Lossless Compression-Statistical Model

Lossless Compression

- One important to note about entropy is that, unlike the thermodynamic measure of entropy, we can see no absolute number for the information content of a given message
- The probability figure we see is actually the probability for a given model that is kept inside the human brain. It is just a model, not an absolute number

Lossless Compression

- In order to compress data well,we need to select models that predict symbols with high probabilities
- A symbol that has a high probability has low information content and will need fewer bits to encode
- Lossless data compression is generally implemented using one of two different types of modeling
  - statistical or dictionary-based

Lossless Compression

- Statistical modeling reads in and encodes a single symbol at a time using the probability of that character’s appearance
  - Huffman coding
  - Shannon-Fano coding
  - Arithmetic coding
- Dictionary-based modeling uses a single code to replace strings of symbols
  - LZ77
  - LZ78
  - LZW

Minimum Redundancy Coding

- Given the probabilities, a table of codes could be constructed that has several important properties
- Different codes have different number of bits
- Codes for symbols with low probabilities have more bits, and codes for symbols with high probabilities have fewer bits

Shannon-Fano Algorithm

- Though the codes are of different bit lengths, they can be uniquely decodes
  - For a given list of symbols, develop a corresponding list of probabilities or frequency counts
  - Sort the lists of symbols in descending order
  - Divide the list into two parts, such that the total frequency counts of the two parts are as close as possible.
  - Assign 0,1 to the upper, lower parts, respectively
  - Recursively apply the same procedure to each of the two halves until each symbol has become a leaf

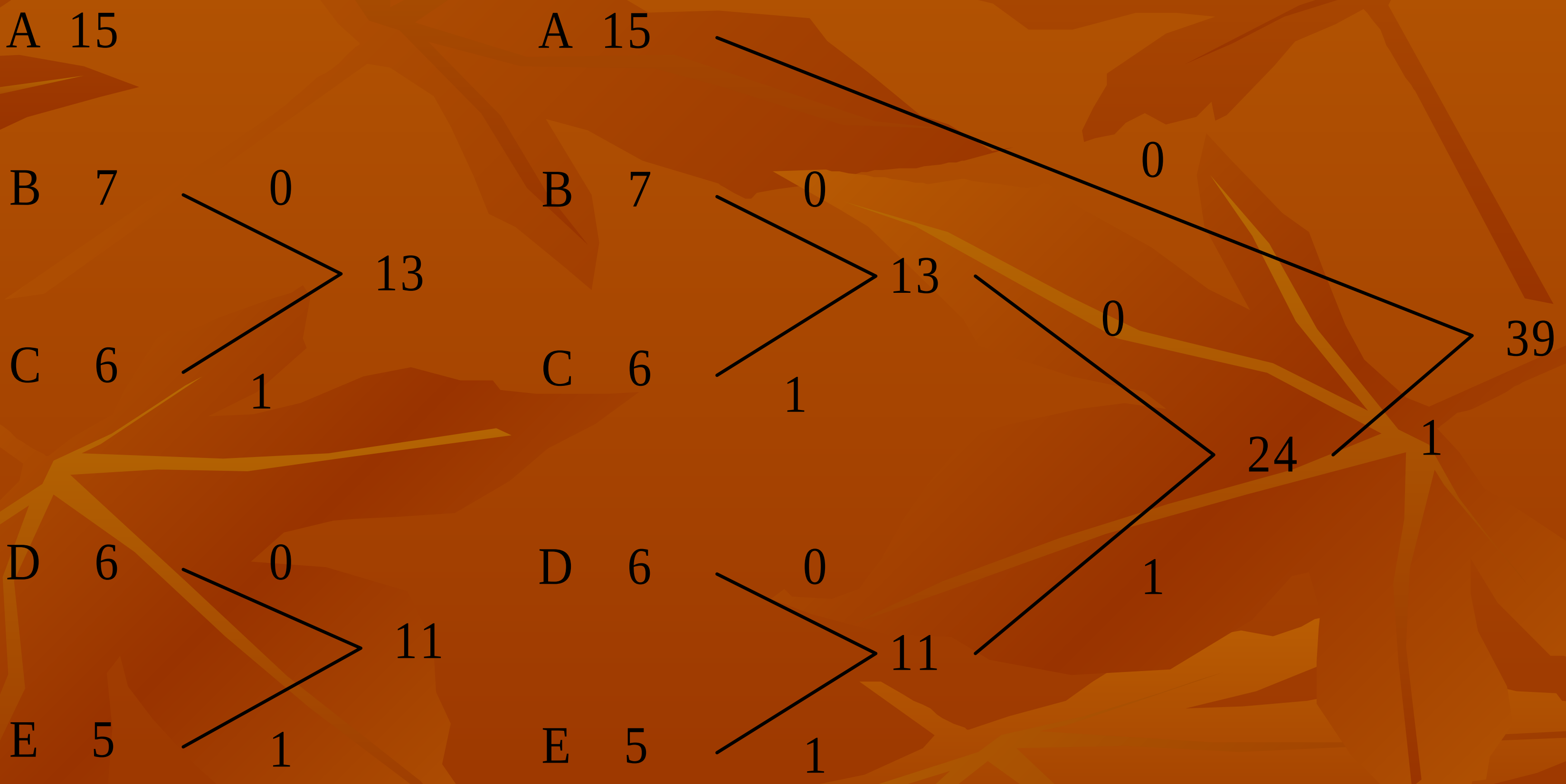
Shannon-Fano Algorithm

Symbol	Count	Code			
A	15	0	0		
					2nd division
B	7	0	1		
					First division
C	6	1	0		
					3rd division
D	5	1	1	0	
					4th division
E	5	1	1	1	

