# PRACTICAL APPLICATION OF IRREDUCIBLE CODINGS: VIRTUAL EXTENSION OF STORAGE CAPACITY

By

P. HANTOS

Department of Automation, Technical University, Budapest

Received September 14, 1976
Presented by Prof. Dr. F. CSÁKI

## Introduction

In our days the minicomputers cope with increasingly serious tasks in the modern information processing systems. By correct organization and decomposition of the data base, the minicomputer may assume, in addition to performing intelligent terminal functions, also data base processing functions. Part of the data base is accommodated in the background storage of the minicomputer permitting an access, faster by an order of magnitude, compared to the adjoined great computer. The fast access may be realized mainly on fix-head magnetic discs and magnetic drums. But a common characteristic of the above means is the limitation of the storage area from above, therefore the virtual extension of the storage capacity is of paramount importance. For reducing the operation period the programs must be prepared in the ASSEMBLER language and the given possibilities of the machine must be exploited at a maximum. Our experiments were performed with an R-10 minicomputer oriented to process control with an average operation period of 2.5 $\mu$s per machine instruction.

## Coding fundamentals and methods

In the model shown in Fig. 1 the information is obtained from the message source K. The messages are fitted to the channel by coding. The aim of the process examined in this paper is to derive X into A in a way resulting so that the elements of A carry average maximum information.

In prefix or irreducibile code systems, none of the coding words is a

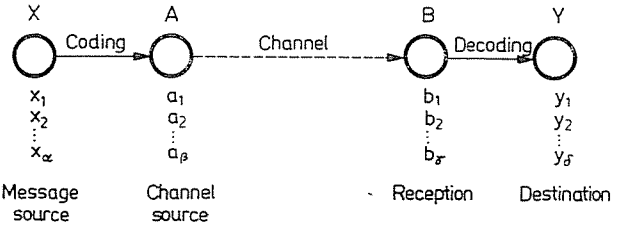


*Fig. 1.* Fundamental model of information theory

continuation of the preceding one. To be prefix feature is the sufficient condition of the unequivocal decodability.

The theorem of irreducible codes to exist is expressed by the Kraft inequality:

$$\sum_{i=1}^{n} r^{-l_i} \leq 1 \tag{1}$$

Validity of this inequality is the necessary and sufficient condition of the existence of the prefix-type code system of code lengths $l_1, l_2 \ldots, l_n$ in the case of $n$ source symbols and a code alphabet of $r$ elements. The theorem is proved in [1].

According to the theorem of McMillan, the inequality (1) is the necessary condition of the unequivocal decodability. This theorem is proved in [2].

The main irreducible code systems published are:

- the Gilbert—Moore alphabetic code [3]
- the Huffmann code [4]
- the Shannon binary code [1]
- the Shannon—Fano process [1]

From among the above codes only the Huffmann-type coding supplies a code, which is optimum in any case, therefore this is the only to be discussed in the following.

Finally the main characteristics of optimum coding may be summed up as follows:

a) The coding is optimal, if the symbols contained in A appear with identical probability;

b) the minimum average word length attainable by the optimal coding in principle is:

$$L_{\min} = \frac{H(x)}{ld\, r} \tag{2}$$

where the average word length is:

$$L = \sum_{i=1}^{n} P(x_i) \cdot l_i \tag{3}$$

Here $P(x_i)$ is the probability of the $i$th symbol of the source of messages to appear.

c) In the case of optimal coding the average word length may be confined between the following limits:

$$\frac{H(x)}{ld\, r} \leq L < \frac{H(x)}{ld\, r} + 1 \tag{4}$$

This means that the value (3) may be well approximated, if a code word of length $l_i$, which is an integer just not inferior to the value $-\frac{ld\,P(x_i)}{ld\,r}$ is assigned to the message $x_i$ of probability $P(x_i)$.

## The optimal message transmission system of information theory

Shannon's first coding theorem supplied the solution of developing the model of Fig. 1. The problem is namely that the number $-\frac{ld\,(Px_i)}{ld\,r}$ is generally not obtained as an integer, so the average word length deviates from the value $H(x)/ldr$ even in the case of the best algorithm. The average number of the symbols of A utilized for transmitting a message from X may be arbitrarily approximated to the value of a message source of any probability $P(x)$, if not the elements $x_i$ *but the series* $-x_i$ *of length* $N$ *are coded.* This process is called the $N$-fold extension of the source "X" (Fig. 2).

