RLE (Run Length Encoding)

- RLE method consists of the replacement of sequence of repeted symbols, by a single representation of the symbol followed by the number of times the symbol repeats;
- Does not use techniques to reduce the size of representing each symbol as is the case in the Huffman coding or arithmetic coding;
- Does not replace strings with dictionary references as in LZ algorithms.

RLE – Exemplo (1)

Consider the following string:

a	a	a	a	b	b	c	d	e	e	e	e	e	f	g	h	h	h	i	j

dim = 20

according to the RLE technique, is transformed in the following sequence:

a	**4**	b	**2**	c	**1**	d	**1**	e	**5**	f	**1**	g	**1**	h	**3**	i	**1**	j	**1**

dim = 20

RLE – Example (2)

- As can be seen the application of the RLE technique does not lead to a decrease in the size of the string, one of the problems is related with the fact of the occurence of several symbols with just one repetition are "expanded" (= count + symbol)

RLE – Example (3)

- One way to solve the problem of "expand" symbols that occur only once, would be considered the count only after the first occurrence. In the example considered would be:

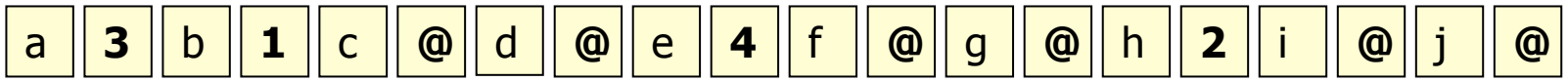
a	**3**	b	**1**	c	d	e	**4**	f	g	h	**2**	i	j

dim = 14

- The problem now is to identify if the next symbol is a count or a symbol. To solve this cou can assign to each symbol a flag indicating whether it is a symbol or a coding.
- Another solution is to put next to each coded symbol a sequence of escape that indicates that next wil be a count (you have to ensure that the sequence of "escape" choosen does not occur in the file to compress).

RLE – Exemplo (4)

- Using an escape sequence (symbol that does not occur in the file to compress, for example '@')



dim = 20

Problems:

- repetitions with size 1 occupy twice the space as codified;
- the '@' symbol can not appear in the text.

RLE – Exemplo (5)

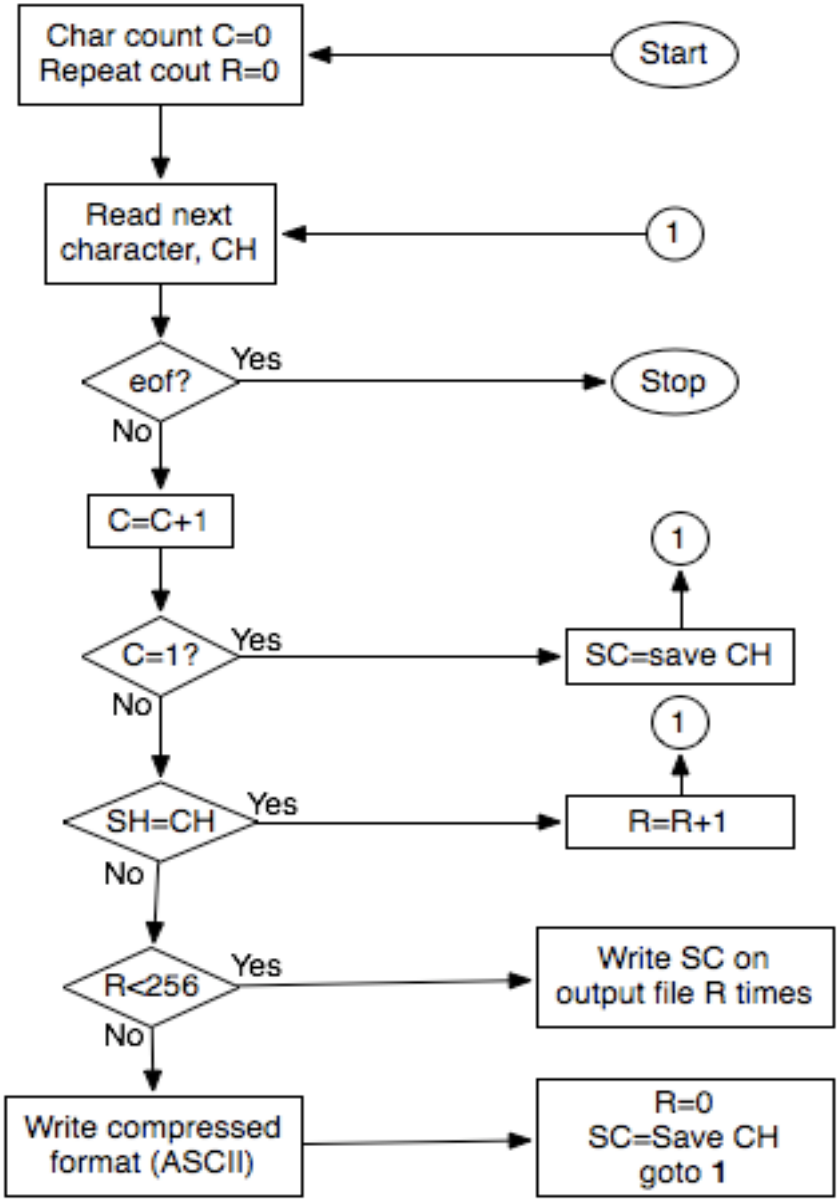
- Another solution: the escape code does not need to have a fixed size, it can be inferred each time a symbol repeats. The value that appears after each repeated symbol indicates the number of additional that follows.

