# Greedy Algorithms: Huffman Coding

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 7

# Efficient communication

- Send messages in English or Hindi or Tamil or ...

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$

# Efficient communication

- Send messages in English or Hindi or Tamil or . . .
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

# Efficient communication

- Send messages in English or Hindi or Tamil or . . .
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots ($0$) and dashes ($1$)

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots ($0$) and dashes ($1$)
- Encoding of $e$ is $0$, $t$ is $1$, $a$ is $01$

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots ($0$) and dashes ($1$)
- Encoding of $e$ is $0$, $t$ is $1$, $a$ is $01$
- Decode $0101$

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length $5$
  - $26$ letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots ($0$) and dashes ($1$)
- Encoding of $e$ is $0$, $t$ is $1$, $a$ is $01$
- Decode $0101$
  - $aa$

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \dots, z\}$ using $\{0, 1\}$
  - Use binary strings of length 5
  - 26 letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots (0) and dashes (1)
- Encoding of $e$ is 0, $t$ is 1, $a$ is 01
- Decode 0101
  - $aa$ , $etet$, $aet$, $eta$?

# Efficient communication

- Send messages in English or Hindi or Tamil or ...
- Each language has its own alphabet
- Digital communication uses $\{0, 1\}$
- Encode $\{a, b, \ldots, z\}$ using $\{0, 1\}$
  - Use binary strings of length 5
  - 26 letters, $2^4 < 26 \leq 2^5$
- Can we do better?
  - Use shorter strings for more frequent letters
  - Optimize data transfer

Variable length encoding

Morse Code

- Encode letters using dots (0) and dashes (1)
- Encoding of $e$ is 0, $t$ is 1, $a$ is 01
- Decode 0101
  - *aa* , *etet*, *aet*, *eta*?
- Use pauses between letters
  - Like adding a third symbol for encoding

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

0 0 1 0 0 0 0 0 1 1 1 0 1

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

$0 \quad 0 \quad 1 \mid 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1$

$c$

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

$$0 \quad 0 \quad 1 \mid 0 \quad 0 \quad 0 \mid 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1$$

$\quad c \qquad e$

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

$$0 \quad 0 \quad 1 \,|\, 0 \quad 0 \quad 0 \,|\, 0 \quad 0 \quad 1 \,|\, 1 \quad 1 \quad 0 \quad 1$$

$$c \qquad e \qquad c$$

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

$$0\ 0\ 1 \mid 0\ 0\ 0 \mid 0\ 0\ 1 \mid 1\ 1 \mid 0\ 1$$

$c$ $e$ $c$ $a$

# Prefix codes

- Encoding of $x$, $E(x)$, is not a prefix of $E(y)$ for any $x$, $y$
  - In Morse Code, $E(e) = 0$ is a prefix of $E(a) = 01$
- Example of a prefix code

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |

- Decode

$$\underbrace{0\ 0\ 1}_{c} \mid \underbrace{0\ 0\ 0}_{e} \mid \underbrace{0\ 0\ 1}_{c} \mid \underbrace{1\ 1}_{a} \mid \underbrace{0\ 1}_{b} \mid$$

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters
- $A = \{x_1, x_2, \ldots, x_m\}$
  - $f(x_1) + f(x_2) + \cdots + f(x_m) = 1$
  - $f(x)$ is "probability" that next letter is $x$ corpus

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters
- $A = \{x_1, x_2, \ldots, x_m\}$
  - $f(x_1) + f(x_2) + \cdots + f(x_m) = 1$
  - $f(x)$ is "probability" that next letter is $x$ corpus
- Message $M$ to be transmitted has $n$ symbols
  - Each letter $x$ occurs $n \cdot f(x)$ times

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters
- $A = \{x_1, x_2, \ldots, x_m\}$
  - $f(x_1) + f(x_2) + \cdots + f(x_m) = 1$
  - $f(x)$ is "probability" that next letter is $x$ corpus
- Message $M$ to be transmitted has $n$ symbols
  - Each letter $x$ occurs $n \cdot f(x)$ times
- Each $x$ is encoded as $E(x)$ with length $|E(x)|$