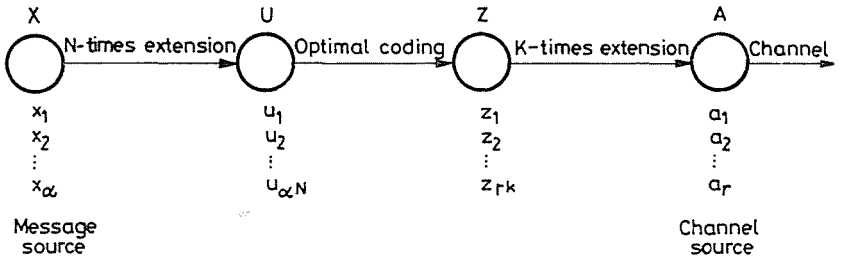


*Fig. 2.* Optimal model of message transmission systems

During the extension of the source a so-called zero-memory source was assumed. But in many cases the symbol emitted at a given instant depends on the previously emitted symbols or in other words, the subsequent symbols are statistically not independent. For example, also written texts have this property. For modelling written texts the Markov source may be utilized:

*The Markov source of order m is given, if the symbols* $x_i$ *and the conditional probabilities* $P(x_i \mid x_j; x_{j2}, \ldots, x_{jm})$ *are given.*

The Markov source may be illustrated with the help of the state constitutional diagram. Fig. 3 shows the state diagram of a second order Markov source. In our case the role of the "channel" is assumed by the background storage. Accordingly, the "optimum message transfer" means practically "optimum storage exploitation" and under certain circumstances an I/O time saving may also be achieved in addition to space saving [5].

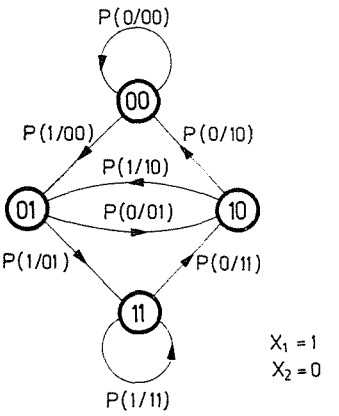


*Fig. 3.* State-diagram of a simple Markov-source

If the original message source X is initially of a uniform distribution, then the optimal coding is not necessary, because the optimum code is obtained directly by deriving the extended source into the block-code of the shortest constant word length.

The above considerations assume a noise-free channel. In the case of a noisy channel an error limit coding is applied in the optimum code Z by "building-in" redundancy for safe transfer.

In designing the code system, the following factors must be examined:

a) On the basis of the distribution function of the message source X it must be determined, whether irreducible coding is useful, or not. The code variable length results in complicated – and possibly time demanding – programs; in the case of "bad statistics" only a minimum compression is possible by much effort.

b) During the extension of the source the dimension of the coding-decoding table increases rapidly, which fact must be taken into account in calculating the compression ratio.

c) By analyzing the channel, the required rate of error limitation must be determined. Any "excessive safety" increases unjustifiably the operation period and may further reduce the compression ratio.

From the above it is evident that no universal, optimal coding, applicable for any arbitrary source and simultaneously error limiting can be realized.

## Analysis problems of sources supplying text information

### 1. *The analysis of the discrete message source*

Publications on statistical coding and quantitative linguistics consider only the probability distribution of the 26 letters of the English alphabet. This may be sufficient from the aspect of linguistics, but in our case any letter,

number and interpunctuation may be emitted by the source. Numbers and interpunctuation have no characteristic distribution functions, as have letters, so coding these characters by codes of variable lengths is not practicable. In this case the extensive growth of the source alphabet would result in several inconveniences:

a) The coding length is much incerased by the great number of separate elements with different low probabilities in the case of irreducible codes. Beyond 16 bits, coding and decoding become very complex, as the accumulator register of the computer is of 16 bits only.

b) Coding and decoding times of very long code words are considerably longer.

c) The big size source results, if extended, in practically untreatably big tables.

To solve the above problems let us suggest the introduction of the "Branch Out" character. The source alphabet has 29 elements:

- the 26 capital letters of the English alphabet
- space
- new line NL
- Branch out character BO.

The probability of the character BO is determined in the knowledge of the probabilities of numbers and interpunctuation, a medium, or shorter irreducible code word length may generally be reckoned with. In the coded text the prefix code of the character BO is followed by the code EBCDIC of the character to be coded, so the latter is very simple to decode. As to space utilization we found that the Huffmann code of a low-probability character in the great size source alphabet is nearly as long as the value (BO code + 8) in the code system belonging to the reduced alphabet. At the same time the source could be extended, which is very important for reducing the data and for keeping the irreducible code word length below 16 bits.

## 2. *The extension of the source*

In our first experiments we studied the irreducible codability of the words. The word is a grammatical conception, but it is also easy to recognize formally. For the recognition a comma, a full stop, a space has to be found in the character strings. The rules of inflexion in the Hungarian language do not permit the efficient application of the process, because the inflected words would be interpreted as new basic words by such a program, based on formal character manipulation.

