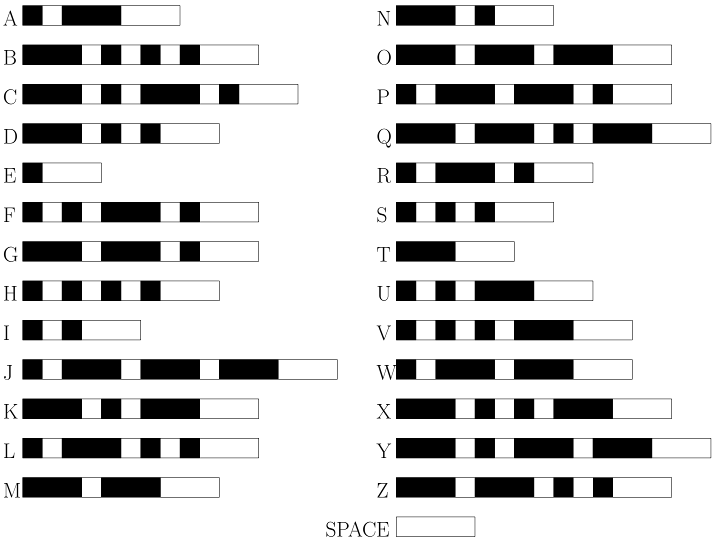
Lecture 2: Morse Code to Huffman Coding

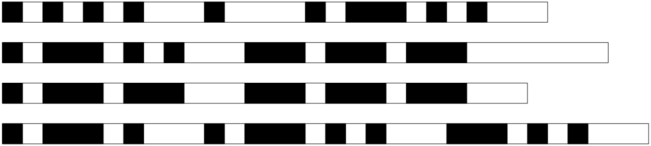
Lecturer: Travis Gagie

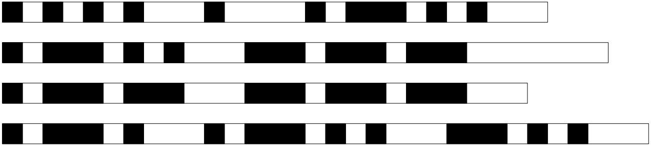
January 15th, 2015

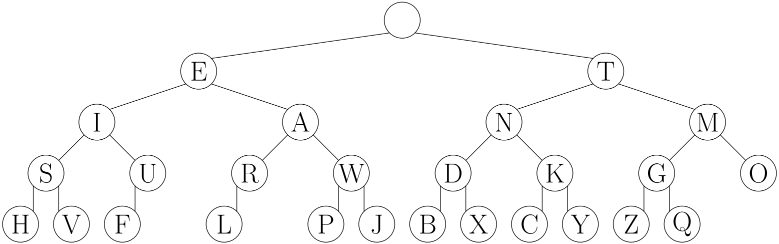
Morse Code

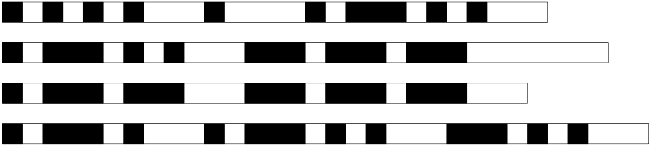
1. designed in the USA in the 1840s
2. redesigned in Germany a few years later
3. in common use for well over 100 years
4. uses dots, dashes and 3 lengths of pauses
5. assigns short codes for common characters, long codes for uncommon ones