Huffman Coding

- Huffman codes are built from the bottom up, starting with the leaves of the tree and working progressively closer to the root
    - The two free nodes with the lowest weights are located
    - Create a parent node, whose weight is the sum of the two son nodes, for these two nodes
    - Let the newly created parent node be a free node and the two son nodes be removed from the list
    - Assign 0,1 to the two son nodes paths, arbitrarily
    - Repeat the Huffman Algorithm until only one free node is left

Huffman Coding



Huffman Coding

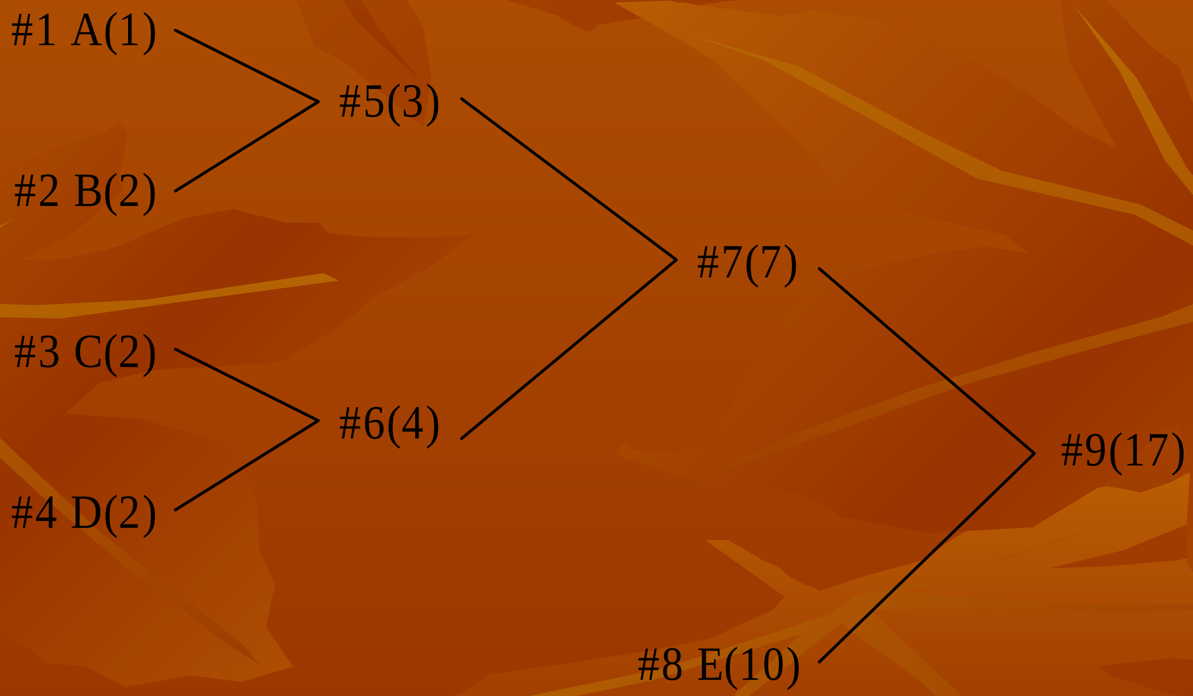
- Huffman coding will always at least equal the efficiency of Shannon-Fano coding, so it has become the predominate coding method of this type
- It was shown that Huffman coding cannot be improved or with any other integral bit-width coding stream
- JPEG, MPEG, and etc

Adaptive Huffman Coding

- Adaptive coding lets us use higher-order modeling without paying any penalty for added statistics
- It does this by adjusting the Huffman tree on the fly, based on data previously seen and having no knowledge about future statistics
- Sibling property
  - A tree is said to exhibit sibling property if the nodes can be listed in order of increasing weight and if every node appears adjacent to its sibling in the list

