CSEP 590
Data Compression
Autumn 2007

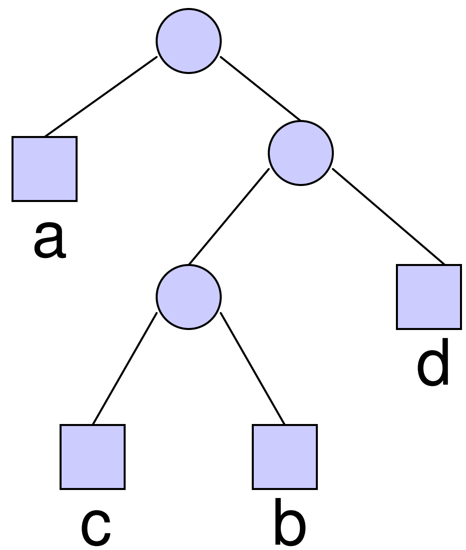
Adaptive Huffman Coding

Adaptive Huffman Coding

- One pass
- During the pass calculate the frequencies
- Update the Huffman tree accordingly
  - Coder – new Huffman tree computed after transmitting the symbol
  - Decoder – new Huffman tree computed after receiving the symbol
- Symbol set and their initial codes must be known ahead of time.
- Need NYT (not yet transmitted symbol) to indicate a new leaf is needed in the tree.

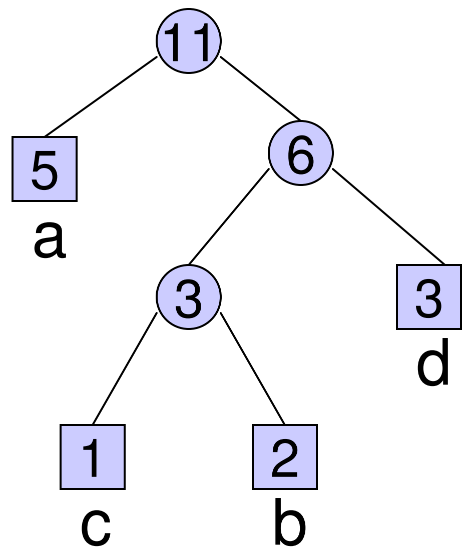
Optimal Tree Numbering

- a : 5, b: 2, c : 1, d : 3



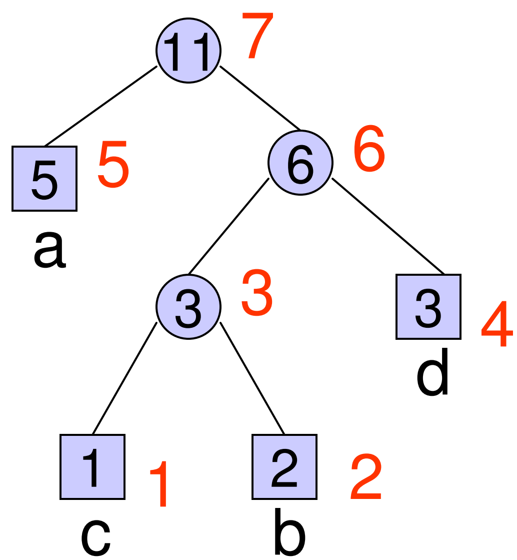
Weight the Nodes

- a : 5, b: 2, c : 1, d : 3



Number the Nodes

- a : 5, b: 2, c : 1, d : 3



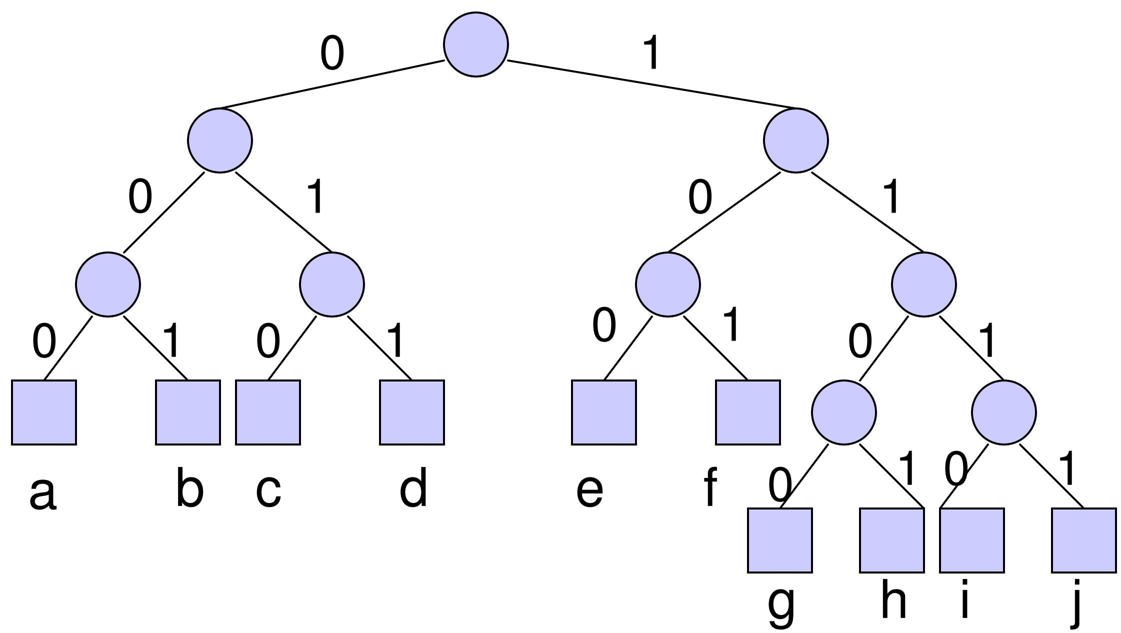
Number the nodes as they are removed from the priority queue.

Adaptive Huffman Principle

- In an optimal tree for n symbols there is a numbering of the nodes $y_1 < y_2 < \ldots < y_{2n-1}$ such that their corresponding weights $x_1, x_2, \ldots, x_{2n-1}$ satisfy:
  - $x_1 \le x_2 \le \ldots \le x_{2n-1}$
  - siblings are numbered consecutively
- And vice versa
  - That is, if there is such a numbering then the tree is optimal. We call this the node number invariant.