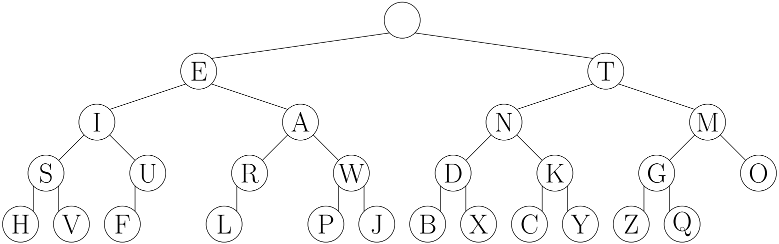

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
SPACE





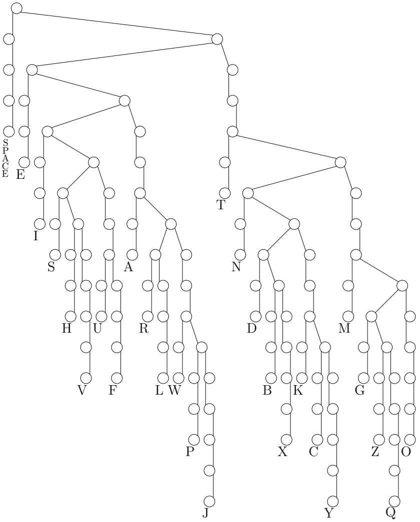

E T
I A N M
S U R W D K G O
H V F L P J B X C Y Z Q





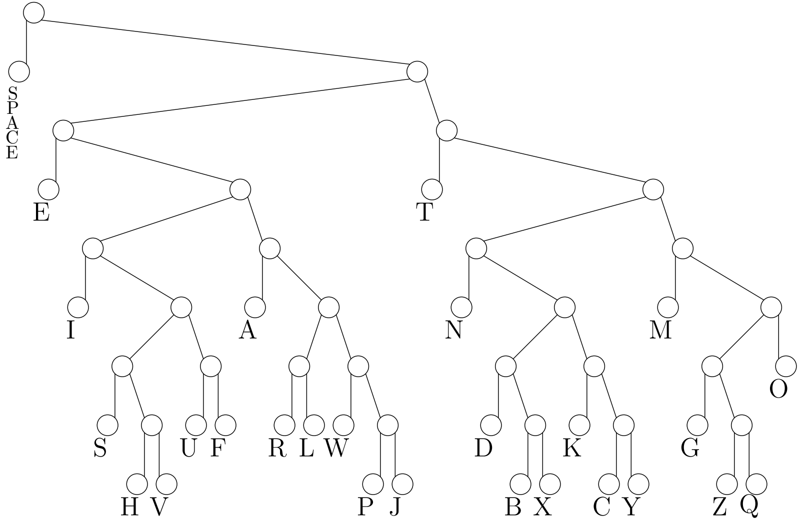
HELLO WORLD

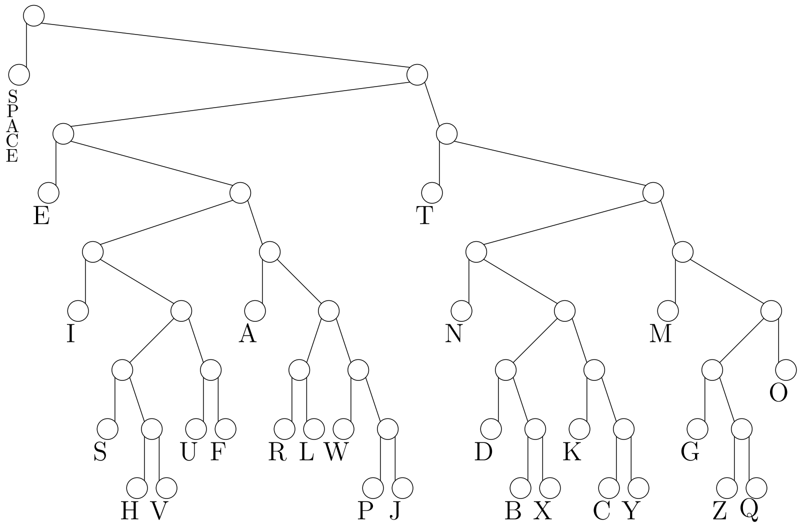
A	10111000	N	11101000
B	111010101000	O	11101110111000
C	11101011101000	P	10111011101000
D	1110101000	Q	1110111010111000
E	1000	R	1011101000
F	101011101000	S	10101000
G	111011101000	T	111000
H	1010101000	U	1010111000
I	101000	V	101010111000
J	1011101110111000	W	101110111000
K	111010111000	X	11101010111000
L	101110101000	Y	1110101110111000
M	1110111000	Z	11101110101000
		SPACE	0000


SPACE
E
T
I
S
A
N
H
U
R
D
M
V
F
L
W
B
K
G
P
X
C
Z
O
J
Y
Q

"Improved" Morse

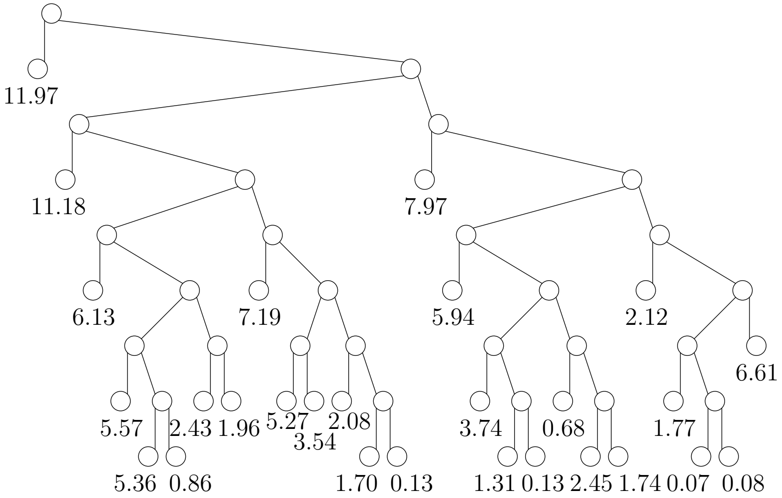
A	10110	N	11100
B	11101010	O	111111
C	11101110	P	10111110
D	1110100	Q	11111011
E	100	R	1011100
F	1010111	S	1010100
G	1111100	T	110
H	10101010	U	1010110
I	10100	V	10101011
J	10111111	W	1011110
K	1110110	X	11101011
L	1011101	Y	11101111
M	11110	Z	11111010
		SPACE	0


SPACE
E
T
I
A
N
M
S
U
F
R
L
W
D
K
G
O
H
V
P
J
B
X
C
Y
Z
Q



10101010100101110110111011111110101110111111101110010111011110100

A	7.19	N	5.94
B	1.31	O	6.61
C	2.45	P	1.70
D	3.74	Q	0.08
E	11.18	R	5.27
F	1.96	S	5.57
G	1.77	T	7.97
H	5.36	U	2.43
I	6.13	V	0.86
J	0.13	W	2.08
K	0.68	X	0.13
L	3.54	Y	1.74
M	2.12	Z	0.07
		SPACE	11.97


11.97
11.18
7.97
6.13
7.19
5.94
2.12
6.61
5.57
2.43
1.96
5.27
3.54
2.08
3.74
0.68
1.77
5.36
0.86
1.70
0.13
1.31
0.13
2.45
1.74
0.07
0.08

M has weight 2.12 and it's codeword 11110 has length 5.

O has weight 6.61 and it's codeword 111111 has length 6.

