PROBLEM OF THE MONTH

September, 2016

MATHEMATICS

5 points: 16 participants have signed up for the Sigma ping pong tournament. How many ways are there to pair these 16 participants in the first round?

Hint: Take an arbitrary participant and ask how many choices (s)he has for a partner in the first round. After the first pair is chosen ask any of the remaining participants the same question (or, note that after the first pair is chosen the problem is reduced to choosing a pair out of two less (14) participants)...

Answer: $\mathbf{16}!/(\mathbf{2^8}\cdot\mathbf{8}!) = \mathbf{15}!! = \mathbf{15}\cdot\mathbf{13}\cdot\mathbf{11}\cdot\ldots\cdot\mathbf{1}$

Solution: (1) Follow hint. The first player has a choice of 15 partners. The next one can choose any of 13 among the remaining players. Continuing we get $\mathbf{15}\cdot\mathbf{13}\cdot\mathbf{11}\cdot\ldots\cdot\mathbf{1}$ possibilities to pair 16 participants.

(2) Let us order 16 participants and assume that the number 1 plays with number 9, number 2 with number 10 etc. There are 16! ways to order participants. The pairing will not change if we interchange 1 with 9 or 2 with 10 etc. We get $\mathbf{16}!/\mathbf{2^8}$**.** In addition, we can simultaneously reorder first 8 and second 8 using the same permutation (the pairing will not change). Therefore, the final answer is $16!/(2^8\cdot 8!)$.

(3) The most straightforward way is to ask yourself, how many possibilities are there to choose 8 pairs out of 16 participants? Pair #1 can be selected in $C_{16}^2 = \frac{16!}{2!14!}$ ways, pair #2 in $C_{14}^2 = \frac{14!}{2!12!}$ ways, and so on. The total number of the possibilities is, $C_{16}^2 C_{14}^2 \ldots C_2^2 = \frac{16!}{2^8}$. However, this counts all possible

permutations of the obtained 8 pairs as distinct pairings, while they are in fact identical (it does not matter whether players A and B are pair #1, or pair #5). Therefore, in order to obtain the number of pairings we divide the obtained number of possibilities by $8!$, $\frac{16!}{2^8 8!} = 15 \cdot 13 \cdot 11 \ldots \cdot 3 \cdot 1$.

10 points: Prove that the product of any m subsequent integer numbers is divisible by m!
Hint: You have to show that the number $(k+1)(k+2)\ldots(k+m)/m!$ is integer. Can you assign any meaning to this number?

Solution: We have to show that the number $(k+1)(k+2)\ldots(k+m)/m!$ is integer. Let us assume that k is non-negative. Then this number is $(k+m)!/(k! \cdot m!)$which is the number of ways to choose m objects from (k+m) objects if the order of objects is not important. The number of ways is always an integer number! Therefore, the statement is proven for k non-negative. If $-m \leq k < 0$then one of the numbers we multiply is zero and the result of multiplication is equal to zero and is divisible by m! Finally, if $k < -m$ then

$$\begin{aligned}(k+1)(k+2)\ldots(k+m)/m! \\ &= (-1)^m(|k|-m)(|k|-m+1)\ldots(|k|-1)/m!) \\ &= (-1)^m(|k|-1)!/((|k|-1-m)! \cdot m!)\end{aligned}$$

The latter number up to a sign is the number of ways of choosing m objects from |k|-1 objects and is integer as well. This completes the proof for all integers.

PHYSICS

5 points:

Two very massive walls are moving toward each other with identical speeds V. A light box is initially resting on a frictionless surface between the two walls as shown. Find the speed of the block after three elastic collisions with the walls.

Hint: Consider a wall that moves towards the box at rest, with speed V. From the point of view of the wall, the box is moving towards it with speed V. After collision, the velocity of the box with respect to the wall simply changes its sign. Now you can find the velocity of the box with respect to the ground, and repeat this procedure for each collision.

Answer: 6V.

Solution: Consider a wall that moves towards the box at rest, with velocity V. From the point of view of the wall, the box is moving with velocity -V.. After the collision, the velocity of the box with respect to the wall changes its sign and becomes +V. With respect to the ground, its velocity is +2V. Now, consider the second wall that moves towards the box with velocity -V. From its point of view, The velocity of the box is +3V. After collision it changes to -3V, or -4V with respect to the ground. In this way, the speed of the box will increase by amount 2V after each collision. Therefore, after 3 collisions it will be 6V.

