Discrete Mathematics

Application of Binary Tree
Data Compression

Dr. Ta Quang Hung

Data Compression

Data compression has come of age in the last 20 years. Both the quantity and the quality of the body of literature in this field provides ample proof of this. However, the need for compressing data has been felt in the past, even before the advent of computers, as the following quotation suggests:

> "I have made this letter longer than usual because
> I lack the time to make it shorter."
>
> Blaise Pascal

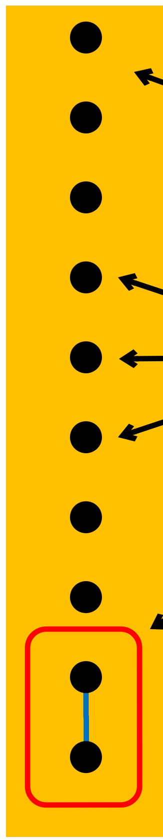
There are many known methods for data compression. They are based on different ideas, are suitable for different types of data, and produce different results, but they are all based on the same principle, namely they compress data by removing *redundancy* from the original data in the source file. Any non-random collection data has some structure, and this structure can be exploited to achieve a smaller representation of the data, a representation where no structure is discernible. The terms *redundancy* and *structure* are used in the professional literature, as well as *smoothness*, *coherence*, and *correlation*; they all refer to the same thing. Thus, redundancy is an important concept in any discussion of data compression.

STATIC HUFFMAN

Static Huffman

Given a string, with probability of appearance each character in the string as

Char	Probability
e	0.274
a	0.171
i	0.149
h	0.130
d	0.092
c	0.057
f	0.052
g	0.042
b	0.031
j	0.002



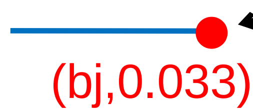
This column will be added to a table

Sort the characters by descending order of probability.

Leaf nodes

Select two nodes with lowest weight

Connect these nodes, then form a new node with weight:
0.031 + 0.002 = 0.033

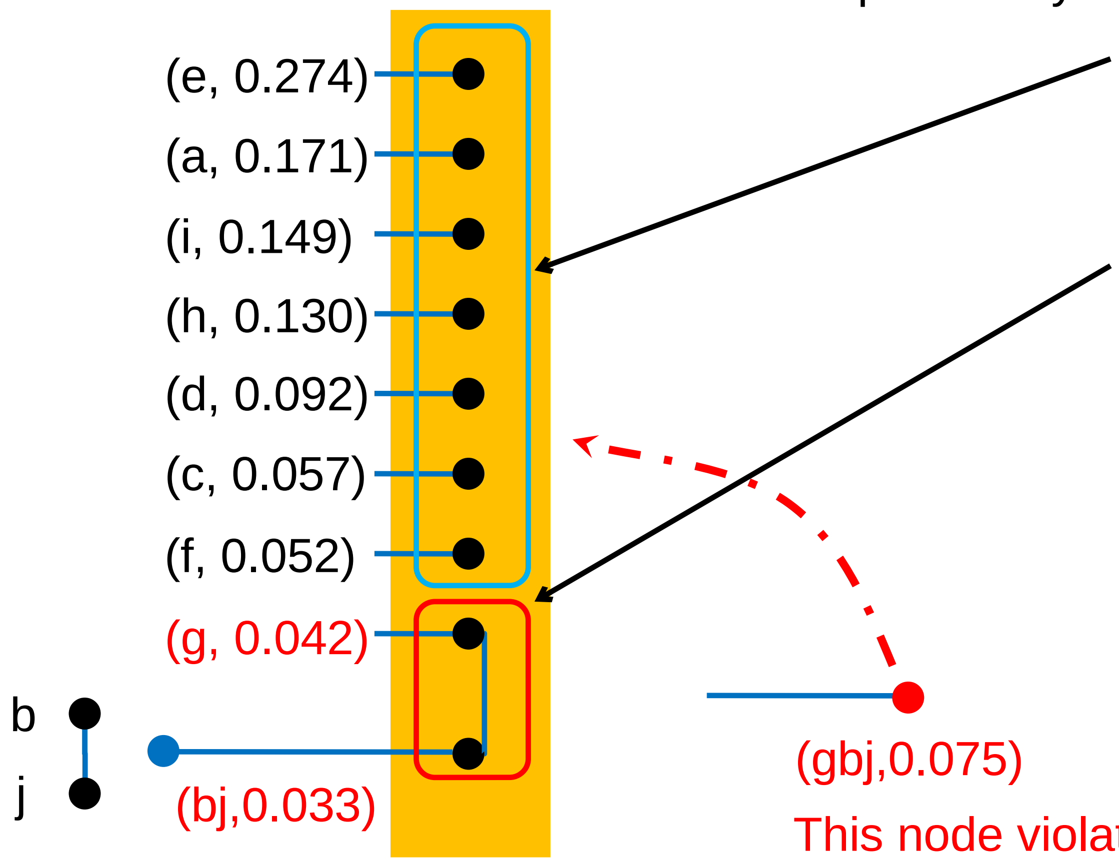


New node (bj, 0.033) will replace (b, 0.031) and (j, 0.002)

Static Huffman

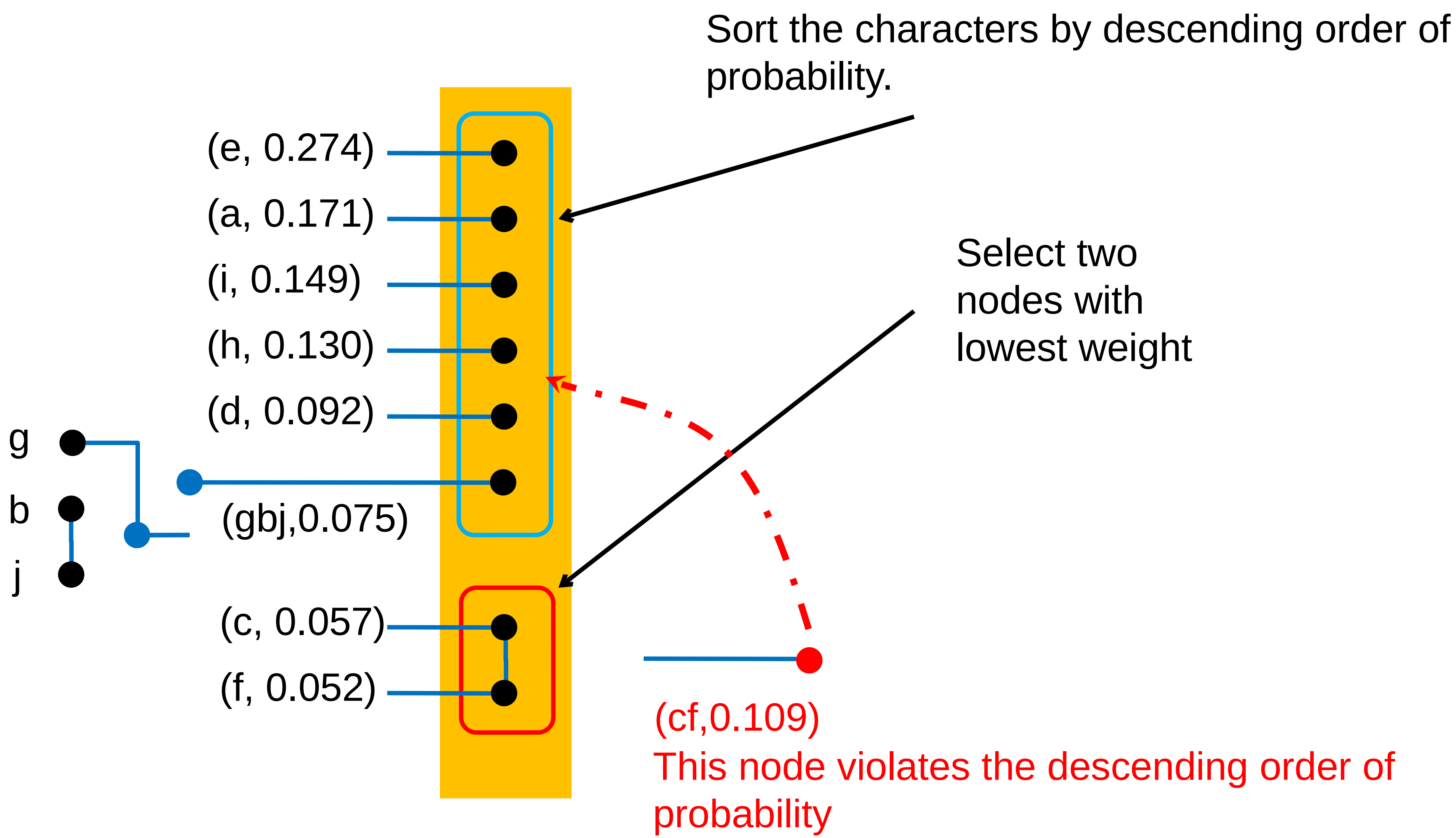
Sort the characters by descending order of probability.