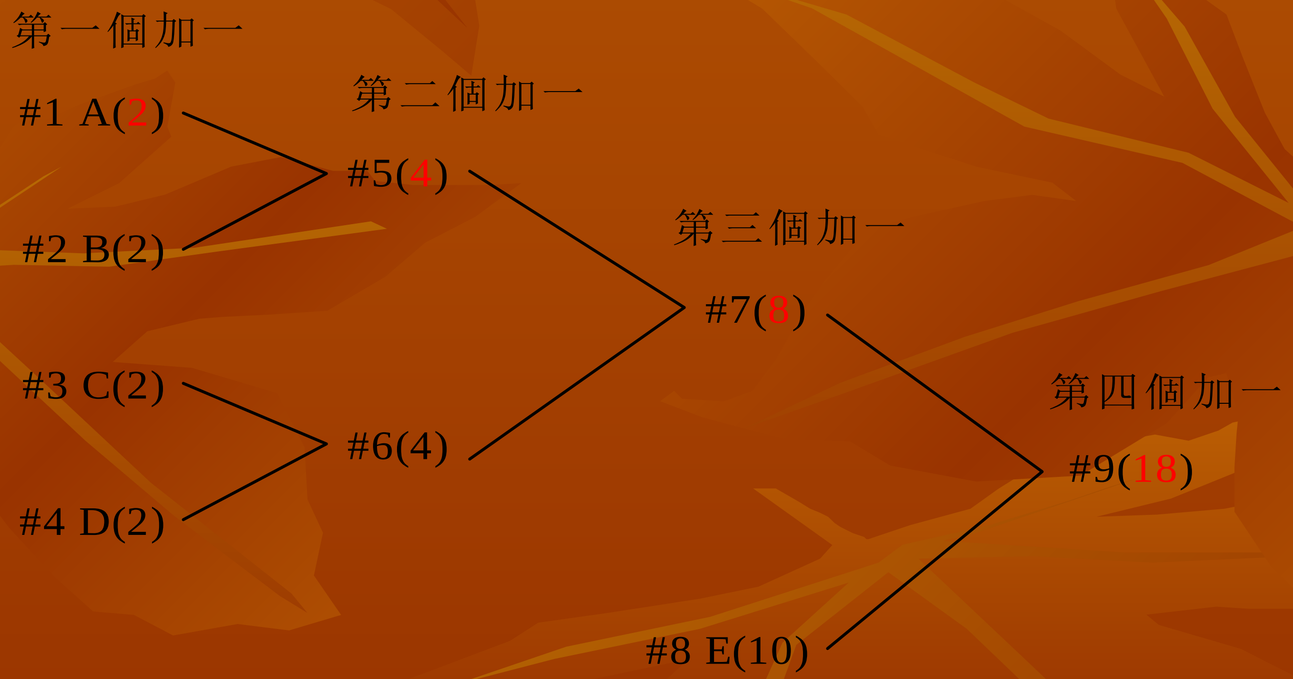
Adaptive Huffman Coding

- A binary tree is a Huffman tree if and only if it obeys the sibling property

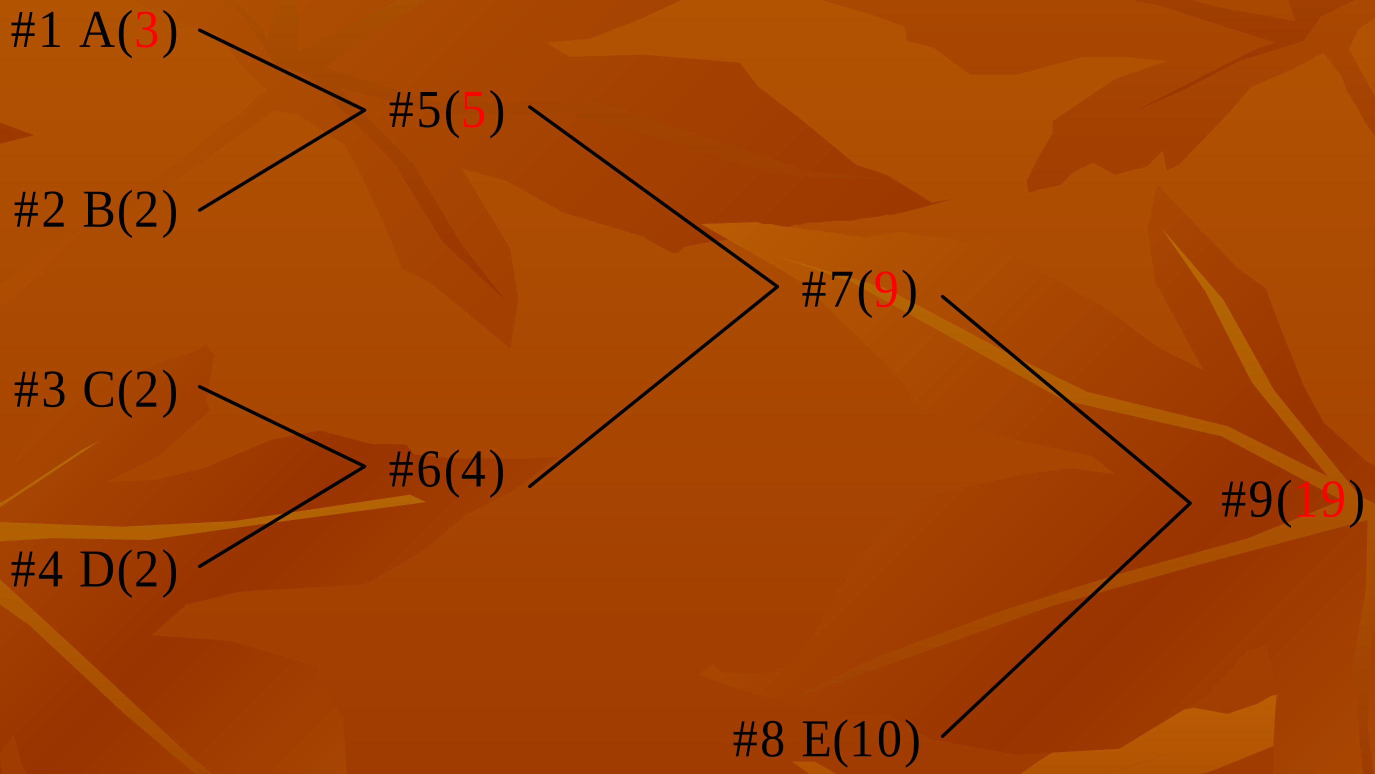


Adaptive Huffman Coding

- Thus, maintain the sibling property during the update assure that we have a Huffman tree before and after the counts are updated