10 points:

A spring is hanging from the ceiling of an elevator. A technician hung a 1 kg weight onto the spring, and after it reaches equilibrium, measured that the spring stretched by 10 cm. The technician then pressed the 50th floor button, and the elevator suddenly started moving up with the constant acceleration of 1 m/s^2 and continues to move with the same acceleration. What will be the maximum additional stretching of the spring (assume that all processes are completely elastic)?

Hint:

What is the new equilibrium position of the weight, and how is it positioned with respect to this new equilibrium?

Answer: 2 cm.

Solution: In equilibrium, the spring stretches proportionally to the force applied. Originally, the force was equal to *mg* *(m=1kg, g=9.8 m/s^2)*. When the elevator starts moving with acceleration *a=1m/s^2* , the force increases by amount *ma*. Therefore, the equilibrium potion will be shifted down by length $\Delta l = (a/g) \cdot 10cm \approx 1cm$. However, the weight will not stop at that pioint, but rather will start oscillating about it. The amplitude of such oscillation is $1cm$ (since its starts motion $1cm$ above the new equilibrium position). Therefore, the lowest point that the weight will reach is $1cm$ below the new equilibrium, ot $2cm$ below the original one.

CHEMISTRY

5 points:

Once upon a time, in a small tavern in Port-Royal one old pirate John Neusilber was telling a fascinating story of his recent enterprise, when he, along with a small crew of his friends, managed to capture a huge Spanish galleon. There they found several heavy chests full of golden coins. John was boasting how brave his crew was, and how cowardly the crew of the Spanish ship surrendered.

“Believe me, this story is absolutely real. I even have some coins with me,” John said and pulled out a small sack with coins from his pocket.

“Something is very suspicious in this story,” another pirate, Prudentio, said. “Why were there almost no guards on this ship, why did they surrender so easily? I am not sure these coins are golden. Let’s ask Alberto.”

Alberto was a pirate too, but he started his career as an alchemist. After many years of tireless attempts to convert lead to gold, he eventually realized that to obtain gold from lead one has to load a lead bullet into musket’s barrel and to find someone who has no loaded musket, but who has a wallet loaded with golden coins. In other words, he left Europe, went to Caribbean and became a pirate.

Alberto took one coin and looked at it. “Interesting” - he said. “Looks like gold, but… Can I have one? I’ll tell for sure by tomorrow.”

After coming to his room, where he had a small alchemist laboratory (apparently, he was still hoping he would be capable of inventing the method of obtaining gold without bloodletting), Alberto took a bottle with *Spiritus of Niter* (modern chemists call it “nitric acid”). Alberto initially planned to prepare *Aqua Regia* (a mixture of nitric and hydrochloric acids), but at the very last moment he decided to take *Spiritus of Niter* alone. He poured it in a glass and dropped the coin there. To his surprise, a violent reaction ensued, and the coin dissolved completely. The resulting solution had a green-blue color. “A-ha, not gold,” - Alberto said, “Let’s see what else is there besides Venus”. He poured the solution in a dish and started to heat gently until the solution evaporated completely, leaving a crystalline residue. He then dissolved this residue in water and added a concentrated

solution of table salt to it. A copious white precipitate formed immediately, which was not soluble neither in *Spiritus of Niter* nor in *Aqua Regia*. ”Not bad”, Alberto said, “not bad…”

Next evening Alberto announced his verdict to impatient pirates: “Prudentio was right, the coins are not golden. That is why there were almost no guards there. But I found something Spaniards themselves didn’t probably know: these coins contain a lot of ……. “

What did Alberto mean under “Venus”, and why did he decide the coins are not a complete junk?
Hint: The number of metals alchemists knew was equal to the number of stellar bodies (the five planets, the Sun and the Moon). Check which of those metals form insoluble chlorides.