Outside grammatics, the extension of the source means practically the arbitrary grouping of the characters. During the N-fold extension of the zero-memory source 29 different alphabets consisting of 29 elements each must

be treated for the irreducible coding. The dimension of the decoding scheme may be estimated as:

$$S = \alpha^2 \cdot (2 + 1) \quad [\text{byte}] \tag{5}$$

as not alone the code, but also the code length must be recorded over 1 byte for the sake of unequivocal decodability. The obtained S = 2.4 Kbyte is an acceptable result, as the decoding scheme may be accommodated in the operative storage.

In the following let us examine what order of Markov source is it advisable to be applied. In the case of $m = 2$ we obtain S = 71 Kbyte. This is a rather high value, but still realizable in a two-level hierarchical structure, if the decoding scheme is stored on a disc. But the following three aspects must be taken into account:

The effective compression ratio is greatly reduced by the big-size decoding table.

The operation time is increased by three orders of magnitude.

The average information per one symbol is not reduced as much as expected. The following data were established by analyzing the English language [6]:

a) Average information per one letter of the alphabet:

$$H(x) = 4.065 \frac{\text{bit}}{\text{symb}} .$$

b) Modelling by the Markov source ($m = 1$)

$$H(X \mid x_i) = 3.32 \frac{\text{bit}}{\text{symb}} .$$

c) Modelling by the Markov source ($m = 2$)

$$H(X \mid x_1; x_2) = 3.1 \frac{\text{bit}}{\text{symb}} .$$

d) Modelling by the Markov source ($m = 100$)

$$H(X \mid x_1; x_2, \ldots; x_{100}) = 1 \frac{\text{bit}}{\text{symb}} .$$

Taking as basis the average information value, which would be obtained in the case of an alphabet of uniform probability distribution: this value is $H = 4.75 \frac{\text{bit}}{\text{symb}}$ in the case of the English alphabet consisting of 27 symbols

(26 letters + space). Assuming the optimal code, the estimated value of space saving is:

a) 14.5 %
b) 30.1 %
c) 34.7 %
d) 78.9 %

(with the space occupied by the decoding scheme left out of consideration !).

The above data show that the Markov-type modelling results in a gain of 15.6 % in the case of ($m = 1$), while the extension $m = 2$ would supply only a growth of 4.6 %. This space saving of 4.6 % would cost 3 *orders of magnitude* (!) of the operation time as an operation in the operative storage is of the order of 10 μs and the average access time to the disc is 10 ms. *Remark*: The gain of 15.6 % is a saving in the sense of information theory. For determining the real compression, 8 bits of the EBCDIC code must be taken into account, so the physical gain amounts to 58.6 % ! The decoding scheme can be accommodated entirely in the operative storage.

Conclusions for designing the algorithms:

- a source alphabet of 29 elements is defined;
- the "Branch Out" symbol is introduced;
- a first-order Markov modelling is applied;
- the optimum Huffmann system is chosen.

## The algorithm of the text analysis

The condition of fast statistics preparation is to have the applied counting fields residing in the storage. The program was prepared in the ASSEMBLER language and it was attempted to exploit maximally the advantages offered by the architecture and the addressing system of the computer R-10. Because of space shortage, for the detailed description of the computer R-10, we refer to [7]. The counting fields are formed in the LDS of the program (Fig. 4). In the initiating phase of the program the counting fields are zeroed and the variables are marked OLDS filled by the serial number of space. The main table (MTAB) is placed at the beginning of the LDS (Local Data Segment). The individual binarily EBCDIC characters give exactly the relative address of the serial number L assigned to the given character ($OLDS_i$, or $NEWS_i$). The main table is followed by 29 small tables (TAB1 to TAB29). *The meaning of the ith field is*:

The probability (i.e. the frequency) of the EBCDIC character of the serial number $i$ to occur in the position following the EBCDIC character of the serial number $j$.

The overall space requirement of the tables is about 1.8 Kbyte.

