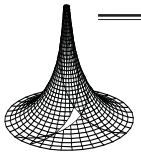


5/1 (2007), 183–193
tmcs@inf.unideb.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Brute force on 10 letters

Zoltán Kovács and István Hudi

Abstract. We deal with two problems in the set of 10-character-long strings. Both problems can be solved by slightly different methods, but our approach for each is brute force. As we point out, there can be differences in effectivity even in different brute force algorithms. As an additional result, we answer an open question of Raymond Smullyan's.

Key words and phrases: puzzles, brute force, C/C++ language, graph algorithms.

ZDM Subject Classification: K30, F90.

1. Self descriptive numbers

My university professor, back in the 90's, raised the following question: Is there a decimal number $\overline{a_0a_1a_2a_3a_4a_5a_6a_7a_8a_9}$ which contains exactly a_0 pieces of zeros, a_1 pieces of ones, a_2 pieces of digit "2", and so on. He went on to say, however, that "the solution should fit on half a page".

I liked these type of puzzles, and even though it was a challenge to explain it in a nutshell, was quite satisfied that after some experimenting I was able to come up with a relatively short solution. My professor kept on shaking his head, however, saying that my solution was too long. I still wonder what he meant when he said that he was looking for a "short" solution.

Searching the Internet, I found that other people were also challenged to find a short explanation as to why the number 6210001000 is the (only) answer to this question, or to find this number by a short process.

The use of computer program in this era makes it easier to solve such problems. Of course, it can still be interesting to determine how one should write an appropriate program. A brute force technique which covers all the possible 10^{10} numbers seems to be feasible if one uses a modern computer and a compilable programming language. The following computer program is just 15 lines of C code (272 characters) and provides the answer within an hour[1] on a desktop PC:

```
int main() {
    unsigned long long n;
    char a[ 11 ];
    char f[ 11 ];
    int i;
    for ( n = 1000000000ULL; n < 10000000000ULL; ++n ) {
        sprintf( a, "%lld", n );
        strcpy( f, "0000000000" );
        for ( i = 0; i < 10; ++i )
            ++f[ a[ i ] - '0' ];
        if ( strcmp( a, f ) == 0 )
            printf( "%s\n", a );
    }
}
```

The programming language and the applied technique are important here: other methods may increase running time and slow down the process too much. ([1] describes a Mathematica code which is more than 500 characters.)

Which is easier or better, or more beautiful? It is difficult to know if there is a decision of Solomon concerning this question. Paul Erdős's idea that "The Book" contains the most elegant proofs of the mathematical theorems, kept by God, has been found recently a printed body by Aigner, Ziegler and Hofmann [2, 3]. Here we refer to the short solutions of other authors [4, 5, 6, 7].

A strange side effect of the growing technological background is that "to be short" in time for calculation, or in addition, to be short in time of programming, may be more elegant than spending many hours of thinking and gathering information to find the only case which fulfills the demands of a problem. Of course, this can be only true for *finite* problems. Clearly, we only have to check a finite amount of possibilities here, namely 10^{10}, which can be easily decreased to much less.

In fact, the possible solutions for the suitable n should be not less than $1\,000\,000\,000$ because of the leading zeros. Further considerations show that n

[1] 51:43 time on an Ubuntu Linux 6.10 system, Intel Pentium 4, 3 GHz, gcc 4.1.2 – no optimization

should be greater than 5 000 000 000 which immediately saves the half of the calculations. (See [8] for details.) One can easily check that $a_7 = a_8 = a_9 = 0$ must also stand and $a_0 \in \{5, 6\}$ is necessary, too. These facts give even smaller running time: modifying the above mentioned, underlined parts to `5000000000ULL`, `7000000000ULL`, `n+=1000`, respectively, we gain the result within 1 second!

However, to find a fast, efficient and smart computer program, is not always easy, even for finite problems. We show two completely different brute force algorithms in the next section for another problem, and we will get two completely different results in time and smartness.

