Workshop: Grammars, Regular Expressions, and Finite-State Automata

One central theme of WFNMC 2010 is bridging the gap between mathematics and computer science, and mathematics competitions. It is possible to create interesting and original problems involving grammars, regular expressions, and finite-state automata – all ideas important in theoretical computer science.

Many problem-writers who do not routinely teach either discrete mathematics or theoretical computer science courses may not have seen these ideas before. The workshop would begin with a brief overview (perhaps 30 minutes) of these concepts, followed by the presentation of solutions to a few typical problems.

Then participants would be given a selection of problems from these various areas and spend time trying to solve them. Included problems come from many years experience teaching discrete mathematics and problem-solving courses. The hope is that participants would get a feel for the nature of these problems and the problem-solving skills needed to solve them, and perhaps devise ways to incorporate these ideas into novel and accessible problems.

The workshop will be organized by myself (Vince Matsko, mathematics faculty member of the Illinois Mathematics and Science Academy, Aurora, IL, USA). The number of participants is not limited. All materials used for the workshop will be available online, so that if more participants attend than are expected, those with computers can access the materials online. The workshop could be up to two hours in length; the idea is to get participants actually solving problems.

For those unfamiliar with these ideas, I am also submitting three problems related to grammars for the Congress Problem Book. I include these problems below so that the reader may get an idea of the type of problem to be discussed in the workshop.

Sincerely,

Vince Matsko

These three problems are of increasing difficulty. The first is a restatement of a familiar number theory problem, while the second and third involve more abstract reasoning. Those familiar with finite-state automata may well recognize the idea behind Problem 2, while a solution to Problem 3 is far from obvious.

Problems 1 and 2 may be written as multiple choice questions as indicated, while Problem 3 would be better suited to a free-response problem.

1. Consider the following method for creating strings of 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{ll} x \mapsto 111x & (1) \\ x \mapsto 11111x & (2) \\ x \mapsto 111 & (3) \end{array}$$

The process ends when only ones remain. For example, we may obtain a string of 16 ones using

$$x \overset{2}{\mapsto} 11111x \overset{1}{\mapsto} 11111111x \overset{2}{\mapsto} 1111111111111x \overset{3}{\mapsto} 1111111111111111.$$

How many strings of one or more 1's CANNOT be obtained by applying these rules?

Solution:

Since the process can only end by applying (3), we cannot produce the strings 1 or 11.

Applying rules (1) and (2) is reminiscent of the postage stamp problem. It is well known that the maximum number of ones which cannot be produced by (1) and (2) only is $(3-1)(5-1)-1=7$, so that a quick inspection shows that only strings of 1, 2, 4, or 7 ones cannot be obtained by using (1) and (2). Since the process ends with an application of (3), strings of 4, 5, 7, or 10 ones cannot be obtained.

Thus, only strings of 1, 2, 4, 5, 7, or 10 cannot be produced by using the above rules. Hence there are six such strings.

ALTERNATE MULTIPLE CHOICE PROBLEM:

How many strings of one or more 1's CANNOT be obtained by applying these rules?

(A) 4 (B) 6 (C) 8 (D) 10 (E) 15

2. Consider the following method for creating strings of 0's and 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{ll} x \mapsto 1 & (1) \\ x \mapsto 0x & (2) \\ x \mapsto 1y & (3) \\ y \mapsto 0z & (4) \\ y \mapsto 1x & (5) \\ z \mapsto 0 & (6) \\ z \mapsto 0y & (7) \\ z \mapsto 1z & (8) \end{array}$$

The process ends when only 0's and 1's remain. For example,

$$x \overset{3}{\mapsto} 1y \overset{4}{\mapsto} 10z \overset{8}{\mapsto} 101z \overset{6}{\mapsto} 1010.$$

Thus, x is replaced by $1y$ using (3), and then the symbol y is replaced by $0z$ using (4), and so on, until the last occurrence of x, y, or z is replaced by a 0 or 1 as described by the rules.

Describe all binary strings which can be produced by these rules.

