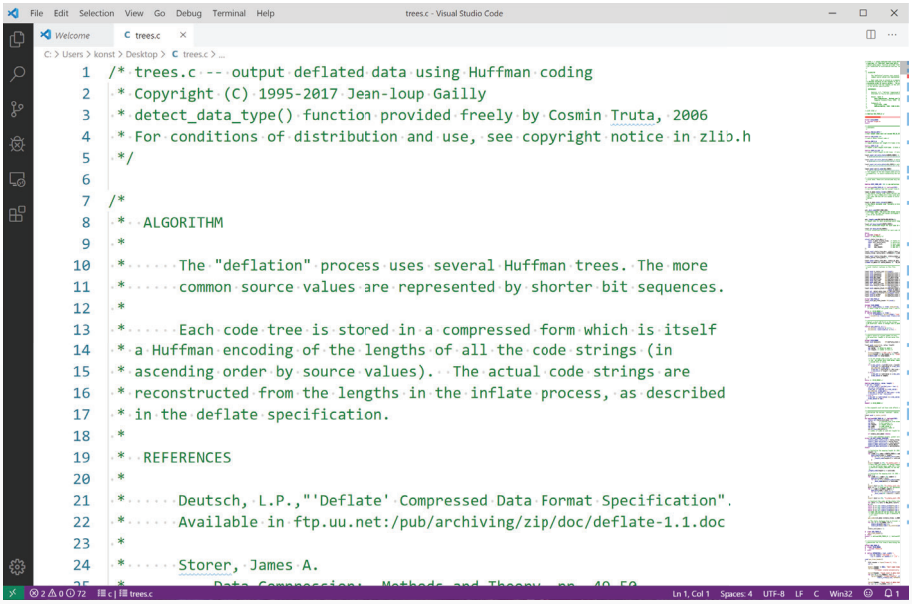
Huffman Coding

CS 336: Design and Analysis of Algorithms

What do these products have in common?

They use zlib and Huffman Coding

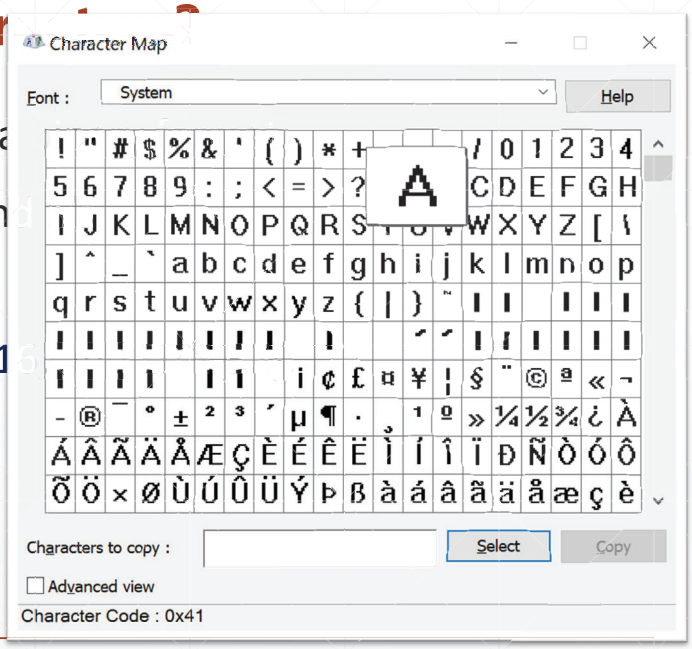


How do we encode characters?

- All data in computers is stored in a binary format.
- Character are encoded with 0's and 1's.
- Standard formats:
  - ASCII and Unicode (UTF-8, UTF-16, and UTF-32).

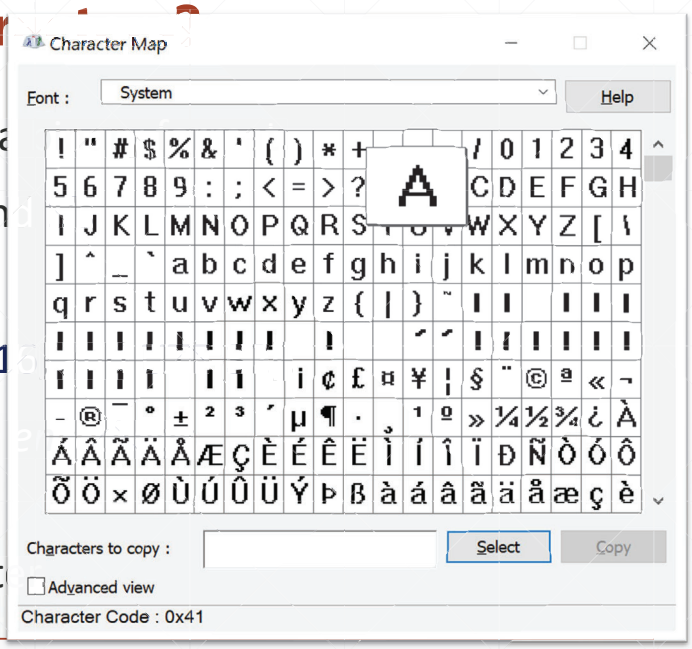
How do we encode char

- All data in computers is stored in a
- Character are encoded with 0's an
- Standard formats:
  - ASCII and Unicode (UTF-8, UTF-1



How do we encode char

- All data in computers is stored in a
- Character are encoded with 0's an
- Standard formats:
  - ASCII and Unicode (UTF-8, UTF-1
- ASCII and UTF-32 are *fixed-length*
  - ➢ASCII uses 8 bits per character;
  - ➢UTF-32 uses 32 bits per charact



How do we encode characters?

- All data in computers is stored in a binary format.
- Character are encoded with 0's and 1's.
- Standard formats:
  - ASCII and Unicode (UTF-8, UTF-16, and UTF-32).
- ASCII and UTF-32 are *fixed-length encodings.*
  - ➢ASCII uses 8 bits per character;
  - ➢UTF-32 uses 32 bits per character.

Codes

- Each character in the alphabet is replaced with a *code word*.

$$A \mapsto 00, C \mapsto 01, G \mapsto 10, T \mapsto 11$$

- Code words are concatenated:

$$ACGCACT \mapsto 00011001000111$$

Codes

- Each character in the alphabet is replaced with a *code word*.

$$A \mapsto 000, C \mapsto 01, G \mapsto 001, T \mapsto 1$$

- Code words are concatenated:

$$ACGCACT \mapsto 00001001010011$$

Fixed-Length vs Variable-Length Encodings

Fixed-Length Encoding Pros:

- Easy to decode.
- Easier to work with.