Brute force methods usually cannot help in infinite problems. The Goldbach conjecture ([9]) can be checked for an arbitrary m integer using a rather easy method if someone has an array with all the prime numbers not greater than m. But there are infinitely many m integers to check and this fact causes the Goldbach problem to remain unsolved.

2. The mystery of the Monte Carlo lock

Raymond Smullyan introduces the story of Inspector Craig of Scotland Yard in his great [10] novel. Craig finds a possible key code for the Monte Carlo lock with his colleagues, Dr. Ferguson and McCulloch.

There seems to be a cultural hunger for new quizzes. Today's mathematicians are often asked to develop new puzzles to entertain TV audience or other groups (e.g. Internet surfers). Let us give two examples: Sudoku ([11]) and Perplex City ([12]), which are popular not only among young people but the older ones too, employ a wide group of mathematicians for months or even years. The submitted problems are sometimes old ones clothed with a new look and style, but others may be new ones. Final decisions to find the best player in such games are usually based on brand-new problems.

The authors were asked to help to develop such a new problem. Smullyan's Monte Carlo story also offered an open question. Smullyan gives the information that he can open the lock with a 10-letter-long code, but does not know if there is a shorter solution. Hereunder we try to find a shorter one if there exists any.

The problem is as follows: Consider the W set of words on the $A \supseteq \{Q, L, V, R\} = A_4$ alphabet and the relation $\rightarrow$ on W. For an arbitrary $x \in W$ let $r(x)$ denote the reversed word (i.e. which contains the same letters in reversed order). We know that $\rightarrow$ has the following properties:

(1) For any $x \in W$, $x \to QxQ$.

(2) $y \to x$ implies

   (a) $Qy \to Lx$,

   (b) $r(y) \to Vx$,

   (c) $yy \to Rx$.

The lock can be opened with the word w if and only if $w \to w$.

In [10], Smullyan describes different methods on how one can get closer to find the solutions of such puzzles. That chapter of his book is also an introduction to Gödel's theorems. He shows that the word $RVLVQRVLVQ$ is a solution but he does not tell more details about other solutions. In this paper we show a method which can give us more solutions in reasonable time.

2.1. The first approach

To explain this problem to a non-expert audience, some examples should be shown. E.g., $C \to QCQ$ stands for any letter C, if C is an allowed letter ($C \in A$).

We assume that only the A_4 alphabet is used. It is clear that all possible solutions can be derived to use only A_4. So to show a really useful example, consider $V \to \boldsymbol{Q}V\boldsymbol{Q}$, and now using the second properties, $\boldsymbol{Q}V \to \boldsymbol{L}QVQ$, $V \to \boldsymbol{V}QVQ$ and $VV \to \boldsymbol{R}QVQ$ also stand. Now we gained three new relations that generate further ones as well:

(1) From $QV \to LQVQ$ we gain $\boldsymbol{Q}QV \to \boldsymbol{L}LQVQ$, $VQ \to \boldsymbol{V}LQVQ$ and $QVQV \to \boldsymbol{R}LQVQ$.

(2) From $V \to VQVQ$ we gain $\boldsymbol{Q}V \to \boldsymbol{L}VQVQ$, $V \to \boldsymbol{V}VQVQ$ and $VV \to \boldsymbol{R}VQVQ$.

(3) Finally, from $VV \to RQVQ$ we gain $\boldsymbol{Q}VV \to \boldsymbol{L}RQVQ$, $VV \to \boldsymbol{V}RQVQ$ and $VVVV \to \boldsymbol{R}RQVQ$.

And so on, using the second properties, arbitrary many relations can be found. Let us call each such three steps a *turn*. In each turn three new relations are gained at most which means that the number of relations in the consecutive turns are 3, 9, 27, ..., 3^n at most. This leads to an exponential calculation.

