SOLVING A CHESSBOARD PUZZLE WITH THE PASCAL

by A. J. DEKKERS *) and A. J. W. DUIJVESTIJN *). 681.14-523.8:685.854

Introduction

The preceding articles in this issue illustrate the use of the electronic computer for various mathematical problems, such as working out arithmetical formulae, solving equations, integrating differential equations, etc. The fact that the machine has to *compute* in all these problems seems self-evident, but this is not so: it can also be used for non-arithmetical tasks, or at least for tasks which in the first instance have nothing to do with computation. Examples are to be found in the various fields of mathematics itself, such as in topology, abstract algebra (group theory), etc. Then there are "logical" problems and routines such as sorting and collating, which are frequently encountered outside the realm of mathematics, especially in the accounting department of a business enterprise. This is reflected in the installation of two nearly identical computers in Philips Computing Centre — the PASCAL and the STEVIN — the PASCAL being employed for mathematical work proper, and the STEVIN almost exclusively for the various administration departments of Philips [1]).

The distinction between arithmetical and non-arithmetical problems is perhaps not strictly tenable. This was the subject of learned debate long before calculating machines existed: some defended the postulate that logical reasoning is nothing but a kind of computation, while others argued that computation is simply a kind of logical reasoning. If one considers the operations which the computer is made to perform in solving all kinds of problems, one may again conclude that the above distinction is indeed superficial. In fact, with all problems which are commonly called non-arithmetical, one is sure to find in the pertaining machine programme the operation of *counting*, probably even more than once. Now, it is evident that this operation is also at the base of the operation of addition and hence of the other mathematical operations proper.

With a view to exploring all the computer's possibilities, it can be useful to attempt to programme it for widely diverse problems, including strictly non-arithmetical ones — again in the commonly understood sense of the word. For example, at the "Studiecentrum voor Administratieve Automatisering" in Amsterdam a programme, commissioned by Euratom, is being designed under the direction of Dr. M. Euwe which will enable the computer to play a game of chess. We have chosen a simpler problem as an object for study — a chessboard puzzle. A procedure for solving this puzzle has been worked out and programmed for the PASCAL. This procedure is discussed below.

Description of the puzzle

The chosen puzzle will now be described [2]). A chessboard is divided up into 12 pieces as shown in *fig. 1*. Pieces 5a and 5b happen to be identical; all other pieces differ from these and from each other. The problem is to fit these 12 pieces together to form a complete chessboard with the proper alternation of black and white squares. Is there more than one solution? If so, give all possible solutions.

The solution can only be found by *trial and error*, but the number of possibilities to be examined is enormous. Therefore first of all we must systematize the trial procedure so that no possibility is overlooked. Secondly, there is little hope of finding the

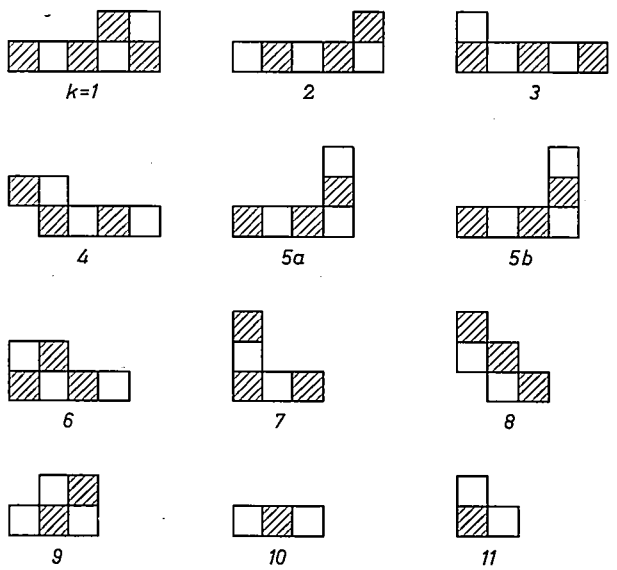

Fig. 1. The 12 pieces into which the chessboard is divided. There are 11 different shapes; one shape (No. 5) occurs twice.

*) Philips Computing Centre, Eindhoven.