Therefore, swapping the leaves for M and O reduces the weighted path length by

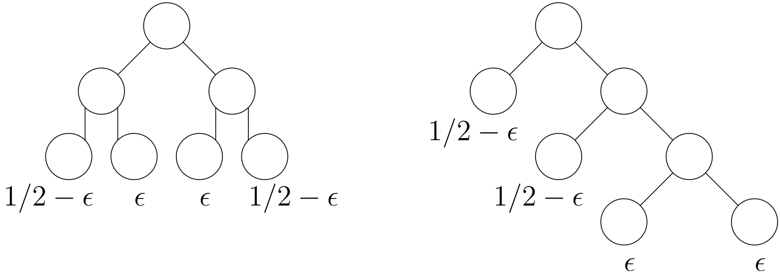
$$6.61 - 2.12 = 4.49\,.$$

Some conditions for optimality:

- The tree should not contain any node with exactly 1 child.
- No leaf with lower weight should appear strictly above another leaf with higher weight.

Some conditions for optimality:

- The tree should not contain any node with exactly 1 child.
- No leaf with lower weight should appear strictly above another leaf with higher weight.
- No *node* with lower weight should appear strictly above another *node* with higher weight.



$1/2-\epsilon$
ϵ
ϵ
$1/2-\epsilon$
$1/2-\epsilon$
$1/2-\epsilon$
ϵ
ϵ

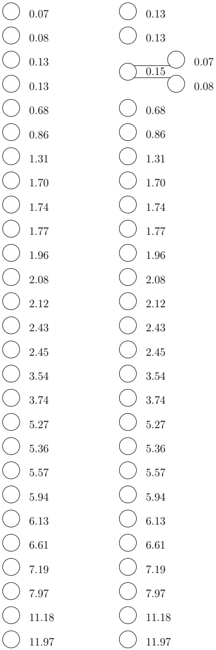
Huffman Coding

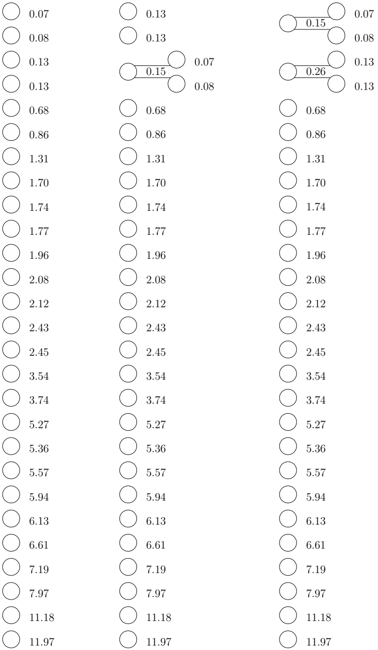
1. published in 1952
2. still used for many purposes today
   (often in combination with other techniques)
3. assigns short codes for common characters,
   long codes for uncommon ones — *optimally*

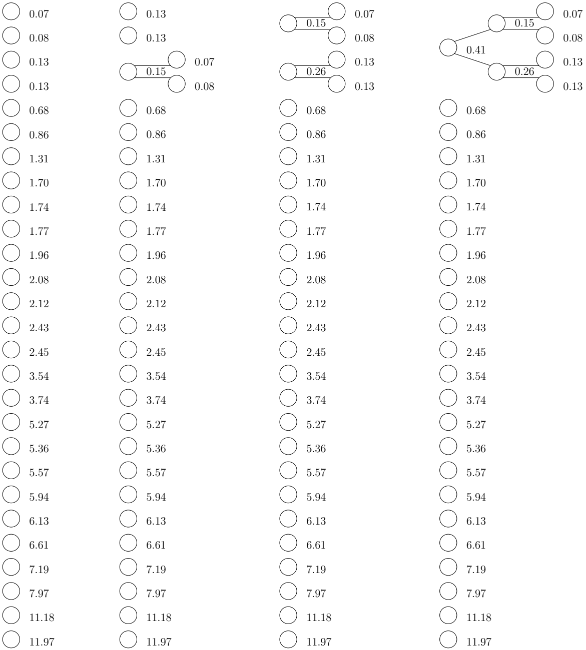
Huffman's Algorithm:

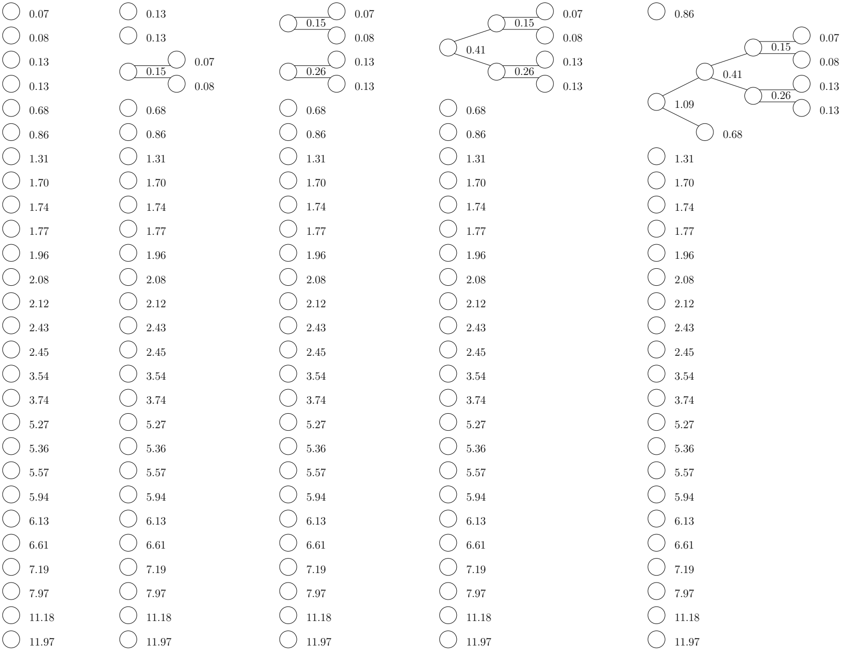
1. For each weight create a tree consisting of a single node, assigned that weight.
2. If there is only one tree left, stop. Otherwise, make the roots of the two lightest tree the children of a new node, whose weight is the sum of theirs.
3. Go back to Step 2.

