The PRIMES 2013 Computer Science Section Problem Set

Dear PRIMES applicant,

If you are applying to the Computer Science Section, please send us your solutions of this problem set as part of your PRIMES application by December 1, 2012. 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 in both parts as you can.

For the general math problems, you can type the solutions or write them down by hand and then scan them. Please attach your math solutions to the application as a PDF file. Write down not only answers, but also proofs (and partial solutions/results/ideas if you cannot completely solve the problem). 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.

Your solutions of the computer science problems may include several different files. In this case, please put them in a folder whose name starts with your last name, for example, "smith-cs-solutions" and zip this folder. Do not use any compression tools other than zip. Attach the zip file to your application email. If it’s a single file, please keep the original file extension (.java, .c, etc.) and name your file similarly.

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. For an elementary introduction to cryptography methods, see, for example, Janet Beissinger and Vera Pless, *The Cryptoclub: Using Mathematics to Make and Break Secret Codes* (AK Peters, 2006), and Simon Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography* (Anchor, 2000). For a more advanced textbook, see Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition (Wiley, 1996).

Note that some of these problems are tricky. We recommend that you do not leave them for the last day, and think about them, on and off, for some time (days and possibly weeks). We encourage you to apply if you can solve at least 50% of the problems in both parts. We note, however, that many factors will play a role in the admission decision besides your solutions of these problems.

Good luck!

General math problems

Problem G1. You toss two coins trying to get two heads. Namely, you toss both coins until you get at least one head. If you get two heads at that point, you finish, and if you get just one, you keep tossing the other coin until you get a head.

(a) What's the chance that you will not finish in m tosses? (Tossing two coins simultaneously counts as one toss).

(b) Give the answer as a fraction for $m = 8$.

Problem G2. A black Bishop and a white King are placed on a $2n$ by $2n$ chessboard (all positions are equally likely). What is the chance that the Bishop attacks the King (i.e., the King is under check)? (Recall that the Bishop strikes along diagonals). Write the answer as a rational function of n.

Problem G3. A sequence $a_n, n \geq 0$ of positive integers satisfies the recursion $a_{n+1} = F(a_n)$, where $F(x)$ is the sum of the squares of all the digits of x (for example, $F(324) = 3^2 + 2^2 + 4^2 = 29$). The number a_0 can be arbitrary.

(a) Show that either $a_n = 1$ starting from some n, or a_n is periodic with minimal period 8 starting from some n. What are the 8 numbers repeating periodically in the second case?

(b) Let h be any nonnegative integer function on the set of digits $0, 1, ..., 9$. Consider the recursion $b_{n+1} = F_h(b_n)$, where $F_h(x) := \sum_i h(x_i)$, and x_i are the digits of x. Show that b_n is eventually periodic, and there are finitely many possible minimal periods for it.

Problem G4. Let $a > 1$.

(a) For which a does there exist a solution $0 \leq x \leq 1$ of the equation

$$(1 + x)^{1/3} + (1 - x)^{1/3} = a?$$

Show that if such a solution exists, it is unique, and find it.

(b) For which a does there exist a solution $0 \leq x \leq 1$ of the equation

$$(1 + x)^{1/4} + (1 - x)^{1/4} = a?$$

Show that if such a solution exists, it is unique, and find it.

Problem G5. Find all rational numbers a, b for which the number $\gamma = \log_2(a + b\sqrt{2})$ is rational.

Computer science problems.

About the problems. The theme of this year's problems is encryption, decryption, and breaking ciphers. The ciphers that you will be exploring in this problem set are not real-life ciphers. Some of them are trivial to break even by hand, the other ones can be easily broken with a computer (you will learn how). However, the study of these ciphers gives you interesting insights into the world of encryption, and you can find ways to modify them to make them more difficult to break.

What you need to do. For these problems we ask you to write a program (or programs) to implement the desired functionality. 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 (which is probably a good idea), 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 example that you were using (you can write them 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. If you were using test data (other than what was given to you as a part of the problem statement) or had your testing code in a separate file, please submit those as well.
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.

Programming language features you might need. *Modulo (or modulus)* operator is very important in cryptography. a modulo b is defined as the remainder of integer division of a by b. For instance, 11 modulo 3 is 2, 10 modulo 2 is 0, and so is 0 modulo 2. Note that -2 modulo 3 is 1 since $-2 = 3 \times (-1) + 1$.

All modern programming languages have a modulus operator, but in most of them it's **incorrect**: it returns a negative result for a negative number. For instance, in C-like languages (including Java) the default "modulus" operator (which is %) returns -2 for -2 modulo 3. The correct answer, in this case 1, is obtained by adding the divisor, in this case 3. If your language has this issue, we suggest that you write a function to perform this adjustment for negative numbers automatically and use this function instead of the default modulus operator.