Solution:
Most metals react with acids, and the products of this reaction are gaseous hydrogen and salt. For example, iron reacts with hydrochloric acids:

$Fe + HCl = FeCl_2 + H_2$

However, some metals that are right of hydrogen in the activity series (you can google what does the “activity series of metals” mean) do not react with acids in this manner. Copper, silver, mercury, gold are right of hydrogen, so Alberto did not expect the coin to react with hydrochloric acid. However, silver and copper react with nitric acid, but no hydrogen is produced. Instead, one molecule of acid oxidizes the metal (the byproducts are nitrogen dioxide and water), and another makes a salt with the oxidized metal, for example:

$Ag + 2HNO_3 = AgNO_3 + H_2O + NO_2$

In contrast, gold does not react even with nitric acid, and it reacts only with *aqua regia* (a mixture of nitric and hydrochloric acids). That is why Alberto was disappointed when he observed a violent reaction of the coin with nitric

acid: it became clear it was not gold.
The blue color of the liquid obtained was an indication that the coin contained copper (“Venus” in alchemists jargon).
However, the question was if the coin contained only copper, or it was something else there. To figure it out, Alberto got rid of the excess of acid (by evaporating the solution obtained). He dissolved it in water and added sodium chloride (table salt). We know that copper chloride ($CuCl_2$) is soluble in water, so there was no reason to expect an exchange reaction according to this equation:

$Cu(NO_3)_2 + 2NaCl = CuCl_2 + 2NaNO_3$

However, silver chloride (AgCl) is not water soluble, so formation of the precipitate is the indication of the presence of silver nitrate in the solution:

$AgNO_3 + NaCl = AgCl + NaNO_3$

Chlorides of iron, zinc, tin, copper are soluble, lead chloride is marginally soluble. These metals, as well as silver, mercury and gold were the only metals known to alchemists (including Alberto himself). The presence of gold was already ruled out (the coin dissolved in nitric acid), mercury is liquid and highly volatile, so it is usually not a component of alloys. That means that the second metal in the coin is silver.

10 points:
Below are fragments from a textbook written in some extinct language. The Cryptography department was capable of translating the text partially, but some words, symbols, and formulas remained unclear, because the cryptographer who did that work had almost zero knowledge of Chemistry. Decipher the unclear parts of the text.

***Seogulc** is a white crystalline compound with a formula $\Psi_{\pi}P_{\xi\xi}\Omega_{\pi}$. It is non-toxic, soluble in water, and has a sweet taste. Its formula can also be written as $(\Psi P_{\xi}\Omega)_{\pi}$, and that is why **Seogulc**, as well as other*

*compounds of that type are called **glewarhyd** (a word composed of “**glew**”, Ψ, and **arhyd**, Ᵽϛ Ϙ).*

…………

*In free form, **glew**, or Ψ, is a solid that burns in gaseous Ϙϛ to produce*

ΨϘϛ (a gas) according to the equation:

Ψ + Ϙϛ → ΨϘϛ

…………..

*Ᵽϛ is a gas. Its mixture with gaseous Ϙϛ burns violently to yield **arhyd**, and the equation of this reaction is:*

ϛᵽϛ + Ϙϛ → ϛᵽϛϘ

*At room temperature, **arhyd** is a colorless transparent liquid. It reacts violently with some active metals (calcium, sodium), and, in a presence of gaseous Ϙϛ , causes corrosion of many other metals (e.g., iron, copper). It is used as a solvent for many inorganic reactions.*

Explain what “*seogulc*”, “*arhyd*”, and “*glew*” mean. Explain the meaning of the symbols Ψ, Ϙ , Ᵽ, Ꝁ, ϛ, and π.

Hint: Obviously, Ꝁ is “1”. Ψ, Ϙ , Ᵽ denote very common elements (you know them very well). Try to think, how many elements exist in a gaseous form?

Solution:

Obviously, “seogulc” is glucose ($C_6H_{12}O_6$). Its formula can be re-written as ($CH2O)_6$, hence the name “carbohydrate”, the common name for this class of compounds. From that, it becomes clear that “glew” is carbon.

Also, by looking at the equation

$$\mathcal{Y} + \mathcal{O}_{\mathcal{f}} \rightarrow \mathcal{Y}\mathcal{O}_{\mathcal{f}}$$

It becomes clear that valence of the element $\mathcal{Y}$ is multiple of the valence of the element $\mathcal{O}$.

