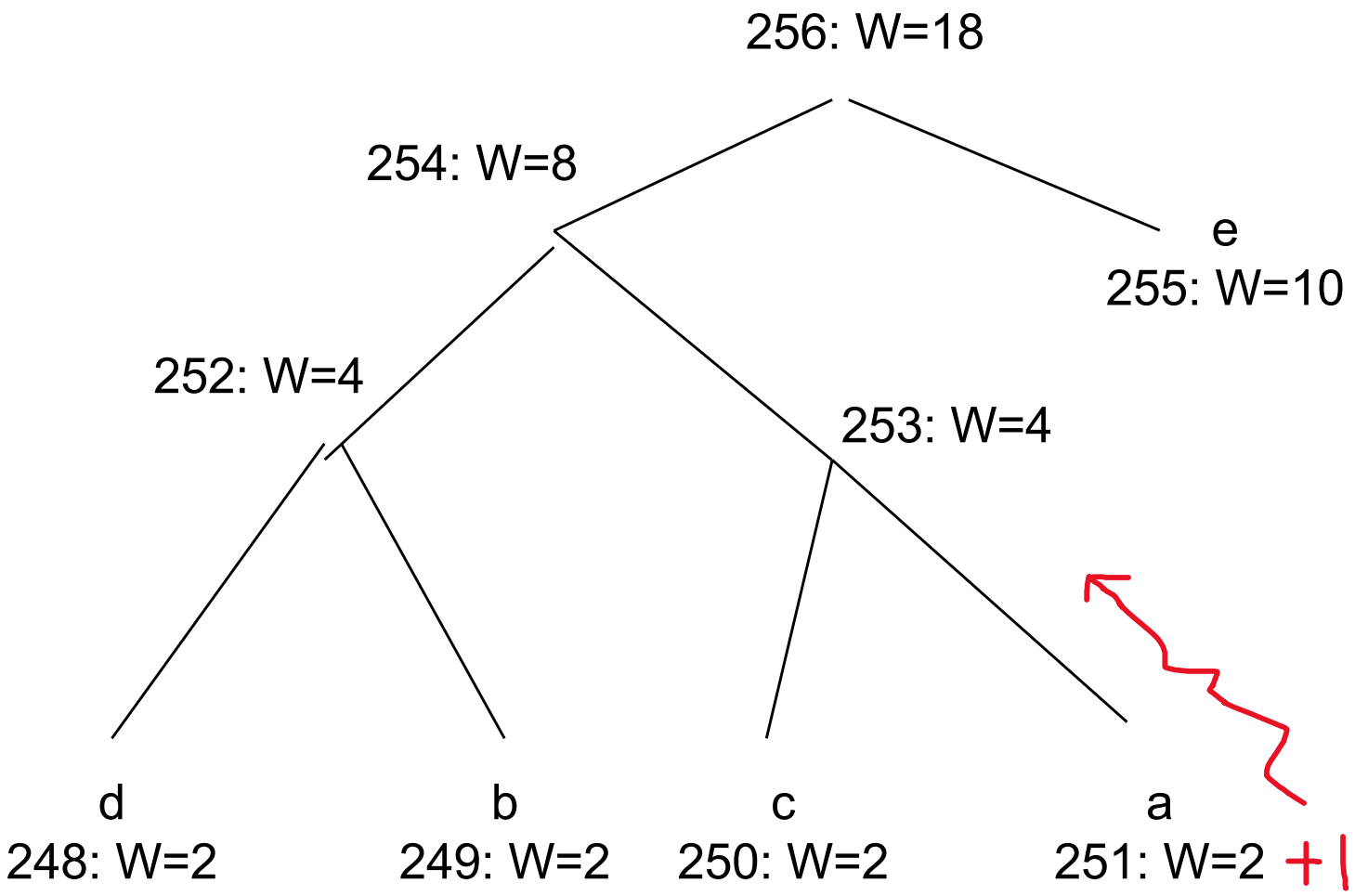
# The Vitter's algo: swaps then slides

*Definition* 4.1. *Blocks* are equivalence classes of nodes defined by $v \equiv x$ iff nodes $v$ and $x$ have the same weight and are either both internal nodes or both leaves. The *leader* of a block is the highest numbered (in the implicit numbering) node in a block.

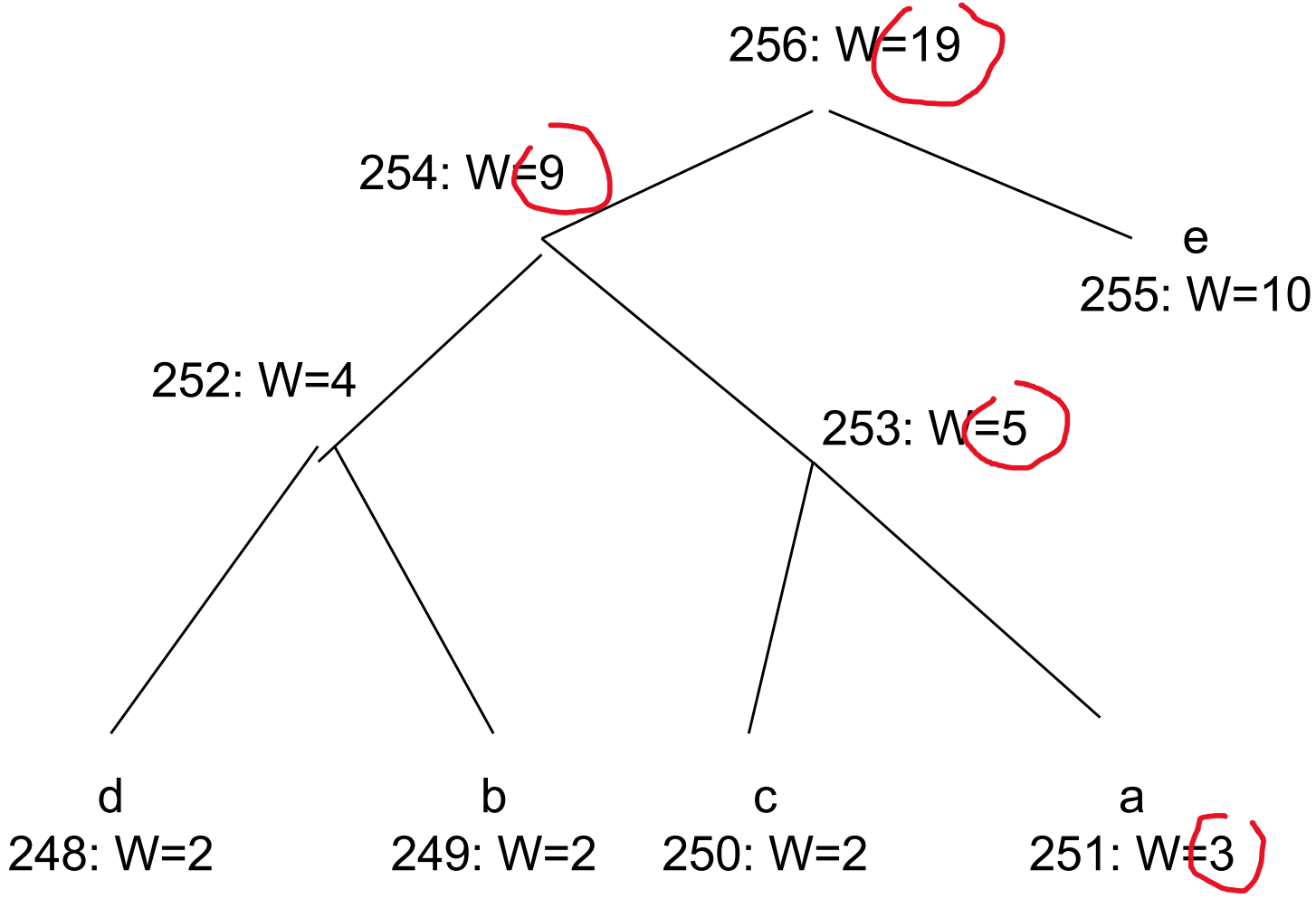
The blocks are linked together by increasing order of weight; a leaf block always precedes an internal block of the same weight. The main operation of the algorithm needed to maintain invariant (*) is the *SlideAndIncrement* operation, illustrated in Figure 6. The version of *Update* we use for Algorithm Λ is outlined below:

```
procedure Update;
begin
leafToIncrement := 0;
q := leaf node corresponding to a_{i_{t+1}};
if (q is the 0-node) and (k < n − 1) then
   begin  {Special Case #1}
   Replace q by an internal 0-node with two leaf 0-node children, such that the right child
         corresponds to a_{i_{t+1}};
   q := internal 0-node just created;
   leafToIncrement := the right child of q
   end
else begin
   Interchange q in the tree with the leader of its block;
   if q is the sibling of the 0-node then
      begin  {Special Case #2}
      leafToIncrement := q;
      q := parent of q
      end
   end;
while q is not the root of the Huffman tree do
   {Main loop; q must be the leader of its block}
   SlideAndIncrement(q);
if leafToIncrement ≠ 0 then  {Handle the two special cases}
   SlideAndIncrement(leafToIncrement)
end;
```

