# A Crash Course on Data Compression

## 5. Dictionary-based Compressors

**Giulio Ermanno Pibiri**

ISTI-CNR, giulio.ermanno.pibiri@isti.cnr.it

@giulio_pibiri

@jermp

# Overview

- LZ77
- LZSS
- LZ78
- LZW

# The Dictionary-based Coding Problem

- **Problem.** We are given a list $L[1..n]$ of $n$ symbols and we are asked to compress it in *as few as possible bits*.
- **Idea.** Assume to have a *dictionary* of $m$ strings, $D[1..m]$.
  If the substring $L[i\,..\,j]$ is equal to the string $D[k]$, then we can represent $L[i\,..\,j]$ with $k$.
- Simple idea but very effective when $L$ is *repetitive,* i.e., it has many equal substrings.
- **Q.** How to build the dictionary $D$ such that $L$ is compressed effectively?
  (As effectively as possible?)

# LZ77

## Lempel and Ziv, 1977

- **Idea.** The dictionary $D$ is *not explicitly built*, rather it is logically represented by *all* the substrings of $L[i - W \,..\, i]$, where $i$ is the length of the prefix of $L$ processed by the algorithm and $W > 0$ is a parameter.
- **Algorithm.**
  - At the beginning: $i = 1$.
  - At each step: determine the *longest common prefix* ($lcp$) between $L[i \,..\, n]$ and the substring starting (at most) $W$ positions before $i$ but possibly ending in $L[i \,..\, n]$.
  - If the $lcp$ is found at distance $d$ from $i$, the algorithm emits the triple $\langle d, |\, lcp \,|, c \rangle$, where $c$ is the next character following the $lcp$.
  - Then the algorithm advances by $|\, lcp \,| + 1$ characters, that is: $i = i + |\, lcp \,| + 1$.
- The algorithm is sometimes referred to as the process of *parsing* $L$ into *phrases*, the actual integer triples. The triples are then compressed using another coding method (e.g., a static code or Huffman).
- The parameter $W$ (the “window” size) controls a trade-off between encoding/decoding speed and compression ratio.

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = | A\ B\ R\ A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A \qquad 1 : \langle 0,0,A \rangle$$

0 1

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R\ A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2

$$1 : \langle 0,0,A \rangle$$
$$2 : \langle 0,0,B \rangle$$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3

$$1 : \langle 0,0,A \rangle$$
$$2 : \langle 0,0,B \rangle$$
$$3 : \langle 0,0,R \rangle$$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3

$$1 : \langle 0,0,A \rangle$$
$$2 : \langle 0,0,B \rangle$$
$$3 : \langle 0,0,R \rangle$$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3 4

$$1 : \langle 0,0,A \rangle$$
$$2 : \langle 0,0,B \rangle$$
$$3 : \langle 0,0,R \rangle$$
$$4 : \langle 3,1,C \rangle$$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3 4

$$1 : \langle 0,0,A \rangle$$
$$2 : \langle 0,0,B \rangle$$
$$3 : \langle 0,0,R \rangle$$
$$4 : \langle 3,1,C \rangle$$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3 4 5

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$

0 1 2 3 4 5

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B|R\ A\ C\ A\ C\ A$$

0 1 2 3 4 5 6

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B|R\ A\ C\ A\ C\ A$$

0 1 2 3 4 5 6

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B|R\ A\ C|A\ C\ A$$

0 1 2 3 4 5 6 7

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B|R\ A\ C|A\ C\ A$$

0 1 2 3 4 5 6 7

Overlap with the suffix!

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

# LZ77 — Encoding Demo

Let us assume an *unbounded* window size $W$, i.e., the algorithm can “look back” as far as it wants.

$$L = |A|B|R|A\ C|A\ D|A\ B\ R\ A\ B|R\ A\ C|A\ C\ A|$$

0 1 2 3 4 5 6 7 8

Overlap with the suffix!