Select two nodes with lowest weight

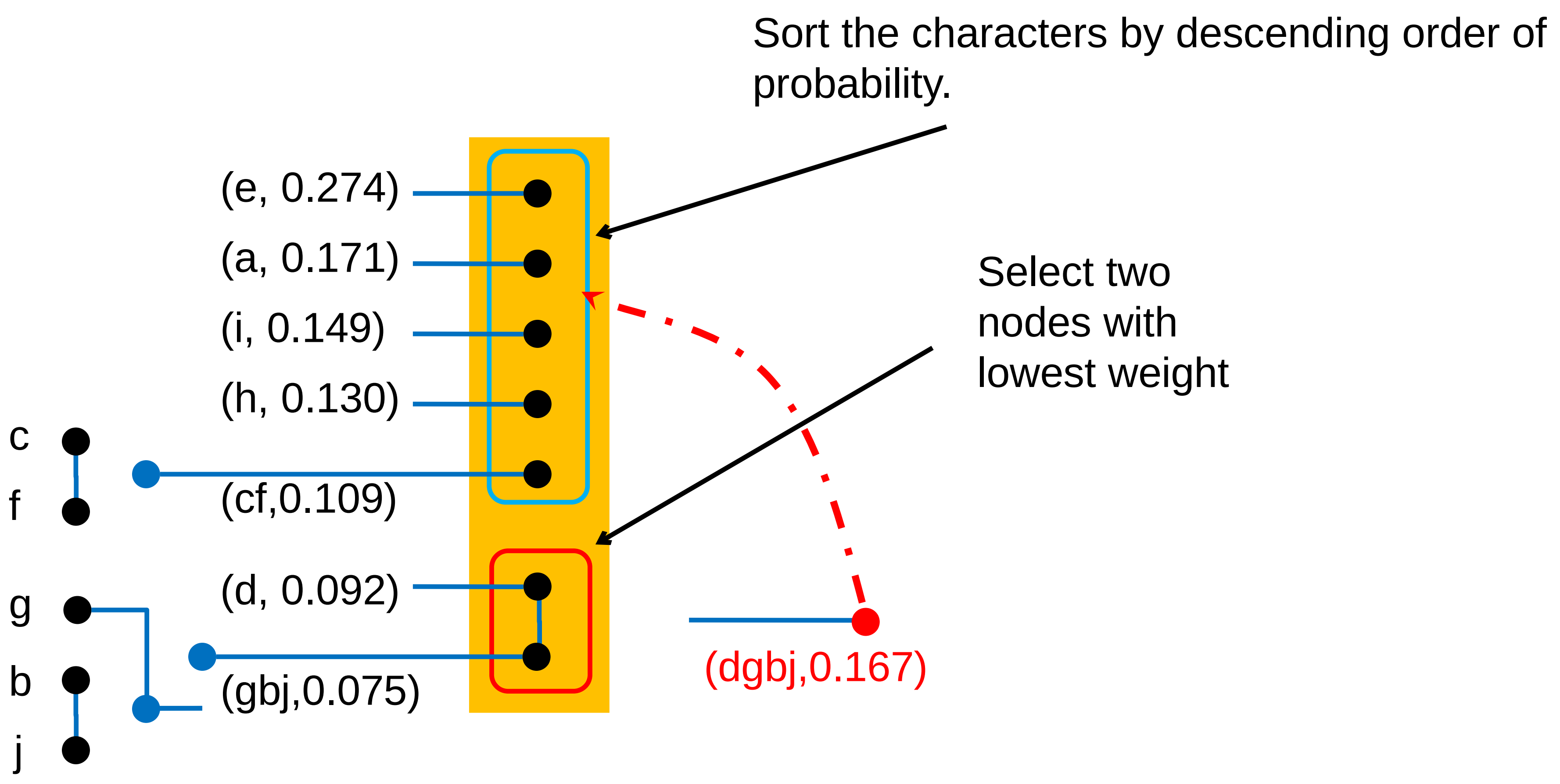


This node violates the descending order of probability

Static Huffman



Static Huffman

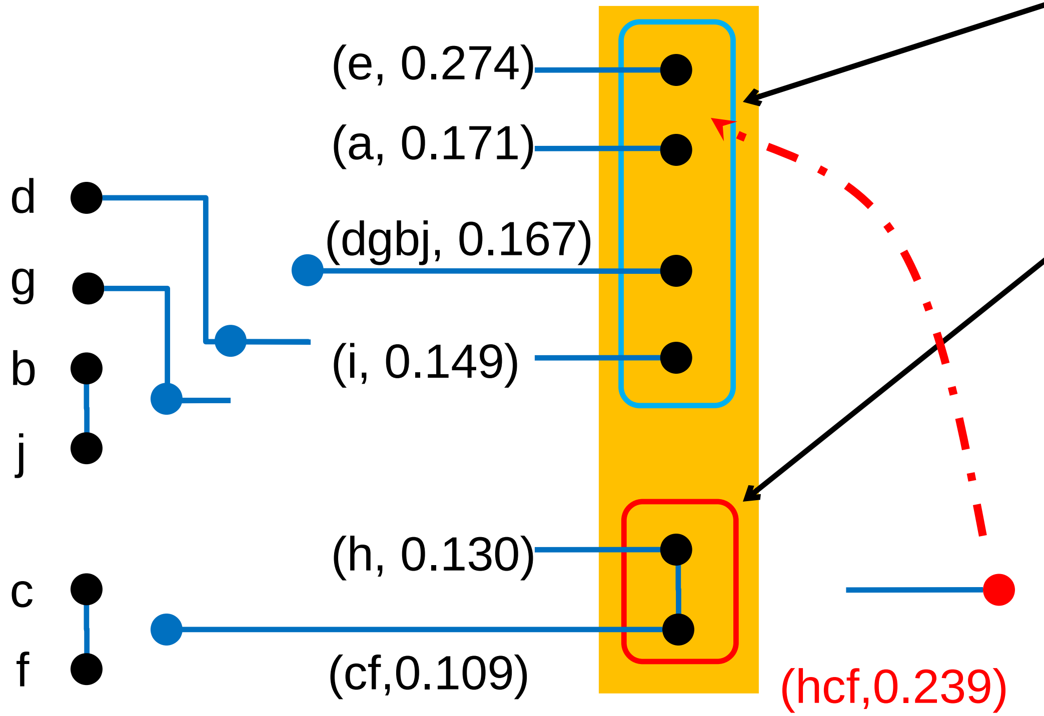


This node violates the descending order of probability

Static Huffman

Sort the characters by descending order of probability.

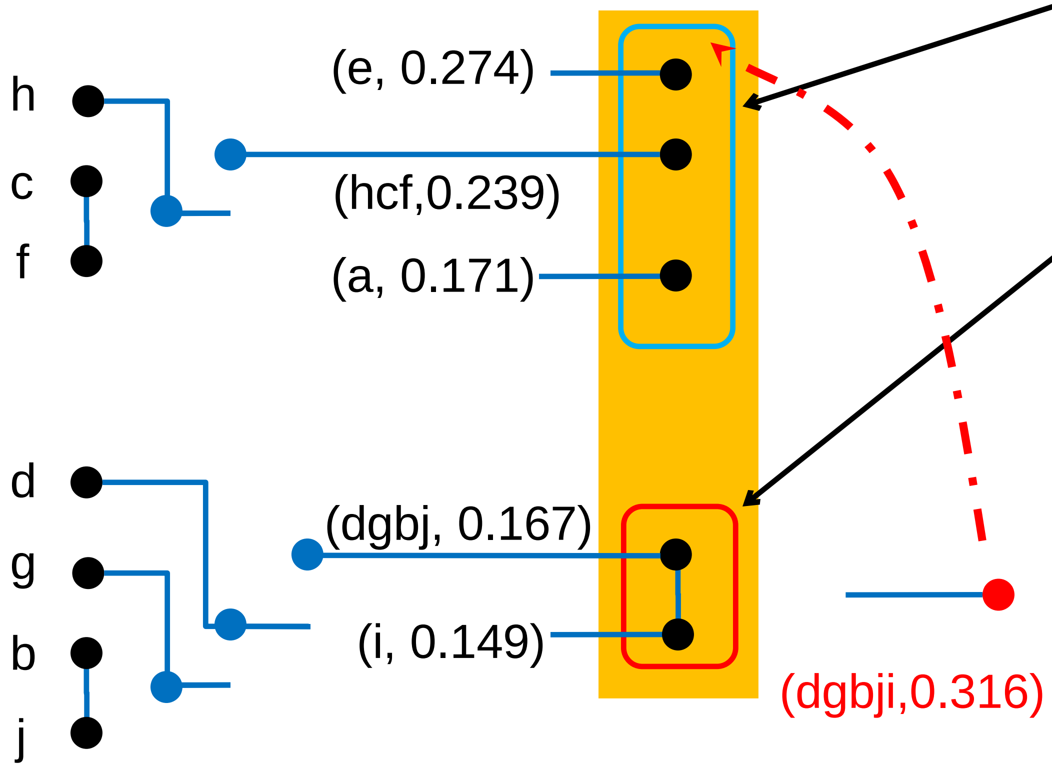
Select two nodes with lowest weight



This node violates the descending order of probability

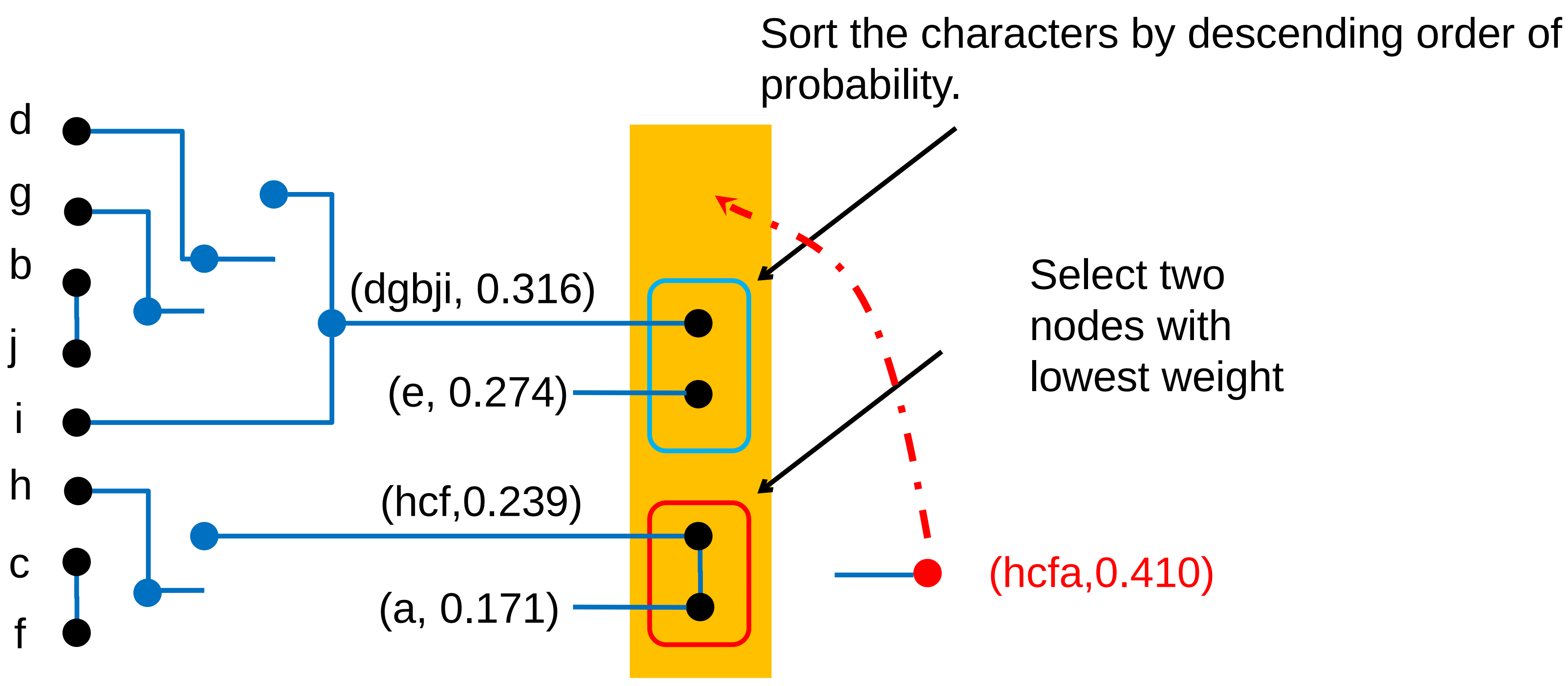
Sort the characters by descending order of probability.