- ◯ 0.07
- ◯ 0.08
- ◯ 0.13
- ◯ 0.13
- ◯ 0.68
- ◯ 0.86
- ◯ 1.31
- ◯ 1.70
- ◯ 1.74
- ◯ 1.77
- ◯ 1.96
- ◯ 2.08
- ◯ 2.12
- ◯ 2.43
- ◯ 2.45
- ◯ 3.54
- ◯ 3.74
- ◯ 5.27
- ◯ 5.36
- ◯ 5.57
- ◯ 5.94
- ◯ 6.13
- ◯ 6.61
- ◯ 7.19
- ◯ 7.97
- ◯ 11.18
- ◯ 11.97

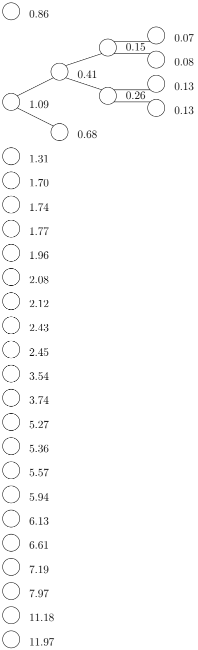

0.07
0.08
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.13
0.13
0.15
0.07
0.08
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97

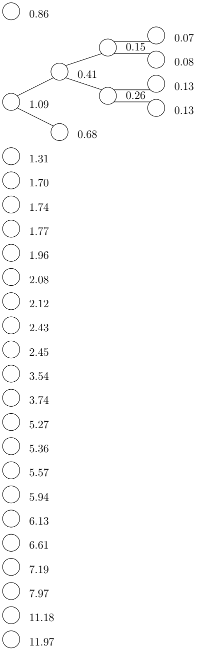

0.07
0.08
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.13
0.13
0.15
0.07
0.08
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.15
0.07
0.08
0.26
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97

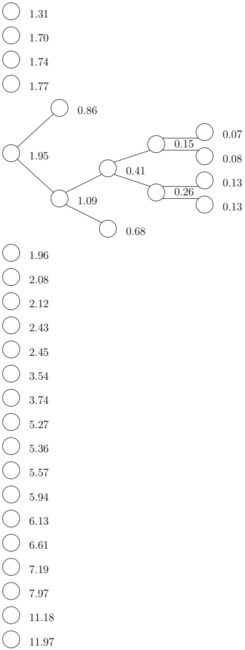

0.07
0.08
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.13
0.13
0.15
0.07
0.08
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.15
0.07
0.08
0.26
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97
0.41
0.15
0.07
0.08
0.26
0.13
0.13
0.68
0.86
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97

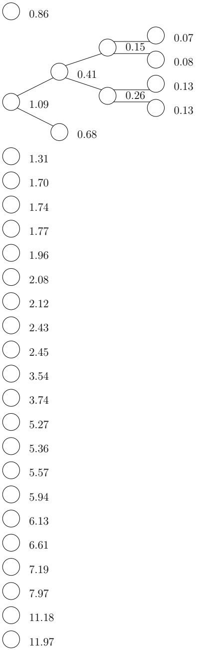


0.07 0.08 0.13 0.13 0.68 0.86 1.31 1.70 1.74 1.77 1.96 2.08 2.12 2.43 2.45 3.54 3.74 5.27 5.36 5.57 5.94 6.13 6.61 7.19 7.97 11.18 11.97
0.13 0.13 0.15 0.07 0.08 0.68 0.86 1.31 1.70 1.74 1.77 1.96 2.08 2.12 2.43 2.45 3.54 3.74 5.27 5.36 5.57 5.94 6.13 6.61 7.19 7.97 11.18 11.97
0.15 0.07 0.08 0.26 0.13 0.13 0.68 0.86 1.31 1.70 1.74 1.77 1.96 2.08 2.12 2.43 2.45 3.54 3.74 5.27 5.36 5.57 5.94 6.13 6.61 7.19 7.97 11.18 11.97
0.15 0.07 0.08 0.41 0.26 0.13 0.13 0.68 0.86 1.31 1.70 1.74 1.77 1.96 2.08 2.12 2.43 2.45 3.54 3.74 5.27 5.36 5.57 5.94 6.13 6.61 7.19 7.97 11.18 11.97
0.86 0.15 0.07 0.08 0.41 0.26 0.13 0.13 1.09 0.68 1.31 1.70 1.74 1.77 1.96 2.08 2.12 2.43 2.45 3.54 3.74 5.27 5.36 5.57 5.94 6.13 6.61 7.19 7.97 11.18 11.97

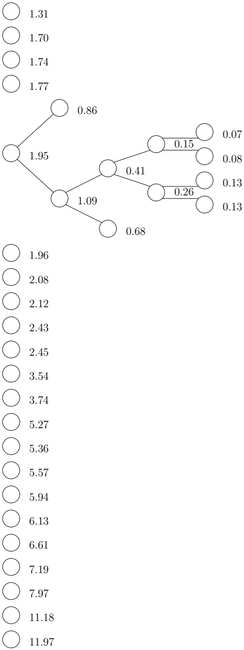

0.86
0.07
0.15
0.08
0.41
0.13
0.26
1.09
0.13
0.68
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97


0.86
0.15
0.07
0.08
0.41
0.26
0.13
0.13
1.09
0.68
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97