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\text{EOF} \rangle$

# LZ77 — Decoding Demo

$L = ?$

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\text{EOF} \rangle$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\text{EOF} \rangle$$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$2 : L = A\,B$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\mathrm{EOF} \rangle$$

# LZ77 — Decoding Demo

$$L = ?$$

$1 : L = A$

$2 : L = A\,B$

$3 : L = A\,B\,R$

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\text{EOF} \rangle$

# LZ77 — Decoding Demo

$$L = ?$$

$1 : L = A$

$2 : L = A\,B$

$3 : L = A\,B\,R$

$4 : L = A\,B\,R\,A\,C$

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\text{EOF} \rangle$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$2 : L = A\,B$$

$$3 : L = A\,B\,R$$

$$4 : L = A\,B\,R\,A\,C$$

$$5 : L = A\,B\,R\,A\,C\,A\,D$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\mathrm{EOF} \rangle$$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$2 : L = A\,B$$

$$3 : L = A\,B\,R$$

$$4 : L = A\,B\,R\,A\,C$$

$$5 : L = A\,B\,R\,A\,C\,A\,D$$

$$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\mathrm{EOF} \rangle$$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$2 : L = A\,B$$

$$3 : L = A\,B\,R$$

$$4 : L = A\,B\,R\,A\,C$$

$$5 : L = A\,B\,R\,A\,C\,A\,D$$

$$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$$

$$7 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\mathrm{EOF} \rangle$$

# LZ77 — Decoding Demo

$$L = ?$$

$$1 : L = A$$

$$2 : L = A\,B$$

$$3 : L = A\,B\,R$$

$$4 : L = A\,B\,R\,A\,C$$

$$5 : L = A\,B\,R\,A\,C\,A\,D$$

$$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$$

$$7 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,?$$

$$1 : \langle 0,0,A \rangle$$

$$2 : \langle 0,0,B \rangle$$

$$3 : \langle 0,0,R \rangle$$

$$4 : \langle 3,1,C \rangle$$

$$5 : \langle 2,1,D \rangle$$

$$6 : \langle 7,4,B \rangle$$

$$7 : \langle 3,2,C \rangle$$

$$8 : \langle 2,3,\mathrm{EOF} \rangle$$

# LZ77 — Decoding Demo

$L = ?$

$1 : L = A$

$2 : L = A\,B$

$3 : L = A\,B\,R$

$4 : L = A\,B\,R\,A\,C$

$5 : L = A\,B\,R\,A\,C\,A\,D$

$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$

$7 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,?$

$8 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C$

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\mathrm{EOF} \rangle$

# LZ77 — Decoding Demo

$L = ?$

$1 : L = A$ $\quad\quad$ $1 : \langle 0,0,A \rangle$

$2 : L = A\,B$ $\quad\quad$ $2 : \langle 0,0,B \rangle$

$3 : L = A\,B\,R$ $\quad\quad$ $3 : \langle 0,0,R \rangle$

$4 : L = A\,B\,R\,A\,C$ $\quad\quad$ $4 : \langle 3,1,C \rangle$

$5 : L = A\,B\,R\,A\,C\,A\,D$ $\quad\quad$ $5 : \langle 2,1,D \rangle$

$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$ $\quad\quad$ $6 : \langle 7,4,B \rangle$

$7 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,?$ $\quad\quad$ $7 : \langle 3,2,C \rangle$

$8 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,?$ $\quad\quad$ $8 : \langle 2,3,\text{EOF} \rangle$

# LZ77 — Decoding Demo

$L = ?$

$1 : L = A$

$2 : L = A\,B$

$3 : L = A\,B\,R$

$4 : L = A\,B\,R\,A\,C$

$5 : L = A\,B\,R\,A\,C\,A\,D$

$6 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B$

$7 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,?$

$8 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,?$

$8 : L = A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$

$1 : \langle 0,0,A \rangle$

$2 : \langle 0,0,B \rangle$

$3 : \langle 0,0,R \rangle$

$4 : \langle 3,1,C \rangle$

$5 : \langle 2,1,D \rangle$

$6 : \langle 7,4,B \rangle$

$7 : \langle 3,2,C \rangle$

$8 : \langle 2,3,\text{EOF} \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = ABRACADABRABRACACA$$

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$ →
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = \underset{1}{|}A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$$

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

$1 : \langle 0,A \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0,c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp|\rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|BRACADABRABRACACA$$

$$1 \quad 2$$

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R\ A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ C\ A\ C\ A$$
$$1\quad 2\quad 3$$

