The PRIMES 2015 Computing Problem Set

Dear PRIMES applicant!

This is the PRIMES 2015 Computing Problem Set. Please send us your solutions as part of your PRIMES application by December 1, 2014. 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-math-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.

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

Enjoy!

General math problems

Problem G1. You roll three dice trying to get all three to be equal. If at some throw two are equal, you keep rolling the third one in the hope of it turning equal to the other two. What is the chance that you *do not* succeed if you are allowed to roll the dice two times? n times? (note that rolling all three dice or just rolling the third one each count as one roll).

Problem G2. John lives 2 miles north from a road, which is separated from John's house by a grove. If he walks from his house to the road along any straight line, the last mile of his walk is through the grove. Find the shape of the grove (describe its northern boundary by an equation).

Problem G3. Two white and two black rooks are placed at random on a standard 8-by-8 chessboard. What is the chance that NO white rook attacks a black rook? (recall that a rook attacks along the vertical and horizontal line it stands on). Give the answer as a fraction in lowest terms.

Problem G4. The number 1 is written on a blackboard. John plays the following game with himself: he chooses a number on the blackboard, multiplies it by 2 or 3, adds 1, and puts the result on the blackboard (if it does not appear there already).

a) How many pairs of consecutive numbers can appear?

b) Can it happen that three consecutive numbers appear on the blackboard?

c) Can it happen that the numbers $n, n+1, n+3, n+4$ appear for some n?

Problem G5. Prove that for every real $C > 0$, there is some finite nonempty set of integers A such that $|A + A| \geq C|A - A|$ (where $|X|$ is the number of elements in a set X). Here

$$A + A := \{a + b : a \in A, b \in A\}$$

and

$$A - A := \{a - b : a \in A, b \in A\}$$

are the set of pairwise sums and the set of pairwise differences of A, respectively.

For example, let $A = \{0, 2, 3, 4, 7, 11, 12, 14\}$. We have $A + A = [0, 28] \setminus \{1, 20, 27\}$ and $A - A = [-14, 14] \setminus \{\pm 6, \pm 13\}$. (Here $[m, n]$ denotes $\{m, m + 1, \ldots, n\}$). Thus $|A + A| = 26$ and $|A - A| = 25$, so the example proves the statement for $C \leq 26/25$.

Problem G6. Let $a_n, n \geq 1$ be the sequence determined recursively by the rule

$$a_1 = 1,\ a_{n+1} = \frac{n+2}{n+1} a_n + \frac{n^3 + 3n^2 + 2n - 2}{n(n+1)}.$$

Find a formula for a_n.

Hint. Compute the first few values and try to guess the answer.

Problem G7. Let M_n denote the $n \times n$ matrix whose entries are 0 below the main diagonal and 1 on and above the main diagonal. E.g.,

$$M_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Determine M_n^r for any $r \geq 1$.

Computer science problems.

About the problems. The theme of this year's problems is data encoding, in particular Huffman coding. Huffman coding is used in data compression. It is used for lossless data compression, i.e. compression in which there is no information loss due to approximation, so that uncompressed data is exactly the same as before the compression. You will experiment with various encodings of text and determine how effective they are.

What you need to do. For these problems we ask you to write a program (or programs) to experiment with various encodings. 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). Make sure that the file names do not contain any identifying information about you, such as your first or last name.

- 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. **Important: do not include your name in comments or in any file names.** If you are submitting your answers to non-code problems in a separate file, also make sure that it does not have your name in the contents or in the file name. The only place where you specify your name is the zip file with your solutions which must be of the form `yourlastname-CS-solution.zip` (replace `yourlastname` by your actual last name). **Make sure that you use zip compression, and not any other one, such as tar**. In the beginning of each file specify, in comments:

    1. Problem number(s) in the file. If you have a file with "helper" functions, mark it as such.
    2. The *programming language*, including the *version* (Java 1.7 or 1.8, 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)
    3. 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.

4. 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.
5. All of your test data.
6. 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.

