PROBLEM OF THE MONTH

December, 2016

2016-2017

MATHEMATICS

5 points:

A sequence of numbers is defined by the following recurrent relation $a_{n+1} = \frac{1}{a_n+4}$, with $a_1 = 1$. Find first few members of the sequence (you can use computer if you wish). What do you notice? Can you guess or derive a_{2017} with good accuracy?

Hint: Notice that a_n is getting closer and closer to some number as n grows. Denote this number a. Can you write an equation for a?

Answer: $-2+\sqrt{5}$

Solution: It is easy to find $a_2 = 0.2,\ a_3 = 0.2381,\ a_4 = 0.23956,\ a_5 = 0.23607,\ a_6 = 0.236068,\ \ldots$ It seems that at large n the sequence is approaching some number $0.236067\ldots$ Let us assume that this is indeed so and call this number a. We write a relation $a = \frac{1}{a+4}$ or $a^2+4a-1=0.$ Solving this equation we obtain $-2\pm\sqrt{5}$. The answer is obviously positive and we have $a = -2+\sqrt{5} = 0.236067977\ldots$ One can also show that if a_n is sufficiently close to $-2+\sqrt{5}$, than a_{n+1} is even closer to this limit.

10 points:

A sequence of numbers is defined by the following recurrent relation $a_{n+1} = \frac{n+1}{n}a_n - \frac{2(2n+1)}{n^2(n+1)}$, with $a_1 = 3$.

Compute a_{2017} with the accuracy of at least 10^{-100}.

Hint: Try to simplify the fraction in the right hand side by decomposing it in more elementary ones using $2n+1 = (n+1)^2 - n^2$.

Answer: $a_{2017} = 2017 + \frac{2}{2017}$

Solution: Let us rewrite $a_{n+1} = \frac{n+1}{n}a_n - \frac{2(2n+1)}{n^2(n+1)} = \frac{n+1}{n}a_n - 2\frac{(n+1)^2-n^2}{n^2(n+1)} = \frac{n+1}{n}a_n - 2\frac{n+1}{n^2} + \frac{2}{n+1}$ or $a_{n+1} - \frac{2}{n+1} = \frac{n+1}{n}(a_n - \frac{2}{n})$. We have $a_n - \frac{2}{n} = \frac{n}{n-1}(a_{n-1} - \frac{2}{n-1}) = \frac{n}{n-1}\frac{n-1}{n-2}(a_{n-2} - \frac{2}{n-2}) = ... = \frac{n}{n-1}\frac{n-1}{n-2}...\frac{2}{1}(a_1 - \frac{2}{1}) = n(a_1 - 2) = n$. Finally, $a_{2017} = 2017 + \frac{2}{2017}$. This is an exact answer.

As an alternative solution one can calculate first few terms of the sequence, guess the formula for the answer and then prove the formula by the method of mathematical induction.

PHYSICS

5 points:

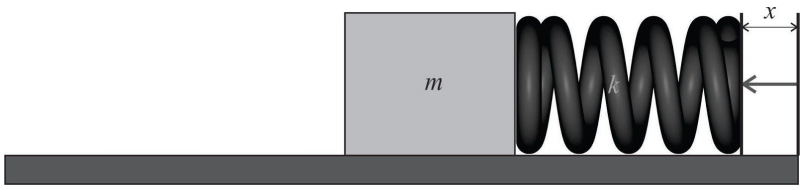
An aluminum sphere of volume V= 200 ml, with a void inside, is floating half-immersed in water. Find the volume of of the void.

Hint: Use Archimedes principle to find mass of the sphere.

Answer: $63ml$.

Solution: According to the Archimedes principle, the bouyancy force is equal to the weight of displaced water. This means that the mass of the sphere is equal to that of water in the volume $V/2 = 100ml$, i.e. $100\,g$. Since Aluminum density is 2.7g/ml, the volume of the aluminum is $(100/2.7)ml = 37ml$. The rest should be the volume of the void: $100ml - 37ml = 63ml$.