Select two nodes with lowest weight



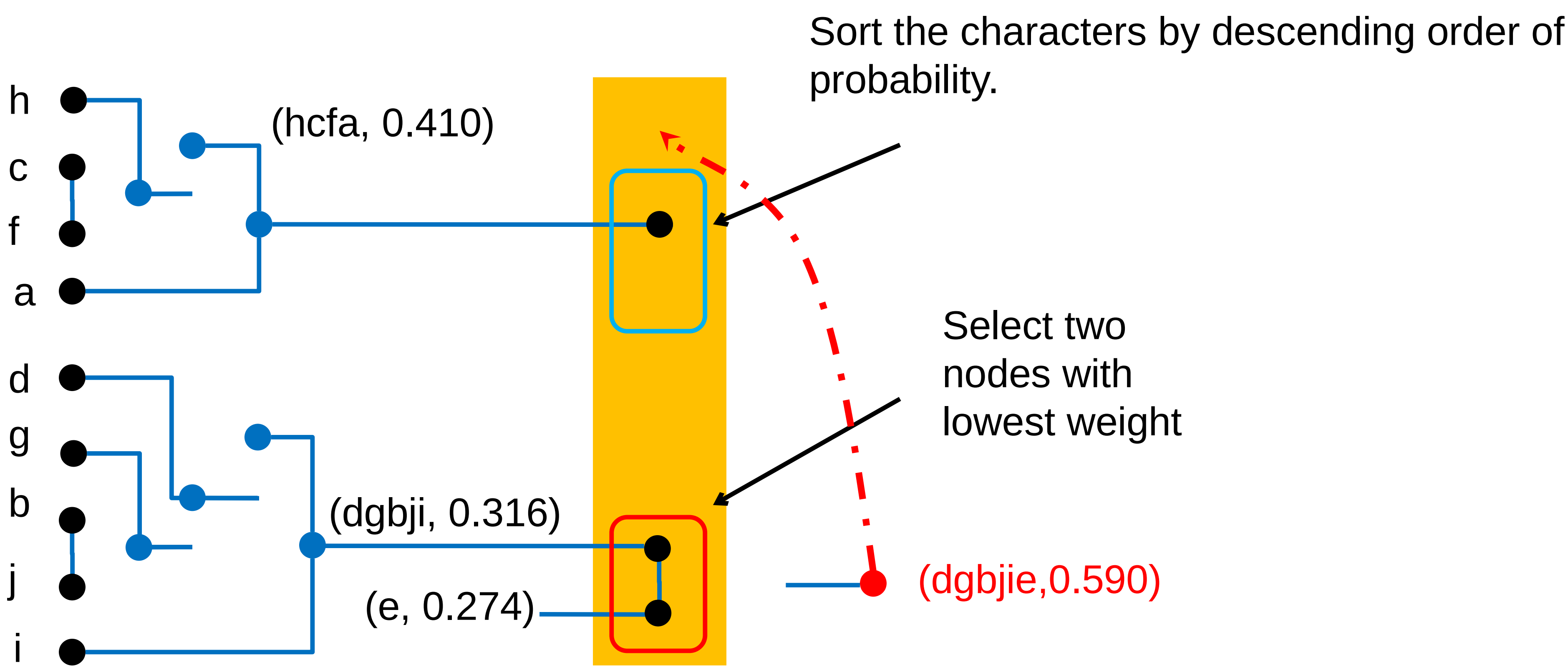
This node violates the descending order of probability

Static Huffman



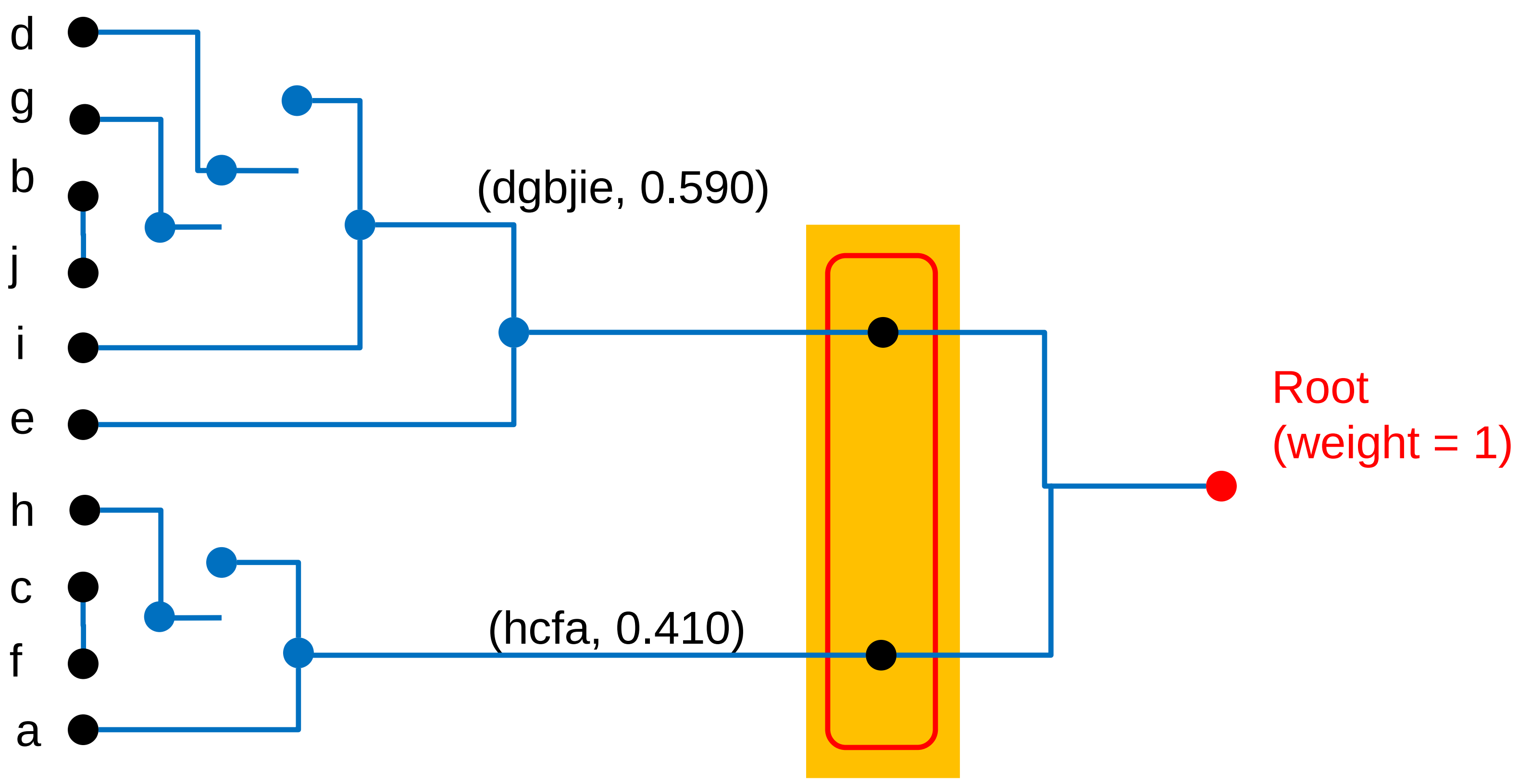
This node violates the descending order of probability

Static Huffman



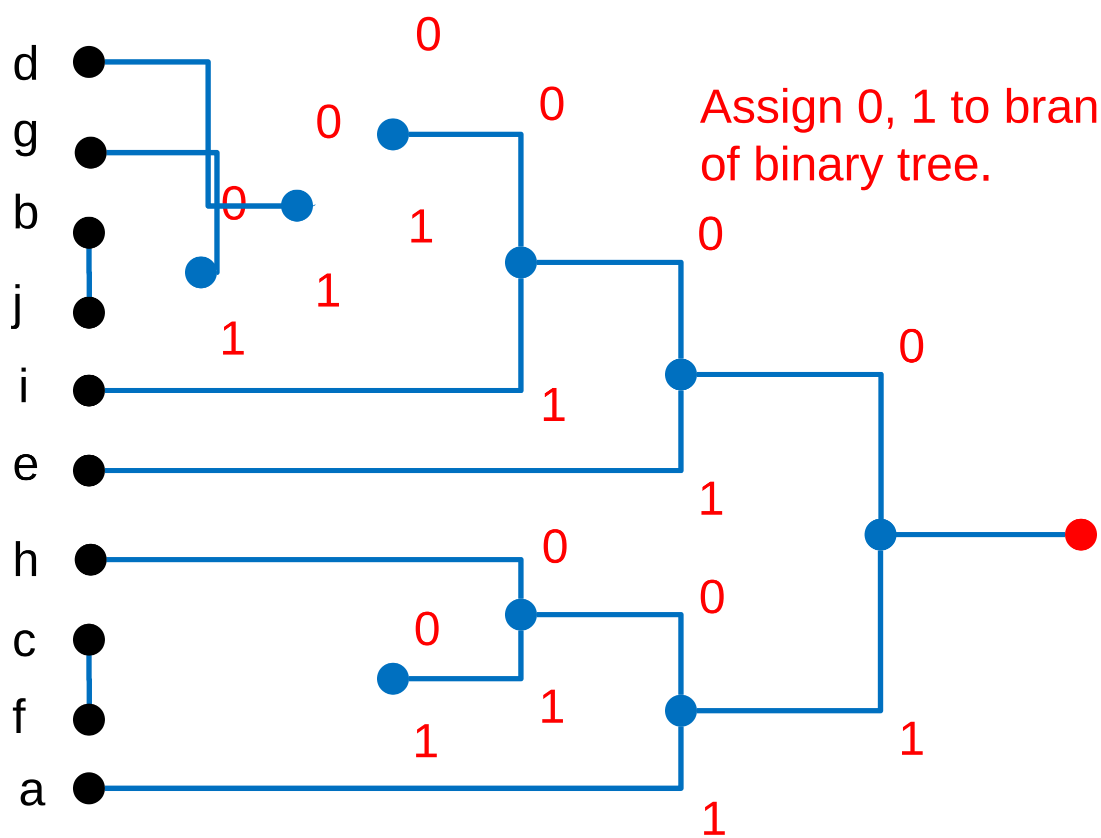
This node violates the descending order of probability

Static Huffman



Huffman Tree

Static Huffman



Assign 0, 1 to branches of binary tree.

Char	Code
a	11
b	000110
c	1010
d	0000
e	01
f	1011
g	00010
h	100
i	001
j	000111

Used 39 bit < 10*8 = 80 bit.

Static Huffman – Create a table

C	1	2	3	4	5	6	7	8	9	10	Code
e	0.274	0.274	0.274	0.274	0.274	0.274	0.316	0.410	0.590	1	**01**
a	0.171	0.171	0.171	0.171	0.171	0.239	0.274	0.316	0.410		**11**
i	0.149	0.149	0.149	0.149	0.167	0.171	0.239	0.274			**001**
h	0.130	0.130	0.130	0.130	0.149	0.167	0.171				**100**
d	0.092	0.092	0.092	0.109	0.130	0.149					**0000**
c	0.057	0.057	0.075	0.092	0.109						**1010**
f	0.052	0.052	0.057	0.075							**1011**
g	0.042	0.042	0.052								**00010**
b	0.031	0.033									**000110**
j	0.002										**000111**

Data in a column is the yellow region

Red line: add two nodes together. Assign “0”, “1” here

Blue line: Trace of the sorting.

Code words, are binary strings (“0”, “1”) obtained from the root to the leaf nodes.

Static Huffman – Need sorting?

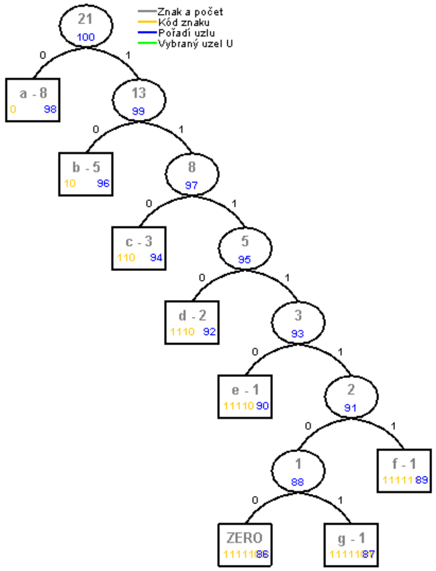
No. However, we must choose two nodes with lowest weight.