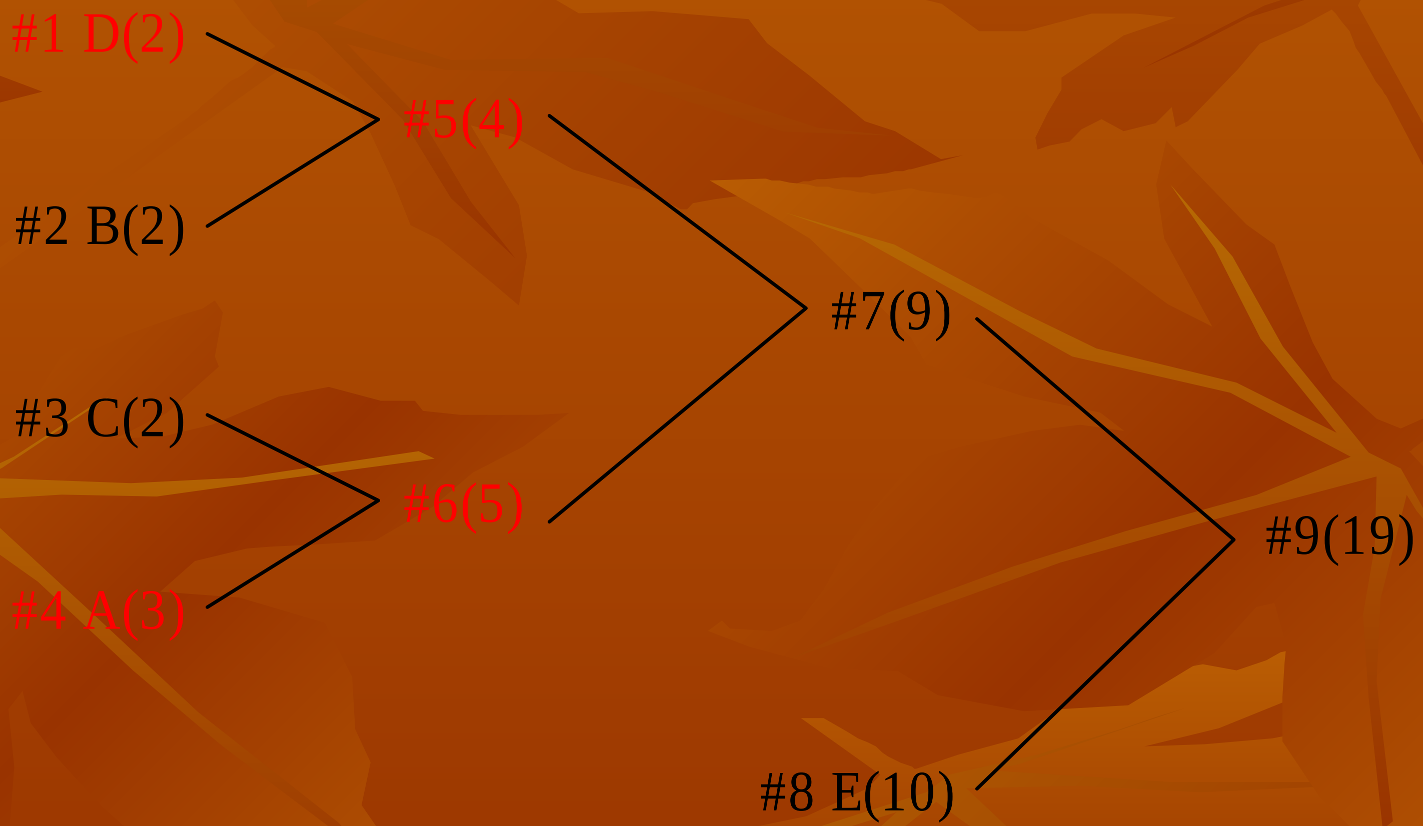


Adaptive Huffman Coding



A tree that does not satisfy sibling property

Adaptive Huffman Coding



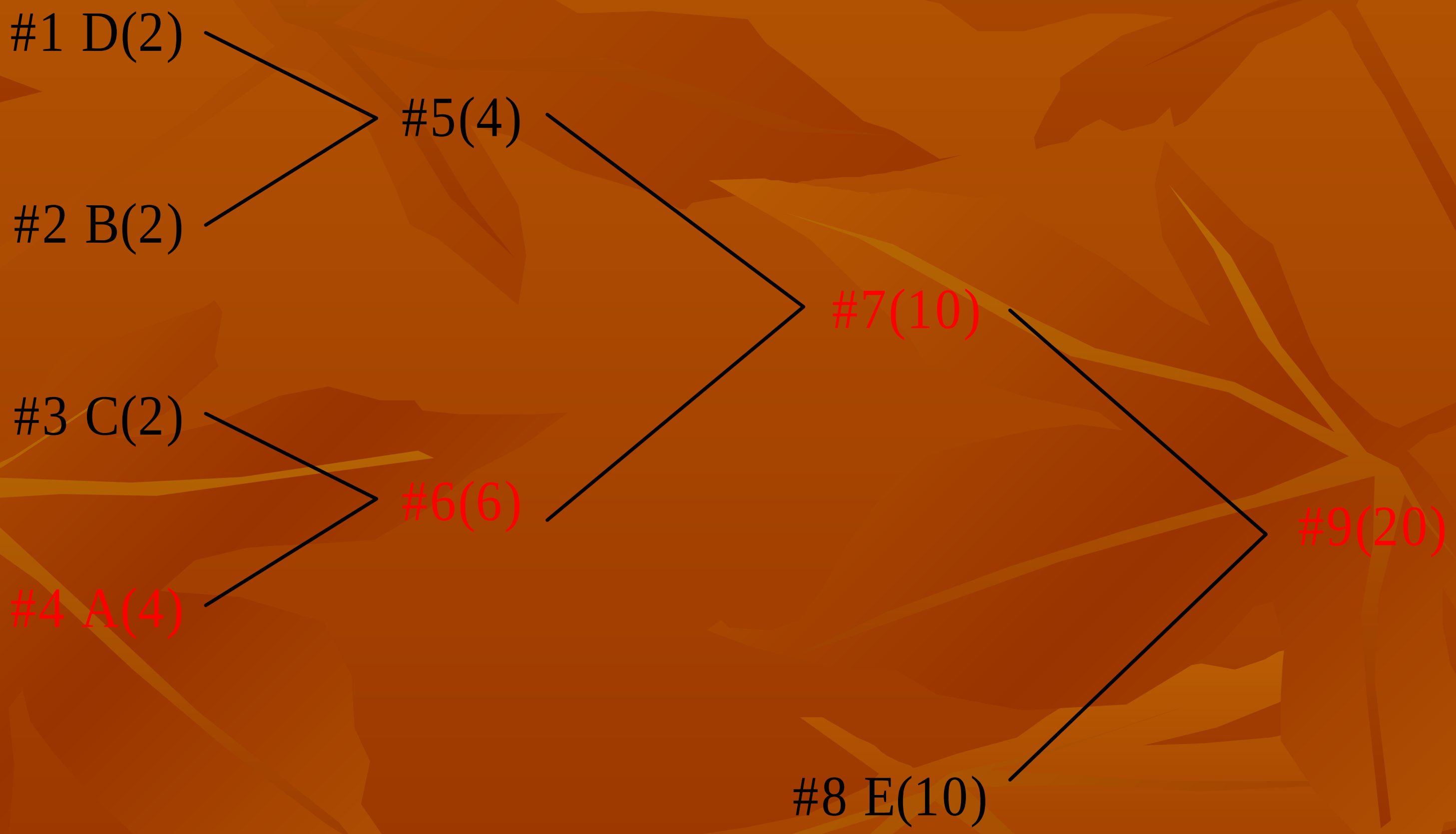
Adaptive Huffman Coding

- If the newly incremented node now has a weight of *w*+1, the next higher node will have a weight of *w* when updating is needed
- There may be more nodes after the next higher one that have a value of *w* as well. The update (swap) procedure