PROBLEM OF THE MONTH

November, 2020

2020 - 2021

MATHEMATICS

5 points:

Sophia said she knew a natural number such that the product of all of its divisors including 1 and the number itself is a number ending with exactly 42 zeros. Can she be right? If yes, give an example of such a number. If no, give proof that such numbers do not exist.

Hint: One gets 10 by multiplying 2 and 5.

Answer: yes

Solution: For example, the number 5×2^{41} has exactly 42 divisors having 5 as a factor: $5 \times 2^0, 5 \times 2, 5 \times 2^2, \ldots, 5 \times 2^{41}$. Therefore, the product of these divisors ends with 42 zeros. Other divisors do not have 5 as a factor and, thus, the product of all divisors of 5×2^{41} also ends with exactly 42 zeros. Can you think of at least one more number with the same property?

10 points:

You are performing the following magic trick. You step out of the room and one of the spectators writes down a sequence of 101 digits on a blackboard. Your assistant, who is present in the room, then covers two subsequent digits in the sequence with a black square. After that you enter the room and, without communicating with anybody, brilliantly guess the two covered digits in the correct order. To be sure that the trick will work 100% of the time you gave precise instructions to your assistant before the show. What could those instructions have been?

Hint: By covering digits at places N and N+1 the assistant communicates the number N to the magician. The number N is a two-digit number and theoretically can be sufficient to encode two covered digits.

Answer: see the solution

Solution: The only information the assistant communicates to the magician by covering digits at places N and N+1 is the number N. Let us count all digits of the trick starting from 00 and ending at 100, so that N can be from 00 to 99. The two-digit number $N = 10m + n$, where m, n are digits 0 to 9.

Here is how the trick might work. The assistant can compute (quickly) the sum of all digits standing on even places and find the last digit of the result m. Then the assistant repeats this for all digits standing on odd places and find the last digit of the result n. The assistant then covers the digits $N = 10m + n$ and $N + 1$. The magician then knows m and n and can restore the covered digits by computing the sum of all digits on odd places and restoring the covered digits occupying the odd place and then doing the same for digits on even places.

PHYSICS

5 points: Power 40 hp (horse-powers) is needed to drive a car at constant speed 60 mph on a horizontal road. How much power is needed to drive the same car at speed 90 mph, on the same road? Assume that the energy efficiency of the car stays the same and that most of the work done by the engine is against air drag.
(To get background information about drag force, check Wiki, or any other source, e.g. https://en.wikipedia.org/wiki/Drag_equation)

Hint: Note that power = force times speed.

Answer: 135 hp

Solution: Note that power = force $\times$ speed, and drag force is proportional to speed squared, v^2. Therefore, power is proportional to v^3 so it should be multiplied by factor $(90/60)^3 = 27/8$. This gives $40hp \times 27/8 = 135\ hp$.

10 points: Rotor (propeller) of Robinson R22 helicopter needs to spin at 600 RPMs (revolutions per minute) for it to lift from the ground. Suppose that you want to build a similar helicopter for use on Mars. The problem is, the density of the atmosphere on Mars is only 2% of that on Earth. Fortunately, gravitation acceleration there is $3.7m/s^2$, rather than our $g = 9.8m/s^2$. Engineers from your team managed to cut the mass of the helicopter by a factor of 2, leaving all its dimensions the same. Estimate how fast the rotor should spin for this helicopter to be able to lift from the surface of Mars.

Hint: gravity force should be balanced by lift force of the propeller. The latter is proportional to speed of rotation squared.

Answer: about 1840 RPM

Solution: Gravity force should be balanced by the lift force acting on the rotor. The latter is proportional to the angular speed squared, and to density of the atmosphere (since all dimensiones of the helicopter are the same). This means that

$$mg = const\ \rho_{air}\varpi^2$$

Hence

$$\varpi = const\sqrt{mg/\rho_{air}} = 600\ RPM\ \sqrt{0.5 / 0.02\ (3.7/9.8)} \approx 1840\ RPM$$