X	p(X)
a	0.171
b	0.031
c	0.057
d	0.092
e	0.274
f	0.052
g	0.042
h	0.130
i	0.149
j	0.002

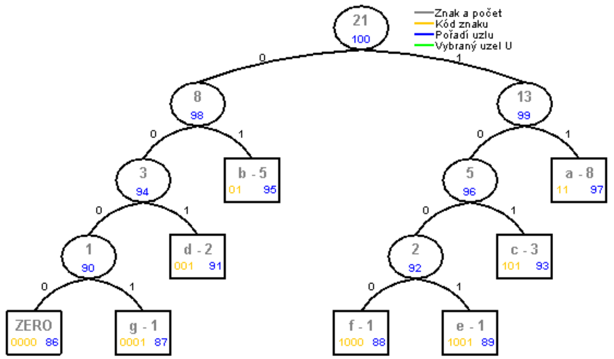
ADAPTIVE HUFFMAN

- Two methods:

Faller-Gallager-Knuth (FGK) Algorithm



Vitter Algorithm



Balance tree

Adaptive Huffman

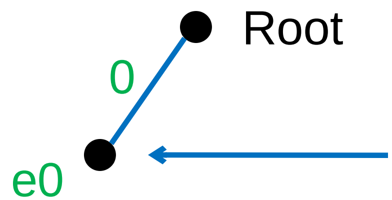
- Describe the algorithm by building binary tree.
- **Sibling property:** Thus the bottom left node has the lowest frequency, and the top right one has the highest frequency.
- String used as input

DYNAMIC HUFFMAN

Characters repeat at the end of string, easy to follow.

Adaptive Huffman

- Initialize an empty tree: 1 root, 1 empty node

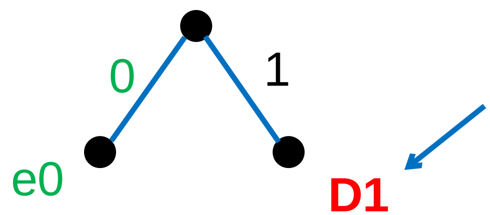


Empty node e0, green

- Put the first character: "D":

D '01000100'

8 bits for D



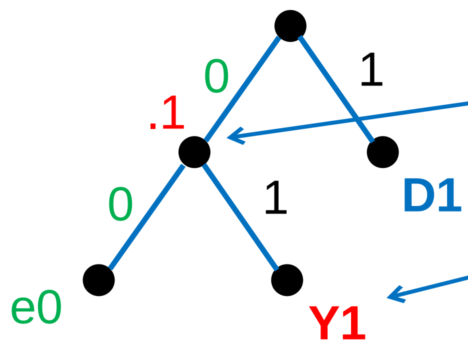
Put D1, with 1 as weighting coefficient.

- Next char "Y" will be connected to e0 ('0')

- Next character "Y" connect to e0='0'

Y '0' $\underbrace{\text{'01011001'}}_{\text{8 bits for Y}}$

'0': position of e0 in the current tree. After adding Y1, code-word of e0 change to '00'



Temporally node 1.

New node is red. Number 1 is weighting coefficient, remind that Y just appeared for the first time.

- Next "N" will be connected to e0='00'

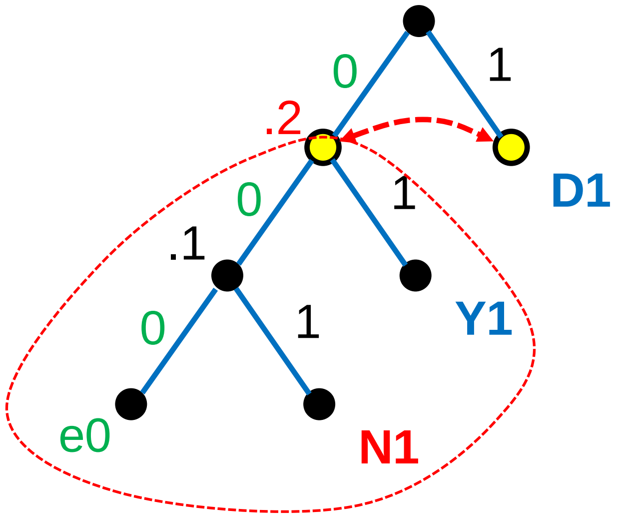
N '00' $\underbrace{\text{'01001110'}}$

8 bits for N because no duplicate chars, send full 8 bits.
New node will be created at '00'

- Add to e0='00'

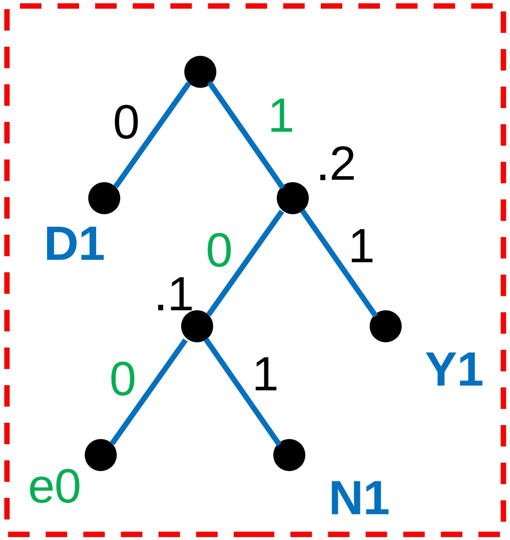
N '00' $\underbrace{\text{'01001110'}}$

N is added to '00' in the current tree



N1 is a new node

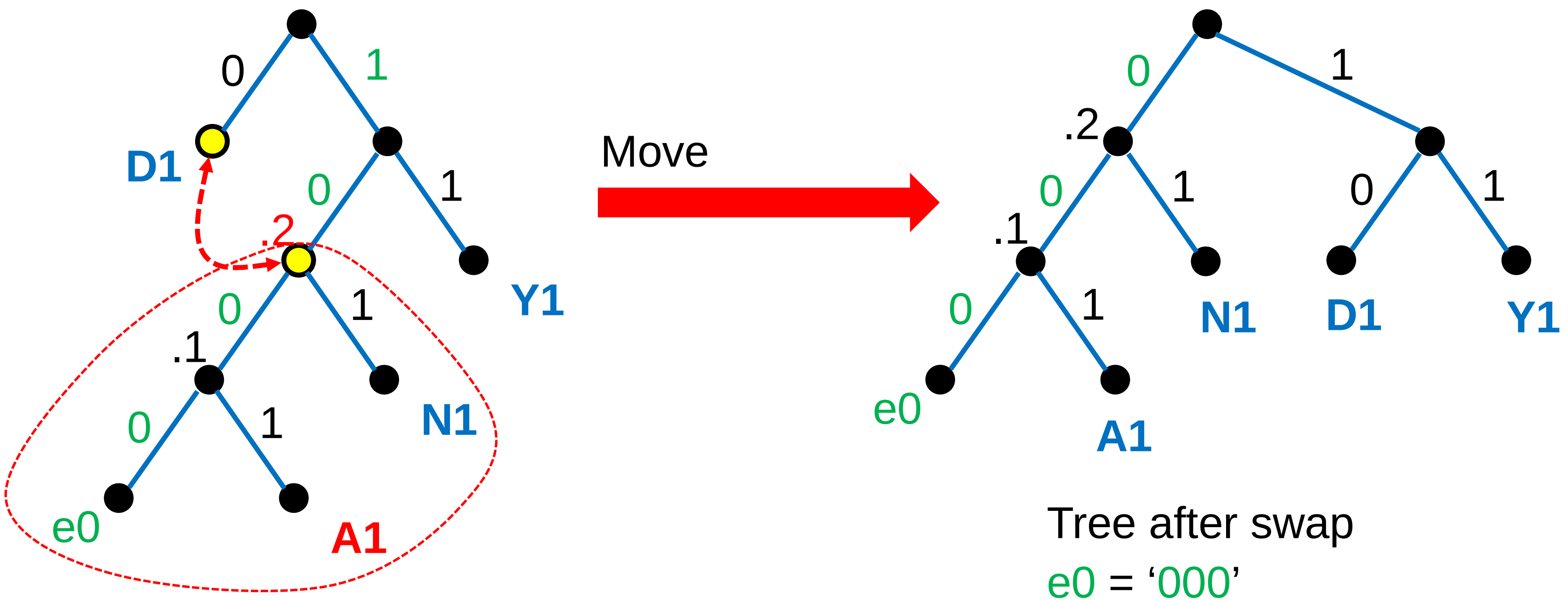
Violate sibling property move



Tree after swap

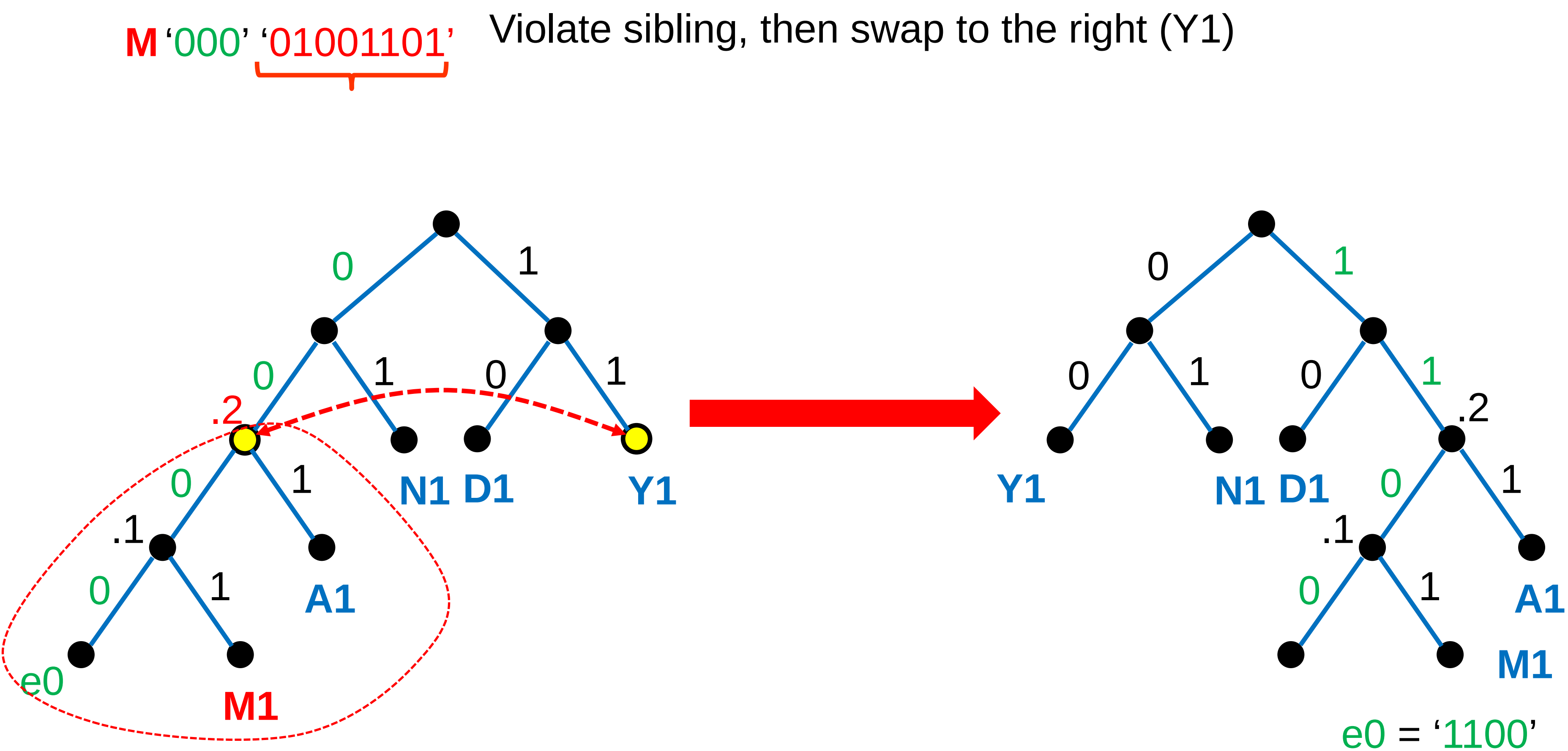
- Add to e0='100'

$\underset{}{\textbf{A}}$ '100' '$\underbrace{01000001}$' A can move upward and to the right. But take the upward to obtain a shorter codeword.