moves up the node list till it finds the last node with a weight of *w*
- That node is swapped with the node with weight *w*+1

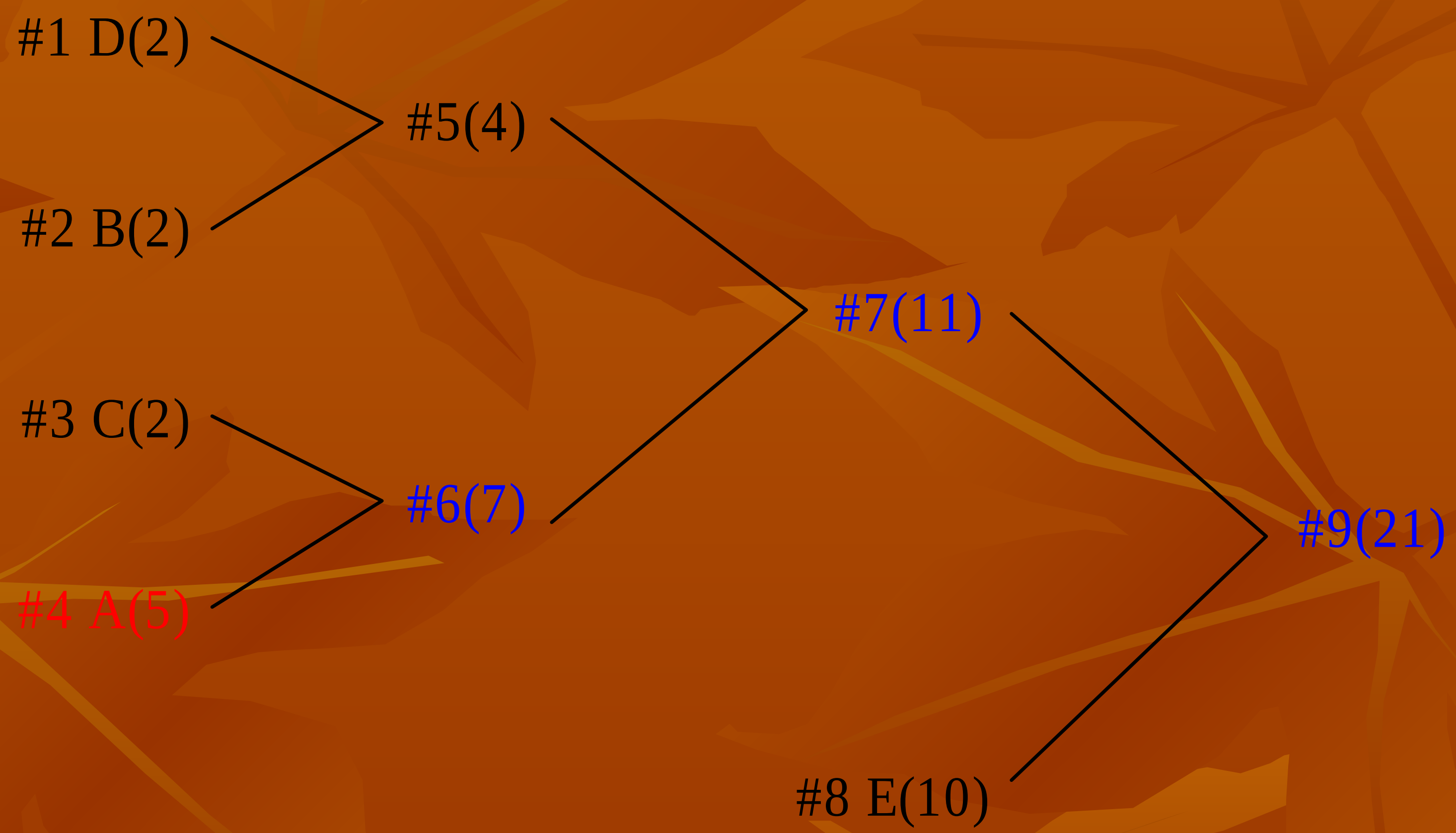
Adaptive Huffman Coding

- In this example, D is swapped with A
- The next node to be incremented will be the new parent of the incremented node. In this example, it is node #6
- As each node is incremented, a check is performed for correct ordering. A swap is performed if necessary