Problem 1. For this problem you need to consider a simple encoding scheme, illustrated here on a small example. Suppose you have an alphabet of seven letters: a, b, c, d, e, f, g. We would like to encode these letters as sequences of zeros and ones of the same length so that each letter has a different encoding. It is clear that we can do it with strings of length 3: A is 000, B is 001, C will be represented by the next string which is 010 (a binary encoding of 2), etc. As the result we get:

a	b	c	d	e	f	g
000	001	010	011	100	101	110

Note that no character has the encoding 111.

Using this encoding, how long would a string of 0s and 1s be in order to encode an alphabet of 26 letters, such as English? What would be the encoding for the letter t? If you are given an alphabet of 33 letters, such as Russian, what would be the length of each letter encoding? Generalize it to an alphabet of N letters for any integer $N > 1$. Please explain all your answers.

Problem 2. Write a program to encode an English text with the character encoding that you proposed in Problem 1. Before you encode the text, your program must remove all the non-letter symbols and convert all letters to lower

case. For instance, the text ''If you can't explain it simply, you don't understand it well enough." is converted to "ifyoucantexplainitsimplyyoudontunderstanditwellenough" before encoding. What will this phrase encode to? What would the phrase "Courage is what it takes to stand up and speak; courage is also what it takes to sit down and listen." encode to? Your program may read text as input or have it stored in a variable.

Problem 3. Write a program to decode an English text encoded with the above scheme. What is the decoding of the phrase

```
10010101000001000010001001001010010000
10011100110110010010001001010011100100
11100010100110011100100001101001100010
11000101110011000010100000010000101110
10010001001001001101110001010000001000
01011101001000100100101100100010011001
11011101010010011010110111010010100100
11100010100100011011001100111101001001
00100000001001001100
```

Problem 4. Now we start exploring the idea of Huffman coding. Huffman codes are based on the idea that more frequent symbols in an alphabet should be encoded by shorter strings than less frequent letters. Consider the following alphabet with percentages of occurrences of each letter as below:

letter	a	b	c	d	e
frequency, %	51	22	15	7	5

The table shows the letter a on average occurs 51% of the time, the letter b 22% of the time, and so on. Below is an encoding of this alphabet with 0s and 1s using Huffman codes. Note that the more frequent letters have shorter encodings. The letter a, which appears, on average, more than half of the time, is encoded with a single character 1, the letter b with two characters 00, and so on:

letter	a	b	c	d	e
encoding	1	00	011	0101	0100

Now we can encode words that use this alphabet. For instance, the word *cab* will be encoded as 011100.

The encodings are of different lengths, but they are constructed so that we know exactly where each letter begins and ends, even though there are no separators between encodings of different letters. This is because no encoding is a prefix (i.e. a beginning) of any other one. For instance, if the encoded word starts with 00, the first letter in it must be b because no other letter starts with 00.

Encode words *ace* and *add* in this alphabet. Decode the following words: 101011, 0010101.

Problem 5. Compare the length of encoding of an average text in the alphabet given in the previous problem using the equal length strings of 0s and 1s (what length do we need?) and Huffman codes. Specifically, assume that we have a text of 100 letters from a to e distributed according to the frequencies

given in the previous problem (i.e. a appears 51% of the time, etc.). How many 0s and 1s do we need to encode 100 letters with equal-length encoding, as in problem 2? How many do we get for Huffman codes? What's the ratio between the two?

Problem 6. Now let us consider construction of Huffman codes for a given alphabet with letter frequencies. The algorithm is given here:
http://en.wikipedia.org/wiki/Huffman_coding#Basic_technique.
Below is an overview and a step-by-step application of the algorithm to the 5-letter alphabet in Problem 4.

