INTERNATIONAL STANDARD

ISO/IEC 12042

First edition
1993-12-15

Information technology — Data compression for information interchange — Binary arithmetic coding algorithm

Technologies de l'information — Compression de données pour l'échange d'information — Algorithme de codage arithmétique binaire

Reference number
ISO/IEC 12042:1993(E)

Contents

		Page
1	Scope	1
2	Normative references	1
3	Conformance	1
4	Conventions and notations	1
5	Algorithm identifier	1
6	Definitions	1
6.1	block	2
6.2	Code Block	2
6.3	Code String	2
6.4	encoding	2
6.5	input event	2
6.6	Logical Data Record	2
6.7	trailer	2
6.8	Unique Table Pair	2
7	List of acronyms	2
8	Compression algorithm	2
8.1	General	2
8.2	Encoders	2
8.3	Formation of a Code Block	2
8.4	Code String	3
8.5	Table Pairs	3
8.6	Encoding	3

8.6.1 Normal Mode **4**
8.6.2 Run Mode **5**

8.7 Completion of the encoding of a block **6**

Annex

A Example of a binary arithmetic coding algorithm **10**

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 12042 was prepared by the European Computer Manufacturers Association (as Standard ECMA-159) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology,* in parallel with its approval by national bodies of ISO and IEC.

Annex A of this International Standard is for information only.

Introduction

In the past decades numerous International Standards for magnetic tapes, magnetic tape cassettes and cartridges, as well as for optical disk cartridges have been published. Media developed recently have a very high physical recording density. In order to make an optimal use of the resulting data capacity, compression algorithms have been designed which allow a reduction of the number of bits required for the representation of user data in coded form.

These compression algorithms are registered by an International Registration Authority set up by ISO/IEC. The registration will consist in allocating to each registered algorithm a numerical identifier which will be recorded on the medium and, thus, indicate which compression algorithm(s) has been used.

The first International Standard for compression algorithms was:

ISO/IEC 11558, *Information technology -Data Compression for Information Interchange - Adaptive Coding with Embedded Dictionary - DCLZ Algorithm*

This International Standard is the next one of this series.

This page intentionally left blank

Information technology — Data compression for information interchange — Binary arithmetic coding algorithm

1 Scope

This International Standard specifies an algorithm for the reduction of the number of bits required to represent information. This process is known as data compression. The algorithm uses binary arithmetic coding. The algorithm provides lossless compression and is intended for use in information interchange.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 11576:1993, Information technology - Procedure for the registration of algorithms for the lossless compression of data.

International Register of Algorithms for Lossless Compression of Data, in accordance with ISO/IEC 11576.

3 Conformance

A compression algorithm shall be in conformance with this International Standard if it satisfies all mandatory requirements of this International Standard.

4 Conventions and notations

The following conventions and notations apply in this International Standard unless otherwise stated:

- In each field the bytes shall be arranged with Byte 1, the most significant, first. Within each byte the bits shall be arranged with Bit 1, the most significant bit, first and Bit 8, the least significant bit, last.
- Letters and digits in parentheses represent numbers in hexadecimal notation.
- The setting of bits is denoted by ZERO or ONE.
- Numbers in binary notation and bit combinations are represented by strings of ZEROs and ONEs.
- Numbers in binary notation and bit combinations are shown with the most significant bit to the left.

5 Algorithm identifier

The numeric identifier of this algorithm in the International Register shall be 16.

6 Definitions

For the purposes of this International Standard, the following definitions apply.

6.1 block: A portion of the Logical Data Record, usually having a length of 512 bytes (see 8.2).

6.2 Code Block: A block after compression with a trailer appended.

6.3 Code String: The encoded Logical Data Record.

6.4 encoding: The process of generating Code Blocks from blocks.

6.5 input event: The sample of the input to an encoder currently being examined; in Run Mode it is a byte; in Normal Mode it is a bit.

6.6 Logical Data Record: The data entity that is the input to the data compressor.

6.7 trailer: data appended to a block after compression and addition of pad bits.

6.8 Unique Table Pair: The last of the 256 Table Pairs, used only in Run Mode.

7 List of acronyms

CV	Current Value
EV	Estimated Value
LDR	Logical Data Record
TP	Table Pair

8 Compression algorithm

8.1 General

The LDR is transformed to a Code String by a one-pass, adaptive encoding technique designed to provide lossless data compression. By the use of a suitable decoding technique the exact original LDR can be recovered from the Code String.

8.2 Encoders

The LDR shall be divided into 512-byte blocks, except for the last block, which may be of any length less than, or equal to, 512 bytes. The blocks shall be routed sequentially to eight encoders, numbered from 0 to 7, commencing with encoder 0. If the LDR contains more than 4 096 bytes the compressor shall return to encoder 0 and repeat the process (see figure 2).