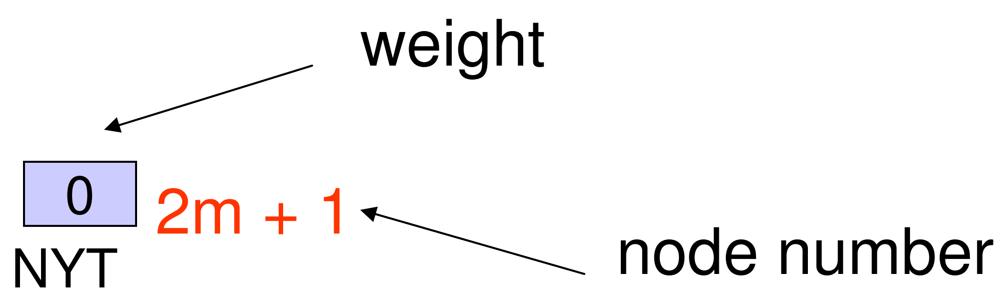
Initialization

- Symbols a_1, a_2, ... ,a_m have a basic prefix code, used when symbols are first encountered.
- Example: a, b ,c, d, e, f, g, h, i, j



Initialization

- The tree will encode up to m + 1 symbols including NYT.
- We reserve numbers 1 to 2m + 1 for node numbering.
- The initial Huffman tree consists of a single node



Coding Algorithm

1. If a new symbol is encountered then output the code for NYT followed by the fixed code for the symbol. Add the new symbol to the tree.
2. If an old symbol is encountered then output its code.
3. Update the tree to preserve the node number invariant.

Decoding Algorithm

1. Decode the symbol using the current tree.
2. If NYT is encountered then use the fixed code to decode the symbol. Add the new symbol to the tree.
3. Update the tree to preserve the node number invariant.

Updating the Tree

1. Let y be leaf (symbol) with current weight x.*
2. If y the root update x by 1, otherwise,
3. Exchange y with the largest numbered node with the same weight (unless it is the parent).**
4. Update x by 1
5. Let y be the parent with its weight x and go to 2.