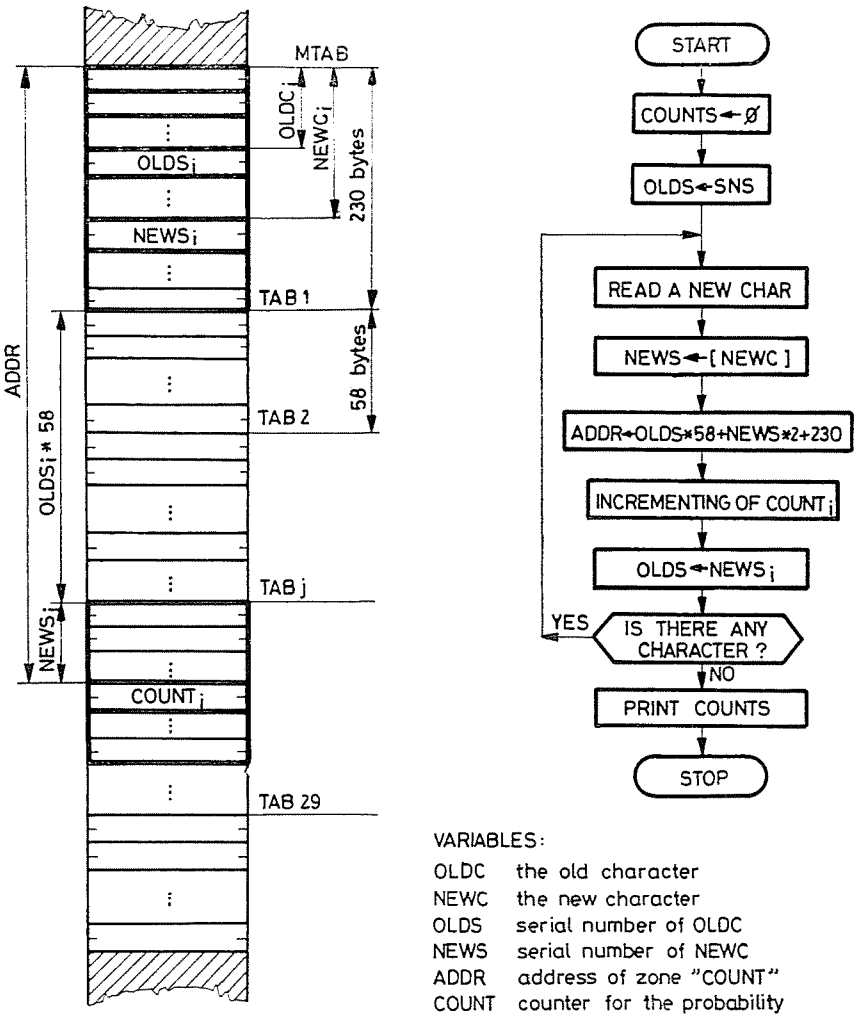


*Fig. 4.* Algorithm of textual analysis

## Generating the coding scheme of the Huffmann method

The Huffmann scheme is generated by arranging the elements of the source alphabet in the increasing order of probabilities, then reducing the set of information and combining both elements of lowest probability into a single information. The element obtained in this way is fitted into the place corresponding to its probability. The process is continued until two elements alone remain. The assigning Ø to one and 1 to the other one had recourse to the reduction scheme. Every combination is decomposed and the codes of the sepa-

rated parts are extended in sequence with by alternating Ø and 1, to get the code system.

The arrangement of the elements of the source alphabet is no problem, as the fields to be arranged are low in size (59 bytes). From among the known arranging algorithms the "ranking sort" has been selected [8]. The space requirement of the process in the case of $n = 29$ elements is 87 bytes, as not only the probabilities, but also the EBCDIC code must be stored (Fig. 5).

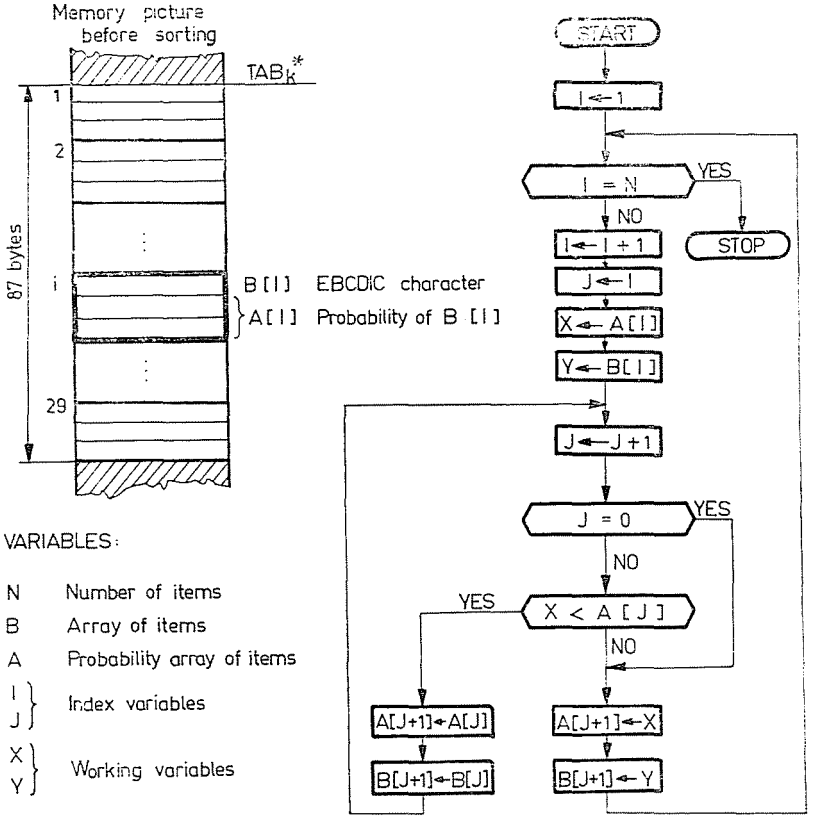


*Fig. 5.* Algorithm of "ranking sort"