10 points:



A block of mass m standing on a flat surface is very slowly pushed through a massless spring with the spring constant k. At first the block stands still and the spring gets compressed, but then at some point when the spring is compressed by distance x the block starts moving. Find the distance that block will travel once it starts moving (assume that the far side of the spring stands still while the block moves). The coefficient of static friction is $\mu_1 = 0.5$, the coefficient of kinetic (sliding) friction is $\mu_2 = 0.4$.

Hint: Change in energy should be equal to the work done by friction.

Answer: $d = 2\left(x - \frac{\mu_2 mg}{k}\right) = \frac{2(\mu_1 - \mu_2)mg}{k}$.

Solution: Change in energy should be equal to the work done by friction. If the distance travelled by the block is d, the work of the (kinetic) friction is $\mu_2 mgd$. Since kinetic energy is 0 both in the beginning and at the end of the motion, we only need to consider the potential energy of the spring, which changes from $\frac{kx^2}{2}$ to $\frac{k(x-d)^2}{2}$. Hence,

$$\mu_2 mgd = \frac{kx^2}{2} - \frac{k(x-d)^2}{2} = \frac{k\left(x^2 - x^2 + 2xd - d^2\right)}{2} = kd\,(x - d/2)$$

Therefore, $d = 2\left(x - \frac{\mu_2 mg}{k}\right)$. Note that the initial compression of the spring x can be found based on the fact that the maximum static friction force at the onset of motion is equal to the spring force: $\mu_1 mg = kx$, so $x = \mu_1 mg/k$.
This gives

$$d = \frac{2(\mu_1 - \mu_2)mg}{k} .$$

Note that this solution only works if the spring is not completely relaxed (otherwise, the block would detach from the spring, and the spring's potential energy would become simply 0). So we need to check that $x - d > 0$. $x - d = (2\mu_2 - \mu_1)mg/k$ is indeed positive since $2\mu_2 - \mu_1 = 0.3 > 0$.

CHEMISTRY

5 points:

In Amazonian selva, many illegal gold miners use a very dangerous and obsolete technology for final step of gold extraction and purification. What this technology consists in, what chemical and physical processes are behind it, and why is this technology so dangerous specifically for selva ecosystem?

There WILL be a hint for this problem.

Hint:

Read about properties of gold amalgam.

Solution:

The procedure of gold refining the illegal miners use in Amazonian selva include formation of the gold amalgam. “Amalgams” is the name of alloys formed by mercury and other metals. Most metals, except iron and few other metals form amalgams with mercury, and gold is not an exception. Interestingly, although alloys are usually formed by melting two metals together, formation of alloys does not require both metals to be liquid. If one metal (with lower melting temperature) is in a liquid state, it can dissolve another metal (with higher melting temperature) even when the temperature is below the melting temperature of the second metal. As a result, the zinc-copper alloy can be made by dropping pieces of solid copper (melting temperature 1085 ºC) into liquid zinc (melting temperature 420 ºC). The same is true for gold. By adding liquid mercury to the enriched gold ore (small gold particles mixed with ordinary sand, rocks etc) a gold amalgam forms, which is a very dense liquid which flows downwards, so everything else can be easily removed. After the amalgam is collected, the most dangerous step starts: the miners start to heat it at open air. Mercury evaporates, whereas the gold stays, and it can be collected. The problem is that mercury vapors precipitate easily, they become easily absorbed by plants, and the mercury reacts with biological molecules (proteins, nucleic acids). As a result, covalent bonds form between mercury atoms and proteins and/or nucleic acids, and the biological properties of them change, which leads to malfunctioning of these molecules, to mutations and other negative effects. However, that is not the end of story. Even after that the mercury atoms do not stop causing damage to the selva ecosystem: the bodies of poisoned organisms are being consumed by other animals or plants, so the mercury continues to produce its toxic effect, and this deadly cycle can potentially last forever, because the specifics of the rainforest ecosystem is that all organic materials are being recycled by next generations of living organisms. That is why amalgam refining of gold is especially dangerous is Amazonian selva.