Adaptive Huffman Coding

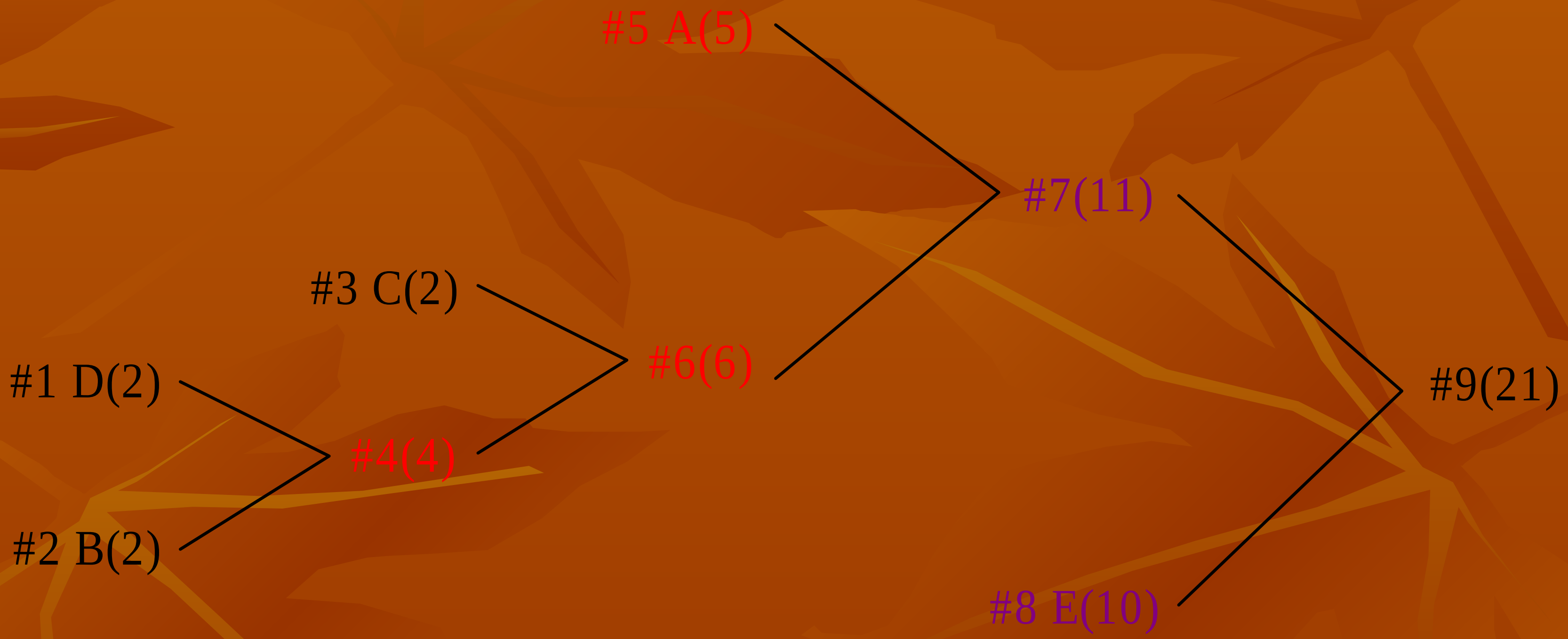


Adaptive Huffman Coding



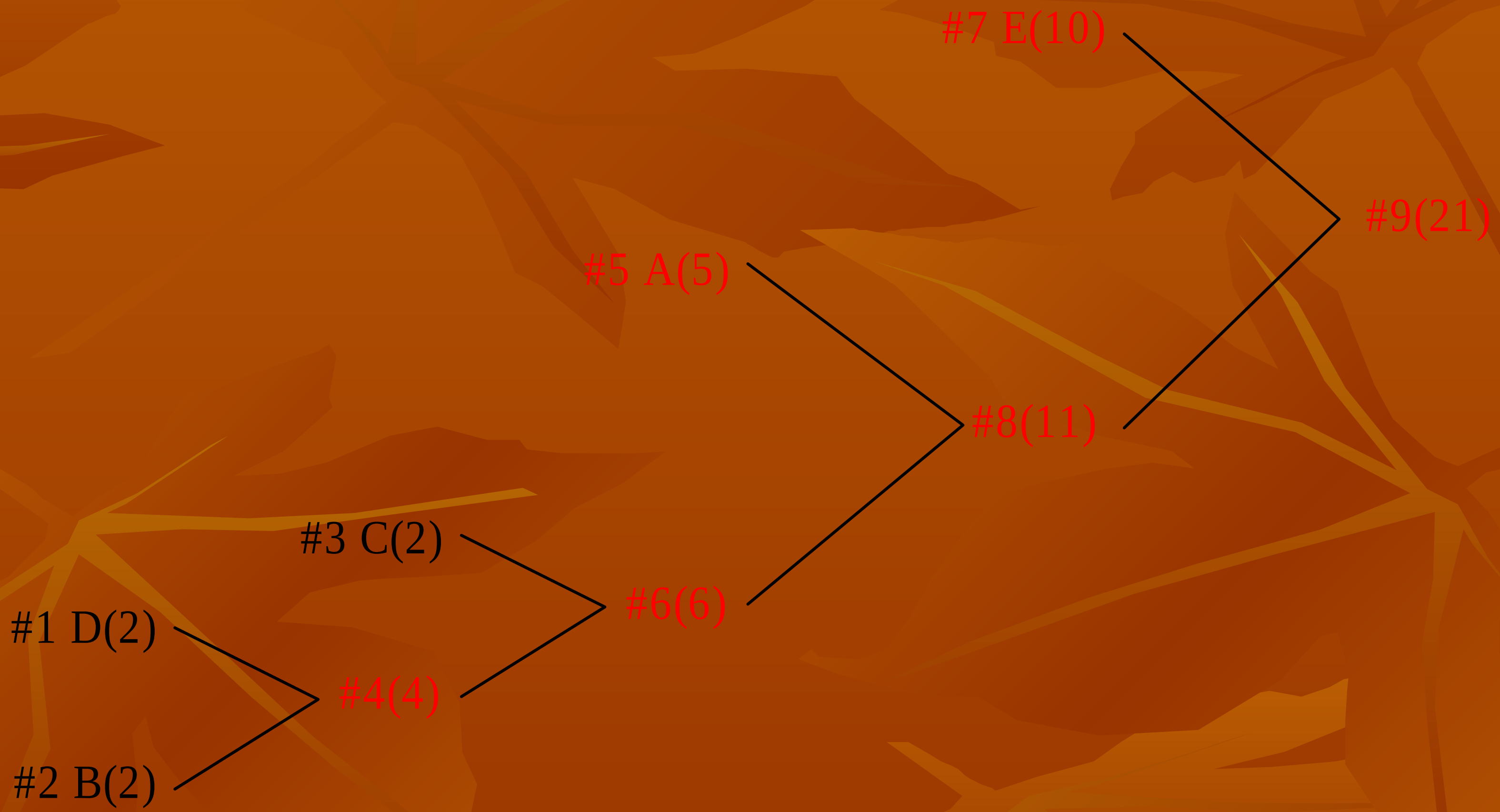
Not a Huffman tree

Adaptive Huffman Coding



Still not a Huffman tree

Adaptive Huffman Coding



Now, it is a Huffman tree again

Adaptive Huffman Coding

- Note that the code length for A changes from 3 to 2
- And the code length for B and D changes from 3 to 4