*We never update the weight of NYT

** This exchange will preserve the node number invariant

Example

- aabcdad in alphabet {a,b,..., j}

0 21

NYT

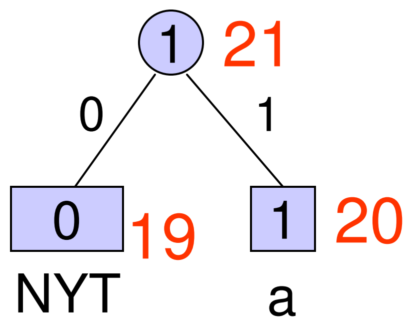
fixed code for a

output = 000

fixed code

Example

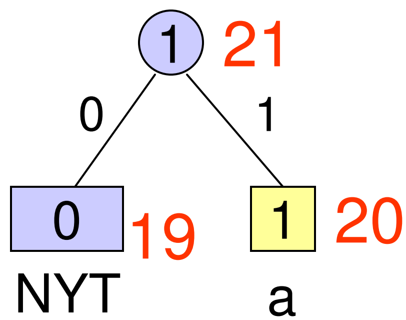
- <u>a</u>abcdad



output = 000

Example

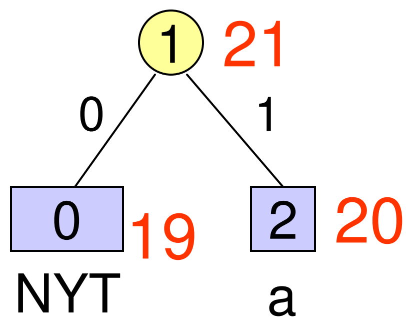
- a<u>a</u>bcdad



output = 000<u>1</u>

Example

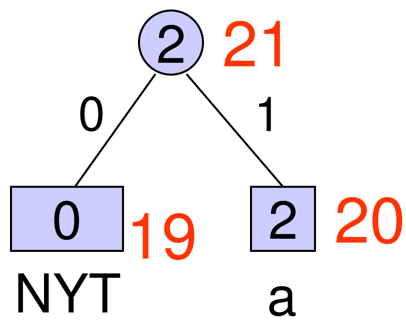
- a<u>a</u>bcdad