10 points:

A glass beaker is separated by a semipermeable membrane in two equal parts as shown in the figure. A saturated solution of sodium chloride is in the left part of the beaker, and a saturated solution of cesium iodide is in the right part. Two strings are attached to the ends of the balance as shown on the figure, and the crystals of sodium chloride and cesium iodide are attached to the ends of these strings (see the figure). Since a

saturated solution of some substance is the solution that cannot dissolve additional amount of that substance, and because each crystal is immersed into the solution of the same salt (sodium chloride crystal is in the saturated solution of NaCl, and cesium iodide is in CsI), each crystal is in equilibrium with the solution, and the balance is in equilibrium too.
Describe what will happen if we leave this system to stand.
Two comments: First, ignore the effect of water evaporation (it is too slow in this case). Second, in this experiment, the semipermeable membrane is permeable for water molecules, but not for cesium iodide or sodium chloride.

There WILL be a hint for this problem.

Hint:

There will be water diffusion from one part of the vessel to another, so one solution will not be saturated anymore, so one of two crystals will start to dissolve, and its weight will change. The key question is what is the direction of water transfer? By answering it you will find the solution.

Solution:

Since water can penetrate the membrane, water molecules will be freely jumping from one side of the vessel to another and back. If we denote the probability of a single event (jumping of one water molecule through the membrane) as p_i , then the rate of this process will be equal to p_i times the number of water molecules (strictly speaking, their molar fraction, i.e., the number of *moles* of water molecules per one liter of the solution). Since cesium, sodium, chloride, and iodide ions do not participate in this process (they are too big to penetrate the membrane), they just occupy the space, thereby decreasing the amount of water molecules per a unit volume, so the overall frequency of water molecule transitions becomes lower.
That means water molecules will be jumping more frequently from the solution where the molar fraction of water is higher to the solution where the molar of water molecules is lower.
In which solution the water molar fraction is higher? For simplicity, let's assume both NaCl and CsI solutions have the same density (strictly speaking, that is not the case, the CsI solution has somewhat greater density, but, as we will see, that will have no effect on our answer). At 20 degrees, a saturated NaCl solution has a concentration of 359 g/L, or 359/58 = 6 M.

A saturated CsI solution has a concentration of 765 g/L, or 765/260 = 2.94 M. Assuming that densities of both solutions are about 1.2 kg/L, the mass of water is 1200-765 = 435 g (or 435/18 = 24.2 mols) in the CsI solution, and 1200 - 359 = 841 g (or 841/18 = 46.7 mols) in the NaCl solution.
The last step in our solution is the calculation of the molar fraction of water. In the CsI solution, it will be 24.2/(24.2+2.94) = 0.892, in the NaCl solution, it will be 46.7/(46.7+6)= 0.886.
That means the molar fraction of water in the NaCl solution is lower, so the water will go *from* the CsI solution *to* the NaCl solution, so the CsI crystal will start to grow (cesium iodide will start to precipitate), and the NaCl crystal will start to dissolve, and the balance will tip accordingly. (If we will take into account the difference in CsI and NaCl solutions density (the density of the former is greater), the result will be qualitatively the same; you can try to do that if you want.)

This observation is somewhat counterintuitive: water seems to migrate from the solution where its *mass fraction* is lower to the solution where its mass fraction is higher. However, that is not surprising, because in chemistry, in most cases not the mass of the substance matter, but the amount of particles. That is why we chemists measure materials in moles, not in grams of kilograms.

BIOLOGY

5 points:

Most of the major animal groups rely on contraction of muscle for movement, respiration, and blood circulation. Muscle contractions are initiated by the firing of motor neurons that often link up to multiple muscle fibers. As a result, only one or a few neurons are needed to effectively trigger a relatively large muscle, firing each time the muscle needs to contract.

However, nature has a few examples where muscles can contract spontaneously, without the input of the nervous system (not to be confused with voluntary vs. involuntary muscle control by the brain). Can you think of an example of an animal or a type of muscle where contractions are uncoupled from neuronal activity? Why might such “uncontrolled” contractions be of an advantage there?

