The PRIMES 2014 Computing Problem Set

Dear PRIMES applicant!

This is the PRIMES 2014 Computing Problem Set. Please send us your solutions as part of your PRIMES application by December 1, 2013. For complete rules, see http://web.mit.edu/primes/apply.shtml

Note that this set contains two parts: "General Math problems" and "Computer Science problems." Please solve as many problems as you can in both parts.

For the General Math problems, you can type the solutions or write them up by hand and then scan them. Please attach your solutions to the application as a PDF (preferred), DOC, or JPG file. The name of the attached file must start with your last name, for example, "smith-solutions." Include your full name in the heading of the file. Please write not only answers, but also proofs (and partial solutions/results/ideas if you cannot completely solve the problem).

For the Computer Science problems, see instructions at the beginning of that part.

Besides the admission process, your solutions will be used to decide which projects would be most suitable for you if you are accepted to PRIMES.

You are allowed to use any resources to solve these problems, *except other people's help*. This means that you can use calculators, computers, books, and the Internet. However, if you consult books or Internet sites, please give us a reference.

Note that posting these problems on problem-solving websites before the application deadline is not allowed. Applicants who do so will be disqualified, and their parents and recommenders will be notified.

Note that some of these problems are tricky. We recommend that you do not leave them for the last day. Instead, think about them, on and off, over some time, perhaps several days. We encourage you to apply if you can solve at least 50% of the problems. [1]

Enjoy!

[1] We note, however, that there will be many factors in the admission decision besides your solutions of these problems.

Math problems

Problem G1. Let $n > 2$ be an integer. Find explicitly a nonzero polynomial P of degree $2n$ with integer coefficients and leading coefficient 1 such that $P(2^{1/2} + 2^{1/n}) = 0$. How many real roots does P have?

Problem G2. Each minute, a drunkard walks one step to the right with probability 1/3, two steps to the right with probability 1/3, and one step to the left with probability 1/3. He is initially k steps to the right from a cliff. What is the probability that he will fall off? (If he is 0 steps away from the cliff, he falls off.)

Problem G3. Does the equation

$$x^3 + 2x - y^2 = 1$$

have integer solutions?

Problem G4. John's secret number is between 1 and 2^{16}, and you can ask him "yes or no" questions, but he may lie in response to one of the questions. Explain how to determine his number in 21 questions.

Problem G5. Let n be a fixed positive integer, and r a fixed positive number. Show that the number of positive integer solutions of the equation

$$\frac{1}{x_1} + ... + \frac{1}{x_n} = r$$

is finite.

Hint. Consider the smallest of the x_i.

Problem G6. A round table has n seats. How many ways are there to seat k people at this table, so that no two of them sit next to each other? (two seatings are viewed as the same if each person sits on the same chair under both seatings).

Problem G7. On a round table of diameter 2 feet there are 132 coins of diameter 1 inch. Show that one can put one more such coin on the table without overlap with the other coins.

Computer science problems.

About the problems. The theme of this year's problems is cellular automata. A cellular automaton is a virtual system that consists of a grid of *cells*. Each cell has a *state* such as *on* or *off*, equivalently 0 or 1. At any point in time a new state of a cell is determined from its currents state and a state of its neighbors. For instance, on a two-dimensional grid a rule may be "if the cell itself is *off*, and two of its neighbors are *on*, then the cell gets the state *on*, otherwise it stays *off*." An automaton starts with an *initial state* (i.e. all cells get initial values), and then proceeds changing its state according to the rules. The study of patterns in automata behavior (Such as: Does it cycle through the same states? Does the number of cells with *on* value increase?) is an area of computer science and mathematics. Cellular automata have applications in cryptography and in other areas of computing and are used to model physical, biological, and even social processes. Probably the most well-known cellular automaton is Conway's game of life.

Recommended readings: Stephen Wolfram, *A New Kind of Science* (Wolfram Media, 2002); Joel Schiff, *Cellular Automata: A Discrete View of the World* (Wiley-Interscience, 2008).

What you need to do. For these problems we ask you to write a program (or programs) to model cellular automata. You may use any programming language you want. It is best to implement each problem as a separate function so that we can run them separately. We will be looking for the following in your submissions:

- Correct code that we can run. You need to send us all your code files, including the header files for languages like C++. If you are using standard libraries, make sure to include all "import" statements, as required by the language you are using. Make sure to send the files under the correct names, including the file extension (.java, .c, etc).
- Test data for your code that you have used (you can write it in comment or in a separate file). Make sure to test your code well – you don't want it to fail our tests!
- Code documentation and instructions. In the beginning of each file specify, in comments:
  (1) Your name.
  (2) Problem number(s) in the file. If you have a file with "helper" functions, mark it as such.
  (3) The *programming language*, including the *version* (Java 1.6 or 1.7, for instance), the *development framework* (such as Visual Studio) that you used, unless you were using just

a plaintext editor (notepad, emacs, etc), and the *platform* (such as Windows, Mac, Linux)

(4) Instructions for running your program (how to call individual functions, pass the input (if any), etc), either in comments in your program file or as a separate file, clearly named. Your program may get input from the user (i.e. it asks to enter some data and then reads it) or you may store the data in specific variables within your program. You need to clearly explain how to input or set the data.