The process is characterized by the following data:

Number of comparisons:

$$\mathrm{NCO} = n(n-1)/4 = 203 \tag{6}$$

Number of place transpositions:

$$\mathrm{NPT} = n(n-1)/4 = 203 \tag{7}$$

Number of runs:

$$\mathrm{NRS} = n - 1 = 28 \tag{8}$$

The process must be applied to all the 29 TAB-s. TAB* in the figure designates the modified ranking field.

Fig. 6 shows the algorithm performing the subsequent reduction of the messages. For case of computation the probabilities* are arranged in increas-

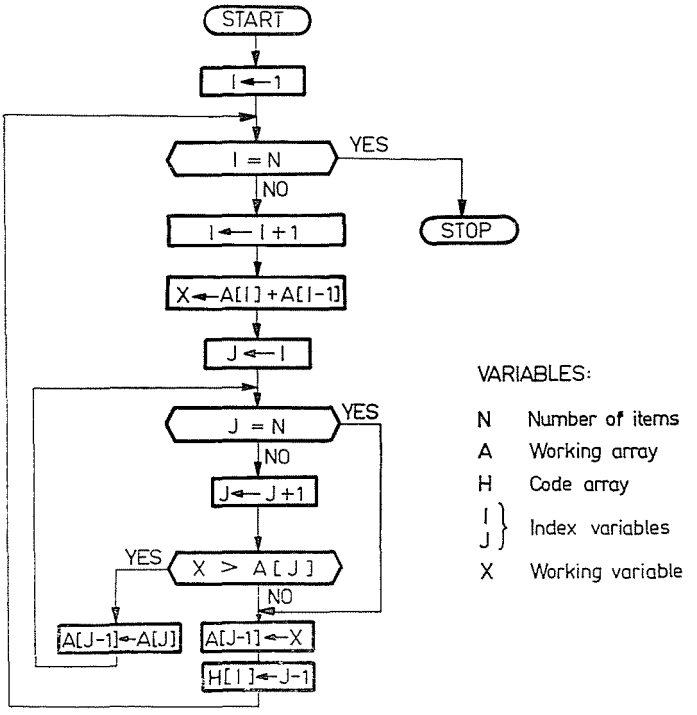


*Fig. 6.* Successive reduction

ing order, as against the description in [4]. The block H contains the output data and block A, the probabilities. H is in principle of $N$–2 dimension, with $N$ being the number of elements in the source. This is completed by another element for programming reasons and the sum is put into the first position.

* *Remark*: For generating the code, the probability is not necessary; a frequency index showing how many times the actual character appeared in the text, is sufficient. In this way running time is saved, as the process is performed with integers all along. In the following "probability" is always understood as an index of the above meaning.

The code generation may be followed up in the flow chart, Fig. 7. The code is created in block A. As the codes are of variable length, the lengths of the code words are preserved in the vector HO.

