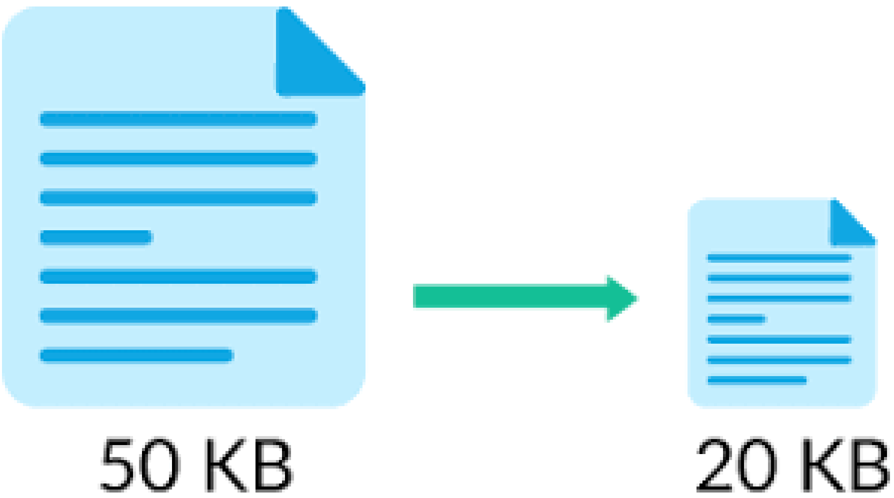
# File Compression by Huffman codes

**Application of Priority queues**

# Why Compress Files?




# ASCII TABLE

https://simple.wikipedia.org/wiki/ASCII

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|
| 32 | 20 | *[SPACE]* | 64 | 40 | @ | 96 | 60 | ` |
| 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | *[DEL]* |

The normal ASCII character set consists of roughly 100 "printable" characters.

# ASCII TABLE

https://simple.wikipedia.org/wiki/ASCII

The normal ASCII character set consists of roughly 100 "printable" characters.

- How many bits are required to distinguish 100 characters ?
  - If we take 6 bits we can have $2^6 = 64$
  - If we take 7 bits we can have $2^7 = 128$

To distinguish 100 characters, **$\log_2 100$ =** $\lceil 6.6 \rceil$ **= 7** bits are required.

**The important point is that if the size of the character set is *C*, then $\lceil log C \rceil$ bits are needed in a standard encoding.**

# ASCII TABLE

https://simple.wikipedia.org/wiki/ASCII

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

Seven bits allow the representation of 128 characters, so the ASCII character set adds some other “nonprintable” characters

# File encoding

FILE

is ieee
tie tea
sat sit set

**29 characters in this file**
**If we use ASCII encoding**
**29*7 = 203 digits**

- The above file contains only the characters *a, e, i, s, t,* plus blank-spaces and *newlines*
- **How many bits needed to represent above characters?**
  - $Log_2 7 = 3$ bits

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | | |
| e | | | |
| i | | | |
| s | | | |
| t | | | |
| sp | | | |
| nl | | | |

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | | |
| e | 001 | | |
| i | 010 | | |
| s | 011 | | |
| t | 100 | | |
| sp | 101 | | |
| nl | 110 | | |

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | |
| e | 001 | | |
| i | 010 | | |
| s | 011 | | |
| t | 100 | | |
| sp | 101 | | |
| nl | 110 | | |

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | |
| e | 001 | 6 | |
| i | 010 | | |
| s | 011 | | |
| t | 100 | | |
| sp | 101 | | |
| nl | 110 | | |

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | |
| e | 001 | 6 | |
| i | 010 | 4 | |
| s | 011 | 4 | |
| t | 100 | 3 | |
| sp | 101 | 7 | |
| nl | 110 | 3 | |

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

FILE

is ieee
tie tea
sat sit set

**29 characters in this file**
**If we use ASCII encoding**
**29*7 = 203 digits**

**If we use 3 bit encoding**
**29*3 = 87 digits**

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

010

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

010011

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

010011101

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

010011101010

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

0100111010100010010001

| Character | Code | Frequency | Total Bits |
| --- | --- | --- | --- |
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Encoding

FILE

is ieee
tie tea
sat sit set

0100111010100010010011101110

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Decoding

010011101010001001001101110

i

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Decoding

0100111010100010010011101110

is

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Decoding

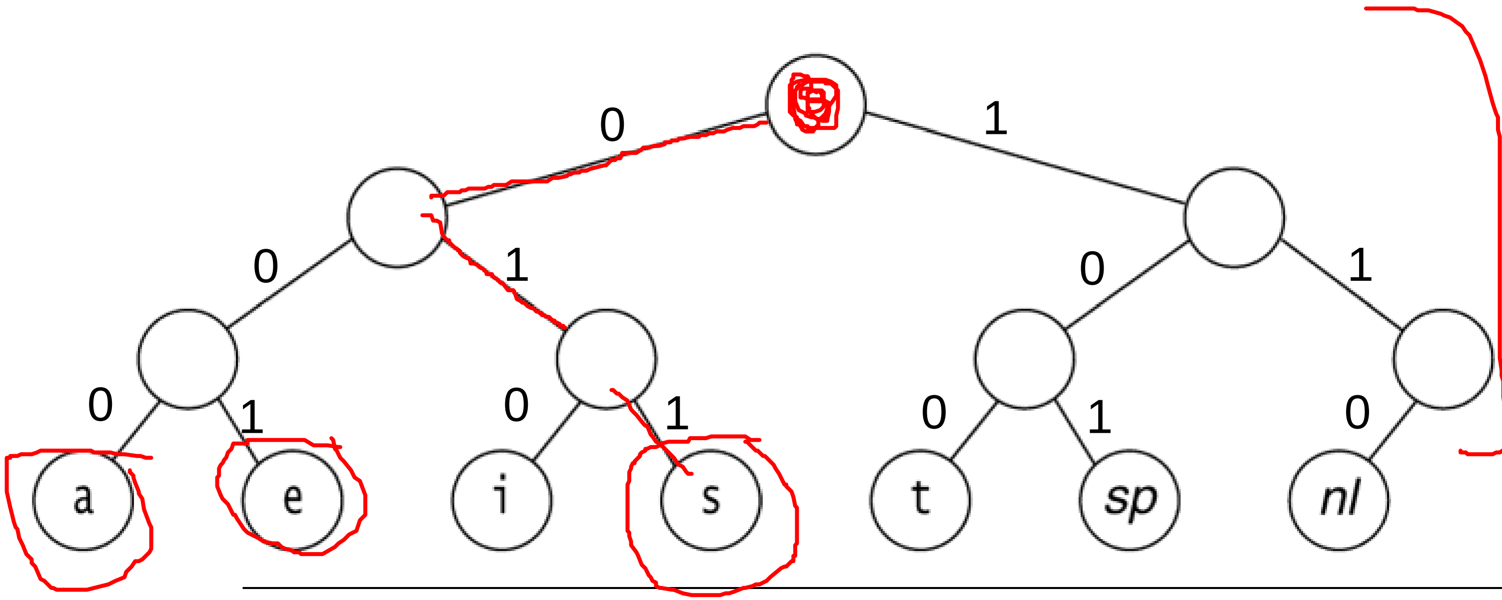
010011101010001001001101110

is ieee

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 2 | 6 |
| e | 001 | 6 | 18 |
| i | 010 | 4 | 12 |
| s | 011 | 4 | 12 |
| t | 100 | 3 | 9 |
| sp | 101 | 7 | 21 |
| nl | 110 | 3 | 9 |

# Represent code by a binary tree

- **The binary tree has data only at the leaves.**
- The representation of each character can be found by
  - Start at the root, a 0 to indicates the left branch and a 1 indicates the right branch.
- For instance, code 011 will give *s*
- This data structure is sometimes referred to as a **trie.**