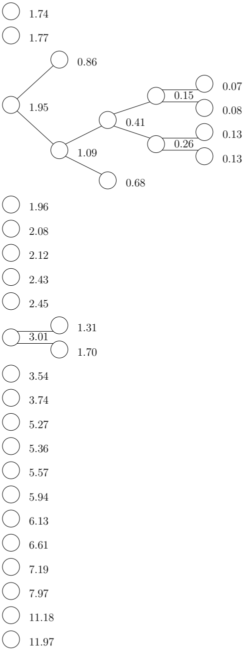

1.31
1.70
1.74
1.77
0.86
1.95
0.15
0.07
0.08
0.41
0.26
0.13
0.13
1.09
0.68
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97



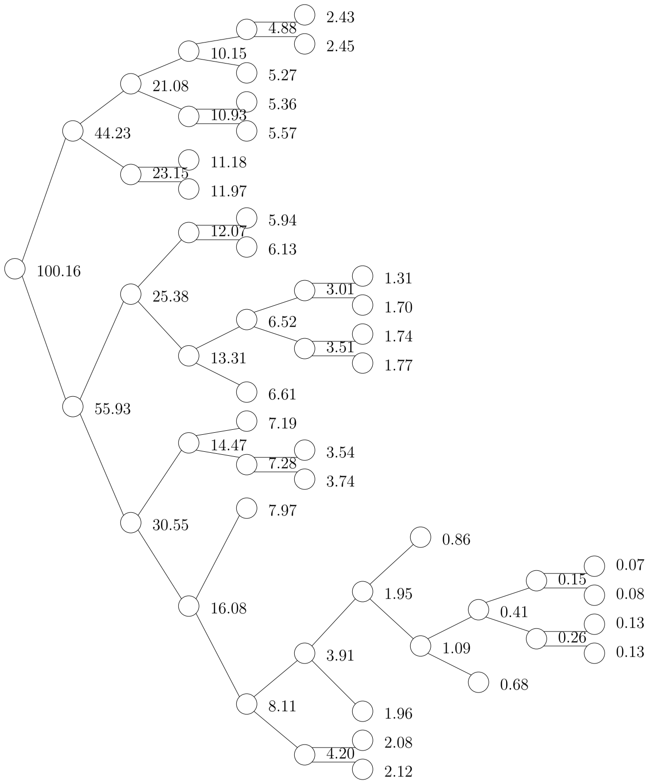
0.86
0.07
0.15
0.08
0.41
0.13
0.26
1.09
0.13
0.68
1.31
1.70
1.74
1.77
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97



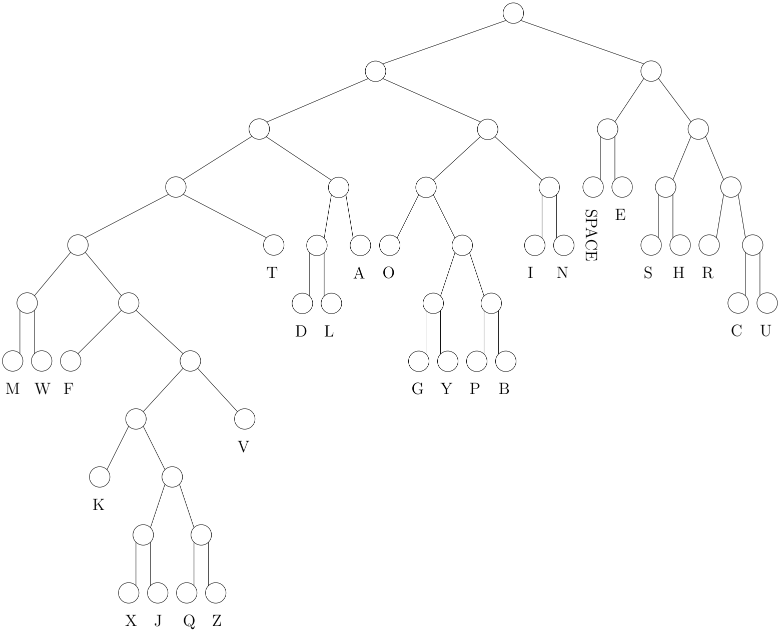
1.31
1.70
1.74
1.77
0.86
1.95
0.07
0.15
0.08
0.41
0.13
0.26
1.09
0.13
0.68
1.96
2.08
2.12
2.43
2.45
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97



1.74
1.77
0.86
0.07
1.95
0.15
0.08
0.41
0.13
1.09
0.26
0.13
0.68
1.96
2.08
2.12
2.43
2.45
1.31
3.01
1.70
3.54
3.74
5.27
5.36
5.57
5.94
6.13
6.61
7.19
7.97
11.18
11.97



2.43
4.88
2.45
10.15
5.27
21.08
5.36
10.93
5.57
44.23
11.18
23.15
11.97
5.94
12.07
6.13
100.16
1.31
3.01
1.70
25.38
6.52
1.74
3.51
1.77
13.31
6.61
55.93
7.19
14.47
3.54
7.28
3.74
7.97
30.55
0.86
0.07
0.15
0.08
1.95
0.41
0.13
0.26
0.13
16.08
1.09
3.91
0.68
8.11
1.96
2.08
4.20
2.12


T
A
O
I
N
SPACE
E
S
H
R
D
L
C
U
M
W
F
G
Y
P
B
V
K
X
J
Q
Z

Proof of correctness:

- Let $W = w_1, \ldots, w_\sigma$.
- Without loss of generality, assume w_1 and w_2 are the smallest weights in W.
- Let $W' = w_1 + w_2, w_3, \ldots, w_\sigma$.