CHEMISTRY

This month, the topic is: **Electrochemistry**
IMPORTANT! In this PoM season, we do an experiment: each month, an online lecture will be given. This lecture may be helpful for those who want to solve Chemistry PoMs, although it is not supposed to provide direct hints.
This month, the lecture will be on Nov 22 morning. At 11:00, a Zoom conference will start where October PoM solutions will be discussed. After that, approximately at 11:30, the lecture starts.
To join the Zoom conference, use this link:
https://us02web.zoom.us/j/4817690592?pwd=T2djSjRETEpDSHFZdWJpYlBTYzdjQT09
Meeting ID: 481 769 0592
Passcode: 879615
If you are unable to connect, email to mark.lukin@gmail.com
The lecture recording is available in our youtube channel:
https://www.youtube.com/channel/UC7e5BDljoP5619Ula2fF4zQ/

5 points:

To test their ability to survive in the wild, a group of students decided to take a five-day hike through the Adirondack Mountains using only a printed map. As a precaution, they took only one smartphone, but agreed to use it just in a case of emergency. At the end of the fifth day, the guys realized that they were lost. They took out a smartphone, but it was completely discharged. They had a car charger, but it was useless because there was no car or even a 12V battery to plug it in.
Tom was the first to get up the next morning, and while the boys were starting the fire and the girls preparing breakfast, he gathered up empty cans and began to rub them with wet sand and cut them into small squares.
“Do you folks have coins? A penny is best. Girls, you definitely have cotton pads, I need a dozen of them."

Two hours later, the smartphone was charged, the students downloaded a map that quickly led them to the nearest town.
How did Tom manage to charge the phone?
What else did he have to use besides the items listed above?

Hint:

The hint is “Voltaic pile”

Answer:

When two different metals are in contact with a solution of some electrolyte, a galvanic cell forms, which means more electronegative metal becomes positively charged, and less electronegative metal becomes negatively charged. If you take copper and zinc plates, wet some porous material with some electrolyte, for example, sodium chloride solution, and put it in between these disks, you may detect a voltage of ~0.8V. That is a classical galvanic cell. If you assemble several cells in a pile, you get so called Voltaic pile (named after Alessandro Volta, another Italian scientist who invented it), and the maximal voltage between the ends of that pile will be 0.8 times the number of cells. How could the students find copper and zinc? If they had new pennies, that was not a big problem, because they have a zinc core. The only thing they had to do was to *completely* remove copper coating from one side of each coin. The word "completely" is important, because residual copper may affect battery's efficiency. However, copper removal is not easy, and the best way to do that could be to find a flat piece of sandstone (which is possible in Adirondack). Sometimes, only the coins made of pure copper (or brass) are available. In that case, another metal could be iron, for example, iron from empty cans. In that case, it is important to clean the iron surface from the oxide and from other contaminations, which can be done by wet sand. The iron-copper couple produces somewhat lower voltage, and that may require the Voltaic pile to be longer.

The idea to use another common metal, aluminium, is not that good, despite the fact that aluminium foil is easy to find and the Al-Cu couple produces, theoretically, higher voltage. The problem is that aluminium forms a very hard and dense oxide layer, which makes it inert. The voltaic pile made of Al-Cu cells may be unreliable, although theoretically possible.

Solution:

10 points:

To demonstrate that some metals react with acids, whereas others do not, a teacher put an iron nail and a piece of copper wire into the same glass beaker and poured 10% aqueous HCl into it (the iron nail and copper wire didn't touch each other, and their tips were protruding from the beaker). Bubbles of the hydrogen gas began to form at the nail's surface, and the nail immediately began to "dissolve", whereas the copper surface remained unchanged, and no gas formation was observed.

Look, if I leave this setup for several hours, the iron nail will "dissolve" nearly completely. However, it is possible to affect the outcome of this reaction. I can do that without adding any extra chemicals to the solution in the beaker. First, I can make the iron nail to react even faster; Second, I can stop the reaction of the naIl with HCl; Third, I can invert the reaction, so the nail will remain unchanged, whereas the copper wire will start to react with HCl. I am going to do that just by doing something with the tips of the iron nail and copper wire that are protruding from the beaker."

How can the teacher do that? Will it be accompanied by some additional visual effects?

Hint:

What if a teacher allows electrical current to flow freely from one piece of metal to another, or forces the current to flow in an opposite direction?