By looking at the equation

$$\mathcal{f}\mathcal{E}_{\mathcal{f}} + \mathcal{O}_{\mathcal{f}} \rightarrow \mathcal{f}\mathcal{E}_{\mathcal{f}}\mathcal{O}$$

we conclude the valence of $\mathcal{O}$ is multiple of the valence of $\mathcal{E}$. In addition, the elements $\mathcal{E}$ and $\mathcal{O}$ are gases with formulas $\mathcal{O}_{\mathcal{f}}$ and $\mathcal{E}_{\mathcal{f}}$, it looks like $\mathcal{f}$ should be 2: firstly, no simple substances (elements) form stable tri- or tetraatomic molecules, but many of them form diatomic molecules. Secondly, if $\mathcal{f}$ is more than 2, e.g., if it is 3, then the valence of $\mathcal{O}$ should be 3, and the valence of $\mathcal{Y}$ should be 9, which is highly improbable.

Taking into account all these considerations, it is easy to conclude $\mathcal{E}$ is hydrogen, $\mathcal{O}$ is oxygen, and $\mathcal{Y}$ is carbon.

Carbon burns in oxygen to produce carbon dioxide.
Hydrogen also burns in oxygen to produce hydrogen oxide (“arhyd”, or water).
Water is a very good solvent for many chemical reactions, it reacts with active metals and causes corrosion of many other metals.

BIOLOGY

5 points:
It is known that in all living organisms three nucleotides are used to encode each amino acid. Does it mean that to encode the peptides with the total length of N amino acids the DNA segment must be at least 3N nucleotides long? Are there any exceptions to this rule, and if yes, what are the benefits and disadvantages of that?

Answer:
The maximum total length of aa sequence coded by 1000 bp DNA is almost 2,000 aa.
This is because the same segment of DNA can potentially encode more than one amino acid sequence. For example, let's consider the sequence

TATCAGAATCAGTG

If reading starts with TAT, the sequence produces the peptide

TAT CAG AAT CAG TG (etc), or

Tyr Glu Asn Gln

But if we start with 'A', the same sequence produces:

(T) ATC AGA ATC AGT G (etc), or
Ile Arg Ile Ser

If we shift the "reading frame" forward for one more nucleotide, we get

(TA) TCA GAA TCA GTG, which means the peptide sequence is:
Ser Glu Ser Val
In all three cases the nucleotide sequence is the same, but it encodes for three totally different peptide sequences.

Moreover, the opposite strand can also encode three different peptide sequences exactly in the same way as described above. In other words, the nucleotide sequence which is N nucleotides long can encode different protein chains 2N amino acids (N in one strand plus N in the complementary strand).
It is not exactly 2N, because at least one codon is a stop codon (it is a part of the sequence, but no amino acid is incorporated), and several nucleotides at the beginning of each gene are needed to initiate peptide synthesis, although they encode no amino acids.

The above described situation takes place in real living organisms rarely, because, as a rule, only one of two DNA strands encodes a protein sequence, and there is just one “reading frame”, so usually a nucleotide sequence with the length of N nucleotides produces a protein that is N/3 nucleotides long. However, in many viruses the genome is organized in such a way that some segments contain up to 3 reading frames, and the second DNA strand encodes a protein chain. That is needed to minimize the size of the viral DNA, although these cases are rare.

10 points:
The gastric brooding frog (Rheobatrachus silus), extinct since 1981, swallowed its eggs and then brooded about twenty tadpoles for 6-7 weeks in its stomach. Ultimately, the mother "give birth" by burping up her froglets. If the species evolved from a more conventional type of frog, as seems likely, propose a possible mechanism that lead to the development of this “method of birth" and identify at least three changes that would have had to take place in order to make it possible.

Answer:

a) Most likely, in distant past the frogs ate some of their eggs. This type behaviour is common among many species of fish and amphibia. Accordingly, some protective mechanisms started to develop to protect

eggs from digestion. When some eggs that were swallowed started to survive they got some advantage over other eggs, because they were better protected in mother’s stomach. As a result, this strategy gradually became predominant. In parallel, mother frog’s behaviour had also gradually evolved: she learn to swallow as many eggs as possible rather than to deposit them in the water as is now done by all other species of frogs. In addition, the following physiological changes occurred in mother frog’s organism.

b) The mother's stomach chemistry had to be radically altered during the reproductive period so as not to kill the young with her digestive enzymes and strong stomach acids.

c) Passage of the eggs from the stomach into the intestines had to be suppressed.

d) The tadpoles had to be converted from a mobile, feeding organism into one that could survive when imprisoned for weeks in a dark, crowded frog stomach.

e) The burping event had to be programmed so as not to occur too early nor too late in the tadpole development (either of which could have been fatal).