Answer:

Muscles are made to contract by motor neuron firing through an effect known as excitation-contraction coupling. The firing of the neuron’s action potential is coupled to the release of calcium deposits within special compartments in muscle cells. It is this release of calcium that in turn allows the filaments of two different proteins (actin and myosin) in each muscle to slide against one another and generate contractile force. The firing of the action potential and the subsequent release of calcium, however, take time and energy. Most insects need to beat their wings at frequencies much higher (up to 1KHz) than those achievable with waves of action potentials. To compensate for this, fibers of insect flight muscles are known to contract spontaneously and on their own after they have been triggered once by a neuronal impulse. It is a phenomenon known as stretch-activation response and, simply put, it works because the muscles that beat the wing down are stretched once the wing is brought back into its upward position, and this stretching generates enough force to bring the wing back down again (not unlike the oscillations of a spring once it is stretched and let go). The stretch-activated contractions will continue between neuronally-triggered contractions. Thus, if the firing of action potentials occurs only a few times a second, the added spontaneous contractions of the stretch-response can being that frequency up several hundred fold. This phenomenon was first described in the 1970s by Drs. Schädler, Steiger, and Rüeg in the Netherlands, and has been studied since. Curiously, in parallel to insect muscle, stretch-activated contractions were also observed and described (by Sr. Steiger among others) in skeletal

and heart muscle of vertebrates such as mice and rabbits. However, the phenomenon is stronger in insect flight muscles, and three quarters of all known flight insects depend on it to fly.

Here are some general links about the topic:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1304450/
https://www.ncbi.nlm.nih.gov/pubmed/10952872

Here's a great paper and comment on the more recent findings. One of the controversies over the subject has been an idea that the structure of the protein components of insect muscle fibers may be different from other animals, which would explain their ability to use stretch-activation so well. In the 2013 paper by Iwamoto and Yagi, the researchers find evidence that seems to agree that stretch-activation contractile patterns of a bumblebee are similar to those of cardiac muscle, suggesting that insect and vertebrate muscles are not so different after all. Also their methods are remarkable:
http://www.nature.com/news/flight-of-the-bumblebee-decoded-1.13587
http://www.sciencemag.org/content/341/6151/1243.full.pdf

10 points:

Tomorrow afternoon you get a phone call from the nearest Center for Disease Control informing you that a brand new hospital is being built in your area. As luck would have it, all of the CDC's own experts have already taken off for the holidays, and you are being asked to help with making sure that the new hospital is built in accordance with safety guidelines for infectious disease control. You call the agency in charge of the hospital's construction, and find out that plans for the new building include an emergency room, an operating room, a tuberculosis ward, and a number of rooms for generally admitted patients. You need to decide how to set up the ventilation in these different parts of the hospital so as to keep infectious microorganism populations from becoming a danger to the people inside the hospital as well as to all those living in your area. You have three choices to set up the airflow in any given room. These are: 1. outside air to flow into the room, 2. inside air to flow out, or 3. air to enter and leave the room through a system of filters separating it from the rest of the hospital's airflow.

What would be the proper choice of airflow for the different rooms in the hospital, and why?

Bonus Hint: Microorganisms flourish when they have easy access to nutrients and (importantly) when they see little or no competition from other organisms.

Answer:

When referring to the different types of airflow in a hospital, it is important to understand the actual picture of what is going on in each case:

If the air flows into the room from the outside, as in option 1, this means there is negative pressure inside the room, the air enters through the door, and leaves through the air vents (where it is ultimately filtered before leaving the hospital altogether). If the air flows from outside, as in option 2, this means there is positive pressure in the room, the air enters through the air vents (having previously been filtered), and exits through the door. When referring to separating the room from the rest of the hospital's airflow, as in option 3, this means that no matter what the pressure inside the room is (usually positive for operating rooms), there are filters at sites where air enters and leaves the room.