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]



$1 : \langle 0,A \rangle$
$2 : \langle 0,B \rangle$
$3 : \langle 0,R \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R|A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$$

1 2 3 4

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

→

$1 : \langle 0,A \rangle$
$2 : \langle 0,B \rangle$
$3 : \langle 0,R \rangle$
$4 : \langle 3,1 \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R|A|C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$$

1 2 3 4 5

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

→

$1 : \langle 0,A \rangle$
$2 : \langle 0,B \rangle$
$3 : \langle 0,R \rangle$
$4 : \langle 3,1 \rangle$
$5 : \langle 0,C \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0,c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$L = |A|B|R|A|C|A\,D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$

1 2 3 4 5 6

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

→

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$
$3 : \langle 0,R\rangle$
$4 : \langle 3,1\rangle$
$5 : \langle 0,C\rangle$
$6 : \langle 2,1\rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0,c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R|A|C|A|D\,A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$$

1 2 3 4 5 6 7

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

→

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$
$3 : \langle 0,R\rangle$
$4 : \langle 3,1\rangle$
$5 : \langle 0,C\rangle$
$6 : \langle 2,1\rangle$
$7 : \langle 0,D\rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0,c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R|A|C|A|D|A\,B\,R\,A\,B\,R\,A\,C\,A\,C\,A$$

1 2 3 4 5 6 7 8

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

→

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$
$3 : \langle 0,R\rangle$
$4 : \langle 3,1\rangle$
$5 : \langle 0,C\rangle$
$6 : \langle 2,1\rangle$
$7 : \langle 0,D\rangle$
$8 : \langle 7,4\rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$L = |A|B|R|A|C|A|D|A\,B\,R\,A|B\,R\,A\,C\,A\,C\,A$

1 2 3 4 5 6 7 8 9

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

→

$1 : \langle 0,A \rangle$
$2 : \langle 0,B \rangle$
$3 : \langle 0,R \rangle$
$4 : \langle 3,1 \rangle$
$5 : \langle 0,C \rangle$
$6 : \langle 2,1 \rangle$
$7 : \langle 0,D \rangle$
$8 : \langle 7,4 \rangle$
$9 : \langle 3,3 \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0 \rangle$.
  We can directly emit $\langle 0, c \rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp| \rangle$ and made the algorithm advance by $|lcp|$ characters.

$$L = |A|B|R|A|C|A|D|A\,B\,R\,A|B\,R\,A|C\,A\,C\,A$$

1 2 3 4 5 6 7 8 9 10

$1 : \langle 0,0,A \rangle$
$2 : \langle 0,0,B \rangle$
$3 : \langle 0,0,R \rangle$
$4 : \langle 3,1,C \rangle$
$5 : \langle 2,1,D \rangle$
$6 : \langle 7,4,B \rangle$
$7 : \langle 3,2,C \rangle$
$8 : \langle 2,3,\text{EOF} \rangle$

[LZ77]

→

$1 : \langle 0,A \rangle$
$2 : \langle 0,B \rangle$
$3 : \langle 0,R \rangle$
$4 : \langle 3,1 \rangle$
$5 : \langle 0,C \rangle$
$6 : \langle 2,1 \rangle$
$7 : \langle 0,D \rangle$
$8 : \langle 7,4 \rangle$
$9 : \langle 3,3 \rangle$
$10 : \langle 10,2 \rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0, c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp|\rangle$ and made the algorithm advance by $|lcp|$ characters.

$L = |A|B|R|A|C|A|D|A\,B\,R\,A|B\,R\,A|C\,A|C\,A$

1 2 3 4 5 6 7 8 9 10 11

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

→

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$
$3 : \langle 0,R\rangle$
$4 : \langle 3,1\rangle$
$5 : \langle 0,C\rangle$
$6 : \langle 2,1\rangle$
$7 : \langle 0,D\rangle$
$8 : \langle 7,4\rangle$
$9 : \langle 3,3\rangle$
$10 : \langle 10,2\rangle$
$11 : \langle 2,2\rangle$

# LZSS

Storer and Szymanski, 1982

- **Idea.** Output *pairs*, not triples.
- **Observation 1.**
  When the $lcp$ has size 0, we always repeat the pair $\langle d = 0, |lcp| = 0\rangle$.
  We can directly emit $\langle 0,c\rangle$.
- **Observation 2.**
  When $|lcp| \neq 0$, we can avoid specifying the next character $c$, but just emit the pair $\langle d, |lcp|\rangle$ and made the algorithm advance by $|lcp|$ characters.