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, or as a zip archive of such files.
  No .pdf, .doc, .docx, etc!

Introduction: Multidimensional Arrays

(If you already know what single-dimensional and multidimensional arrays are, you are welcome to skip straight to the problems below)

As you know, in computer languages, values representing quantities, IDs, text, and so on, are stored in *variables*: a = 1, b = 3.14, c = "Hello World!". If we want to store the year's precipitation by day in inches, it would be inconvenient to store the data in 365 variables: day0 = 0.0, day1 = 0.5… Instead, we use a *one-dimensional array*: We *allocate* (meaning *reserve*) space for 365 variables in one long block of memory, and call the block **dailyPrecipitation**. To access the variable for each day, we write **dailyPrecipitation[i],** where **i** goes from 0 to 364. So to write or read the value for day 54, we write **dailyPrecipitation[54]**, which means "look in the 54th block of the array". Behind the scenes, when the computer reads the expression **dailyPrecipitation[54]**, it goes to the beginning of this block of memory, then moves 54 blocks over, and reads the value there. *Be careful: although in most languages, the first block is numbered 0, there are some (rare) exceptions where labeling starts at 1.*

We can extend the notion of one-dimensional arrays to two dimensions. From the perspective of writing code, it's easy. Say, if you want to record the players' positions on a checkerboard, and denote each square as 0 → empty, 1→ black piece, 2→ red piece, then you create an 8x8 *two dimensional array* of integers, call it, say, **checkerboard**. Now, to write or read the state

of square A8 (i.e. 1,8), we just write **checkerboard[0][7].** Behind the scenes, things are slightly more complicated than in one dimensional case: Computer memory is single-dimensional, so it can only reserve 1-dimensional arrays. One way that a computer can create the 8x8 2D array **checkerboard[8][8]**, is to allocate 8*8 = 64 blocks, and then map the 2D array onto these 64 blocks like this: **checkerboard[n][m]** = look in the (8*n + m) block from the beginning of **checkerboard**. Naturally, it is possible to make arrays of any dimension this way.

Different languages have different syntax for arrays. Look up the array documentation for the language of your choice to learn how to implement them in your code.

5 points:

If numbers 1 through 9 are placed in the 3x3 grid so that sums of numbers in each row, column and two major diagonals are 15, they form a magic square. An example of magic square is:

2 7 6
9 5 1
4 3 8

Write a program where you receive first 2 rows of a square from standard input. Then your program needs to figure out whether third row can be added so that a magic square is formed. If it's possible, you should output the resulting magic square. If not possible, the program should state so. Use two-dimensional array (see *Introduction*) in your solution.

Solution:

Java:

```
// fill last row of a magic square
// Java version

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class MagicSquare {
  private int[][] msquare;
  private int n;         // number of rows/column in msquare
  private int mconstant; // the magic constant (the sum)

  public MagicSquare() {
    // for debugging
```

```
    //msquare = new int[][] {{2, 7, 6}, {9, 5, 1}, {0, 0, 0}};
    //n = 3;
    //mconstant = 15;
  }
  public void input() throws IOException {
    System.out.println("Enter the first row of a magic square separating the numbers with a
space:");
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String[] numbers = br.readLine().trim().split("\\s+");

    n = numbers.length;
    msquare = new int[n][n];
    mconstant = 0;
    for(int j=0; j<n; j++) {
      msquare[0][j] = Integer.parseInt(numbers[j]);
      mconstant += msquare[0][j];
    }

    // input the rest rows except for the last one
    for(int i=1; i<n-1; i++) {
      System.out.printf("Enter the row number %d of a magic square separating the numbers with a
space:\n", i+1);
      numbers = br.readLine().trim().split("\\s+");
      for(int j=0; j<n; j++)
        msquare[i][j] = Integer.parseInt(numbers[j]);
    }
  }

  public void fillLastRow() {
    for(int col=0; col<n; col++) {
      msquare[n-1][col] = mconstant;
      for(int row=0; row<n-1; row++) {
        msquare[n-1][col] -= msquare[row][col];
      }
    }
  }

  /**
   * @param v
   * @return number of times v occurs in msquare[][]
   */
  public int findVal(int v) {
    int count = 0;
    for(int row=0; row<n; row++) {
      for(int col=0; col<n; col++) {
        if(msquare[row][col] == v)
          count++;
      }
    }
    return count;
  }

  public boolean verify() {
    for(int row=1; row<n; row++) { // skip the 1st row since we calculated mconstant from it
      int s = 0;
      for(int col=0; col<n; col++)
        s += msquare[row][col];
      if(s != mconstant)
        return false;
    }
```

```
    for(int col=0; col<n; col++) {
      int s = 0;
      for(int row=0; row<n; row++)
        s += msquare[row][col];
      if(s != mconstant)
        return false;
    }

    // main diagonal
    int s = 0;
    for(int row=0; row<n; row++)
      s += msquare[row][row];
    if(s != mconstant)
      return false;

    // secondary diagonal
    s = 0;
    for(int row=0; row<n; row++)
      s += msquare[n-row-1][row];
    if(s != mconstant)
      return false;

    // uniqueness
    for(int row=0; row<n; row++) {
      for(int col=0; col<n; col++) {
        if(findVal(msquare[row][col]) > 1)
          return false; // duplicate value
      }
    }
    return true;
  }

  public void output() {
          for(int row=0; row<n; row++) {
      for(int col=0; col<n; col++) {
        if(col != 0)
          System.out.print(", ");
        System.out.print(msquare[row][col]);
      }
      System.out.println();
         }
  }

  public static void main(String[] args) throws IOException {
    MagicSquare ms = new MagicSquare();
    ms.input();
    ms.fillLastRow();
    if(ms.verify())
      System.out.println("matrix is valid");
    else
      System.out.println("matrix is invalid");
    ms.output();
    System.out.println("end.");
  }
}
```

Python:

```
#!/usr/bin/python3
```

```
# fill last row of a magic square
# note: using numpy is totally acceptable but in this solution we'll use basic python
# solution for arbitrary N - square size

# from __future__ import print_function
import sys

class MagicSquare:
  msquare = []    # magic square - will become a 2D array which is a list of lists in python
  n = 0           # number of rows/column in msquare
  mconstant = 0   # the magic constant (the sum)

  # for debugging
  #msquare = [[2, 7, 6], [9, 5, 1], [0, 0, 0]]
  #n = 3
  #mconstant = 15

  def input(self):
    print("Enter the first row of a magic square separating the numbers with a space:")
    line = sys.stdin.readline().strip()
    numbers = [int(x) for x in line.split()]
    self.n = len(numbers)
    self.mconstant = sum(numbers)
    self.msquare = [[0 for x in range(self.n)] for x in range(self.n)] # allocate 2D array
    self.msquare[0] = numbers                                          # assign the 1st row
    # input the rest rows except for the last one
    for i in range(1,self.n-1):
      print("Enter the row number {} of a magic square separating the numbers with a
space:".format(i+1))
      line = sys.stdin.readline().strip()
      self.msquare[i] = [int(x) for x in line.split()]
    pass

  def fill_last_row(self):
    for column in range(self.n):
      self.msquare[self.n-1][column] = self.mconstant - sum([row[column] for row in self.msquare])
    pass

  def verify(self):
    for row in range(1,self.n):  # skip the 1st row since we calculated mconstant from it
      s = sum(self.msquare[row])
      if s!= self.mconstant:
        return False
      pass

    for column in range(self.n):
      s = sum([row[column] for row in self.msquare])
      if s!= self.mconstant:
        return False
      pass

    # main diagonal
    s = 0
    for i in range(self.n):
      s += self.msquare[i][i]
    if s!= self.mconstant:
      return False
```

```
    # secondary diagonal
    s = 0
    for i in range(self.n):
      s += self.msquare[self.n-i-1][i]
    if s!= self.mconstant:
      return False

    # uniqueness
    s = set()
    for i in range(self.n):
      for j in range(self.n):
        if self.msquare[i][j] in s:
          return False # duplicate value
        else:
          s.add(self.msquare[i][j])

    return True

  def output(self):
    print(*self.msquare, sep='\n')

if __name__ == "__main__":
  ms = MagicSquare()
  ms.input()
  ms.fill_last_row()
  if ms.verify():
    print("matrix is valid")
  else:
    print("matrix is invalid")
  ms.output()
  print("end.")
```

10 points:

Problem:

Your program receives from standard input an “image” of an arbitrary shape cut from an NxN square of quad-ruled paper (cutout follows grid lines). First the dimension N of the square is given. Then N lines of text are provided, where "-" is standing for "cutout", and “x” is standing for the shape. For example:

```
(0,0)   (5,0)
-x----
-xxx--
```

```
-xxx--
-xxx--
---x--
------
(0,5)   (5,5)
```

(Note: coordinates are provided here for clarity, they are not included in the input).