[1]) See W. Nijenhuis, The PASCAL, a fast digital electronic computer for the Philips Computing Centre, Philips tech. Rev. 23, 1-18, 1961/62 (No. 1).

[2]) The puzzle is one of the Peter Pan series marketed by Cecil Coleman Ltd.

complete solution of the puzzle without the help of an electronic computer, for, as will presently be seen, it is necessary to examine 3 million possibilities, yielding *eleven* different solutions.

Method of solving the puzzle

Systematization of the trial procedure mentioned reduces to trying to cover the 64 squares of an empty chessboard in a fixed order by laying down, again in a fixed order, the given 12 pieces, each in their different possible orientations.

It is essential in this procedure to keep a complete *record* of the squares already filled and of the pieces and orientations already used and tried. For, when each piece is laid (i.e. upon every "move") it is necessary to see that we are not in conflict with the requirements, e.g. that the piece is not covering a square that has already been occupied. If we come to the conclusion that none of the remaining pieces can cover the open square whose turn it now is, without somewhere coming into conflict, then the piece laid in the previous move must be withdrawn and the same piece tried in its next orientation (in the prescribed order), or the next piece tried. When all possibilities have been exhausted with the still available pieces, we must revise the move before last, and so on. All data on every move must therefore be preserved.

For this purpose we can of course make use of the large *memory* of the PASCAL.

We shall now first discuss the systematization of the trial procedure adopted, give an example of how it works, describe how the machine "tries" a piece and how the data are recorded, and finally examine the solving process as a whole.

The systematization

The systematization consists in arranging the possibilities in order of the series of whole numbers, so that a "call" for each possibility can be made by simply increasing an address number in a memory by 1.

To this end the first step is to give the squares on the chessboard serial numbers n. We call the bottom right square $n = 1$, and in our approach to the problem we decide, quite arbitrarily, that this shall be a black square (with due apologies to the chess players among our readers). The numbers n on the bottom row increase from right to left, then on the next row from right to left again, and so on.

The next step has already been seen in fig. 1, where the eleven *dissimilar* pieces are numbered $k = 1, 2, \ldots, 11$. (The complication that piece No. 5 is duplicated will presently be taken into account in a simple manner.)

In principle each piece can be laid on the board in four distinct orientations, each turned through 90°. All the possibilities thus obtained we place in a row and give them serial numbers j. Each of these possibilities we call a "lay" — a piece that is laid experimentally in a particular orientation on the board.

The row of available lays is shown in *fig. 2*. It can be seen that we have in reality made *two* rows of lays, the "black" and the "white", depending on the colour of the "master square", i.e. the square at the right end of the bottom row of the piece. A lay is used such that this square is laid on the board on the empty field (to avoid confusion a square of the board will be called a field) whose turn it is to be tried (the "trial field"). If that field is white the machine must try a "white lay"; if the trial field is black then the lay has to be black. Further, it will be seen that only two of the four possible orientations of piece No. 1 (fig. 1) are included in the row (black lay 1 and white lay 1). This ensures that once a solution has been found there is no search made later in the solving process for the trivial reverse form of that solution (the form turned 180°). Piece No. 10 (fig. 1) also occurs only twice in the row of lays, but that is simply because only two distinct orientations of this piece are possible (white lays 17 and 18). Altogether there are now 19 black and 21 white lays that have to be tried in their numerical order, j_b and j_w respectively.

Example of attempt at a solution

To illustrate the trial procedure a situation is presented in *fig. 3* which arises after a few moves at the very beginning of the solving process. Lay $j_b = 1$ (piece $k = 1$) has been placed on the first trial field $n = 1$, then lay $j_w = 3$ (piece $k = 2$) on the next available trial field $n = 6$, which is white. The next available lay $j_b = 4$ does not go on the next (black) trial field, $n = 7$, for it would project beyond the board; nor do lays $j_b = 5$, 6, 7, 8 and 9, but $j_b = 10$ (piece $k = 6$) does go. The next trial field is on the second row and is white. Lay $j_w = 4$ is not a fit, nor is $j_w = 5$, but $j_w = 6$ is (piece $k = 4$). With lay $j_b = 5$ ($k = 3$) on the next trial field, followed by $j_w = 18$ ($k = 10$), the second row on the board is filled. On the first trial field on the third row we place lay $j_b = 19$ (piece $k = 11$), but now we can go no farther: for the next trial field, which is the last on the third row, there is no suitable lay available.