The algorithm represents letters in the alphabet or groups of such letters as nodes that eventually are combined into a binary tree. At the start of the algorithm each letter, together with its frequency, makes up a node that is not connected to any other node. All nodes are placed into a set. I will represent each node as a pair of its frequency and its contents, in parentheses. For instance, $(51, a)$ is a node with frequency 51 that contains a. In our example, the set will look like this:

$$\{(51, a), (22, b), (15, c), (7, d), (5, e)\}$$

At each step of the algorithm we will take out two nodes with the smallest frequencies, combine them into one, and put them back into the set. The frequency of the combined node will be the sum of the frequencies of the nodes we are combining. In our case we need to combine $(5, e)$ and $(7, d)$. We will denote this combination as $(12, (5, e) + (7, d))$: the combined frequency of the node is 12, and it consists of $(5, e)$ on the left and $(7, d)$ on the right. Putting the combined node into the set, we get:

$$\{(51, a), (22, b), (15, c), (12, (5, e) + (7, d))\}$$

Now we repeat the process by picking the two nodes with the smallest frequencies and combing them. In this case the smallest two frequencies are 12 and 15. By convention we always put the node with the smaller frequency to the left when combining nodes. We get three nodes in the set: two simple nodes for a and b, and one combined node that consists of a node for c on the right and a combined node for e and d on the left.

$$\{(51, a), (22, b), (27, (12, (5, e) + (7, d)) + (15, c))\}$$

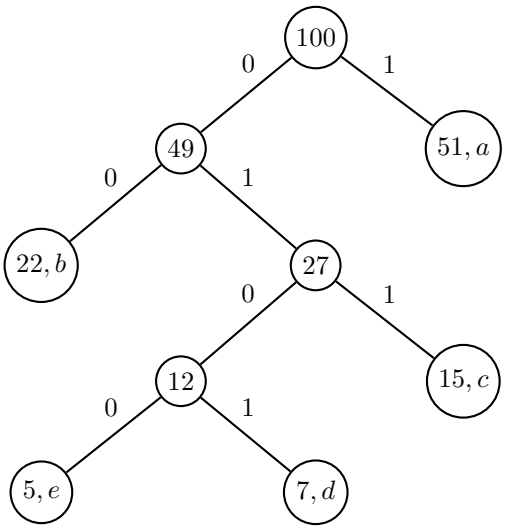
We continue this process until we get one large combined node in the set:

$$\{(51, a), (49, (22, b) + (27, (12, (5, e) + (7, d)) + (15, c)))\}$$

and finally

$$\{(100, (49, (22, b) + (27, (12, (5, e) + (7, d)) + (15, c))) + (51, a))\}$$

The result of the last iteration may be a bit difficult to read, so we show it as a tree that also contains 0s and 1s used for encoding:



The encoding of a letter is the path from the root of the tree to that letter, where left edges are labeled with 0, and right edges are labeled as 1. For instance, in our tree a is the immediate right child of the root, so its encoding is 1 (one step to the right). The path to b is left to the combined node with frequency 49%, and then left again, so its encoding is 00. Since no letter is on the path to any other letter, it is guaranteed that no encoding is a prefix of any other one.

Your task for this problem is to construct (by hand, no programming) Huffman codes for the following alphabet:

letter	a	b	c	d	e	f
frequency, %	25	22	18	15	11	9

Show all your intermediate work, i.e. how you combine nodes, and the resulting tree.

Problem 7. Write a program that reads in an alphabet with frequencies, i.e. pairs of a symbol (a letter) and a number, and constructs the Huffman codes for it. Try it on the two examples considered above, make sure you get the right answers.

The alphabet may be given in any order, not necessarily in the order of decreasing frequencies. If two nodes that you are combining have the same frequencies, the one whose leftmost letter is earlier in the given alphabet goes to the left. The leftmost letter in a node is its single letter if the node contains just one, or the one on the path that always goes to the left if the node is a combined one.

Your program should store and print the letters in the order in which they are specified in the given alphabet and their encodings.

Problem 8. Extend the program written in the previous program so that after it has computed the Huffman codes, it reads a text in the given alphabet (either as an input, or stored in a string) and encodes it using the Huffman codes that it has computed. Also extend it so that it reads an encoded text and

decodes it. Show test examples for both encoding and decoding.