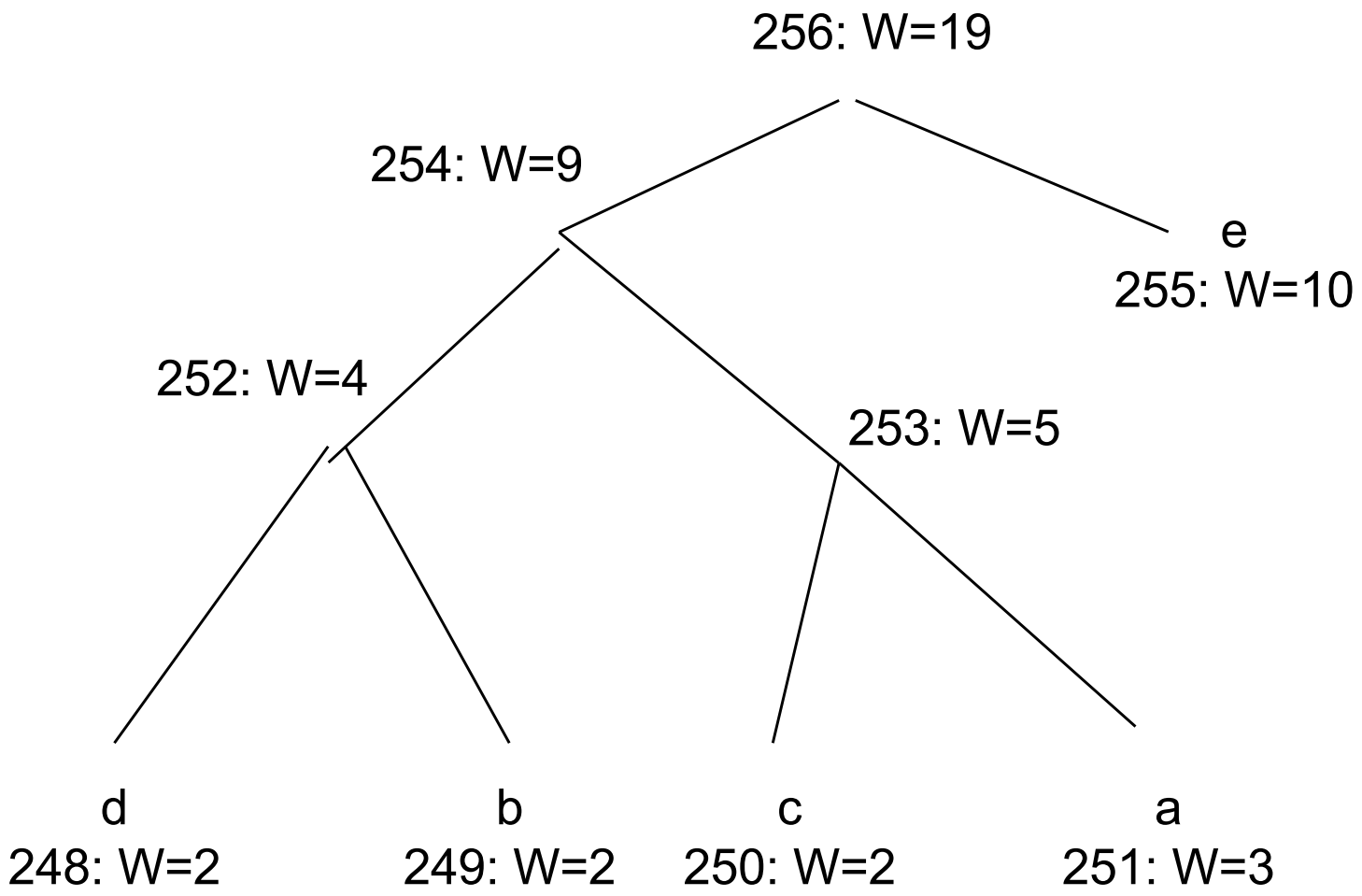
# More example



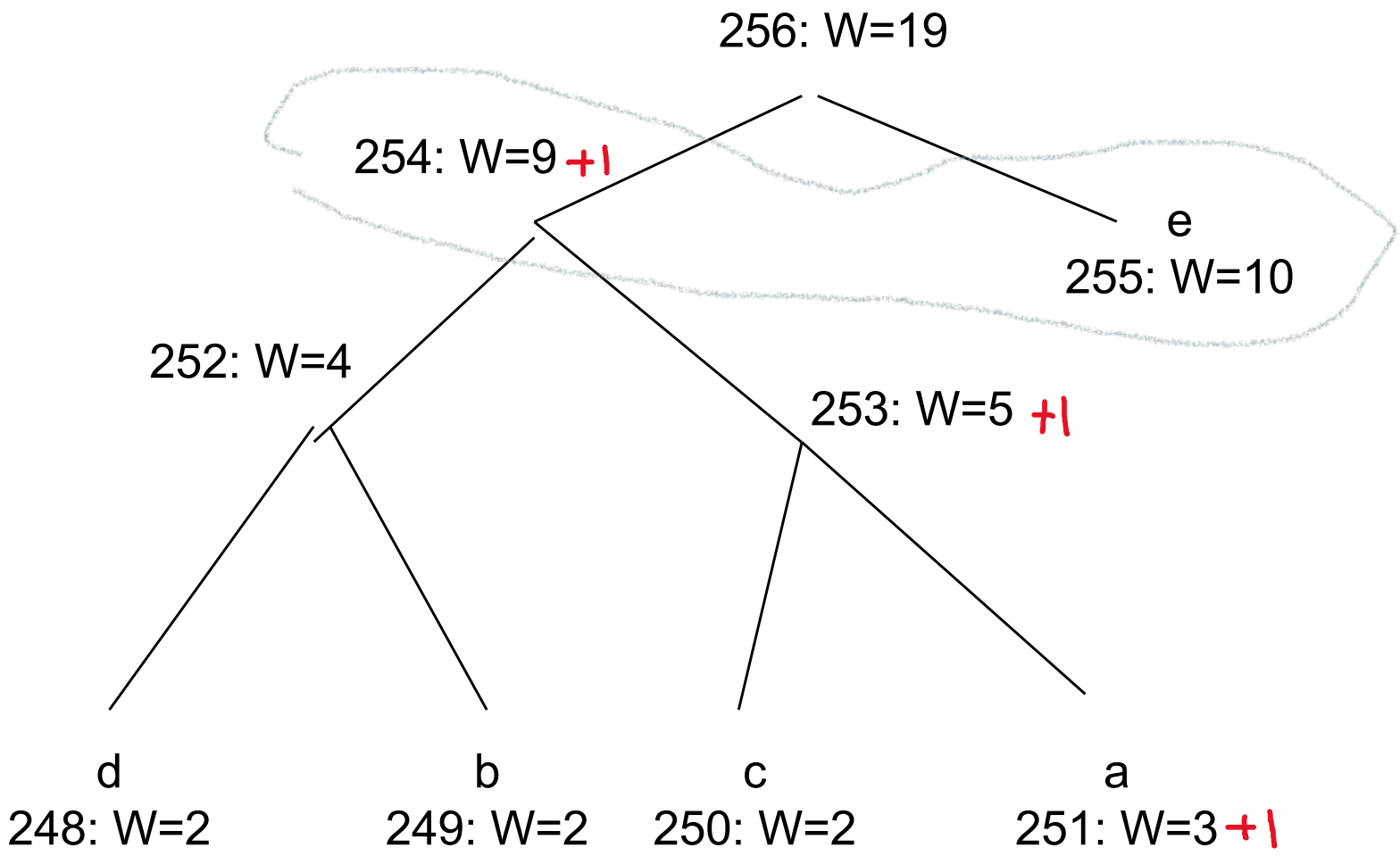
# More example



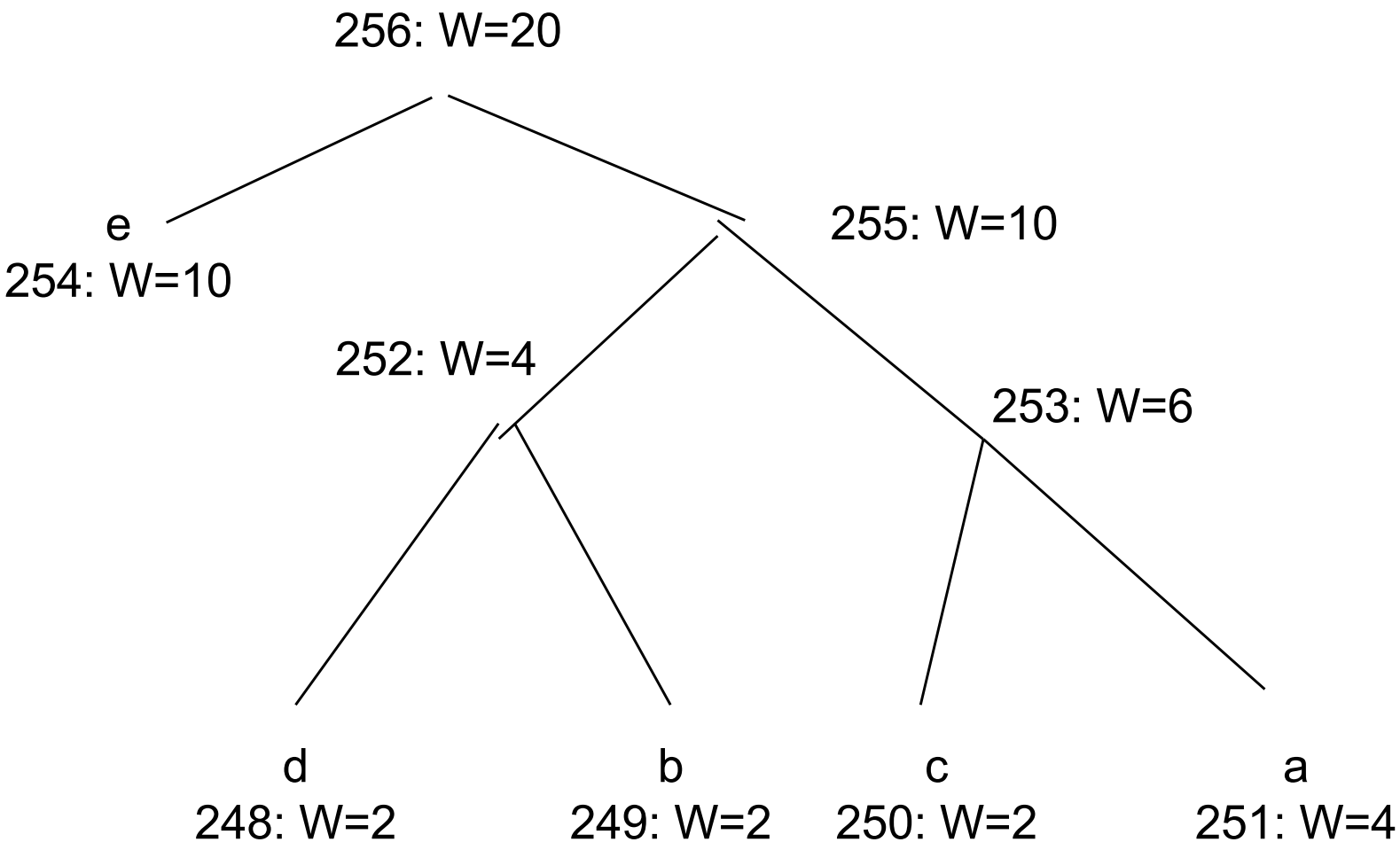
# More example



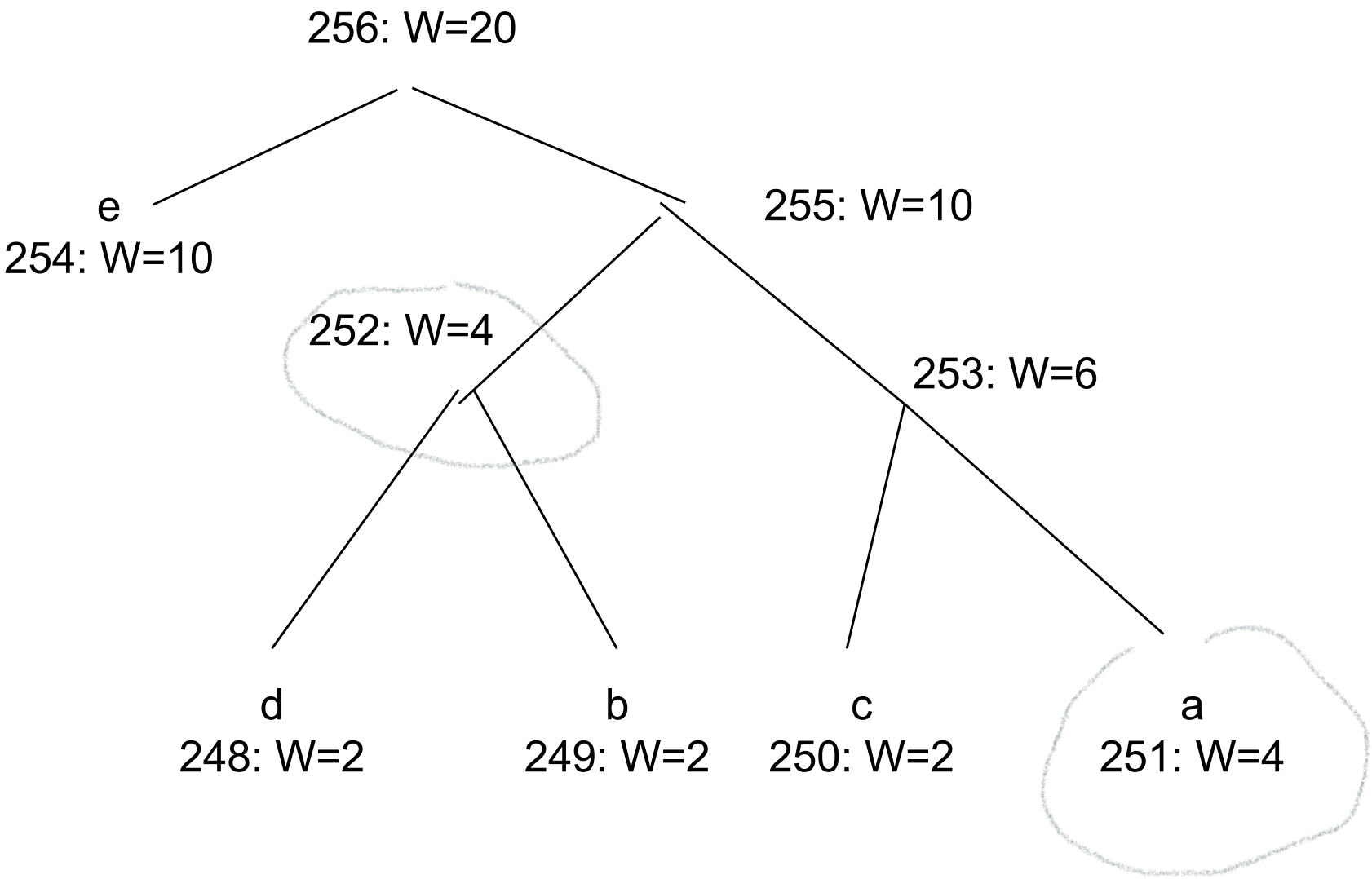
# More example



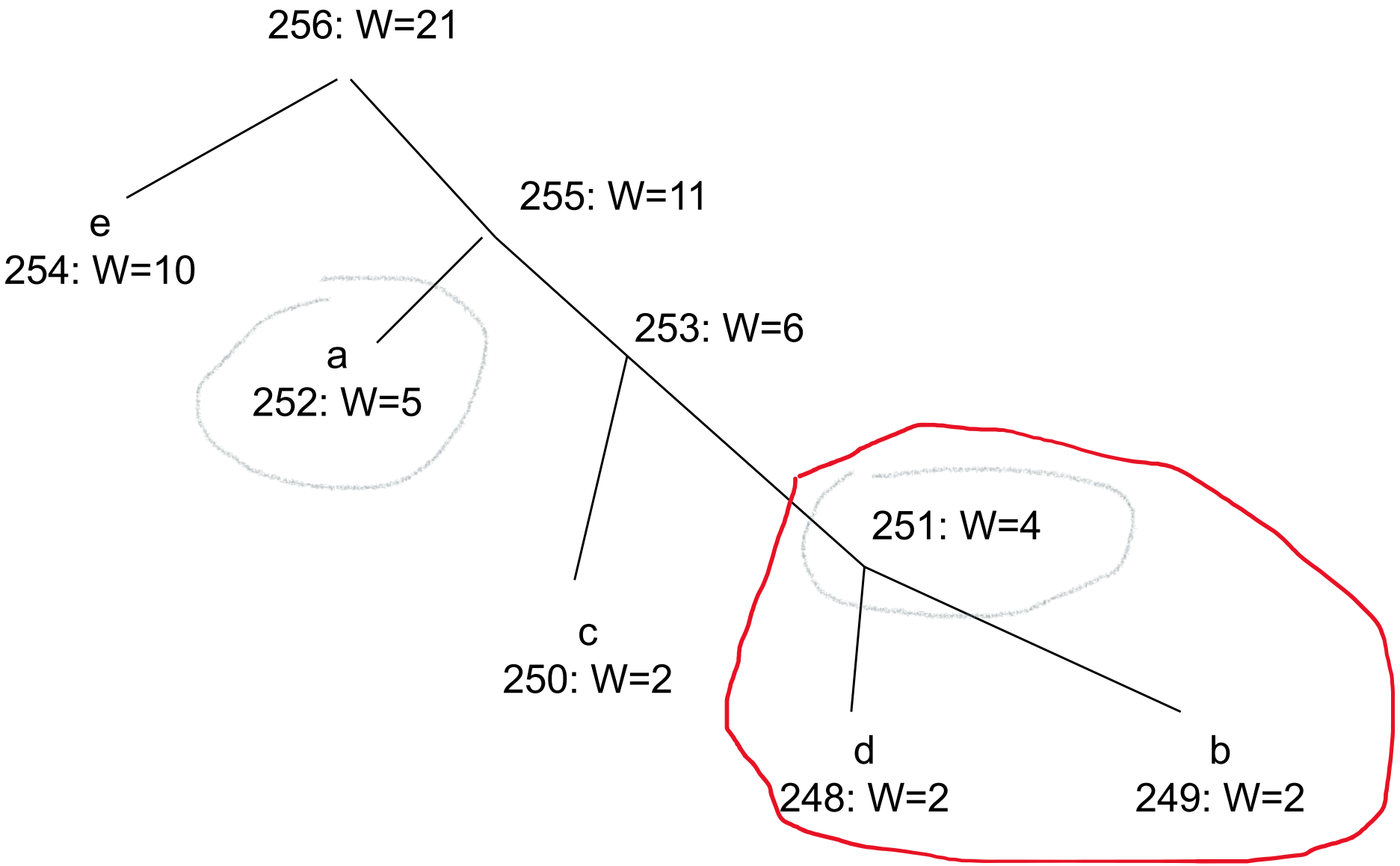
# More example



# More example



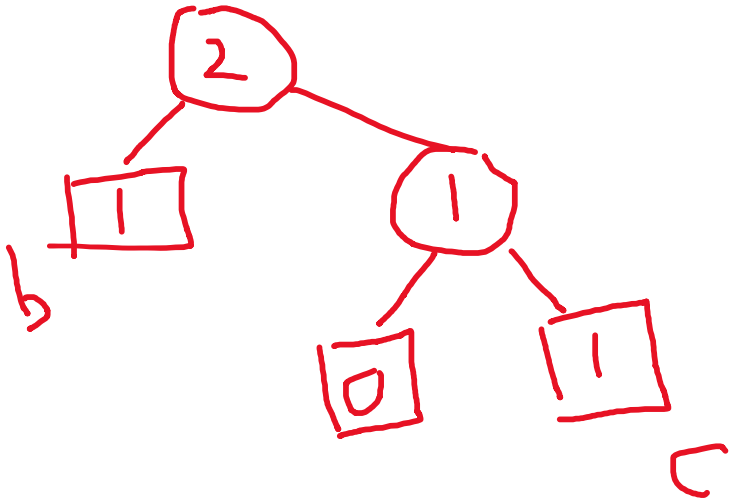
# More example



# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