The action to be taken has already been indicated. The last piece, No. 11, must be withdrawn from the

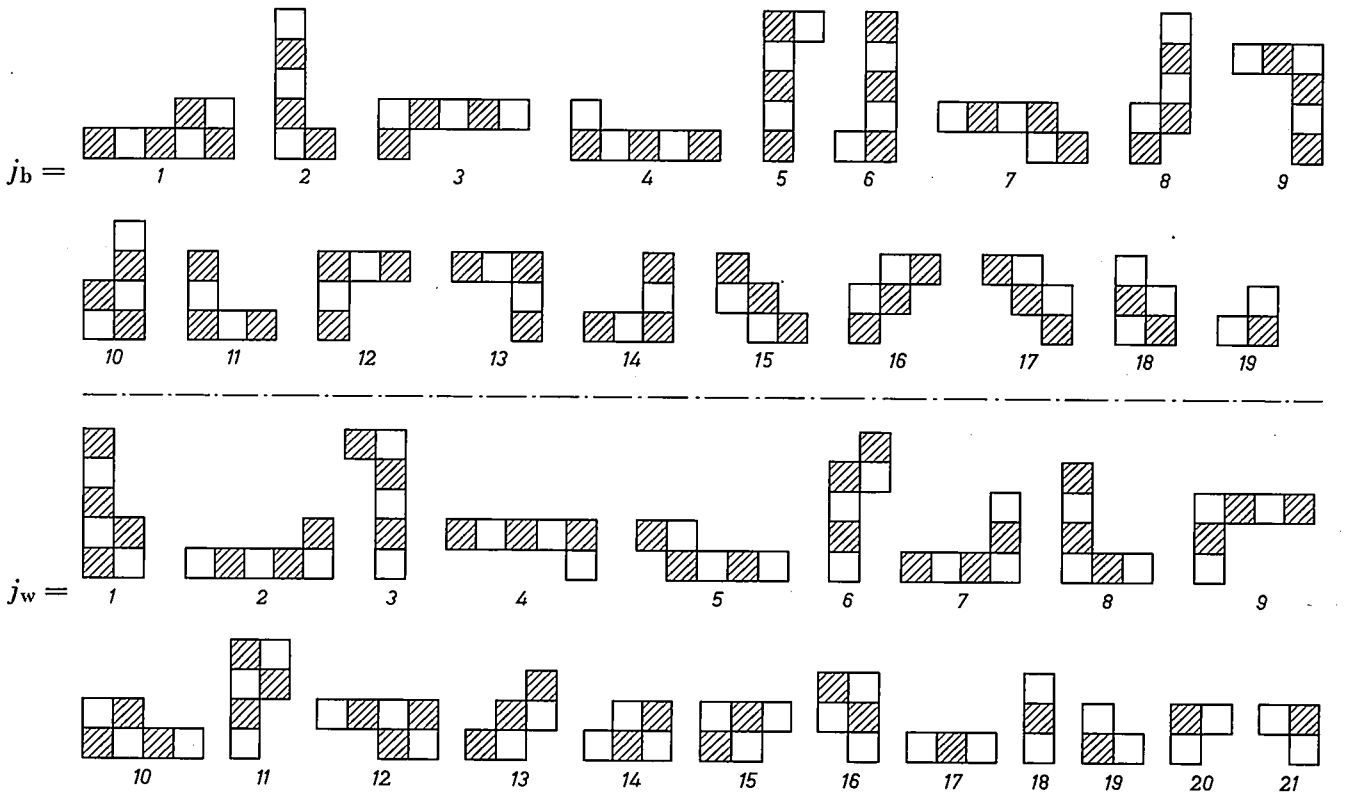

Fig. 2. The complete row of "black" and "white" "lays" (pieces laid in a particular orientation) that must be tried successively in a "move".

board (and the machine must amend all the appertaining steps in the records, which we shall presently discuss). We are now back in the situation of having to fill the preceding trial field, and to do so we must try the next lay j_b after $j_b = 19$. But no such lay exists, and therefore we must also cancel the step preceding the one just cancelled, and so on.

