May be of interest to understand the principle of the ZIP compression algorithm.
Not needed for the Digital Library exam.

The LZ77 algorithm

https://archive.is/20130107232302/http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz77.html

Terms used in the algorithm

- *Input stream*: the sequence of characters to be compressed;
- *Character*: the basic data element in the input stream;
- *Coding position*: the position of the character in the input stream that is currently being coded (the beginning of the *lookahead buffer*);
- *Lookahead buffer*: the character sequence from the coding position to the end of the input stream;
- The *Window* of size W contains W characters from the coding position backwards, i.e. the last W processed characters;
- A *Pointer* points to the match in the window and also specifies its length.

The principle of encoding
The algorithm searches the window for the longest match with the beginning of the lookahead buffer and outputs a pointer to that match. Since it is possible that not even a one-character match can be found, the output cannot contain just pointers. LZ77 solves this problem this way: after each pointer it outputs the first character in the lookahead buffer after the match. If there is no match, it outputs a null-pointer and the character at the coding position.

The encoding algorithm

1. Set the coding position to the beginning of the input stream;
2. find the longest match in the window for the lookahead buffer;
3. output the pair (P,C) with the following meaning:
    - P is the pointer to the match in the window;
    - C is the first character in the lookahead buffer that didn't match;
4. if the lookahead buffer is not empty, move the coding position (and the window) L+1 characters forward and return to step 2.

An example
The encoding process is presented in Table 1.

- The column **Step** indicates the number of the *encoding step*. It completes each time the encoding algorithm makes an output. With LZ77 this happens in each pass through the step 3.
- The column **Pos** indicates the coding position. The first character in the input stream has the coding position 1.
- The column **Match** shows the longest match found in the window.
- The column **Char** shows the first character in the lookahead buffer after the match.
- The column **Output** presents the output in the format (B,L) C:
    - (B,L) is the pointer (P) to the **Match**. This gives the following instruction to the decoder: "Go back B characters in the window and copy L characters to the output";
    - C is the explicit **Character**.

Input stream for encoding:

Pos	1	2	3	4	5	6	7	8	9
Char	A	A	B	C	B	B	A	B	C

Table 1: The encoding process

Step	Pos	Match	Char	Output
1.	1	--	A	(0,0) A
2.	2	A	B	(1,1) B
3.	4	--	C	(0,0) C
4.	5	B	B	(2,1) B
5.	7	A B	C	(5,2) C

Decoding
The window is maintained the same way as while encoding. In each step the algorithm reads a pair (P,C) from the input. It outputs the sequence from the window specified by P and the character C.

Practical characteristics
The compression ratio this method achieves is very good for many types of data, but the encoding can be quite time-consuming, since there is a lot of comparisons to perform between the lookahead buffer and the window. On the other hand, the decoding is very simple and fast. Memory requirements are low both for the encoding and the decoding. The only structure held in memory is the window, which is usually sized between 4 and 64 kilobytes.

The LZ78 algorithm

https://archive.is/20130107200800/http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz78.html

Terms used in the algorithm

- *Charstream*: a sequence of data to be encoded;
- *Character*: the basic data element in the charstream;
- *Prefix*: a sequence of characters that precede one character;
- *String*: the prefix together with the character it precedes;
- *Code word*: a basic data element in the codestream. It represents a string from the dictionary;
- *Codestream*: the sequence of code words and characters (the output of the encoding algorithm);
- *Dictionary*: a table of strings. Every string is assigned a code word according to its index number in the dictionary;
- *Current prefix*: the prefix currently being processed in the encoding algorithm. Symbol: P;
- *Current character*: a character determined in the endocing algorithm. Generally this is the character preceded by the current prefix. Symbol: C.
- *Current code word*: the code word currently processed in the decoding algorithm. It is signified by W, and the string which it denotes by string.W.

Encoding
At the beginning of encoding the dictionary is empty. In order to explain the principle of encoding, let's consider a point within the encoding process, when the dictionary already contains some strings.

We start analyzing a new prefix in the charstream, beginning with an empty prefix. If its corresponding string (prefix + the character after it -- P+C) is present in the dictionary, the prefix is extended with the character C. This extending is repeated until we get a string which is not present in the dictionary. At that point we output two things to the codestream: the code word that represents the prefix P, and then the character C. Then we add the whole string (P+C) to the dictionary and start processing the next prefix in the charstream.

