# Run-length encoding

**Run-length encoding** (**RLE**) is a form of lossless data compression in which *runs* of data (consecutive occurrences of the same data value) are stored as a single occurrence of that data value and a count of its consecutive occurrences, rather than as the original run. As an imaginary example of the concept, when encoding an image built up from colored dots, the sequence "green green green green green green green green green" is shortened to "green x 9". This is most efficient on data that contains many such runs, for example, simple graphic images such as icons, line drawings, games, and animations. For files that do not have many runs, encoding them with RLE could increase the file size.

RLE may also refer in particular to an early graphics file format supported by CompuServe for compressing black and white images, that was widely supplanted by their later Graphics Interchange Format (GIF).

RLE also refers to a little-used image format in Windows 3.x that is saved with the file extension `rle`; it is a run-length encoded bitmap, and was used as the format for the Windows 3.x startup screen.

## History and applications

Run-length encoding (RLE) schemes were employed in the transmission of analog television signals as far back as 1967.[1] In 1983, run-length encoding was patented by Hitachi.[2][3][4] RLE is particularly well suited to palette-based bitmap images (which use relatively few colours) such as computer icons, and was a popular image compression method on early online services such as CompuServe before the advent of more sophisticated formats such as GIF.[5] It does not work well on continuous-tone images (which use very many colours) such as photographs, although JPEG uses it on the coefficients that remain after transforming and quantizing image blocks.

Common formats for run-length encoded data include Truevision TGA, PackBits (by Apple, used in MacPaint), PCX and ILBM. The International Telecommunication Union also describes a standard to encode run-length colour for fax machines, known as T.45.[6] That fax colour coding standard, which along with other techniques is incorporated into Modified Huffman coding, is relatively efficient because most faxed documents are primarily white space, with occasional interruptions of black.

# Algorithm

RLE has a space complexity of $O(n)$, where $n$ is the size of the input data.

## Encoding algorithm

Run-length encoding compresses data by reducing the physical size of a repeating string of characters. This process involves converting the input data into a compressed format by identifying and counting consecutive occurrences of each character. The steps are as follows:

1. Traverse the input data.
2. Count the number of consecutive repeating characters (run length).
3. Store the character and its run length.

### Python implementation

**Imports and helper functions** [show]

```
def rle_encode(iterable, *, length_first=True):
    """
    >>> "".join(rle_encode("AAAABBBCCDAA"))
    '4A3B2C1D2A'
    >>> "".join(rle_encode("AAAABBBCCDAA", length_first=False))
    'A4B3C2D1A2'
    """
    return (
        f"{ilen(g)}{k}" if length_first else f"{k}{ilen(g)}" # 
ilen(g): length of iterable g
        for k, g in groupby(iterable)
    )
```

[7]

## Decoding algorithm

The decoding process involves reconstructing the original data from the encoded format by repeating characters according to their counts. The steps are as follows:

1. Traverse the encoded data.

2. For each count-character pair, repeat the character count times.
3. Append these characters to the result string.

**Python implementation**

**Imports** [show]

```
def rle_decode(iterable, *, length_first=True):
    """
    >>> "".join(rle_decode("4A3B2C1D2A"))
    'AAAABBBCCDAA'
    >>> "".join(rle_decode("A4B3C2D1A2", length_first=False))
    'AAAABBBCCDAA'
    """
    return chain.from_iterable(
        repeat(b, int(a)) if length_first else repeat(a, int(b))
        for a, b in batched(iterable, 2)
    )
```

[7]

# Example

Consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

`WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW`

With a run-length encoding (RLE) data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows:

`12W1B12W3B24W1B14W`

This can be interpreted as a sequence of twelve Ws, one B, twelve Ws, three Bs, etc., and represents the original 67 characters in only 18. While the actual format used for the storage of images is generally binary rather than ASCII characters like this, the principle remains the same. Even binary data files can be compressed with this method; file format specifications often dictate repeated bytes in files as padding space. However, newer compression methods such as DEFLATE often use LZ77-based algorithms, a generalization of run-length encoding that can take advantage of runs of strings of characters (such as `BWWBWWBWWBWW`).