Your program should:

1. Find the center of mass of the shape. For the purpose of the calculation of the center of mass, presume that each little square “x” of the shape has the same mass. In the example above the center of mass is at (2, 2).
   Note: it is possible for the center of mass to be located between the coordinate lines. For example, the following shape
   ```
   ----
   -xx-
   -xx-
   ----
   ```
   has a center of mass located at (1.5, 1.5). Your program should take this into consideration.

2. Determine if the shape is contiguous (i.e. one piece). For the shape to be contiguous its constituent little squares should touch each other horizontally, vertically or diagonally. For example, the following shape:

   ```
   ----
   -x--
   --x-
   ----
   ```
   is contiguous, but this one:
   ```
   ----
   -x--
   ----
   ```

-xx-

is not.

Solution:

Java:

```
// find center of mass and determine continuity
// Java 5 version

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MassCenterJ {
  class Coordinate<T> {
    public Coordinate(T x, T y) {
      this.x = x;
      this.y = y;
    }

    T x, y;
  }

  private int n;          // number of rows/column in figure
  private int[][] figure; // we'll map 'x' and '-' -> 1 and 0

  public MassCenterJ() {
    // for debugging
    //n = 4;
    //figure = new int[][] {{0,0,0,0}, {0,1,0,0}, {0,0,0,0}, {0,1,1,0}};
  }

  public void input() throws IOException {
    System.out.println("Enter number of rows/columns:");
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    n = Integer.parseInt(br.readLine().trim());

    figure = new int[n][n];
    Pattern p = Pattern.compile("^[\\-xX]+$");
    for(int i=0; i<n; i++) {
      System.out.printf("Enter the row number %d:\n", i+1);
      String line = br.readLine().trim();
      Matcher m = p.matcher(line);
      if(!m.find()) {
        System.out.println("invalid input; try again");
        i--;
      }
      else {
        if(line.length() != n) {
          System.out.println("invalid input; try again");
          i--;
        }
```

```
      else {
        for(int j=0; j<n; j++)
          figure[i][j] = line.charAt(j) == '-' ? 0 : 1;
      }
    }
  }
}

public void output() {
  for(int row=0; row<n; row++) {
    for(int col=0; col<n; col++) {
      if(col != 0)
        System.out.print(", ");
      System.out.print(figure[row][col]);
    }
    System.out.println();
  }
}

public Coordinate<Double> findCenterMass() {
  Coordinate<Double> c = new Coordinate<Double>(0.0, 0.0);
  int count = 0;
  for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
      if(figure[i][j] > 0) {
        c.x += j;
        c.y += i;
        count++;
      }
    }
  }
  c.x /= count;
  c.y /= count;
  return c;
}

public int getSum() {
  int sum = 0;
  for(int row=0; row<n; row++) {
    for(int col=0; col<n; col++) {
      sum += figure[row][col];
    }
  }
  return sum;
}

public boolean isContiguous() {
  int i=0, j=0;
  find_first:
  for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
      if(figure[i][j] > 0)
        break find_first;
    }
  }
  List<Coordinate<Integer>> todo = new LinkedList<Coordinate<Integer>>();
  todo.add(new Coordinate<Integer>(i, j));
  while(todo.size() > 0) {
    Coordinate<Integer> c = todo.remove(0);
    i = c.x;
```

```
      j = c.y;
      figure[i][j] = 0;
      if(j+1<n && figure[i][j+1]>0) {
        todo.add(new Coordinate<Integer>(i,j+1));
        figure[i][j+1] = 0;
      }
      if(j-1>=0 && i+1<n && figure[i+1][j-1]>0) {
        todo.add(new Coordinate<Integer>(i+1,j-1));
        figure[i+1][j-1] = 0;
      }
      if(i+1<n && figure[i+1][j]>0) {
        todo.add(new Coordinate<Integer>(i+1,j));
        figure[i+1][j] = 0;
      }
      if(j+1<n && i+1<n && figure[i+1][j+1]>0) {
        todo.add(new Coordinate<Integer>(i+1,j+1));
        figure[i+1][j+1] = 0;
      }
    }
    // we zeroed all contiguous elements, so...
    int sum = getSum();
    if(sum > 0)
      return false;
    return true;
  }

  public static void main(String[] args) throws IOException {
    MassCenterJ mc = new MassCenterJ();
    mc.input();
    mc.output();
    Coordinate<Double> c = mc.findCenterMass();
    System.out.printf("Cx=%.2f, Cy=%.2f\n", c.x, c.y);
    if(mc.isContiguous())
      System.out.println("contiguous");
    else
      System.out.println("not contiguous");
    System.out.println("end.");
  }
}
```