Realization of the trial procedure by the computer; keeping the records

In the example just discussed we have assumed a human operator who can check whether a lay, "called upon" in accordance with the systematic procedure, can be placed or not. The criteria are (1) that no field already occupied can be occupied again, and (2) that no square of a lay may project outside the board.

How can the computer make this check? The checking is linked with the keeping of records, and we therefore cannot avoid describing also part of the latter process.

Each field on the board is assigned a memory location with an address number n. All empty fields receive in their address an arbitrary negative number, viz —1. When a field is occupied this number is replaced by a positive one, viz the serial number (k) of the piece in occupation. The changes in the "contents" of the fields, $V(n)$, are recorded by the machine, and this part of the memory forms, as it were, an "operating list".

A field can only be occupied if its address contains —1. The machine can immediately verify this and thus see that the first criterion is satisfied.

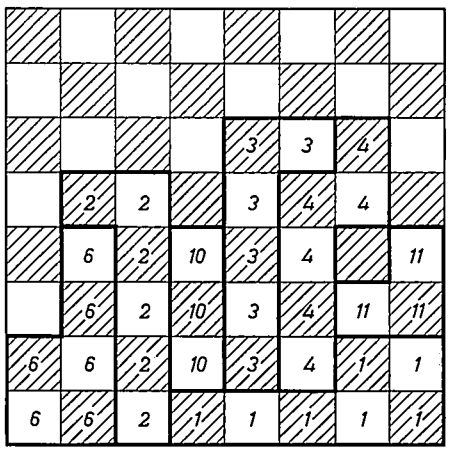

Fig. 3. Situation after the first eight moves have been made in the systematic procedure described. The fields (squares of the board) covered are marked with the number k of the piece covering them. This situation is a dead end, and the last lay must be withdrawn, and even the lay before the last, in order to try further lays.

To enable the computer to check whether the second criterion is satisfied, i.e. that a lay must not project outside the board, we add an extra row of fields along three edges of the board (no extra row is needed along the bottom edge because of our system of laying the pieces). These fields are also assigned memory locations, in which is placed an arbitrary positive number, e.g. +20. The computer can now automatically reject any lay that would cover such a field.

Along the left-hand edge of the board we have added not one but two extra rows of fields. This makes it a very simple matter for the machine to ascertain whether a particular trial field is white or black — it needs this information to decide whether it should try a white or a black lay. With these additions the complete board on which the machine operates is as shown in *fig. 4*; it has 96 fields and the numbering is consecutive along each row, that is $n = 1, 2, \ldots, 96$. It is easily verified that, as a result of this device, all odd fields must be black. The difference between odd and even can immediately be established by a binary machine like the PASCAL by simply checking one bit.

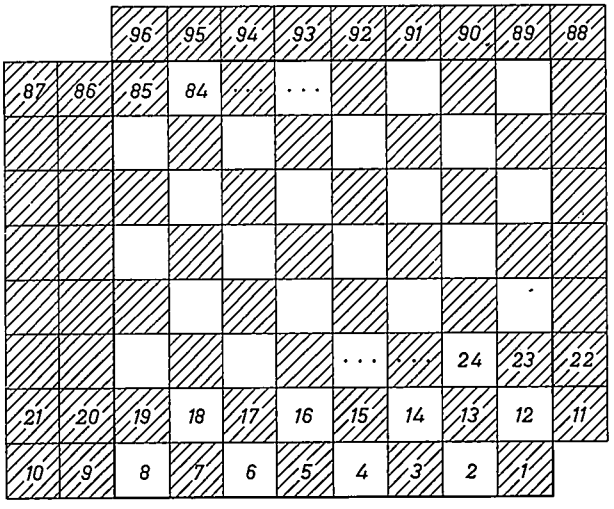

Fig. 4. The board on which operations are made. Extra (fictive) rows of fields are added to the chessboard along three edges so that the machine can test whether a lay projects outside the chessboard and ascertain quickly whether a field should be covered with a black or with a white square of a piece.