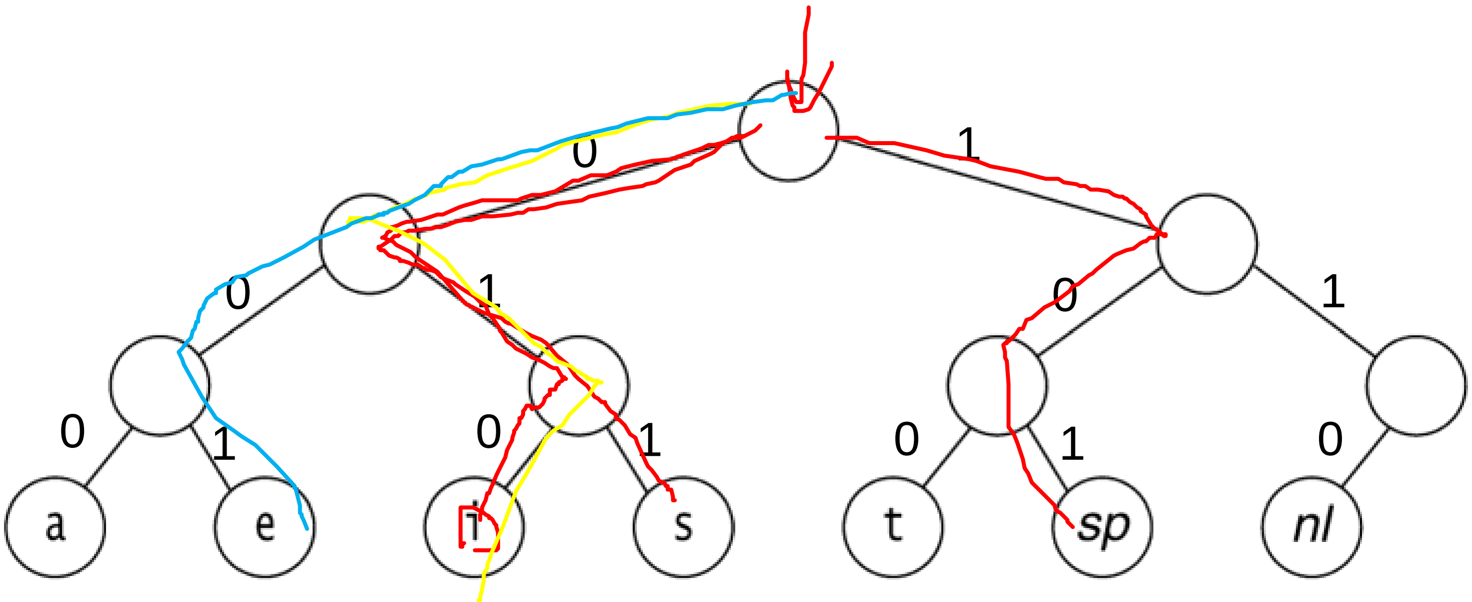


# Represent code by a binary tree

010011101010001001001101110

i s __ i e e e

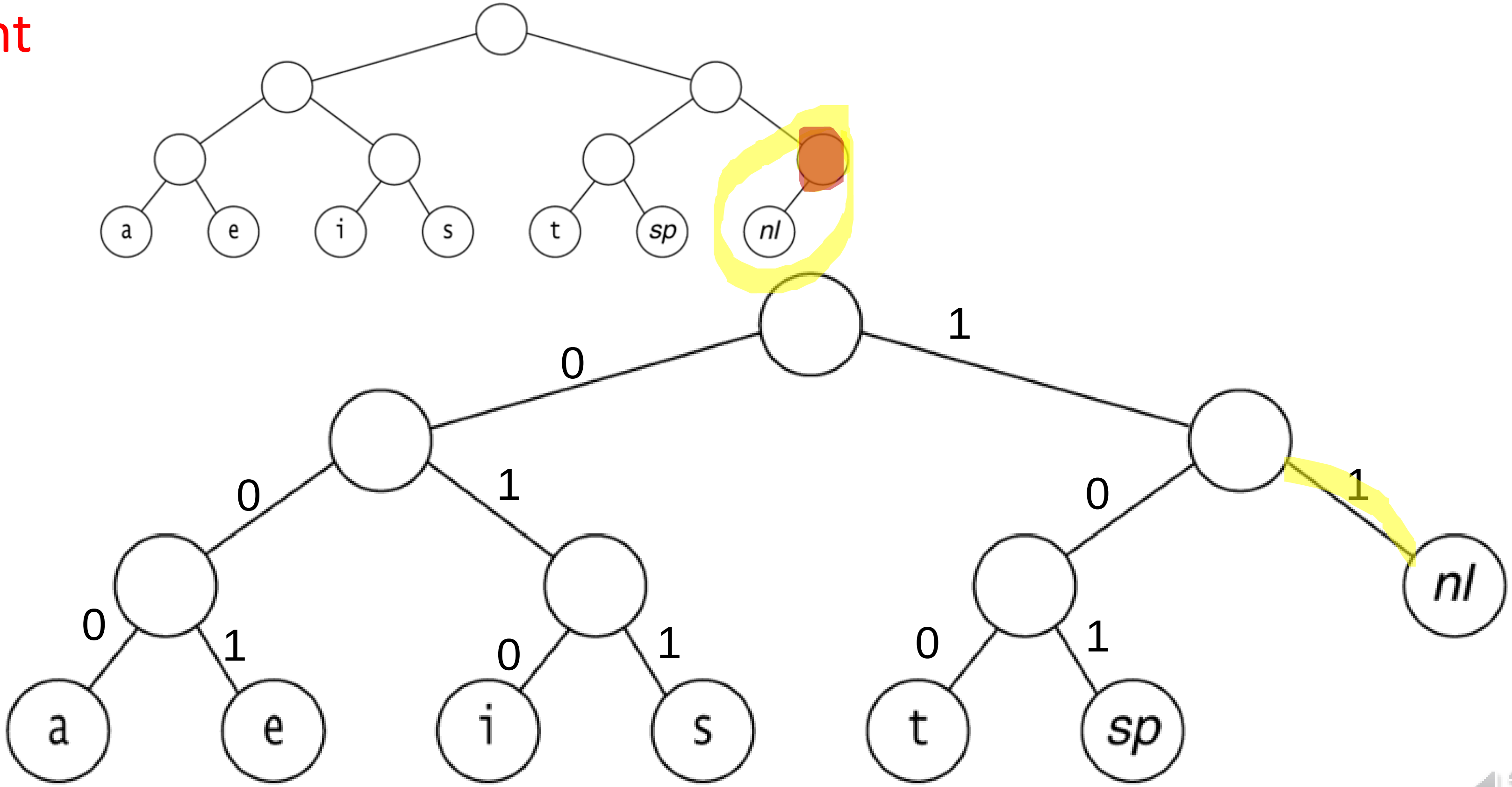
is ieee



# A slightly better tree

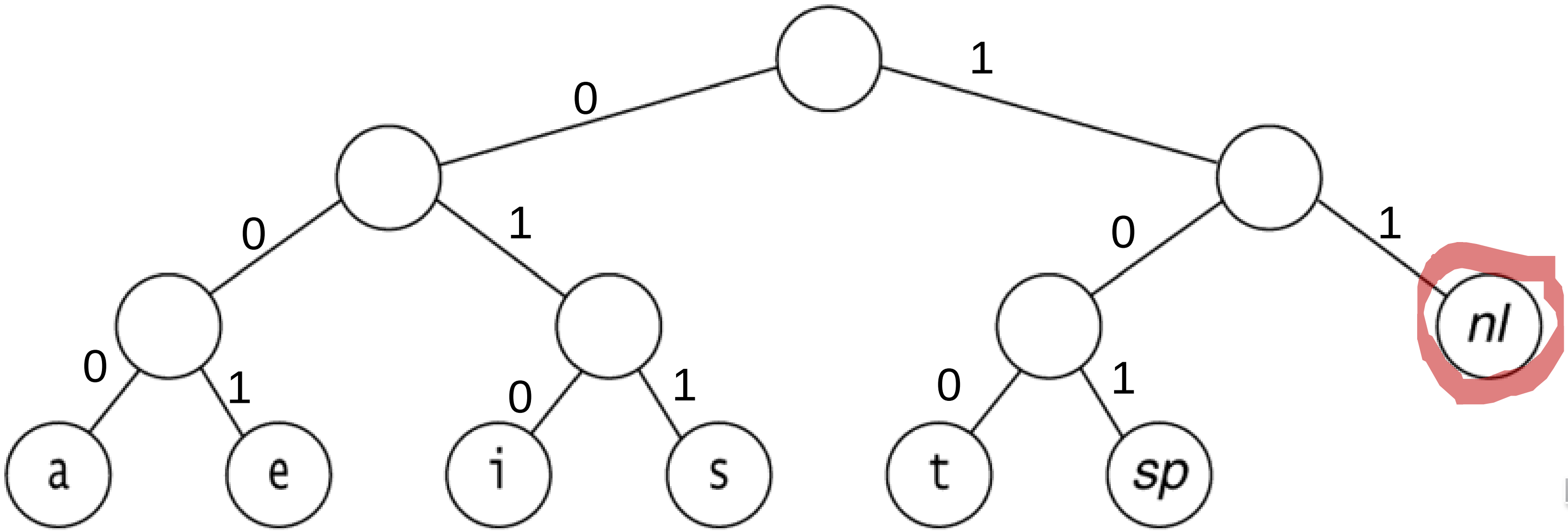
A better code than the one given before can be obtained by noticing that the ***newline* is an only child.**