output = 0001

Example

- aabcdad



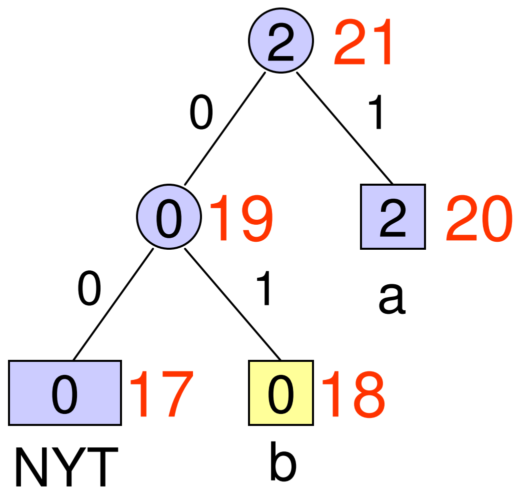
NYT

fixed code for b

output = 00010001

Example

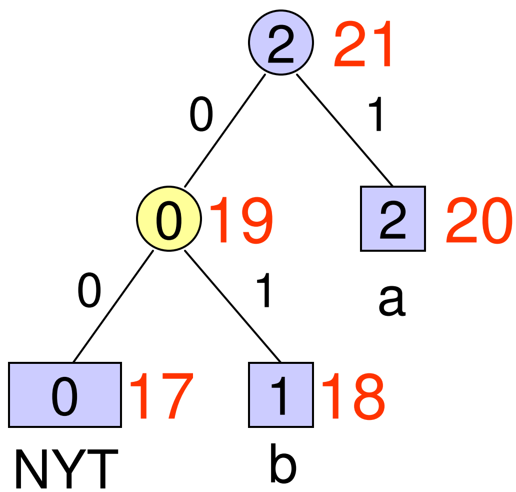
- aa<u>b</u>cdad



output = 00010001

Example

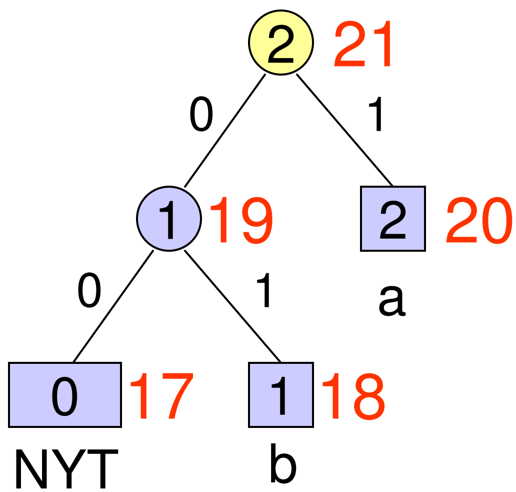
- aa<u>b</u>cdad



output = 00010001

Example

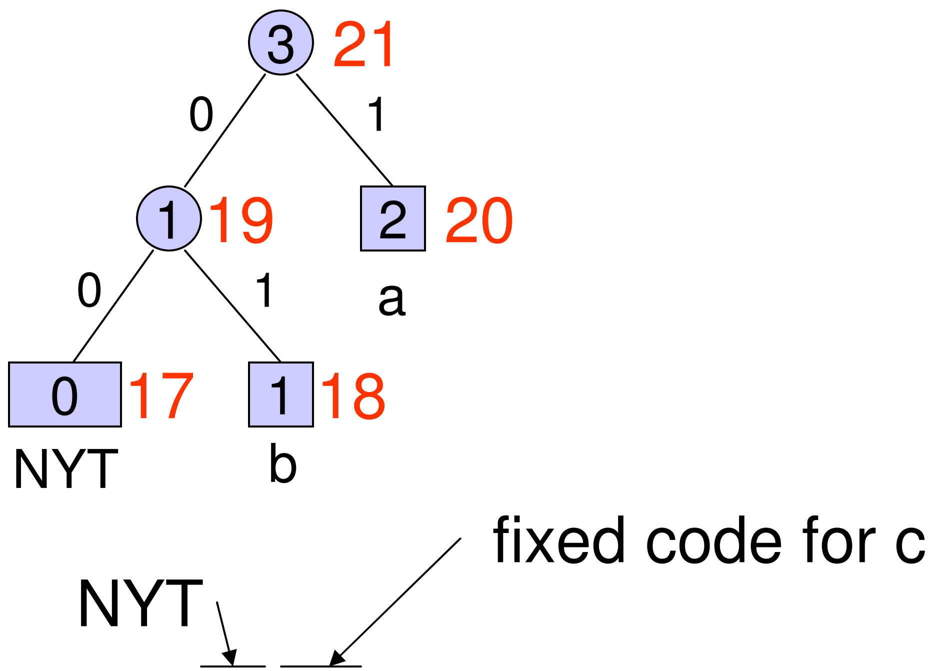
- aa<u>b</u>cdad



output = 00010001

Example

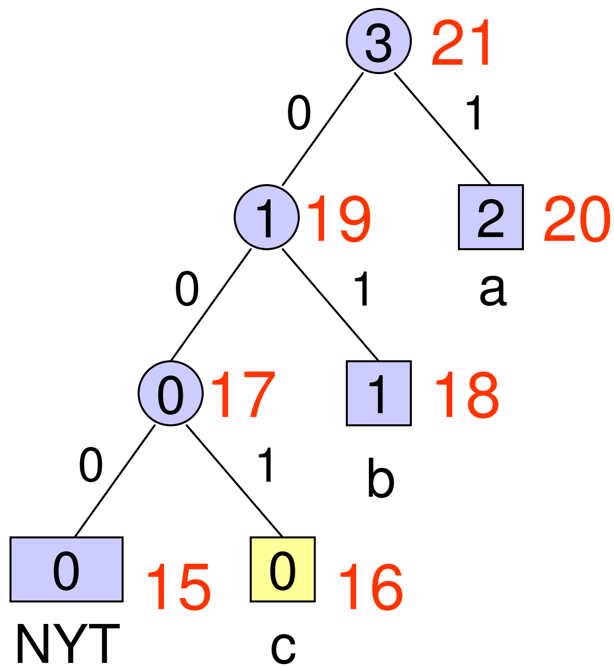
- aab<u>c</u>dad



output = 000100010<u>00010</u>