$L = |A|B|R|A|C|A|D|A\,B\,R\,A|B\,R\,A|C\,A|C\,A|$

1 2 3 4 5 6 7 8 9 10 11 12

$1 : \langle 0,0,A\rangle$
$2 : \langle 0,0,B\rangle$
$3 : \langle 0,0,R\rangle$
$4 : \langle 3,1,C\rangle$
$5 : \langle 2,1,D\rangle$
$6 : \langle 7,4,B\rangle$
$7 : \langle 3,2,C\rangle$
$8 : \langle 2,3,\text{EOF}\rangle$

[LZ77]

→

$1 : \langle 0,A\rangle$
$2 : \langle 0,B\rangle$
$3 : \langle 0,R\rangle$
$4 : \langle 3,1\rangle$
$5 : \langle 0,C\rangle$
$6 : \langle 2,1\rangle$
$7 : \langle 0,D\rangle$
$8 : \langle 7,4\rangle$
$9 : \langle 3,3\rangle$
$10 : \langle 10,2\rangle$
$11 : \langle 2,2\rangle$
$12 : \langle 0,\text{EOF}\rangle$

[LZSS]

# LZ77 in practice: `gzip`

http://www.gzip.org
https://zlib.net

**Gailly and Adler, 1995**

- **Q.** How do we determine efficiently the $lcp$ string between the suffix $L[i\,..\,n]$ and the preceding $W$ characters?
  **A.** Use a hash table of $q$-grams (strings of $q$ characters). Usually $q$ is small, say, $q = 3$. Insert all the $q$-grams of the window into the hash table: the $q$-gram is the key, the value is the list of *positions* of all the occurrences of the $q$-gram in the window.
- **Idea.** Use the $q$-grams' positions as "pointers" for the actual determination of the $lcp$ string.
- At each step: search for $L[i\,..\,i+q]$ in the hash table. If not found, the pair $\langle 0, L[i]\rangle$ is emitted (and the algorithm advances by 1 character); otherwise we determine the list $R$ of all the occurrences of $L[i\,..\,i+q]$. For each position $p$ in $R$, compare $L[p\,..\,n]$ and $L[i\,..\,n]$ to determine the $lcp$ string.
- If $p_{lcp}$ is the position at which the $lcp$ string is found, then the pair $\langle i - p_{lcp}, |\,lcp\,|\,\rangle$ is emitted and the algorithm advances by $|\,lcp\,|$ characters. All the $q$-grams starting in $window[1..|\,lcp\,|\,]$ are *deleted* from the hash table, and all the $q$-grams starting in $L[i\,..\,i+|\,lcp\,|\,]$ are *added* to the hash table.
- `gzip` has 9 different compression levels, specified with the options −1, −2, … , −9, and corresponding to $W$ = 100, 200, …, 900 KiB.
- The integer pairs are compressed using Huffman.

# LZ78

## Lempel and Ziv, 1978

- **Idea.** Differently from LZ77, the dictionary $D$ *is explicitly built* during the encoding of $L$ and *not limited* by the window size $W$.
- **Algorithm.**
  - At the beginning: $i = 1$.
  - At each step: determine the longest string of $D$, $lcp$, that is a prefix of $L[i\,..\,n]$.
  - If $index(lcp)$ is the index of $lcp$ in $D$, the algorithm emits the pair $\langle index(lcp), c\rangle$, where $c$ is the next character following $lcp$ in $L[i\,..\,n]$.
  - The concatenation of $lcp$ and $c$, the string $lcp \cdot c$, is added to $D$.
  - Then the algorithm advances by $|lcp| + 1$ characters: $i = i + |lcp| + 1$.
- The stream of integer pairs is compressed using another coding method.