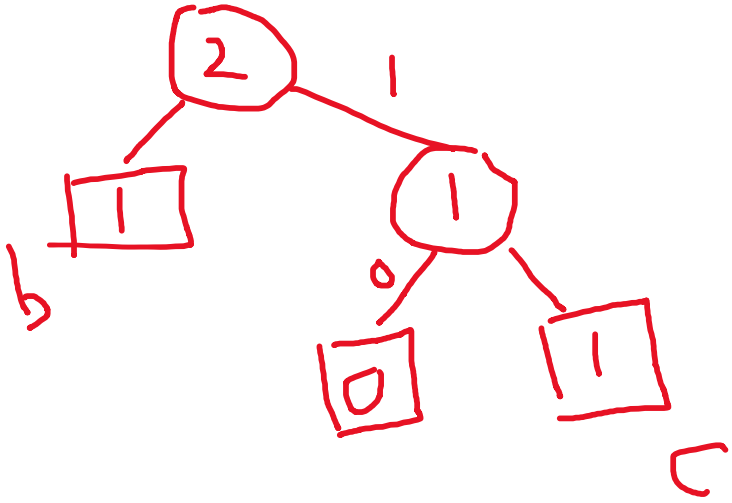



01100010 001100011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

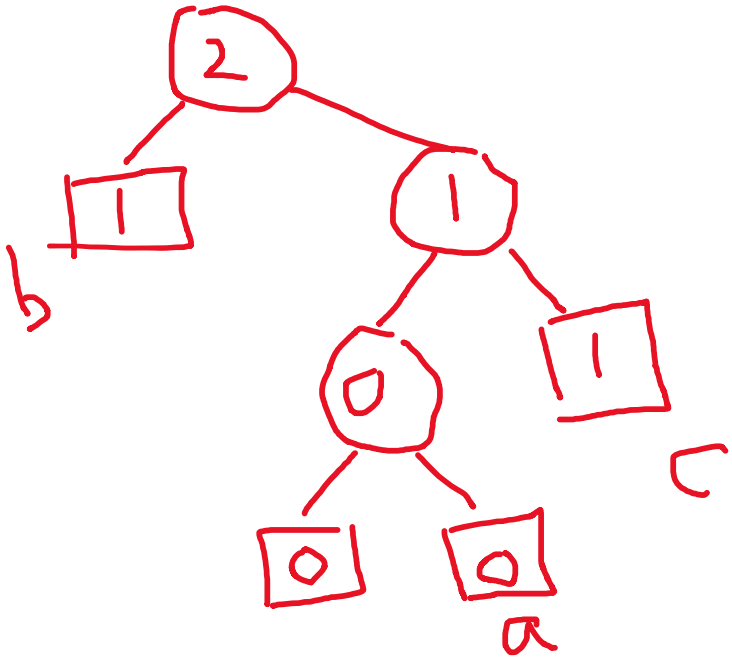



01100010 001100011 1001100001

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

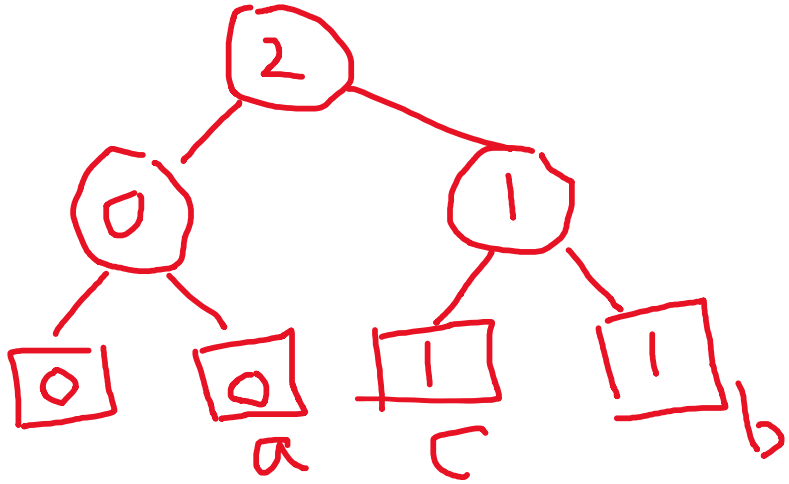



01100010 001100011 1001100001

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

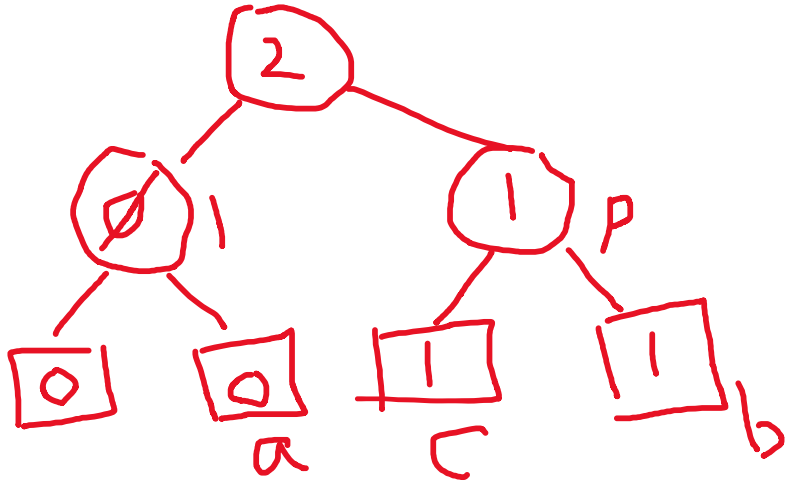