8.3 Formation of a Code Block

The output of each encoder is a Code Block (see figure 1).

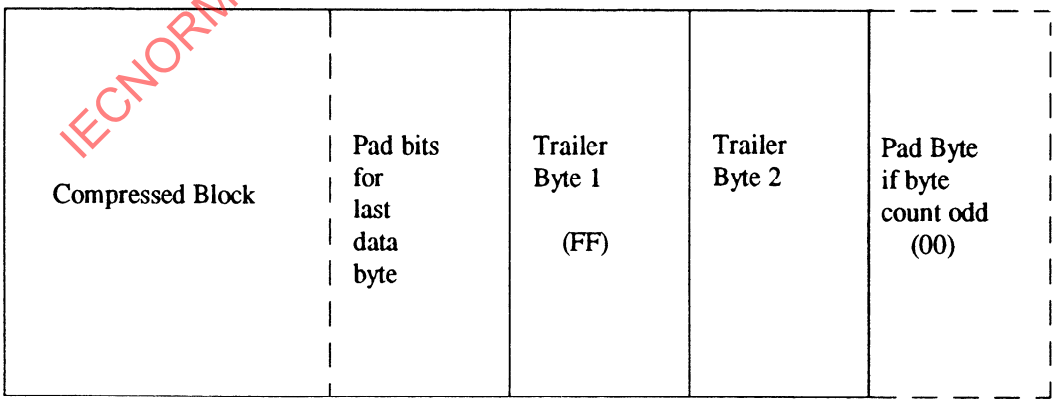

Figure 1 - Code Block

Since the degree of compression achieved in an encoder depends upon the relative frequency of the bit patterns in the LDR, and upon the presence of sequences of identical bytes, the length of the Compressed Block cannot be predicted. Pad bits set to ZERO shall be added at the end to form an integral number of 8-bit bytes.

The Code Block shall be completed by appending a trailer. The trailer shall consist of two Trailer Bytes possibly followed by a Pad Byte.

Trailer Byte 1 shall be set to (FF).

Trailer Byte 2

Bits 1 to 4 shall be set to 1100 if the Code Block has been generated from the last block of the LDR.
shall be 1001 for all other Code Blocks.

Bit 5 shall be set to ZERO if the number of bytes after encoding is even.
shall be set to ONE if the number of bytes after encoding is odd.

Bits 6 to 8 shall specify the number of pad bits that have been added to form an integral number of bytes.

If the number of bytes in the Compressed Block plus the pad bits, is odd, a Pad Byte set to (00) shall be appended after Trailer Byte 2 to give an even number of bytes.

8.4 Code String

The Code String shall be assembled from the outputs of the encoders, with the first portion being that generated by encoder 0, the second that generated by encoder 1, and so on (see figure 2).

8.5 Table Pairs

Each encoder shall be allocated a table of 256 pairs of numbers, numbered from 1 to 256. The first number of each Table Pair shall be the estimated value (EV) of the Input Event to be encoded; it shall be 1 or 0. The second number (*K*) shall be a measure of the probability of the Input Event being equal to the EV. *K* shall have the value 1, 2, 3 or 4, with the probability shown in table 1.

Table 1- Probability values of *K*

K	Probability
1	1 - 2
2	2 - 4
3	4 - 8
4	8 - 16

The probabilities shall be a measure of how much more likely it is that the value of the Input Event is equal to the EV rather than being unequal (e.g. for *K*=2 the probability that the Input Event is equal to the EV shall be 2 to 4 times as great as the probability that it would not be equal).

Before commencing the encoding of the LDR all EVs shall be set to ZERO and all values of *K* shall be set to ONE.

8.6 Encoding

The data shall first be examined on a byte basis. Bytes shall be fetched sequentially from the block, starting with the first byte, and compared with the previous byte. The first byte in a block shall be compared with (40).

Run Mode (see 8.6.2) shall be disabled when the first byte is fetched.

If the current byte differs from the previous byte and Run Mode is not enabled, then the byte shall be encoded, bit by bit, in Normal Mode (see 8.6.1).

If the current byte differs from the previous byte and Run Mode is enabled, then encoding shall proceed as defined in 8.6.2.2.

If the two bytes are identical and Run Mode is not enabled, then Run Mode shall be enabled and the byte shall be encoded, bit by bit, in Normal Mode.

If the two bytes are identical and Run Mode is enabled, then encoding shall proceed as defined in 8.6.2.1.

8.6.1 Normal Mode

The first (most significant) bit of the byte shall be compared with the EV in the first Table Pair. Depending upon the result of this comparison, one of two actions, which are described in 8.6.1.1, shall result. The choice of which Table Pair to select for the remaining bits of the byte shall be determined by the bits previously encoded in the byte.