Tree after swap
e0 = '000'

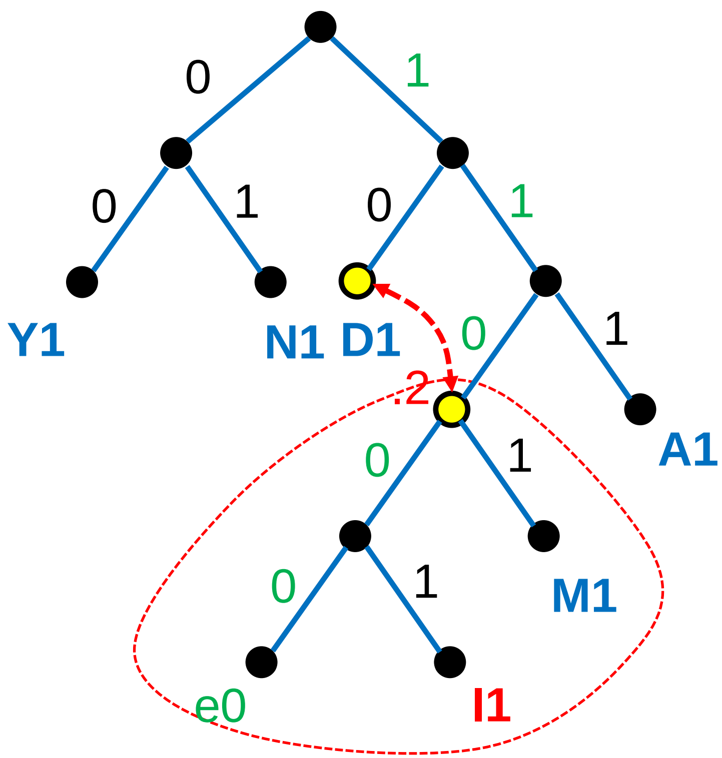
- Add to e0='000'

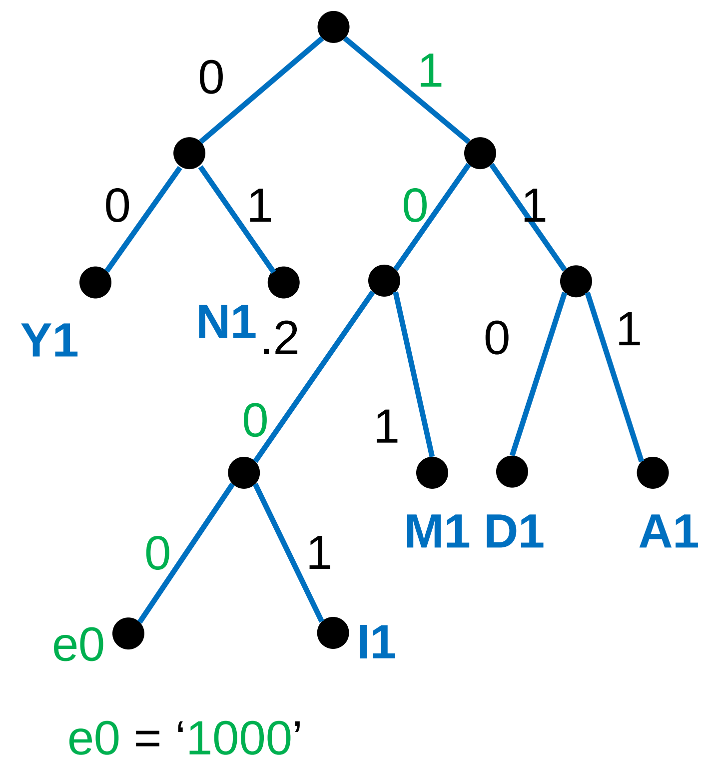

M '000' '01001101'
Violate sibling, then swap to the right (Y1)
0
1
0
1
0
1
.2
0
1
.1
0
1
e0
M1
A1
N1
D1
Y1
0
1
0
1
0
1
.2
Y1
N1
D1
0
1
.1
0
1
A1
M1
e0 = '1100'

Adaptive Huffman

I '1100' '01001001'

Violate sibling, then move upward, then to the right.

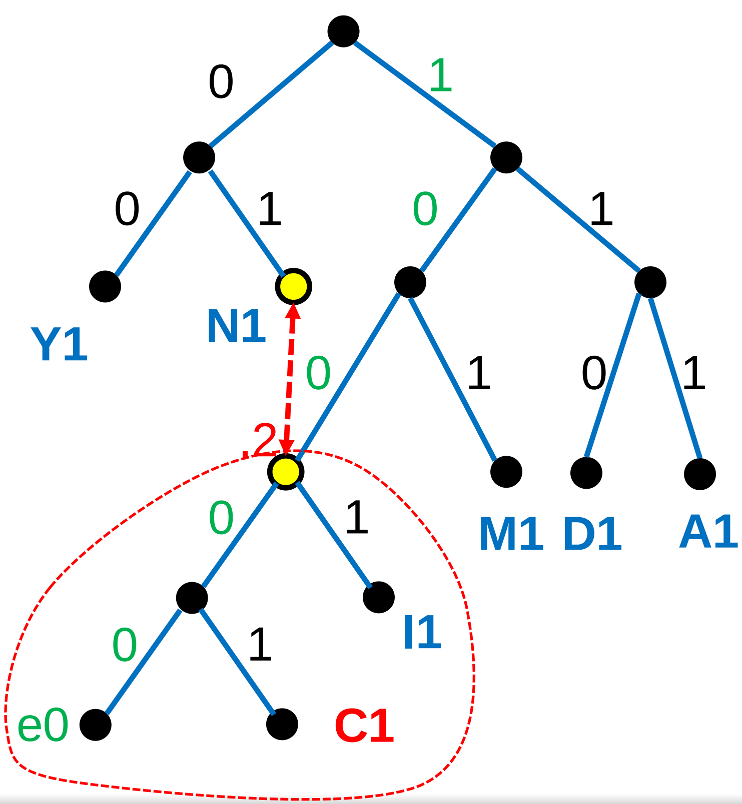


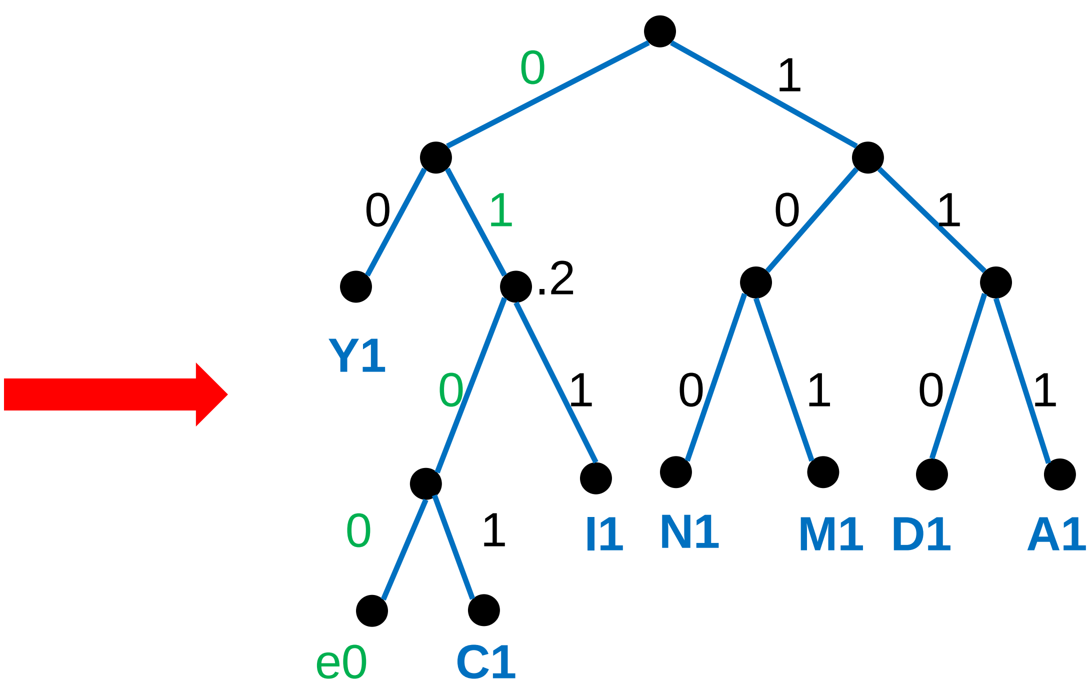


Adaptive Huffman

C'1000' '01000011'