01100010 001100011 1001100001

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

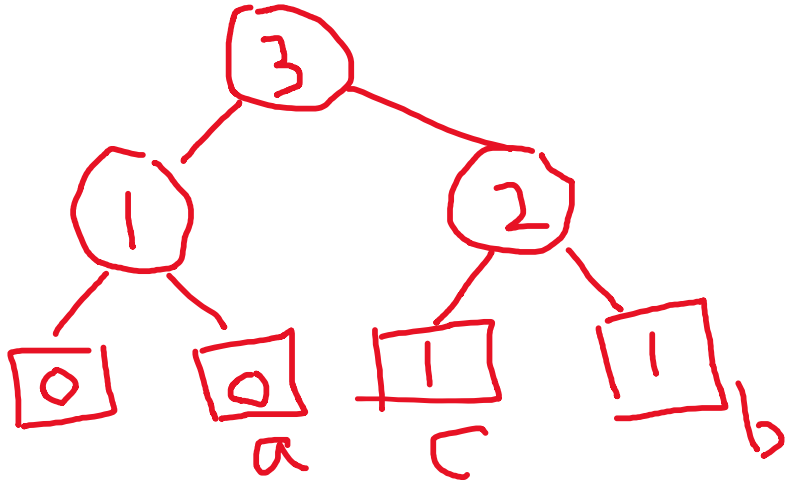



01100010 001100011 1001100001

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

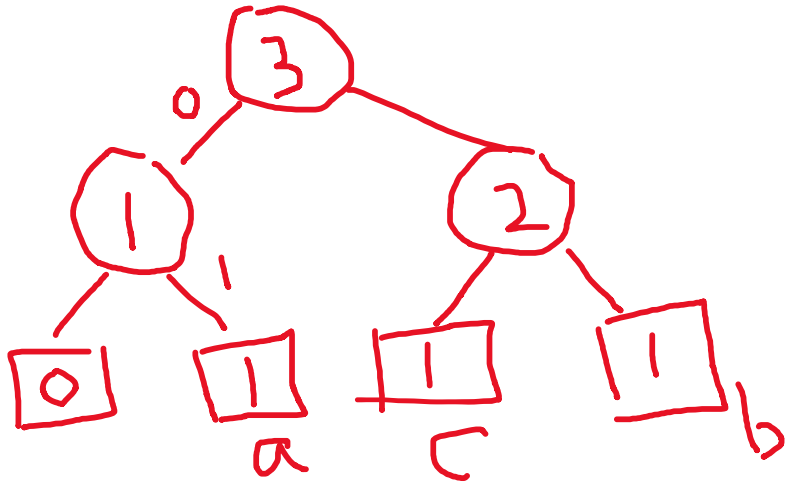



01100010 001100011 1001100001

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

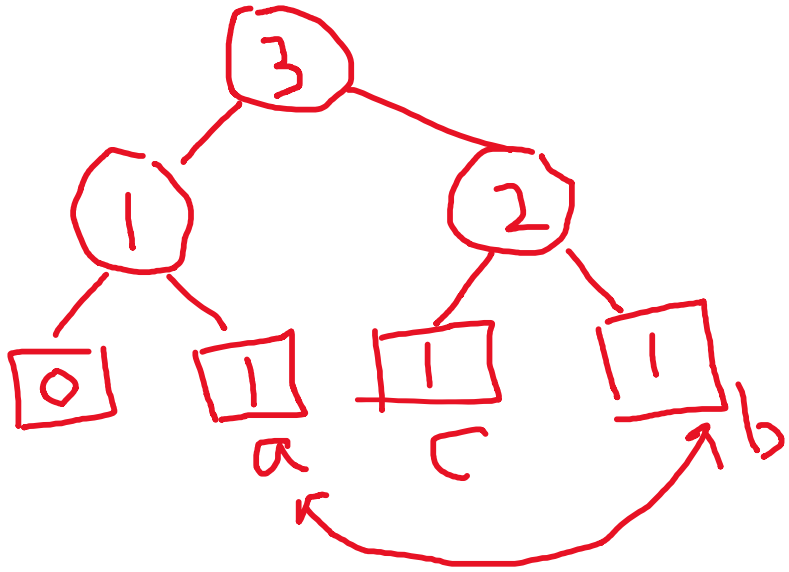