The first bit of each byte shall always use the first Table Pair.

The second bit shall use the second or third Table Pair, depending upon whether the first bit was ZERO or ONE, respectively.

The third bit shall use one of the next four Table Pairs depending upon the first two bits. The fourth Table Pair shall be used if the first two bits were 00, and so on (see figure 3).

This procedure requires the first 255 Table Pairs. The remaining Table Pair is the Unique Table Pair; it shall be used in the Run Mode only (see 8.6.2).

The process of encoding is then performed in two stages:

- The bit is encoded as in 8.6.1.1.
- The values of K and EV are revised as described in 8.6.1.2.

8.6.1.1 Bit encoding

Two values shall be developed during the bit comparison portion of the compression process. Both are fractional binary numbers to four binary places. One value, termed the Current Value (CV), shall be used to generate the output (Code Block).

The second value, termed the Width, shall have values in the range 0,0000 to 1,1111.

For each block the CV shall be initialized to 0,0000 and the Width value shall be initialized to 1,0000.

Each Input Event shall cause the two values to be modified according to the following rules:

i) The Input Event is equal to the EV

The CV shall be increased by 2^{-K}

The Width shall be decreased by 2^{-K}

where K is the measure of the probability of the Input Event (see 8.5).

If the Code Block is not null the bit to the left of the point in the CV shall be logically added to the last bit appended to the Code Block and replaced in the CV by a ZERO. If this addition causes the most recently generated complete byte in the Code Block to become (FF), then (0) shall be appended to the Code Block at the end of that byte to prevent a carry from propagating beyond the last complete byte.

If the Code Block is null the bit to the left of the point in the CV will already be a ZERO.

The Width shall then be compared with 1,0000.

If it is equal to, or greater than, 1,0000, then the Table Pair shall be revised (see 8.6.1.2) and the next bit fetched for encoding.

If it is less than 1,0000, then all the bits of the Width shall be shifted left one position; the leftmost bit, set to ZERO, shall be dropped, and the rightmost position shall be filled with a ZERO. The bit in the first position to the right of the point in the CV shall be appended to the encoder output as part of the Code Block. The remaining bits to the right of the point in the CV shall be shifted left one position and the rightmost position shall be filled with a ZERO.

To prevent a carry from propagating beyond the last complete byte, each byte in the Code Block shall be examined as it is completed. If it equals (FF) then (0) shall be appended to the Code Block.

ii) The Input Event is not equal to the EV.

The Width shall be reset to 1,0000.

The *K* bits to the right of the point in the CV shall be shifted out and appended to the Code Block, the leftmost bit first. As each bit is appended to the Code Block a check shall be made to determine whether a byte boundary has been reached. If it has, then the byte just completed shall be checked; if it equals (FF) then (0) shall be appended to the Code Block.

The rightmost *K* bit positions of the CV shall be set to ZEROs.

8.6.1.2 Revision of *K* and EV (see figure 4)

As each Input Event is compared with the EV the values of *K* and the EV for that Table Pair shall be amended.

A four-stage binary counter (Mc counter) shall be set to 0000 at the beginning of each block and incremented as described below.

i) The Input Event is equal to the EV

K shall be incremented according to table 2, where bits marked X shall be ignored.

Table 2 - Rules for incrementing *K*

Current value of *K*	State of Counter	New value of *K*
1	XX11	2
2	X111	3
3	1111	4
4	XXXX	4

For all other states of the counter *K* shall be unchanged.

The counter shall then be incremented by 1. If the counter value was 1111 incrementing by 1 shall result in a counter value of 0000.

The EV shall be unchanged.

ii) The Input Event is not equal to the EV

For *K* greater than 1 : The value of *K* shall be decremented by 1

the counter shall not be incremented

the EV shall be unchanged

For *K* equal to 1 : the value of *K* shall be unchanged

the counter shall not be incremented

the EV shall be inverted

8.6.2 Run Mode

Run Mode shall have been enabled if the last two bytes to be encoded were identical.

8.6.2.1 If the current byte is identical with these two bytes, then the Unique Table Pair is selected and the EV is compared with ONE; the values of the Width, CV, *K* and EV are amended, as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended.

8.6.2.2 If the current byte differs from these two bytes then the Unique Table Pair is selected and the EV is compared with ZERO; the values of the Width, CV, *K* and EV are amended, as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended.

The byte is then encoded in Normal Mode and the values of the Width, CV, *K* and EV are amended as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended. Run Mode shall then be disabled.

8.7 Completion of the encoding of a block

If, when the final byte of the block has been encoded and Run Mode is enabled, action shall be taken as defined in 8.6.2.2, except that no byte remains to be encoded in Normal Mode.