So we can place the *newline* symbol one level higher at its parent




# Decode unambiguously

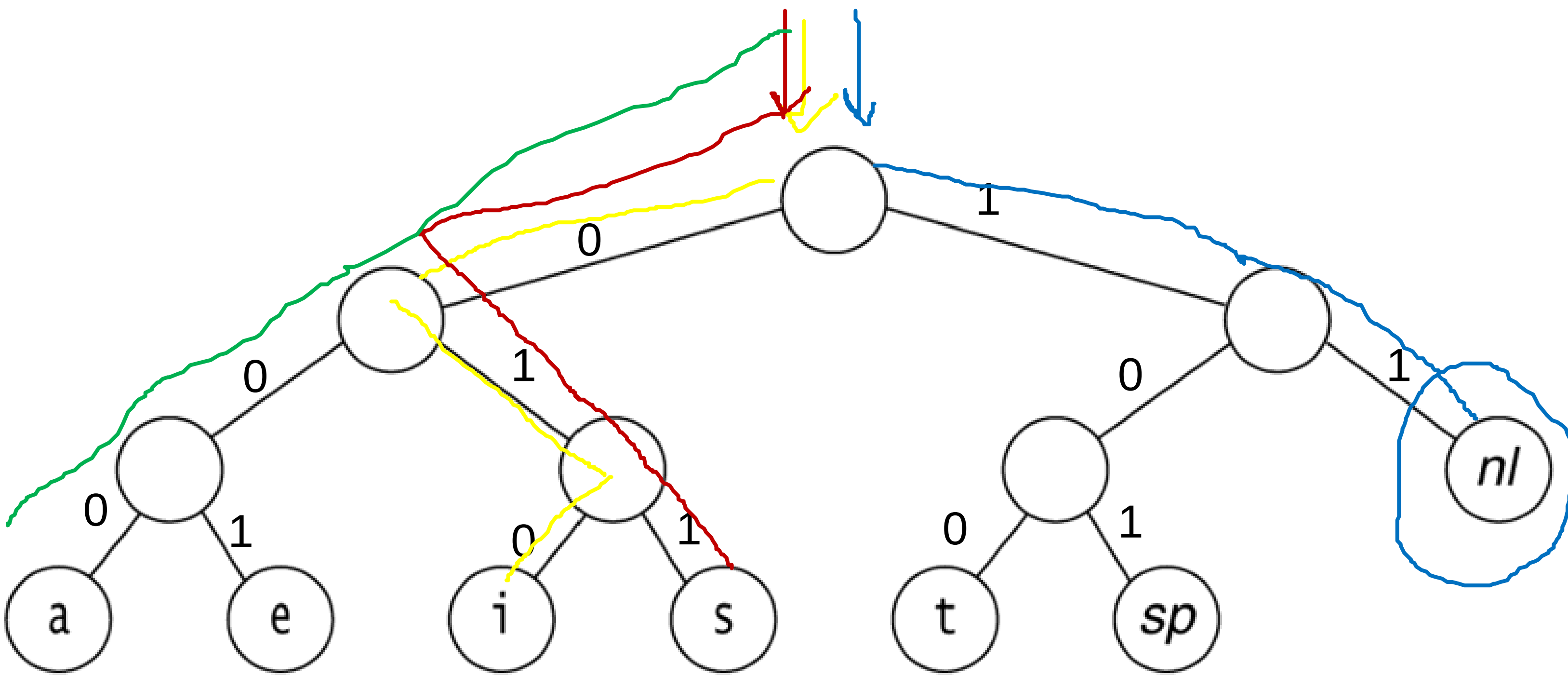
- If the characters are placed only at the leaves, any sequence of bits can always be decoded unambiguously.



# Decode unambiguously

- If the characters are placed only at the leaves, any sequence of bits can always be decoded unambiguously.
  - For instance, suppose **010011110001011000100011 1** is the encoded string.

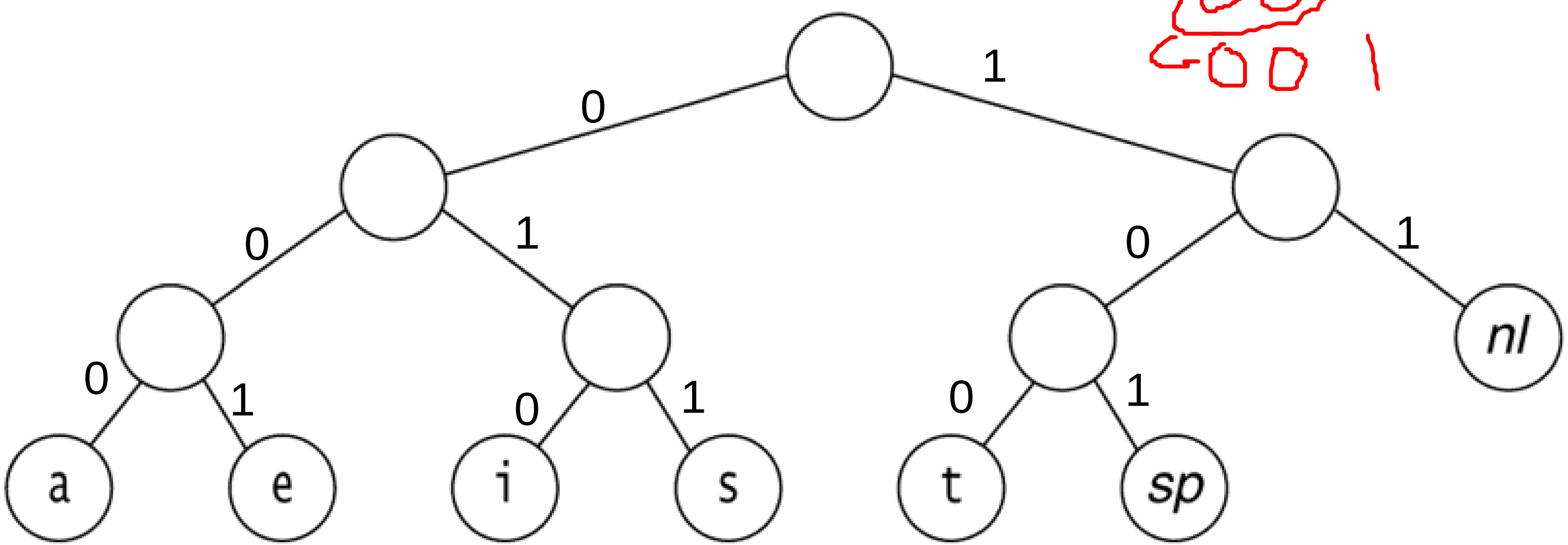
i s a nl



# Decode unambiguously

- If the characters are placed only at the leaves, any sequence of bits can always be decoded unambiguously.

- **Thus, it does not matter if the character codes are different lengths, as long as no character code is a <u>prefix</u> of another character code.**



# Decode

- Conversely, if a character is contained in a nonleaf node, it is no longer possible to guarantee that the decoding will be unambiguous.

- Putting these facts together, we see that our basic problem is to find the **full binary tree** of minimum total cost (as defined above), where all characters are contained in the leaves.

# Disparity between the most frequent & least frequent characters

- In real life, files can be quite large.
- There is usually a big disparity between the **most frequent and least frequent characters**.
- For instance, many large data files have an inordinately large amount of **digits**, blanks, and *newlines*, but few *q*'s and *x*'s.
- **We want to compress the file.**

# Standard coding scheme

- How many bits needed to encode the file with following specs using standard coding scheme

| Character | Code | Frequency | Total Bits |
|---|---|---|---|
| a | 000 | 10 | 30 |
| e | 001 | 15 | 45 |
| i | 010 | 12 | 36 |
| s | 011 | 3 | 9 |
| t | 100 | 4 | 12 |
| *sp* | 101 | 13 | 39 |
| *nl* | 110 | 1 | 3 |
| **Total** | | | **174** |