01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

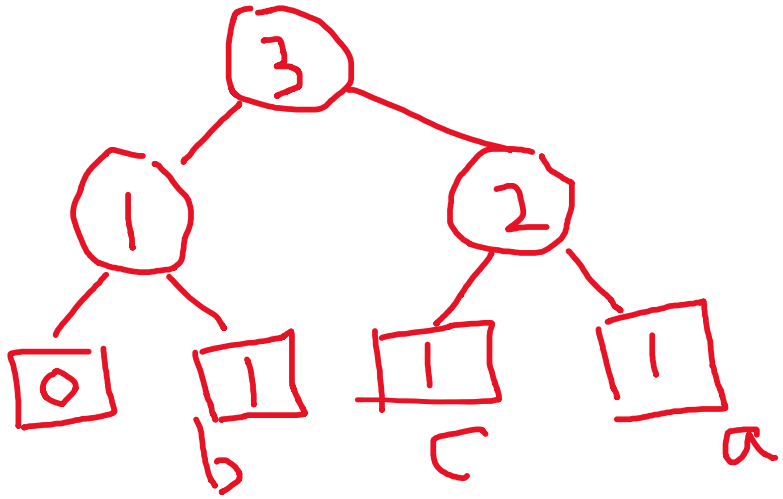



01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

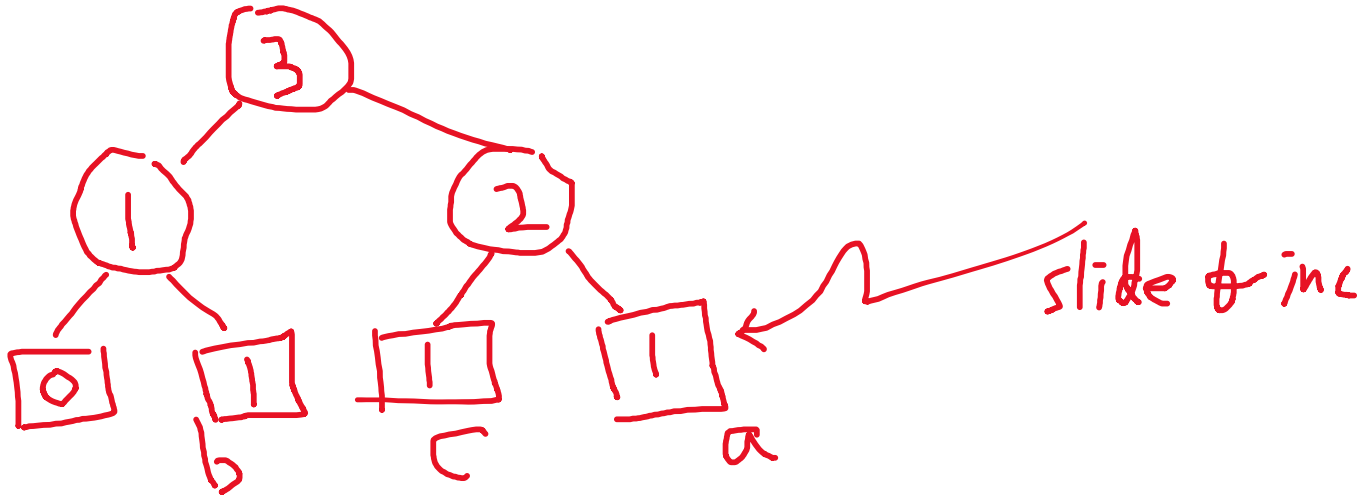



01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.




01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

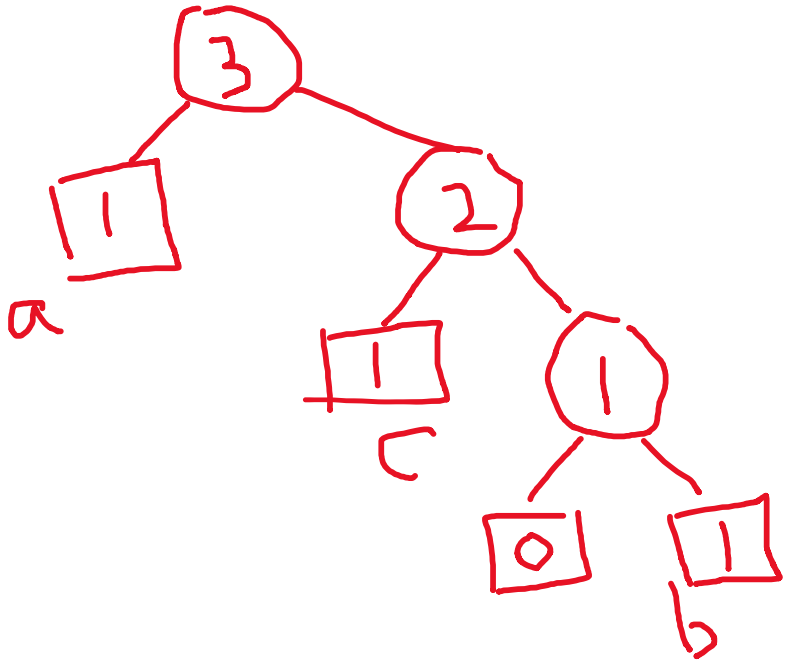
Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.


