Lempel-Ziv-Welch algorithm

Peter Hajnal

Bolyai Institute, University of Szeged, Hungary

2023 fall

Coding: Reminder notes

Coding: Reminder notes

We have a non-empty, finite Σ alphabet.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Enoding texts

$$e : \Sigma^* \to \{0,1\},$$

where e is an encoding function.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Enoding texts

$$e : \Sigma^* \to \{0, 1\},$$

where e is an encoding function.

Coding scheme

$$e \qquad ''+'' \qquad d : \{0, 1\} \to \Sigma^*,$$

where d is a dencoding function.

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Enoding texts

$$e : \Sigma^* \to \{0, 1\},$$

where e is an encoding function.

Coding scheme

$$e \qquad \text{"}+\text{"} \qquad d : \{0, 1\} \to \Sigma^*,$$

where d is a dencoding function.

Two parties/sides are involved in coding: Sender/Receiver,

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Enoding texts

$$e : \Sigma^* \to \{0, 1\},$$

where e is an encoding function.

Coding scheme

$$e \qquad "+" \qquad d : \{0, 1\} \to \Sigma^*,$$

where d is a dencoding function.

Two parties/sides are involved in coding: Sender/Receiver, A/B,

Coding: Reminder notes

We have a non-empty, finite Σ alphabet. The elements of Σ are called characters. Σ^* is the set of finite character sequences. A finite character sequence is called *text*/*file*/*sentence*/*word*. ($\varepsilon \in \Sigma^*$ is a special text, the empty text.)

Enoding texts

$$e : \Sigma^* \to \{0, 1\},$$

where e is an encoding function.

Coding scheme

$$e \qquad " + " \qquad d : \{0, 1\} \to \Sigma^*,$$

where d is a dencoding function.

Two parties/sides are involved in coding: Sender/Receiver, A/B, Alice/Bob.

Character based coding with fixed length

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^{\ell},$$

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0, 1\}^{\ell},$$

(we assume that $|\Sigma| \leq 2^{\ell}$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^\ell,$$

(we assume that $|\Sigma| \leq 2^\ell$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).
Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^{\ell},$$

(we assume that $|\Sigma| \leq 2^{\ell}$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).
Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively)

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^\ell,$$

(we assume that $|\Sigma| \leq 2^\ell$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).
Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^\ell,$$

(we assume that $|\Sigma| \leq 2^\ell$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).
Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k (compute k at the character, the code of the actual character),

Character based coding with fixed length

Character based encoding with fixed length

There is a constant ℓ and a 1-1 map

$$k : \Sigma \to \{0,1\}^\ell,$$

(we assume that $|\Sigma| \leq 2^\ell$, i.e. $\ell \geq \lceil \log_2 |\Sigma| \rceil$).
Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k (compute k at the character, the code of the actual character), and concatenate the codes of the characters.

Character based coding with variable length

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" ($ell(b)$ is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" ($ell(b)$ is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0, 1\}^*.$$

Character based coding with variable length

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" (*ell*(b) is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" (*ell*(b) is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively)

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" ($ell(b)$ is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" ($ell(b)$ is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k (compute k at the character, the code of the actual character),

Character based coding with variable length

H-tree and prefix-free encoding of characters

There is T rooted, binary, plane (0/1 labels at the two edges going to the two children) tree, and a bijective map between Σ and the leaves of T. The root-$\ell(b)$ path defines the code of the character "b" (*ell*(b) is the leaf matched to the character b):

$$k : \Sigma \to \mathcal{L} \subset \{0,1\}^*.$$

Character based coding with variable length

Encoding based on k is

$$\widehat{k} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{k}(\tau)$ by slicing τ into characters (if τ is not empty, then we take the first character, and we process the leftover text recursively) we encode the characters using k (compute k at the character, the code of the actual character), and concatenate the codes of the characters.

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Encoding based on d is

$$\widehat{d} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{d}(\tau)$ by slicing τ into words($\in D$)

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Encoding based on d is

$$\widehat{d} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{d}(\tau)$ by slicing τ into words($\in D$) (if τ is not empty, then we take the LONGEST prefix of it, that is in the dictionary, and we process the leftover text recursively)

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Encoding based on d is

$$\widehat{d} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{d}(\tau)$ by slicing τ into words($\in D$) (if τ is not empty, then we take the LONGEST prefix of it, that is in the dictionary, and we process the leftover text recursively) we encode the word, actually cut off, using k

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Encoding based on d is

$$\widehat{d} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{d}(\tau)$ by slicing τ into words($\in D$) (if τ is not empty, then we take the LONGEST prefix of it, that is in the dictionary, and we process the leftover text recursively) we encode the word, actually cut off, using k (compute k at the word, the code of the actual word),

Dictionary based encoding with fixed length

Let D be a finite set of keywords: $D \subset \Sigma^*$. We always assume that $\Sigma \equiv \Sigma^1 \subset D$.