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters
- $A = \{x_1, x_2, \ldots, x_m\}$
  - $f(x_1) + f(x_2) + \cdots + f(x_m) = 1$
  - $f(x)$ is "probability" that next letter is $x$ corpus
- Message $M$ to be transmitted has $n$ symbols
  - Each letter $x$ occurs $n \cdot f(x)$ times
- Each $x$ is encoded as $E(x)$ with length $|E(x)|$
- Total message length is

$$\sum_{x \in A} n \cdot f(x) \cdot |E(x)|$$

# Optimal prefix codes

- Measure frequency $f(x)$ of each letter
  - Fraction of occurrences of $x$ over large text corpus
  - Number of times $x$ appears divided by total number of letters
- $A = \{x_1, x_2, \ldots, x_m\}$
  - $f(x_1) + f(x_2) + \cdots + f(x_m) = 1$
  - $f(x)$ is "probability" that next letter is $x$ corpus
- Message $M$ to be transmitted has $n$ symbols
  - Each letter $x$ occurs $n \cdot f(x)$ times
- Each $x$ is encoded as $E(x)$ with length $|E(x)|$
- Total message length is

$$\sum_{x \in A} n \cdot f(x) \cdot |E(x)|$$

- Average number of bits per letter in encoding

$$\sum_{x \in A} f(x) \cdot |E(x)|$$

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.25
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 3)$ $+(0.18 \cdot 2) + (0.05 \cdot 3)$

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.25
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 3)$ $+(0.18 \cdot 2) + (0.05 \cdot 3)$
- Fixed length encoding would require 3 bits per letter — $2^2 < 5 \leq 2^3$
  - 25% saving using variable length code

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.25
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 3)$ $+(0.18 \cdot 2) + (0.05 \cdot 3)$
- Fixed length encoding would require 3 bits per letter — $2^2 < 5 \leq 2^3$
  - 25% saving using variable length code

- A better encoding

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.25
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 3)$ $+(0.18 \cdot 2) + (0.05 \cdot 3)$
- Fixed length encoding would require 3 bits per letter — $2^2 < 5 \leq 2^3$
  - 25% saving using variable length code

- A better encoding

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.23
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 2)$ $+(0.18 \cdot 3) + (0.05 \cdot 3)$

# Optimal prefix codes

- Suppose we have the following frequencies for our earlier example

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 01 | 001 | 10 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.25
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 3)$ $+(0.18 \cdot 2) + (0.05 \cdot 3)$
- Fixed length encoding would require 3 bits per letter — $2^2 < 5 \leq 2^3$
  - 25% saving using variable length code

- A better encoding

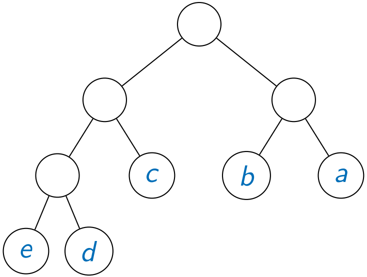
| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

- Average number of bits per letter is 2.23
  - $(0.32 \cdot 2) + (0.25 \cdot 2) + (0.20 \cdot 2)$ $+(0.18 \cdot 3) + (0.05 \cdot 3)$
- Given a set of letters $A$ and frequences $f(x)$ for each $x$, produce the most efficient prefix code possible
  - Minimize $ABL(A)$ — Average Bits per Letter

# Codes as trees

- Encoding can be viewed as a binary tree

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |

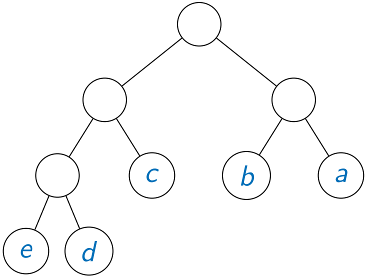


# Codes as trees

- Encoding can be viewed as a binary tree

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |

- Letters are at the leaves

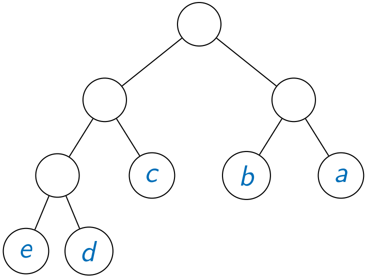


# Codes as trees

- Encoding can be viewed as a binary tree

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |

- Letters are at the leaves
- Path to leaf describes encoding — 0 is left, 1 is right

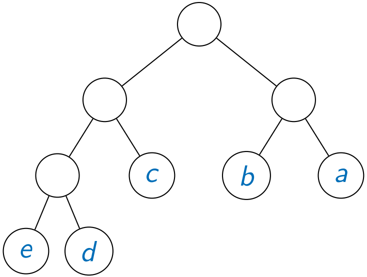


# Codes as trees

- Encoding can be viewed as a binary tree

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $E(x)$ | 11 | 10 | 01 | 001 | 000 |