Now consider what happens in the different rooms of the hospital:

The Operating Room needs to be fully isolated from the hospital's airflow (option 3). The OR will contain patients with exposed tissues and organs and require absolute sterility to avoid infection. Any patient would need to be cleaned before entering the OR in order to keep microorganisms from hitchhiking along and contaminating this sterile environment.

The Emergency Room and the TB ward need to have outside air going in (option 1). Tuberculosis bacteria (Mycobacterium tuberculosis) is an airborne pathogen, and air from the TB ward should not be allowed to enter another, uncontaminated part of the hospital. Why not separate the TB ward from the hospital air supply, like the OR? This is due to the danger of creating superbugs, or drug-resistant pathogens, and is not limited only to TB. Any part of the hospital that houses patients with infectious disease needs to be connected with the outside fauna in order that the bugs brought in with patients experience competition from other bugs. If this competition is removed, any potential treatment-resistant strain could propagate uninhibited, and present a danger not only to the patients already in the hospital, but become a threat of a general superbug outbreak. As far as the Emergency Room is concerned, full sterility could never be achieved - patients come in there directly from the outside world and bring along the microorganisms they may be carrying. Preventing the spread of these microorganisms

throughout the hospital is the primary concern, so the ventilation must be set up to blow air from outside in.

The rooms for generally admitted patients are better off with inside air going out (option 2). General admission patients are usually recovering from procedures. They are not presumed to be carriers of infectious pathogens (otherwise they would be in the infectious disease ward), and introducing microorganisms into their room would not have any added benefit of competition. Furthermore, the stress of fighting any potential infection brought in from the outside would be detrimental to a patient's recovery in general admission. A filter could be installed at the door of a positively pressurized room, making it a dual-filtered system similar to the OR, but this would be unnecessary both in terms of safety and by adding to the cost of the hospital's construction.

COMPUTER SCIENCE

- You can write and compile your code here: http://www.tutorialspoint.com/codingground.htm
- Your program should be written in Java or Python
- No GUI should be used in your program: eg., easygui in Python. All problems in POM require only text input and output. GUI usage complicates solution validation, for which we are also using *codingground* site. Solutions with GUI will have points deducted or won't receive any points at all.
- Please make sure that the code compiles and runs on http://www.tutorialspoint.com/codingground.htm before submitting it.
- Any input data specified in the problem should be supplied as user input, not hard-coded into the text of the program.
- Submit the problem in a plain text file, such as .txt, .dat, etc. **No .pdf, .doc, .docx, etc!**

Introduction: Maps

We all quite familiar with geographical maps. They depict continents, oceans, etc. on a sheet of paper. The key point here is that they "translate" 3 dimensional surface of Earth into 2 dimensional picture. By the way, this representation is not unique. You may heard about tens of different map projections. Mathematics borrows this meaning and says, for example, that a function is basically a map that gets an input (say, x) and produces an output (say, y) or "maps x to y". This sense of association is also used in computer science. In some computer languages maps are initialized with arrows which stresses this property, for example,

In *Perl* (maps are called hashes there):

%data = (1 => "one", 2 => "two");

In *Scala*:

val data = Map(1 -> "one", 2 -> "two")

Thus, we are talking about 2 things where one is associated with the other. The first thing is called "key" and the second one is called "value", i.e. key => value. So, given a key it's very easy and fast to find its value. The reverse is not true.

If you remove the values from this data structure you essentially end up with Sets that we studied last time.

The keys are usually stored in an array. As you know, you can read any Key from it very fast if you know it's location, index in our case. So, how we translate a Key into its index? That's what a hash function does. In general case it's quite complicated topic, but we can give you a taste of how it can be done. Let's get some number out of Key. If you started with an integer then you already have it. But what if we have a string? We can assign some number to each letter, for example, 'A' can be 1, 'Z' = 26, 'a' = 27, 'z' = 52. Then you sum up all the numbers.

Suppose you create an array of 10 elements. Then you compute the key value modulo 10. For example, for the word “ABC” in our example it’ll be 1+2+3 % 10 = 6 % 10 = 6.