Character based coding with fixed length

There is a constant ℓ and a 1-1 "dictionary" map

$$d : D \to \{0,1\}^\ell.$$

Encoding based on d is

$$\widehat{d} : \Sigma^* \to \{0,1\}^*,$$

where for a text $\tau \in \Sigma^*$, we obtain $\widehat{d}(\tau)$ by slicing τ into words($\in D$) (if τ is not empty, then we take the LONGEST prefix of it, that is in the dictionary, and we process the leftover text recursively) we encode the word, actually cut off, using k (compute k at the word, the code of the actual word), and concatenate the codes of the words.

Dictionary based decoding with fixed length

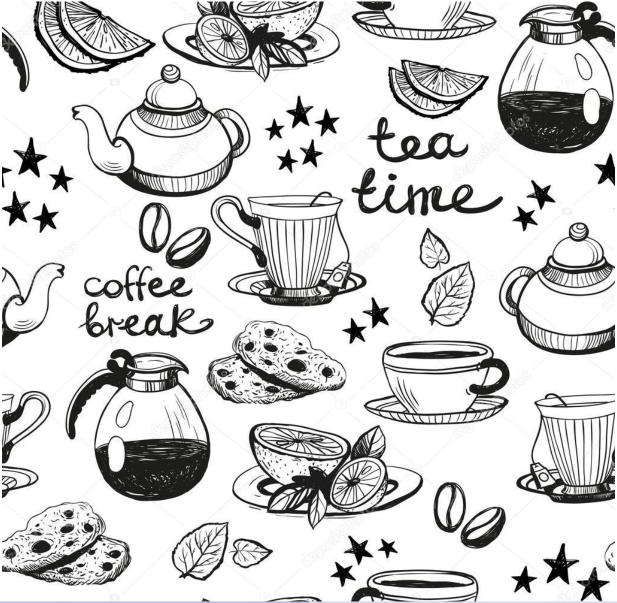
Dictionary based decoding with fixed length

If the dictionary, (D, d) is known for both parties,

Dictionary based decoding with fixed length

If the dictionary, (D, d) is known for both parties, then the decoding is very easy.

Break



Fixed length LZW: The initial dictionary

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

We choose a suitable length $\ell > 7 = \log_2 |\Sigma_{ASCII}|$.

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

We choose a suitable length $\ell > 7 = \log_2 |\Sigma_{ASCII}|$. Our dictionary is capable to store 2^ℓ words.

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

We choose a suitable length $\ell > 7 = \log_2 |\Sigma_{ASCII}|$. Our dictionary is capable to store 2^ℓ words.

The initial dictionary contains Σ and two special "messages" (not words): START, STOP.

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

We choose a suitable length $\ell > 7 = \log_2 |\Sigma_{ASCII}|$. Our dictionary is capable to store 2^ℓ words.

The initial dictionary contains Σ and two special "messages" (not words): START, STOP.

The code of ASCII characters are the ASCII code padded by $0^{\ell-7}$ at the beginning. The code of "START" is 128, the code of "STOP" is 129.

Fixed length LZW: The initial dictionary

We assume that $\Sigma = \Sigma_{ASCII}$, the character set of the ASCII code.

We choose a suitable length $\ell > 7 = \log_2 |\Sigma_{ASCII}|$. Our dictionary is capable to store 2^ℓ words.

The initial dictionary contains Σ and two special "messages" (not words): START, STOP.

The code of ASCII characters are the ASCII code padded by $0^{\ell-7}$ at the beginning. The code of "START" is 128, the code of "STOP" is 129.

Example

We assume $\ell = 12$. The ASCII code of the letter 'a' is $97 \equiv 110\ 0001$. In the dictionary its code is $97 \equiv 0000\ 0110\ 0001$.

Fixed length LZW: Encoding with extending the dictionary

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:`
Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^+ = w"c"$ was not prefix.