Variable-Length Encoding Pros:

- More efficient – take less space, because some character are rare.

Example: Variable-Length Encoding

Letter	Frequency	Fixed Length	Variable Length
A	0.47	000	00
B	0.47	001	01
$\alpha, \beta, \gamma, \xi, \zeta, \eta$	0.01 each	010, 011,100 101, 110, 111	1000, 1001,1010 1011, 1100, 1101

- F-L: Expected length of a message: 3 bits/character
- V-L: Expected length of a message: $\approx$ 2 bits/character

Example: Variable-Length Encoding

Letter	Frequency	Encoding A	Encoding B
A	0.47	00	0
B	0.47	01	01
$\alpha, \beta, \gamma, \xi, \zeta, \eta$	0.01 each	1000, 1001,1010, 1011, 1100, 1101	00, 1, 10, 11, 110, 111

- Is Encoding B better than Encoding A?
- How to decode "00"? Is it "AA" or "α"? Or how about "101"?

Prefix Codes

- Each character is encoded by a *code word*.
- No code word is a prefix of another one.
- Decoding:
  - Read the stream till you find a code word; remove it; and continue.

Code words
00
01
1000,1001,1010, 1011,1100,1101

```
00  1000  01  1010  00  01  01  00  1101
```

Prefix Codes

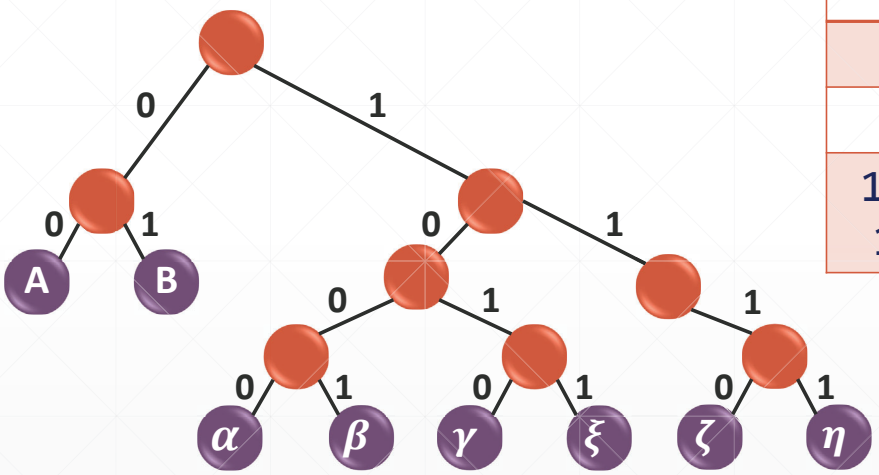
- All *fixed-length codes* are *prefix codes*.
- UTF-8 and UTF-16 are *prefix codes*.

How to find the most efficient variable-length prefix code?

Optimal Prefix Codes

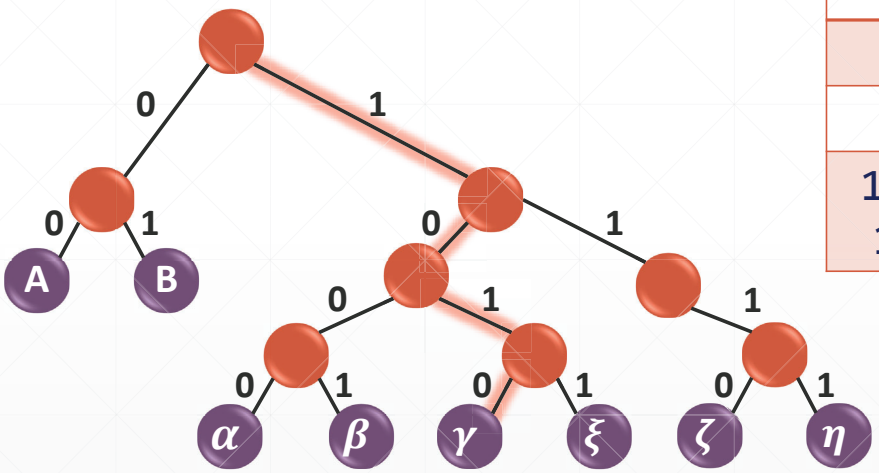
CS 336: Design and Analysis of Algorithms

Tree Representation



Code words
00
01
1000,1001,1010, 1011,1110,1111

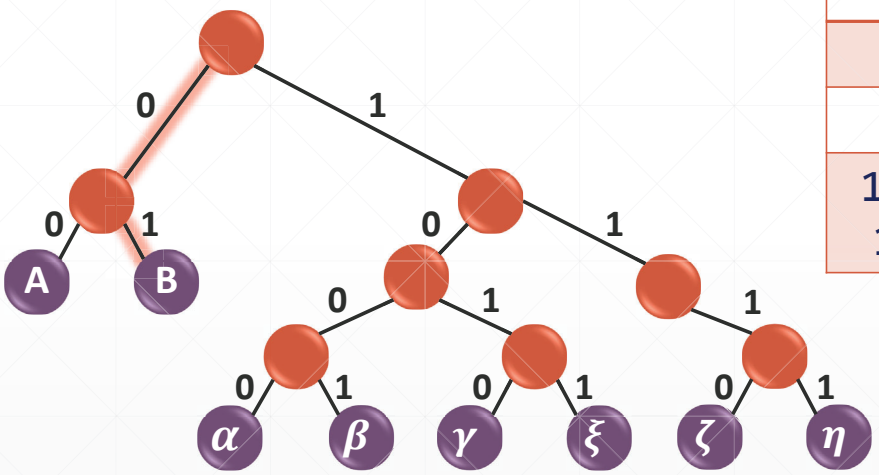
Tree Representation



Code words
00
01
1000,1001,1010, 1011,1110,1111

1010

Tree Representation



Code words
00
01
1000,1001,1010, 1011,1110,1111

1010 01 …

Expected Encoding Length

- Expected cost per character equals:

$$\text{cost} = \sum p_i \, \text{len}(w_i)$$

where

- p_i is the probability of character i;
- w_i is the code word for character i.

