PROBLEM OF THE MONTH

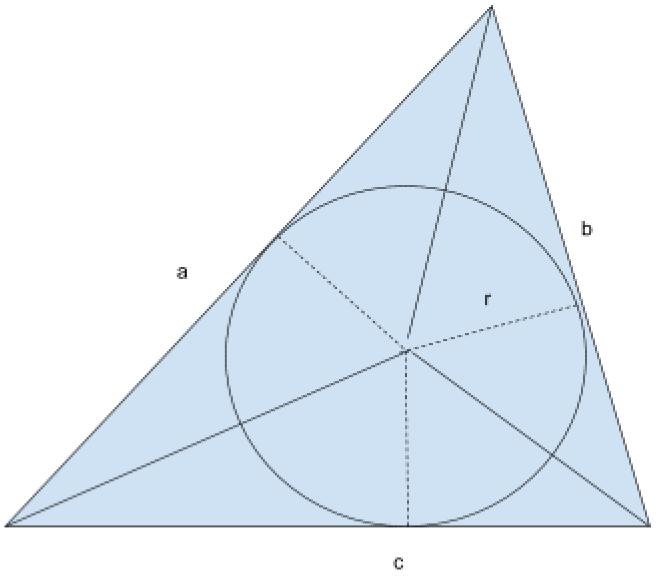
POM 2017-2018

November, 2017

MATHEMATICS

5 points: A circle is inscribed in a triangle. The perimeters of the circle and the triangle are 1cm and 10 cm, respectively. Find the area of the triangle.

Hint: Look at the picture.



Answer: $\frac{5}{2\pi}cm^2$

Solution: The picture shows how the original triangle can be cut on 3 triangles: their bases are the three sides of the big triangle (a,b and c), and they all share the center of the circle as a vertex. At each of the three touching point, the corresponding side is perpendicular to the line segment connecting that point with the center of the circle. Therefore, the height of each of the three small triangles is equal to the radius of the circle. We can now find the area of the big triangle by adding areas of the three small ones:

$$A = \frac{ar}{2} + \frac{br}{2} + \frac{cr}{2} = pr/2$$

Here $p = a + b + c = 10cm$ is the perimeter of the triangle, and $r = \frac{1}{2\pi}cm$ is the radius of the inscribed circle. Therefore,

$$A = pr/2 = \frac{5}{2\pi}cm^2$$

10 points: A sphere is inscribed in a polyhedron so that it touches all of its faces. The volume of the sphere is 9 cm^3. The surface area of the polyhedron is 50 cm^2. What is its volume?

Hint: same as for 5 pt problem. You'll need to imagine a similar construction in 3D. Note that volume of a pyramid is Bh/3 (B is the area of its base which may have any shape), h is its height.

Answer: $\frac{50}{\sqrt[3]{4\pi}} cm^3$

Solution: Similarly to the 5 pt. problem, we can split the original polyhedron onto several pyramids, each having one face of that polyhedron as as its base, and all sharing the center of the sphere as a vertex. We can find the volume of the large polyhedron as a sum of the areas of these pyramids, each being $Br/3$ (B is the area of the base, r is the radius of the inscribed sphere which is also the height of all these pyramids):

$V = AR/3$

Here A=50 cm^2 is the total surface area of the polyhedron, and R is the radius of the inscribed sphere. To find the radius note that the volume of the sphere is
$\frac{4\pi}{3} R^3 = 9cm^3$, so
$R = \frac{3}{\sqrt[3]{4\pi}} cm$.

We now can calculate the total volume of the polyhedron:

$V = AR/3 = \frac{50}{\sqrt[3]{4\pi}} cm^3$.

PHYSICS

This month Physics problems are on the mechanical work and mechanical energy conservation. You might find the following links useful.
https://en.wikipedia.org/wiki/Work_(physics)
http://hyperphysics.phy-astr.gsu.edu/hbase/ke.html
http://hyperphysics.phy-astr.gsu.edu/hbase/pegrav.html
http://hyperphysics.phy-astr.gsu.edu/hbase/frict.html

5 pt

A spring gun using a massless spring with the spring constant $k = 100\ N/m$ is loaded with the ball of the mass $m = 0.01\ kg$, so that the maximal compression of the spring is $A = 10\ cm$. What is the maximal height reached by the ball if the gun shoots strictly vertically. Neglect friction and air resistance. Assume that the free fall acceleration $g \approx 10\ m/s^2$.