Problem 9. Add an option to your program so that it computes the ratio between the space need for encoding 1000 letters in a given alphabet using fixed-length encoding to the space expected to be taken by Huffman code encoding, assuming that the given 1000 letters have exactly the same frequencies as the ones used to construct the Huffman codes. Test this option on the alphabet and frequencies given in Problem 4, the result should be equal to what you have obtained in Problem 5.

Problem 10. This page http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies gives you frequencies of letter in English alphabet. Compute the Huffman codes for the alphabet. Encode the following text, ignoring non-letter symbols and converting all letters to one case (from Wikipedia article on integrated circuits):

```
An integrated circuit or monolithic integrated circuit (also
referred to as an IC, a chip, or a microchip) is a set of
electronic circuits on one small plate ("chip") of
semiconductor material, normally silicon. This can be made
much smaller than a discrete circuit made from independent
components. ICs can be made very compact, having up to several
billion transistors and other electronic components in an area
the size of a fingernail. The width of each conducting line in
a circuit can be made smaller and smaller as the technology
advances.

ICs were made possible by experimental discoveries showing
that semiconductor devices could perform the functions of
vacuum tubes and by mid-twentieth-century technology advancements
in semiconductor device fabrication. The integration of large
numbers of tiny transistors into a small chip was an enormous
improvement over the manual assembly of circuits using
discrete electronic components. The integrated circuit's mass
production capability, reliability, and building-block
approach to circuit design ensured the rapid adoption of
standardized integrated circuits in place of designs using
discrete transistors.

There are two main advantages of ICs over discrete circuits:
cost and performance. Cost is low because the chips, with all
their components, are printed as a unit by photolithography
rather than being constructed one transistor at a
time. Furthermore, much less material is used to construct a
packaged IC die than to construct a discrete
circuit. Performance is high because the components switch
quickly and consume little power (compared to their discrete
counterparts) as a result of the small size and close
```

`proximity of the components.`

Then decode the following sequence of 0s and 1s encoded with the same Huffman codes:

```
11001010010111101010011110010101000001110001001010111
00011101001011011111110101011111001010001011101100010
01111100011001101101000000110111101100010110111110110
10100010100101010011111101001011110110111011100100111
11101011011110010001100010011111111011110011000011001O
11100010001010010111011011101100101110101010101010111
01010100111001010001000000011011111111101001001101100O
11000000011010011011101101000001111101110111001011101
11101100101101101111111010000110010111001101010001010O
10101001111110101110000011110011101011001101111100011
11001111111100100111011110010000101010111011010000111
10000100111000000111101010111111001010101011001110111O
01001001101110111110010100000011011111001111111001011
10001111010010110000011110110111001011101100000000110l
10101111011010100100010000101001011000110011011010110l
00101010011100101011000100101010010011000011101110101O
10101011111111010010111011010101111011011101110010000O
11011111001100011010101101110011001110100101001011100l
01010111110010100010011111010101111101100001010100111O
01010001001001010100101100111111010111011011101111001
00101011111111101111010111011100110101000101001010100l
11111010010111101101111100101000100111111010101111100l0
01111101011110101001010011101101000001101011001110101l
00010001011001010011110010100111010000100011011110011O
11010111010010111101101111100101000100110010100101111l
01010101011111001010001001100101011101110110011001111O
10111001100110111111100011110101111010100111100110000l
11001100111001101101000000110111111111010010011011000l
10000000110100110011101010100001110010101000001011111l
10101011000110100001100011111000011101111111100111110l
00001111100011001111101100111010111101011101000100000
00110101111011011101100011111011001110101100101111011OO
00111010010111101101111100101000100111110011101010101O
00010110111001101101010011011001101111011001010100101l
00111111100111110101100111111100011110101100000111000ll
00101011110010111001110101100110101000101001010100111l
11010100111111110100100110110001100000001101001101110l
11000110001000001110010000011110010010001111101100111O
10111011001111100101110111111011010000001101101110010O
11000100111110000111011111100110001101101100111110110O
11001110111000001011011101100000101100001101000101100l
01001110100101110111011101001011101010110101101010001
```

```
000011101001011011111000011110000100001100000101110110
000111111010110011111001110110011001110010110011101011
010000111111101010100111101001010000011011100111010101
010000101101110011011010100110110011011110111111001110
010101101000001101110000111010000100011011110011011010
111010010111101101111100101000100010010110000011110110
111100101001110110000110101111010100101001110110101001
101101010011011001101111011001111110000110010001110100
111000111111101111011011011000011111001100001110010111
010000011101110101010011100101000100001010111110000111
000101100110000010010000011111110101110101110011001010
111101010010100111011010100110110101001101100110111101
100100110101000101001010100111111010100101001011000001
111011011101011111110101011000110110010100101110001011
110001110100001110110110111000100111100111101011010010
111010000011101110010111001010101111100101000100111101
011110110111011110010101110111001010011111110000001100
010010110001101000001101110100101001001101100011100011
010010111001010111001010100001110000111100001000011000
100111000010100000110111001011111000011110000100001100
010011111100001110011101100000001110111001111011110000
110100000111000111111101000001111010101000100001111011
110010001100010011111111101111101001110110100010010010
110000011110110111100101110100101111011011111001010001
001001101101010011011001101111011001100101111100111111
000111100001111101101101001101010001010010101001111110
100101111011011111001010001001100100110101101000111111
101111000111111011011001110111100100011000100111111101
111101001110110100010011111111011110101011110101001110
011000011100110011100110110100100101100000111101101111
001
```