Expected Encoding Length

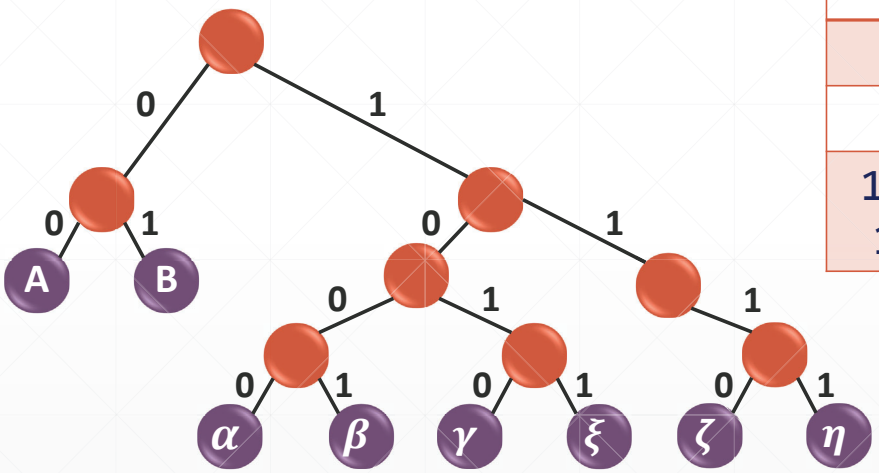
- Expected cost per character equals:

$$\text{cost} = \sum p_i \, \text{len}(w_i) = \sum p_i \ \text{depth}\ (v_i)$$

where

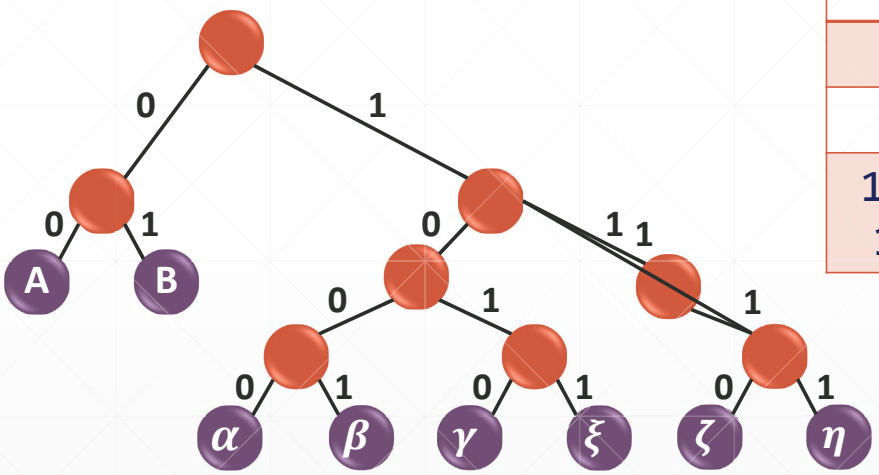
- p_i is the probability of character i;
- w_i is the code word for character i.
- v_i is the leaf assigned to character i.

Tree Representation



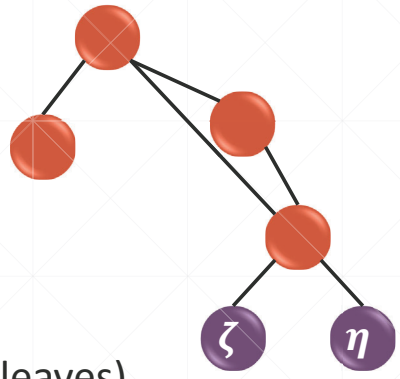
Code words
00
01
1000,1001,1010, 1011,1110,1111

Tree Representation



Code words
00
01
1000,1001,1010, 1011,1110,1111

Full Binary Tree



Claim:

- The optimal tree is a *full binary tree.*
- In a *full binary tree* all internal nodes (not leaves) have exactly two children.
- In other words, each node has 0 or 2 children.

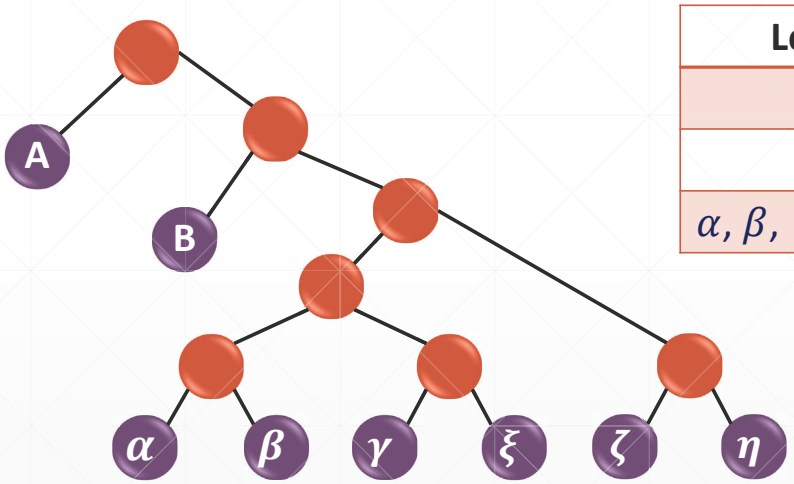
Optimal Labeling?

- The optimal tree is a *full binary tree.*
- In a *full binary tree* all internal nodes (not leaves) have exactly two children.
- In other words, each node has 0 or 2 children.

Question

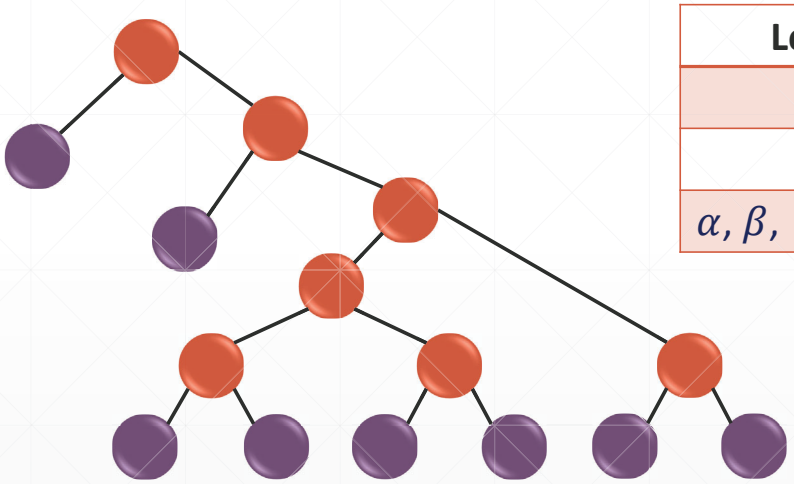
➢ Suppose I give you a tree (or, equivalently, set of code words), can you find an optimal labeling?