- Letters are at the leaves
- Path to leaf describes encoding — 0 is left, 1 is right
- Prefix code — no internal nodes encode letters

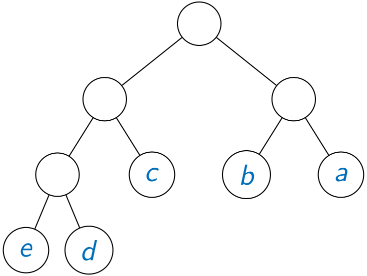


# Codes as trees

**Claim 1**

Any optimal prefix code produces a full tree

- Full tree Each node has 0 or 2 children

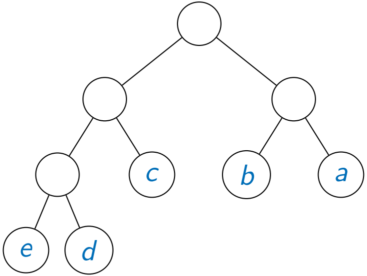


# Codes as trees

**Claim 1**

Any optimal prefix code produces a full tree

- Full tree: Each node has 0 or 2 children
- For a node with only one child, “promote” child to get a shorter tree



# Codes as trees

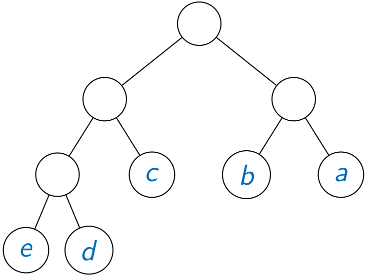
**Claim 1**

Any optimal prefix code produces a full tree

- Full tree  Each node has 0 or 2 children
- For a node with only one child, "promote" child to get a shorter tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$



# Codes as trees

**Claim 1**

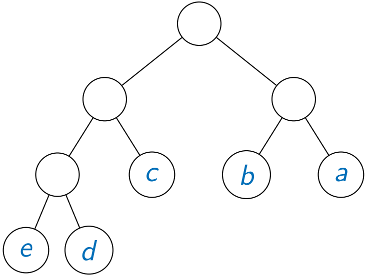
Any optimal prefix code produces a full tree

- Full tree  Each node has 0 or 2 children
- For a node with only one child, "promote" child to get a shorter tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

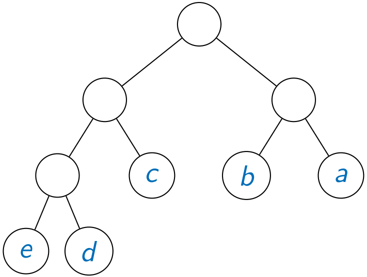
- If $f(y) > f(x)$, exchange labels, improve tree



# Codes as trees

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

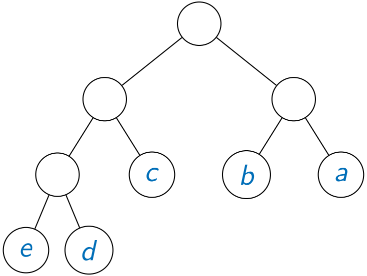


# Codes as trees

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

- If not, the sibling of this leaf has children

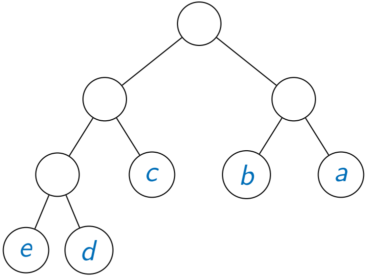


# Codes as trees

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

- If not, the sibling of this leaf has children
- There is a leaf at lower depth

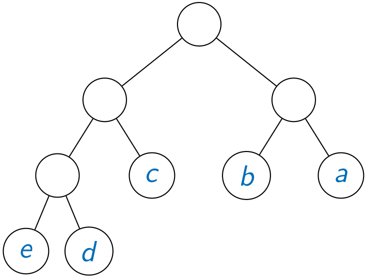


# Codes as trees

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

- If not, the sibling of this leaf has children
- There is a leaf at lower depth
- The leaf we started with is not a maximum depth!