Problem 11. It is a proven fact that Huffman encoding is an optimal encoding (i.e. it guaranteed to produce the shortest encoded text of 0s and 1s from which the original text can be decoded) if the frequency of letters in the text is the same as the letter frequencies used to compute the Huffman codes. Experiment with different English texts to see how close to the optimal length you get in practice. You are allowed to use any English text of at least 1000 letters available online (make sure to include the link and clearly indicate what part of it you are using). Your goal is to determine how far from optimal encoding (in percentages) do various texts deviate. Use three texts, try to come up with interesting examples.

Problem 12. Suppose you are given an alphabet of N letters in which all letters have the same frequency. What is the Huffman encoding of this alphabet? Clearly explain why. How does the encoding with Huffman codes compare to the equal-length encoding that you studied in Problem 1? Be specific. Assume

that in any text that you encode all letters appear with equal frequencies.

Problem 13. Continue your study of how Huffman encoding efficiency depends on letter frequency distributions. Consider the following cases for 8-letter alphabet:

- One letter has 50% frequency, the other seven are equally distributed (disregard rounding errors),
- Four letters have 20% frequency each, the remaining four have 5% frequency each,
- The frequencies are, in percentages: $50, 25, 12.5, 6.25, 3.125, 1.5625, 0.78125, 0.78125$.

In each of the cases compute the efficiency of Huffman coding compared to the equal-length coding in Problem 1. Which of these cases give you a better ratio? Come up with a hypothesis of what frequency distribution produce better efficiency. Can you come up with an 8-letter alphabet with Huffman encoding more efficient than the best case above? Show the alphabet or explain why it's not possible.

Problem 14. Of the languages listed here http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/, are there any that are more efficient than English in terms of Huffman encoding? Compute efficiency of at least one language other than English that you consider a good efficiency candidate, show the Huffman codes and the computation of efficiency. You don't need to encode/decode texts in this alphabet. If your program does not read unicode characters, feel free to denote your extra letters with numbers or any other ASCII symbols.

Problem 15. Consider the following enhancement of Huffman codes: in addition to individual letters, common pairs of letters (known as bigrams), such as *th* in English, are each given their own Huffman code. Consider bigram frequencies given here, would this enhancement be beneficial? How many bigrams would you include in Huffman codes, if any at all? Show all your computations. You may perform your computations by hand, but it's better to include them in your program. Once again, you don't need to encode/decode texts using the alphabet, just compute the Huffman codes and efficiency.