Tree & Frequency Table



Letter	Frequency
A	0.47
B	0.47
$\alpha, \beta, \gamma, \xi, \zeta, \eta$	0.01 each

Tree & Frequency Table



Letter	Frequency
A	0.47
B	0.47
$\alpha, \beta, \gamma, \xi, \zeta, \eta$	0.01 each

Label Assignment

- Sort all leaves by their depth in decreasing order.
- Sort all letters according to their frequencies in increasing order.
- Assign more frequent letters to leaves on top; assign less frequent letters to leaves at the bottom.

- Sort all leaves by their depth in decreasing order.
- Sort all letters according to their frequencies in increasing order.
- Assign more frequent letters to leaves on top; assign less frequent letters to leaves at the bottom.

Proof: Bubble sort! If we swap characters p_1 and p_2 at depth d_1 and d_2, then the cost decreases by

$$(p_1 d_1 + p_2 d_2) - (p_2 d_1 + p_1 d_2) = (d_1 - d_2)(p_1 - p_2)$$

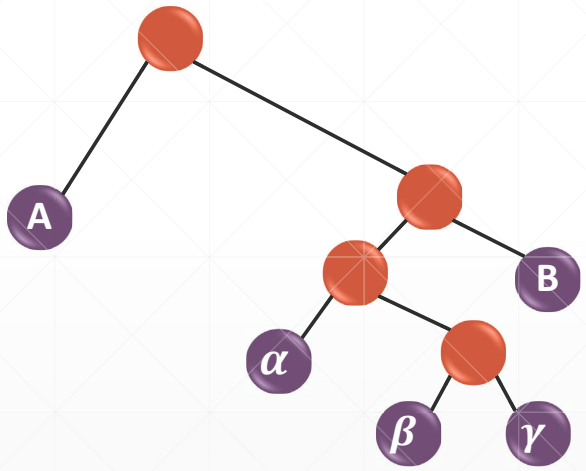
Hence, in the optimal solution if $d_1 \geq d_2$, then $p_1 \leq p_2$.

Label Assignment

- Sort all leaves by their depth in decreasing order.
- Sort all letters according to their frequencies in increasing order.
- Assign more frequent letters to leaves on top; assign less frequent letters to leaves at the bottom.

Corollary: Two least frequent letters must be siblings in one of the optimal trees.

Tree & Frequency Table

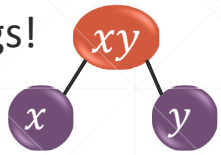


Letter	Frequency
A	0.5
B	0.3
α	0.1
β, γ	0.05

Corollary: Two least frequent letters must be siblings in the tree.

Good News

- Two least frequent characters x and y must be siblings!
- Assign them to a common node xy.
- …and recursively find best coding for the remaining characters.
- That is, remove x and y from the list of characters and add a special character xy.



Huffman Coding

CS 336: Design and Analysis of Algorithms

Huffman Coding

- Huffman proposed a greedy algorithms for constructing prefix codes in 1952.

to the knowledge of the author. It is the purpose of this paper to derive such a procedure.

Derived Coding Requirements

For an optimum code, the length of a given message code can never be less than the length of a more probable message code. If this requirement were not met, then a reduction in average message length could be obtained by interchanging the codes for the two messages in question in such a way that the shorter code becomes associated with the more probable message. Also, if there are several messages with the same probability, then it is possible that the codes for these messages may differ in length. However, the codes for these messages may be interchanged in any way without affecting the average code length for the message ensemble. Therefore, it may be assumed that the messages in the ensemble have been ordered in a fashion such that

$$P(1) \geq P(2) \geq \cdots \geq P(N-1) \geq P(N) \qquad (3)$$

and that, in addition, for an optimum code, the condition

$$L(1) \leq L(2) \leq \cdots \leq L(N-1) \leq L(N) \qquad (4)$$

holds. This requirement is assumed to be satisfied throughout the following discussion.

It might be imagined that an ensemble code could assign q more digits to the Nth message than to the $(N-1)$st message. However, the first $L(N-1)$ digits of the Nth message must not be used as the code for any other message. Thus the additional q digits would serve no useful purpose and would unnecessarily increase L_{av}. Therefore, for an optimum code it is necessary that $L(N)$ be equal to $L(N-1)$.

The kth prefix of a message code will be defined as the first k digits of that message code. Basic restriction (b) could then be restated as: No message shall be coded in such a way that its code is a prefix of any other message, or that any of its prefixes are used elsewhere as a message code.

Imagine an optimum code in which no two of the messages coded with length $L(N)$ have identical prefixes of order $L(N)-1$. Since an optimum code has been assumed, then none of these messages of length $L(N)$ can have codes or prefixes of any order which correspond to other codes. It would then be possible to drop the last digit of all of this group of messages and thereby reduce the value of L_{av}. Therefore, in an optimum code, it is necessary that at least two (and no more than D) of the codes with length $L(N)$ have identical prefixes of order $L(N)-1$.

One additional requirement can be made for an optimum code. Assume that there exists a combination of the D different types of coding digits which is less than $L(N)$ digits in length and which is not used as a message code or which is not a prefix of a message code. Then this combination of digits could be used to replace the code for the Nth message with a consequent reduction of L_{av}. Therefore, all possible sequences of $L(N)-1$ digits must be used either as message codes, or must have one of their prefixes used as message code.

The derived restrictions for an optimum code are summarized in condensed form below and considered in addition to restrictions (a) and (b) given in the first part of this paper:

(c) $L(1) \leq L(2) \leq \cdots \leq L(N-1) = L(N)$. (5)

(d) At least two and not more than D of the messages with code length $L(N)$ have codes which are alike except for their final digits.

(e) Each possible sequence of $L(N)-1$ digits must be used either as a message code or must have one of its prefixes used as a message code.

Optimum Binary Code

For ease of development of the optimum coding procedure, let us now restrict ourselves to the problem of binary coding. Later this procedure will be extended to the general case of D digits.