When this action has been taken, or if Run Mode was not enabled, the encoder shall be cleared as follows:

The four bits to the right of the point in the CV shall be appended to the Code Block, starting with the leftmost bit. As each bit is appended to the Code Block a check shall be made to determine whether a byte boundary has been reached. If it has, then the byte just completed shall be checked; if it equals (FF) then (0) shall be appended to the Code Block. Any remaining bits from the CV shall be appended to the Code Block. Pad bits and the trailer shall then be appended to the Code Block as described in 8.3.

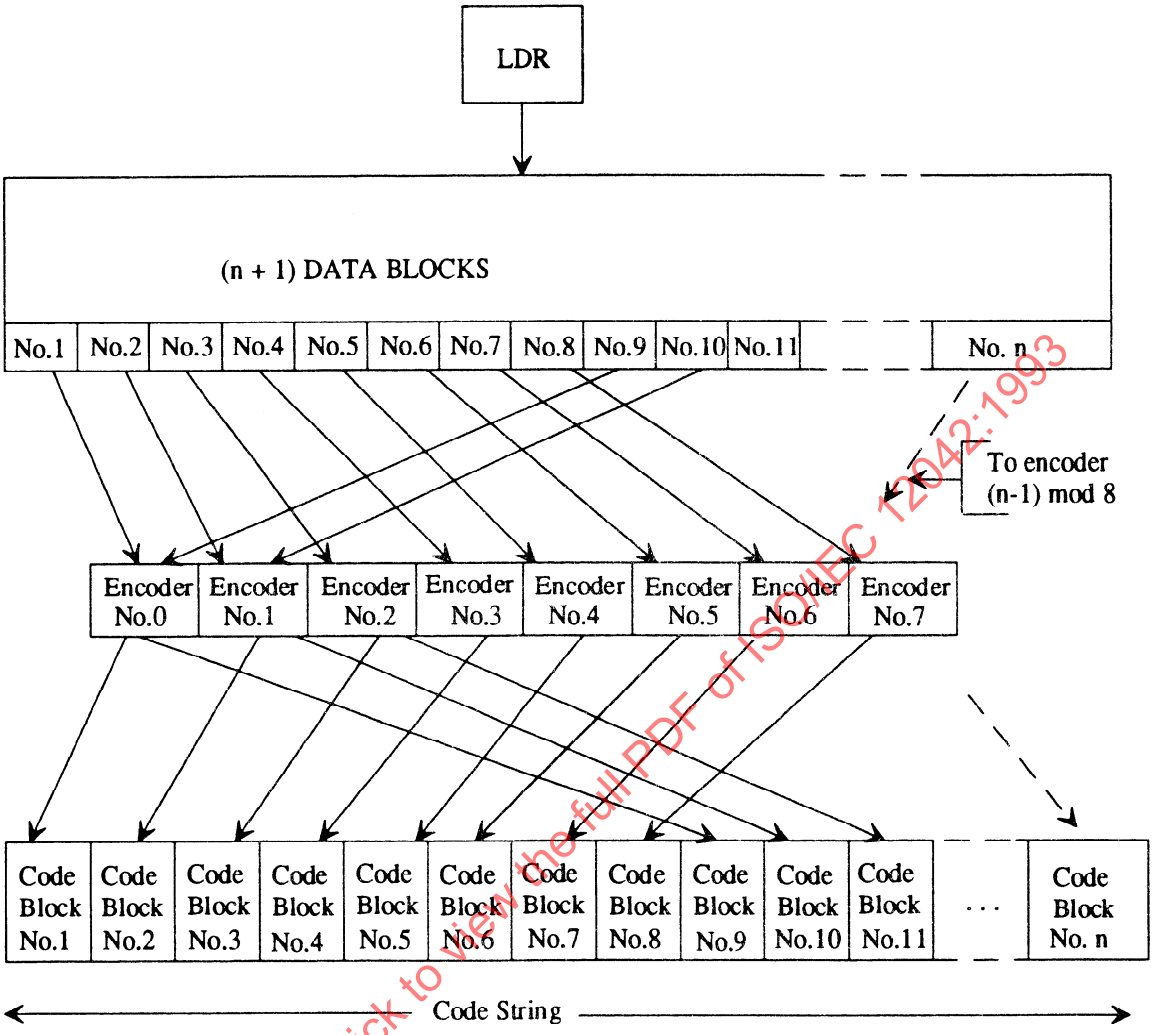

Figure 2 - Sequence of encoding

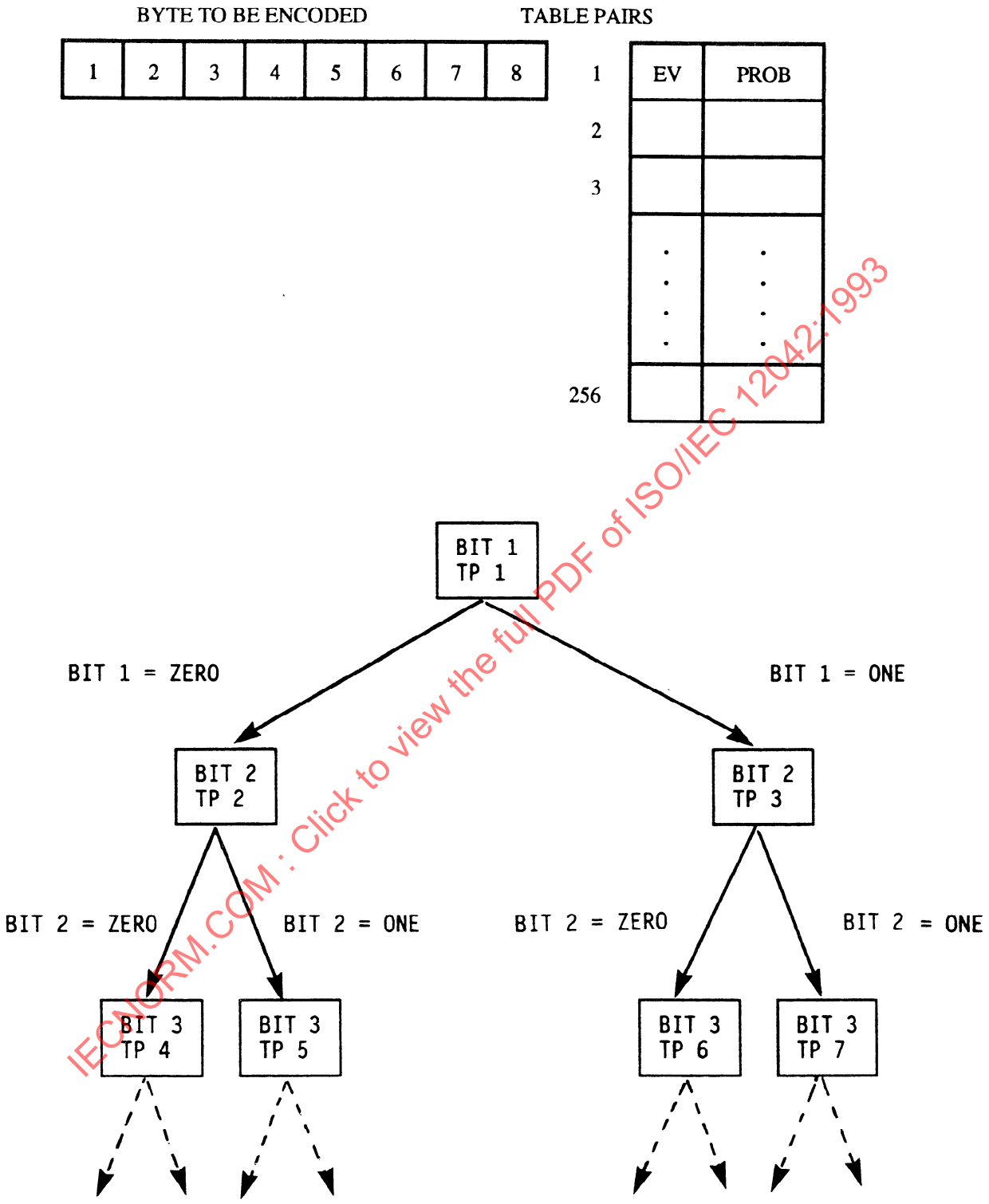

Figure 3 - Choice of Table Pairs

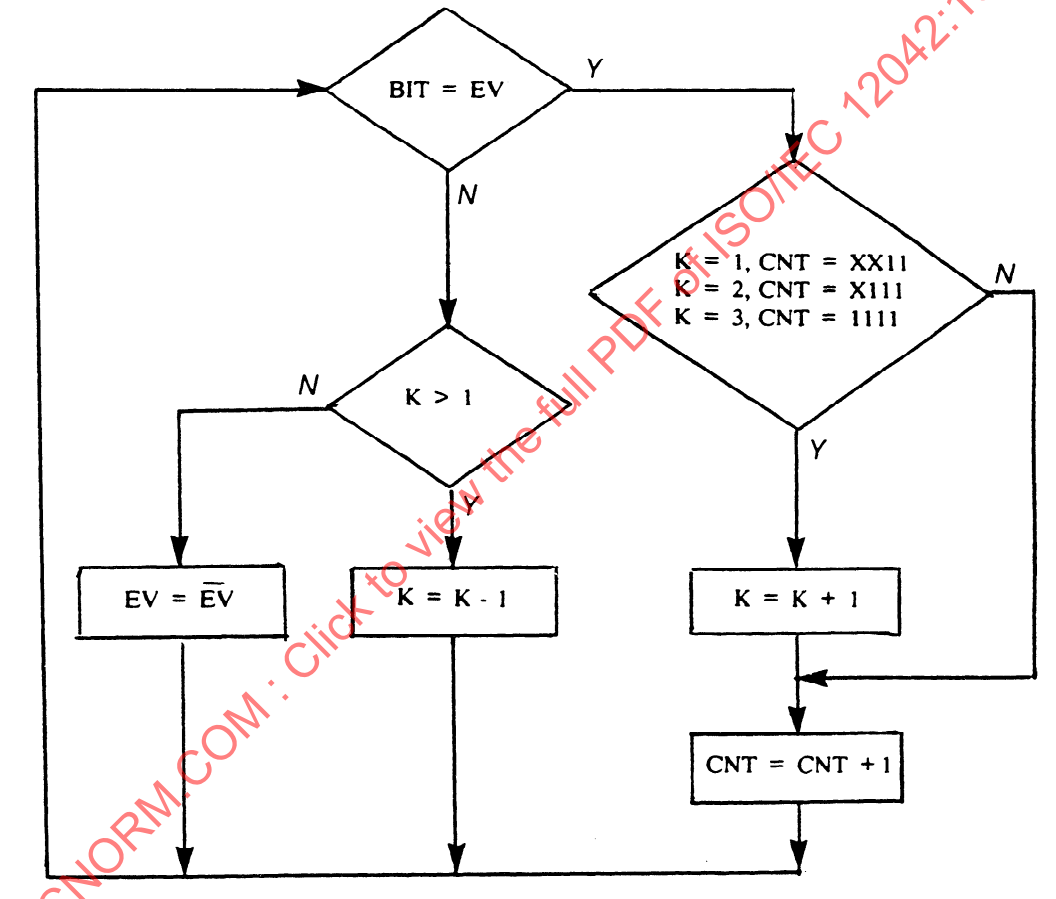

Figure 4 - Determination of *K*

Annex A
(informative)

Example of a binary arithmetic coding algorithm

A.1 Introduction

The following is a description of a binary arithmetic coding algorithm, expressed in structured English, also known as pseudo code. It is intended to give an overall understanding of the algorithm; it is not intended to be a complete and accurate description of the particular implementation described in this International Standard.