a	a	**2**	b	b	**0**	c	d	e	e	**3**	f	g	h	h	**1**	i	j

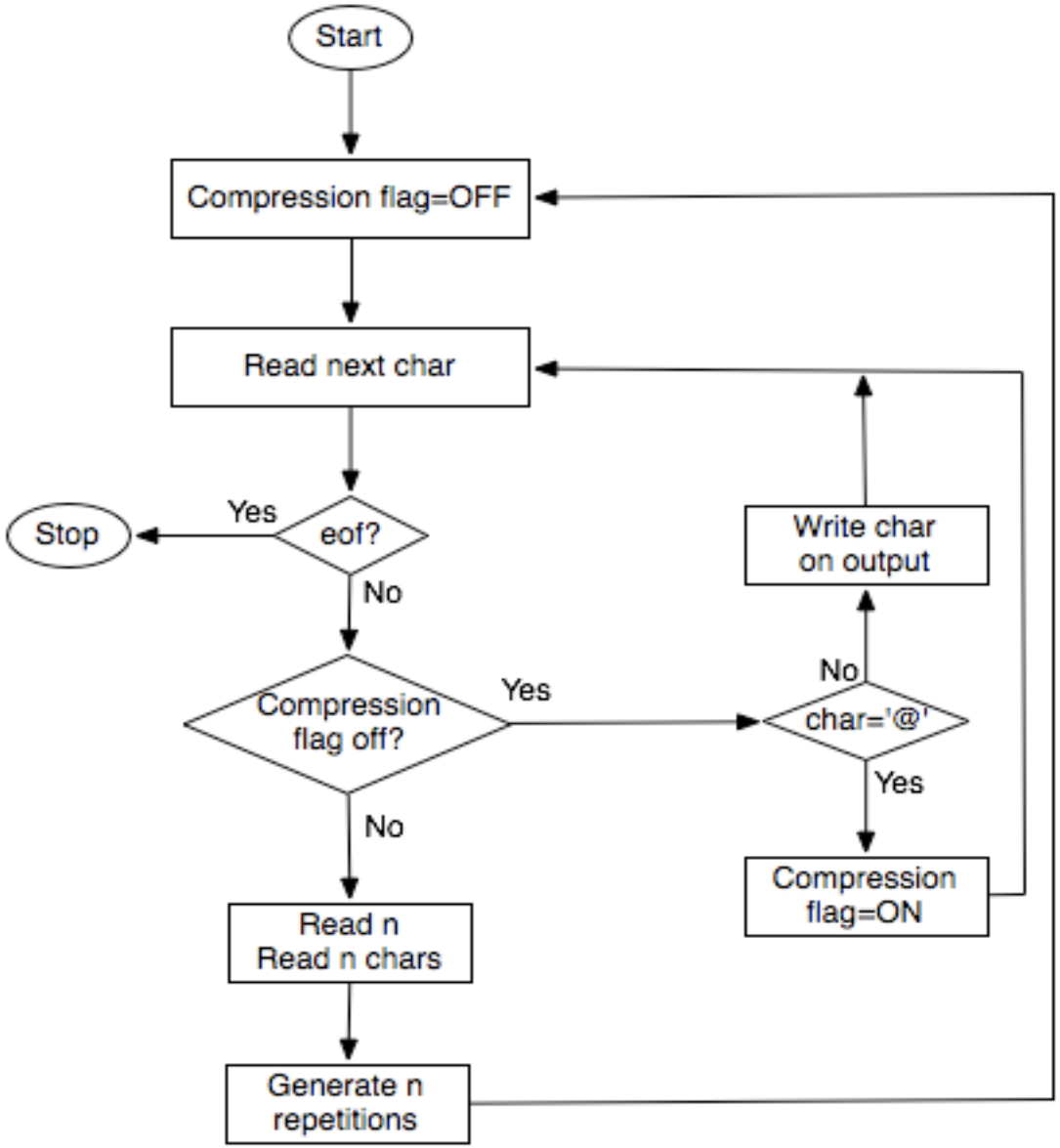
dim = 18

Problem: repetitions with size 2 occupy more space as codified

Compression



Decompression



RLE – Image compression

- At an image given a point (pixel) there is a good possibility that neighboring points have the same color (are equal);
- Takes advantage of of the uniformity of the areas of an image;
- The scan may be carried out line by line.