Proof of correctness:

- Let $W = w_1, \ldots, w_\sigma$.
- Without loss of generality, assume w_1 and w_2 are the smallest weights in W.
- Let $W' = w_1 + w_2, w_3, \ldots, w_\sigma$.
- There exists an optimal tree for W in which w_1 and w_2 are assigned to leaves that are siblings. Why?

Proof of correctness:

- Let $W = w_1, \ldots, w_\sigma$.
- Without loss of generality, assume w_1 and w_2 are the smallest weights in W.
- Let $W' = w_1 + w_2, w_3, \ldots, w_\sigma$.
- There exists an optimal tree for W in which w_1 and w_2 are assigned to leaves that are siblings. Why?
- If we remove the leaves with weights w_1 and w_2 from any such optimal tree for W and assign weight $w_1 + w_2$ to their parent, then:
  - the weighted path length decreases by $w_1 + w_2$;
  - we obtain an optimal tree for W'. Why?

Proof of correctness:

- Let $W = w_1, \ldots, w_\sigma$.
- Without loss of generality, assume w_1 and w_2 are the smallest weights in W.
- Let $W' = w_1 + w_2, w_3, \ldots, w_\sigma$.
- There exists an optimal tree for W in which w_1 and w_2 are assigned to leaves that are siblings. Why?
- If we remove the leaves with weights w_1 and w_2 from any such optimal tree for W and assign weight $w_1 + w_2$ to their parent, then:
  - the weighted path length decreases by $w_1 + w_2$;
  - we obtain an optimal tree for W'. Why?
- If we attach leaves with weights w_1 and w_2 as children to the leaf with weight $w_1 + w_2$ in any optimal tree for W', then:
  - the weighted path length increases by $w_1 + w_2$;
  - we obtain an optimal tree for W. Why?

Theorem (Kraft Inequality, 1949)

If there exists a binary tree T whose leaves (in any order) have depths $\ell_1, \ldots, \ell_\sigma$, then $\sum_i 2^{-\ell_i} \leq 1$ with equality if and only if the tree is strictly binary. Conversely, if $\ell_1, \ldots, \ell_\sigma$ is a sorted sequence of integers with $\sum_i 2^{-\ell_i} \leq 1$, then there exists a binary tree T whose leaves (from left to right) have depths $\ell_1, \ldots, \ell_\sigma$.

First suppose we have a binary tree T whose leaves (in any order) have depths $\ell_1, \ldots, \ell_\sigma$. Let $\ell_{\mathsf{max}} = \mathsf{max}_i\{\ell_i\}$. For $1 \leq i \leq \sigma$, we attach a complete binary tree of height $\ell_{\mathsf{max}} - \ell_i$ to the leaf with depth ℓ_i. This does not change the height ℓ_{max} of the tree, so the number of leaves $\sum_i 2^{\ell_{\mathsf{max}} - \ell_i}$ is at most $2^{\ell_{\mathsf{max}}}$, with equality if an only if the original tree was strictly binary. Dividing both sides of

$$\sum_i 2^{\ell_{\mathsf{max}} - \ell_i} \leq 2^{\ell_{\mathsf{max}}}$$

by $2^{\ell_{\mathsf{max}}}$, we have $\sum_i 2^{-\ell_i} \leq 1$ with equality if an only if the original tree was strictly binary.

Now suppose we have a sorted sequence $\ell_1, \ldots, \ell_\sigma$ of integers such that $\sum_i 2^{-\ell_i} \leq 1$. Again, let $\ell_{\max} = \max_i\{\ell_i\}$. We build a complete binary tree T' of height $\ell_{\max}$ and perform an in-order traversal on it. For $1 \leq i \leq \sigma$, we find the next unvisited node with depth ℓ_i and remove its proper descendants. After this, either there is at least one more unvisited node with depth ℓ_i, or there are no more unvisited leaves. In the first case, we can continue if necessary and

$$\sum_{j \leq i} 2^{\ell_{\max} - \ell_j} < 2^{\ell_{\max}}$$

so $\sum_{j \leq i} 2^{-\ell_j} < 1$. In the second case,

$$\sum_{j \leq i} 2^{\ell_{\max} - \ell_j} = 2^{\ell_{\max}}$$

so $\sum_{j \leq i} 2^{-\ell_j} = 1$ and $i = \sigma$.

Definition

The *0th-order empirical entropy* of a string $s[1..n]$ over an alphabet of size σ is defined as

$$H_0(s) = \sum_{a \in s} \frac{\mathrm{occ}(a, s)}{n} \log \frac{n}{\mathrm{occ}(a, s)}\,,$$

where $a \in s$ means character a occurs in s and $\mathrm{occ}(a, s)$ is the number of its occurrences.

Theorem (Katona and Nemetz, 1978)

If a character has probability at least $1/\phi^{\ell}$, *where* $\phi \approx 1.618$ *is the golden ratio, then no Huffman code can assign it a codeword of length more than* ℓ.