But what happens when your key hash value is the same for different keys? It’s called a hash collision. There are multiple strategies how to deal with this. The simplest one is to store a list of key-value pairs that correspond to this hash value. Obviously this degrades performance. You just jumped with 1 operation (of computing the hash value) into a proper key place but then you have linearly scan this list to find what you’re looking for. Usually this happens when you’ve put a lot of values into a small map, i.e. your hash array is not large enough. The usual remedy in this case is to create a new bigger map and rehash all the content. This will disperse all the key-value pairs into more hash buckets and thus your lists will become smaller (remember that you modulo k, where k is your hash array length). You could have started with a large array, but then your map structure would be sparse, and you would waste a lot of memory. This is a classical trade-off between CPU (number of things you have to compute) and memory.

Similar to Sets, keys in maps are uniquely identify their values, so when you assign

data{1} = “one”

and then

data{1} = “uno”

then your map will have only one pair: 1=>”uno” which will override the previous value. Different languages have different syntax for maps. They all typically allow to add a new key-value pair, delete one, find whether a given key is already in the map, and find its associated value. Look up the maps documentation for the language of your choice to learn how to implement them in your code. (Note: in Python maps are called Dictionaries).

5 points:

You are probably familiar with a simple method to scramble your secret messages where each letter in the original text is replaced with some other letter in “ciphertext”. Your assignment this month will be to write a simple descrambler. Your program should input: original plaintext alphabet and the corresponding ciphertext alphabet, for example like this:

Original: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext: ZYXWVUTSRQPONMLKJIHGFEDCBA

Then your program should input an encrypted message and print a corresponding descrambled one. Your program should be using maps.

Solution:

(in Python)

```
import sys

# input
print("Enter original plain text alphabet: ")
orig_alphabet = sys.stdin.readline().strip()
print("Enter corresponding ciphertext alphabet: ")
cipher_alphabet = sys.stdin.readline().strip()
if len(orig_alphabet) != len(cipher_alphabet) or len(orig_alphabet) < 1:
  raise("plain and cipher alphabets must be of the same length and non empty")
print("Enter an encrypted message: ")
encrypted = sys.stdin.readline().strip()

# parsing and building decipher
orig_alphabet_list = list(orig_alphabet)
cipher_alphabet_list = list(cipher_alphabet)
decipher = {}
for i in range(len(orig_alphabet_list)):
  letter = cipher_alphabet_list[i]
  if letter in decipher:
    raise("duplicate mapping for "+letter)
  decipher[letter] = orig_alphabet_list[i]

# deciphering
print("deciphered message:")
for letter in list(encrypted):
  if letter not in decipher:
    print("unknown letter "+letter)
  else:
    ch = decipher[letter]
    print(ch, end="")
print()
```

10 points:

Problem: Scrabble-Boggle game.

This month your assignment will be to implement a particular variant of the game Boggle. Not sure if you have ever played Boggle, and if not - it's a lot of fun. In Boggle, you are given 16 letters organized as a 4x4 square. Your task is to find all 3 or more letter long words using the following rule: The letters must be adjoining in a 'chain': letter squares in the chain may be adjacent horizontally, vertically, or diagonally. For example, given this board:

Z	**E**	**G**	Z
L	Z	**O**	Z
Z	Z	Z	Z

X	**O**	**B**	Z

you could find words LEG, LEGO, EGO, GEL and BOX. Note that each letter square can be used only once per word: you cannot make word EGG in our example. To make your program simpler, all the words will have to belong to a given set of allowed words (super-abridged dictionary of English language ς). We suggest not more than 50-100 words. To make the game even more interesting, each found word will have associated numeric value, using letter value system from Scrabble. Specifically, letters from A-Z have the following corresponding values:
A-1,B-3,C-3,D-2,E-1,F-4,G-2,H-4,I-1,J-8,K-5,L-1,M-3,N-1,O-1,P-3,Q-10,R-1,S-1,T-1,U-1,V-4,W-4,X-8,Y-4,Z-10
So, LEG has a value of 4, LEGO - 5, etc.