# A recursive algorithm

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$
- Update alphabet $A' = (A \setminus \{x, y\}) \cup \{xy\}$

**Claim 1**

Any optimal prefix code produces a full tree

**Claim 2**

In an optimal tree, if leaf labelled $x$ is at smaller depth (higher) than leaf labelled $y$, $f(x) \geq f(y)$

**Claim 3**

In an optimal tree, for any leaf at maximum depth, its sibling is also a leaf $x$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$
- Update alphabet $A' = (A \setminus \{x, y\}) \cup \{xy\}$
- Recursively find an optimal tree $T'$ for $A'$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$
- Update alphabet $A' = (A \setminus \{x, y\}) \cup \{xy\}$
- Recursively find an optimal tree $T'$ for $A'$
- $T'$ will have a leaf labelled $xy$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$
- Update alphabet $A' = (A \setminus \{x, y\}) \cup \{xy\}$
- Recursively find an optimal tree $T'$ for $A'$
- $T'$ will have a leaf labelled $xy$
- Replace this leaf by internal node with two children labelled $x$, $y$

# A recursive algorithm

- From Claim 3, leaves at maximum depth occur in pairs
- From Claim 2, these must have lowest frequencies
- Pick $x$, $y$ with smallest $f(x)$, $f(y)$
- Assign $x$, $y$ to lowest pair of leaves (left/right does not matter)
- "Combine" $x$, $y$ into new letter $xy$ with $f(xy) = f(x) + f(y)$
- Update alphabet $A' = (A \setminus \{x, y\}) \cup \{xy\}$
- Recursively find an optimal tree $T'$ for $A'$
- $T'$ will have a leaf labelled $xy$
- Replace this leaf by internal node with two children labelled $x$, $y$
- Huffman coding — David E Huffman

# Huffman's algorithm

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

# Huffman's algorithm

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

- Combine $d$, $e$ as $de$

# Huffman's algorithm

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$

# Huffman's algorithm

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$

# Huffman's algorithm

| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$
- Two letters, base case, build a tree

# Huffman's algorithm

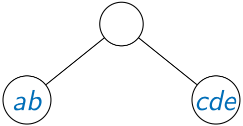
| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$
- Two letters, base case, build a tree



# Huffman's algorithm

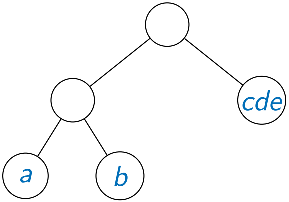
| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$
- Two letters, base case, build a tree
- Repeatedly split compound leaves



# Huffman's algorithm

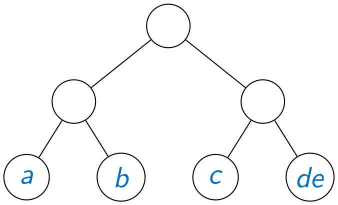
| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$
- Two letters, base case, build a tree
- Repeatedly split compound leaves



# Huffman's algorithm

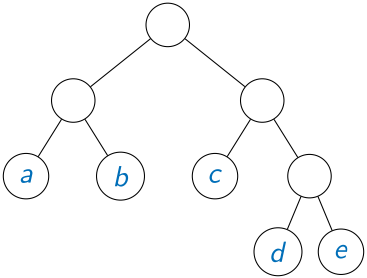
| $x$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.18 | 0.05 |

| $x$ | $a$ | $b$ | $c$ | $de$ |
|---|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.20 | 0.23 |

| $x$ | $a$ | $b$ | $cde$ |
|---|---|---|---|
| $f(x)$ | 0.32 | 0.25 | 0.43 |

| $x$ | $ab$ | $cde$ |
|---|---|---|
| $f(x)$ | 0.57 | 0.43 |

- Combine $d$, $e$ as $de$
- Combine $c$, $de$ as $cde$
- Combine $a$, $b$ as $ab$
- Two letters, base case, build a tree
- Repeatedly split compound leaves



# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0,1\}$ is optimal
- Assume optimality for $|A| = k-1$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- From $T'$ to $T$, only change to $ABL$ is due to $xy$, $x$, $y$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- From $T'$ to $T$, only change to $ABL$ is due to $xy$, $x$, $y$
- Subtract $depth(xy)f(xy)$, add $depth(x)f(x) + depth(y)f(y)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0,1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- From $T'$ to $T$, only change to $ABL$ is due to $xy$, $x$, $y$
- Subtract $depth(xy)f(xy)$, add $depth(x)f(x) + depth(y)f(y)$
- $f(xy) = f(x) + f(y)$, $depth(x) = depth(y) = 1 + depth(xy)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k{-}1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- From $T'$ to $T$, only change to $ABL$ is due to $xy$, $x$, $y$
- Subtract $depth(xy)f(xy)$, add $depth(x)f(x) + depth(y)f(y)$
- $f(xy) = f(x) + f(y)$, $depth(x) = depth(y) = 1 + depth(xy)$
- Net increase is $f(x) + f(y)$, which is $f(xy)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$
- Shuffle the labels in $S$ so that lowest frequency $x$, $y$ are siblings