Restriction (c) makes it necessary that the two least probable messages have codes of equal length. Restriction (d) places the requirement that, for D equal to two, there be only two of the messages with coded length $L(N)$ which are identical except for their last digits. The final digits of these two codes will be one of the two binary digits, 0 and 1. It will be necessary to assign these two message codes to the Nth and the $(N-1)$st messages since at this point it is not known whether or not other codes of length $L(N)$ exist. Once this has been done, these two messages are equivalent to a single composite message. Its code (as yet undetermined) will be the common prefixes of order $L(N)-1$ of these two messages. Its probability will be the sum of the probabilities of the two messages from which it was created. The ensemble containing this composite message in the place of its two component messages will be called the first auxiliary message ensemble.

This newly created ensemble contains one less message than the original. Its members should be rearranged if necessary so that the messages are again ordered according to their probabilities. It may be considered exactly as the original ensemble was. The codes for each of the two least probable messages in this new ensemble are required to be identical except in their final digits; 0 and 1 are assigned as these digits, one for each of the two messages. Each new auxiliary ensemble contains one less message than the preceding ensemble. Each auxiliary ensemble represents the original ensemble with full use made of the accumulated necessary coding requirements.

The procedure is applied again and again until the number of members in the most recently formed auxiliary message ensemble is reduced to two. One of each of the binary digits is assigned to each of these two composite messages. These messages are then combined to form a single composite message with probability unity, and the coding is complete.

Huffman Coding

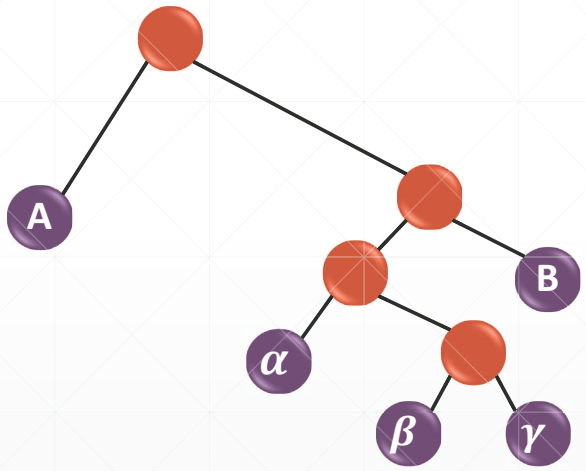
Priority queue Q for all characters ordered by frequencies p_x.

function HuffmanCode

if Q has more than one element
Remove two least frequent elements x and y from Q.
Create a new character xy with frequency $p_{xy} = p_x + p_y$.
Insert xy in the queue Q.
Call HuffmanCode recursively
Attach x and y to xy.

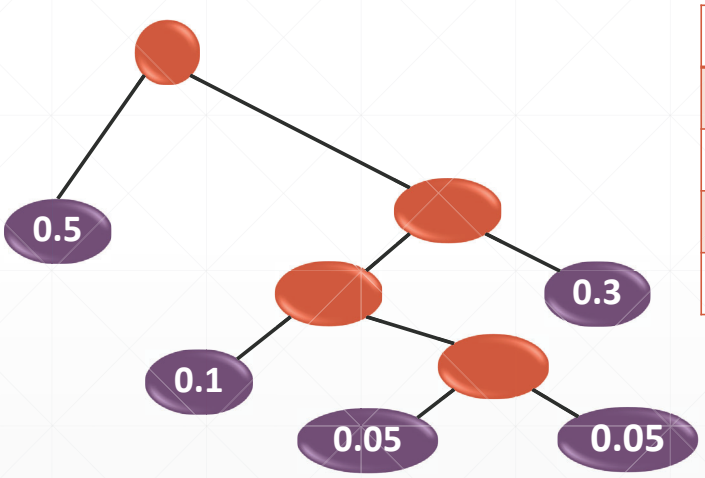
Return obtained tree.

Proof



Letter	Frequency
A	0.5
B	0.3
α	0.1
β, γ	0.05

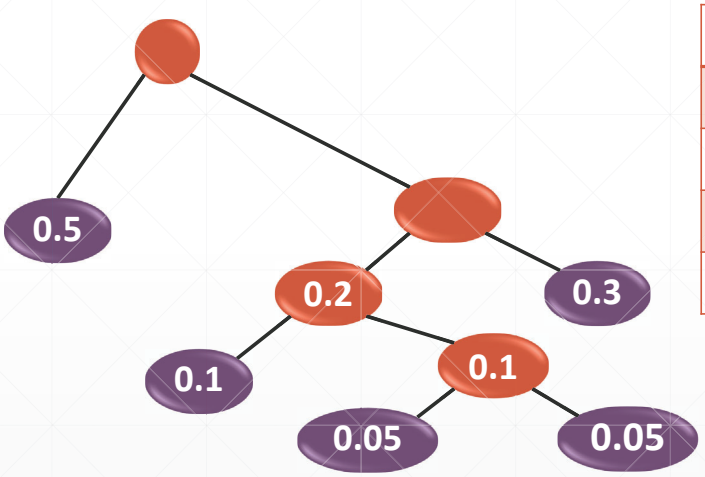
Proof



Letter	Frequency
A	0.5
B	0.3
α	0.1
β, γ	0.05

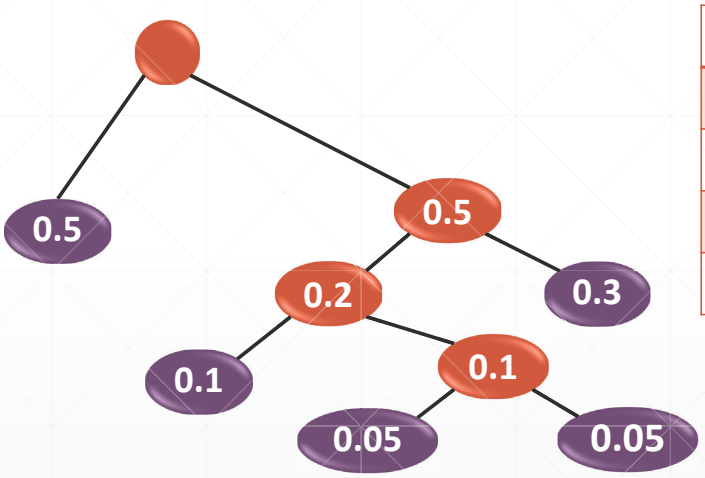
Write frequencies in the leaves.