A special case occurs if the dictionary doesn't contain even the starting one-character string (for example, this always happens in the first encoding step). In that case we output a special code word that represents an empty string, followed by this character and add this character to the dictionary.

The output from this algorithm is a sequence of code word-character pairs (W,C). Each time a pair is output to the codestream, the string from the dictionary corresponding to W is extended with the character C and the resulting string is added to the dictionary. This means that when a new string is being added, the dictionary already contains all the substrings formed by removing characters from the end of the new string.

The encoding algorithm

1. At the start, the dictionary and P are empty;
2. C := next character in the charstream;
3. Is the string P+C present in the dictionary?
    1. if it is, P := P+C (extend P with C);
    2. if not,
        1. output these two objects to the codestream:
            - the code word corresponding to P (if P is empty, output a zero);
            - C, in the same form as input from the charstream;
        2. add the string P+C to the dictionary;
        3. P := empty;
    3. are there more characters in the charstream?
        - if yes, return to step 2;
        - if not:
            1. if P is not empty, output the code word corresponding to P;
            2. END.

Decoding

At the start of decoding the dictionary is empty. It gets reconstructed in the process of decoding. In each step a pair code word-character -- (W,C) is read from the codestream. The code word always refers to a string already present in the dictionary. The string.W and C are output to the charstream and the string (string.W+C) is added to the dictionary. After the decoding, the dictionary will look exactly the same as after the encoding.

The decoding algorithm

4. At the start the dictionary is empty;
5. W := next code word in the codestream;
6. C := the character following it;
7. output the string.W to the codestream (this can be an empty string), and then output C;
8. add the string.W+C to the dictionary;
9. are there more code words in the codestream?
    - if yes, go back to step 2;
    - if not, END.

An example
The encoding process is presented in Table 1.

- The column **Step** indicates the number of the encoding step. Each encoding step is completed when the step 3.b. in the encoding algorithm is executed.
- The column **Pos** indicates the current position in the input data.
- The column **Dictionary** shows what string has been added to the dictionary. The index of the string is equal to the step number.
- The column **Output** presents the output in the form (W,C).
- The output of each step decodes to the string that has been added to the dictionary.

Charstream to be encoded:

Pos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Char	A	B	B	C	B	C	A	B	A

Table 1: The encoding process

Step	**Pos**	**Dictionary**	**Output**
1.	1	A	(0,A)
2.	2	B	(0,B)
3.	3	B C	(2,C)
4.	5	B C A	(3,A)
5.	8	B A	(2,A)

Practical characteristics
The biggest advantage over the LZ77 algorithm is the reduced number of string comparisons in each encoding step. The compression ratio is similar to the LZ77. Since the derived method, LZW, is much more popular, you should see there for further info.

The LZW algorithm

https://archive.is/20130107170646/http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lzw.html

In this algorithm, the same terms are used as in LZ78, with the following addendum:

- A *Root* is a single-character string.

Differences to the LZ78 in the principle of encoding

- Only code words are output. This means that the dictionary cannot be empty at the start: it has to contain all the individual characters (roots) that can occurr in the charstream;
- Since all possible one-character strings are already in the dictionary, each encoding step begins with a one-character prefix, so the first string searched for in the dictionary has two characters;
- The character with which the new prefix starts is the last character of the previous string (C). This is necessary to enable the decoding algorihtm to reconstruct the dictionary without the help of explicit characters in the codestream.

The encoding algorithm

1. At the start, the dictionary contains all possible roots, and P is empty;
2. C := next character in the charstream;
3. Is the string P+C present in the dictionary?
    1. if it is, P := P+C (extend P with C);
    2. if not,
        1. output the code word which denotes P to the codestream;
        2. add the string P+C to the dictionary;
        3. P := C (P now contains only the character C);
    3. are there more characters in the charstream?
        - if yes, go back to step 2;
        - if not:
            1. output the code word which denotes P to the codestream;
            2. END.

Decoding: additional terms

- *Current code word*: the code word currently being processed. It's signified with cW, and the string it denotes with string.cW;
- *Previous code word*: the code word that precedes the current code word in the codestream. It's signified with pW, and the string it denotes with string.pW.