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:` Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^{+} = w\text{"}c\text{"}$ was not prefix.

`Encoding the actual chunk:` From the dictionary we get the code for w. We send it over.

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:` Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^+ = w"c"$ was not prefix.

`Encoding the actual chunk:` From the dictionary we get the code for w. We send it over.

`Update:` Update the processed and leftover arts of the text.

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:` Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^+ = w"c"$ was not prefix.

`Encoding the actual chunk:` From the dictionary we get the code for w. We send it over.

`Update:` Update the processed and leftover arts of the text.

`Extending the dictionary:` We add the word w^+ with the first available bit sequence in the dictionary.

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:` Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^+ = w"c"$ was not prefix.

`Encoding the actual chunk:` From the dictionary we get the code for w. We send it over.

`Update:` Update the processed and leftover arts of the text.

`Extending the dictionary:` We add the word w^+ with the first available bit sequence in the dictionary. We skip the extension step if the dictionary is full.

Fixed length LZW: Encoding with extending the dictionary

`Finding the new chunk of the text to be processed:` Assume that the sender found the word w as a prefix of the unprocessed/leftover text, but $w^+ = w$" c" was not prefix.

`Encoding the actual chunk:` From the dictionary we get the code for w. We send it over.

`Update:` Update the processed and leftover arts of the text.

`Extending the dictionary:` We add the word w^+ with the first available bit sequence in the dictionary. We skip the extension step if the dictionary is full.

`Stop:` If we processed the whole set we send "129".

Fixed length LZW: Sender vs receiver, example 1

Fixed length LZW: Sender vs receiver, example 1

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Fixed length LZW: Sender vs receiver, example 1

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

Fixed length LZW: Sender vs receiver, example 1

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —— 128 109 97 ——→ receiver

Fixed length LZW: Sender vs receiver, example 1

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —— 128 109 97 ——→ receiver

m|a|ma ma mamaligát főz ——→ m|a|??. . .

Fixed length LZW: Sender vs receiver, example 1

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —— 128 109 97 ——→ receiver

m|a|ma ma mamaligát főz ——→ m|a|??...

START	128	START	128
STOP	129	STOP	129
ma	130	ma	130
am	131	a?	131

Fixed length LZW: Sender vs receiver, example 2

Fixed length LZW: Sender vs receiver, example 2

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Fixed length LZW: Sender vs receiver, example 2

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

Fixed length LZW: Sender vs receiver, example 2

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender $\xrightarrow{\quad 128 \quad 109 \quad 97 \quad 130 \quad}$ receiver

Fixed length LZW: Sender vs receiver, example 2

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —— 128 109 97 130 ——→ receiver

m|a|ma| ma mamaligát főz ——→ m|a|ma|??...

Fixed length LZW: Sender vs receiver, example 2

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —128 109 97 130→ receiver

m|a|ma| ma mamaligát főz → m|a|ma|??...

START	128	START	128
STOP	129	STOP	129
ma	130	ma	130
am	131	am	131
ma␣	132	ma?	132

Fixed length LZW: Sender vs receiver, example 3

Fixed length LZW: Sender vs receiver, example 3

The text: ”mama ma mamaligát főz” (the ASCII codes are m:109; a:97; SPACE:32)

Fixed length LZW: Sender vs receiver, example 3

The text: ”mama ma mamaligát főz” (the ASCII codes are m:109; a:97; SPACE:32)

Example

Fixed length LZW: Sender vs receiver, example 3

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender $\xrightarrow{128 \quad 109 \quad 97 \quad 130 \quad 32}$ receiver

Fixed length LZW: Sender vs receiver, example 3

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender $\xrightarrow{128 \quad 109 \quad 97 \quad 130 \quad 32}$ receiver

m|a|ma| |ma mamaligát főz $\longrightarrow$ m|a|ma|␣|??. . .

Fixed length LZW: Sender vs receiver, example 3

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —— 128 109 97 130 32 ——→ receiver

m|a|ma| |ma mamaligát főz ——→ m|a|ma|␣|??. . .

START	128	START	128
STOP	129	STOP	129
ma	130	ma	130
am	131	am	131
ma␣	132	ma␣	132
␣m	133	␣?	133

Fixed length LZW: Sender vs receiver, example 4

Fixed length LZW: Sender vs receiver, example 4

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Fixed length LZW: Sender vs receiver, example 4

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

Fixed length LZW: Sender vs receiver, example 4

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender $\xrightarrow{128 \quad 109 \quad 97 \quad 130 \quad 32 \quad 132}$ receiver

Fixed length LZW: Sender vs receiver, example 4

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender $\xrightarrow{128 \quad 109 \quad 97 \quad 130 \quad 32 \quad 132}$ receiver

m|a|ma| |ma |mamaligát főz $\longrightarrow$ m|a|ma|␣|ma␣|??...