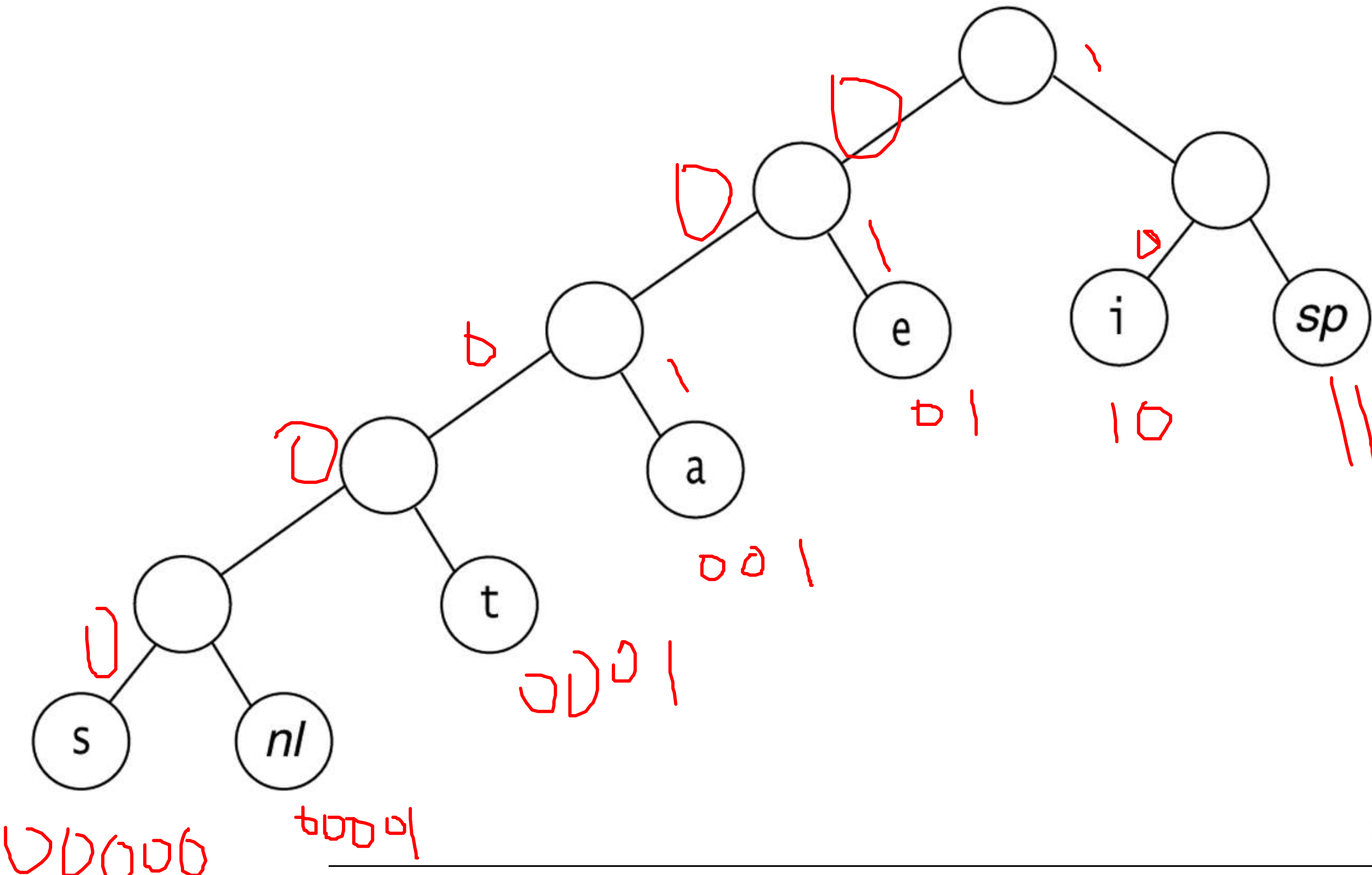
# Optimal prefix code

| Character | Code | Frenquency | Total Bits |
|---|---|---|---|
| a | 001 | 10 | 30 |
| e | 01 | 15 | 30 |
| i | 10 | 12 | 24 |
| s | 00000 | 3 | 15 |
| t | 0001 | 4 | 16 |
| *sp* | 11 | 13 | 26 |
| *nl* | 00001 | 1 | 5 |
| **Total** | | | 146 |

# Optimal prefix code

| Character | Code | Frenquency | Total Bits |
|---|---|---|---|
| a | 001 | 10 | 30 |
| e | 01 | 15 | 30 |
| i | 10 | 12 | 24 |
| s | 00000 | 3 | 15 |
| t | 0001 | 4 | 16 |
| *sp* | 11 | 13 | 26 |
| *nl* | 00001 | 1 | 5 |
| **Total** | | | **146** |

# An optimal prefix code tree



# Examples

# Coding theory

| | | |
|---|---|---|
| A | | |
| B | | |
| C | | |
| D | | |
| E | | |
| F | | |

For **fixed-length binary** coding of a 6-character alphabet, how many bits are needed?

# Decode the following

| | |
|---|---|
| E | 0 |
| T | 11 |
| N | 100 |
| I | 1010 |
| S | 1011 |

11010010010101011

| | |
|---|---|
| E | 0 |
| T | 10 |
| N | 100 |
| I | 0111 |
| S | 1010 |

100100

# Prefix(-free) codes

- No prefix of a codeword is a codeword

- Uniquely decodable

| | A | B | C |
|---|---|---|---|
| A | 00 | 1 | 00 |
| B | 010 | 01 | 10 |
| C | 011 | 001 | 11 |
| D | 100 | 0001 | 0001 |
| E | 11 | 00001 | 11000 |
| F | 101 | 000001 | 101 |

# Prefix codes and Binary trees

Draw the Tree representation of given prefix codes

| | |
|---|---|
| A | 00 |
| B | 010 |
| C | 0110 |
| D | 0111 |
| E | 10 |
| F | 11 |

# Huffman codes

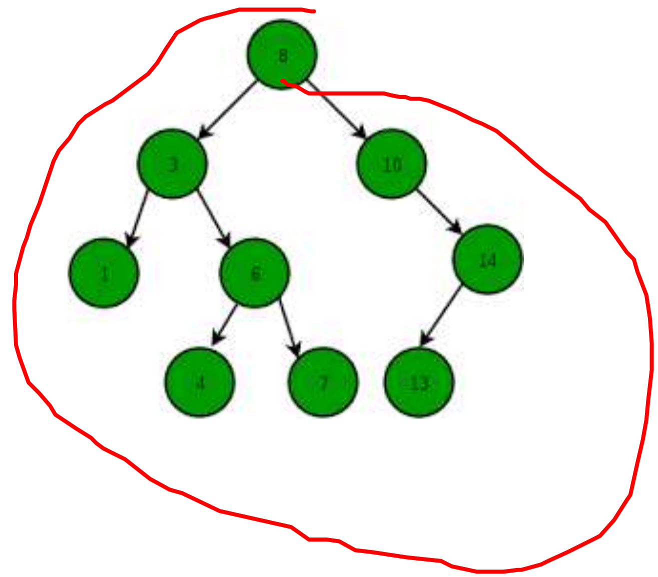
Mark Allen Weiss

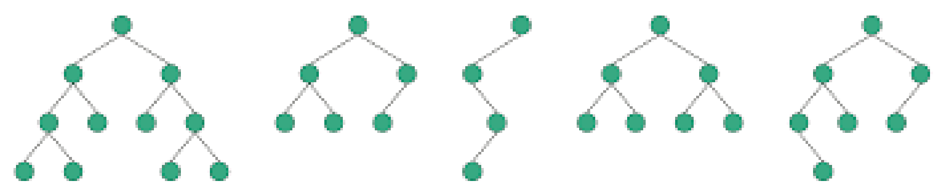
# Huffman's Algorithm

- Huffman's algorithm Basic Idea:
  - We maintain a forest of trees.


# Huffman's Algorithm

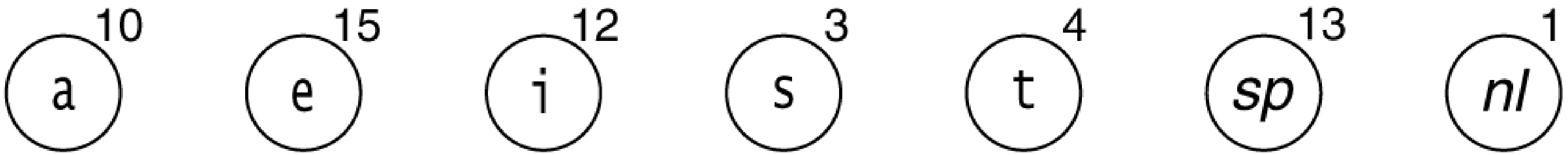
- Huffman's algorithm Basic Idea:
  - We maintain a forest of trees.

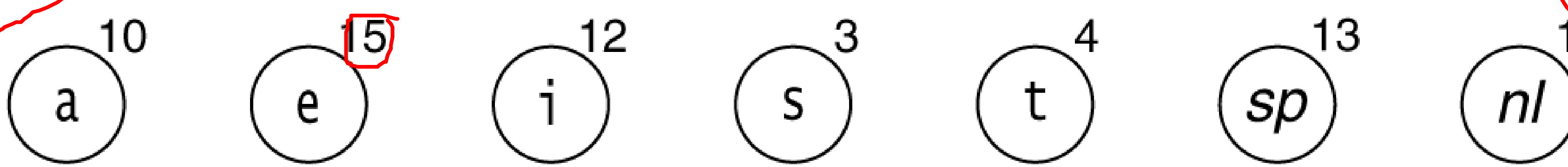
# Huffman's Algorithm

- Huffman's algorithm can be described as follows:
  - We maintain a forest of trees.
  - **The weight of a tree is equal to the sum of the frequencies of its leaves.**



# Huffman's Algorithm

- Huffman's algorithm can be described as follows:
  - We maintain a forest of trees.
  - **The weight of a tree is equal to the sum of the frequencies of its leaves.**



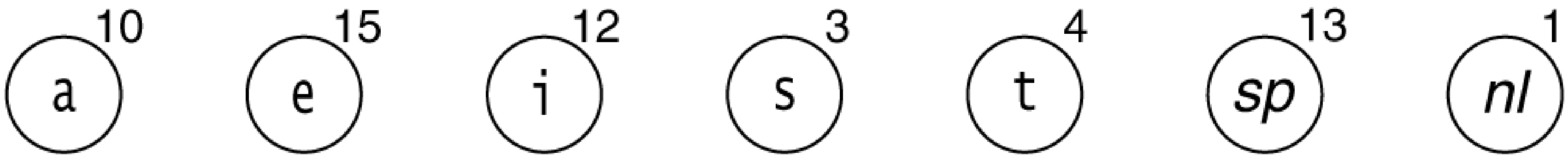
# Huffman's Algorithm

- Huffman's algorithm can be described as follows:
  - We maintain a forest of trees.
  - The *weight* of a tree is equal to the sum of the frequencies of its leaves.
  - ***For N*-1 times,** *(where N is the number of characters)*
    - **select the two trees, *T*1 and *T*2, of smallest weight, breaking ties arbitrarily, and**
    - **form a new tree with subtrees *T*1 and *T*2.**

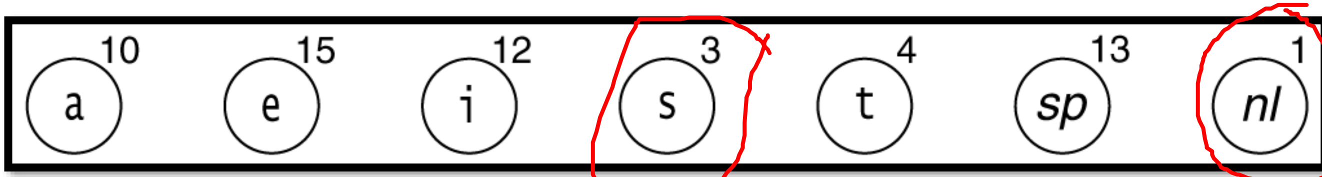
# Initial stage of Huffman's algorithm

At the beginning of the algorithm, there are N single-node trees—one for each character.