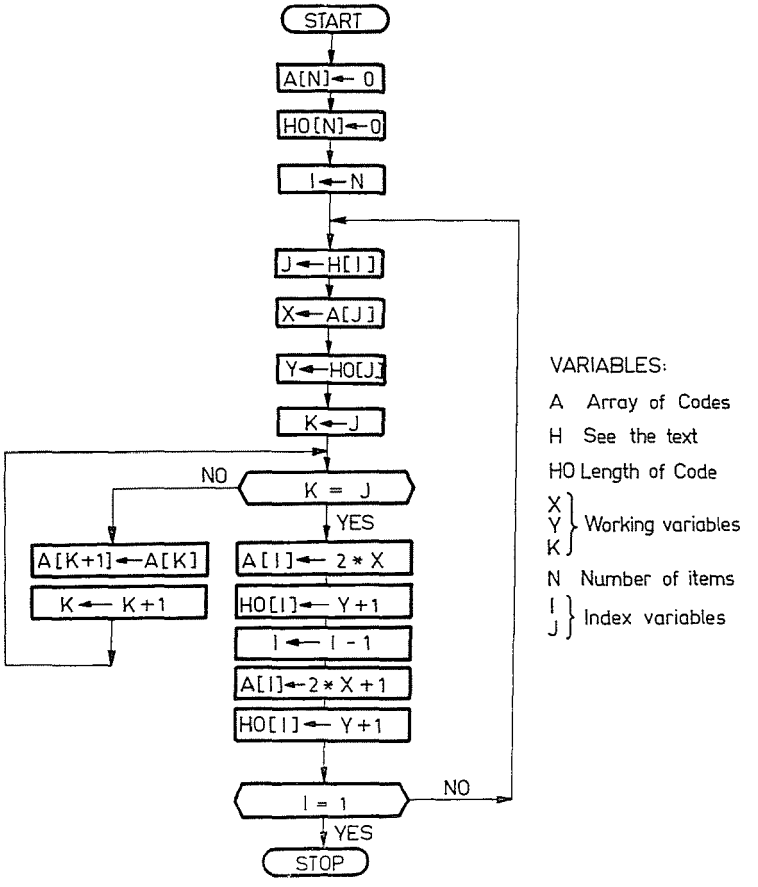


*Fig. 7.* Generating the coding scheme

## The coding algorithm

The algorithm of the coding is relatively simple, Fig. 8. The codes appear in the TAB** fields in the original in identical order, and not in the order of probabilities. Although this means an additional ranking task, space is saved. Let us observe the use of the "Branch Out" character. The "blank", the "New Line" and the alphabetical characters are selected coding in all other cases "BO", with the original EBCDIC code catenated to it.

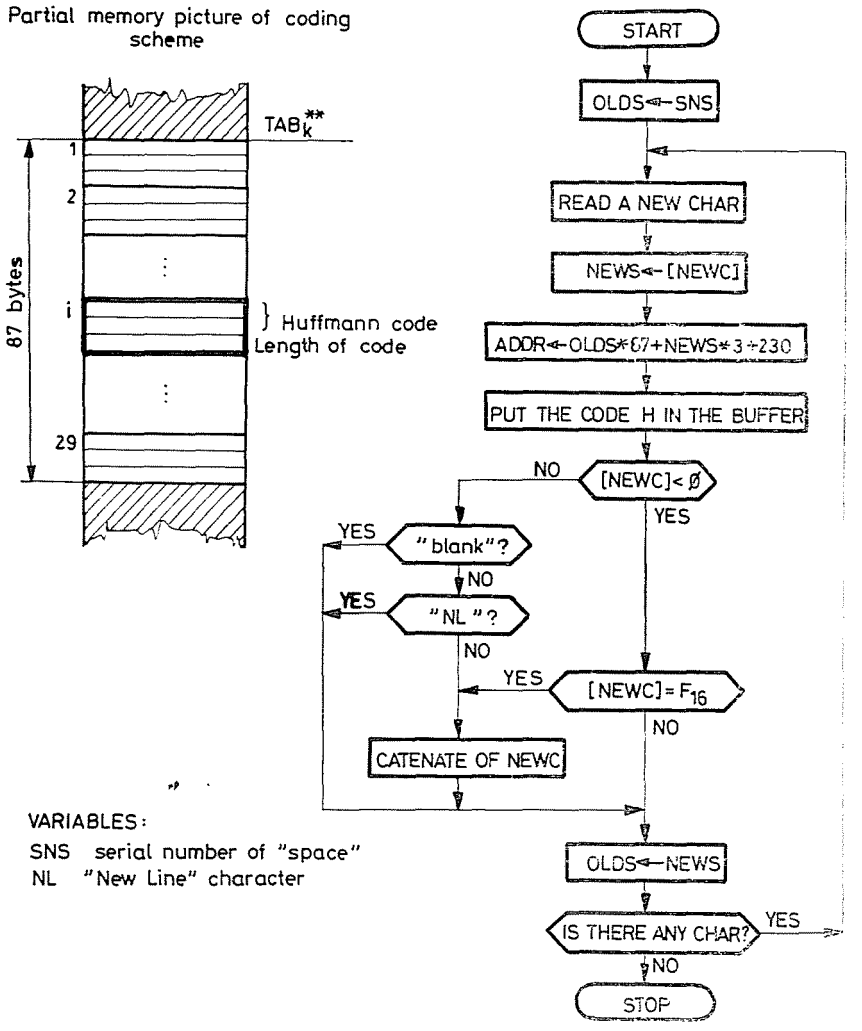


*Fig. 8.* Algorithm of coding

## The decoding algorithm

The prefix codes may be decoded with the help of a "code tree". A code is identified by proceeding along branches Ø, or 1 of the three, depending on the arriving signals until the end point is reached. This process excludes the application of fast, eventually parallelly operating algorithms. The decoding speed depends anyway on the efficient derivation of its code.