Theorem (Katona and Nemetz, 1978)

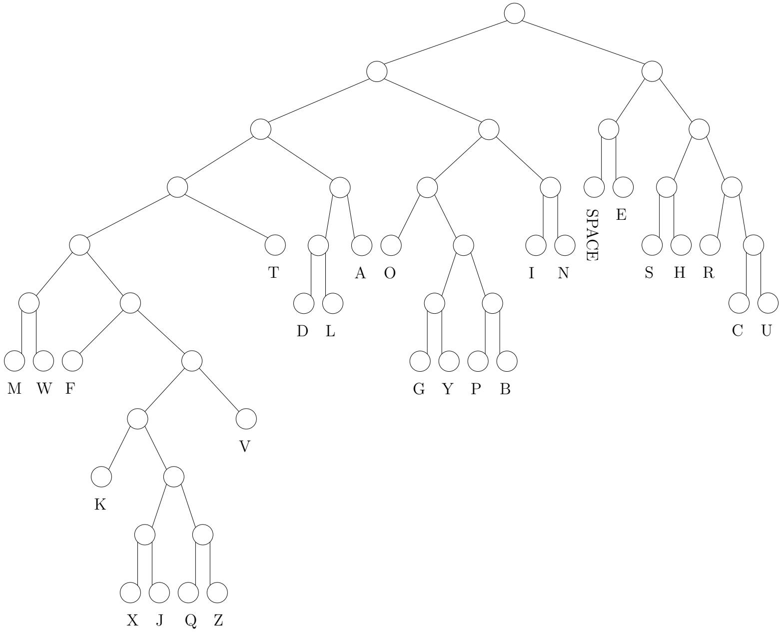
If a character has probability at least $1/\phi^{\ell}$, where $\phi \approx 1.618$ is the golden ratio, then no Huffman code can assign it a codeword of length more than ℓ.

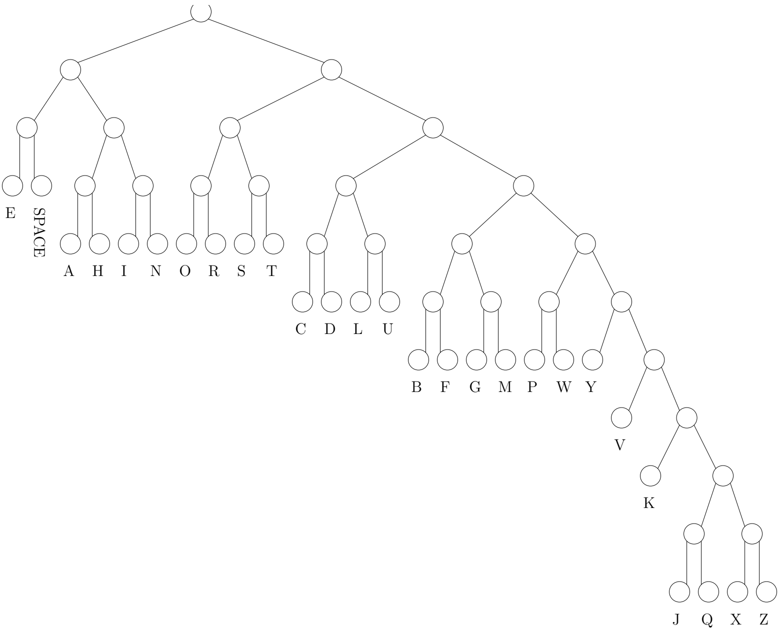
Therefore, if a character occurs in s, then no Huffman code can assign it a codeword of length more than $\lceil \log_\phi n \rceil \approx 1.44 \log n$. So, in the word RAM model, any codeword fits in $\mathcal{O}(1)$ machine words.

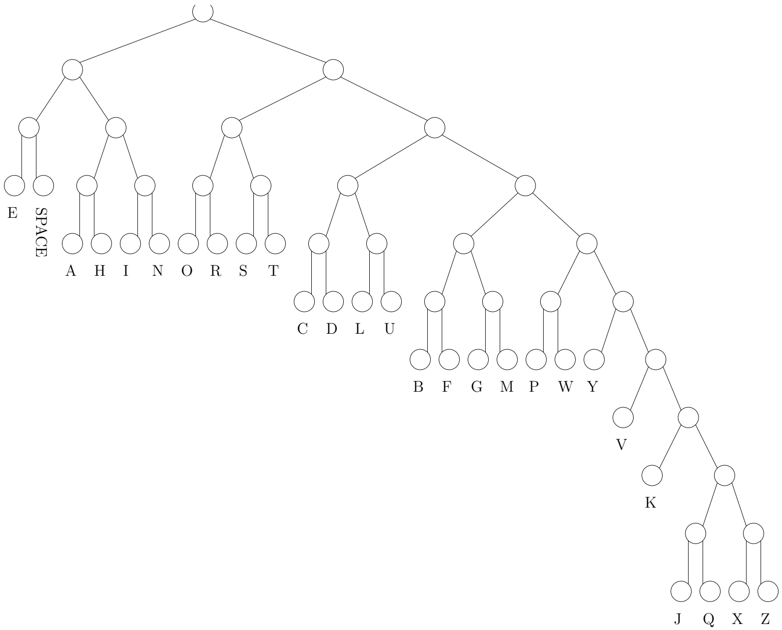
We can easily encode each character in $\mathcal{O}(1)$ time by table look-up.

How can we decode each character quickly?

With the Kraft Inequality, we can rebuild the code-tree such that the leaves, from right to left, are in non-decreasing order by depth. This takes linear time...


T
A
O
I
N
SPACE
E
S
H
R
D
L
C
U
M
W
F
G
Y
P
B
V
K
X
J
Q
Z


E
SPACE
A H I N O R S T
C D L U
B F G M P W Y
V
K
J Q X Z



[E, SPACE, A, H, I, N, O, R, S, T, C, D, L, U, B, F, G, M, P, W, Y, V, K, J, Q, X, Z]

1) 000
2) 001
3) 0100
4) 0101
5) 0110
6) 0111
7) 1000
8) 1001
9) 1010
10) 1011
11) 11000
12) 11001
13) 11010
14) 11011
15) 111000
16) 111001
17) 111010
18) 111011
19) 111100
20) 111101
21) 111110
22) 1111110
23) 11111110
24) 1111111100
25) 1111111101
26) 1111111110
27) 1111111111