Solution:

When iron comes to contact with some asid, each hydrogen ions (H^+) takes one electron from the piece of iron. When two electrons are taken away, the following events occur. First, two hydrogen atoms (H, without a plus sign) form at the iron surface, which recombine to produce one hydrogen molecule, H_2. Second, one iron ion, Fe^{2+} goes to the solution to keep the electric charge of the piece of iron exactly zero. Chemists record that process as

$$Fe + 2\ HCl \rightarrow FeCl_2 + H_2$$

In other words, formation of hydrogen makes the iron rod positively charged, and “dissolution” iron cations just keeps the rod electrically neutral. What if we withdraw electrons from iron using some different means? For example, if we connect the iron rod to a piece of a copper wire. Copper is more electronegative, so the electrons will flow to copper, and iron will start to dissolve *without forming the hydrogen gas”*. That is important: the iron nail will be dissolving faster, but formation of the hydrogen gas will essentially stop. Instead, the gas will start to form at the surface of copper. It is not a surprise: since copper becomes slightly negative, H^+ can take an excess of electrons from it. However, the overall charge of the copper never becomes positive, because it quickly replenishes the electron loss by withdrawing them from iron. In other words, the processes of hydrogen formation and iron dissolution become separated in space: hydrogen forms (mostly) at the surface of copper, but chemical transformation of iron to iron chloride occurs, as usual (before the Decemped PoM lecture, we will do that experiment).
Can we accelerate that process even more? Yes, if we attach a battery to both metals in such a way that electrons will be being forcibly pumped from iron to copper. Can we stop the process of the reaction of iron with the acid? Yes, if we will be compensating the electrons lost by the iron rod. In that case, hydrogen gas will be forming at the iron surface as usual, but no iron dissolution will occur, because the electrons will be pumped into the iron nail, and it will remain slightly negatively charged. Obviously, in that case, the second side of that battery is attached to the copper wire, which will start to release copper ion to the solution (a totally unexpected reaction, because we know that copper does not react with HCl).

BIOLOGY

As we predicted during the Sigma-Opening ceremony in 2018, the developers of CRISPR-Cas9 technology got the Nobel prize, although that happened just one month ago. To celebrate this event, we declare this month a **CRISPR-Cas9 month**. Both problems are related to this subject.
IMPORTANT! To help you solve these problems, there will be a Biology PoM lecture.
This month, the lecture will be on Nov 21 at 11:00 AM via Zoom conference.
To join the Zoom conference, use this link:
https://us02web.zoom.us/j/4817690592?pwd=T2djSjRETEpDSHFZdWJpYlBTYzdjQT09
Meeting ID: 481 769 0592
Passcode: 879615
If you are unable to connect, email to mark.lukin@gmail.com
The lecture recording is available in our youtube channel:
https://www.youtube.com/channel/UC7e5BDljoP5619Ula2fF4zQ/

5 points:

A bioinformatic researcher performed analysis of a genome of some newly discovered bacteria, whose genes had virtually nothing in common with any other bacteria. In this genome, the researcher found strange segments with repeating motifs:

Sequence 1:

...TGGGTTTGAACCCGTCGTTGCGGGTTGAAGA(...)TGATTTAACCCGTGCTATGCGGGTTGAGCT
...

Sequence 2:

...AAAGTCCGACGGACTTATCCGACGGAATC(...)ATATCCCGACGGGCTAGTCCGACGGAAGC…

Sequence 3:

...GGACTCTTTGGCTTACTTTGGCAATC(...)ACACCCTTTGGCCTAGTCACTTTGGCTTTGGC…

The researcher concluded that all these repeats are the CRISPR cassettes. Is this conclusion correct for each of those sequences?

Hint:

For a CRISPR cassette to work, its non-variable parts must form so called stem-loop (a.k.a. hairpin) structures. What is needed for that?

Answer:

The non-variable part of CRISPR is that RNA that can fold to form a stem-loop, or hairpin structure. To make that possible, a sequence of that segment has to be something like:

```
AATCCGNNNCGGATT
```

which folds as follows:

```
AATCCGN
|||||| N
TTAGGCN
```

In this scheme, vertical lines denote a Watson-Crick base pair.