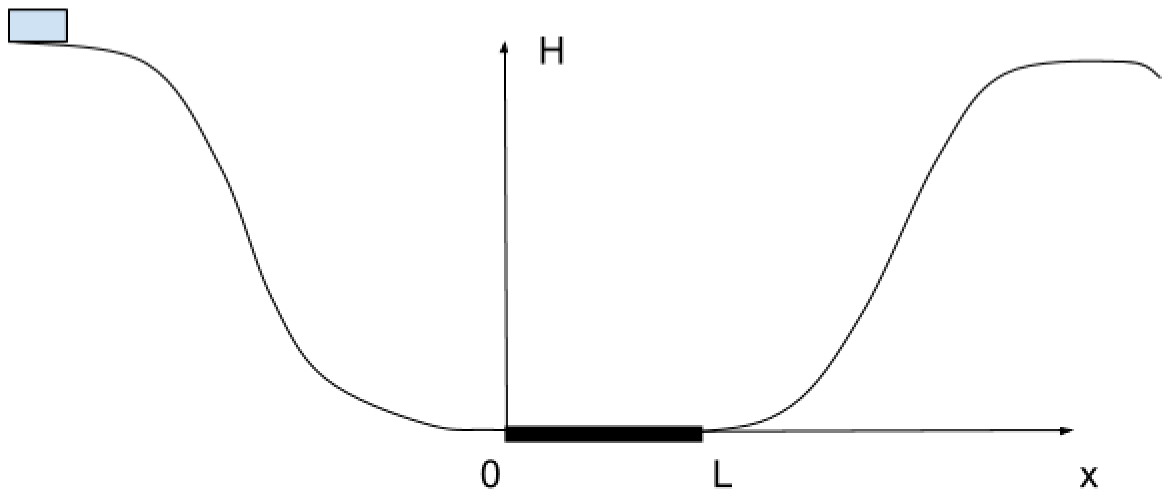
Hint: Use the conservation of energy.

Answer: $h \approx 5$ m

Solution:
The elastic energy of the compressed spring is equal to the gravitational potential energy of the ball at the highest point of its trajectory. We have $\frac{kA^2}{2} = mgh$ and $h = \frac{kA^2}{2mg} \approx \frac{100*0.1^2}{2*0.01*10} = 5$ m .

10 pt

A small block is sliding from a hill of the height $H = 5m$ down the frictionless slope towards the similar hill. There is a small horizontal patch of the surface with kinetic friction coefficient $\mu = 0.3$ of the length $L = 1m$ between the hills. At what position will the block stop?



Hint: Find the work done by friction force.

Answer: $x \approx 0.67$ m from the left end of the patch

Solution:

It is clear that the block will oscillate from left to right until all its initial potential energy will be converted into heat through the work done by friction force. The work of the friction force is negative and equal to $W = FS = -\mu mgS$, where m is the mass of the block and S is the total distance traveled by the block until it stopped. Here we used the value of the friction force $F = -\mu mg$. We have from the conservation of energy $mgH + W = mgH - \mu mgS = 0$ and find $S = H/\mu = 5/0.3 \approx 16.67$ m. This means that the block will go from left to right and back 8 times ($8 \times 2 = 16$ m) and then will go from left to right and stop at the distance of $x \approx 0.67$ m from the left end of the patch.

CHEMISTRY

5 points:

DNA means “deoxyribonucleic *acid*”, but if you take a commercially available solution of DNA and measure its pH, the solution will be nearly neutral. Similarly, cell nuclei are full of densely packed DNA, but the media is neutral there. Why does it happen, and is DNA a real acid?

Hint:

Classical (“Bronsted’s”) acids always contain a constant part (hydrogen atoms) and a variable part (this variable part may be huge and complex, especially in organic acids, and it is this part which makes acids different from each other). In solution, hydrogens usually dissociate from the acid molecule, and most biochemists prefer to (...continue by yourselves).

Solution:

From chemist’s point of view, DNA is a phosphoric acid derivative. There are many phosphate fragments in DNA, and these fragments are connected by nucleoside moieties. Two nucleosides are attached to each DNA’s phosphate, which means only one acidic hydrogen is remaining. However, the presence of even one hydrogen in each phosphate group is sufficient for DNA to display its acidic nature. Yes, DNA *is* an acid, and it is a moderately strong acid (for example, it is stronger than acetic acid).
However, it is necessary to keep in mind that deoxynucleic *acid* is a substance that is virtually insoluble in water. Pure DNA precipitates immediately, and to make it soluble, one has to convert it into a salt. When DNA reacts with some appropriate base, for example, NaOH or KOH, it, as well as any other acid, becomes a *salt* (sodium or potassium salt), and this salt is quite soluble. In other words, the acronym DNA usually means not an acid itself, but a salt of this acid, and it is not a surprize that the solution of that salt is not acidic, but neutral.

Nevertheless, we prefer to say “DNA”, not “*sodium 2’-deoxyribonucleoside phosphate*” despite the fact that the last name is more correct. Why?
First, the word “DNA”, although it is a jargon, is much easier to pronounce.
Second, biochemists usually deal with huge biological molecules that have ionic groups, and the charges of these groups are compensated by small and simple inorganic ions (sodium ions, potassium ions, chloride ions, etc). These ions are called “counter ions”, and their type usually do not affect the overall properties of biomolecules. For example, sodium and potassium salts of DNA have almost identical properties. That is why biochemists and molecular biologists prefer to “*forget”* about these ions, and focus on large biomolecule itself. For most scientists, DNA or its salt are pretty much the same, so they use the term “DNA” as an umbrella term for the acid proper and for all salts it can form. Such a terminology is quite acceptable, however, one must remember it is just a jargon, which sometimes, although very unfrequently, may lead to problems.

10 points:

In a ***L'Alpagueur*** movie, Jean-Paul Belmondo's hero uses a nitrous oxide, N_2O ("laughing gas") to put mafia bosses to sleep. To do that, he drilled holes in the floor of a trailer where the mafia meeting took place and directed a stream of laughing gas there. The gas cylinders he used for narcotizing mafia are shown on the figure 1 (he had two). The size of the trailer can be estimated based on the picture of its interior (Fig. 2).