Example

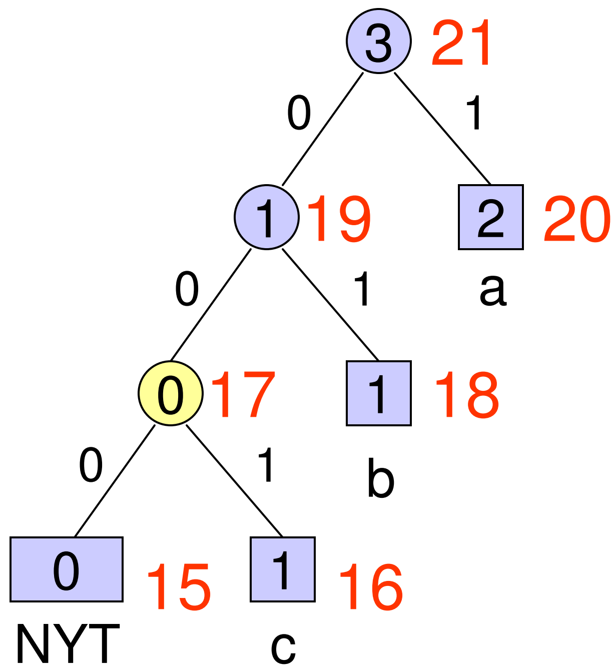
- aab<u>c</u>dad



output = 0001000100010

Example

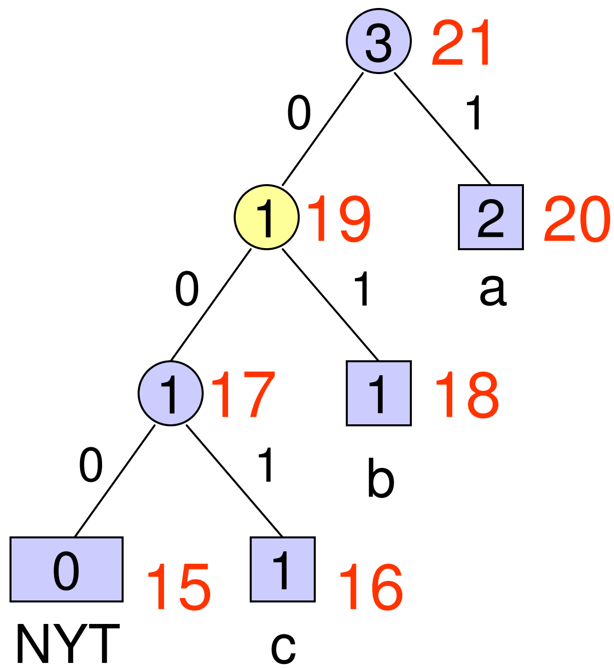
- aab<u>c</u>dad



output = 0001000100010

Example

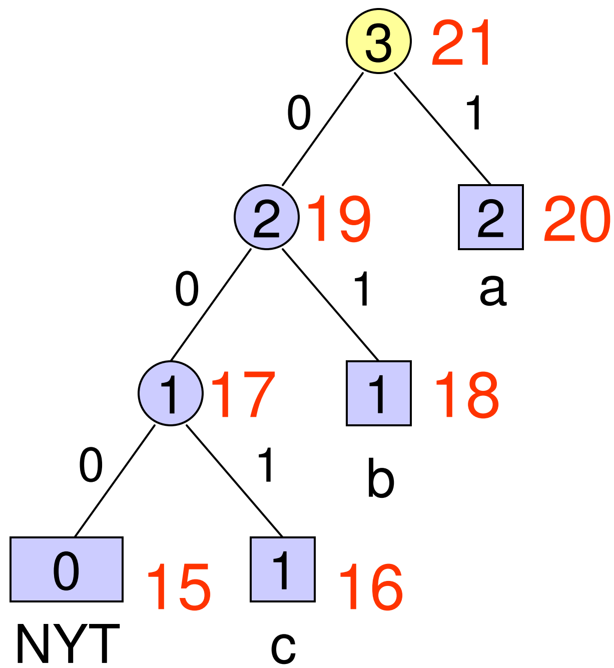
- aab<u>c</u>dad



output = 0001000100010

Example

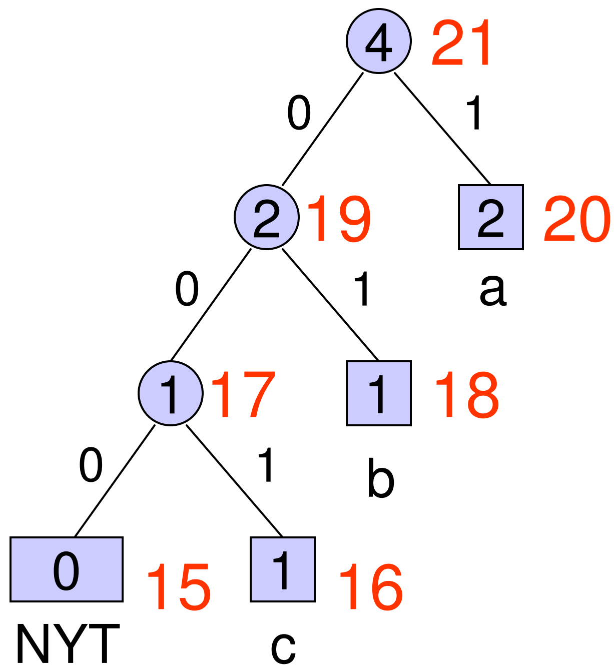
- aab<u>c</u>dad



output = 0001000100010

Example

- aabcdad



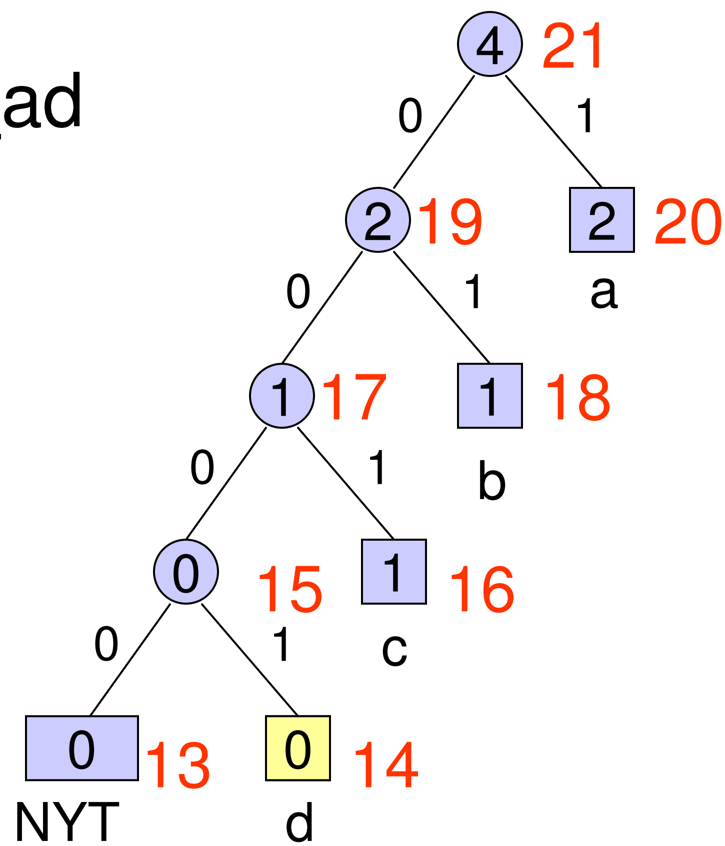
fixed code for d

NYT

output = 000100010001000001 1