01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

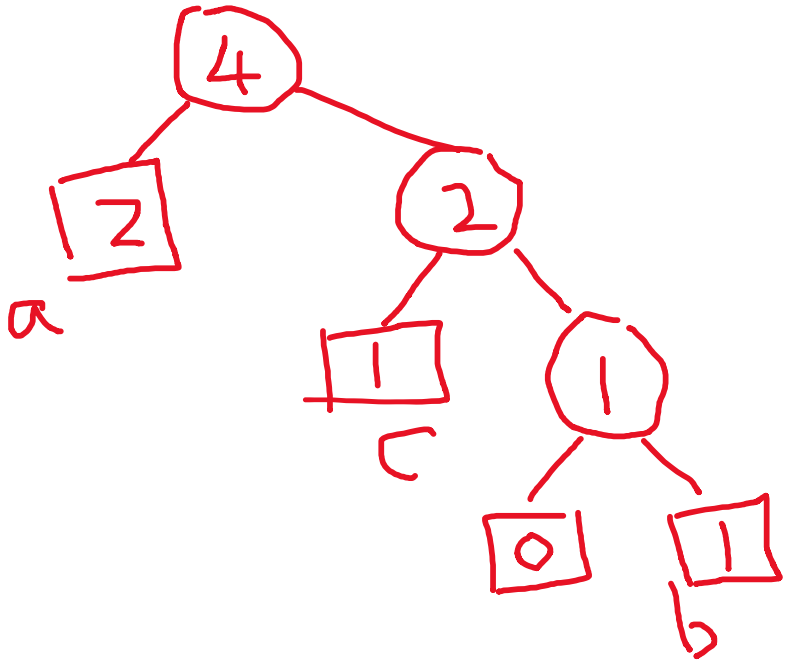



01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

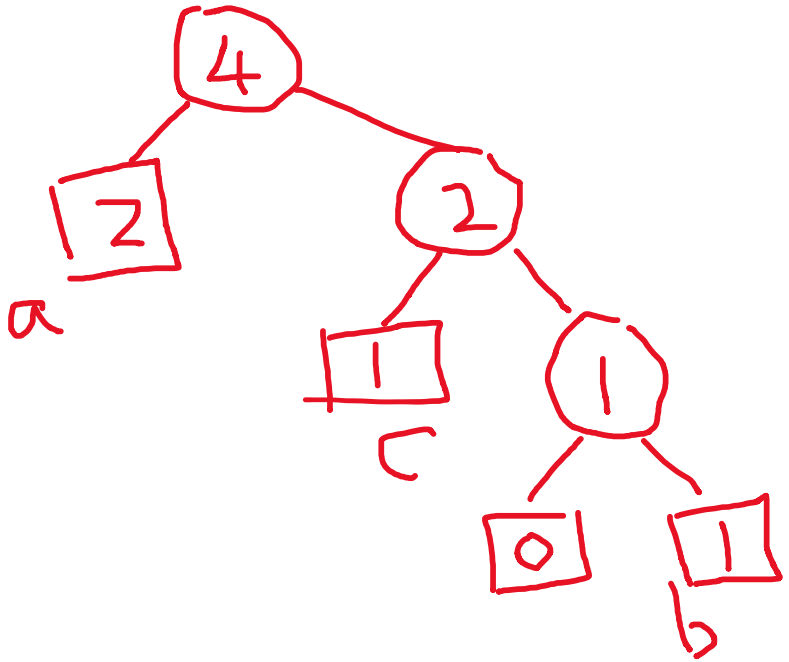



01100010 001100011 1001100001 01

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

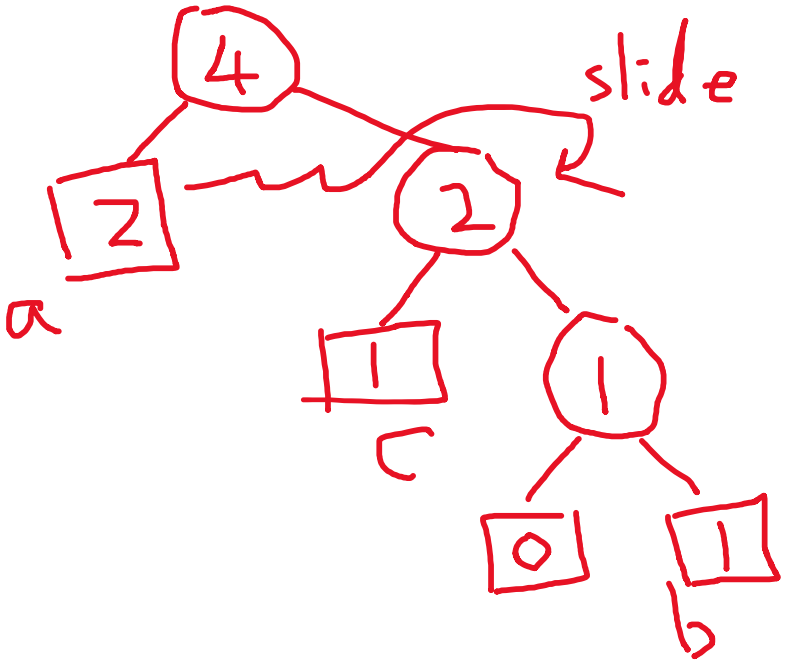



01100010 001100011 1001100001 01 0

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

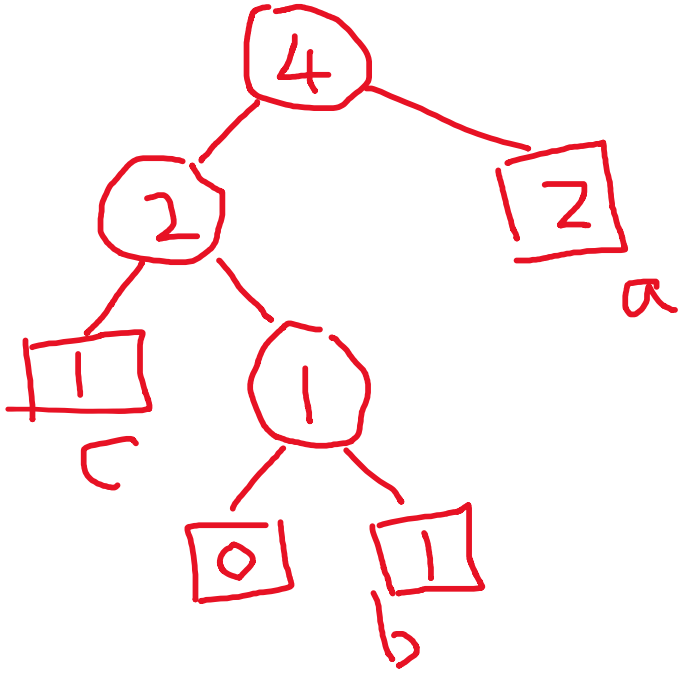



01100010 001100011 1001100001 01 0

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

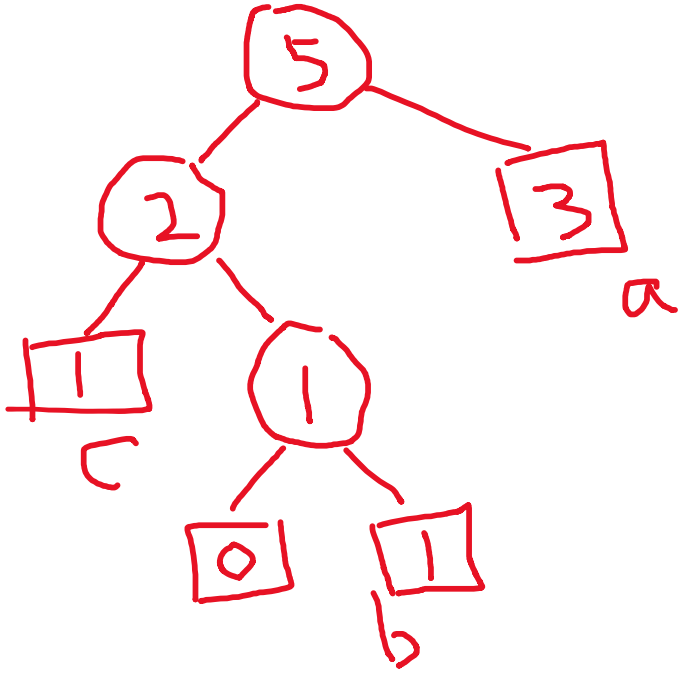



01100010 001100011 1001100001 01 0

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

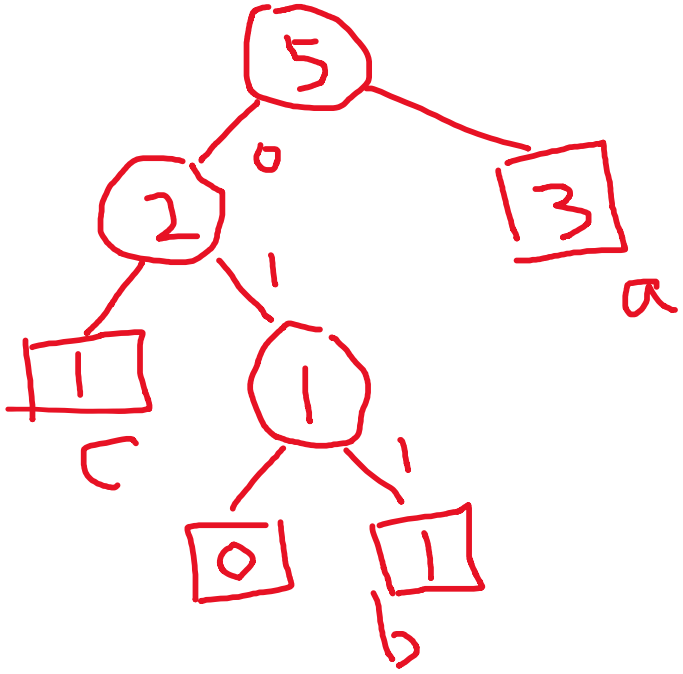



01100010 001100011 1001100001 01 0

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

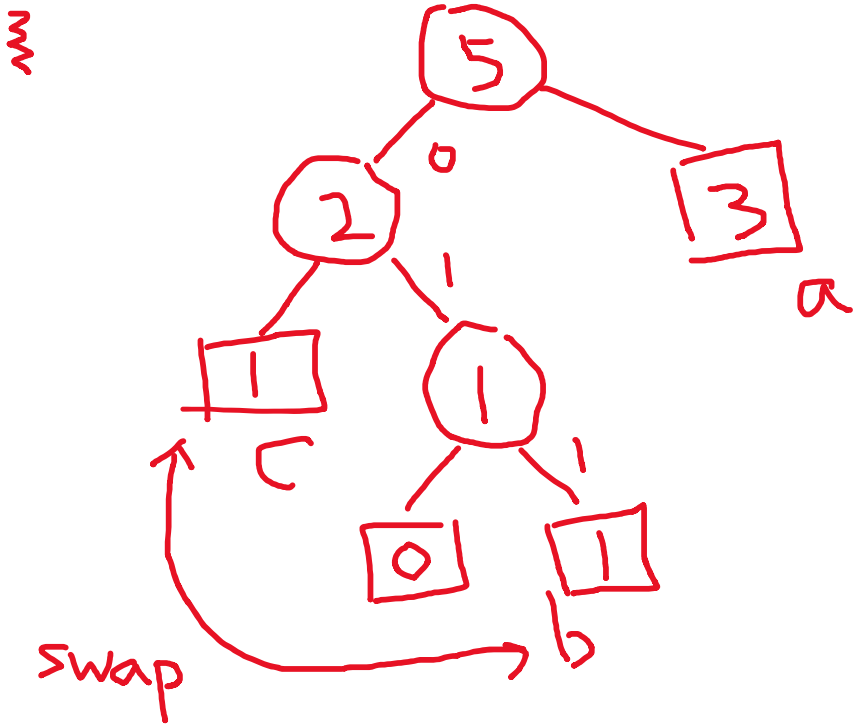



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

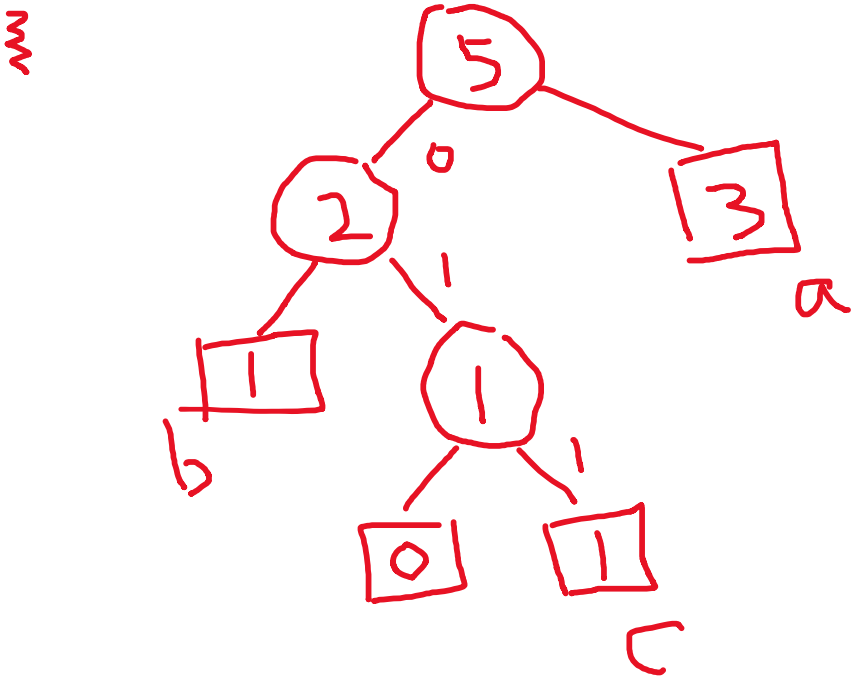



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