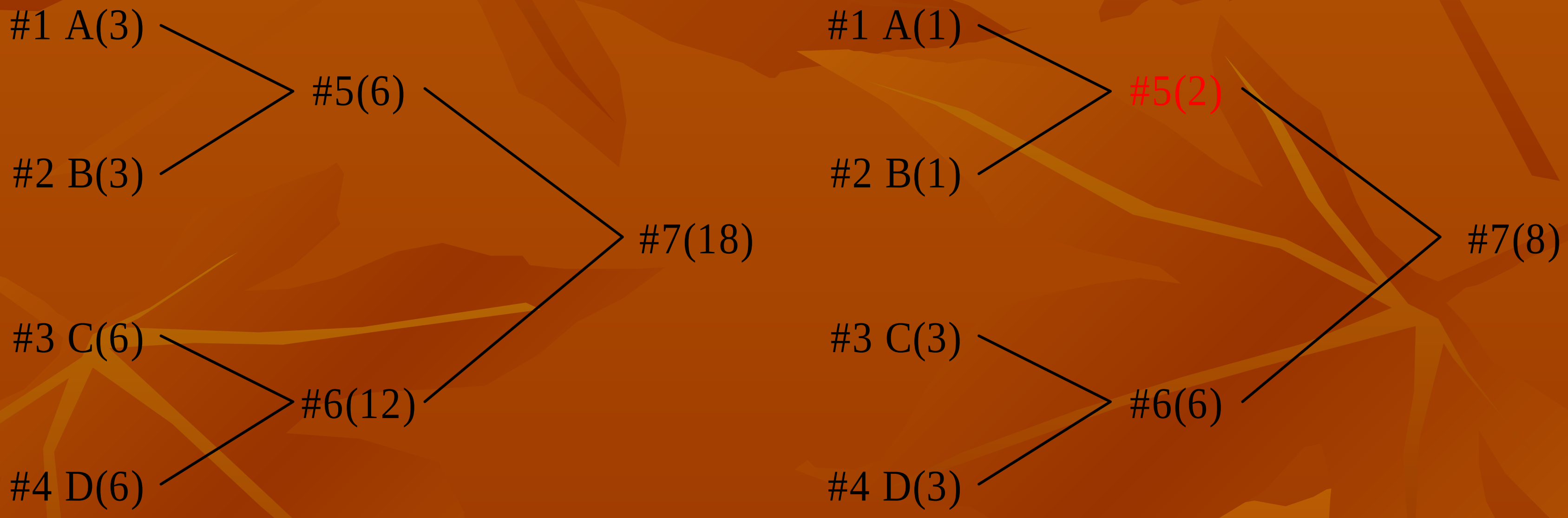
Adaptive Huffman Coding

- Initialization
  - Create a Huffman tree with two symbol EOF and ESCAPE with weight being 1
  - Let C be the incoming character
  - If C is in the Huffman tree, then transmit the Huffman code for C and update the Huffman tree, else transmit the Huffman code for ESCAPE and the plain 8-bit character for C, and then update the Huffman tree for ESCAPE and C