Proof



Letter	Frequency
A	0.5
B	0.3
α	0.1
β, γ	0.05

Proof

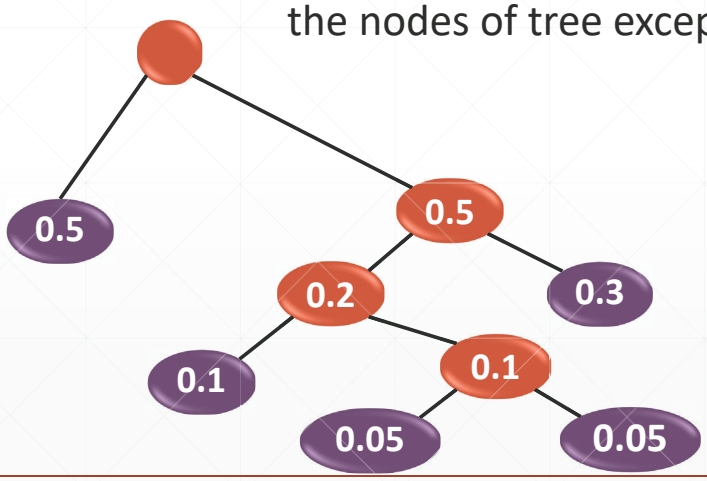


Letter	Frequency
A	0.5
B	0.3
α	0.1
β, γ	0.05

Write frequencies in leaves. Then, write frequencies of proper subtrees.

Observation

The cost of the solution equals to the sum of numbers written in the nodes of tree except for the root.



Cost of this tree equals:

$$0.5 \cdot 1 + 0.1 \cdot 3 + 0.05 \cdot 4 + 0.05 \cdot 4 + 0.3 \cdot 2 = 1.8$$

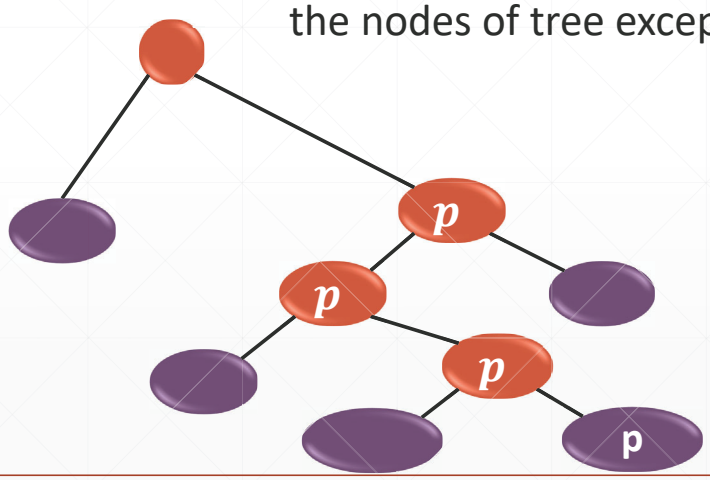
Cost of this tree equals:

$$0.5 + 0.5 + 0.2 + 0.3 + 0.1 + 0.1 + 0.05 + 0.05 = 1.8$$

Write frequencies in leaves. Then, write frequencies of proper subtrees.

Observation

The cost of the solution equals to the sum of numbers written in the nodes of tree except for the root.



Equivalent Problem

- Given numbers $p_1, \dots, p_n$.
- Find a tree with leaves $p_1, \dots, p_n$ that minimizes the sum over all proper subtrees.

$$\sum_{u \neq root} \text{cost}\,(T_u)$$

T_u — subtree rooted at u.

Main Lemma

Suppose $p_1 \geq \cdots \geq p_n$. Then,

$$OPT(p_1, \ldots, p_n) = OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

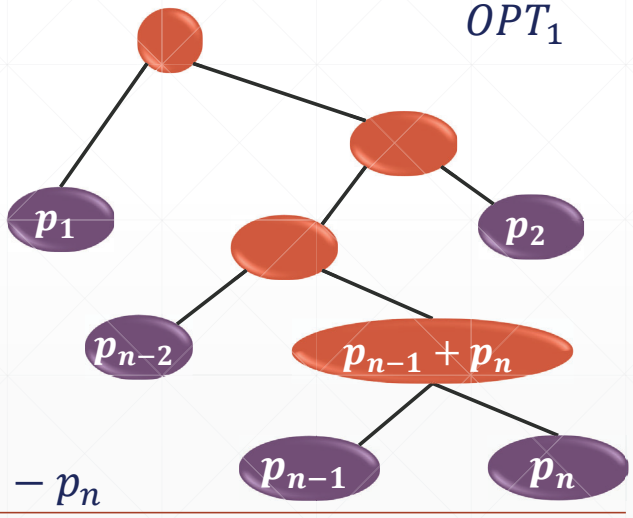
Suppose $p_1 \geq \cdots \geq p_n$. Then,

$$OPT(p_1, \ldots, p_n) \geq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

OPT_1

- In the optimal solution OPT_1, p_{n-1} and p_n are siblings. Their parent has cost $(p_{n-1} + p_n)$.
- Remove p_{n-1} and p_n from OPT_1. We get a *feasible* solution for $p_1, \ldots, p_{n-2}, p_{n-1} + p_n$.
- Hence,

$$OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) \leq \leq OPT(p_1, \ldots, p_n) - p_{n-1} - p_n$$



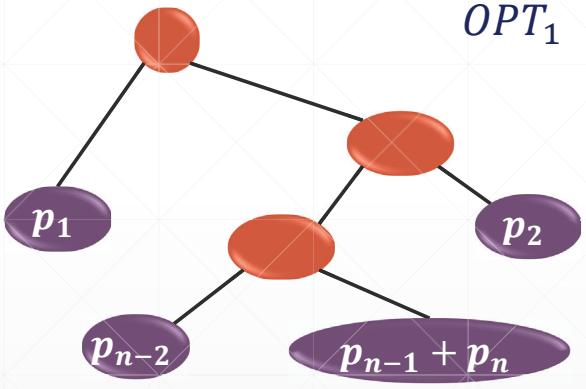
Suppose $p_1 \geq \cdots \geq p_n$. Then,

$$OPT(p_1, \ldots, p_n) \geq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