Example

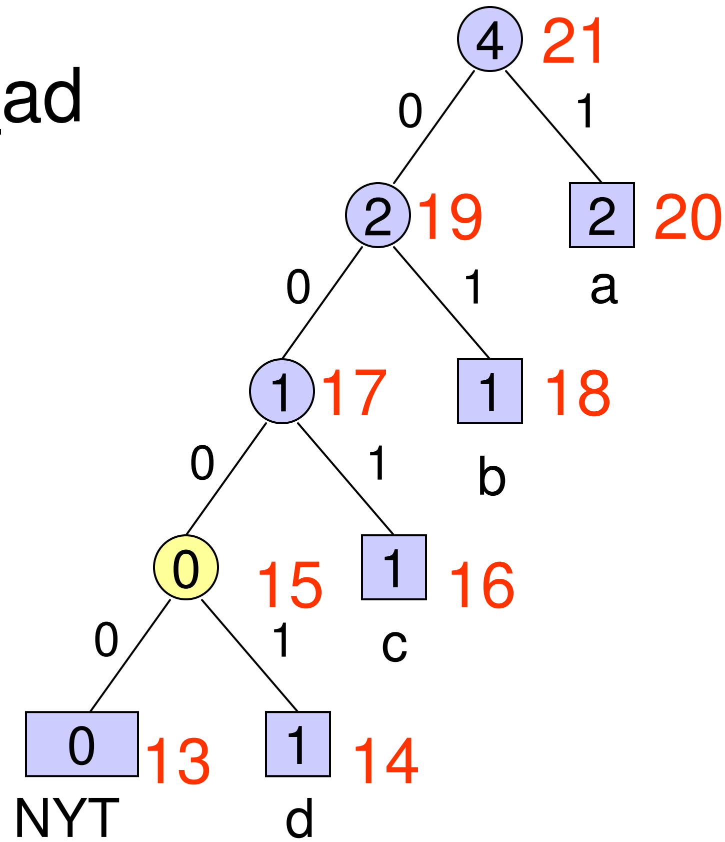
- aabcdad



output = 000100010001000011

Example

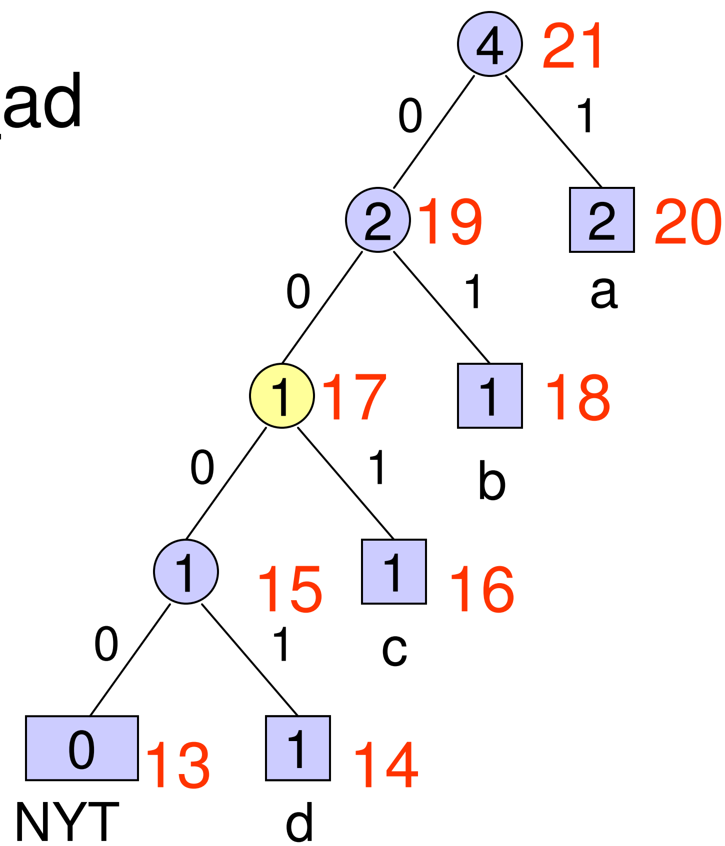
- aabc<u>d</u>ad



output = 000100010001000011

Example

- aabcdad

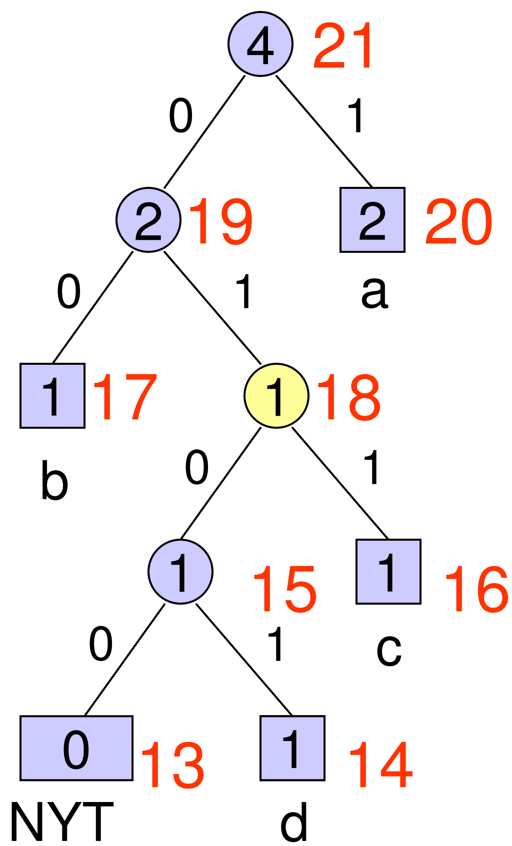


exchange!

output = 000100010001000011

Example

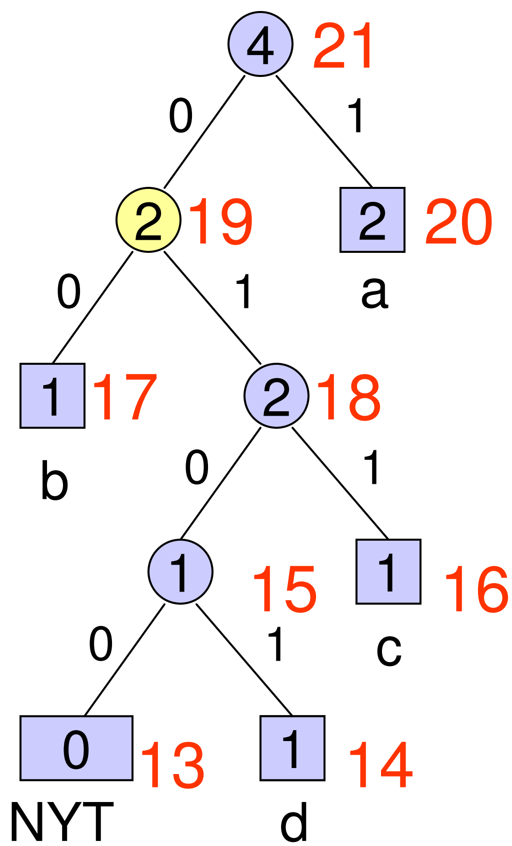
- aabc<u>d</u>ad



output = 0001000100010000011

Example

- aab*c*dad

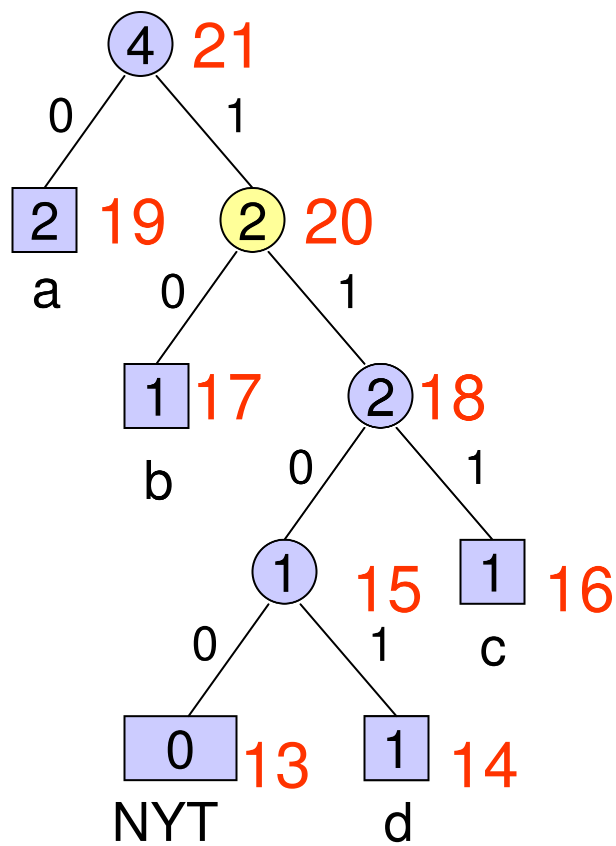


exchange!