The compression of a data record requires several successive processes. This example defines nine separate processes, which are linked together by Calls, that is, when a particular process requires the action of another process to be completed, the required process is called from within the calling process. These Calls are denoted by the called process being written in upper case.

The pseudo code structure uses basic operators such as IF, THEN, ELSE, SET, and DO WHILE. The only exception is the CASE statement which is used to replace several IF, THEN, ELSE statements. Indentation is used to clarify which operators are related. For instance, the IF, THEN, and ELSE operators of a given IF operation are all indented by the same distance.

A.2 Description of the processes

A.2.1 Compact

Compact divides the Logical Data Record into blocks of 512 bytes or less, and directs them sequentially to one of eight encoders. The blocks are encoded and then concatenated to form the output, called the Code String.

A.2.2 Block encode

This process covers the actual encoding of the individual blocks. The Width and Current Value are introduced. The Current Value is used to generate the actual encoded output whilst the Width controls the level of compression achieved. Compression is achieved by deciding, based on the previous content of the Logical Data Record, what is the most probable state of the next bit in the byte. If this estimate is correct and the probability of the estimate being correct is high, the Width changes very little and one bit, or no bit is added to the Code Block. If the estimate is incorrect, then one to four bits are added to the Code Block.

Obviously, the more repetitive the data is, the more likely it is that the actual data will match the estimate and the greater will be the compression achieved.

Another mode of operation is also defined in this process. If the current byte is equal to the previous two bytes, then the whole byte is handled at once in Run Mode. The Unique Table Pair is used and the estimate is compared with 1. Width and Current Values are modified in the same way as they are for the comparison of individual bits.

A.2.3 Compute

Compute determines how the Width and Current Value are changed when the bit being examined matches the Estimated Value.

A.2.4 Bit encode

This details how the individual bits within a byte of a block are processed, and the order in which they are handled.

A.2.5 Check

This process is used to move data from the Current Value to the Code Block. There are two ways of doing this; the first is to take the value to the left of the point in the Current Value and logically add it to the Code Block. Since this may cause a carry, a mechanism has been added to limit how much of the Code Block can be modified by the carry. The most recently formed complete byte of the Code Block is examined; if it is equal to (FF), four ZERO bits are appended after this byte. Any further carries caused by logically adding the value to the left of the point in the Current Value will thus be prevented from propagating further by the (0).