# LZ78 — Encoding Demo

$$L = \underset{0}{|} A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,R\,A$$

$$D$$

$$0 : \varepsilon \quad \text{(empty string)}$$

# LZ78 — Encoding Demo

$$L = \underset{0}{|} A \underset{1}{|} B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,R\,A$$

$$D$$

$$0 : \varepsilon \quad \text{(empty string)}$$

$$1 : A$$

$$\underset{1}{\langle 0, A \rangle}$$

# LZ78 — Encoding Demo

$$L = |A|B|R\ A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ B\ A\ R\ A$$

0 1 2

$$D$$

$$0 : \varepsilon \quad \text{(empty string)}$$

$$1 : A$$

$$2 : B$$

$$\langle 0, A \rangle \langle 0, B \rangle$$

1 2

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,R\,A$$

0 1 2 3

$$D$$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle$$

1 2 3

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,R\,A$$

0 1 2 3 4

$$\langle 0,A\rangle\langle 0,B\rangle\,\langle 0,R\rangle\langle 1,C\rangle$$

1 2 3 4

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D|A\,B\,R\,A\,B\,R\,A\,B\,A\,R\,A$$

0 1 2 3 4 5

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle$$

1 2 3 4 5

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D|A\,B|R\,A\,B\,R\,A\,B\,A\,R\,A$$

0 1 2 3 4 5 6

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle$$

1 2 3 4 5 6

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\ C|A\ D|A\ B|R\ A|B\ R\ A\ B\ A\ R\ A$$

0 1 2 3 4 5 6 7

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle$$

1 2 3 4 5 6 7

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D|A\,B|R\,A|B\,R|A\,B\,A\,R\,A$$

0 1 2 3 4 5 6 7 8

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle$$

1 2 3 4 5 6 7 8

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D|A\,B|R\,A|B\,R|A\,B\,A|R\,A$$

0 1 2 3 4 5 6 7 8 9

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle$$

1 2 3 4 5 6 7 8 9

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

$9 : ABA$

# LZ78 — Encoding Demo

$$L = |A|B|R|A\,C|A\,D|A\,B|R\,A|B\,R|A\,B\,A|R\,A|$$

0 1 2 3 4 5 6 7 8 9 10

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6 7 8 9 10

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

$9 : ABA$

# LZ78 — Decoding Demo

$L =$

| | $D$ |
|---|---|
| $0 : \varepsilon$ | (empty string) |

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

# LZ78 — Decoding Demo

$$L = A$$

$$D$$

$0 : \varepsilon$ (empty string)

$1 : A$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1

# LZ78 — Decoding Demo

$$L = A\,B$$

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2

# LZ78 — Decoding Demo

$$L = A\,B\,R$$

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD\,AB$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD\,AB\,RA$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6 7

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD\,AB\,RA\,BR$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6 7 8

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD\,AB\,RA\,BR\,ABA$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6 7 8 9

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

$9 : ABA$

# LZ78 — Decoding Demo

$$L = A\,B\,R\,AC\,AD\,AB\,RA\,BR\,ABA\,RA$$

$$\langle 0,A\rangle\langle 0,B\rangle\langle 0,R\rangle\langle 1,C\rangle\langle 1,D\rangle\langle 1,B\rangle\langle 3,A\rangle\langle 2,R\rangle\langle 6,A\rangle\langle 7,\text{EOF}\rangle$$

1 2 3 4 5 6 7 8 9 10

$D$

$0 : \varepsilon$ (empty string)

$1 : A$

$2 : B$

$3 : R$

$4 : AC$

$5 : AD$

$6 : AB$

$7 : RA$

$8 : BR$

$9 : ABA$

# LZW

## Welch, 1984

- **Idea.** Try to avoid the second component — the "next" character — of the pairs emitted by LZ78. Since it does not emit the character, the parsing and the building of the dictionary are *misaligned*, which may induce a tricky decoding case (actually the same that can happen in LZ77).