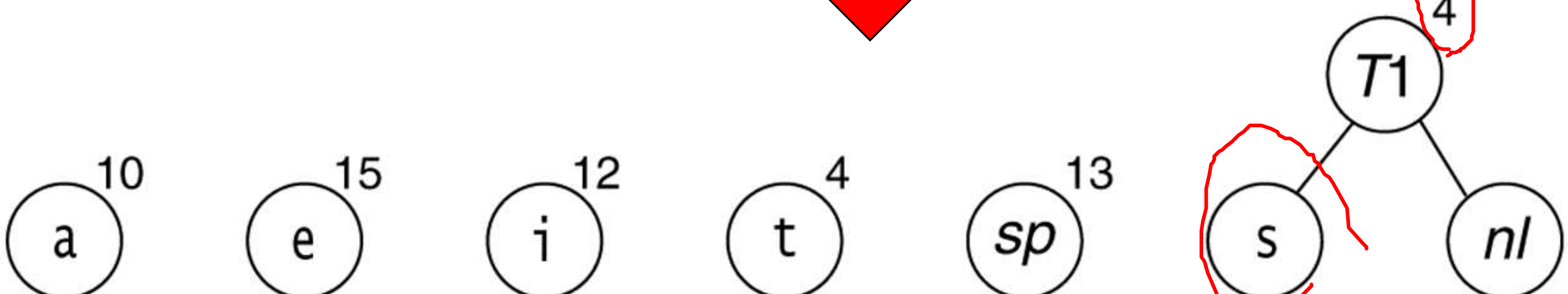
At the end of the algorithm there is one tree, and this is the optimal Huffman coding tree.



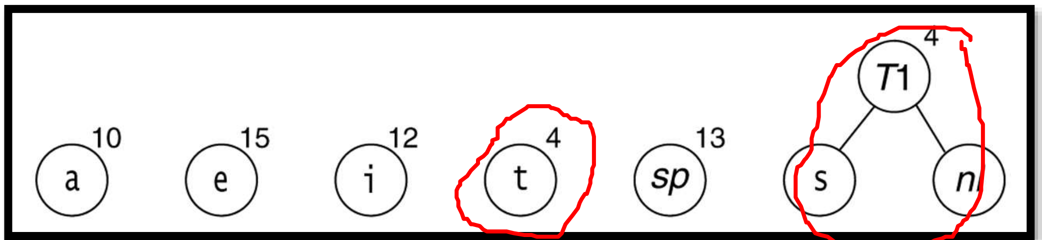
# Huffman's algorithm after the first merge



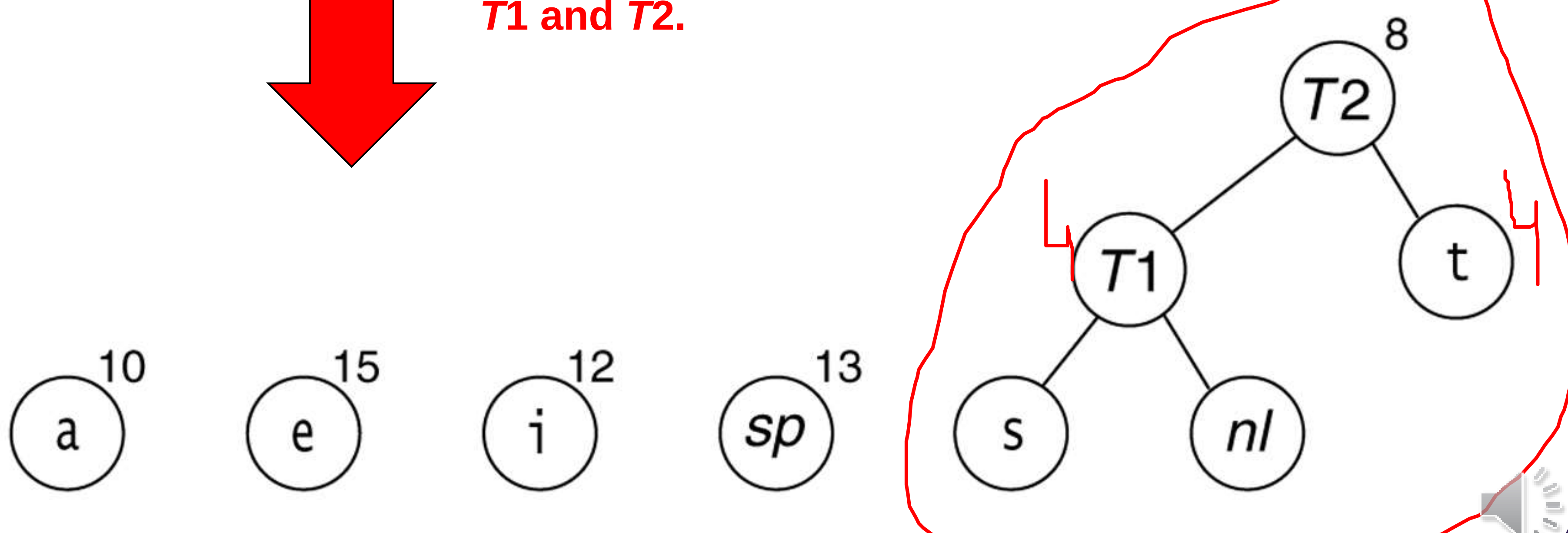

**select the two trees, *T*1 and *T*2, of smallest weight, and form a new tree with subtrees *T*1 and *T*2.**



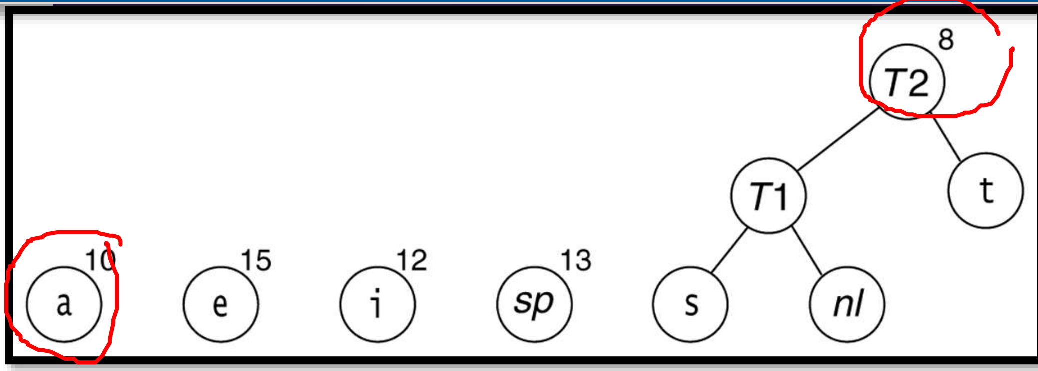
# Huffman's algorithm after the second merge