OPT_1

- In the optimal solution OPT_1, p_{n-1} and p_n are siblings. Their parent has cost $(p_{n-1} + p_n)$.
- Remove p_{n-1} and p_n from OPT_1. We get a *feasible* solution for $p_1, \ldots, p_{n-2}, p_{n-1} + p_n$.
- Hence,

$$OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) \leq$$
$$\leq OPT(p_1, \ldots, p_n) - p_{n-1} - p_n$$



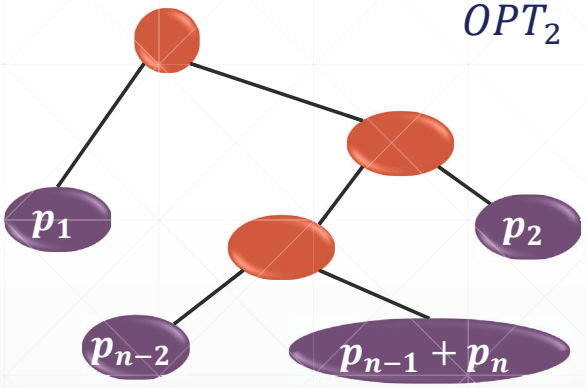
Suppose $p_1 \geq \cdots \geq p_n$. Then,

OPT_2

$$OPT(p_1, \ldots, p_n) \leq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

- Consider optimal solution OPT_2.
- Attach p_{n-1} and p_n to the node $p_{n-1} + p_n$. We obtain a *feasible* solution for $p_1, \ldots, p_{n-2}, p_{n-1}, p_n$.
- Hence,

$$OPT(p_1, \ldots, p_n) \leq$$
$$\leq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$



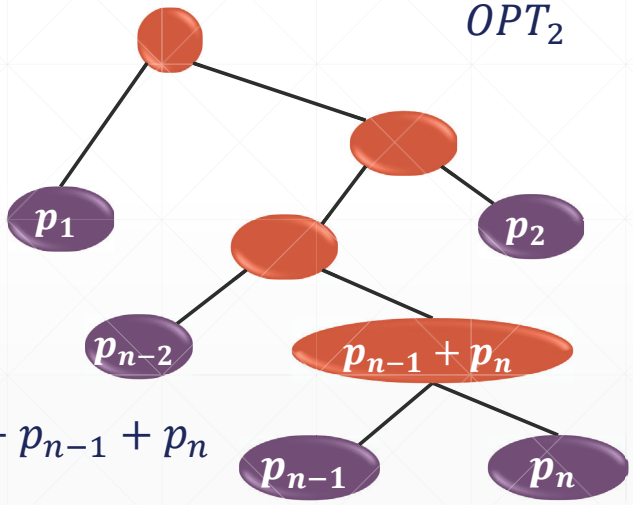
Suppose $p_1 \geq \cdots \geq p_n$. Then,

OPT_2

$$OPT(p_1, \ldots, p_n) \leq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

- Consider optimal solution OPT_2.
- Attach p_{n-1} and p_n to the node $p_{n-1} + p_n$. We obtain a *feasible* solution for $p_1, \ldots, p_{n-2}, p_{n-1}, p_n$.
- Hence,

$$OPT(p_1, \ldots, p_n) \leq \leq OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$



Main Lemma

Suppose $p_1 \geq \cdots \geq p_n$. Then,

$$OPT(p_1, \ldots, p_n) = OPT(p_1, \ldots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$$

Huffman Coding

Priority queue Q for all nodes x ordered by p_x.

function HuffmanCode

if Q has more than one node
Remove two least frequent elements x and y from Q.
Create a new node xy with frequency $p_{xy} = p_x + p_y$.
Insert xy in the queue Q.
Call HuffmanCode
Attach x and y to xy.

return obtained tree.

Proof

Theorem: For any $p_1, \dots, p_n$ we have

$$ALG(p_1, \dots, p_n) = OPT(p_1, \dots, p_n)$$

Proof by induction on n.

OPT and ALG satisfy the same recurrence relation:

- $OPT(p_1, \dots, p_n) = OPT(p_1, \dots, p_{n-2}, \boldsymbol{p_{n-1} + p_n}) + p_{n-1} + p_n$;
- $ALG(p_1, \dots, p_n) = ALG(p_1, \dots, \boldsymbol{p_{n-1} + p_n}) + p_{n-1} + p_n$;

where p_{n-1} and p_n are the smallest numbers among $p_1, \dots, p_n$.

Proof

Theorem: For any $p_1, \dots, p_n$ we have

$$ALG(p_1, \dots, p_n) = OPT(p_1, \dots, p_n)$$

Proof by induction on n.

- Base case: For $n = 2$, the tree is unique, which is optimal.
- Inductive step: Suppose we proved theorem for all $n' < n$. We know that
  - $OPT(p_1, \dots, p_n) = OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n.$
  - $OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) = ALG(p_1, \dots, p_{n-1} + p_n).$
  - $ALG(p_1, \dots, p_{n-1}, p_n) = ALG(p_1, \dots, p_{n-1} + p_n) + p_{n-1} + p_n.$

Proof

Theorem: For any $p_1, \dots, p_n$ we have

$$ALG(p_1, \dots, p_n) = OPT(p_1, \dots, p_n)$$

Proof by induction on n.

- Base case: For $n = 2$, the tree is unique, which is optimal.
- Inductive step: Suppose we proved theorem for all $n' < n$. We know that
  - $OPT(p_1, \dots, p_n) = OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n.$
  - $OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) = ALG(p_1, \dots, p_{n-1} + p_n).$
  - $ALG(p_1, \dots, p_{n-1}, p_n) = ALG(p_1, \dots, p_{n-1} + p_n) + p_{n-1} + p_n.$

Main Lemma

Inductive Hypothesis

That's how Algorithm works

Proof

Theorem: For any $p_1, \dots, p_n$ we have

$$ALG(p_1, \dots, p_n) = OPT(p_1, \dots, p_n)$$

Proof by induction on n.

- Base case: For $n = 2$, the tree is unique, which is optimal.
- Inductive step: Suppose we proved theorem for all $n' < n$. We know that
  - $OPT(p_1, \dots, p_n) = OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) + p_{n-1} + p_n$.
  - $OPT(p_1, \dots, p_{n-2}, p_{n-1} + p_n) = ALG(p_1, \dots, p_{n-1} + p_n)$.
  - $ALG(p_1, \dots, p_{n-1}, p_n) = ALG(p_1, \dots, p_{n-1} + p_n) + p_{n-1} + p_n$.