It is easy to see that the sequence #1 has such motifs, and it may be, potentially, a CRISPR cassette.

In the sequence #3, both the first and the second heptanucleotide segments are identical, and they, obviously, cannot form a stem-loop.

The sequence #2 is more complex. Each pair of yellow segments is almost self complementary, so it may form a stem-loop, although with some internal bulge. However, importantly, each heptanucleotide segment in the sequence #2 is capable of forming a small stem-loop:

```
GGG
|| A
CCC
```

This hairpin (spem-loop) is known to be very stable, so it is very likely that that segment will form four mini-hairping instead of one bigger. So the correct answer is **#1**.

10 points:

As we know, the CRISPR-Cas9 technology, which was awarded the Nobel prize one month ago, allows genome editing directly in a living organism. That creates excellent opportunities for treatment of various diseases caused by defects in genomic DNA. Which diseases listed below can be treated with CRISPR-Cas9 technology?

1. Sickle cell anemia;
2. Beta-thalassemia;
3. Huntington disease;
4. Hemophilia;

5. Down’s syndrome;
6. MELAS syndrome;
7. Cancer.

Explain your answers.

Hint:

Currently the CRISPR-Cas9 technology is capable of fixing (or changing) some single gene in some fraction of all cells in an organism or a tissue.

Answer:

The best target of CRISPR-Cas9 would be a disease that is caused by a single mutation in one protein, which makes that protein non-functional. Good examples are sickle cell anemia GAG codon changing to GTG in one concrete position), beta-thalassemia (non-deletion forms), hemophilia. CRISPR-Cas9 can easily cut out the segment with the defect and replace it with a correct sequence. These diseases do not require all cells in the affected tissue to be fixed, if only a fraction of them is fixed, that may work. Another possibility is to take the affected tissue (for example, bone marrow, treat it with CRISPR-Cas9 and reintroduce it back to the patient.

Other diseases may pose a greater challenge. Thus, MELAS is caused by mutations in mitochondrial DNA, and the delivery of CRISPR-Cas9 to mitochondria (which is surrounded with a double membrane) is very problematic. In addition, some cellular mechanisms CRISPR-Cas9 relies upon, such as including homology recombination, may be absent or working differently in mitochondria.

Huntington disease may also pose some challenge, because it is caused by spontaneous expansion of CAG repeats in the gene that encodes the protein called hungtintin. This protein is necessary for normal functioning of neural cells, and it contains a relatively short segment of CAG repeats, so the goal is not to eliminate CAGs completely, but to make that segment reasonably short. That is not easy to do using CRISPR-Cas9, because the standard length of the variable part of CRISPR is shorter that the normal lengths of CAG repeats, so that technology cannot “see” this defect.

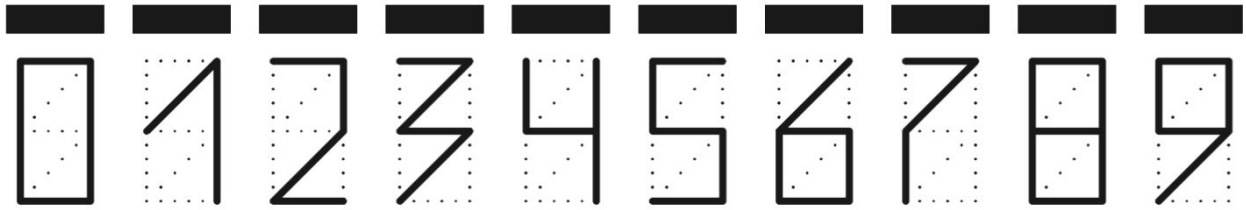
Down syndrome is resulted by trisomy 21 (duplication of the 21th chromosome), which is virtually impossible to fix using CRISPR-Cas9.

Cancer therapy requires that ALL cancer cells be killed. That is hard to achieve using CRISPR-Cas9, especially, taking into account that many processes are deregulated in cancer cells, and the delivery of drugs into a tumor is a separate task. In addition, keeping in mind high division and high mutation rate of cancer cells, a natural selection of cells that are resistant to putative CRISPR-Cas9 treatment may occur. That doesn’t mean that CRISPR-Cas9 cannot be used as a component of some future cancer treatment procedure, but that is another story.