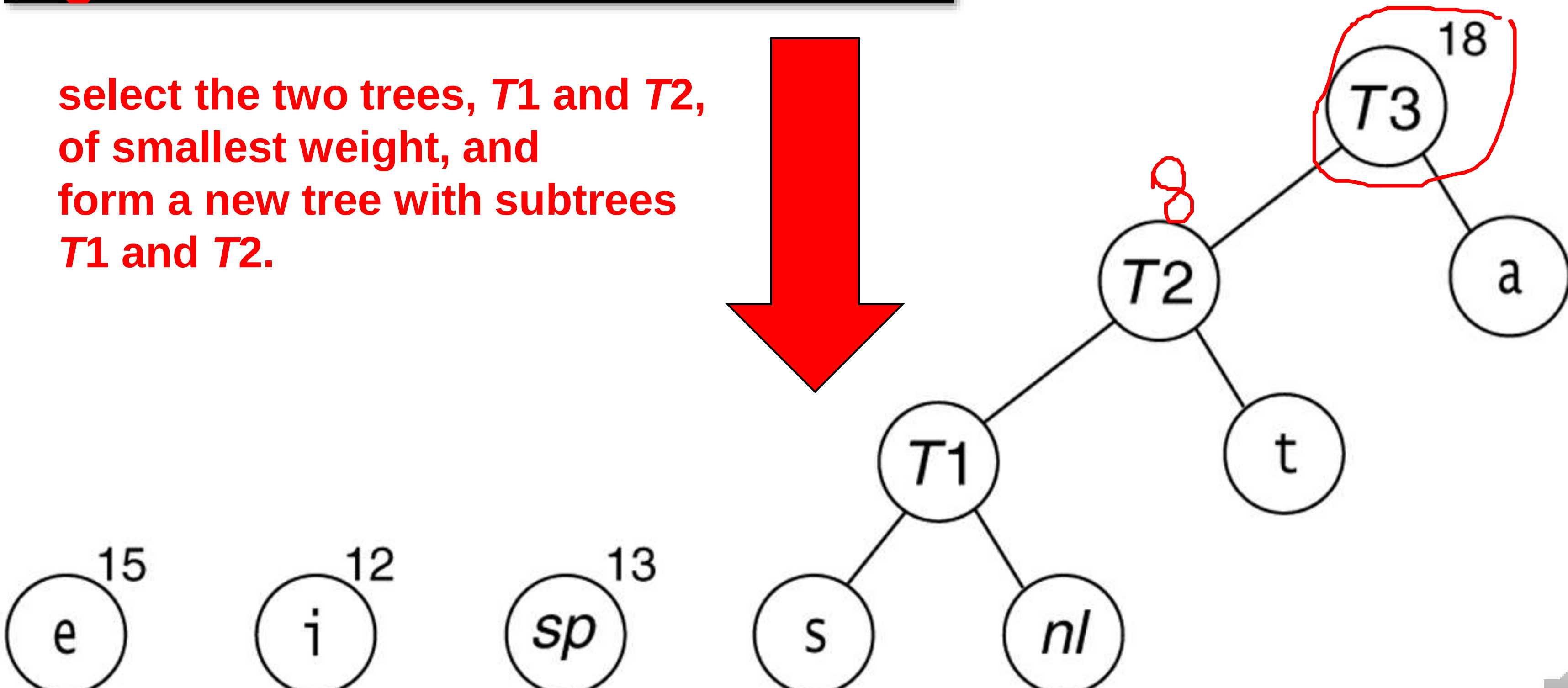

**select the two trees, *T*1 and *T*2, of smallest weight, and form a new tree with subtrees *T*1 and *T*2.**



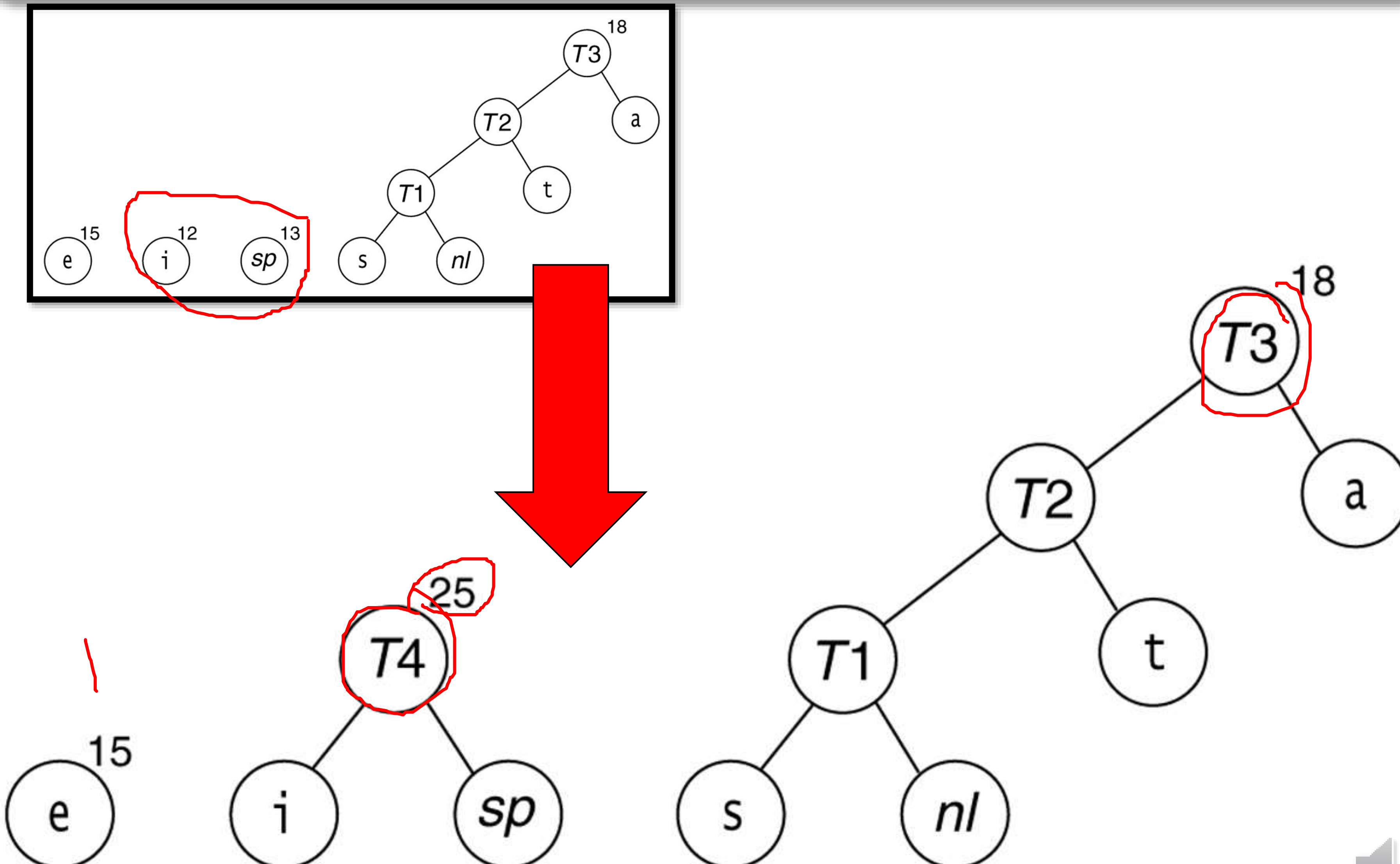
# Huffman's algorithm after the third merge