Please, tell if the amount of the gas was sufficient to bring mafia bosses into an unconscious state.

To answer this question, assume that the gas cylinders are standard cylinders used for medical purposes, and they are full. Estimate their actual size from the attached picture. The concentration of N_2O in air that is necessary to provide a desirable effect can be googled.

If, according to your estimate, the amount of N_2O is too small or too big, how many cylinders of that size should the hero have used to achieve a desirable result?

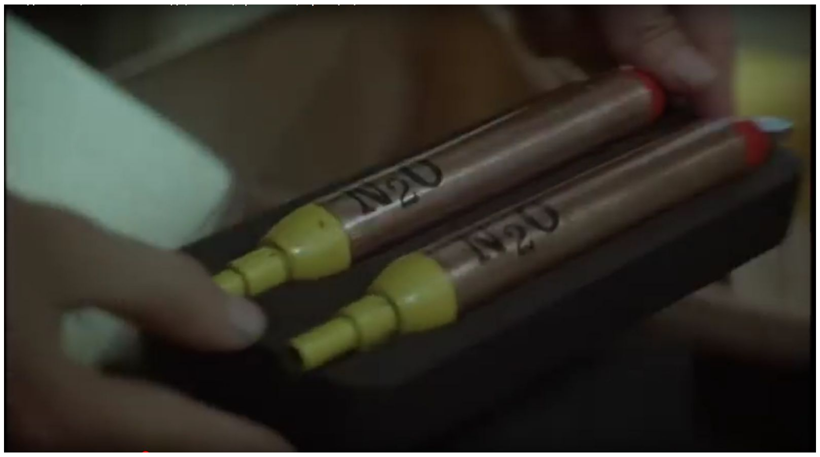


Fig. 1. The cylinders.

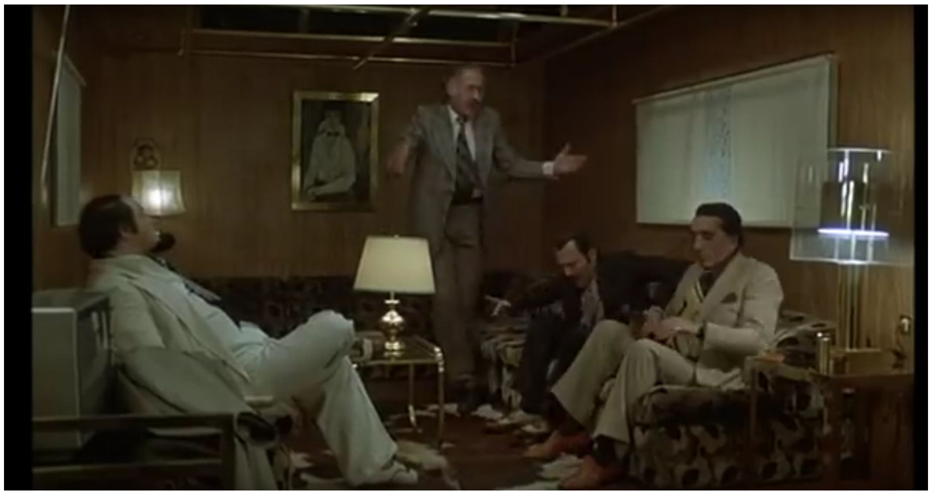


Fig. 2. The trailer. For your estimate assume one half of the trailer is seen on the picture.

Hint:

English Wikipedia contains all essential information needed to answer this question. Due to its low boiling point value, N_2O is in a gaseous state in cylinders. Calculate the gas amount assuming it obeys ideal gas laws (when the gas volume decreases twice, the pressure doubles).

Solution:

First, if our calculations are based on several estimated parameters, the accuracy of the final answer cannot be greater than the accuracy of least precise parameter. The parameters we have to estimate are: the trailer's volume, N_2O concentration that is needed to obtain a desirable effect, cylinder's size. The size of the trailer can be estimated only approximately: if we assume it is 5 by 2 by 2.5 meters, its volume is 25 m^3, however, our estimate may be incorrect, so the volume may range from 20 to 40 m^3. The exact concentration of N_2O needed to obtain a desirable effect is also not known, most likely it is 50-70% (by volume). A possibility of leakage due to working ventilation system in the trailer cannot be ruled out. In other words, we can speak only about a lower estimate, i.e. about the case when the trailer has a volume of ca 20 m^3, a ventilation system is not working, all windows and doors are closed, the gas mixes with air uniformly, and the desirable concentration is about 50%.
In that case, 10 m^3 (10 000 L) of laughing gas are needed. A standard M2 type cylinder (one of the smallest available cylinders) contains about 42 L (0.04 m^3) of gas at maximal pressure (130 bar). The size of M2 cylinders is 13 x 6 cm, and the cylinders shown on the figure 2 are a little bit longer, but their diameter is smaller, so it would be correct to assume its capacity is about 40 L. That means two cylinders would be definitely not enough to obtain a desirable effect, and the detective would have to bring 10000/40 = 250 cylinders on that size. That is a lower estimate.