Move upward, then to the right



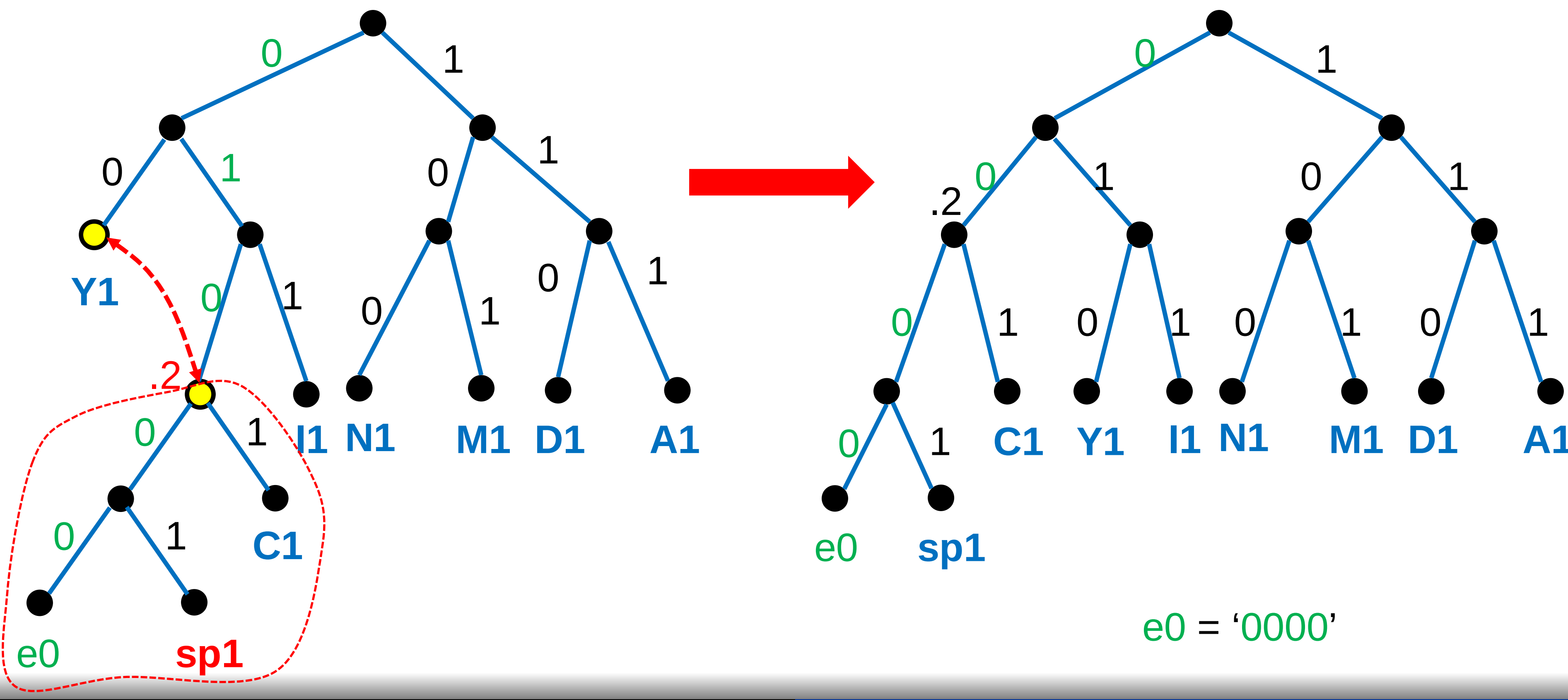


e0 = '0100'

Adaptive Huffman

sp '0100' '00100000'

Sp: Space. Move upward.

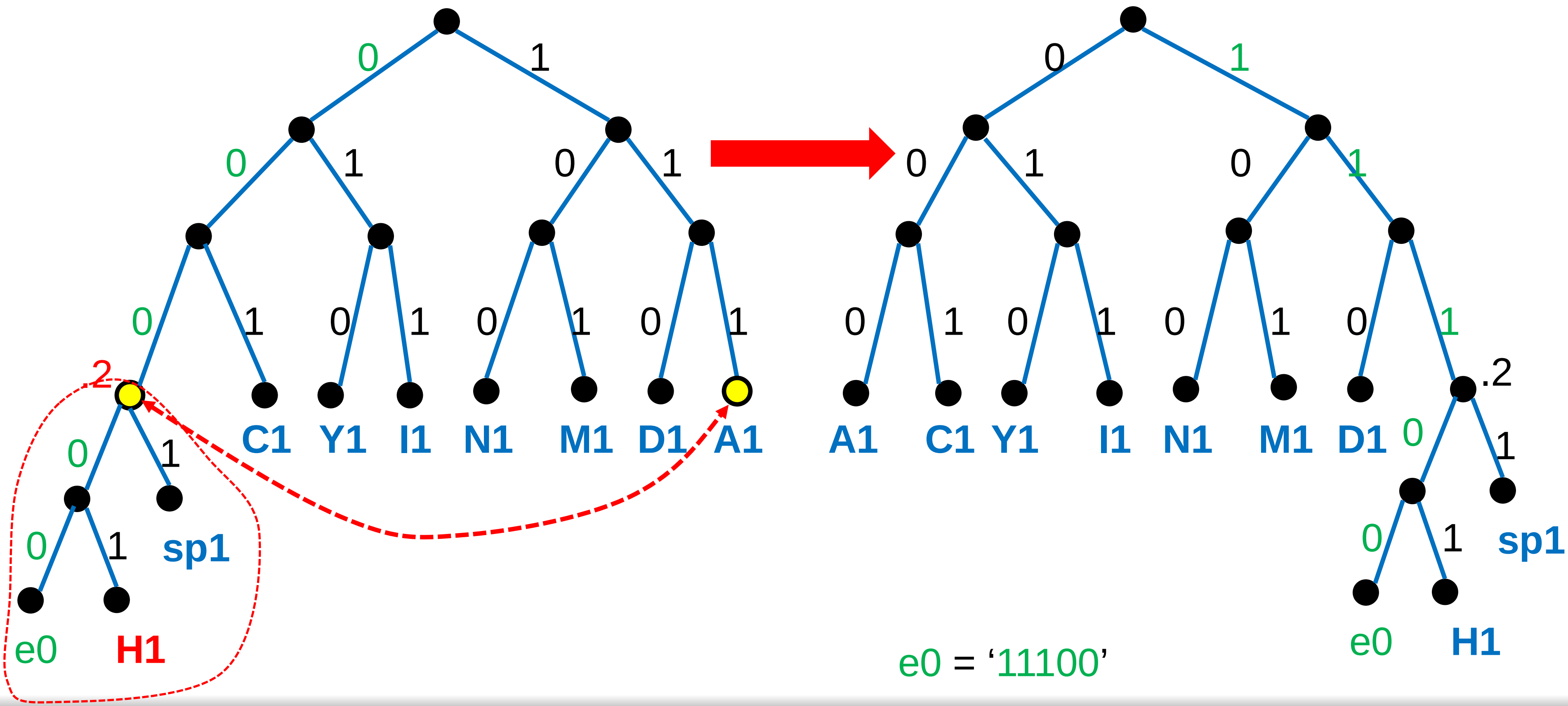


e0 = '0000'

Adaptive Huffman

H ‘0000’ ‘01001000’

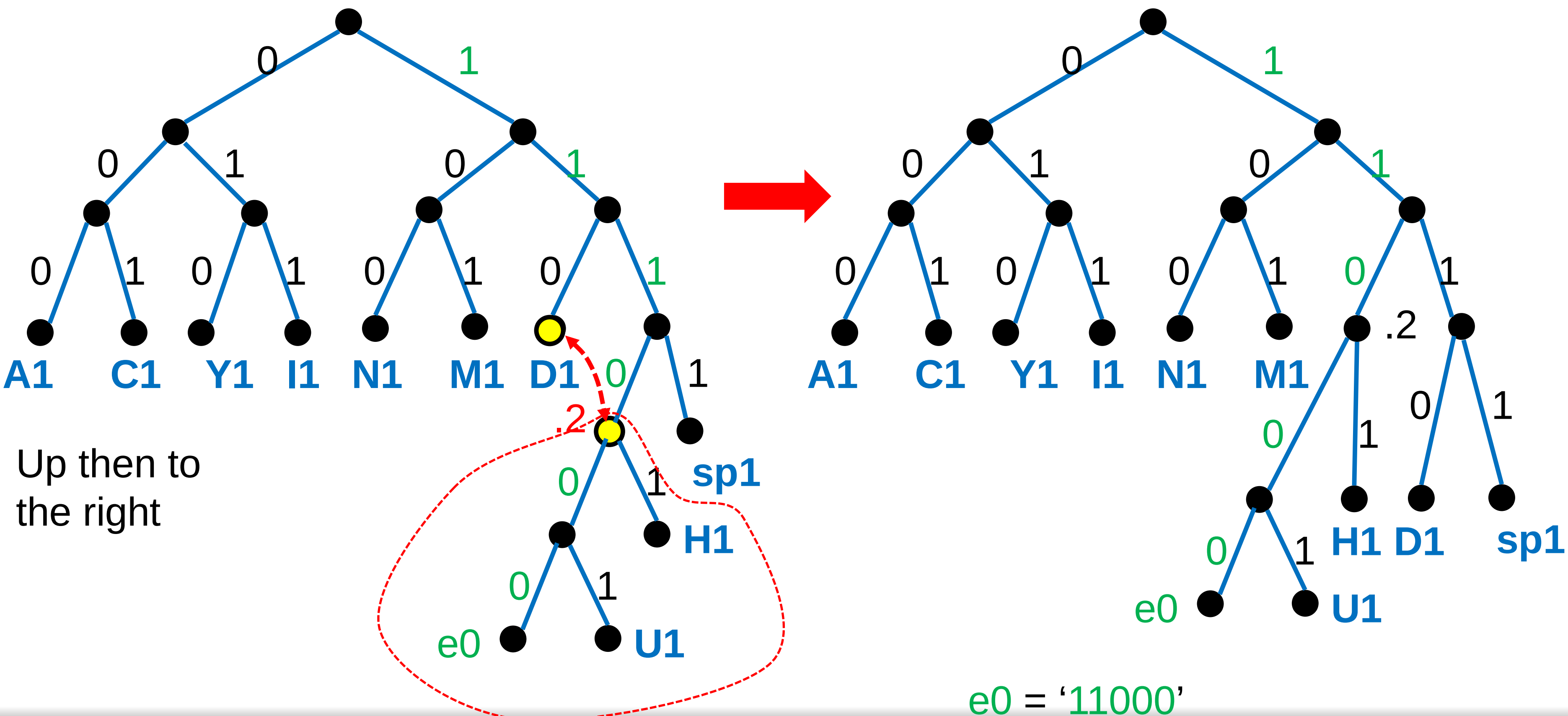
Cant move upward, so move to the right



e0 = ‘11100’