The principle of decoding
At the start of decoding, the dictionary looks the same as at the start of encoding -- it contains all possible roots.

Let's consider a point in the process of decoding, when the dictionary contains some longer strings. The algorithm remembers the previous code word (pW) and then reads the current code word (cW) from the codestream. It outputs the string.cW, and adds the string.pW extended with the first character of the string.cW to the dictionary. This is the character that would have been explicitly read from the codestream in LZ78. Because of this principle, the decoding algorithm "lags" one step behind the encoding algorithm with the adding of new strings to the dictionary.

A special case occurrs if the cW denotes an empty entry in the dictionary. This can happen because of the explained "lagging" behind the encoding algorithm. It happens if the encoding algorithm reads the string that it has just added to the dictionary in the previous step. During the decoding this string is not yet present in the dictionary. A string can occurr twice in a row in the charstream only if its first and last character are equal, because the next string always starts with the last character of the previous one. This leads to the following decoding rule: the string.pW is extended with its own first character and the resulting string is added to the dictionary and output to the charstream.

The decoding algorithm

6. At the start the dictionary contains all possible roots;
7. cW := the first code word in the codestream (it denotes a root);
8. output the string.cW to the charstream;
9. pW := cW;
10. cW := next code word in the codestream;
11. Is the string.cW present in the dictionary?
    - if it is,
        1. output the string.cW to the charstream;
        2. P := string.pW;
        3. C := the first character of the string.cW;
        4. add the string P+C to the dictionary;
    - if not,
        1. P := string.pW;
        2. C := the first character of the string.pW;
        3. output the string P+C to the charstream and add it to the dictionary (now it corresponds to the cW);
12. Are there more code words in the codestream?
    - if yes, go back to step 4;
    - if not, END.

An example
The encoding process is presented in Table 1.

- The column **Step** indicates the number of the encoding step. Each encoding step is completed when the step 3.b. in the encoding algorithm is executed.
- The column **Pos** indicates the current position in the input data.
- The column **Dictionary** shows the string that has been added to the dictionary and its index number in brackets.
- The column **Output** shows the code word output in the corresponding encoding step.

Contents of the dictionary at the beginning of encoding:

(1) A
(2) B
(3) C

Charstream to be encoded:

Pos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Char	A	B	B	A	B	A	B	A	C

Table 1: The encoding process

Step	Pos	Dictionary	Output
1.	1	(4) A B	(1)
2.	2	(5) B B	(2)
3.	3	(6) B A	(2)
4.	4	(7) A B A	(4)
5.	6	(8) A B A C	(7)
6.	--	--	(3)

Table 2. explains the decoding process. In each decoding step the algorithm reads one code word (**Code**), outputs the corresponding string (**Output**) and adds a string to the dictionary (**Dictionary**).

Table 2: The decoding process

Step	Code	Output	Dictionary
1.	(1)	A	--
2.	(2)	B	(4) A B
3.	(2)	B	(5) B B
4.	(4)	A B	(6) B A
5.	(7)	A B A	(7) A B A

6.	(3)	C	(8) A B A C

Let's analyze the step **4.** The previous code word (2) is stored in pW, and cW is (4). The string.cW is output ("A B"). The string.pW ("B") is extended with the first character of the string.cW ("A") and the result ("B A") is added to the dictionary with the index (6).

We come to the step **5.** The content of cW=(4) is copied to pW, and the new value for cW is read: (7). This entry in the dictionary is empty. Thus, the string.pW ("A B") is extended with its own first character ("A") and the result ("A B A") is stored in the dictionary with the index (7). Since cW is (7) as well, this string is also sent to the output.

Practical characteristics
This method is very popular in practice. Its advantage over the LZ77-based algorithms is in the speed because there are not that many string comparisons to perform. Further refinements add variable code word size (depending on the current dictionary size), deleting of the old strings in the dictionary etc. For example, these refinements are used in the GIF image format and in the UNIX `compress` utility for general compression.
Another interesting variation is the LZMW algorithm. It forms a new entry in the dictionary by concatenating the two previous ones. This enables a faster buildup of longer strings.

The LZW method is patented -- the owner of the patent is the Unisys company. It allows free use of the method, except for the producers of commercial software.