The second way to move data is to append one or more bits from the right of the point in the Current Value to the Code Block. The number of bits to add is determined by whether the bit in question matches the estimate and, if not, what was the probability that the two would match. Again, as bits are appended to the Code Block, if an (FF) byte is formed, (0) is appended to prevent any subsequent carries from propagating more than one byte into the Code Block.

A.2.6 Table Pairs update

The manner in which the estimates and probabilities are updated is defined. The CASE statement can be read as a cascading IF statement; that is, each set of conditions is checked until a match is found or until the end of the CASE statement is reached. If a match is found, the probability is set as described for that set of conditions.

A.2.7 Next Table Pair

This process determines which Table Pair will be used to evaluate the next bit in the byte. The first Table Pair is always used for the first bit of each byte. The second or third Table Pair is used for the second bit, depending upon whether the first bit was a ZERO or a ONE, respectively, In a similar fashion, the proper Table Pair to be used for each bit is determined, based on all the previously processed bits of the byte.

A.2.8 Clear encoder

This terminates the encoding when the complete block has been processed. If Run Mode is enabled, then the estimate in the Unique Table Pair is used to close out Run Mode.

If Run Mode is not enabled, or once Run Mode has been closed out, then the four bits to the right of the point in the Current Value are appended to the Code Block, again checking for an (FF) byte. ZEROs are appended to the Code Block to reach a byte boundary.

A.2.9 Trailer

This final process adds the trailer, which is composed of an (FF) and an information Trailer Byte, and, if necessary, an all ZEROs Pad Byte to make the total number of bytes in the completed Code Block an even number.

A.3 Description of the process in Pseudo Code

A.3.1 COMPACT - Compact describes the overall process of breaking the Logical Data Record into blocks and encoding them in the appropriate encoder.

PROCESS = COMPACT

RECEIVE Logical_Data_Record

RESET Code_String

SET Encoder_Number = 0

SET 256 Table_Pairs (1:256) to {0,1} in each of 8 Tables (0:7)

IF Logical_Data_Record > 512 bytes

THEN **ROUTE** first 512 byte block from the Logical_Data_Record to Encoder 0 and BLOCK_ENCODE (see A.3.2) using Table 0. The Code_Block (output of Encoder) is the first portion of the Code_String (compressor output).

DO WHILE remainder of Logical_Data_Record is > 512 bytes

SET Encoder_Number = Encoder_Number + 1

IF Encoder_Number is ≤ 7

THEN ROUTE next 512-byte block to Encoder (Encoder_Number) and BLOCK_ENCODE using Table (Encoder_Number).

ELSE **SET** Encoder_Number = 0

ROUTE next 512-byte block to Encoder 0 and BLOCK_ENCODE using Table 0.

APPEND Code_Block to Code_String.

END DO

SET Encoder_Number = Encoder_Number + 1

IF Encoder_Number is ≤ 7

THEN **ROUTE** remainder of Logical_Data_Record to Encoder (Encoder_Number) and BLOCK_ENCODE using Table (Encoder_Number).

ELSE **SET** Encoder_Number = 0

ROUTE remainder of Logical_Data_Record to Encoder 0 and BLOCK_ENCODE using Table 0.

APPEND Code_Block to Code_String.

ELSE **ROUTE** Logical_Data_Record to Encoder 0 and BLOCK_ENCODE using Table 0. Code_Block (output of Encoder) is the Code_String (compressor output).

END PROCESS=COMPACT

A.3.2 BLOCK_ENCODE - Encoding blocks is the process of converting blocks of 512 bytes or less into Code_Blocks. Code_Blocks are terminated with trailers.

```
PROCESS   = BLOCK_ENCODE
SET       Width = 1,0000
SET       Current_Value (CV) = 0,0000
SET       Previous_Byte = (40)
SET       Run_Mode = 0
SET       Mc_Count = 0000
SET       Block_Bytes_Remaining = number of bytes in block
SET       Byte_Number = 1
DO WHILE  Block_Bytes_Remaining > 0
          SET       Current_Byte = byte (Byte_Number) of block
          SET       Byte_Number = Byte_Number + 1
          SET       Block_Bytes_Remaining = Block_Bytes_Remaining - 1
          IF        Current_Byte = Previous_Byte
          THEN      IF   Run_Mode = 1,
                    THEN      IF        first value of Unique_Table_Pair = 1
                              THEN      COMPUTE (see A.3.3)
                              ELSE      SET       Width = 1,0000
                                        SET       Compare = false
                                        SET       Shift_Left = second value in Unique_Table_Pair
                                        CHECK
                                        TABLE_PAIRS_UPDATE
                    ELSE      BIT_ENCODE (see A.3.4)
                              SET       Run_Mode = 1
          ELSE      SET       Previous_Byte = Current_Byte
                    IF        Run_Mode = 1
                    THEN IF first value of Unique_Table_Pair = 0
                              THEN      COMPUTE
                              ELSE      SET       Width = 1,0000
                                        SET       Compare = false
                                        SET       Shift_Left = second value of Unique_Table_Pair
                                        CHECK
                                        TABLE_PAIRS_UPDATE
                              SET       Run_Mode = 0
                    ELSE      continue
                    BIT_ENCODE
END DO
CLEAR_ENCODER (see A.3.8)
END PROCESS = BLOCK_ENCODE
```