Adaptive Huffman

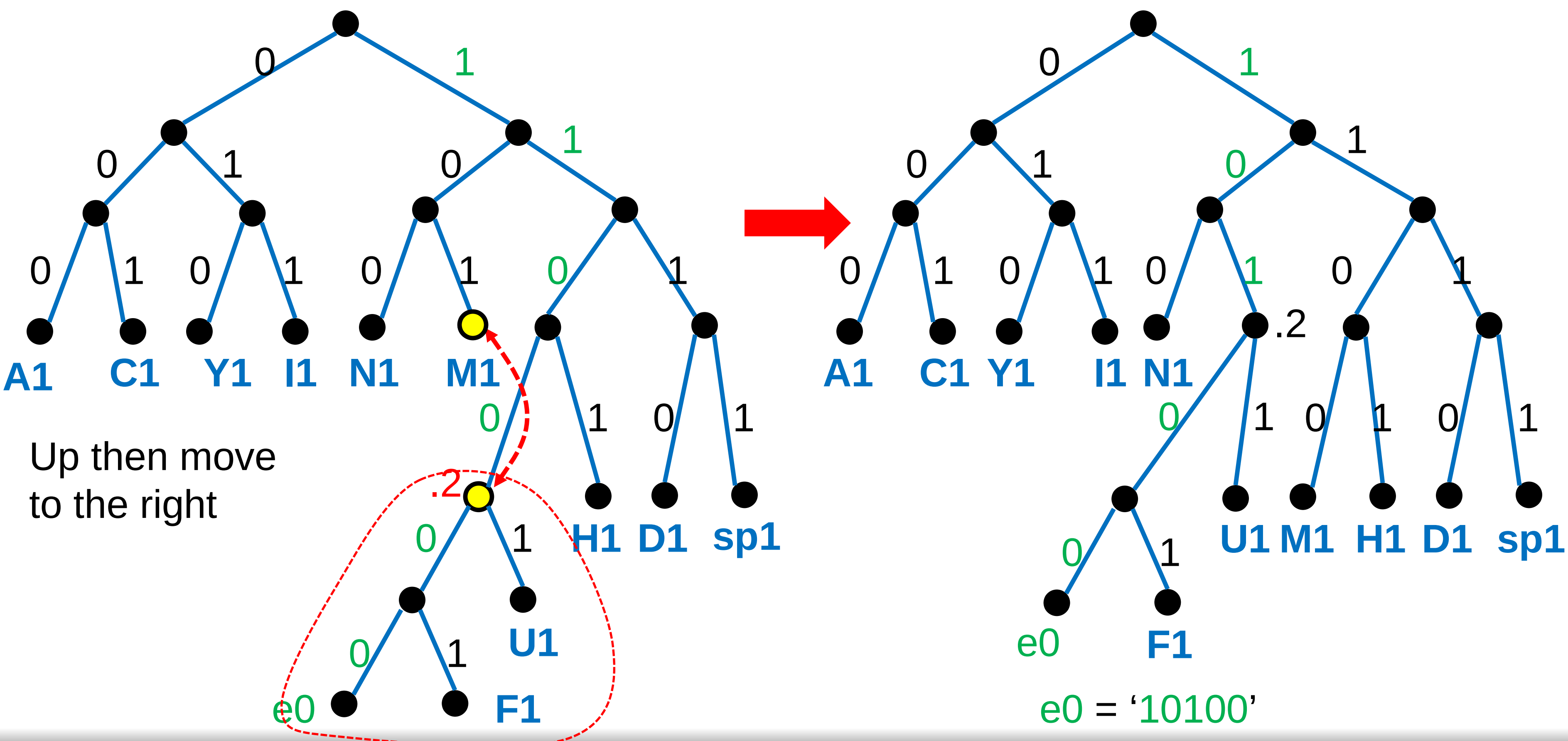
U '11100' '01010101'   Where to move?



Up then to the right

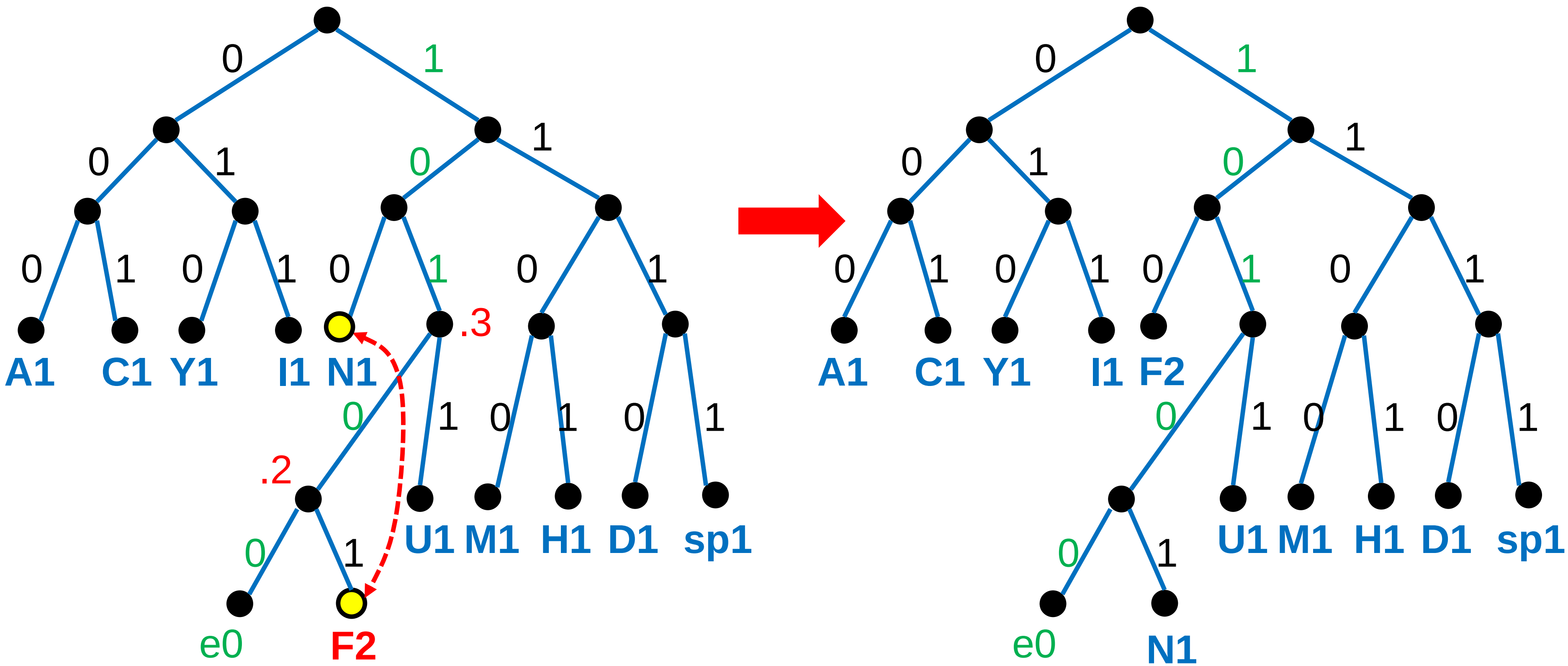
e0 = '11000'

F ‘11000’ ‘01000110’ Where to move?



Adaptive Huffman

F ‘10101’ Where to move?



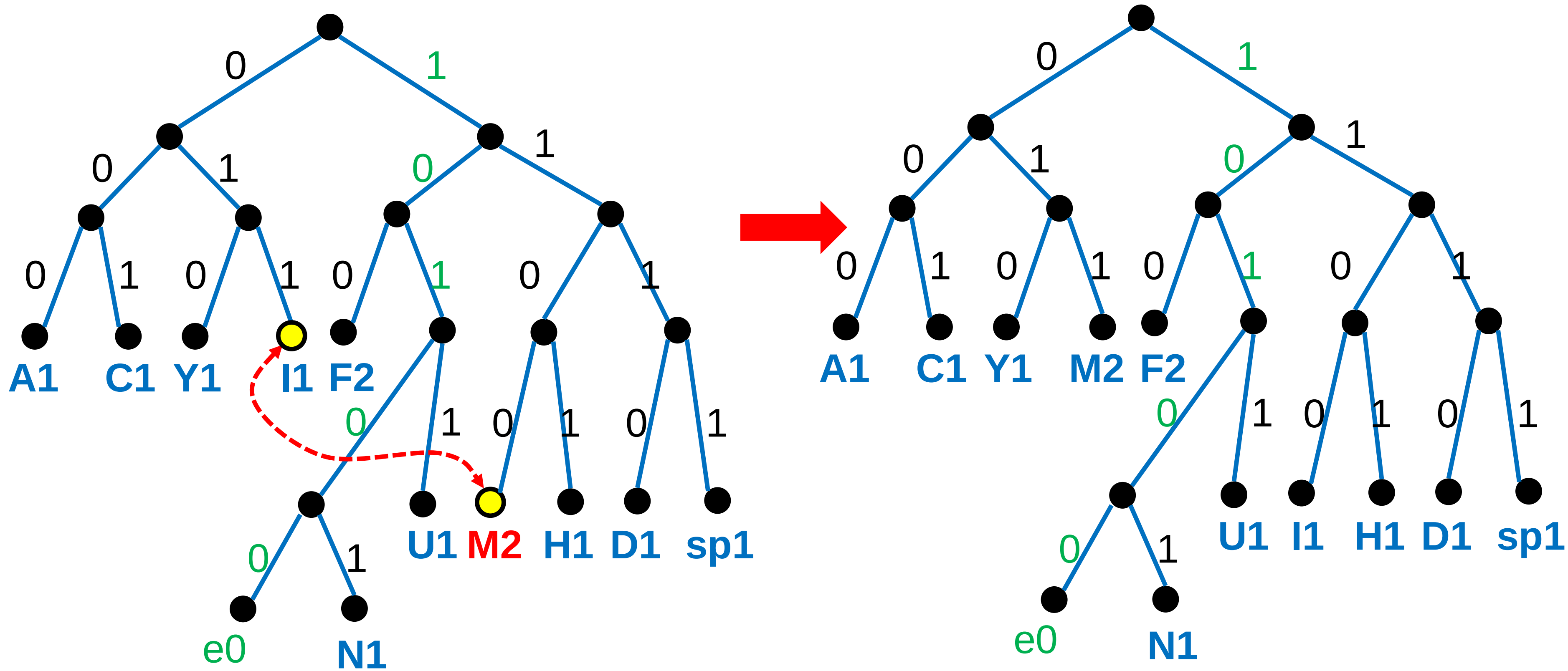
Up, then to the right.

No new node -> compressing

Adaptive Huffman

M ‘1100’

Where to move?



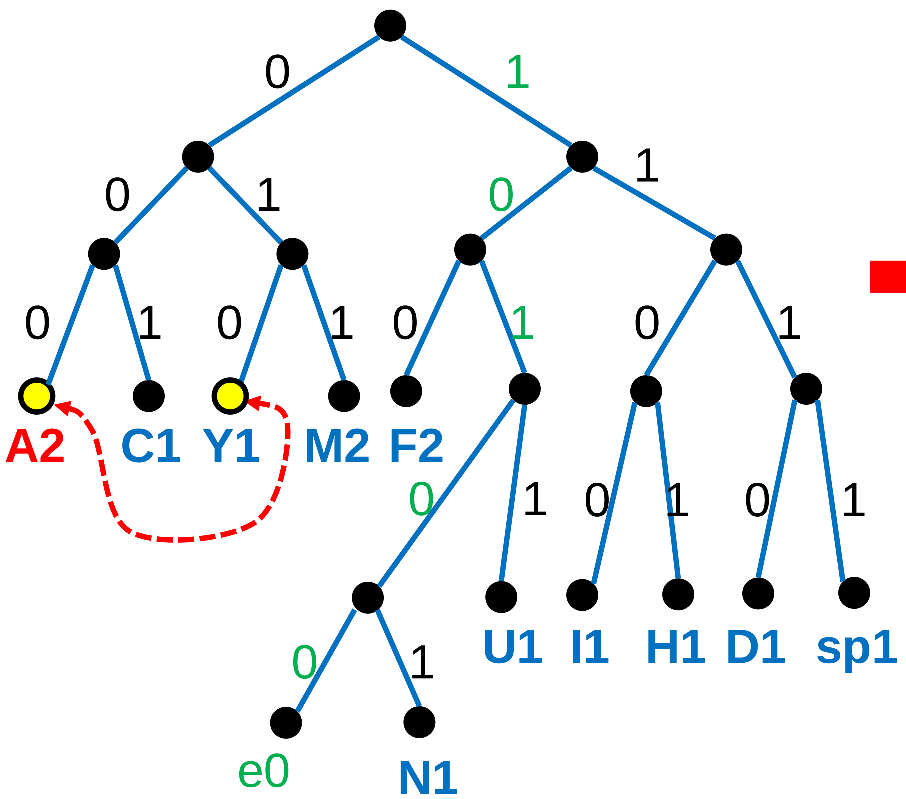
Up, then to the right.