output = 000100010001000011

Example

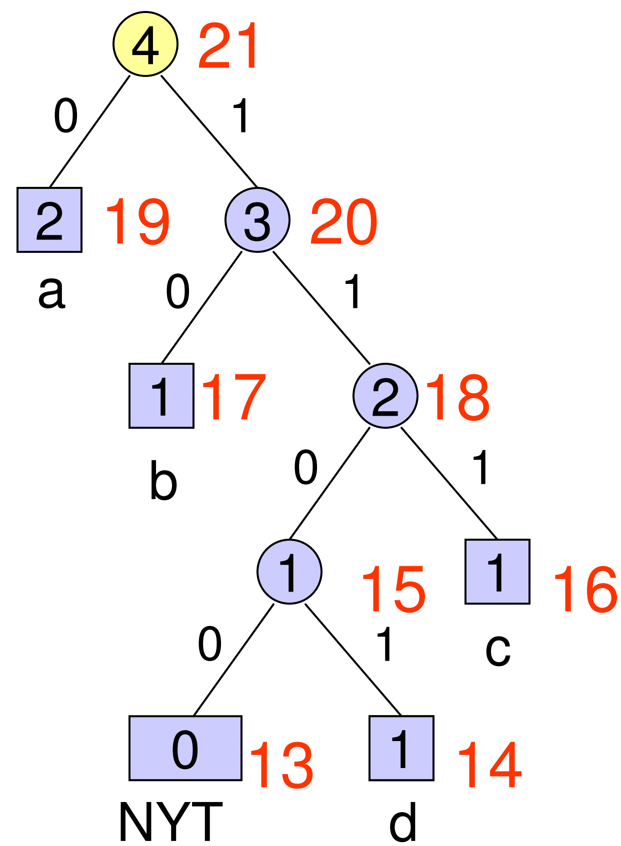
- aabc<u>d</u>ad



output = 00010001000100000 11

Example

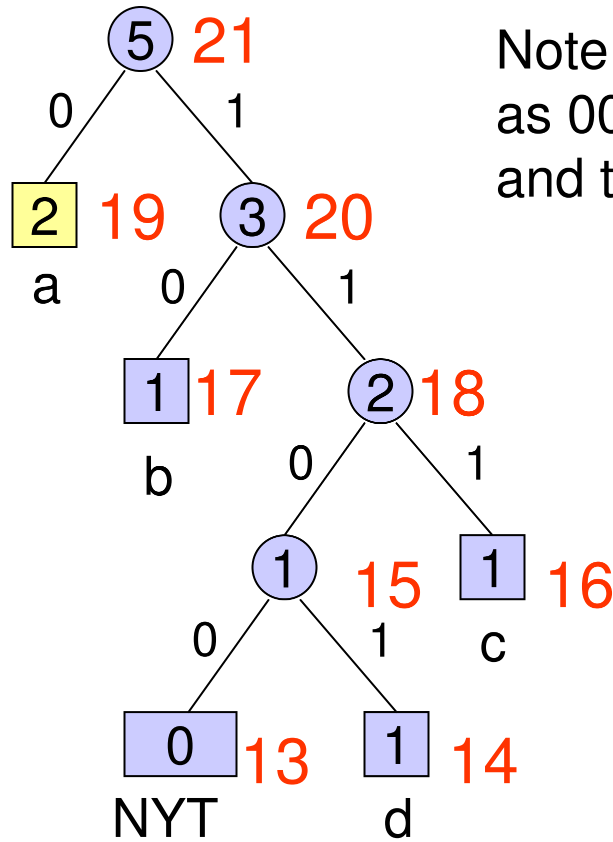
- aabc<u>d</u>ad



output = 0001000100010000011

Example

- aabcd<u>a</u>d

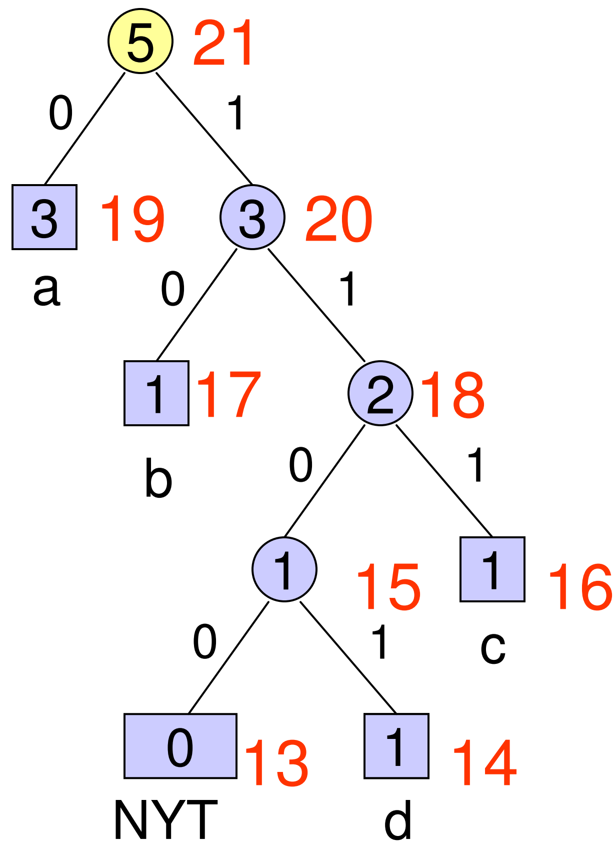


Note: the first a is coded as 000, the second as 1, and the third as 0

output = 000100010001000001<u>0</u>

Example

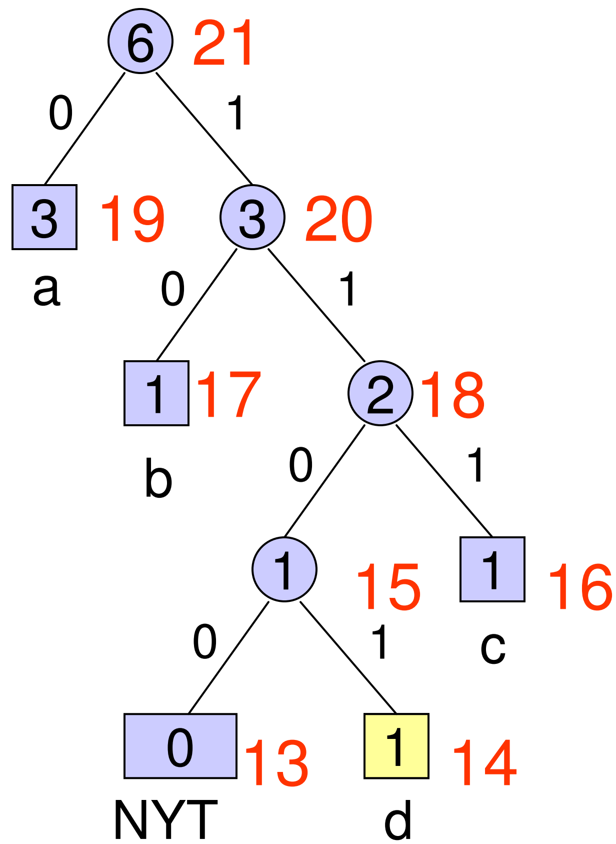
- aabcd<u>a</u>d



output = 0001000100010000011O

Example

- aabcda<u>d</u>

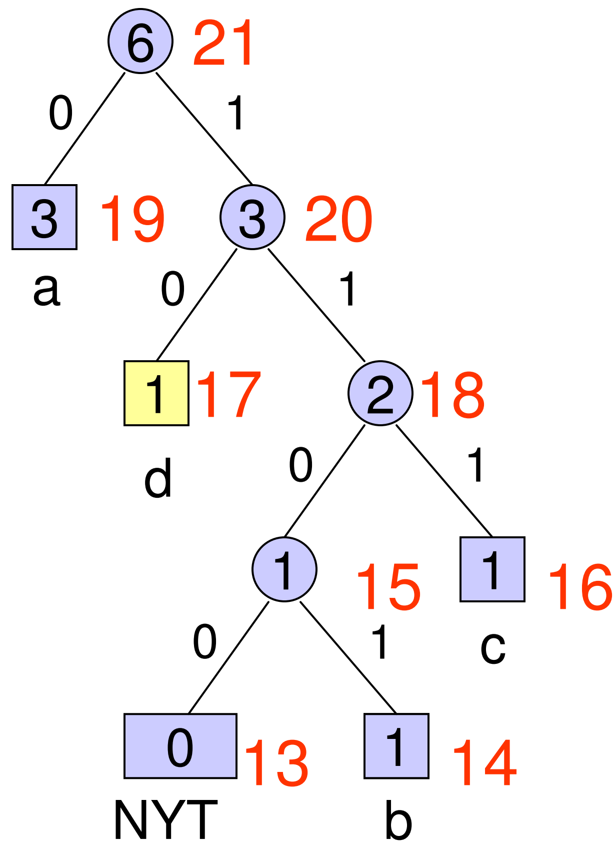


exchange!