Fixed length LZW: Sender vs receiver, example 4

The text: "mama ma mamaligát főz" (the ASCII codes are m:109; a:97; SPACE:32)

Example

sender —128 109 97 130 32 132→ receiver

m|a|ma| |ma |mamaligát főz ⟶ m|a|ma|␣|ma␣|??...

START	128	START	128
STOP	129	STOP	129
ma	130	ma	130
am	131	am	131
ma␣	132	ma␣	132
␣m	133	␣m	133
ma␣m	134	ma␣?	134

Fixed length LZW: Sender vs receiver

Fixed length LZW: Sender vs receiver

Theorem

Before the whole text is encoded the receiver dictionary is the same as the sender dictionary except the last line, where the word''s last character is unknown.

Fixed length LZW: Sender vs receiver

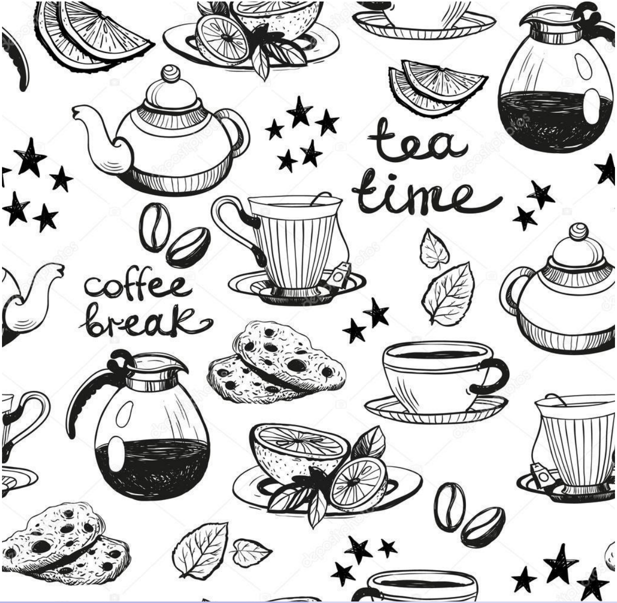
Theorem

Before the whole text is encoded the receiver dictionary is the same as the sender dictionary except the last line, where the word''s last character is unknown.

Corollary

After obtaining a new part of the code the receiver side can make up for the disadvantage in the previous dictionary.

Break



Dictionaries with increasing length

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility. If we set ℓ too small, then the dictionary might be full very soon.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility. If we set ℓ too small, then the dictionary might be full very soon.

The solution is a simple modification:

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility. If we set ℓ too small, then the dictionary might be full very soon.

The solution is a simple modification:

`Initialization:` Set $\ell = 8$.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility. If we set ℓ too small, then the dictionary might be full very soon.

The solution is a simple modification:

`Initialization:` Set $\ell = 8$.

`The NEW extending the dictionary:` We add the word w^+ with the first available bit sequence in the dictionary.

Dictionaries with increasing length

In our first version of LZW the length ℓ of code bit sequences is fixed.

This is a problem. If we set ℓ too large, then the final dictionary will be short compared to the possibility. If we set ℓ too small, then the dictionary might be full very soon.

The solution is a simple modification:

`Initialization:` Set $\ell = 8$.

`The NEW extending the dictionary:` We add the word w^+ with the first available bit sequence in the dictionary. If the dictionary is full, then $\ell \leftarrow \ell + 1$. Available bit sequences will appear, the dictionary extension is possible.

Dictionaries with increasing length: Receiver side

Dictionaries with increasing length: Receiver side

Theorem

The receiver side can decode the actual length, set by the sender side.

Dictionaries with increasing length: Receiver side

Theorem

The receiver side can decode the actual length, set by the sender side.

Proof: Easy.

Dictionaries with increasing length: Receiver side

Theorem

The receiver side can decode the actual length, set by the sender side.

Proof: Easy. The dictionary on the receiver side has the same number of lines. The timing of the incrementation of the length depends on the number of lines.

This is the end!

Thank you for your attention!