And now to the problems:

Problem 1. Most textbook encryption schemes work on the 26 letters of the English alphabet, leaving out punctuation marks. Our encryption will work on the 26 letters of the English alphabet and, additionally, four common punctuation marks:

- a period **.**

- a comma **,**
- an exclamation point **!**
- a question mark **?**

They are added to the end of the alphabet in the above order. **All other characters are omitted when encryption is performed.** We also don't distinguish between lower-case and upper-case letters. We will refer to our alphabet of the 26 letters and four punctuation marks as A.

Every letter in the alphabet is assigned a number, in order, starting at 0. This means that a is assigned 0, b is assigned 1, z is assigned 25, . is assigned 26, and so on.

The first encryption scheme that you need to implement is very simple. You are given a text to encrypt, for instance a string `Why not?`, and another symbol in the alphabet A, for instance a letter c, that serves as the encryption key. In order to encrypt the given text using the given key, you need to do the following:

1. convert all letters to lower-case,
2. remove all the symbols that are not in A from the input text. We refer to the result of the first two steps as *filtered input.*
3. convert the key character to its assigned number. We will call it the *numeric key.*
4. for each character in the filtered input do the following:
   (a) add the numeric key to the character,
   (b) take the result modulo 30, i.e. if the result is larger than 29, subtract 30 from it.
   (c) convert it from a number to a character in A according to its position in A (for instance, 3 is converted to d).

   This encryption approach is called a Caesar cipher.

For the above example we get the encryption `yj.pqvb`. The empty space character got filtered out, all letters got converted to the lower case. Then every character gets shifted forward by 2 since c has a position 2 in the alphabet (recall that a has a position 0). The ? character is the last in the alphabet (its assigned number is 29). Adding 2 to it gives 31 which is outside the alphabet, so we take the result modulo 30 and get 1, which corresponds to b.

As another example, the same string `Why not?` encrypted with a symbol ! results in `ufwlmr,` (the comma is a part of the encryption).

Your task for this problem is to write a function that takes an input text and a key (a symbol in A) and returns the corresponding encrypted text.**Then show the result of this function for the string**

`Neil Armstrong, Mike Collins, and Buzz Aldrin flew on the Apollo 11 mission.`

using the key m.

Problem 2. The task for this problem is to write a function that performs decryption: you need to write a function that takes encrypted text and a key (as a character) and returns the filtered original (unencrypted) text. Note that there is no way to recover characters that got filtered out during encryption or the distinction between upper/lower-case characters, so the decryption of the examples in Problem 2 will result in `whynot?` (make sure to test your function on these examples).

Submit the result of decrypting the string
`?i?mbt?sdnondihzitajmhnc?zowgdbcow,c?hd,zg?i?mbtwzi!?g?,omd,zg?i?mbtv`
with the key z.

Problem 3. This type of encryption is, of course, not secure at all: there are only 30 possibilities of a key. You can try them all even by hand, and stop once you recognize English words in the decryption. However, we would like to have an automated way of breaking this simple encryption. This approach is based on the fact that some letters in English occur more often than others. For instance, a is very common, whereas x appears much less frequently. This means that if we have a sufficiently long encrypted text, we can measure frequencies of characters in it. The most frequent character in encrypted text is likely to decrypt to a, t, or e. Knowing this, as well as other frequencies, allows you to automatically determine the likely key without even trying any dictionary words.

Frequencies of English letters are well known, but our alphabet is not quite English: it also includes the four punctuation marks. Thus you need to compute frequencies of characters in this alphabet based on some sample English texts.

Write a program (or a function) that, given a text as input, returns the number of occurrences of every character of A in the filtered text (i.e. after all non-A characters are removed and all letters are converted to the lower case) and the percentage of occurrences of each character in the filtered text. As a small example, in the string `This string is a test 1-2-3.`

- s occurs 4 times (22.22%)
- t occurs 4 times (22.22%)
- i occurs 3 times (16.67%)
- each of `a, e, g, h, n, r`, and . occurs once (5.56% each)

Spaces, digits, and dashes are filtered out.

Since short texts may have unusual distributions of frequencies, you need to use a long text to get an accurate estimate. This link has full text of *Pride and Prejudice* which should be sufficient (also available for download as test1.txt from "How to Apply" page). Write a function that reads text and determines the number of occurrences and frequencies of each character in the corresponding filtered text. **Compute and submit the resulting numbers of occurrence for each character of the alphabet A and the corresponding frequencies in this sample text** (in comments or as a separate file).