BIOLOGY

5 points:

There are numerous examples of synchronized behavior in nature: claw waving (fiddler crabs), synchronized respiration (honey bees), or chewing (termites). One of the famous cases is synchronized flashing in fireflies frequently seen in Asia and North America. In the United States it is attributed to the rover firefly (*Photinus carolinus*) and can be observed in early June in, for example, Great Smoky Mountains National Park. Males emit well-synchronized periodic flashes of bright light. It is considered to be a mating mechanism to attract females. Why do these bugs need "dark intervals" between flashes? Interestingly, not all fireflies need dark intervals. For example, European fireflies emit constant light with no flashes. What can be the reason for such a difference?

Answer:

Fireflies emit light mostly to attract mates, although they also communicate for other reasons as well, such as to defend territory and warn predators away. Larvae use their glows as warning displays to communicate their distastefulness. In some firefly species, only one sex lights up. In most, however, both sexes glow; often the male will fly, while females will wait in trees, shrubs and grasses to spot an attractive male. If she finds one, she'll signal it with a flash of her own - hence the need of the “dark interval”. Several studies have shown that female fireflies choose mates depending upon specific male flash pattern characteristics. Higher male flash rates, as well as increased flash intensity, have been shown to be more attractive to females.

Fireflies produce a chemical reaction inside their bodies that allows them to light up. This type of light production is called bioluminescence. The method by which fireflies produce light is perhaps the best known example of bioluminescence. When oxygen combines with calcium, adenosine triphosphate (ATP) and the chemical luciferin in the presence of luciferase, a bioluminescent enzyme, light is produced. Unlike a light bulb, which produces a lot of heat in addition to light, a firefly's light is cold light, without a lot of energy being lost as heat.

It seems the difference between North American/Asian and European fireflies is in communication strategy: in steady glowing species, females attract males, who are not glowing; in flashing species, one bug flashes, and another bug flashes in response.

10 points:

Imagine that there is a rare disease which is caused by a mutation in a single autosomal gene. It is known that the disease develops only in individuals who have two mutant alleles (homozygots). They are unable to have children. Heterozygous (one mutant and one normal alleles) individuals are healthy and can have children. The incidence of this disease is 1 in 100,000. Do you think that this illness will completely disappear from the population in 100 generations? Why or why not?

Answer:

There is no single correct answer to this question, but there are several lines of thought which can bring you to one or the other conclusion.

1. The "disease" allele eliminated only when it is homozygous. Heterozygotes are still present. The intuitive conclusion is that heterozygotes will continue to persist after 100 generations. Let's see if we can formally demonstrate this.

   Let' s assume that the population is very large and mating happens completely randomly. Let' s denote "normal" and "disease" alleles of the gene as *A* and *a*. Let's assume that their frequencies are *p* and *q*. Then, according to Hardy-Weinberg law, frequency of different genotypes will be:

   $$\left.\begin{array}{l} AA - p^2 \\ Aa - 2pq \\ aa - q^2 \end{array}\right] \qquad (1)$$

   We know that disease appears in 1:100,000 people, which means that frequency of genotype *aa* is 1/100,000 = q^2. Therefore

   $$q = \sqrt{(1/100{,}000)} = 0.00316 \qquad (2)$$

   There are only two alleles of this gene, *A* and *a*. We know that frequency of *a* is *q*.Then frequency of allele *A* is

   $$p = 1-q \qquad (3)$$

   We know from (2) that $q = 0.00316$. Then

   $p = 1-q = 0.99684$

   Now we can calculate frequency of heterozygotes *Aa*, which are carriers of a "disease" allele *a*, using (1):

   $$2pq = 0.00316 \times 0.99684 = 0.006 \qquad (4)$$

   which makes it 600 out of 100,000.

   These calculations lead us to somewhat surprising conclusion that even when a disease coded by a recessive allele is quite rare, the proportion of disease carriers is relatively large. In this example, for each person with the disease there are 600 carriers.
   This conclusion suggests that 100 generations might not be enough to "weed-out" allele *a* from the population.
   Let's try to make formal calculations to check this.

   From (1) we can take frequencies of genotypes before the selection started (first line in the table below). We know that individuals with genotype aa do not reproduce (do not produce gametes), so let's write frequency of genotypes capable to reproduce (third line in the table).

	AA	Aa	aa
Initial frequency	p^2	$2pq$	q^2

Can produce gametes?	yes	yes	no
Frequency after selection	p^2	$2pq$	0

In the next generation frequency of allele *a* will be lower than in the initial generation, since only heterozygotes *Aa* can produce gametes with this allele. Let's call this new frequency q_1. It will be equal half of the frequency of heterozygotes Aa divided by total.

$q_1 = ½(2pq)/(p^2 + 2pq)$ (5)

After simplification and applying equitation (3)

$q_1 = q/(1+q)$ (6)

From (2), $q = 0.00316$.

Then $q_1 = 0.00315$

After 100 generation of selection $q_{100} = 0.0024$.

To calculate frequencies of genotypes in 100th generations, we can apply (1):

AA – p^2

Aa – $2pq = 0.0048$, i.e. 479 out of 100,000 individulas

aa – $q^2 = 0.0000058$, i.e. 1 out of 172,413 individuals

As you can see, frequency of individuals with this disease decreased less than two-fold over the course of 100 generations.