Adaptive Huffman Coding

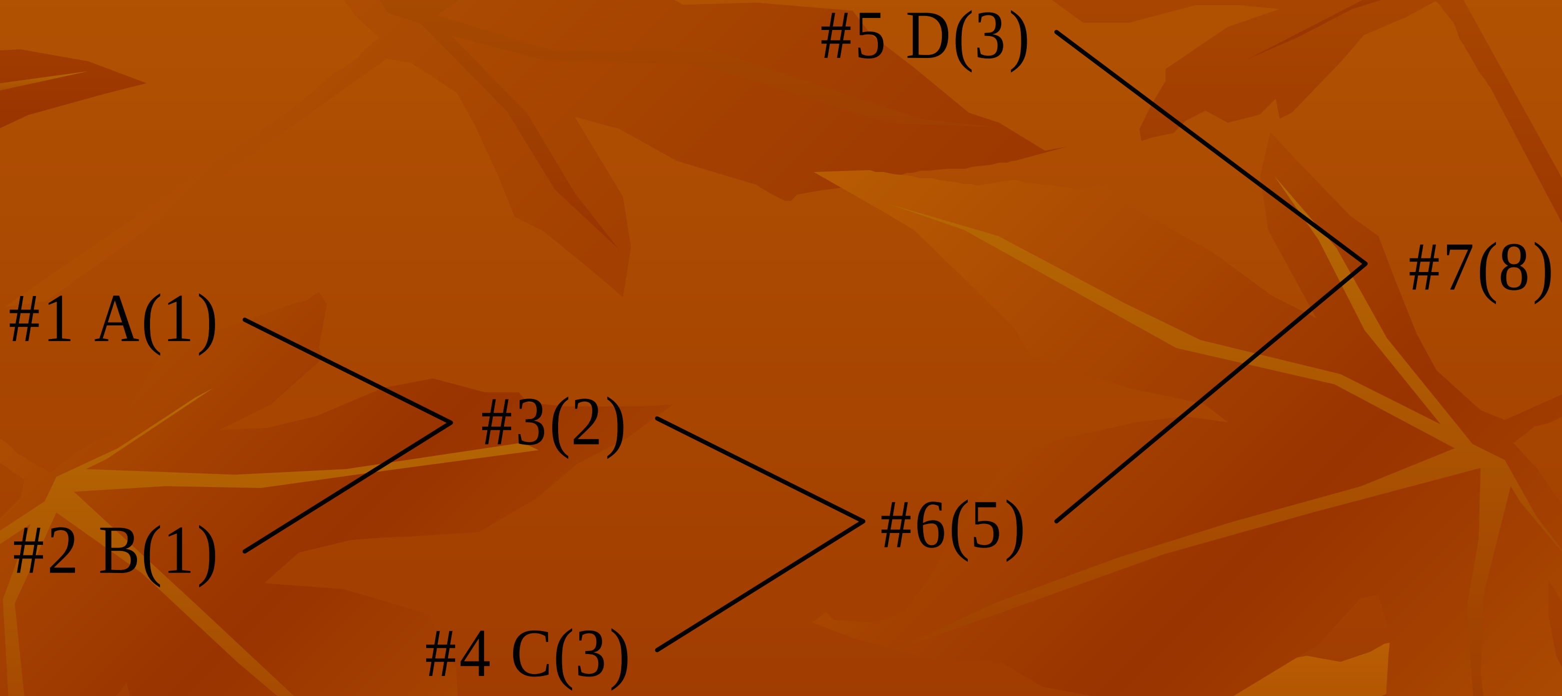
- The overflow problem
  - the weight of the root exceeds the capacity as the program progresses long enough
  - The longest possible Huffman code exceeds the range of the integer used as a stack
    - It is needed because the encoding and decoding phases treat the Huffman tree path in reverse order
- Solution
  - Recalling the weight by a factor of 1/2
  - It can possibly reshape the tree

Adaptive Huffman Coding



Problems occur when divided by 2

Adaptive Huffman Coding



Rebuilding the tree is sometimes needed

Arithmetic Coding

- Huffman coding has been proven the best fixed length coding method available
- Yet, since Huffman codes have to be an integral number of bits long, while the entropy value of a symbol may (as a matter of fact, almost always so) be a faction number, theoretical possible compressed message cannot be achieved

Arithmetic Coding

- For example, if a statistical method assign 90% probability to a given character, the optimal code size would be 0.15 bits
- The Huffman coding system would probably assign a 1-bit code to the symbol, which is six times longer than necessary
- Arithmetic coding bypasses the idea of replacing an input symbol with a specific code. It replaces a stream of input symbols with a single floating-point output number

Character	probability	Range
^(space)	1/10	$0.00 \le r < 0.10$
A	1/10	$0.10 \le r < 0.20$
B	1/10	$0.20 \le r < 0.30$
E	1/10	$0.30 \le r < 0.40$
G	1/10	$0.40 \le r < 0.50$
I	1/10	$0.50 \le r < 0.60$
L	2/10	$0.60 \le r < 0.80$
S	1/10	$0.80 \le r < 0.90$
T	1/10	$0.90 \le r < 1.00$

Suppose that we want to encode the message

BILL GATES

Arithmetic Coding

- To encode the first character B properly, the final coded message has to be a number greater than or equal to 0.20 and less than 0.30
- After the first character is encoded, the low end for the range is changed from 0.00 to 0.20 and the high end for the range is changed from 1.00 to 0.30
- The next character to be encoded, the letter I, owns the range 0.50 to 0.60 in the new subrange of 0.20 to 0.30
- So, the new encoded number will fall somewhere in the 50th to 60th percentile of the currently established.
- Thus, this number is further restricted to 0.25 to 0.26

Arithmetic Coding

- Encoding algorithm for arithmetic coding

```
Low = 0.0 ; high =1.0 ;
while not EOF do
   range = high - low ; read(c) ;
   high = low + range×high_range(c) ;
   low = low + range×low_range(c) ;
enddo
output(low);
```

Arithmetic Coding

New character	Low value	high value
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
^(space)	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Arithmetic Coding

- Decoding is the inverse process
- Since 0.2572167752 falls between 0.2 and 0.3, the first character must be ‘B’
- Removing the effect of ‘B’ from 0.2572167752 by first subtracting the low value of B, 0.2, giving 0.0572167752
- Then divided by the width of the range of ‘B’, 0.1. This gives a value of 0.572167752

Arithmetic Coding

- Decoding algorithm

```
r = input_code
repeat
    search c such that r falls in its range ;
    output(c) ;
    r = r - low_range(c) ;
    r = r/(high_range(c) - low_range(c));
until EOF or the length of the message is reached
```

r	c	Low	High	range
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	^(space)	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

Arithmetic Coding

- In summary, the encoding process is simply one of narrowing the range of possible numbers with every new symbol
- The new range is proportional to the predefined probability attached to that symbol
- Decoding is the inverse procedure, in which the range is expanded in proportion to the probability of each symbol as it is extracted