Problem 4. Now that you know the frequencies of characters in A you can guess an encryption key based on matches of its frequencies to A. Below is an encrypted text:

```
jurlnbj,knprwwrwp!xpn!an.d!r.nmxo,r!!rwpkdqn.,r,!n.xw!qnkjwtgjwmx
oqjarwpwx!qrwp!xmxxwlnx.!brln,qnqjmynnynmrw!x!qnkxxtqn.,r,!n.bj
,.njmrwpgk?!r!qjmwxyrl!?.n,x.lxwan.,j!rxw,rwr!gjwmbqj!r,!qn?,nxojkx
xtg!qx?pq!jurlnbr!qx?!yrl!?.n,x.lxwan.,j!rxwi,x,qnbj,lxw,rmn.rwprwqn
.xbwvrwmj,bnuuj,,qnlx?umgox.!qnqx!mjdvjmnqn.onnuan.d,unnydjwm
,!?yrmgbqn!qn.!qnyunj,?.nxovjtrwpjmjr,dlqjrwbx?umknbx.!q!qn!.x?ku
nxopn!!rwp?yjwmyrltrwp!qnmjr,rn,gbqnw,?mmnwudjbqr!n.jkkr!br!qyr
wtndn,.jwlux,nkdqn.f
```

Compute its frequencies of characters, compare them to frequencies of A, and guess the encryption key. Then check your guess by decrypting the text. Note that your frequencies would not be exactly the same as the ones computed in the previous problem, but they should be close. **Submit the decrypted text and the key, and give a brief explanation of how you figured out the key.**

Your solution would be ranked higher if you automate this process to some degree: write a function that, given a text, returns the most likely key or several top guesses.

Problem 5. In this and the following problems we change the cipher to a more sophisticated one, known as Vigenere cipher. A key for this cipher is a short word or a phrase in which all characters are in A. To encrypt a text, you need to do the following:

1. Filter out all characters not in A and convert all the letters to lower-case, just like in Problem 1.

2. Use the first character of the keyword to encrypt the first character of the filtered text (by adding the numeric key to the character's position in the alphabet and taking the result modulo 30), the second character of the keyword to encrypt the second character of the filtered text, and so on. When you reach the end of the keyword, start from its beginning, and continue cycling over the keyword until the end of the filtered text.

As an example, the encryption of `A short sample text` with the keyword `primes!` is `pfp.vhqp?xxihcig`. This cipher has an advantage that a character is not always encrypted with the same character (note that the two occurrences of s in the input text get encrypted to two different characters).

Write a function (or a program) that takes a text and a keyword and returns the text encrypted with this keyword. **Submit the encryption of the text**

`Cryptography is the study of techniques for secure communication in the presence of third parties.`

using the keyword `primes!` (the ! is a part of the keyword).

Problem 6. Write a function to decrypt a text given a keyword. **Submit the decryption of the string**

```
,blqvbaclxbsyppcpgmg?pfmpsbkpgpqqsrxtixxzc?ecmrva??xcxwpdtqqruca
eiox.at
```

also using the keyword `primes`!

Problem 7. One might think that because Vigenere cipher is *polyalphabetic*, i.e. one character may be encrypted by different ones, it is very difficult to find out the encrypted text without knowing the key. However, it turns out that if you have a sufficiently long text, the same method of frequencies that you used for breaking the Caesar cipher can be used here.

The approach consist of two parts: guessing the length of the keyword and guessing the keyword itself (which, of course, allows one to successfully decrypt the text). The idea is the following: suppose the key length is N. Then every N^{th} character of the text is encrypted with the same character of the keyword, so if we take every N^{th} character of the encrypted text, we would get character distribution very similar to what you have discovered in Problem 3. On the other hand, if the real length of the key is N, but you take every M^{th} letter, where $M \neq i \times N$ for any integer i, and assuming that all characters of the keyword are different, the distribution of the frequencies of characters separated by M would be much closer to uniform (i.e. evenly distributed characters). Experiment with different texts to see the difference.

Thus in order to find the length of the key you need to try different lengths N and find one that makes the frequencies of all characters at a distance N from each other closest to those of an English text, as determined in Problem 3. Once you have discovered the length of the keyword, you can determine each character of the keyword by breaking down the ciphertext into groups of characters separated by N and then applying the method used in Problem 4 to each group.

For this problem you need to write a function (or a program) that automatically suggests likely length of the keyword, given an encrypted text. You may assume that the keyword does not have repeated characters and its length is between 3 and 15 characters, inclusive. You don't need to automate finding the keyword itself once you found the length (you can determine it manually by looking at the frequencies), but **your solution will rank higher** if you automate this process as well (perhaps just reuse your solution for Problem 4). Even if you are determining the keyword characters manually, you still need a function to print out the frequencies of characters at a distance N from each other, and perhaps other helper functions: the more convenient – the better.

The sample text to decrypt (without knowing the keyword!) is here (also available for download as test2.txt from "How to Apply" page). The length of the keyword is between 3 and 15, and it doesn't have repeated characters. **Submit the keyword that was used to encrypt the text, and explain how you found it.** You don't need to submit the decryption.

Problem 8. This problem is open-ended and deals with developing improvements to Vigenere cipher. Suppose that you are starting with the same alphabet A, but the alphabet that you use for the resulting ciphertext may be larger, for instance it may have 40 characters (say, we add the 10 digits). What

can you do to make breaking the cipher harder? Your goal is to defeat the frequencies method. A complete solution for this problem would include the description of the method, the reasons why it works, the code that implements it (both encryption and decryption) and some statistics that show that the decryption is harder to break than the traditional Vigenere cipher. Make sure that every encrypted text always decrypts back to the original (filtered) text when the keyword is known.