A.3.3 **COMPUTE** - Computes the correct value to add to the Current_Value and subtract from the Width. It also determines how much of the Current_Value to shift to the Code_Block.

PROCESS = COMPUTE

SET Width = Width - (2 raised to negative power of second value in Table_Pair (Number))

SET CV = CV + (2 raised to negative power of second value in Table_Pair (Number))

SET Compare = true

IF Width < 1.0000

THEN **SHIFT** Width left one place

APPEND zero to rightmost end of Width

SET Shift_Left = 1

ELSE **SET** Shift_Left = 0

CHECK (see A.3.5)

TABLE_PAIRS_UPDATE (see A.3.6)

END PROCESS = COMPUTE

A.3.4 **BIT_ENCODE** - When the Encoders are not in Run Mode, block bytes are encoded on a bit by bit basis.

PROCESS = BIT_ENCODE

SET Number = 1

SET Bit_Count = 1

DO WHILE Bit_Count < 9

IF bit (Bit_Count) of Current_Byte = first value in Table_Pair (Number)

THEN **COMPUTE**

NEXT_TABLE_PAIR (see A.3.7)

ELSE **SET** Width = 1.0000

SET Shift_Left = second value in Table_Pair (Number)

SET Compare = false

CHECK

TABLE_PAIRS_UPDATE

NEXT_TABLE_PAIR

SET Bit_Count = Bit_Count + 1

END DO

END PROCESS = BIT_ENCODE

A.3.5 **CHECK** - The Current_Value is checked and the proper data is appended to the Code_Block.

```
PROCESS = CHECK
IF          Code_Block = null
THEN        continue
ELSE        SET       Count = 0
            ADD       bit (Count) of CV to rightmost end of Code_Block
            SET       bit (Count) of CV to 0
IF          Rightmost integral 8 bits of Code_Block = (FF)
THEN        INSERT (0) after last integral 8 bits of Code_Block
ELSE        continue
SET         Count = Count + 1
DO WHILE    Shift_Left > 0
            APPEND    bit (Count) of CV to rightmost end of Code_Block
            SET       bit (Count) of CV to 0
            SHIFT     CV left one place
            APPEND    ZERO to rightmost end of CV
            SET       Shift_Left = Shift_Left - 1
            IF        Code_Block = integral multiple of 8 bits
            THEN      IF        rightmost 8 bits of Code_Block = (FF)
                      THEN      APPEND    0000 to Code_Block
                      ELSE      continue
            ELSE      continue
END DO
END PROCESS = CHECK
```

A.3.6 TABLE_PAIRS_UPDATE - The Table_Pairs values are updated to the correct values, if required. If Mc_Count = 1111 and 1 is added to it, Mc_Count = 0000.

PROCESS = **TABLE_PAIRS_UPDATE**
IF Compare = true

THEN **CASE** second value of Table_Pair (Number) AND Mc_Count

[second value of Table_Pair (Number) = 1 AND Mc_Count (3:4) = 11]
second value of Table_Pair (Number) = 2

[second value of Table_Pair (Number) = 2 AND Mc_Count (2:4) = 111]
second value of Table_Pair (Number) = 3

[second value of Table_Pair (Number) = 3 AND Mc_Count (1:4) = 1111]
second value of Table_Pair (Number) = 4

END CASE

SET Mc_Count = Mc_Count + 1

ELSE **IF** second value of Table_Pair (Number) > 1

THEN second value of Table_Pair (Number) = second value of Table_Pair (Number) - 1

ELSE invert first value of Table_Pair (Number)

END PROCESS = TABLE_PAIRS_UPDATE

A.3.7 NEXT_TABLE_PAIR - The next Table_Pair to be used in the encoding process is determined.

PROCESS = NEXT_TABLE_PAIR

IF Bit_Count < 8

THEN **IF** bit (Bit_Count) of Current_Byte = 1

THEN Number = Twice Number + 1

ELSE Number = Twice Number

ELSE **SET** Number = 1

END PROCESS = NEXT_TABLE_PAIR