Solution:

The solution may be obtained by examining what it means for a string to end in x, y, or z. The rules make sense if we interpret a string ending in x to mean "the string created so far, interpreted as a binary number, is congruent to 0 mod 3." Similarly, that a string ends in y means "the string created so far, interpreted as a binary number, is congruent to 1 mod 3," and that a string ends in z means "the string created so far, interpreted as a binary number, is congruent to 2 mod 3."

Each rule is consistent with this interpretation. For example, (5) indicates that if the string so far ends in y (the string is congruent to 1 mod 3), adding a 1 on the end multiplies the number by 2 and adds 1, leaving a string which is congruent to $2 \cdot 1 + 1 \equiv 0 \bmod 3$, so that it must now end in x.

Now notice that (3) and (7) leave the string ending in y, so that after applying one of these rules, the string so far may be interpreted as a binary number which is equivalent to 1 mod 3. Since (1) and (6) are similar to (3) and (7) except for adding the trailing "y," we see that when the last x, y, or z is replaced by a 0 or a 1, a binary number congruent to 1 mod 3 remains.

With the same interpretation of the above rules, it is not difficult to see that given a binary number congruent to 1 mod 3, a sequence of rules may be found which produce this number. For example, we obtain 10011 as follows:

$$x \overset{3}{\mapsto} 1y \overset{4}{\mapsto} 10z \overset{7}{\mapsto} 100y \overset{5}{\mapsto} 1001x \overset{1}{\mapsto} 10011.$$

It is not difficult to make the above arguments rigorous by using induction on the length of the binary strings produced. Thus, the rules produce precisely those binary strings which, when interpreted as binary numbers, are congruent to 1 mod 3.

Alternate multiple choice problem:

Which of the following numbers, interpreted as a binary string, can be obtained using the above rules?

(A) 5^{2011} (B) 6^{2011} (C) 7^{2011} (D) 8^{2011} (E) 9^{2011}

3. Consider the following method for creating strings of 0's and 1's. Begin with the symbol a, and apply the following replacement rules as many times as desired and in any order.

$$\begin{array}{ll} a \mapsto 1a1 & (1) \\ a \mapsto 11a & (2) \\ a \mapsto 1 & (3) \\ 11a1 \mapsto b0 & (4) \\ b \mapsto b0 & (5) \\ b000 \mapsto a & (6) \end{array}$$

The process ends when only 0's and 1's remain. For example, the string 11 may be obtained as follows:

$$a \overset{1}{\mapsto} 1a1 \overset{2}{\mapsto} 111a1 \overset{4}{\mapsto} 1b0 \overset{5}{\mapsto} 1b00 \overset{5}{\mapsto} 1b000 \overset{6}{\mapsto} 1a \overset{3}{\mapsto} 11.$$

Describe all binary strings obtained by these rules.

Solution:

First, notice that it is not possible using these rules to produce a string beginning with a 0. A brief induction argument on the length of strings containing the symbols a, b, 0, and 1 establishes this.

It is, however, possible to produce any string beginning with a 1. First note that it is possible to replace the symbol a with $a1$ as follows:

$$a \overset{1}{\mapsto} 1a1 \overset{1}{\mapsto} 11a11 \overset{4}{\mapsto} b01 \overset{5}{\mapsto} b001 \overset{5}{\mapsto} b0001 \overset{6}{\mapsto} a1.$$

It is also possible to replace the symbol a with $a0$, as follows:

$$\begin{aligned} a &\overset{2}{\mapsto} 11a \overset{1}{\mapsto} 111a1 \overset{4}{\mapsto} 1b0 \overset{5}{\mapsto} 1b00 \overset{5}{\mapsto} 1b000 \overset{5}{\mapsto} 1b0000 \\ &\overset{6}{\mapsto} 1a0 \overset{1}{\mapsto} 11a10 \overset{4}{\mapsto} b00 \overset{5}{\mapsto} b000 \overset{5}{\mapsto} b0000 \overset{6}{\mapsto} a0. \end{aligned}$$

Thus, we may repeatedly add either a 0 or 1 at the end of any string. A final application of (3) finishes the process and adds a 1 to the beginning of the string. Thus, any string beginning with a 1 can be produced. (Note that a single 1 may be produced just by an application of (3).)