LINGUISTICS

5 points:

In some of the countries in Europe the numbers in zip-codes used to be written according to the following template in order to allow for automatic recognition:



The system checked one of the 9 intervals (2 diagonal, 6 on the outside of the box, 1 horizontal interval inside the box) to determine whether it is filled out or not. In order to determine which digit is written, the system had to be programmed to check a certain set of these intervals, and based on the result of these checks, it had to determine which digit was written.

Question 1: What is the minimum number of intervals that had to be checked to determine the digit?
Question 2: If the digits 2, 3, 6, and 9 had the following way of writing, what would be the minimal number of checks?

Hint:

Answer:

Question 1: 4
Question 2: 6

Solution:

Question 1: The minimal number of checks is 4. To achieve the recognition, one has to check:

1) Middle horizontal interval (the only difference between 0 and 8)
2) Upper right interval
3) Lower left interval
4) Lower slanted interval

Question 2:

1) Middle horizontal interval (the only difference between 0 and 8)
2) Upper right interval (difference between 6 and 8)
3) Lower left interval (difference between 8 and 9; 5 and 6)

Using these 3 checks, we will be able to distinguish between the following groups: 0; 1; 6 ; 7; 8; 2,4,6; 3,5. To distinguish between 3 and 5 one needs to check either the upper left or the upper slanted intervals, but it would help with the 2,4,6 group. To distinguish between those, one would need about 2 checks. Therefore, in total one would need 6 checks.

10 points:

Below is a list of translated words from a language in northeast Asia. Each word below uses the concept of **stress**, where one syllable in a word is emphasized over the others. In the examples below, the stressed syllable is **bolded**. (The symbol **ə** below indicates a **short vowel** that occurs for example in English at the beginning of the word *about*, or is represented by *e* in the word *taken*.),

Original	English	Original	English
tatul	fox	**pun**ta	liver
nət**gəl**qin	hot	qe**tum**gən	relative
nu**ra**qin	far	**piw**tak	(to) pour
gəlgən	skin	nə**mit**qin	handy
neqəqin	fast	**tum**gətum	friend
nə**səq**qin	cold	**tət**ka	walrus
tap**la**ngətkən	he sews shoes	**kət**til	forehead
kəmgətək	(to) curl up	qal**pu**qal	rainbow
itək	(to) be	kə**pi**rik	hold (a child) in one's hands
pa**qət**kuk	(to) gallop	tə**vi**tatətkən	I work
nilgəqinat	white (pl.)	**pin**təvəlngək	to attack (someone)

Using this information, set the proper stress mark on the following words. You may bold, italicise, or set an accent mark to indicate stress on a syllable.

Original	English
sawat	lasso
pantawwi	fur boots
nəktəqin	sturdy
gətgan	late autumn
nəminəm	soup stock
nirvəqin	spicy
puygən	spear
tilmətil	eagle
wiruwir	salmon
wintatək	(to) help
nəmalqin	good

yaqyaq	seagull
yatək	(to) show up
tavitətkən	I will work
pintətkən	He is attacking (someone)
tayəsqəngki	in the evening

Hint:

Answer:

Original	English
sawat	lasso
pan**taw**wi	fur boots
nəktəqin	sturdy
gətgan	late autumn
nə**mi**nəm	soup stock
nirvəqin	spicy
puygən	spear
tilmətil	eagle
wiruwir	salmon
win**ta**tək	(to) help
nə**mal**qin	good
yaqyaq	seagull
yatək	(to) show up
ta**vi**tətkən	I will work
pin**tət**kən	He is attacking (someone)
ta**yəs**qəngki	in the evening

Solution:

1) 2-syllable words always have a stress on the 1st syllable.
2) For 3- or more syllable words, the stress is either on the 1st or on the 2nd syllable. The only difference in the shape of the syllables between these two groups is the shape of the 2nd syllable: if the stress is on the 1st syllable, the 2nd syllable always has a shape [Consonant+ə]; we don't see such shape of the 2nd syllable for the words with the stress on the 2nd syllable.
3) Notice that none of the 2-syllable words have a syllable of the shape [Consonant+ə] stressed!
4) Hence, we can deduce a pattern: the stress falls on the 2nd syllable if it's not of the shape [Consonant+ə]. If it is of that shape, then the stress falls on the 1st syllable. This pattern is true for 3- or more syllable words. For two syllable words, we can just stipulate that the stress is always initial, or alternatively we can say, that the stress in this language is never final -- we don't have enough data to understand what is actually important here!