We have outlined how the machine handles the two criteria. To "try the lays" the machine must be able to determine which fields are occupied by a lay when it is laid. For this purpose we supply the machine with a description of each lay in the form of a row of numbers, which are as it were the co-ordinates of the squares on the lay. This can be done in various ways. In our case we consecutively number the squares of a lay row by row (serial number r) in the same way as the fields on the board. If for example we take lay $j_b = 1$ (see fig. 2) and place the "master square" of the lay on field 1, then the numbers of the squares will be 1, 2, 3, 4, 5, 12 and 13, and the *differences* between these numbers give the co-ordinates of the lay, which are: 1, 1, 1, 1, 7, 1, 0. The "master square" of the lay of course needs no co-ordinate. The 0 at the end of the row is added for each lay to indicate that there is no other square to come. It is obvious that no matter where the lay is placed on the board, it will always have the same co-ordinates. The 19 + 21 rows of numbers $U_b(j_b,r)$ and $U_w(j_w,r)$ which describe our black and white lays are stored in the form of two lists in the machine's memory. Unlike the earlier mentioned list $V(n)$, these are not "operating lists" but "material lists" with permanent contents.

We can now describe how the machine tries the lay whose turn it is. It starts from the number n_0 of the trial field whose turn it is and it adds one after the other all numbers $U(j,r)$ of the lay. In this way it finds the numbers of the series of fields that will be occupied by the lay. The stored content V at all these addresses is examined. If one positive content is found, the lay is wrong. If all addresses contain negative contents, the latter are all replaced by the number k of the lay. This completes the move.

The solving process as a whole; more record-keeping

The final action just mentioned no longer belongs to the trial procedure proper but to the machine's record-keeping, the writing into list V, and we see at once that the lists mentioned so far are not yet sufficient: another list is evidently needed which will indicate the piece k appertaining to every lay j_b and j_w. We shall call this list $S_b(j_b)$ and $S_w(j_w)$ respectively. Moreover, to keep a complete tally we also need a move counter C (move number = m) and three further lists $M(k)$, $J(m)$ and $N(m)$ (see *fig. 5*), whose functions may be understood as follows.

When the machine has ascertained whether a given trial field n_0 is odd or even, it must proceed to try on it, one after the other, all black or white lays, as the case may be. Before trying a lay j_b or j_w, however, it must see whether the relevant piece is still available. This is the purpose served by list M, which has eleven memory locations corresponding to the piece numbers $k = 1$ to 11. In each location the machine keeps a record of how many pieces with that number are left; at the start of the solving process, location No. 5 contains a 2 and all other locations a 1. Thus, when the machine wants to try a lay, it first makes sure that the relevant location in M does not yet contain a 0.

S_b

j_b	k
1	1
2	2
3	2
.	.
.	.
18	9
19	11

U_b

j_b	$r=$ 1	2	3	4	5	6	7
1	1	1	1	1	7	1	0
2	1	11	11	11	11	0	
3	7	1	1	1	1	0	
.	.	.	.	.	.		
.	.	.	.	.	.		
18	1	10	1	11	0		
19	1	10	0				

S_w

j_w	k
1	1
2	2
3	2
.	.
.	.
20	11
21	11

U_w

j_w	$r=$ 1	2	3	4	5	6	7
1	1	10	1	11	11	11	0
2	1	1	1	1	7	0	
3	11	11	11	11	1	0	
.	.	.	.	.	.		
.	.	.	.	.	.		
20	10	1	0				
21	11	1	0				

V

n	
1	(−1)
2	(−1)
3	(−1)
4	(−1)
5	(−1)
6	(−1)
7	(−1)
8	(−1)
9	+20
10	+20
11	+20
12	(−1)
.	.
.	.
.	.
.	.
95	+20
96	+20

C

m

M

k	
1	(1)
2	(1)
3	(1)
4	(1)
5	(2)
6	(1)
.	.
11	(1)