RLE – Image compression

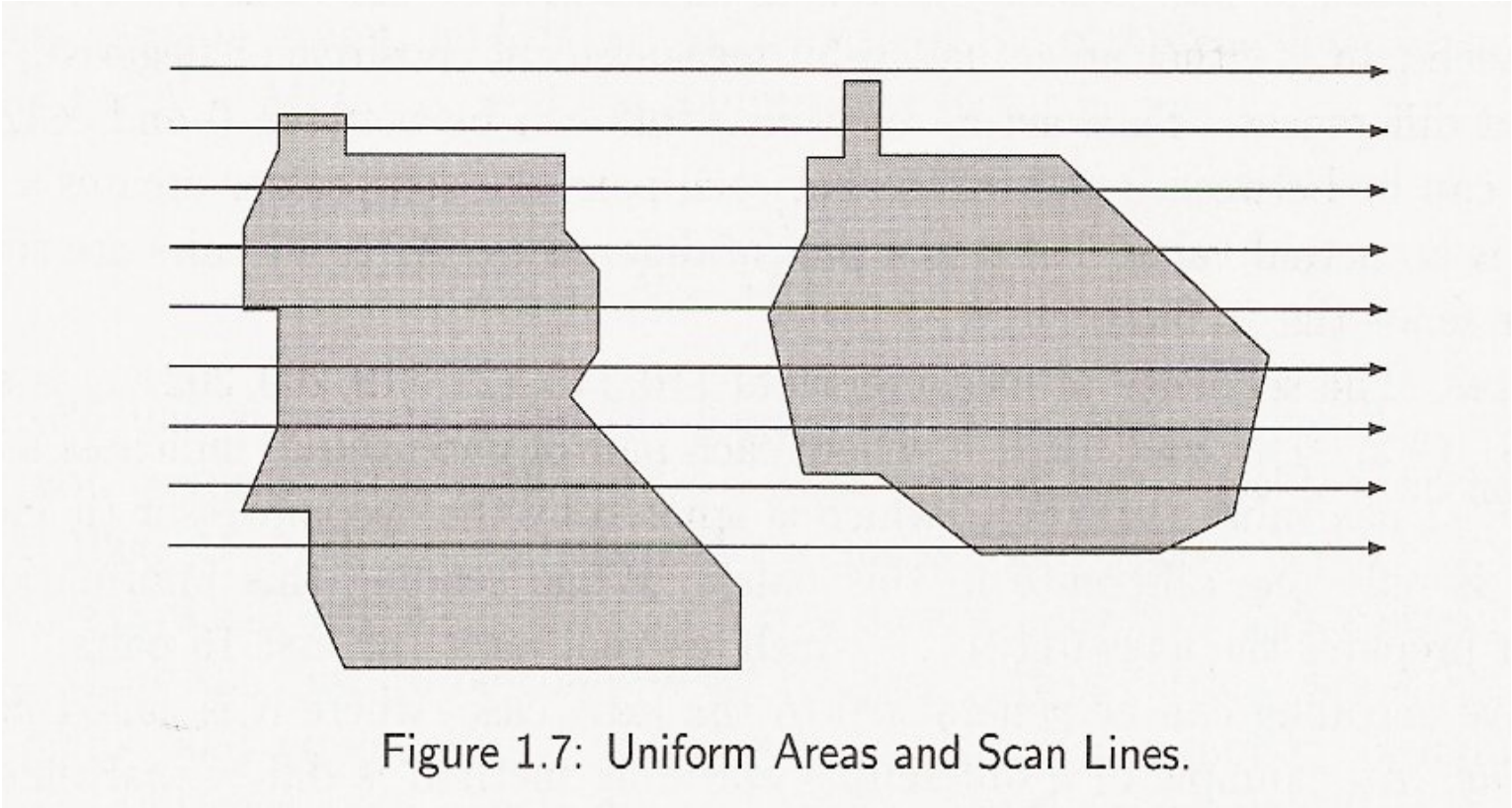


Figure 1.7: Uniform Areas and Scan Lines.