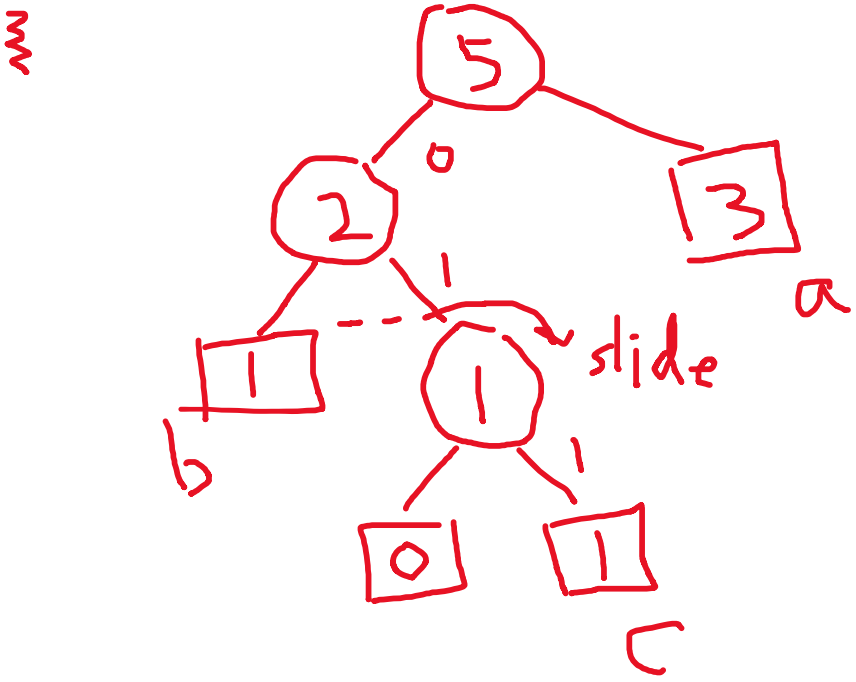



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

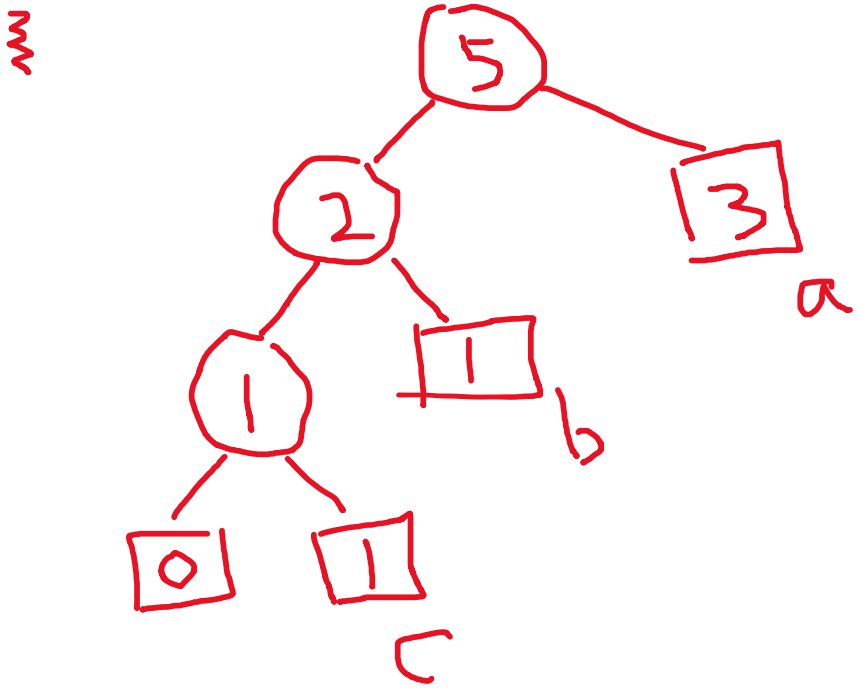



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

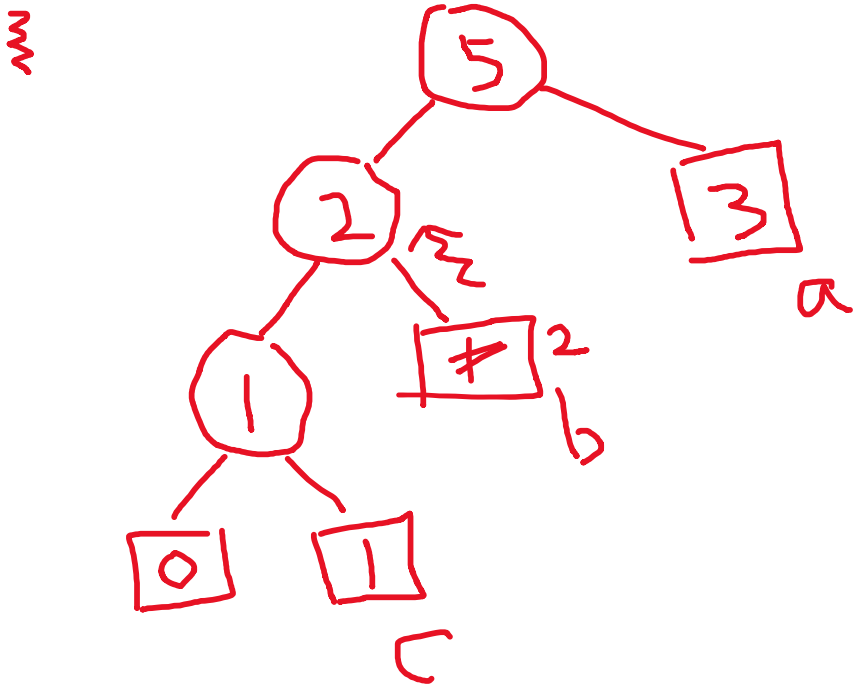



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

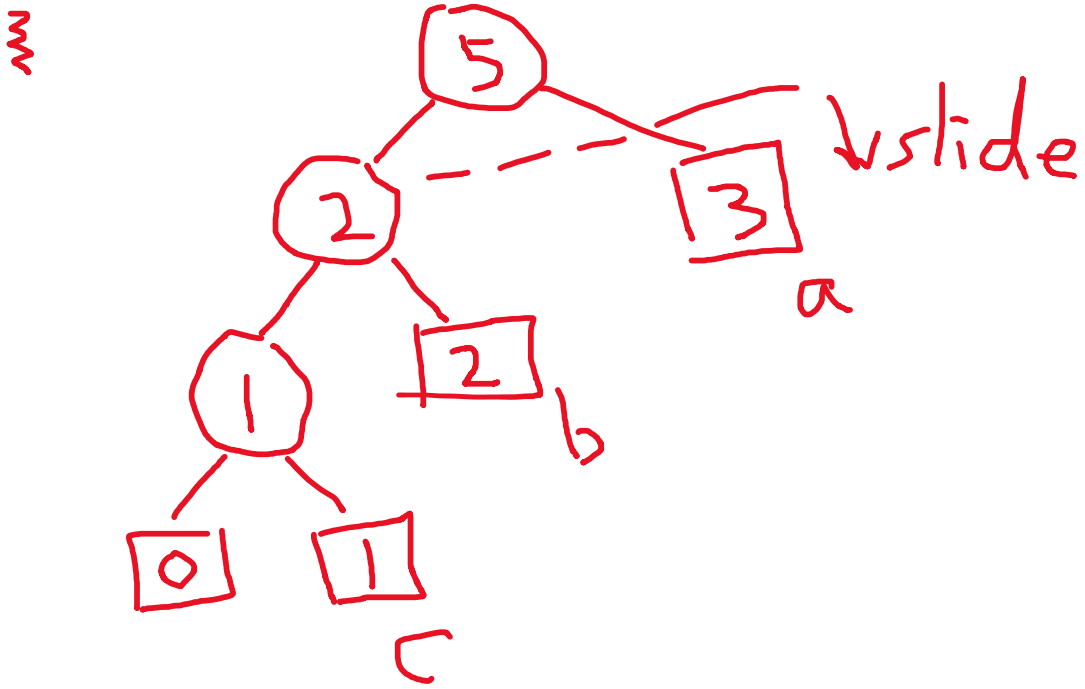



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

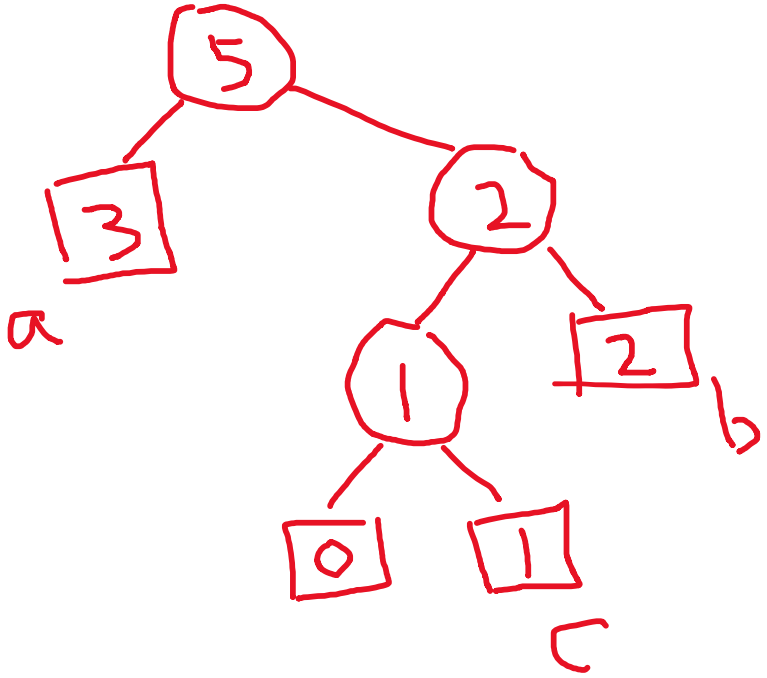



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

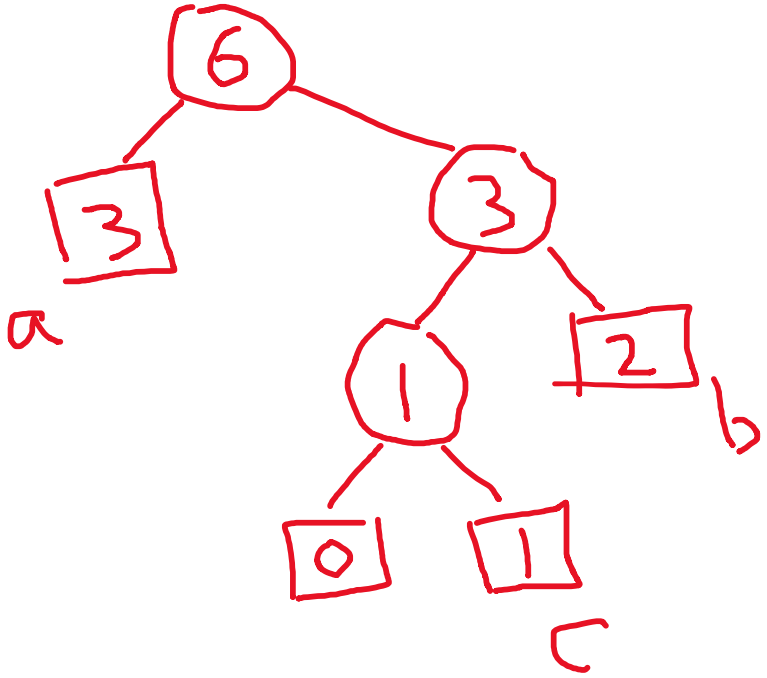



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

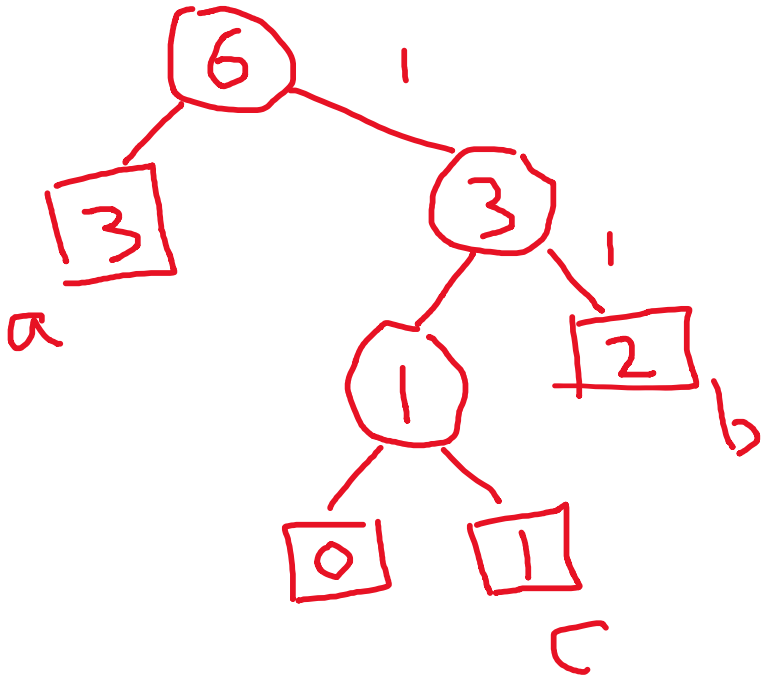



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

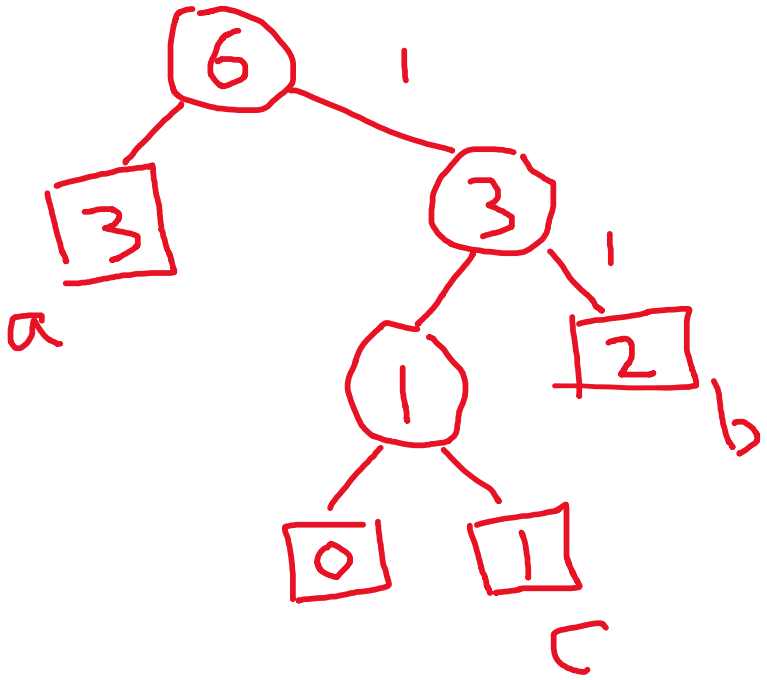



01100010 001100011 1001100001 01 0 011

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

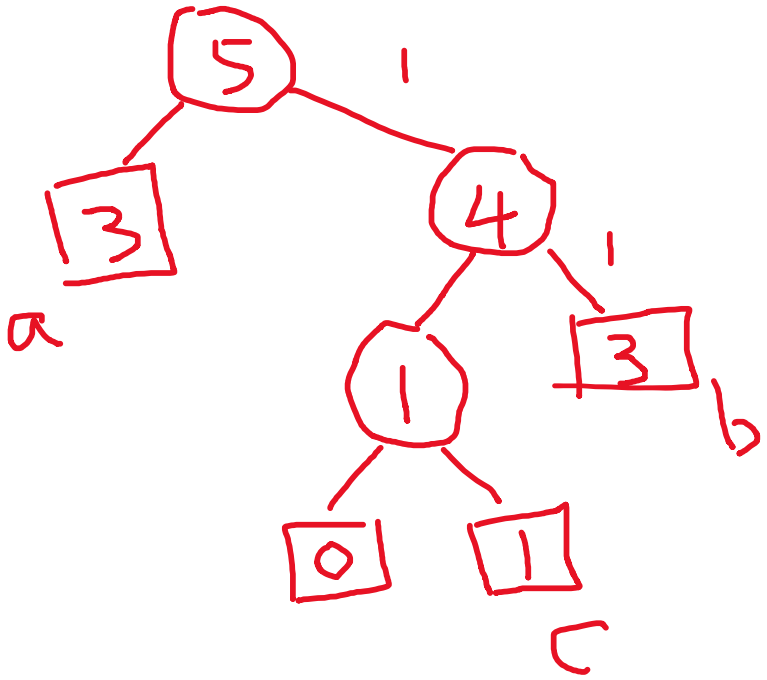



01100010 001100011 1001100001 01 0 011 11

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.

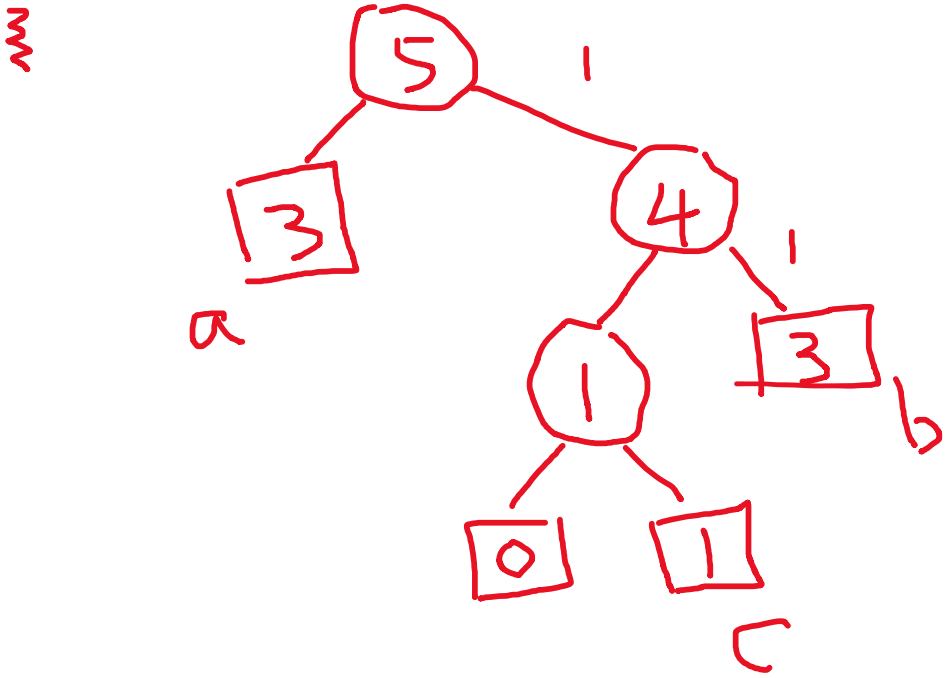



01100010 001100011 1001100001 01 0 011 11

# Adaptive Huffman (Ex2, Q2)

The initial coding before any transmission is: a=01100001, b=01100010, c=01100011.

Derive the encoded bitstream produced by the encoder for the string **bcaaabbb**.




01100010 001100011 1001100001 01 0 011 11 11

# Assignment 1 FAQ

```
%wagner> cat a.txt
BILL GATES%wagner>
%wagner> aencode < a.txt
  1
A 1
B 1
E 1
G 1
I 1
L 2
S 1
T 1
0.2572167752 0.2572167756
%wagner>
%wagner> cat a.txt | aencode | adecode > b.txt
%wagner> diff a.txt b.txt
%wagner>
```

[0.43, 0.43)

0.43 =< x < 0.43

?

Since adecode only read the first decimal number of the last line and ignore the rest of the line, we may modify the output above and use any number in the interval of [0.2572167752, 0.2572167756), then the decoder will still generate the same output. For example:

```
%wagner> cat c.txt
  1
A 1
B 1
E 1
G 1
I 1
L 2
S 1
T 1
0.2572167755 # This is a legit AC encoded value
%wagner>
%wagner> cat c.txt | adecode > d.txt
%wagner> diff a.txt d.txt
%wagner>
```