Fortunately, we only search for solutions that are shorter than Smullyan's published one, so there can be a limit for our calculation. It can be also seen that the length of words on each side of the relations follow a monotonic behavior. In addition, the right side is increasing in a strict way. Our first implementation, a C++ code, 36 kilobytes (with many comments), works with string data types,

can deduce all possible relations from a starting word if an upper limit for the length of both sides are given. It must be emphasized that a starting word must be included by the user. The following output shows the running behavior of the program:

```
$ make
g++ -o montecarlo1 my_str.c rule.cpp struct.cpp
$ ./montecarlo1
Few number of arguments:1
Usage: ./montecarlo1 StartingWord MaximalWordLength.
MaximalWordLength is called 'maxlen_glb' in the source of the program.
$ ./montecarlo1 V 4
The turn, numbered as 0, is finished.
The turn, numbered as 1, is finished.
The turn, numbered as 2, is finished.
The turn, numbered as 3, is finished.
The turn, numbered as 4, is finished.
After 5 turns:
   QV LQVQ
    QV QQVQ
  VQ LQVQ
   VQ QQVQ
 V QVQ
   V VQVQ
     QV QVQQ
      VQ QVQQ
    VV QVVQ
  VV RQVQ
List of invariant words:
```

The program intelligently filters out the repeated relations and clews up all the possible cases. In the above mentioned example there is no invariant word found. I.e. no $w \to w$ patterns occurred in the deduced word pairs.

All we had to do after creating this tool was to run it for all possible starting words having the maximal length of 10 letters. After approximately 3 days of full running time, we concluded that there is no shorter solution than Smullyan's one, however there is a second one which is also 10 letters long: $VRLVQVRLVQ$. To check this, we should see that

(1) from the first property, $VRLV \to \boldsymbol{Q}VRLV\boldsymbol{Q}$.

(2) From 2b, $VLRV \to \boldsymbol{V}QVRLVQ$,

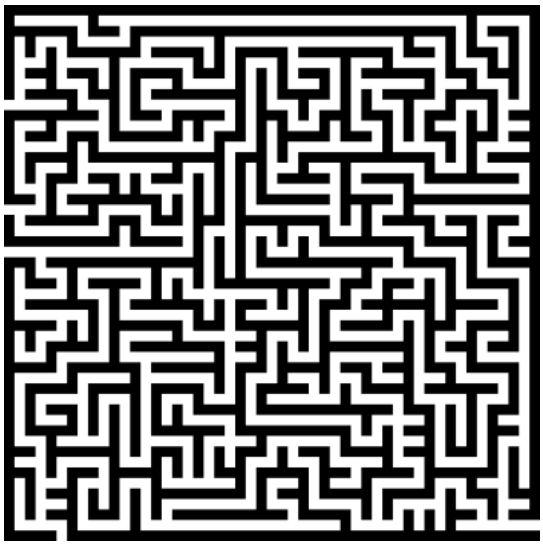
(3) from 2a, $\boldsymbol{Q}VLRV \to \boldsymbol{L}VQVRLVQ$,

(4) from 2c, $QVLRVQVLRV \to \boldsymbol{R}LVQVRLVQ$,

(5) and finally from 2b again, $VRLVQVRLVQ \to \boldsymbol{V}RLVQVRLVQ$.
Our program code is available on [13].

2.2. The second approach

The thoughts above show that we search for a way in a graph from all existing $x \to QxQ$ entry points to find a possible $w \to w$ end in fact. It looks like a labyrinth which has many entrances but just a very few exits. So our second approach is based on the well known maze solving strategy which also occurs in artificial intelligence, namely in heuristic methods: finding a route between two distant points of a "big" graph. The idea is to begin the search at the end of the maze and continue the search towards the entrances. (The following figure shows a maze with several entrances, but only one exit. This picture is a modified version of an auto-generated output published on [14].)