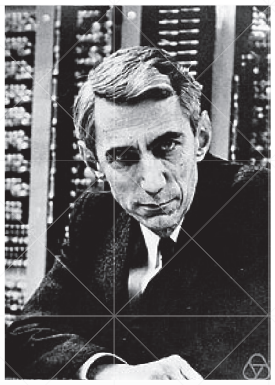
Running Time

- At every step of the algorithm, we remove two elements and insert one element.
  - The total #steps is $n - 1$.
  - The #operations per step is $O(\log n)$.
  - Total running time is $O(n \log n)$.

Shannon Entropy

CS 336: Design and Analysis of Algorithms

Detour – Shannon Entropy

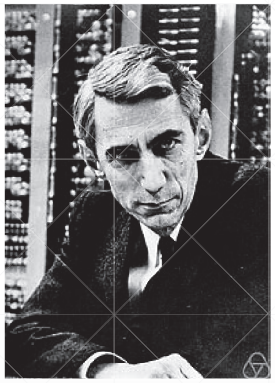


- Easy lower bound on the cost of a prefix code?
- Shannon Entropy:

$$H(p) = \sum_i p_i \log_2 \frac{1}{p_i}$$

- **Shannon's source coding theorem[Informal]**
  Best compression achieves rate $H(p)$ bits per character.

Detour – Shannon Entropy

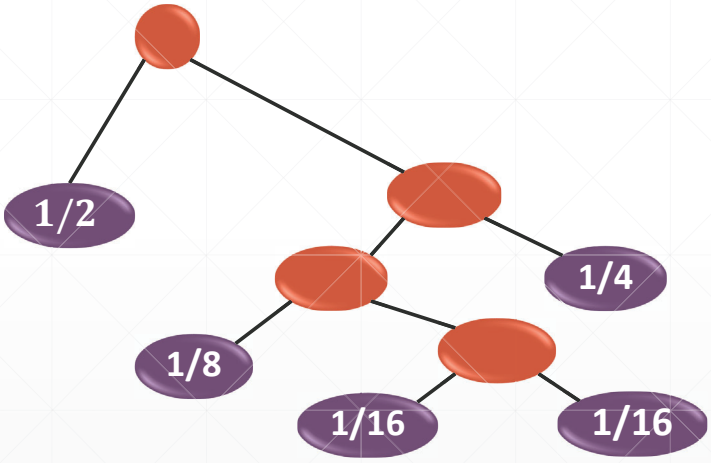


- Easy lower bound on the cost of a prefix code?
- Shannon Entropy:

$$H(p) = \sum_i p_i \log_2 \frac{1}{p_i}$$

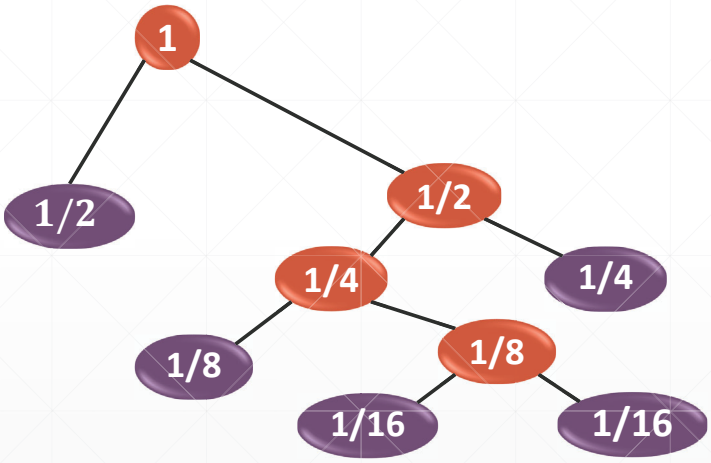
- **Shannon's source coding theorem[Informal]**
  Best compression achieves rate $H(p)$ bits per character.
- Expected rate for any prefix code $\geq H(p)$.

Shannon Entropy



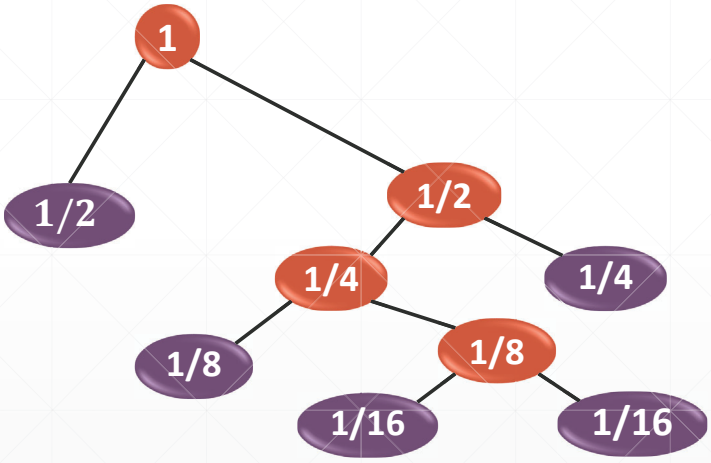
- Let $q_i = 1/2^{\text{depth}(i)}$.
- Write q_i in each leaf i.

Shannon Entropy



- Let $q_i = 1/2^{\text{depth}(i)}$.
- Write q_i in each leaf i.
- Sum of all q_i equals 1.

Shannon Entropy



- Let $q_i = 1/2^{\text{depth}(i)}$.
- Write q_i in each leaf i.
- Sum of all q_i equals 1.
- cost $= \sum p_i \text{ depth } (i)$
  $= \sum p_i \log_2 1/q_i$

Proof.*

Encoding cost:

$$\text{cost} = \sum p_i \log_2 1/q_i$$

Entropy:

$$H = \sum p_i \log_2 1/p_i$$

The minimum of the function $f(x_1, \dots, x_n) = \sum p_i \log_2 1/x_i$ on the hyperplane $\sum_i x_i = \langle x, \mathbf{1} \rangle = 1$ is at the point $x_i = p_i$. Because,

$$\nabla f = -c \left(\frac{p_1}{x_1}, \dots, \frac{p_n}{x_n} \right)$$

must be colinear with the vector **1** of all 1's at an extreme point.

*The proof is given for completeness. It's not essential for the future material.