2. The gene sequence can mutate, producing same mutated allele (allele *a*), therefore maintaining presence of the "disease" allele in the population. If frequency of new mutation is enough to replenish alleles eliminated from reproduction, the disease will not completely disappear from the population.

   Let's make quantitative estimates.
   Frequency of new mutations is quite low: ~ 10^{-5} - 10^{-6} for most loci. This means that 1 in 100,000 to 1 in 1,000,000 gametes would carry a newly mutated allele *a* in each generation, which can produce additional heterozygote *Aa*.
   As we can see from previous discussion, frequency of allele *a* after selection can be described by equitation (6). Difference in allele frequency between initial generation and generation after one round of selection is
   $q - q_1 = \sim 0.00001 \sim 10^{-5}$.
   This is comparable to the mutation rate, suggesting that mutations in this case can maintain presence of the disease in population.

3. So far we assumed that the population is very large and that mating is random. However, in real life populations are not that large. It is possible that frequency of alleles change due to random factors. This phenomenon is called genetic drift.
   Let's imagine that a relatively small group of individuals, say, 1,000, went to Mars and started colony over there. According to our previous calculations, this group may have

just 6 carrier individuals (heterozygotes *Aa*). It is possible that for random reasons these individuals never reproduced, and allele *a* disappeared from the population.

4. It is possible that the mutation causing this disease (allele *a*) is in close proximity to a beneficial variant (allele *B* of a different gene). Since these two variant are close to each other on the chromosome, they will tend to be inherited together, i.e. individual with allele *a* will almost always have allele *B* in the neighboring gene. If B is beneficial, there will be a selective pressure toward preserving it, and this pressure will also be applied to the allele *a* when it is in heterozygote.

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!**

5 points:

You need to write a program that analyzes the results of a Tic-Tac-Toe game.

The program should enter a Tic-Tac-Toe board as a 3x3 array of X's and O's from the input. The program should print whether X won, O won, it was a draw or the position on the board is not possible.

Solution:

Python (courtesy of Victor Turbiner):

```
# The algorithm here is quite simple: Check every cell if it is the middle of a
# connected line.
# The cells can be X, O or blank.
# The board size is always constant, as is the amount of cell checks, so the
# algorithm's speed is O(1).
# X Goes first!

# Test performed on every cell (it should only be ran in the R cells as the other cells
# will always return none, but it such a short algorithm there is little performance
# benefit):
# NRN
# RRR
# NRN
def test(line,col,board):
    val = board[line][col]
    if val == " ": return None,0 # Blank cells can't form lines.
    numWins = 0 # Count amount of connected lines, if there is > 1, it's an invalid game!

    if line == 1 and board[line+1][col] == board[line-1][col] == val: # Check for vertical
line.
        numWins += 1
    if col == 1 and board[line][col+1] == board[line][col-1] == val: # Check for horizontal
line.
        numWins += 1

    if line == 1 and col == 1: # Check for diagonal.
        if board[line+1][col+1] == board[line-1][col-1] == val:
            numWins += 1
        if board[line-1][col+1] == board[line+1][col-1] == val:
            numWins += 1

    if numWins == 0:
        return None,0
    else:
        return val,numWins

def main():
    board = []

    print("Welcome to Tic-Tac-Toe analyzer!\nPlease enter board:")
    # User Input
    for i in range(3):
        line = list(input())
        try:
            assert len(line) == 3
            for c in line:
                assert c in ["X","O"," "]
        except:
            print("Bad Input")
            return

        board.append(line)

    # Count number of Os and Xs for validation purposes.
    numX = sum(l.count("X") for l in board)
    numO = sum(l.count("O") for l in board)

    assert numX == numO or numX == numO+1, "Invalid Gameboard!" # Either there is the same
amount of Xs and Os or there is one more X than Os because X goes first

    numWins = 0
    win = None
    for i,l in enumerate(board):
        for j,c in enumerate(l):
            val,nW = test(i,j,board) # Test every cell
            if val is not None:
                numWins += nW
                win = val

    if numWins == 0: # No connected lines
        if numX+numO == 9: # Board is full
            print("It was a draw")
        else: # Unfinished game
            print("Game is unfinished")
        return

    if numWins != 1: # Too many connected lines!
        print("Invalid Game!")
        return

    print("Player",win,"won the game!")

main()
```