Run-length encoding can be expressed in multiple ways to accommodate data properties as well as additional compression algorithms. For instance, one popular method encodes run lengths for runs of two or more characters only, using an "escape" symbol to identify runs, or using the character itself as the escape, so that any time a character appears twice it denotes a run. On the previous example, this would give the following:

```
WW12BWW12BB3WW24BWW14
```

This would be interpreted as a run of twelve Ws, a B, a run of twelve Ws, a run of three Bs, etc. In data where runs are less frequent, this can significantly improve the compression rate.

One other matter is the application of additional compression algorithms. Even with the runs extracted, the frequencies of different characters may be large, allowing for further compression; however, if the run lengths are written in the file in the locations where the runs occurred, the presence of these numbers interrupts the normal flow and makes it harder to compress. To overcome this, some run-length encoders separate the data and escape symbols from the run lengths, so that the two can be handled independently. For the example data, this would result in two outputs, the string "`WWBWWBBWWBWW`" and the numbers (`12,12,3,24,14`).

## Variants

- Sequential RLE: This method processes data one line at a time, scanning from left to right. It is commonly employed in image compression. Other variations of this technique include scanning the data vertically, diagonally, or in blocks.
- Lossy RLE: In this variation, some bits are intentionally discarded during compression (often by setting one or two significant bits of each pixel to 0). This leads to higher compression rates while minimally impacting the visual quality of the image.
- Adaptive RLE: Uses different encoding schemes depending on the length of runs to optimize compression ratios. For example, short runs might use a different encoding format than long runs.

## See also

- Kolakoski sequence
- Look-and-say sequence
- Comparison of graphics file formats
- Golomb coding

- Burrows–Wheeler transform
- Recursive indexing
- Run-length limited
- Bitmap index
- Forsyth–Edwards Notation, which uses run-length-encoding for empty spaces in chess positions.
- DEFLATE
- Convolution
- Huffman coding
- Arithmetic coding

## References


1. Robinson, A. H.; Cherry, C. (1967). "Results of a prototype television bandwidth compression scheme". *Proceedings of the IEEE*. **55** (3). IEEE: 356–364. doi:10.1109/PROC.1967.5493 (https://doi.org/10.1109%2FPROC.1967.5493) .
2. "Run Length Encoding Patents" (http://www.ross.net/compression/patents_notes_from_ccfaq.html) . Internet FAQ Consortium. 21 March 1996. Retrieved 14 July 2019.
3. "Method and system for data compression and restoration" (https://patents.google.com/patent/US4586027A) . *Google Patents*. 7 August 1984. Retrieved 14 July 2019.
4. "Data recording method" (https://patents.google.com/patent/JPH0828053B2/en) . *Google Patents*. 8 August 1983. Retrieved 14 July 2019.
5. Dunn, Christopher (1987). "Smile! You're on RLE!" (http://csbruce.com/cbm/transactor/pdfs/trans_v7_i06.pdf) (PDF). *The Transactor*. **7** (6). Transactor Publishing: 16–18. Retrieved 2015-12-06.
6. *Recommendation T.45 (02/00): Run-length colour encoding* (http://www.itu.int/rec/T-REC-T.45) . International Telecommunication Union. 2000. Retrieved 2015-12-06.
7. "more-itertools 10.4.0 documentation" (https://more-itertools.readthedocs.io/en/stable/_modules/more_itertools/more.html#run_length) . August 2024.

## External links

- Run-length encoding implemented in different programming languages (http://rosettacode.org/wiki/Run-length_encoding) (on Rosetta Code)
- Single Header Run-Length Encoding Library (https://gitlab.com/bztsrc/rle) smallest possible implementation (about 20 SLoC) in ANSI C. FOSS, compatible with Truevision TGA, supports 8, 16, 24 and 32 bit elements too.