1)	000	14)	11011
2)	001	15)	111000
3)	0100	16)	111001
4)	0101	17)	111010
5)	0110	18)	111011
6)	0111	19)	111100
7)	1000	20)	111101
8)	1001	21)	111110
9)	1010	22)	1111110
10)	1011	23)	11111110
11)	11000	24)	1111111100
12)	11001	25)	1111111101
13)	11010	26)	1111111110
		27)	1111111111

$$(111101)_2 - (111000)_2 = 20 - 15$$

We store in a predecessor data structure the lexicographically first codeword of each length, together with its rank as auxiliary information, which takes $\mathcal{O}(\sigma \log n)$ bits:

$$\left\{ \begin{array}{l} \langle 000, 1 \rangle, \\ \langle 0100, 3 \rangle, \\ \langle 11000, 11 \rangle, \\ \langle 111000, 15 \rangle, \\ \langle 1111110, 22 \rangle, \\ \langle 11111110, 23 \rangle, \\ \langle 1111111100, 24 \rangle \end{array} \right\} .$$

Given an encoding that starts $1111011001011100\ldots$,

Given an encoding that starts 1111011001011100 . . .,

1. we use our data structure to find its lexicographic predecessor 111000 in the set

$$\left\{ \langle 000, 1 \rangle, \ldots, \langle 1111111100, 24 \rangle \right\} \ ;$$

Given an encoding that starts $1111011001011100\ldots$,

1. we use our data structure to find its lexicographic predecessor 111000 in the set

$$\{\langle 000, 1\rangle, \ldots, \langle 1111111100, 24\rangle\}\ ;$$

2. we truncate all but the first $|111000| = 6$ bits 111101 of the encoding;

Given an encoding that starts $1111011001011100\ldots$,

1. we use our data structure to find its lexicographic predecessor 111000 in the set

$$\left\{\langle 000, 1\rangle, \ldots, \langle 1111111100, 24\rangle\right\} ;$$

2. we truncate all but the first $|111000| = 6$ bits 111101 of the encoding;
3. we add $(111101)_2 - (111000)_2 = 5$ to 111000's rank 15 to obtain 111101's rank 20;

Given an encoding that starts $1111011001011100\ldots$,

1. we use our data structure to find its lexicographic predecessor 111000 in the set

$$\{\langle 000, 1\rangle, \ldots, \langle 1111111100, 24\rangle\}\ ;$$

2. we truncate all but the first $|111000| = 6$ bits 111101 of the encoding;
3. we add $(111101)_2 - (111000)_2 = 5$ to 111000's rank 15 to obtain 111101's rank 20;
4. finally, we return the 20th character in the array [E, SPACE, ..., Q, X, Z], which is W.

If we use **binary search** to find the predecessor then, since the maximum codeword length is $\mathcal{O}(\log n)$ bits, we use a total of $\mathcal{O}(\log \log n)$ time to decode each character.

If we use **binary search** to find the predecessor then, since the maximum codeword length is $\mathcal{O}(\log n)$ bits, we use a total of $\mathcal{O}(\log \log n)$ time to decode each character.

If we use a (completely impractical) data structure called a **fusion tree**, then we use $\mathcal{O}(1)$ time to decode each character.

If we use **binary search** to find the predecessor then, since the maximum codeword length is $\mathcal{O}(\log n)$ bits, we use a total of $\mathcal{O}(\log \log n)$ time to decode each character.

If we use a (completely impractical) data structure called a **fusion tree**, then we use $\mathcal{O}(1)$ time to decode each character.

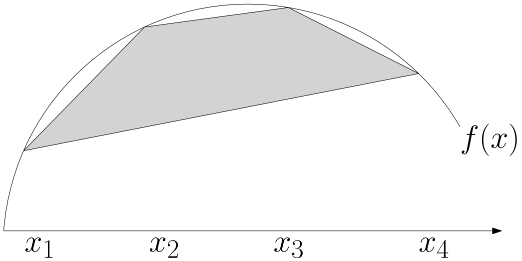
If we use **doubling search** then, for a character a with codeword $c(a)$, we use $\mathcal{O}(\log |c(a)| + 1)$ time.

How can we bound $\mathcal{O}\left(\sum_{i=1}^{n}\left(\log\left|c(s[i])\right|+1\right)\right)$?

Theorem (Jensen's Inequality)

If $f(x)$ is a concave function, $P = p_1, \ldots, p_\sigma$ is a probability distribution and $x_1, \ldots, x_\sigma$ are values in the domain of $f(x)$, then

$$f\left(\sum_i p_i x_i\right) \leq \sum_i p_i f(x_i).$$



$f(x)$
x_1
x_2
x_3
x_4

Since log is a concave function, if $P = p_1, \ldots, p_\sigma$ and each p_j is again the number $\mathrm{occ}(a_j, s)$ of occurrences of the jth distinct character a_j in s, then

$$\begin{aligned}
& \sum_{i=1}^{n} (\log |c(s[i])| + 1) \\
& \quad = \quad n \sum_{j=1}^{\sigma} p_j (\log |c(a_j)| + 1) \\
& \quad \leq \quad n \log \sum_{j=1}^{\sigma} |c(a_j)| + n \\
& \quad \leq \quad n \log(H_0(s) + 2)\,.
\end{aligned}$$

Theorem

Given the sorted frequencies of characters in a string s of length n, we can build a canonical Huffman code for s in $\mathcal{O}(\sigma)$ time. We can then encode s in fewer than $nH_0(s) + n$ bits (plus $\mathcal{O}(\sigma \log n)$ bits to store the code) in $\mathcal{O}(n)$ time on a word RAM. Later, we can decode s in $\mathcal{O}(n \log(H_0(s) + 2))$ time.