The answer is the following:

Original	English
sawat	lasso
pan**taw**wi	fur boots
nəktəqin	sturdy
gətgan	late autumn
nə**mi**nəm	soup stock
nirvəqin	spicy
puygən	spear
tilmətil	eagle
wiruwir	salmon
win**ta**tək	(to) help
nə**mal**qin	good
yaqyaq	seagull
yatək	(to) show up
ta**vi**tətkən	I will work
pin**tət**kən	He is attacking (someone)
ta**yəs**qəngki	in the evening

COMPUTER SCIENCE

- Your program should be written in Java or Python-3
- No GUI should be used in your program: eg., easy gui in Python
- All the input and output should be via files with specified in the problem statement
- Java programs should be submitted in a file with extension .java; Python-3 programs should be submitted in a file with extension .py.
  No .txt, .dat, .pdf, .doc, .docx, etc. Programs submitted in incorrect format will not receive any points!

Introduction:

Consider the game of Connect4 (see rules in https://en.m.wikipedia.org/wiki/Connect_Four).

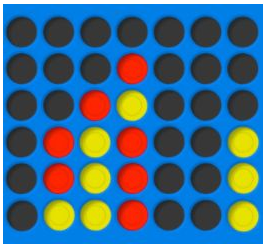
In this assignment as an input you will be given a position in the Connect4 game where Red goes next.

Input:

- The input of your program is a file **input.txt**
- The file contains 6 lines, each line being 7 characters long with the position of Connect4 board.
- The character is one of the following:
  - Letter O: Empty space
  - Letter R: Red disk
  - Letter Y: Yellow disk
- The first line represents the top of the board. The last line represents the bottom of the board.

For example, file input.txt containing below text represents the game position in the picture on the right:

```
OOOOOOO
OOOROOO
OORYOOO
ORYROOY
ORYROOY
OYYROOY
```



Output:

- The output of your program is a file **output.txt**
- A move in the game is specified by the number that represents the column where a Red disk will be dropped
- 1 represents the left-most column, 7 represents the right-most column

5 points:

Write a program that given a position in input.txt finds the move that brings an immediate win to the Red.

Output should contain:

- number 1-7, corresponding to the move
- X - if the win for Red in one move is not possible
- Z - if position is such that the one of the sides has already won

Solution:

Java:

```
/*
  Connect 4 - 5 pointer

Even though the complete solution exists, it's quite long ;-)
So, we'll use a "brute force" approach because of its simplicity and because we only need to
solve for 1 or 2 moves, i.e. the number of possible combinations is not large.
A win can be achieved by either completing a horizontal or vertical or "left" or "right"
diagonals. We'll check these moves separately.
Note: we will not check if a given position is valid.

algorithm:
  1) check if red or yellow has already won                                    => Z
     again, we assume a correct position, so the winning sequence must end at top
  2) try all possible red moves and see if it results in a win
  3) otherwise, the red win in 1 step is not possible                          => X
*/

import java.io.*;
import java.util.*;
import java.util.stream.Stream;

public class Connect4 {
  static final int m = 6; // rows
  static final int n = 7; // columns
  static final Set<Character> validChars = new HashSet<>(Arrays.asList('R', 'Y', 'O'));
  char[][] board = {{'O','O','O','O','O','O','O'}, // 0
                    {'O','O','O','R','O','O','O'}, // 1
                    {'O','O','R','Y','O','O','O'}, // 2
                    {'O','R','Y','R','O','O','Y'}, // 3
                    {'O','R','Y','R','O','O','Y'}, // 4
                    {'O','Y','Y','R','O','O','Y'}}; // m-1

  void input(String fname) throws IOException {
    try(BufferedReader br = new BufferedReader(new FileReader(fname))) {
      for(int i=0; i<m; i++) {
        String line = br.readLine().trim();
```

```
        char[] chars = line.toCharArray();
        if(chars.length != n)
          throw new IllegalArgumentException(String.format("there must be %d columns", n));
        // Stream<Character> x = new String(chars).chars().mapToObj(ch->(char)ch);
        if(!(new String(chars).chars().mapToObj(ch->(char)ch)).allMatch(ch ->
validChars.contains(ch)))
        // if(!Stream.of(chars).allMatch(ch -> { System.out.println(ch); return
validChars.contains(ch); }))
          throw new IllegalArgumentException("only R, Y and O are allowed");
        board[i] = chars;
      }
    }
  }

  void output(String fname, String answer) throws IOException {
    try(BufferedWriter bw = new BufferedWriter(new FileWriter(fname))) {
      bw.write(String.format("%s\n", answer));
    }
  }

  // check if the disk at [i,j] is in a winning position
  boolean haveWon(int i, int j) {
    char symbol = board[i][j];
    if(symbol!='R' && symbol!='Y')
      throw new IllegalArgumentException("symbol must be either R or Y");

    // horizontal
    int k1 = j - 1;
    while(k1>=0 && board[i][k1]==symbol)
      k1--;
    int k2 = j + 1;
    while(k2<n && board[i][k2]==symbol)
      k2++;
    if(k2-k1-1 >= 4)
      return true;

    // vertical
    k1 = i - 1;
    while(k1>=0 && board[k1][j]==symbol)
      k1--;
    k2 = i + 1;
    while(k2<m && board[k2][j]==symbol)
      k2++;
    if(k2-k1-1 >= 4)
      return true;

    // "left" diagonal
    int i2 = i - 1;
    int j2 = j - 1;
    while(i2>=0 && j2>=0 && board[i2][j2]==symbol) {
      i2--;
      j2--;
    }
    int i3 = i + 1;
    int j3 = j + 1;
    while(i3<m && j3<n && board[i3][j3]==symbol) {
      i3++;
```

```
    j3++;
  }
  if(i3-i2-1 >= 4)
    return true;

  // "right" diagonal
  i2 = i - 1;
  j2 = j + 1;
  while(i2>=0 && j2<n && board[i2][j2]==symbol) {
    i2--;
    j2++;
  }
  i3 = i + 1;
  j3 = j - 1;
  while(i3<m && j3>=0 && board[i3][j3]==symbol) {
    i3++;
    j3--;
  }
  if(i3-i2-1 >= 4)
    return true;

  return false;
}

// get the row number of the topmost disk for each column
int[] findTop() {
  int[] top = new int[n];
  Arrays.fill(top, -1); // we'll use -1 as an "empty" marker
  for(int j=0; j<n; j++) {
    int i = 0;
    while(i<m && board[i][j]=='O')
      i++;
    if(i < m)
      top[j] = i;
  }
  return top;
}

// check if red or yellow has already won
// we assume a correct position, so the winning sequence must end at top
char checkWon(int[] top) {
  for(int j=0; j<n; j++) {
    if(top[j]>=0 && haveWon(top[j], j))
      return 'Z';
  }
  return '_';
}

// return array of (i,j) of all possible next moves
ArrayList<ArrayList<Integer>> getAllMoves(int[] top) {
  ArrayList<ArrayList<Integer>> a = new ArrayList<>();
  for(int j=0; j<n; j++) {
    if(top[j] == 0)
      ; // full column
    else if(top[j] < 0)
      a.add(new ArrayList<>(Arrays.asList(m-1, j)));
    else
```

```
        a.add(new ArrayList<>(Arrays.asList(top[j]-1, j)));
      }
      return a;
    }

    ArrayList<Object> move1step(char disk, int[] top) {