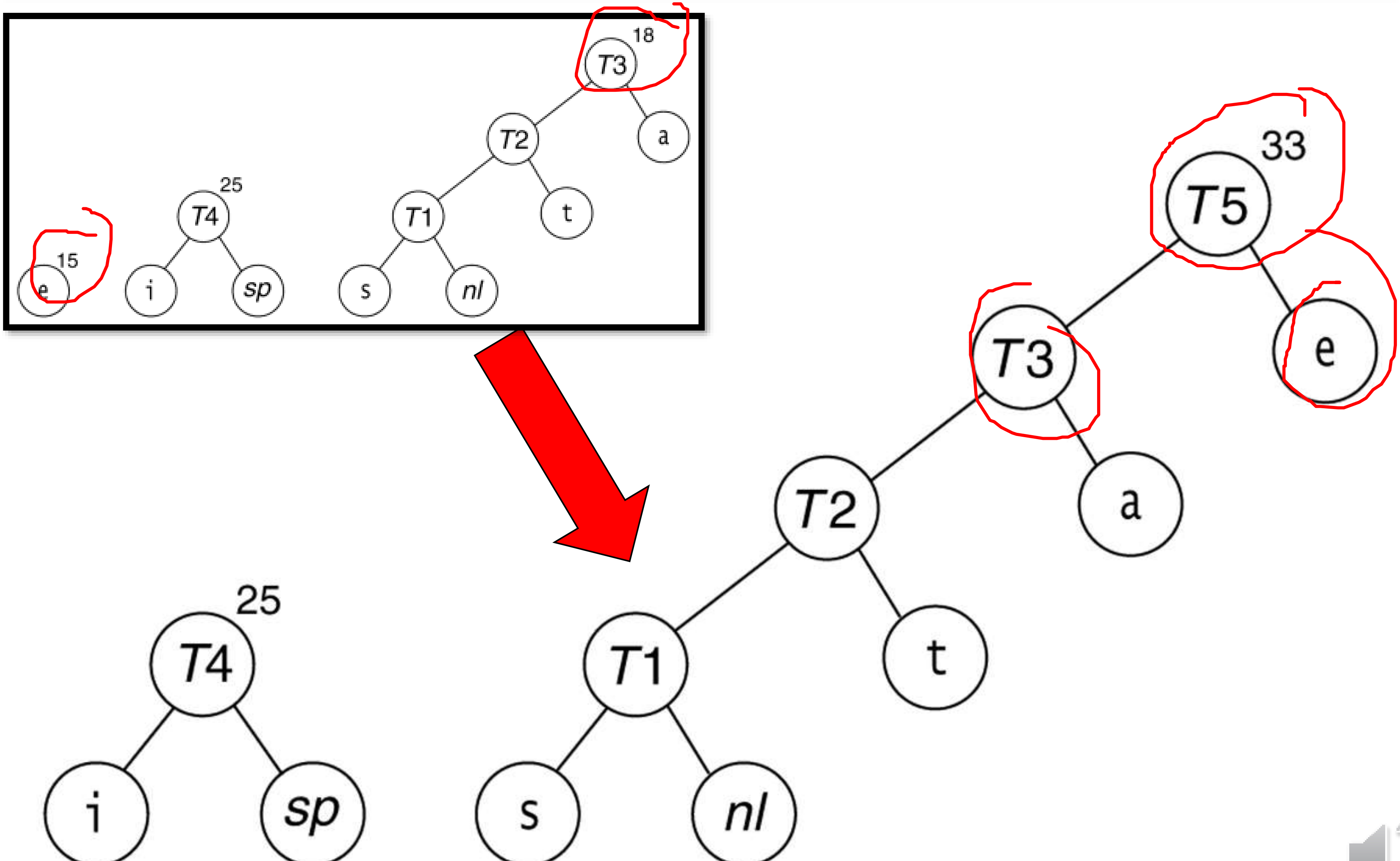

**select the two trees, *T*1 and *T*2, of smallest weight, and form a new tree with subtrees *T*1 and *T*2.**



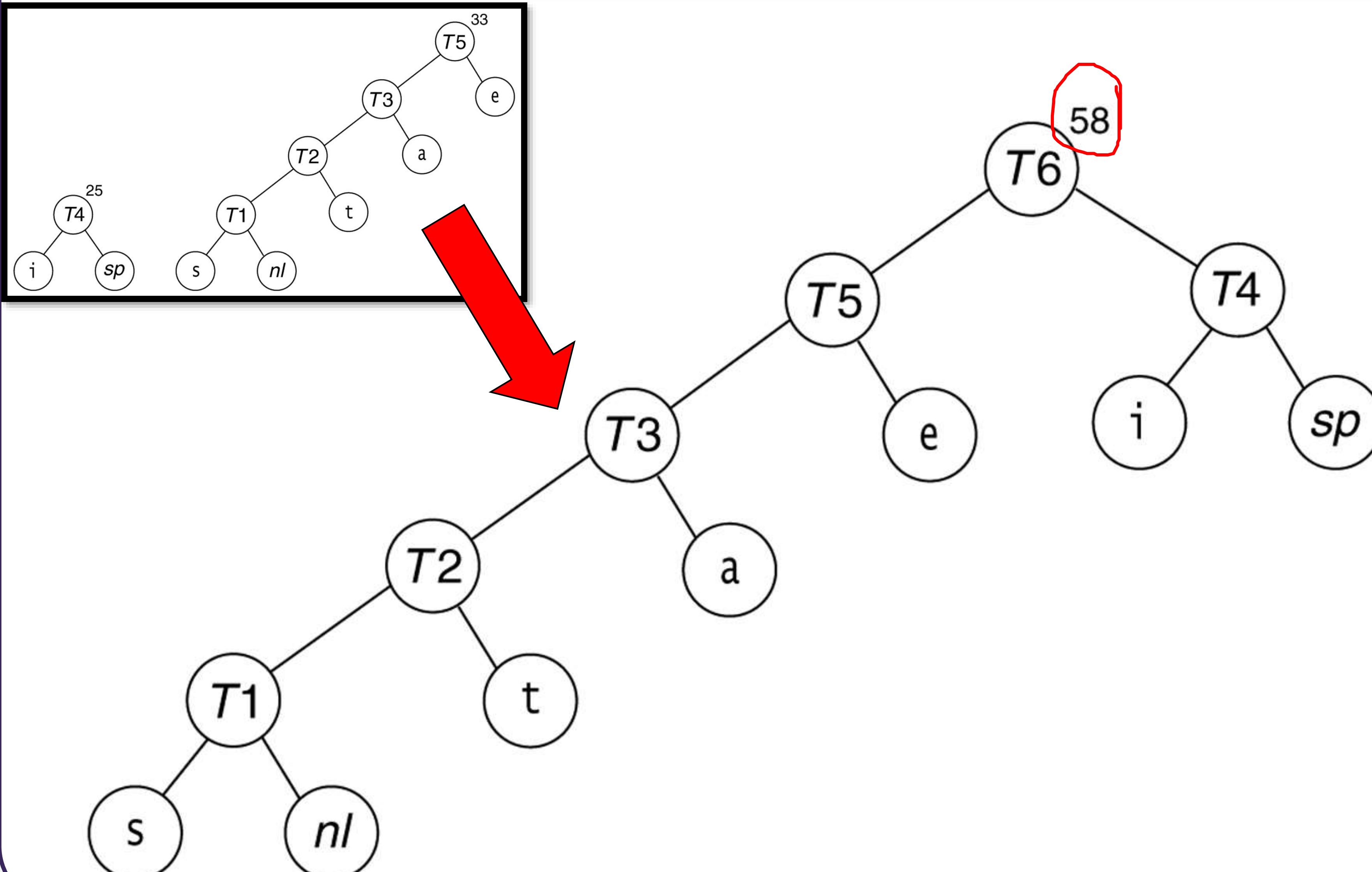
# Huffman's algorithm after the Fourth merge



# Huffman's algorithm after the fifth merge

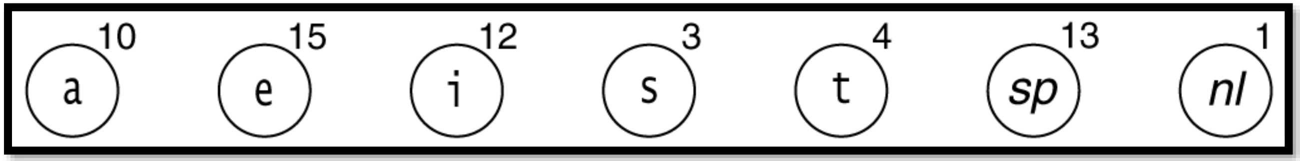


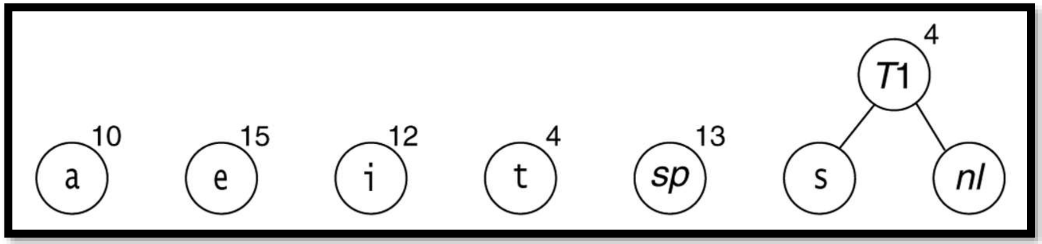
# Huffman's algorithm after the final merge



# How to implement?

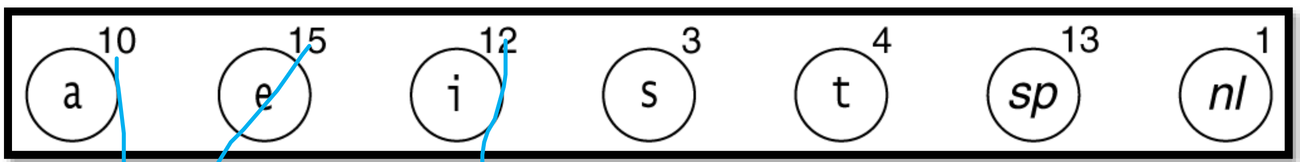
- **Use Priority Queue**
- Maintain the trees in a priority queue, ordered by weight