J

m	j
1	
2	
3	
.	
.	
.	
12	

N

m	n_0
1	
2	
3	
.	
.	
.	
12	

Fig. 5. The complete set of "lists" used by the computer for solving the puzzle. Each list takes up part of the machine's memory. The $S(j)$ lists give for each black or white lay j_b and j_w the number k of the piece. Lists $U(j,r)$ contain the description of all lays, i.e. the co-ordinates of the squares r of each lay. The S and U lists are "material lists" with permanent contents. The others are "operating lists". In V is noted the content of the 96 fields of the board (serial number n); an empty field is given $V=-1$, the fictive fields have a permanent content $+20$, and each field covered by a piece k is given the content $+k$. List M contains information on how many pieces with the number k are still available (0, 1 or 2 — the latter only in the case of piece No. 5). At every move m the machine writes in list N the number n_0 of the trial field whose turn it is, and in list J the number j of the lay whose turn it is (positive for odd n_0, i.e. for black lays, and negative for even n_0, i.e. for white lays). The move number m is in the move counter C.

At every move m, the machine writes in list J, which has 12 locations, the number j of the lay being tried — positive for a black and negative for a white lay. If the lay turns out to be wrong, the machine looks up the relevant number noted in J, raises it by 1 and thereby obtains the address $j+1$ of the next lay to be tried (of the same colour, of course).

The machine is not able to look up the number noted in J so easily as a human operator: it does not "know" offhand which location in J was occupied by the number j. This is where the move counter C comes in. After every successful move the number m in this counter (i.e. in the permanent memory location where the count is made) is raised by 1, and the new number serves as an address for writing and searching in $J(m)$.

It also serves as an address for using the last list, $N(m)$. In this at every move, the machine writes the number n_0 of the trial field on which it is about to try lays. After a successful move the machine finds this number by checking consecutively through all fields in list $V(n)$ and taking the first in which it finds —1.

When the machine is in the situation of having tried *all* lays in a move without result, it can now, by looking up the records, go back on its steps as was mentioned at the beginning. First, it withdraws the lay j tried in the last move and continues that move with the lay $j+1$. In order to do this it decreases the number in the move counter by 1, that is from m to $m-1$. Using the address $m-1$ it looks up in list J the number j of the unsuccessful lay, and in list N the relevant trial field n_0. At the address n_0 in list V it finds the number k of the piece employed. In this location it again writes —1, and likewise in all locations in V which were covered by the unsuccessful lay j (and which it finds from lists S and U). At the same time it turns to list M and raises

by 1 again the wrongly lowered number at the relevant address k. All it now has to do is to raise by 1 the number j of the lay at the address $m-1$ in list J. The move is then continued with $j+1$, as described.

— the numbers k in that list indicating which piece is on which fields. The result is the first chessboard shown in *fig. 6*.

The machine then proceeds as if move 13 had been a failure. It withdraws the last lay, that is the piece