Now our consideration follows. Assume that we have a solution w. This possible end point can be $w = w_n = Lw_{n-1}$ or $w_n = Vw_{n-1}$ or $w_n = Rw_{n-1}$ where each possible w_{n-1} (as right side of a relation) belongs to an appropriate v_{n-1} (as left side of that relation), $v_{n-1} \to w_{n-1}$. To see an example, take Smullyan's solution, $w_n = RVLVQRVLVQ$. Now clearly $w_{n-1} = VLVQRVLVQ$ (with $v_{n-1} = RVLVQ$) and the last turn had to be based on 2c. But now this consideration can be used again: $w_{n-2} = LVQRVLVQ$ (with $v_{n-2} = QVLVR$) and

this turn had to be based on 2b. Respectively we get $w_{n-3} = VQRVLVQ$ (with $v_{n-3} = VLVR$), based on 2a, and $w_{n-4} = QRVLVQ$ (with $v_{n-4} = RVLV$), based on 2b. Now the starting letter of w_{n-4} is none of L, V or R, but $w_{n-4} = QsQ$ for some $s \in W$. It means that we can only use the first property now. So $s = RVLV = v_{n-4}$ which means that we can deduce the solution w with the starting word s. Summarizing the above, in general, a solution w can be deduced from a starting word s if and only if finally we reach the first property, $w_1 = QsQ$, while using the steps described.

Obviously for a given candidate w this is a reasonably fast process to decide whether w is a solution or not. There is nothing exponential in this algorithm, it is purely linear, i.e. in at most n steps we can find out if $w \to w$ stands for w or not. In each (k^{th}) step we can decide if the ending w is still valid or not, depending on the starting letter of w_{n-k} and the possibility of constructing v_{n-k}. The final step must always be based on the first property. Our second implementation, written in C, exactly 1500 bytes of code (without comments), runs within 1 second and gives both solutions for the word length at most of 10. In addition, we also gain additional solutions within 20 minutes:

- *VLRVQVLRVQQ*, *VLVRQVLVRQQ* (11 letters),
- *RVVVLVQRVVVLVQ*, *VRVVLVQVRVVLVQ*, *VVRVLVQVVRVLVQ*, *VVVRLVQVVVRLVQ*, *RVLVVVQRVLVVVQ*, *VRLVVVQVRLVVVQ* (14 letters)
- and *VLRVVVQVLRVVVQQ*, *VLVRVVQVLVRVVQQ*, *VVVLRVQVVVLRVQQ*, *VLVVRVQVLVVRVQQ*, *VVVLVRQVVVLVRQQ*, *VLVVVRQVLVVVRQQ* (15 letters)

if we multiply the underlined constant ($\sum_{k=1}^{10} 4^k+1$) by 1000 in the program code:

```
#define MAX_LENGTH 30
#define SHOW_ROUTE 0

int is_double( char *a ) {
    int i, l, m;
    l = strlen( a );
    if ( l & 1 )
        return 0;    // no
    m = l / 2;
    if ( strncmp( a, a + m, m ) == 0 )
        return 1;    // yes
    return 0;        // no
}
```

```
void reverse( char *a ) {
    int i, l, m;
    char c;
    l = strlen( a );
    m = l / 2;
    for ( i = 0; i < m; ++i ) {
        c = a[ i ];
        a[ i ] = a[ l - i - 1 ];
        a[ l - i - 1 ] = c;
    }
}

int is_correct( char *a ) {
    char *b = malloc( strlen( a ) + 1 );
    strcpy( b, a );
    void *b_ = b;
    int c = 1;    // continue?
    while ( c == 1 ) {
        if ( SHOW_ROUTE == 1 )
            printf( "%s %s\n", a, b );
        c = 0;
        if ( b[ 0 ] == 'L' && a[ 0 ] == 'Q' ) {
            a++;
            b++;
            c = 1;
            continue;
        }
        if ( b[ 0 ] == 'V' ) {
            reverse( a );
            b++;
            c = 1;
            continue;
        }
        if ( b[ 0 ] == 'Q' && b[ strlen( b ) - 1 ] == 'Q' ) {
            b++;
            b[ strlen( b ) - 1 ] = '\0';
            if( strcmp( a, b ) == 0 )
                c = 2;    // solution found
            continue;
        }
        if ( b[ 0 ] == 'R' && is_double( a ) == 1 ) {
            b++;
            a[ strlen( a ) / 2 ] = '\0';
            c = 1;
            continue;
```

```
        }
    }
    free( b_ );
    return c;
}

const char *c[4] = { "L", "V", "Q", "R" };

void nth_case( long n ) {
    char a[ MAX_LENGTH ], a_[ MAX_LENGTH ];
    int n_ = n;
    strcpy( a, "" );
    do {
        strcat( a, c[ n % 4 ] );
        n /= 4;
    } while ( n > 0 );
    strcpy( a_, a );
    int answer = is_correct( &a );
    if ( answer == 2 )
        printf( "case %d: %s\n", n_, a_ );
}

main() {
    long n;
    for ( n = 0; n < 1398101000; ++n )
        nth_case( n );
}
```

Finally we leave to the reader to try to answer the following questions:

(1) Are there infinitely many solutions?