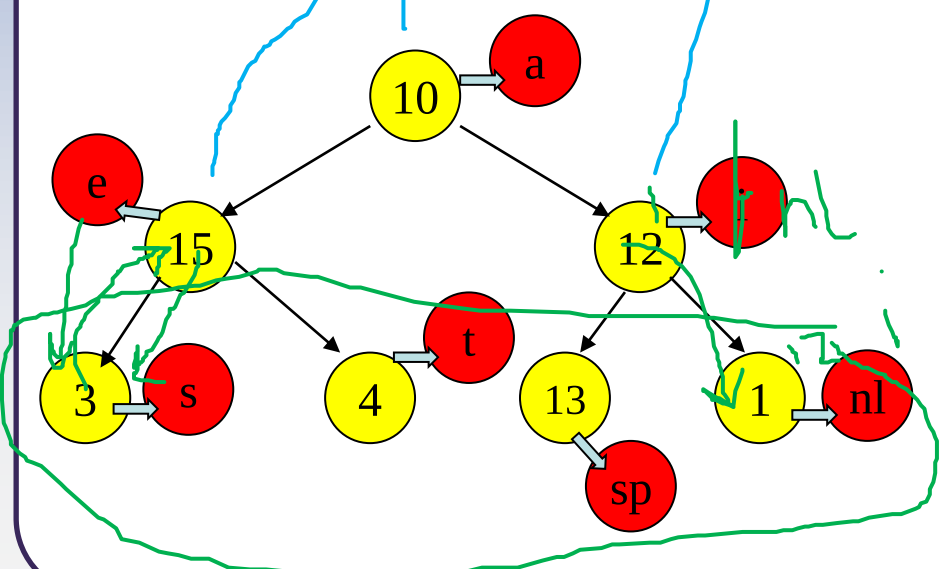



# How to implement?

- Priority Queue must be implemented using **Minheap**

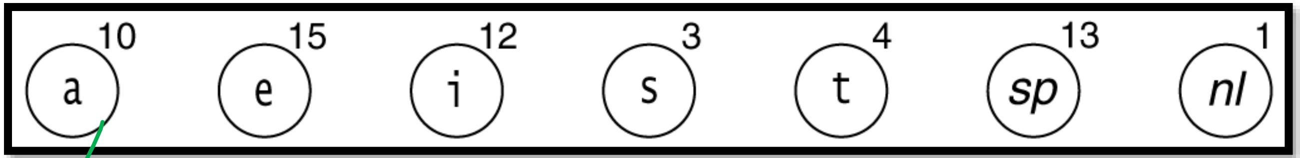



**But characters and frequency in an array and buildHeap**

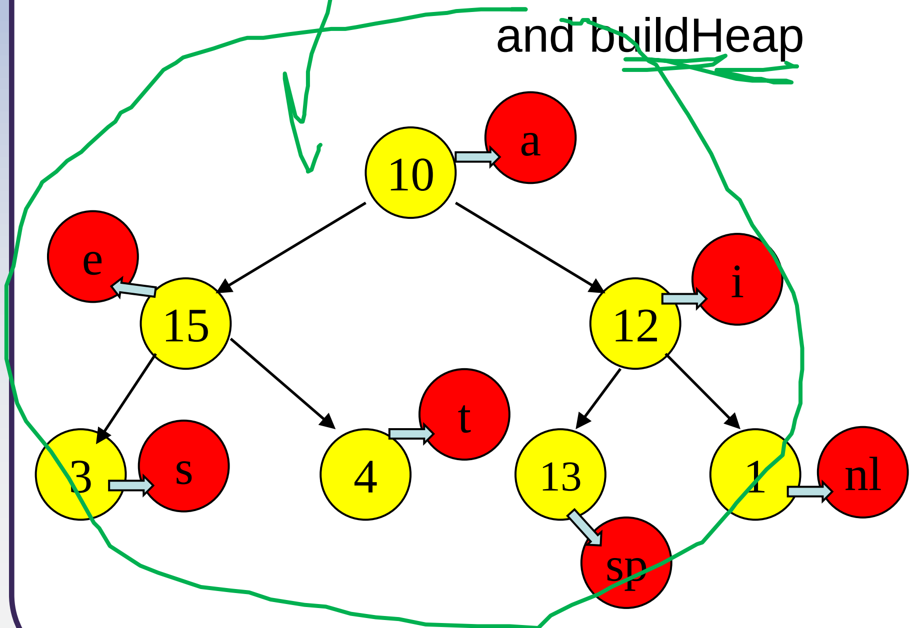


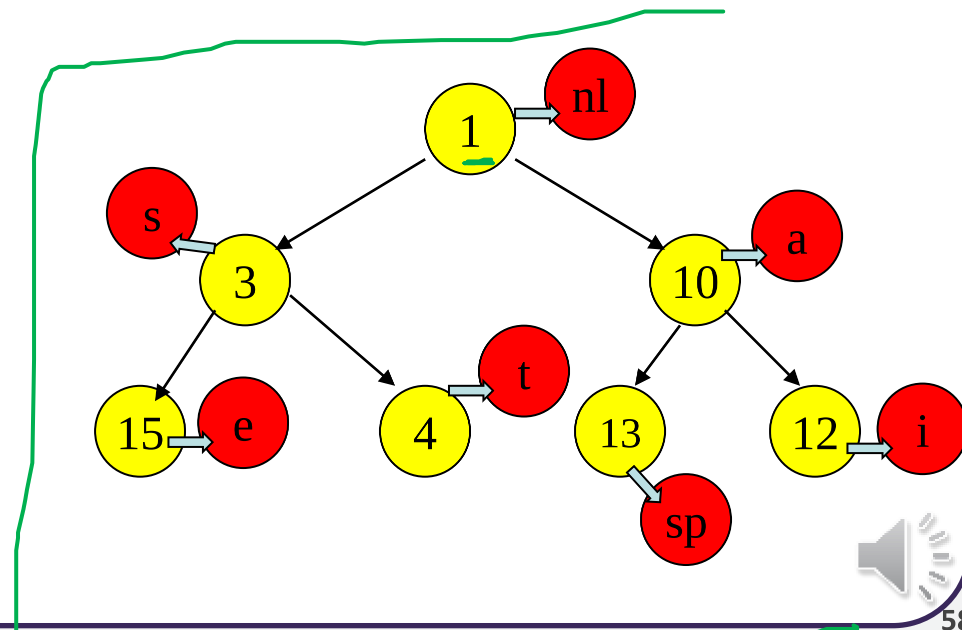
# How to implement?

- Priority Queue must be implemented using Minheap




But characters and frequency in an array
and buildHeap



# Time Complexity

- If we maintain the trees in a priority queue, ordered by weight, then the running time will be ?
  - there will be one buildHeap, $O(N \log N) \longrightarrow O(N)$
  - 2N-1 deleteMins, and $O(N \log N)$
  - N- 2 inserts $O(N \log N)$
  - $O(N \log N)$,

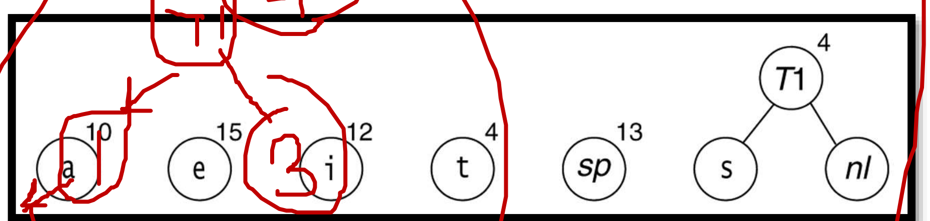
# Time Complexity

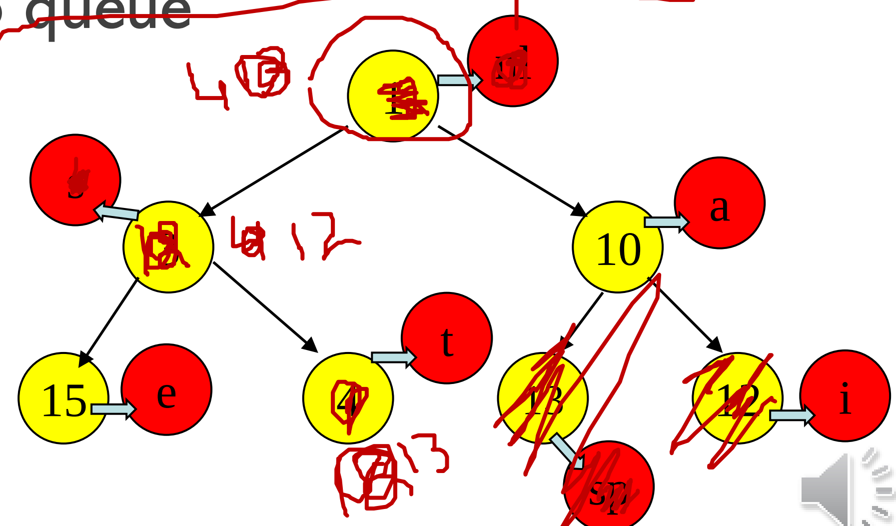
- If we maintain the trees in a priority queue, ordered by weight, then the running time will be ?
  - there will be one buildHeap,
  - 2N-1 deleteMins, and
  - N- 2 inserts
  - $O(N \log N)$,

- On a priority queue that never has more than $N$ elements. A simple implementation of the priority queue, **using a list,** would give an $O(N^2)$ algorithm.

# Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue a node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue



# Huffman code

```
H = new minHeap()
for each wi
      T = new Tree(wi)
      H.Insert(T)
while H.Size() > 1
      T1 = H.DeleteMin()
      T2 = H.DeleteMin()
      T3 = Merge(T1, T2)
      H.Insert(T3)
```

# Compression Basic Algorithm

1. Scan text to be compressed and tally occurrence of all characters.
2. Prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

# Huffman Code Properties

- **Prefix code**
  - No code is a prefix of another code
  - Example
    - Huffman("I") $\Rightarrow$ 00
    - Huffman("X") $\Rightarrow$ 001 // not legal prefix code
  - Can stop as soon as complete code found
  - No need for end-of-code marker

01111

- **Nondeterministic**
  - Multiple Huffman coding possible for same input
  - If more than two trees with same minimal weight

# Huffman Code Properties

- Greedy algorithm
  - Chooses best local solution at each step
  - Combines 2 trees with lowest frequency
- **Still yields overall best solution**
  - Optimal prefix code
  - Based on statistical frequency
- **Better compression possible (depends on data)**
  - Using other approaches (e.g., pattern dictionary)

# Issue

- **There are two details that must be considered.**
  - First, the encoding information must be transmitted at the start of the compressed file, since otherwise it will be impossible to decode.
  - **For small files**,
    - the cost of transmitting this table will override any possible savings in compression, and the result will probably be file expansion.
    - Of course, this can be detected and the original left intact.
  - **For large files**,
    - the size of the table is not significant.

# Issue

- The **second problem** is that, as described, this is a two-pass algorithm.
  - The first pass collects the frequency data, and
  - the second pass does the encoding.

- This is obviously not a desirable property for a program dealing with large files.

# Practice Question

# In-class exercises

1. a. Construct a Huffman code for the following data:

| character | A | B | C | D | _ |
|---|---|---|---|---|---|
| probability | 0.4 | 0.1 | 0.2 | 0.15 | 0.15 |

b. Encode the text ABACABAD using the code of question a.

c. Decode the text whose encoding is 100010111001010 in the code of question a.

# In-class exercises

What is the maximal length of a codeword possible in a Huffman encoding of an alphabet of n characters?

# Summary

- Huffman coding is a technique used to compress files for transmission
- Uses statistical coding
  - more frequently used symbols have shorter code words
- Works well for text and fax transmissions
- An application that uses several data structures

THE END