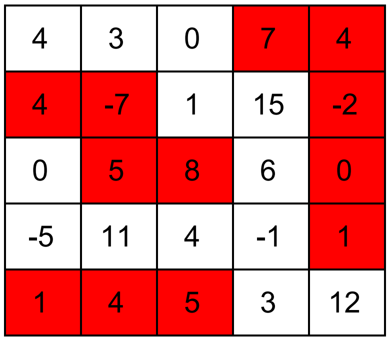
Java:

```
/*
You need to write a program that analyzes the results of a Tic-Tac-Toe game. The program should
enter a Tic-Tac-Toe board as a 3x3 array of
X’s and O’s from the input. The program should print whether X won, O won, it was a draw or the
position on the board is not possible.
*/

import java.util.Scanner;

public class TicTacToe {
  public static void main(String[] args) {
    //char[][] board = {{'X', 'X', 'X'},
    //                  {'O', 'O', 'X'},
    //                  {'X', 'O', 'O'}};
    //int n = board.length;
    //int m = board[0].length;

    int m = 3; // number or rows
    int n = 3; // number or columns
    char[][] board = new char[m][n];
    Scanner input = new Scanner(System.in);
    for(int i=0; i<m; i++) {
      System.out.printf("enter comma separated row %d of %d (use only X or O or space): ", i +
1, m);
      String row = input.nextLine();
      String[] s = row.split(",(?=([^\"]*\"[^\"]*\")*[^\"]*$)", -1);
      if(s.length != n)
        throw new IllegalArgumentException("wrong number of columns");
      for(int j=0; j<n; j++) {
        String st = s[j].trim();
        if(st.length()==1 || st.length()==0) {
          char ch = st.length()==1 ? st.charAt(0) : ' ';
          if(ch=='X' || ch=='O' || ch==' ')
            board[i][j] = ch;
          else
            throw new IllegalArgumentException("enter only X or O");
        }
        else
          throw new IllegalArgumentException("enter only X or O or space separated by commas");
      }
    }
    // I assume that after a person has won, the game is stopped, so I'm not checking for some
extra moves.

    // number of X's must be the same or 1 more than O's
    int countX = 0;
    int countO = 0;
    for(int i=0; i<n; i++) {
      for(int j=0; j<m; j++) {
        if(board[i][j] == 'X')
          countX++;
        else if(board[i][j] == 'O')
          countO++;
      }
    }
    if(countX!=countO && countX!=countO+1) {
      System.out.println("invalid combination");
      return;
    }

    // if X's occupy an entire line they won
    for(int i=0; i<n; i++) {
      int count = 0;
      for(int j=0; j<m; j++) { // horizontal
        if(board[i][j] == 'X')
          count++;
      }
      if(count == n) {
        System.out.println("X won");
        return;
      }
    }
    for(int j=0; j<m; j++) {
      int count = 0;
      for(int i=0; i<n; i++) { // vertical
        if(board[i][j] == 'X')
          count++;
      }
      if(count == m) {
        System.out.println("X won");
        return;
      }
    }
    int count = 0;
    for(int i=0; i<n; i++) { // main diagonal
      if(board[i][i] == 'X')
        count++;
    }
    if(count == n) {
      System.out.println("X won");
      return;
    }
    count = 0;
    for(int i=0; i<n; i++) { // the other diagonal
      if(board[n-1-i][i] == 'X')
        count++;
    }
    if(count == n) {
      System.out.println("X won");
      return;
    }

    // if O's occupy an entire line they won
    for(int i=0; i<n; i++) {
      count = 0;
      for(int j=0; j<m; j++) { // horizontal
        if(board[i][j] == 'O')
          count++;
      }
      if(count == n) {
        System.out.println("O won");
        return;
      }
    }
    for(int j=0; j<m; j++) {
      count = 0;
      for(int i=0; i<n; i++) { // vertical
        if(board[i][j] == 'O')
          count++;
      }
      if(count == m) {
        System.out.println("O won");
        return;
      }
    }
    count = 0;
    for(int i=0; i<n; i++) { // main diagonal
      if(board[i][i] == 'O')
        count++;
    }
    if(count == n) {
      System.out.println("O won");
      return;
    }
    count = 0;
    for(int i=0; i<n; i++) { // the other diagonal
      if(board[n-1-i][i] == 'O')
        count++;
    }
    if(count == n) {
      System.out.println("O won");
      return;
    }

    System.out.println("draw");
  }
}
```

10 points:

You have an NxN matrix, each cell of which contains an integer number (could be zero or negative). A ***path*** in such matrix can start in any cell and then go through adjacent cells to the right or down. A path could be of any length (obviously, the longest path would be 2*N-1 cells long).

3 possible paths are shown in red in the diagram below:



A ***value*** of the path is a sum of all the integer numbers contained in the cells along the path. For example, all the paths in red in the diagram above have value of 10. Your task is to find all the paths in the given matrix with a specific value.

Your program should enter the matrix's dimension and all the cell values from input, as well as enter the target path value. For each path of the given value the program should print the values of the constituent cells.

Solution:

Python:

```
"""
You have an NxN matrix, each cell of which contains an integer number (could be
zero or
negative). A path in such matrix can start in any cell and then go through
adjacent cells to the
right or down. A path could be of any length (obviously, the longest path would
be 2*N-1 cells long).
A value of the path is a sum of all the integer numbers contained in the
cells along the path.
Your task is to find all the paths in the given matrix with a specific value.
Your program should enter the matrix's dimension and all the cell values from
input, as well as
enter the target path value. For each path of the given value the program should
print the values
of the constituent cells.
"""
# Notes:
#   * Since a path can start and end anywhere, both top-down and bottom-up approaches seem
#     equivalent.
#   * This is a classical dynamical programming problem: we've got both traits:
```

```
#      - "sub-problem" - given a part of a path (with a calculated value) the solution is to
find
#       a sub-path with a specific value;
#      - We'll use recursion to reflect this. This approach, although simpler to write, can
quickly
#       end up with "stack overflow".
#       It can always be rewritten with a loop which as an additional bonus will run slightly
#       faster. Anyway, for the sake of clarity this implementation uses recursion.
#     - we can apply "memoization" - in our case it'll be a running sum of path value.
#    * Once we found path(s) with a given value we need to keep searching for continuation with
the
#     path value of 0.

# field = [[ 4,  3,  0,  7,  4],
#          [ 4, -7,  1, 15, -2],
#          [ 0,  5,  8,  6,  0],
#          [-5, 11,  4, -1,  1],
#          [ 1,  4,  5,  3, 12]]
# m = len(field)
# n = len(field[0])
# g = 10 # our goal (target path value)

m = int(input("enter number of rows: "))
n = int(input("enter number of columns: "))
field = [[0]*n]*m
for i in range(m):
  row = input("enter comma separated row %d of %d: " % (i+1, m))
  field[i] = [int(x) for x in row.split(',')]
  assert(len(field[i]) == n)
g = int(input("enter the target path value: "))

# we'll collect found paths in this sub-graph
class Tree(object):
  def __init__(self):
    self.right = None
    self.down = None
    self.i = None
    self.j = None


# i, j - current cell
# v - path value so far
# return true if a path found adding notes to res
def calc_path_value(field, m, n, g, i, j, v, res):
  v += field[i][j]
  if v == g:
    res.i = i
    res.j = j
    return True

  # descend to right
  x1 = False
  if j < n-1:
    res.right = Tree()
    res.i = i
    res.j = j
    x1 = calc_path_value(field, m, n, g, i, j+1, v, res.right)
    if not x1:
      del res.right

  # descend down
  x2 = False
  if i < m-1:
    res.down = Tree()
    res.i = i
    res.j = j
    x2 = calc_path_value(field, m, n, g, i+1, j, v, res.down)
    if not x2:
      del res.down

  return x1 or x2

def print_graph(node, field, cells):
  if node.i is not None:
    cells.append((node.i,node.j,field[node.i][node.j]))
  if hasattr(node,'right') and node.right:
    print_graph(node.right, field, cells)
  if hasattr(node,'down') and node.down:
    print_graph(node.down, field, cells)
   if not ((hasattr(node,'right') and node.right) or (hasattr(node,'down') and node.down)): #
terminal node
    print(["(%d,%d)->%2d" % (cells[k][0],cells[k][1],cells[k][2]) for k in range(len(cells))])
  cells.pop()

def find_leaves(node, leaves):
  if hasattr(node,'right') and node.right:
    find_leaves(node.right, leaves)
  if hasattr(node,'down') and node.down:
    find_leaves(node.down, leaves)
   if not ((hasattr(node,'right') and node.right) or (hasattr(node,'down') and node.down)): #
terminal node
    leaves.append((node.i, node.j))

leaves = []
for i in range(m):
  for j in range(n):
    v = 0
    root = Tree()
    found = calc_path_value(field, m, n, g, i, j, v, root)
    if found:
      print("starting from (%d,%d):" % (i,j))
      print_graph(root, field, [])
      print()
      # now that we found the shortest chains we need to check if there are continuations with
      # sub-sum 0; there can be multiple of these
      # we don't need to check all the cells, just start where the previous path ends
      # collect them into the list: leaves
      find_leaves(root, leaves)

visited = [[0]*n]*m # to avoid cycles and not to duplicate work
while len(leaves):
  i,j = leaves.pop(0)
  if not visited[i][j]:
    visited[i][j] = 1
    v = 0
    root = Tree()
    found = calc_path_value(field, m, n, 0, i, j, v, root)
    if found:
      print("starting from (%d,%d):" % (i,j))
      print_graph(root, field, [])
      print()
      # there can be more 0 value paths
      leaves2 = []
      find_leaves(root, leaves2)
      leaves += leaves2

print("end.")
```