(5) Some of your code may be commented out if it is not used in the final run of your program. Make sure it is clear what needs to be uncommented to run code for each of the problems.

(6) All of your test data.

(7) If you were using sources other than the ones listed here (i.e. textbooks, online resources, etc) for ideas for your solutions, please clearly credit these contributions. This is a courtesy to work of others and a part of ethics code for scholars.

- Clear, understandable, and well-organized code. This includes:

(1) Clear separation between problems; comments that help find individual problems and explain how to run the corresponding functions.

(2) Breaking down code into functions that are clearly named and described (in comments), using meaningful names for variables and function parameters. Your code should be as self-explanatory as possible. While using comments helps, naming a variable `average` is better that naming it `x` and writing a comment "x represents the average".

(3) Minimization of code repetition. Rather than using a copy-paste approach, use functions for repeated code and reuse these functions.

(4) Using well-chosen storage structures (use an array or a list instead of ten variables, for instance) and well-chosen programming constructs (use loops or recursion when you can, rather than repeated code).

(5) While we are not asking for the fastest program (it's better to make it more readable), you should avoid unnecessary overhead.

In this problem set we are considering a one-dimensional cellular automaton. To simplify the problem, we consider a finite number of

cells c_i, $1 \leq i \leq n$, where n is the length of the automaton. The *state* of a cell at a given moment is the symbol in it at that moment. All possible symbols that can be in any cell of an automaton make up its *alphabet*. We only consider automata with an alphabet that include all or some of the symbols $0, 1, 2, 3$.

Each automaton comes with a set of rules that determine the new symbol in a cell based on its current symbol and the cell to its immediate right (its *right neighbor*). The rightmost cell c_n in an automaton uses the leftmost cell c_1 as its right neighbor (so you can think of an automaton as a circle). Rules are of the form

$$a_1\ a_2 \to a_3$$

which means that every cell c_i that has a symbol a_1 and whose right neighbor c_{i+1} (or c_1 if $i = n$) has a symbol a_2 gets the value a_3 as its next state. For instance, the rule $0\ 2 \to 1$ means that every cell with the state 0 whose right neighbor has the state 2 gets 1 as its next state.

If no rule for a pair of symbols is specified, the next state is set to 0.

An automaton starts with an initial state, i.e each of the cells contains a symbol of the alphabet. The next automaton state is computed by applying the rules to the current state of each cell and its right neighbor to determine the new state for the cell. The state change happens simultaneously for all cells of the automaton.

As an example, consider an automaton with $n = 10$, the alphabet $0, 1$, and the following rules (only those with non-zero results are listed):

$$0\ 1 \to 1,$$
$$1\ 0 \to 1.$$

The remaining two cases (0 0 and 1 1) result in 0. Assume that the initial state of the automaton is

$$0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0$$

Then the new state of c_1 is 1 since the first rule is applicable. The new state of c_2 is 0 since 1 1 results in 0 by the convention (no rule is given for it, so the result is 0), and so on. After computing the new state for every cell, we get the following state of the automaton:

$$1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0$$

We compute the next automaton state by applying the rules to the current state:

$$1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1$$

The process continues indefinitely.

And now to the problems:

Problem 1. Write and test a program that, given the initial automaton state and the rules

$$0\ 1 \to 1,$$
$$1\ 0 \to 1,$$

prints the next 20 automaton states (including the initial state), one per line, starting from the initial state 0 1 1 0 1 1 0 1 1 0. The first three states are given above, so you can use them for testing. Use any format of printing that you would like.

What is the 20th state of the automaton, assuming that the initial state is counted as the first one?

Problem 2. Some initial states may result in all cells becoming zeros. For the automaton in problem 1, find three different initial states that eventually lead to all zeros. You may use your program to find them or just check them by hand. Explain how you found them.

Problem 3. Change your program (if needed) so that the number of cells and the initial state of the automaton can be changed, either as input to your program or in variables. Give clear instructions on how to input data or which variable they are stored in. Test your program on an automaton with the rules as in problem 1 and answer the following questions:

(1) The length of the automaton is 5 and the initial state is 0 1 1 1 0. What is the 50th state of the automaton?
(2) The length of the automaton is 7 and the initial state is 0 1 1 1 1 1 1. What is the 50th state of the automaton?

Problem 4. Change your program (if needed) so that you set automata rules by reading them as input or by specifying them as variables. The format of the rules is `0 1 -> 1` (the arrow is a minus sign followed by >). You may assume that the alphabet cannot have symbols other than $0, 1, 2, 3$. Recall that pairs of symbols for which no rule is given result in 0. This convention allows you to provide fewer rules. However, if it's more convenient for you, you may require specifying zeros as well.