Python:

```
#!/usr/bin/python3

# find center of mass and determine continuity

# from __future__ import print_function
import sys
import re

class MassCenter:
  n = 0       # number of rows/column in figure
  figure = [] # this will become a 2D array which is a list of lists in python
              # we'll map 'x' and '-' -> 1 and 0

  # for debugging
  #n = 6
  #figure = [[0,1,0,0,0,0], [0,1,1,1,0,0], [0,1,1,1,0,0], [0,1,1,1,0,0], [0,0,0,1,0,0],
[0,0,0,0,0,0]]
```

```
  #figure = [[0,1,0,0,0,0], [0,1,0,0,0,0], [0,0,1,0,0,0], [0,1,1,0,0,0], [0,0,1,1,1,0],
[0,0,1,1,1,0]]
  #n = 4
  #figure = [[0,0,0,0], [0,1,1,0], [0,1,1,0], [0,0,0,0]]
  #figure = [[0,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,0]]
  #figure = [[0,0,0,0], [0,1,0,0], [0,0,0,0], [0,1,1,0]]

  def input(self):
    print("Enter number of rows/columns:")
    self.n = int(sys.stdin.readline().strip())
    self.figure = [[0 for x in range(self.n)] for x in range(self.n)] # allocate 2D array

    for i in range(self.n):
      print("Enter the row number {}:".format(i+1))
      line = sys.stdin.readline().strip()
      matched = re.match("^[\-xX]+$", line)
      if matched:
        letters = list(line)
        if len(letters) != self.n:
          raise Exception("invalid input")
        self.figure[i] = [0 if x=='-' else 1 for x in letters]
      else:
        raise Exception("invalid input")
    pass

  def output(self):
    print(*self.figure, sep='\n')

  def find_center_mass(self):
    count = 0
    Cx = 0
    Cy = 0
    for i in range(self.n):
      for j in range(self.n):
        if self.figure[i][j]:
          Cx += j
          Cy += i
          count += 1
    Cx /= count
    Cy /= count
    return Cx, Cy

  def find_first(self):
    for i in range(self.n):
      for j in range(self.n):
        if self.figure[i][j]:
          return i,j
    return 0,0

  def is_contiguous(self):
    i, j = self.find_first()
    todo = [(i,j)]
    while len(todo) > 0:
      i,j = todo.pop(0)
      self.figure[i][j] = 0
      if j+1 < self.n and self.figure[i][j+1]:
        todo.append((i,j+1))
        self.figure[i][j+1] = 0
      if j-1 >= 0 and i+1 < self.n and self.figure[i+1][j-1]:
```

```
        todo.append((i+1,j-1))
        self.figure[i+1][j-1] = 0
      if i+1 < self.n and self.figure[i+1][j]:
        todo.append((i+1,j))
        self.figure[i+1][j] = 0
      if j+1 < self.n and i+1 < self.n and self.figure[i+1][j+1]:
        todo.append((i+1,j+1))
        self.figure[i+1][j+1] = 0
    s = sum(sum(self.figure,[]))
    if s:
      return False
    return True

if __name__ == "__main__":
  mc = MassCenter()
  mc.input()
  mc.output()
  Cx, Cy = mc.find_center_mass()
  print("Cx=%.2f, Cy=%.2f" % (Cx, Cy))
  if mc.is_contiguous():
    print("contiguous")
  else:
    print("not contiguous")
  print("end.")
```