Java:

```
/*
     You  have  an  NxN  matrix,  each  cell  of  which  contains  an  integer  number  (could
be  zero  or
    negative).  A   path   in  such  matrix  can  start  in  any  cell  and  then  go  through
adjacent  cells  to  the
     right  or  down.   A   path  could  be  of  any  length  (obviously,  the  longest  path
would  be  2*N-1  cells long).
     A  value   of  the  path  is  a   sum  of  all  the  integer  numbers  contained  in  the
cells  along  the  path.
      Your  task  is  to  find  all the  paths  in  the  given  matrix  with  a   specific
value.
     Your  program  should  enter  the  matrix's  dimension  and  all  the  cell  values  from
input,  as  well  as
     enter  the  target  path  value.  For  each  path  of  the  given  value  the  program
should  print  the  values
    of  the  constituent  cells.

    Notes:
       * Since a path can start and end anywhere, both top-down and bottom-up approaches seem
equivalent.
      * This is a classical dynamical programming problem: we've got both traits:
          - "sub-problem" - given a part of a path (with a calculated value) the solution is to
find a sub-path with a specific value;
              - We'll use recursion to reflect this. This approach, although simpler to write,
can quickly end up with "stack overflow".
                 It can always be rewritten with a loop which as an additional bonus will run
slightly faster. Anyway, for the sake of
              clarity this implementation uses recursion.
          - we can apply "memorization" - in our case it'll be a running sum of path value.
        * Once we found path(s) with a given value we need to keep searching for continuation
with the path value of 0.
*/

import java.util.ArrayList;
import java.util.Scanner;

class Tree {
  int i, j;
  Tree right;
  Tree down;
}
```

```
class Pair<T> {
  T i, j;
  Pair(T i, T j) {
    this.i = i;
    this.j = j;
  }
}

class Triple<T> {
  T i, j, k;
  Triple(T i, T j, T k) {
    this.i = i;
    this.j = j;
    this.k = k;
  }
}

public class PathSum {
  // i, j - current cell
  // v - path value so far
  // return true if a path found adding notes to res
  private static boolean calc_path_value(int[][] field, int m, int n, int g, int i, int j, int
v, Tree res) {
    v += field[i][j];
    if(v == g) {
      res.i = i;
      res.j = j;
      return true;
    }

    // descend to right
    boolean x1 = false;
    if(j < n-1) {
      res.right = new Tree();
      res.i = i;
      res.j = j;
      x1 = calc_path_value(field, m, n, g, i, j + 1, v, res.right);
      if(!x1)
        res.right = null;
    }

    // descend down
    boolean x2 = false;
    if(i < m-1) {
      res.down = new Tree();
      res.i = i;
      res.j = j;
      x2 = calc_path_value(field, m, n, g, i + 1, j, v, res.down);
      if(!x2)
        res.down = null;
    }

    return x1 || x2;
  }

  private static void print_graph(Tree node, int[][] field, ArrayList<Triple<Integer>> cells) {
    if(node != null)
      cells.add(new Triple<>(node.i, node.j, field[node.i][node.j]));
    if(node.right != null)
      print_graph(node.right, field, cells);
    if(node.down != null)
      print_graph(node.down, field, cells);
    if(node.right==null && node.down==null) { // terminal node
      for(Triple<Integer> cell : cells)
        System.out.printf("(%d,%d)->%2d, ", cell.i, cell.j, cell.k);
      System.out.println();
    }
    cells.remove(cells.size()-1);
  }

  private static void find_leaves(Tree node, ArrayList<Pair<Integer>> leaves) {
    if(node.right != null)
      find_leaves(node.right, leaves);
    if(node.down != null)
      find_leaves(node.down, leaves);
    if(node.right==null && node.down==null) // terminal node
      leaves.add(new Pair<>(node.i, node.j));
  }

  public static void main(String[] args) {
    int[][] field = {{ 4,  3,  0,  7,  4},
                     { 4, -7,  1, 15, -2},
                     { 0,  5,  8,  6,  0},
                     {-5, 11,  4, -1,  1},
                     { 1,  4,  5,  3, 12}};
    //int m = field.length;
    //int n = field[0].length;
    //int g = 10; // our goal (target path value)

    Scanner input = new Scanner(System.in);
    System.out.print("enter number of rows: ");
    int m = Integer.parseInt(input.nextLine());
    System.out.print("enter number of columns: ");
    int n = Integer.parseInt(input.nextLine());
    System.out.print("enter the target path value: ");
    int g = Integer.parseInt(input.nextLine());
    for(int i=0; i<m; i++) {
      System.out.printf("enter comma separated row %d of %d: ", i + 1, m);
      String row = input.nextLine();
      String[] s = row.split(",(?=([^\"]*\"[^\"]*\")*[^\"]*$)", -1);
      if(s.length != n)
        throw new IllegalArgumentException("wrong number of columns");
      for(int j=0; j<n; j++) {
        String st = s[j].trim();
        field[i][j] = Integer.parseInt(st);
      }
    }

    ArrayList<Pair<Integer>> leaves = new ArrayList<>();
    for(int i=0; i<m; i++) {
      for(int j=0; j<n; j++) {
        int v = 0;
        Tree root = new Tree();
        boolean found = calc_path_value(field, m, n, g, i, j, v, root);
```

```
        if(found) {
          System.out.printf("starting from (%d,%d):\n", i, j);
          print_graph(root, field, new ArrayList<>());
          System.out.println();
           // now that we found the shortest chains we need to check if there are continuations
with sub - sum 0
          // there can be multiple of these
          // we don't need to check all the cells, just start where the previous path ends
          // collect them into the list: leaves
          find_leaves(root, leaves);
        }
      }
    }

    System.out.println("0 paths:");
    boolean[][] visited = new boolean[m][n]; // to avoid cycles and not to duplicate work // by
default initialized to false
    while(leaves.size() > 0) {
      Pair<Integer> pair = leaves.remove(0);
      int i = pair.i;
      int j = pair.j;
      if(!visited[ i][j]) {
        visited[i][j] = true;
        int v = 0;
        Tree root = new Tree();
        boolean found = calc_path_value(field, m, n, 0, i, j, v, root);
        if(found) {
          System.out.printf("starting from (%d,%d):\n", i, j);
          print_graph(root, field, new ArrayList<>());
          System.out.println();
          // there can be more 0 value paths
          ArrayList<Pair<Integer>> leaves2 = new ArrayList<>();
          find_leaves(root, leaves2);
          leaves.addAll(leaves2);
        }
      }
    }
    System.out.println("end.");
  }
}
```