Your program may have a built-in map of letter values, and we also suggest storing a set of allowed words. Then your assignment is: enter the 4x4 Boggle board, then figure out all possible words that belong to the set of allowed words, calculate their Scrabble value and print them in the order of decreasing value.

For example, with the board from the example above, your program would print:
BOX - 12
LEGO - 5
EGO - 4
LEG - 4
GEL - 4
(assuming all these 5 words are in allowed words set)

Solution:

Here we have a program written in Python. Inside there are two similar implementations. In the first one the dictionary is implemented using a set, just like we did last month. Here is the output for the test board from the problem statement:

```
BOZO            - 15
BOX             - 12
ZOE             - 12
LEGO            -  5
EGO             -  4
GEL             -  4
GEO             -  4
LEG             -  4
LEO             -  3
it took 219.7 s
```

So, on my laptop it took about 3.7 min to run. Why so long? Because it examined all 16 letter words, even those that start with zz and therefore do not exist in the dictionary. We can use this insight to optimize. There is a special structure called Trie, which is useful here (see https://en.wikipedia.org/wiki/Trie). It's a tree arranged by prefixes. In our example it would not have the zz branch, so we can cut our search short. So, let's rewrite our program. Instead of:

```
  dictionary = set()
```

we'll have

```
  dictionary = Trie()
```

and we'll add an additional check to abort the search. To save some time let's grab a Trie implementation off the web, e.g. from here: http://stackoverflow.com/questions/11015320/how-to-create-a-trie-in-python. Now it produce the following output:

```
  BOZO             - 15
  BOX              - 12
  ZOE              - 12
  LEGO             -  5
  EGO              -  4
  GEL              -  4
  GEO              -  4
  LEG              -  4
  LEO              -  3
  it took 0.3 s
```