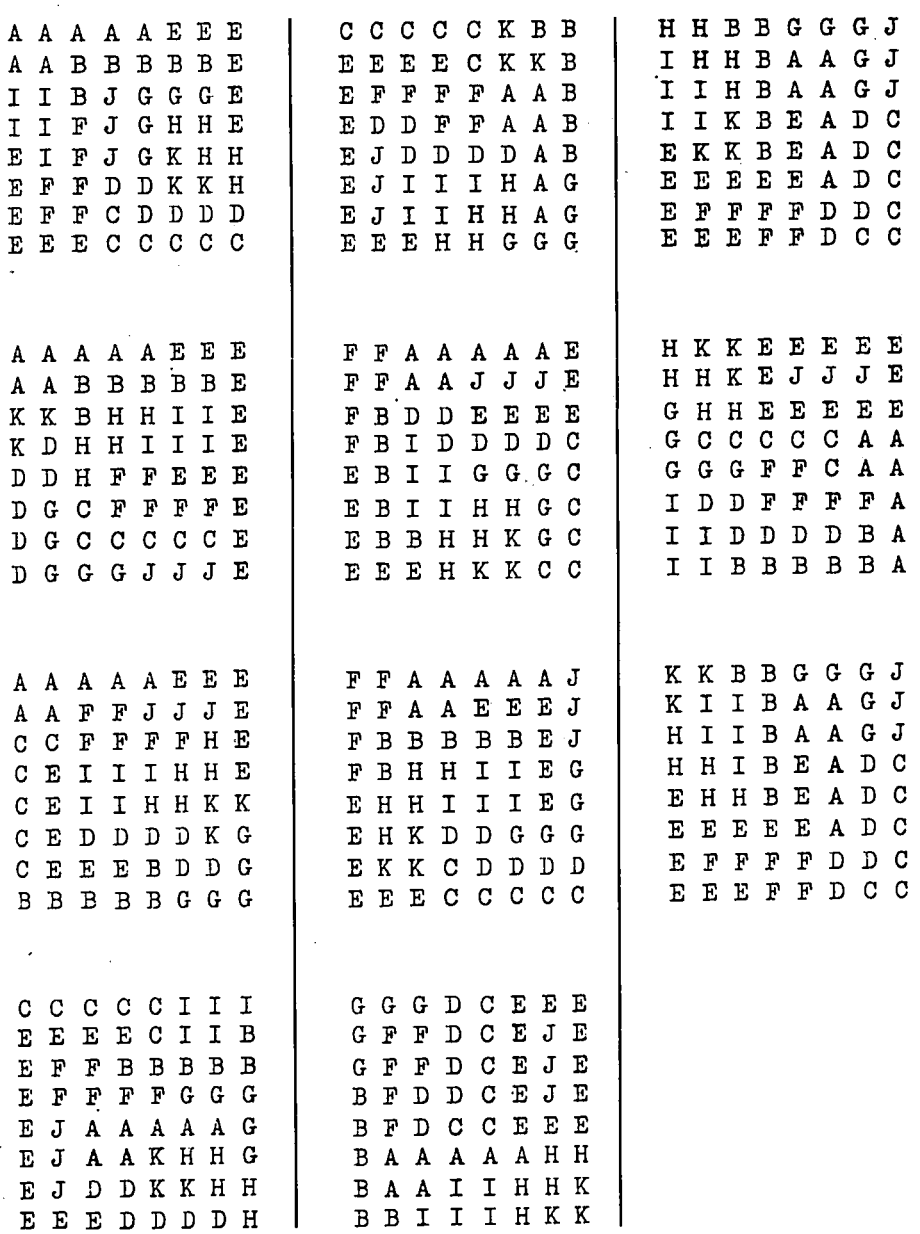

Fig. 6. The eleven solutions of the puzzle typed out by the PASCAL. Instead of the numbers $k = 1$ to 11 we have had the machine print the letters A to K to give a clearer picture. As the typewriter goes from left to right and from top to bottom, the solutions appear here turned 180° with respect to the description given in the text (cf. fig. 3).

Once all twelve successive moves have been made successfully — that is, when a solution is found — the machine is made aware of the fact by the appearance of the number $m = 13$ in the move counter. This is the signal for the machine to type out in chessboard array the complete contents then present in list V laid in move 12, and tries to replace it by another lay, which of course fails. It therefore cancels move 12 and tries move 11 in another way, and so on.

In this situation the computer's actions seem to be rather "unintelligent", for it is perfectly plain that the withdrawal of one piece or even of two pieces will

not lead to another solution. But the machine can do no more than rigidly keep to the system, which guarantees completeness. We must just accept the few microseconds wasted on moves which to us seem pointless.

After retracing its steps far enough, the machine will have success with a new series of moves. In this way, as mentioned at the beginning, it successively finds *eleven* solutions. These are all shown in fig. 6.

Even after it has typed out the last possible solution, the computer goes on withdrawing moves and trying new ones. It is evident that it has then to go back farther and farther, and upon every move withdrawn it must reduce the number m in the move counter by 1. When all moves have been withdrawn, the number $m = 0$ appears in the counter, indicating to the machine that all possibilities have been tried and that it can stop.

The PASCAL takes eight minutes to complete the whole solving process, including the return to $m = 0$.

Summary. Electronic computers can be used for solving many problems that — at least in the first instance — have nothing to do with computing in the strictly arithmetical sense. As an example of such a problem a chessboard puzzle is described which served as an object of study for programming the PASCAL. The trial procedure worked out for solving the puzzle involves systematization and record-keeping, for which comprehensive tallies ("material lists", "operating lists", etc.) are stored in the computer's memory. It takes the PASCAL eight minutes to complete the whole solving process, in which it tries some 3 million possibilities and finds eleven essentially distinct solutions.