- **Algorithm.**
  - At the beginning: pre-fill the dictionary with all possible characters. If there are $m$ of these, the assigned indexes are $[0..m-1]$, so we set the next available $index$ into $D$ to $index = m$. Set $i = 1$. (For ASCII, all the $m = 256$ characters correspond to the indexes from $0$ to $255$.)
  - At each step: determine the longest string $lcp$ of $D$ that is a prefix of $L[i\,..\,n]$.
  - If $index(lcp)$ is the index of $lcp$ in $D$ and $c$ is the next character following $lcp$ in $L[i\,..\,n]$, the algorithm emits the single integer $index(lcp)$.
  - The string $lcp \cdot c$ is added to $D$ and it takes the next available $index$, that is: $index = index + 1$.
  - Then the algorithm advances by $|lcp|$ characters, that is: $i = i + |lcp|$.

- The stream of integers is compressed using another coding method.

# LZW — Encoding Demo

$$L = \underset{0}{|}A\,B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,B\,A$$

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

# LZW — Encoding Demo

$$L = \underset{0}{|}A\underset{1}{|}B\,R\,A\,C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,B\,A$$

$$\underset{1}{65}$$

$$D$$

$$\ldots$$

$$65 : A \qquad 256 : AB$$

$$66 : B$$

$$67 : C$$

$$68 : D$$

$$\ldots$$

$$82 : R$$

$$\ldots$$

# LZW — Encoding Demo

$$L = |A|B|R\ A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ B\ A\ B\ A$$

0 1 2

65 66

1 2

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

256 : $AB$

257 : $BR$

# LZW — Encoding Demo

$$L = |A|B|R|A\ C\ A\ D\ A\ B\ R\ A\ B\ R\ A\ B\ A\ B\ A$$

0 1 2 3

$$65 \quad 66 \quad 82$$

1 2 3

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

# LZW — Encoding Demo

$$L = |A|B|R|A|C\,A\,D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,B\,A$$

0 1 2 3 4

$$65 \quad 66 \quad 82 \quad 65$$

1 2 3 4

$D$

…

$65 : A$
$66 : B$
$67 : C$
$68 : D$

…

$82 : R$

…

$256 : AB$
$257 : BR$
$258 : RA$
$259 : AC$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A\ D\ A\ B\ R\ A\ B\ R\ A\ B\ A\ B\ A$$

0 1 2 3 4 5

$$65\quad 66\quad 82\quad 65\quad 67$$

1 2 3 4 5

$D$

…

$65 : A$
$66 : B$
$67 : C$
$68 : D$

…

$82 : R$

…

$256 : AB$
$257 : BR$
$258 : RA$
$259 : AC$
$260 : CA$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D\,A\,B\,R\,A\,B\,R\,A\,B\,A\,B\,A$$

0 1 2 3 4 5 6

65 66 82 65 67 65

1 2 3 4 5 6

$D$

…

65 : $A$
66 : $B$
67 : $C$
68 : $D$
…
82 : $R$
…

256 : $AB$
257 : $BR$
258 : $RA$
259 : $AC$
260 : $CA$
261 : $AD$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B\ R\ A\ B\ R\ A\ B\ A\ B\ A$$

0 1 2 3 4 5 6 7

| 65 | 66 | 82 | 65 | 67 | 65 | 68 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B|R\ A\ B\ R\ A\ B\ A\ B\ A$$

$$\begin{array}{cccccccc} 65 & 66 & 82 & 65 & 67 & 65 & 68 & 256 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array}$$

$$\begin{array}{l} D \\ \dots \\ 65 : A \\ 66 : B \\ 67 : C \\ 68 : D \\ \dots \\ 82 : R \\ \dots \end{array} \qquad \begin{array}{l} 256 : AB \\ 257 : BR \\ 258 : RA \\ 259 : AC \\ 260 : CA \\ 261 : AD \\ 262 : DA \\ 263 : ABR \end{array}$$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B|R\ A|B\ R\ A\ B\ A\ B\ A$$

0 1 2 3 4 5 6 7 8 9

65 66 82 65 67 65 68 256 258

1 2 3 4 5 6 7 8 9

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

256 : $AB$

257 : $BR$

258 : $RA$

259 : $AC$

260 : $CA$

261 : $AD$

262 : $DA$

263 : $ABR$

263 : $RAB$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B|R\ A|B\ R|A\ B\ A\ B\ A$$

0 1 2 3 4 5 6 7 8 9 10

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

$263 : RAB$

$264 : BRA$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B|R\ A|B\ R|A\ B|A\ B\ A$$