Here you see that a good selection of a data structure to use sped up the program 700+ times!

```
# Any Unix system has /usr/share/dict/words file which contains a list of English words,
# one word per line. This program requires this file to be locally available. A copy of it can
be
# downloaded from  https://raw.githubusercontent.com/eneko/data-repository/master/data/words.txt
# (please rename the downloaded file to words with no .txt extension). Alternatively, you can
# create your own dictionary by creating a text file called words with one word per line.

# Let's use the upper case throughout.
#
# Since it is possible that, e.g. a 3 letter word is not in the dictionary, but a subsequent 4
letter is,
# therefore we have to examine all the possibilities. Luckily there is the upper bound for the
word length.
# It is n*n (a cell/square can only be once in a word), in our case 4*4=16.

import copy
import time
import queue
from collections import defaultdict
```

```
n = 4 # board size
m = 3 # min word length

# input
board = [[0 for x in range(n)] for y in range(n)]
board[0] = [x.upper() for x in ['Z','E','G','Z']]
board[1] = [x.upper() for x in ['L','Z','O','Z']]
board[2] = [x.upper() for x in ['Z','Z','Z','Z']]
board[3] = [x.upper() for x in ['X','O','B','Z']]

def program1():
  dictionary = set()
  with open('words', 'r') as f: # assume 1 word per line
    for line in f:
      word = line.strip().upper()
      if len(word) >= m and len(word) <= n*n: # other words cannot match anyway
        dictionary.add(word)

  words = set() # found words

  # examine if current cell adds to the word
  # and call recursively for all adjacent cells
  # assume row and col are within bounds and not visited yet
  def match_next(row, col, word, visited):
    word = word + board[row][col]
    visited[row][col] = True

    if len(word) >= m:
      if word in dictionary:
        words.add(word)

    for i in (-1, 0, 1): # all adjacent cells
      for j in (-1, 0, 1):
        if i==0 and j==0: # the same cell
          continue
        new_row = row + i
        new_col = col + j
       if new_row>=0 and new_row<n and new_col>=0 and new_col<n and not
visited[new_row][new_col]:
          match_next(new_row, new_col, word, copy.deepcopy(visited)) # must copy here,
otherwise visited will be passed by reference
                                                                    # and when the function
returns we'll see changed values
    pass

  # start examining all cells
  for row in range(n):
    for col in range(n):
      word = ""
      visited = [[False for x in range(n)] for y in range(n)]
      match_next(row, col, word, visited)
      pass

  # print(words)

  # let's score found words
  score = {'A':1,'B':3,'C':3,'D':2,'E':1,'F':4,'G':2,'H':4,'I':1,'J':8,'K':5,'L':1,'M':3,
```

```
          'N':1,'O':1,'P':3,'Q':10,'R':1,'S':1,'T':1,'U':1,'V':4,'W':4,'X':8,'Y':4,'Z':10}
  q = queue.PriorityQueue() # priority queues are essentially sorted lists
  for word in words:
    value = 0
    for letter in word:
      value += score[letter]
    q.put((-value, word)) # insert a pair into the queue: word score and word itself
                          # priority queue is sorted in descendant order (the lower number, the
higher priority),
                          # so we will trick it with negative score values
  while not q.empty():
    item = q.get()
    print("%-16s - %2d" % (item[1], -item[0]))
  pass

################################################################################

class Trie:
  def __init__(self):
    self.root = defaultdict()

  def insert(self, word):
    current = self.root
    for letter in word:
      current = current.setdefault(letter, {})
    current.setdefault("_end")

  def search(self, word):
    current = self.root
    for letter in word:
      if letter not in current:
        return False
      current = current[letter]
    if "_end" in current:
      return True
    return False

  def startsWith(self, prefix):
    current = self.root
    for letter in prefix:
      if letter not in current:
        return False
      current = current[letter]
    return True

def program2():
  dictionary = Trie()
  with open('words', 'r') as f: # assume 1 word per line
    for line in f:
      word = line.strip().upper()
      if len(word) <= n*n: # longer words cannot match anyway
        dictionary.insert(word)

  words = set() # found words

  # examine if current cell adds to the word
  # and call recursively for all adjacent cells
```

```
  # assume row and col are within bounds and not visited yet
  def match_next(row, col, word, visited):
    word = word + board[row][col]
    visited[row][col] = True

    if not dictionary.startsWith(word): # no word in dictionary starts with these letter
      return                            # abort the search

    if len(word) >= m and dictionary.search(word):
      words.add(word)

    for i in (-1, 0, 1): # all adjacent cells
      for j in (-1, 0, 1):
        if i==0 and j==0: # the same cell
          continue
        new_row = row + i
        new_col = col + j
        if new_row>=0 and new_row<n and new_col>=0 and new_col<n and not
visited[new_row][new_col]:
          match_next(new_row, new_col, word, copy.deepcopy(visited)) # must copy here,
otherwise visited will be passed by reference
                                                                     # and when the function
returns we'll see changed values
    pass

  # start examining all cells
  for row in range(n):
    for col in range(n):
      word = ""
      visited = [[False for x in range(n)] for y in range(n)]
      match_next(row, col, word, visited)
      pass

  # print(words)

  # let's score found words
  score = {'A':1,'B':3,'C':3,'D':2,'E':1,'F':4,'G':2,'H':4,'I':1,'J':8,'K':5,'L':1,'M':3,
           'N':1,'O':1,'P':3,'Q':10,'R':1,'S':1,'T':1,'U':1,'V':4,'W':4,'X':8,'Y':4,'Z':10}
  q = queue.PriorityQueue() # priority queues are essentially sorted lists
  for word in words:
    value = 0
    for letter in word:
      value += score[letter]
    q.put((-value, word)) # insert a pair into the queue: word score and word itself
                          # priority queue is sorted in descendant order (the lower number, the
higher priority),
                          # so we will trick it with negative score values
  while not q.empty():
    item = q.get()
    print("%-16s - %2d" % (item[1], -item[0]))
  pass

if __name__ == "__main__":
  start_time = time.clock()
  program1()
  print("it took %.1f s" % (time.clock() - start_time))
```

```
start_time = time.clock()
program2()
print("it took %.1f s" % (time.clock() - start_time))

print("end.")
```