(2) Can we show a solution which contains a letter outside of A_4?

(3) Is there a way to speed up this program code even more?

3. Conclusion

Brute force algorithms can be smart and quick despite the fact that we usually shrink from considering all cases. Sometimes a well written brute force code can save many hours of case investigation. However, finding a really good technique among the brute force alternatives may still be a challenge for today's mathematicians and programmers. The two example illustrations above can also give ideas

for today's teachers in mathematics and computer science to show the students when and how it is suggested to use brute force.

A general discovery of the current state of mathematical problem solving is that careful combination of human thinking and effective computer algorithms may increase effectivity in research. Today, 30 years after the personal computer boom, many university professors still avoid using computers in their teaching courses to protect students of possible abuse of automatism. Indeed, human thinking remains a creative process and it cannot be replaced with computer methods. In addition, the opposite group of teachers sometimes use computers also in such cases when pure thinking has much more advances in developing thinking skills. An ultimate example of such puritanism is Jakob Steiner who held his geometry lectures in a dark hall to build up the creative fantasy of his audience.

Our investigations show that finding a harmony between the two parties is not always easy, but it can really be fruitful. As we already have been having the technological background for three decades, we should still learn when and how to use it to gain all its advantages without stepping back from any points of view, especially in education. Our two illustrations above confirm that a "dumb" computer algorithm strengthened by crafty human consideration can be a step forward in both researching and stimulating student thinking as well.

4. Acknowledgments

The authors are grateful to Géza Makay for his kind advices which helped us making our codes even shorter and more effective. We also thank Dániel Molnár for his gentle encouragement.

References

[1] The Online Encyclopedia of Integer Sequences, A108551 (Self-descriptive numbers in various bases represented in base 10), updated June 7 2005, `http://www.research.att.com/~njas/sequences/?q=A108551`.

[2] M. Aigner, G. M. Ziegler, K. H. Hofmann, *Proofs from THE BOOK*, 3/e, Springer, 2004, ISBN 3-540-40460-0.

[3] `http://en.wikipedia.org/wiki/Proofs_from_THE_BOOK`.

[4] `http://en.wikipedia.org/wiki/Self-descriptive_number`.

[5] `http://planetmath.org/encyclopedia/SelfDescriptiveNumber.html`.

[6] http://brainyplanet.com/index.php/Self%20Ref%20Solution.

[7] http://www.mateklap.hu/forum.php?cmd=olv&f=16&eltol=3 (in Hungarian).

[8] http://begghilos2.ath.cx/~jyseto/Academia/Math-Problem-1.php.

[9] http://mathworld.wolfram.com/GoldbachConjecture.html.

[10] R. Smullyan, *The Lady or the Tiger? and Other Logic Puzzles: Including a Mathematical Novel That Features Godel's Great Discovery*, Times Books, 1992, ISBN 0812921175.

[11] http://www.sudokupuzz.com.

[12] http://www.perplexcity.com.

[13] http://www.math.u-szeged.hu/~kovzol/montecarlo1.zip.

[14] http://www.billsgames.com/mazegenerator/.

ZOLTÁN KOVÁCS
BOLYAI INSTITUTE
UNIVERSITY OF SZEGED
ARADI VÉRTANÚK TERE 1.
H–6720 SZEGED
HUNGARY

E-mail: kovzol@math.u-szeged.hu

ISTVÁN HUDI
PALÁNK 1.
H–6720 SZEGED
HUNGARY

E-mail: h1963istvan@freemail.hu

(Received March, 2007)