output = 000100010001000001101<u>101</u>

Example

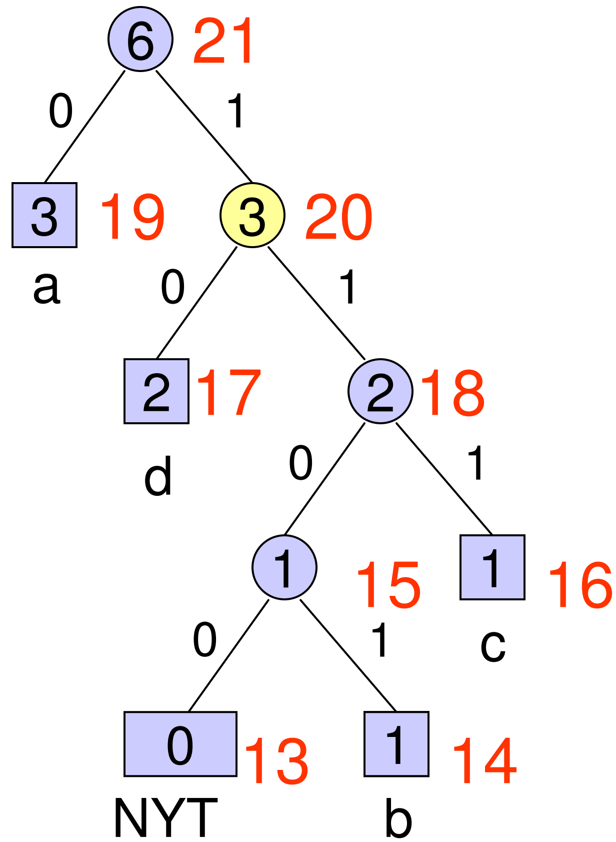
- aabcda<u>d</u>



output = 0001000100010000001101101

Example

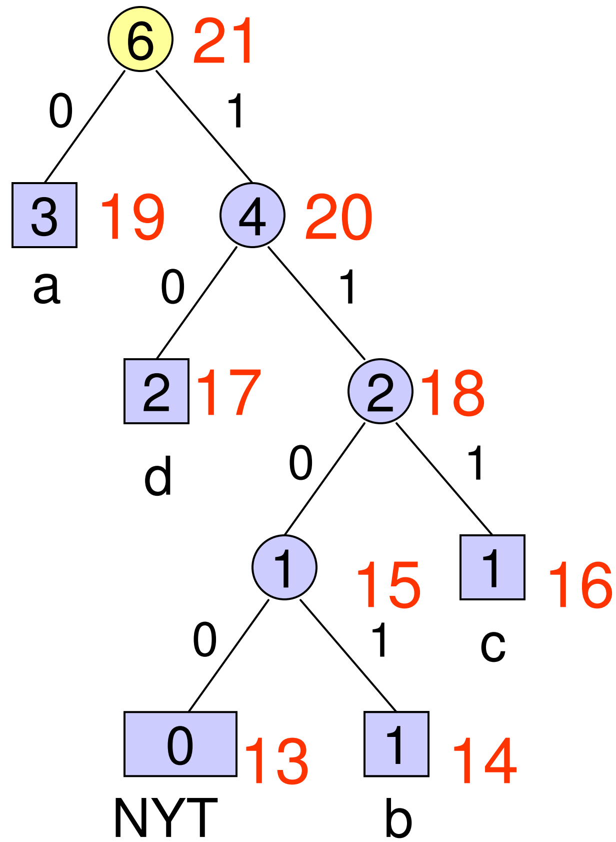
- aabcda<u>d</u>



output = 0001000100010000001101101

Example

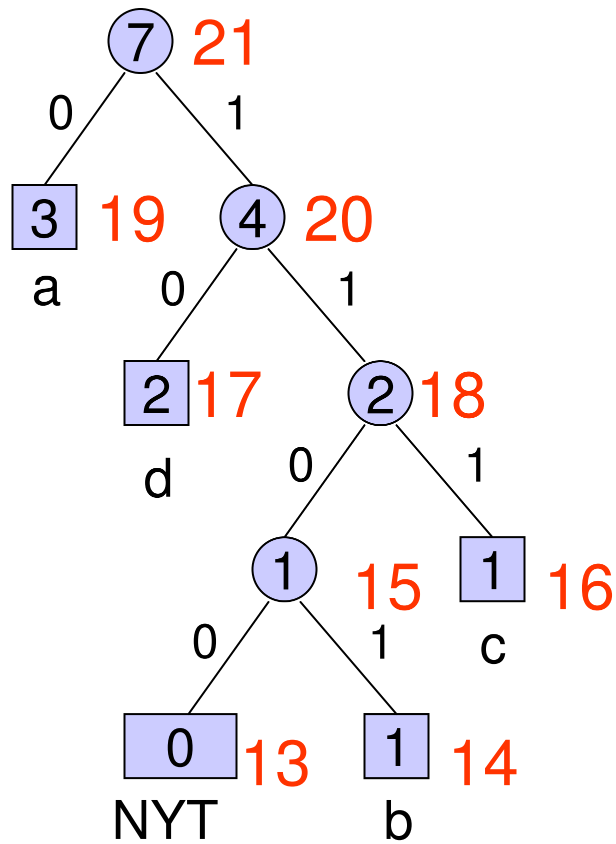
- aabcda<u>d</u>



output = 0001000100010000001101101

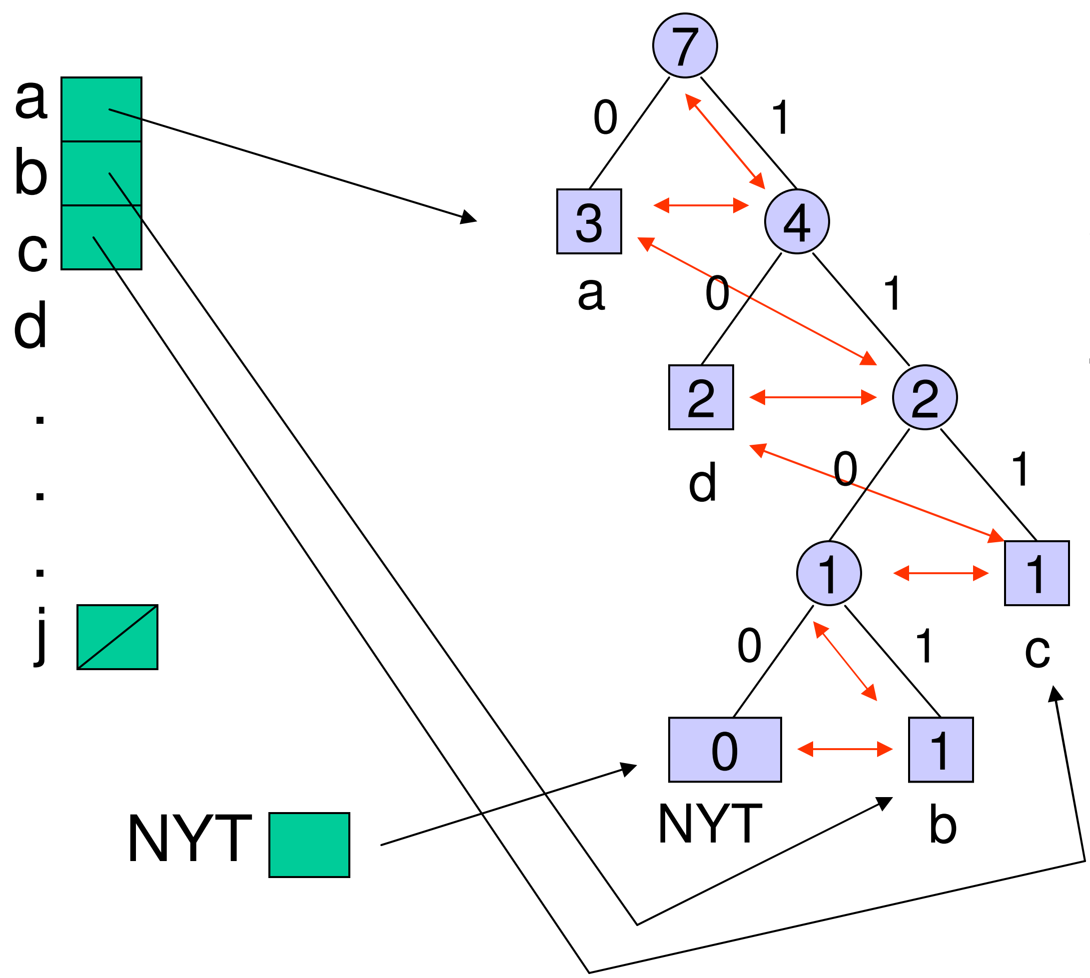
Example

- aabcdad



output = 00010001000100000110110 1

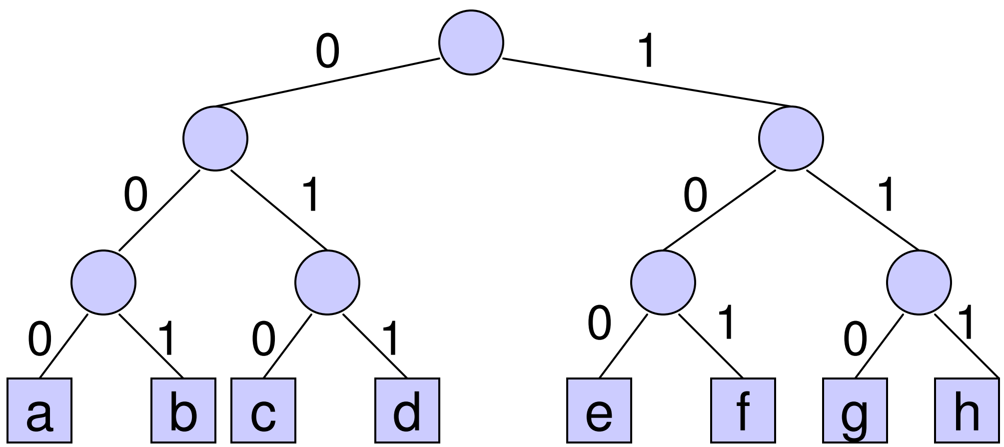
Data Structure for Adaptive Huffman



1. Fixed code table
2. Binary tree with parent pointers
3. Table of pointers nodes into tree
4. Doubly linked list to rank the nodes

In Class Exercise

- Decode using adaptive Huffman coding assuming the following fixed code



- 00110000

Huffman Summary

- Statistical compression algorithm
- Prefix code
- Fixed-to-variable length code
- Optimization to create a best code
- Symbol merging
- Context
- Adaptive coding
- Decoder and encoder behave almost the same
- Need for data structures and algorithms