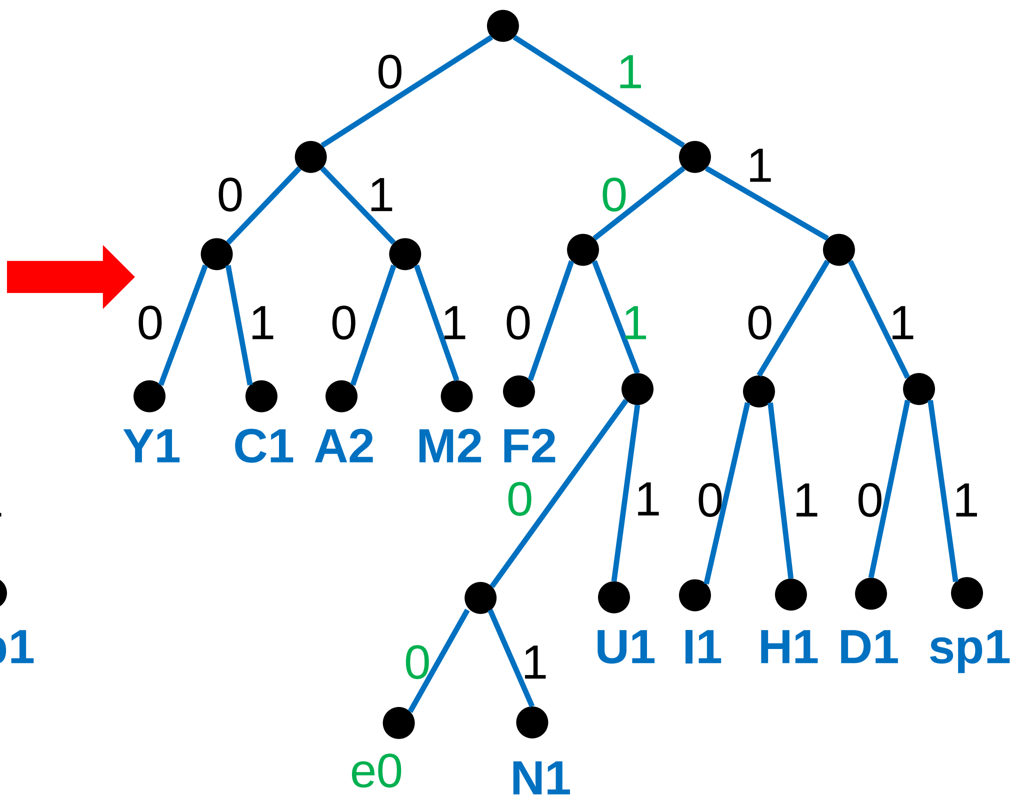
No new node -> compressing

Adaptive Huffman

A '000'

Where to move?





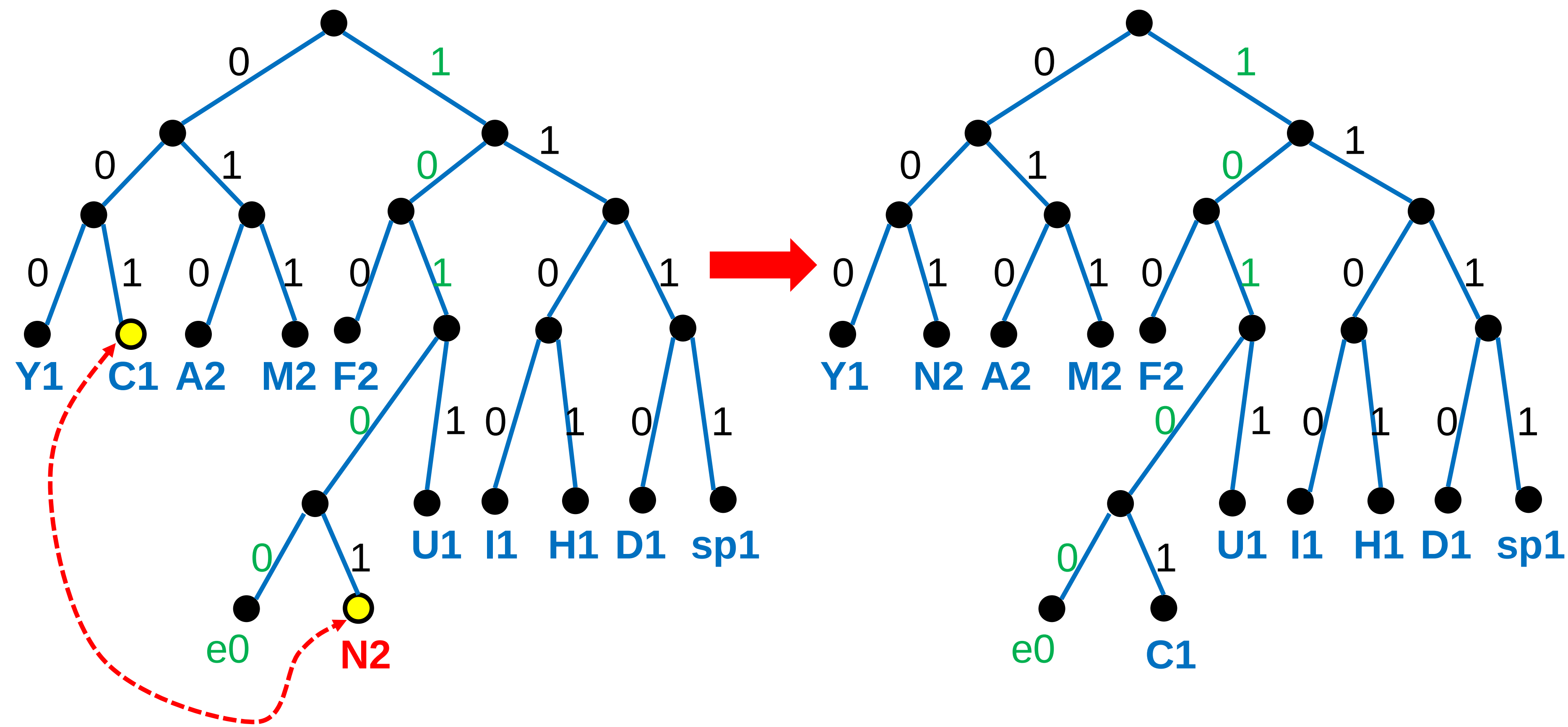
Up then to the right.

No new node -> compressing

Adaptive Huffman

N '10101'

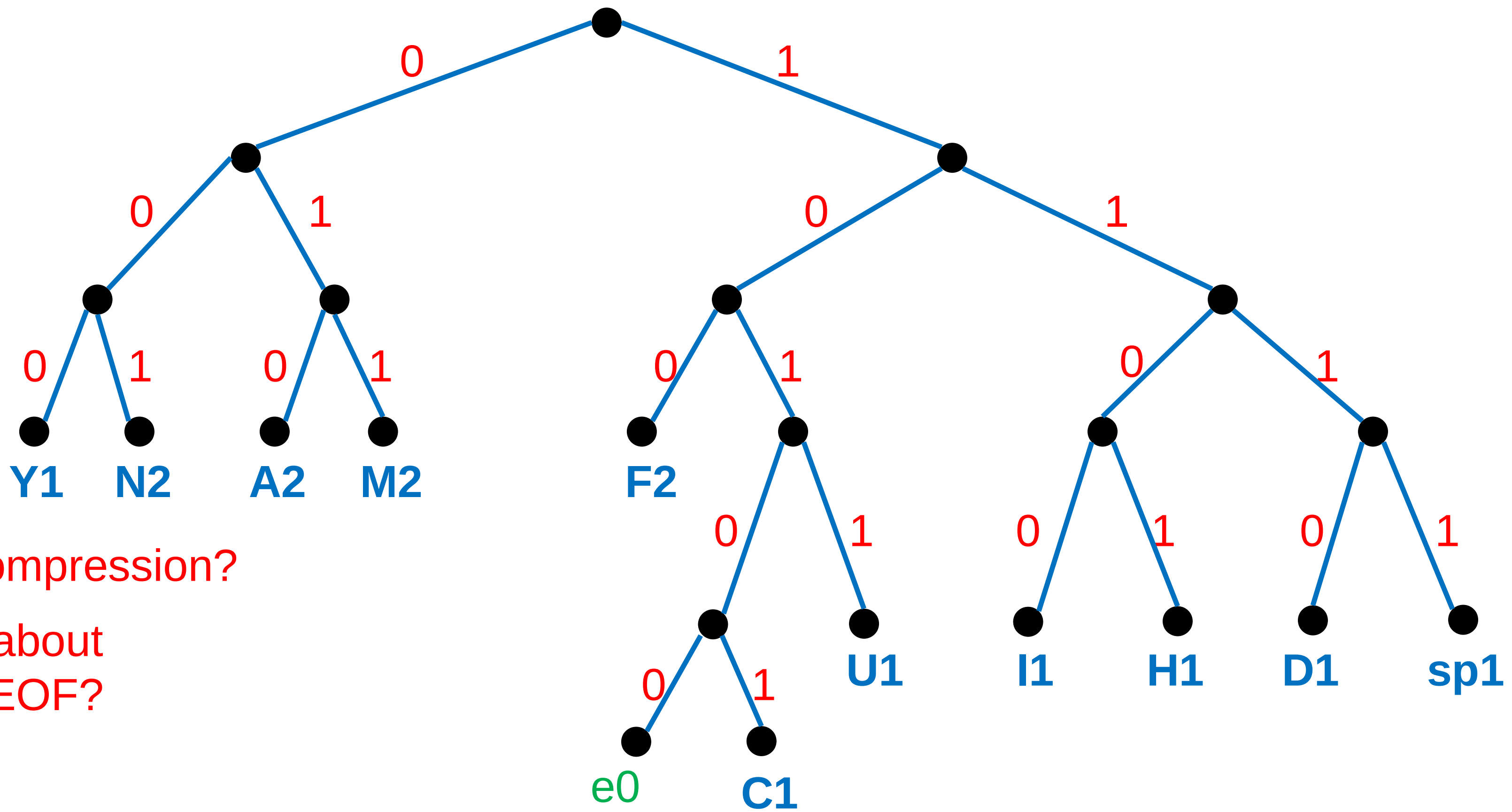
Where to move?



Up then to the right.

No new node -> compressing

Finish, code-words forms by reading from root to leaf nodes.



File compression?

What about
BOF, EOF?

Pros & Cons

Static Huffman	Adaptive Huffman
Scan all data to calculate probabilities.	No scan, can use for online.
Send along the codeword tree.	Don’t have to send codeword tree.
Many node with same probability make different codeword tree.	Position of the node depend on the apperance order, there is one codeword
Unbalance tree.	Balace tree
Easy to build tree	Difficult to build tree.
Receiver: First get the codeword tree, then get data.	Receiver: get data from the beginning.
Encode and Decode in the same tree, so the efficient of compression is low, cannot use online.	Encode and Decode depend on many tree, especially to a tree at the current processing.

Tài liệu tham khảo

- The Data Compression Book – Mark Nelson & Jean-Loup Gailly (3rd Edition).
- Understanding Data Communications and Networks – William A. Shay (3rd Edition).
- Data Compression – David Salomon (3rd Edition).
- http://www.data-compression.com