0 1 2 3 4 5 6 7 8 9 10 11

65 66 82 65 67 65 68 256 258 257 256

1 2 3 4 5 6 7 8 9 10 11

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

256 : $AB$

257 : $BR$

258 : $RA$

259 : $AC$

260 : $CA$

261 : $AD$

262 : $DA$

263 : $ABR$

263 : $RAB$

264 : $BRA$

265 : $ABA$

# LZW — Encoding Demo

$$L = |A|B|R|A|C|A|D|A\ B|R\ A|B\ R|A\ B|A\ B\ A|$$

0 1 2 3 4 5 6 7 8 9 10 11 12

65 66 82 65 67 65 68 256 258 257 256 265

1 2 3 4 5 6 7 8 9 10 11 12

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

256 : $AB$

257 : $BR$

258 : $RA$

259 : $AC$

260 : $CA$

261 : $AD$

262 : $DA$

263 : $ABR$

263 : $RAB$

264 : $BRA$

265 : $ABA$

# LZW — Decoding Demo

$L =$

65 66 82 65 67 65 68 256 258 257 256 265

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

# LZW — Decoding Demo

$L = A$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | |

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

# LZW — Decoding Demo

$$L = A\ B$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | | | | | | | | | | |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

# LZW — Decoding Demo

$$L = A\ B\ R$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | | | | | | | | | |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

# LZW — Decoding Demo

$$L = A\ B\ R\ A$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | | | | | | | | |

$D$

…

65 : $A$
66 : $B$
67 : $C$
68 : $D$

…

82 : $R$

…

256 : $AB$
257 : $BR$
258 : $RA$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | | | | | | | |

$D$

…

$65 : A$
$66 : B$
$67 : C$
$68 : D$

…

$82 : R$

…

$256 : AB$
$257 : BR$
$258 : RA$
$259 : AC$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | | | | | | |

$D$

…

65 : $A$
66 : $B$
67 : $C$
68 : $D$

…

82 : $R$

…

256 : $AB$
257 : $BR$
258 : $RA$
259 : $AC$
260 : $CA$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | | | |

$D$

…

$65 : A$
$66 : B$
$67 : C$
$68 : D$

…

$82 : R$

…

$256 : AB$
$257 : BR$
$258 : RA$
$259 : AC$
$260 : CA$
$261 : AD$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | |

$D$

…

65 : $A$

66 : $B$

67 : $C$

68 : $D$

…

82 : $R$

…

256 : $AB$

257 : $BR$

258 : $RA$

259 : $AC$

260 : $CA$

261 : $AD$

262 : $DA$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA\ BR$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

$263 : RAB$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA\ BR\ AB$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

$263 : RAB$

$264 : BRA$

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA\ BR\ \ AB\ AB\ ?$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

$263 : RAB$

$264 : BRA$

Not in the dictionary yet! →

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA\ BR\ \boxed{AB\ A}B\ ?$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

$D$

…

$65 : A$

$66 : B$

$67 : C$

$68 : D$

…

$82 : R$

…

$256 : AB$

$257 : BR$

$258 : RA$

$259 : AC$

$260 : CA$

$261 : AD$

$262 : DA$

$263 : ABR$

$263 : RAB$

$264 : BRA$

Not in the dictionary yet! →

# LZW — Decoding Demo

$$L = A\ B\ R\ A\ C\ A\ D\ AB\ RA\ BR\ AB\ AB\ A$$

| 65 | 66 | 82 | 65 | 67 | 65 | 68 | 256 | 258 | 257 | 256 | 265 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

$D$

…

$65 : A$
$66 : B$
$67 : C$
$68 : D$

…

$82 : R$

…

$256 : AB$
$257 : BR$
$258 : RA$
$259 : AC$
$260 : CA$
$261 : AD$
$262 : DA$
$263 : ABR$
$263 : RAB$
$264 : BRA$
$265 : ABA$