In Fig. 9 a decoding algorithm is suggested. Though not shown in the flow chart, obviously TAB$_k^{***}$ is selected in knowledge of the character decoded in the previous step. The information is entered from the working a-

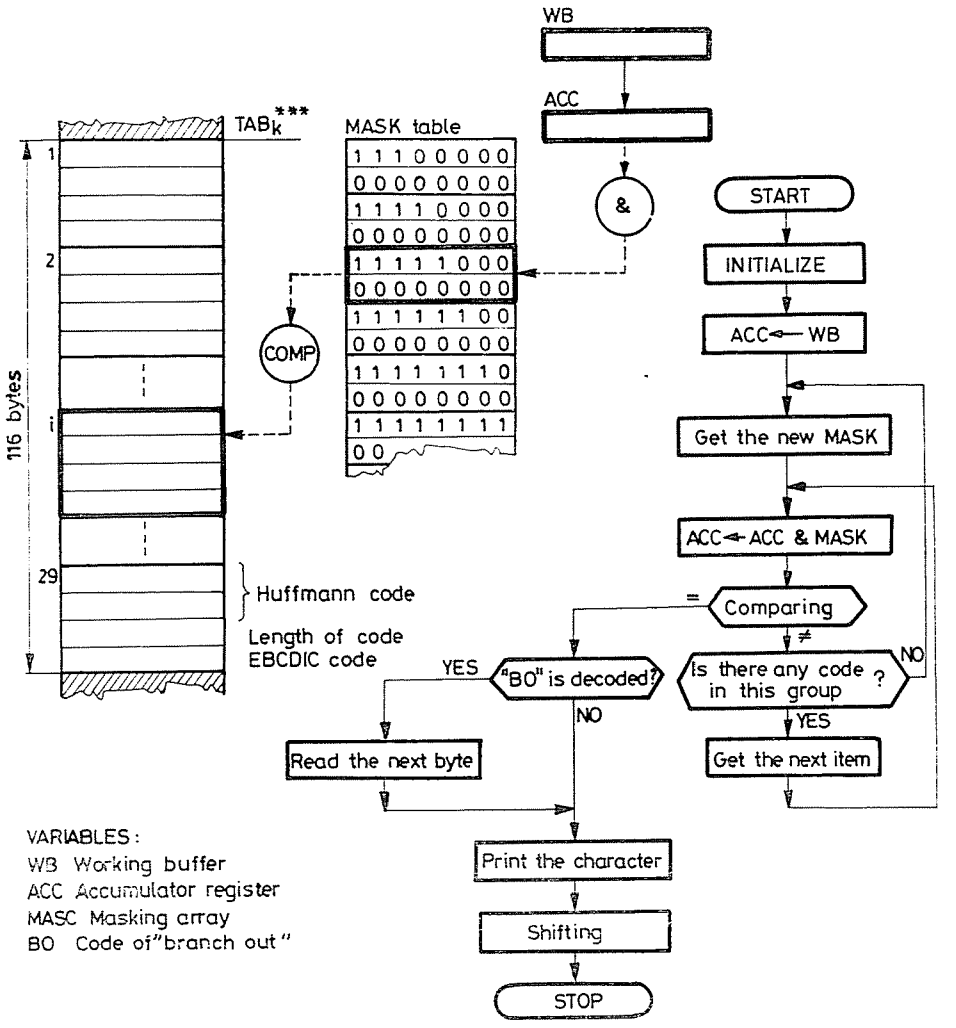


*Fig. 9.* Algorithm of decoding

rea word by word into the accumulator, whose contents is then masked by the first element of the table "MASK"; the length of the mask agrees with that of the shortest Huffmann-code (3 in the figure). Then the program "surveys" all the elements of $TAB_k$***, whose length agrees with that of the mask. Note that the codes are arranged in the order of probabilities, so the most probable element is found by the least steps. On decoding the "Branch Out" character, also the next byte is entered from the work area, as this will contain the effective EBCDIC code.

If the required code is not found in the group of elements of the given length, the process is continued with the next element of the "MASK" table.

This is a kind of sequential processing of the set ranked in the sequence of the probabilities. This artful handling by the "MASK" table is seen to consist essentially in bypassing a bit-manipulation task in the byte-organized computer. In the case of an average word length of 5 bits, the realizable maximum decoding speed is about 5000 characters/sec. This is not too high a value, but as the decoding phase precedes directly the print-out, a high speed is generally not required. The possible maximum output speed to alpha-numerical display units is about 2000 char/sec, so our decoding program supplies even this fast output at a satisfactory rate.

## The reliability of coding

If a single bit of the prefix code changes, then all the message after this bit becomes undecodable. For this reason the problem of reliability is of an enhanced importance. The question arises "what degree of error limitation in coding is necessary after this adaptation coding".