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$
- Shuffle the labels in $S$ so that lowest frequency $x$, $y$ are siblings
- Merge $x$,$y$ as $xy$, contract $S$ to $S'$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$
- Shuffle the labels in $S$ so that lowest frequency $x$, $y$ are siblings
- Merge $x$,$y$ as $xy$, contract $S$ to $S'$
- $S'$ is over same $A'$ as $T'$, $T'$ is optimal for $A'$, so $ABL(T') \leq ABL(S')$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$
- Shuffle the labels in $S$ so that lowest frequency $x$, $y$ are siblings
- Merge $x$,$y$ as $xy$, contract $S$ to $S'$
- $S'$ is over same $A'$ as $T'$, $T'$ is optimal for $A'$, so $ABL(T') \leq ABL(S')$
- $ABL(S) - ABL(S') = ABL(T) - ABL(T') = f(xy)$

# Optimality of Huffman's algorithm

By induction on the size of alphabet $A$

- Base case, $|A| = 2$
  - Single letter code $\{0, 1\}$ is optimal
- Assume optimality for $|A| = k-1$
- For $|A| = k$
  - Combine lowest frequency $x$,$y$ as $xy$
  - Construct tree $T'$ for $A'$
  - $ABL(T')$ is optimal by induction
  - Expand leaf $xy$ to get $T$

**Claim**

$ABL(T) - ABL(T') = f(xy)$

- Suppose there is an optimal tree $S$ for $A$ with $ABL(S) < ABL(T)$
- Shuffle the labels in $S$ so that lowest frequency $x$, $y$ are siblings
- Merge $x$,$y$ as $xy$, contract $S$ to $S'$
- $S'$ is over same $A'$ as $T'$, $T'$ is optimal for $A'$, so $ABL(T') \leq ABL(S')$
- $ABL(S) - ABL(S') = ABL(T) - ABL(T') = f(xy)$
- Hence $ABL(T) \leq ABL(S)$, a contradiction

# Implementation, complexity

- At each recursive step, extract letters with minimum frequency and replace by composite letter with combined frequency

# Implementation, complexity

- At each recursive step, extract letters with minimum frequency and replace by composite letter with combined frequency
- Store frequencies in an array
  - Linear scan to find minimum values
  - $|A| = k$, number of recursive calls is $k-1$
  - Complexity is $O(k^2)$

# Implementation, complexity

- At each recursive step, extract letters with minimum frequency and replace by composite letter with combined frequency
- Store frequencies in an array
  - Linear scan to find minimum values
  - $|A| = k$, number of recursive calls is $k-1$
  - Complexity is $O(k^2)$
- Instead, maintain frequencies in an heap
  - Extracting two minimum frequency letters and adding back compound letter are both $O(\log k)$
  - Complexity drops to $O(k \log k)$

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice
- Never go back and consider other pairings of letters

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice
- Never go back and consider other pairings of letters

## Historical notes

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice
- Never go back and consider other pairings of letters

## Historical notes

- Claude Shannon invented information theory
  - Mathematical lower bounds on the size of optimal encodings
  - Does not describe how to construct optimal codes

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice
- Never go back and consider other pairings of letters

## Historical notes

- Claude Shannon invented information theory
  - Mathematical lower bounds on the size of optimal encodings
  - Does not describe how to construct optimal codes
- Shannon and Robert Fano came up with a recursive solution (Shannon-Fano codes) that was not optimal

# Summary

## Why is Huffman coding greedy?

- Recursively combine letters with lowest frequencies
- Locally optimal choice
- Never go back and consider other pairings of letters

## Historical notes

- Claude Shannon invented information theory
  - Mathematical lower bounds on the size of optimal encodings
  - Does not describe how to construct optimal codes
- Shannon and Robert Fano came up with a recursive solution (Shannon-Fano codes) that was not optimal
- Huffman was a graduate student in Fano's class and discovered his algorithm during the course