RLE – Image compression

- There may be different ways to scan the image

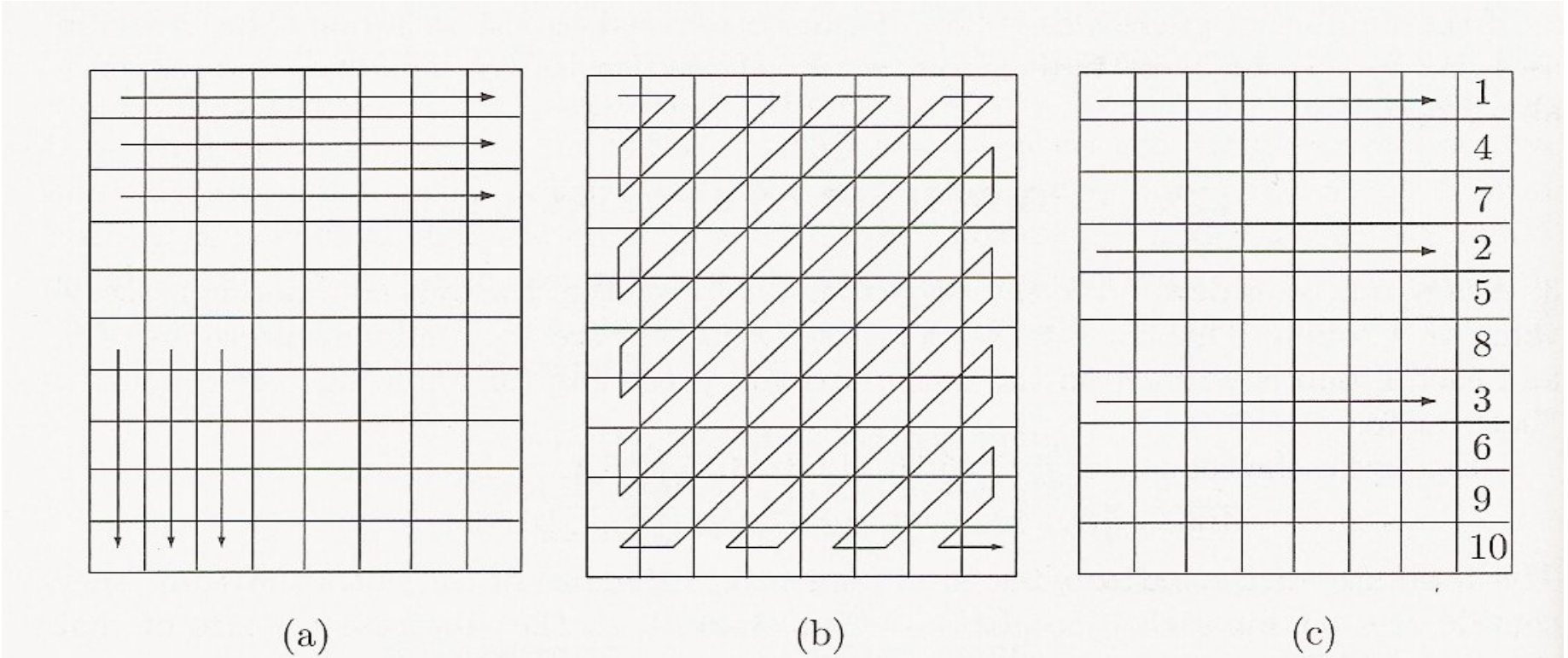


Figure 1.8: RLE Scanning.

RLE – Image compression

- The result of compress an image using an RLE technique may result greater than the original file:
  - Horizontal scan of a graphic image with many vertical lines
  - Photographic Image ?

Lab. work

- Goal:
  - Building a program that, given an image file in PPM format, compress it using an RLE technique (Run Length Encoding).
  - Build a program that recovers a PPM file that was compressed using an RLE technique.

Lab Work – Implementation

- Write two programs *rle_encode* e *rle_decode*. Each one of them receive two files as arguments:
  - *rle_encode*, compress the *f1* file to *f2* file;
  - *rle_decode*, decompress the *f1* file to *f2* file;
- To test your program use the files: interior.ppm, duende.ppm e taz.ppm
  - Note: PPM files can be visualized, for example, with the *xview* program or with the *display* program from the ImageMagick pakage.

Lab Work – Implementation

- The program must carry out an implementation of a compression technique using a simple RLE, where each sequence of **N** identical words **W**, is replaced by a byte **N** followed by the word **W**.
- Example: **AAAbbbbbbbYYYYYYYYYYYYYYYUXXXY!!!** will became: **3A7b13Y1U3X1Y3!**
- Note: When more that 255 consecutive identical words **W** appear, for instance 300 at the output file it should appear **255W45W**

PPM Format (1)

- The PPM format consists of a header:
  - the string 'P6‘ followed by a newline (\n);
  - one or more comment lines, these lines begin with an "#”;
  - 2 ASCII integers (W e H) that represent the width and height of the image followed by a newline;
  - an ASCII integer (between 0 and 65535) representing the maximum values that B1, B2 and B3 can assume, followed by a newline;
- Sequence of pixels:
  - pixels are stored by lines starting at the top left - i.e. x is the index that varies more quickly.

PPM Format(2)

- At the PPM format each pixel is represented by 3 bytes:
  - byte B1: red (R)
  - byte B2: green (G)
  - byte B3: blue (B)

- Example: header file duende.ppm

```
P6
# CREATOR: XV Version 3.10a  Rev: 12/29/94 (PNG patch 1.2)
360 270
255
```

RLE applyed to PPM

- The RLE is not going to be applied in the header of the file. This means, that the `rle_encode` should make a direct “copy” of that area to the output file;
- The RLE should be applied to pixels. The comparison is done between each one of the components of the pixels.

  pixel 1 = pixel 2 if (R1=R2 e G1=G2 e B1=B2)

Adicional references

- “Run Length Encoding (RLE) Discussion and Implementation”, Michael Dipperstein, http://michael.dipperstein.com/rle/index.html
- “RLE compression”, http://www.prepressure.com/techno/compressionrle.htm
- “Interactive Data Compression Tutor”, http://www.eee.bham.ac.uk/WoolleySI/All7/run_1.htm

RLE Compression (ratios)

Files	Original	RLE	Ratio(%)
duende	291674	334434	114.66
interior	291674	362274	124.21
taz	1027769	262170	25.51