Figure 10 shows the simplified scheme of a computer. With conceptions in Fig. 1, two models may be established with the data paths taken into account:

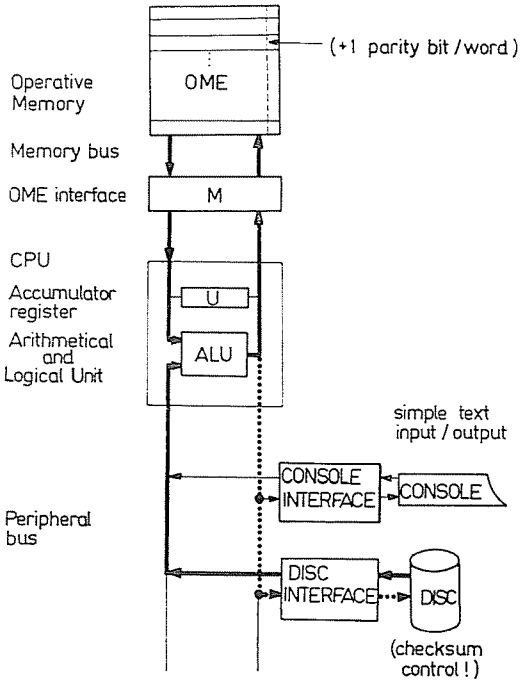


*Fig. 10.* Schematic model and data paths of minicomputers

a) General operation:

$$\text{Console} \rightarrow \text{CPU} \rightarrow \text{OME} \rightarrow \text{CPU} \rightarrow \text{DISC} \rightarrow \text{CPU} \rightarrow \text{Console}$$

The role of the "channel" is assumed by the disc, the discrete source and the absorber are represented by the complex Console → CPU "supported" by OME.

b) Decoding:

Here the disc may be regarded as a discrete source, OME as an absorber and the role of the "channel" is assumed by the complex of the disc-interface, CPU, OME-interface.

At the CPU level the probability of error is low. The main sources of error are the OME-interface and the DISC-interface. The construction of the hardware provides a satisfactory protection: in OME a $+1$ parity bit for each word serves for checking, on the disc sum control is checked sector-wise. Although these methods offer no protection against transposition errors, in practice they meet our purposes. For instance, in the case of a relatively high channel error probability of $p = 10^{-3}$ the indication capacity of the code with the parity elements is 98.4 % which may be regarded good enough.

It may be of interest to note that having reduced the average word length to 4 bits/symb; a Hamming-type (8,4) code [9] yielded a code, whose average word length agrees with the 8 bit word length of the source; but it may also be applied for correcting single errors and for *indicating* double errors. The operation period of decoding is naturally increased by the correction coding.

## Summary

A significant data reduction of 30 to 50% has been realized at the cost of an operation time increase still acceptable for practical applications. The source emitting text information has been modelled as a first-order Markov source. The design of the coding scheme involved the "Branch Out" character providing for the applicability of the Huffmann-type coding in practice. Maximum use of the peculiarities of the computer and the programming language resulted in minimizing the coding-decoding time. No radical reduction of the running time was seen to be feasible by software methods. An additional reduction by about one order of magnitude may be attained by single-purpose microprogrammed, or microprocessor arithmetics. The main problem is that the architecture is fundamentally byte-manipulation oriented while in applying variable length codes the elementary operations have mostly to be performed at bit level. The abrupt development of microprogramming technology and microprocessors permits to develop fast and economical irreducible coding and decoding units.

## References

1. Reza, F. M.: An Introduction to Information Theory. McGraw-Hill Book Co., Inc. 1960.
2. McMillan, B.: Two Inequalities Implied by Unique Decipherability. IRE Trans. On Information Theory, 1956. pp. 115—116.
3. Gilbert, E. N.—Moore, E. F.: Variable Length Binary Encodings. Bell System Techn. J. .1959. pp. 933—968.

4. HUFFMANN, D. A.: A Method for the Construction of Minimum Redundancy Codes. Proc. IRE, 1952, pp. 1098—1101.
5. ALSBERG, P. A.: Space and Time Saving Through Large Data Base Compression and Dynamic Restructuring. Proc. of the IEEE, Vol. 63, No. 8. August, 1975, pp. 1114—1122
6. ABRAMSON, N.: Information Theory and Coding. McGraw-Hill, 1963.
7. EC 1010 Reference Handbook. 201.017.11.01-SW, 1974.
8. FLORES, I.: Computer Sorting. Englewood Cliffs, N. J.: Prentice-Hall, Inc. 1969.
9. HAMMING, R. W.: Error Detecting and Error Correcting Codes. Bell Sys. Techn. J., 1950, pp. 147—160.
10. PETERSON, W. W.: Error-Correcting Codes. J. Wiley and Sons Inc., New York, 1961.
11. GALLAGER, R. G.: Information Theory and Reliable Communication. J. Wiley and Sons Inc., New York, 1968.

Péter HANTOS H-1521 Budapest