(1) Try your program on the automaton with the rules

$$0\ 1 \to 1,$$
$$1\ 0 \to 1,$$
$$1\ 1 \to 2,$$
$$0\ 2 \to 2,$$
$$2\ 0 \to 2,$$
$$2\ 2 \to 1$$

The initial state is 0 1 2 2 1 0 1 2 1 0. What is the 50th state of this automaton?

(2) Now try the following automaton:

$$\begin{aligned}
0\ 1 &\to 3,\\
1\ 0 &\to 3,\\
1\ 1 &\to 2,\\
0\ 2 &\to 2,\\
2\ 0 &\to 2,\\
2\ 2 &\to 1\\
1\ 2 &\to 1,\\
2\ 1 &\to 2,\\
0\ 3 &\to 2,\\
3\ 0 &\to 1,
\end{aligned}$$

The initial state is 0 1 2 3 3 1 2 3 0 1. What is the 50th state of this automaton?

Problem 5. Since our automata have a finite number of cells, they eventually start repeating states. How many different possibilities of a state does an automaton with the alphabet $0, 1$ and the length 10 have? What about an automaton with the alphabet $0, 1, 2$ and the length 6? Explain your answer.

Problem 6. While in theory a large number of different states is possible, in practice many automata go into much shorter cycles. The length of a cycle is defined as the number of different states in the longest sequence of states between two occurrences of the same state. For instance, the automaton with the rules

$$\begin{aligned}
0\ 1 &\to 1,\\
1\ 1 &\to 1,
\end{aligned}$$

and the initial state 1 1 1 1 0 1 1 1 1 0 has a cycle of length 5. Note that sometimes an automaton goes through a non-repeated sequence of states before settling into a cycle.

Write a function that, given an automaton (i.e. the length, the rules, and the initial state), determines the shortest cycle that appears within the first 1000 rounds. Note that you will need to store all states that the automaton passes through and compare every new state to each of the ones you have already encountered. Think carefully about how you are going to store states and how you are going to compare them. Your program needs to do it efficiently, otherwise it would be too slow.

Sufficiently long automata may not go into a cycle within the first 1000 rounds. If there are no cycles, your program should print out "no cycles". You don't need to print out all the states for this part, but it

would be a good idea to print them out while testing and debugging your program on small examples.

(1) Try your program on the automaton with the rules (alphabet $0, 1, 2$):

$$\begin{aligned} 0\ 0 &\to 1, \\ 0\ 1 &\to 2, \\ 0\ 2 &\to 1, \end{aligned}$$

The initial state is 0 0 1 0 1 0 1 0 1 0. Print out the length of the cycle, or 'no cycles" if there are no cycles.

(2) Now try the following automaton (zeros are included in the rules, for easier checking):

$$\begin{aligned} 0\ 0 &\to 2, \\ 0\ 1 &\to 1, \\ 0\ 2 &\to 2, \\ 0\ 3 &\to 0, \\ 1\ 0 &\to 0, \\ 1\ 1 &\to 1 \\ 1\ 2 &\to 3, \\ 1\ 3 &\to 1, \\ 2\ 0 &\to 2, \\ 2\ 1 &\to 0, \\ 2\ 2 &\to 1, \\ 2\ 3 &\to 3, \\ 3\ 0 &\to 0, \\ 3\ 1 &\to 1, \\ 3\ 2 &\to 3, \\ 3\ 3 &\to 2, \end{aligned}$$

The initial state is 0 1 2 3 3 1 2 3 0 1. Print out the length of the cycle, or "no cycles" if there are no cycles.

Problem 7. Consider all automata of length 4 (how many are there?) and all their initial states (how many are there?). Write a program that runs each of these automata with all possible initial states, stores the length of the cycle (how many rounds do we need to simulate to guarantee a repeated state?). You may be able to eliminate some cases that are guaranteed to produce the same results as others. At the end the program prints how many of these automata have cycles of each encountered length, and for how many initial states. What can you conclude about how common cycles are in cellular automata of length 4?

Problem 8. Now consider larger cellular automata over the alphabet $0, 1, 2, 3$. You cannot try them all, but you can do a statistical sampling. Randomly generate rules for automata over this alphabet and an initial state of length 20. Run it for a specified number of rounds (say, 1000) or until a cycle is detected, whichever comes first. Why is it not feasible to run it until a state is guaranteed to be repeated? Record the length of the cycle, if one is discovered. Run 100 of such simulations (one automaton and one initial state each) and collect the data. Write down your conclusions. How common are cycles in such automata?

Problem 9. There are many cases and features of cellular automata that are interesting to study. Come up with a question about cellular automata that can be answered by a computer simulation, state your hypothesis, test it, and write your conclusions.