Not in the dictionary yet! →

# Variants

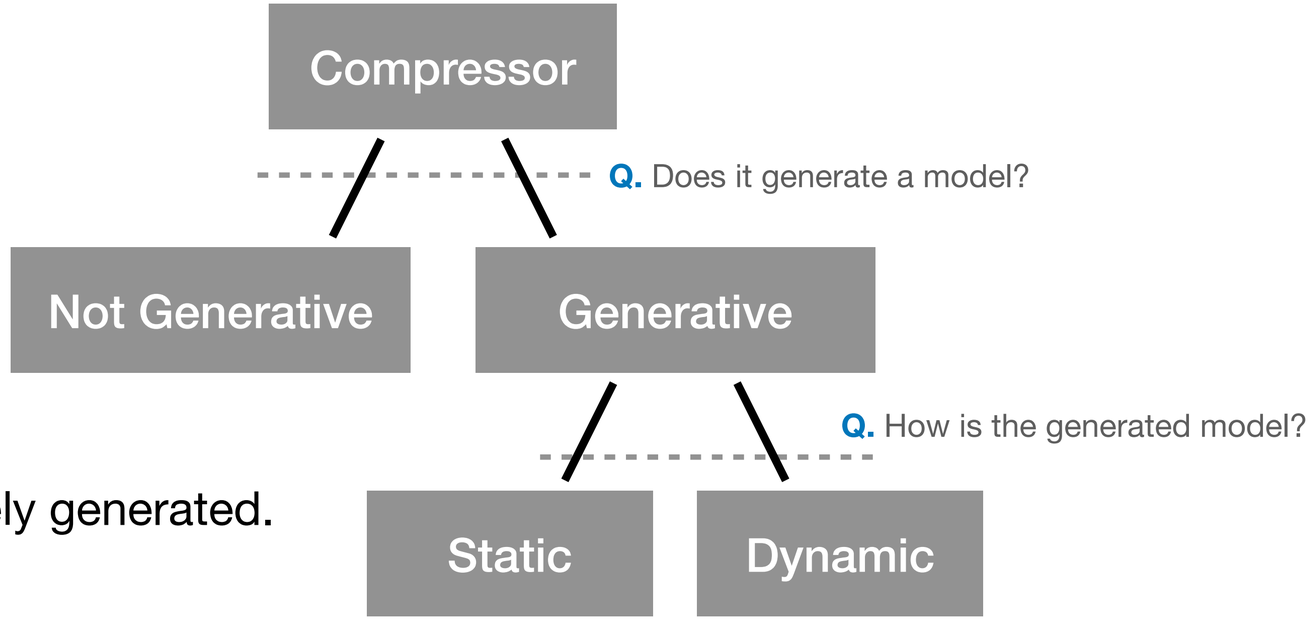
- **BZip.** https://en.wikipedia.org/wiki/Bzip2
- **LZ4.** https://en.wikipedia.org/wiki/LZ4_(compression_algorithm))
- **Zstd.** http://facebook.github.io/zstd
- **LZMA.** https://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm
- **LZO.** https://en.wikipedia.org/wiki/Lempel-Ziv-Oberhumer

# Overview of Compressors

- **Not Generative.** No model generated.
  - Fast to encode/decode.
  - Moderately effective.
  - Example: Delta, Gamma, Golomb, Elias-Fano.
  - [Module 2 and 3]

- **Generative/Static.** The model is not updated.
  - First pass to actually generate the model.
  - Moderately fast to encode/decode.
  - Entropy-optimal.
  - Must transmit the model.
  - Example: Huffman, Arithmetic Coding.
  - [Module 4]

- **Generative/Dynamic.** The model is progressively generated.
  - No need to transmit the model.
  - Decoding is only sequential.
  - Very effective.
  - Example: LZ77, LZ78, LZW.
  - [Module 5]



# Further Readings

- Chapter 5.5 (pages 839-845) of:
  Robert Sedgewick and Kevin Wayne. 2011. *Algorithms*. 4-th Edition.
  Addison-Wesley Professional, ISBN 0-321-57351-X.

- https://ethw.org/History_of_Lossless_Data_Compression_Algorithms

- Jacob Ziv and Abraham Lempel. 1977. *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, IT-23(3):337-343.

- Jacob Ziv and Abraham Lempel. 1978. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory, IT-24(5):530-536.

- Terry A. Welch. 1984. *A technique for high-performance data compression*.
  Computer, pages 8-19.