      // try all possible moves and see if it results in a win
      ArrayList<ArrayList<Integer>> firstMoves = getAllMoves(top);
      for(ArrayList<Integer> item : firstMoves) {
        int i = item.get(0);
        int j = item.get(1);
        board[i][j] = disk; // try move
        if(haveWon(i, j)) {
          board[i][j] = 'O'; // undo
          return new ArrayList<>(){{ add(j); }};
        }
        board[i][j] = 'O'; // undo
      }

      // otherwise, the win is not reachable
      return new ArrayList<>(Arrays.asList('X', firstMoves));
    }

    ArrayList<Object> check1step(char disk) {
      // check if red or yellow has already won
      int[] top = findTop();
      char answer = checkWon(top);
      if(answer == 'Z')
        return new ArrayList<>(){{ add(answer); }};

      // Red tries all possible moves and see if it results in a win in 1 step
      return move1step(disk, top);
    }

    Object move2steps(char disk) {
      ArrayList<Object> x = check1step(disk);
      Object answer = x.get(0);
      ArrayList<ArrayList<Integer>> firstRedMoves = (ArrayList<ArrayList<Integer>>)x.get(1);
      if(answer instanceof Integer)
        return answer;
      assert(answer instanceof Character && (char)answer=='X');

      // if not then for each red move
      //   yellow tries all possible moves; if wins then this red move is not taken
      ArrayList<ArrayList<Integer>> redMoves = new ArrayList<>(); // safe moves such that does
not result in yellow's immediate win
      char opDisk = 'Y'; // opponent's disk, yellow in our case
      if(disk == 'Y')
        opDisk = 'R';
      for(ArrayList<Integer> item : firstRedMoves) {
        int i = item.get(0);
        int j = item.get(1);
        board[i][j] = disk; // try move
        int[] top2 = findTop();
        x = move1step(opDisk, top2);
        answer = x.get(0);
        if(answer instanceof Character && (char)answer == 'X')
```

```
        redMoves.add(new ArrayList<>(Arrays.asList(i,j)));
      board[i][j] = 'O'; // undo
    }

    // if remaining set of red moves is empty then yellow always wins
    if(redMoves.isEmpty())
      return 'Y';

    // now we try all combinations if red could move twice
    // and find the winning combinations
    // if for any such 2 combinations the 1st move is the same but the 2nd is different
    // then yellow could not block both of them, therefore red wins anyway after 2nd step
    for(ArrayList<Integer> item : redMoves) {
      int i = item.get(0);
      int j = item.get(1);
      board[i][j] = disk; // try move
      boolean won = false;
      int[] top2 = findTop();
      ArrayList<ArrayList<Integer>> moves = getAllMoves(top2);
      for(ArrayList<Integer> item2 : moves) {
        int i2 = item2.get(0);
        int j2 = item2.get(1);
        board[i2][j2] = disk;
        if(haveWon(i2, j2)) {
          if(!won)
            won = true;
          else { // 2nd win for different j2
            board[i2][j2] = 'O'; // undo
            board[i][j] = 'O'; // undo
            return j;
          }
        }
        board[i2][j2] = 'O';
      }
      board[i][j] = 'O'; // undo
    }

    // otherwise, the red win is not possible in 2 moves
    return 'X';
  }
}

public class Connect4_5 {
  public static void main(String[] args) throws Exception {
    Connect4 connect4 = new Connect4();
    connect4.input("input.txt");
    ArrayList<Object> ret = connect4.check1step('R');
    Object answer = ret.get(0);
    if(answer instanceof Character) {
      System.out.println("answer = " + (char)answer);
      connect4.output("output.txt", String.valueOf(answer));
    }
    else if(answer instanceof Integer) {
      int x = (int)answer + 1;
      System.out.println("answer = " + x);
      connect4.output("output.txt", String.valueOf(x));
    }
```

```
    else
      throw new Exception("bug: answer should have been either char or int");
  }
}
```

Python:

```
"""
  Connect 4 - 5 pointer

Even though the complete solution exists, it's quite long ;-)
So, we'll use a "brute force" approach because of its simplicity and because we only need to
solve for 1 or 2 moves,
i.e. the number of possible combinations is not large.
A win can be achieved by either completing a horizontal or vertical or "left" or "right"
diagonals. We'll check these
moves separately.
Note: we will not check if a given position is valid.

algorithm:
  1) check if red or yellow has already won                                    => Z
     again, we assume a correct position, so the winning sequence must end at top
  2) try all possible red moves and see if it results in a win
  3) otherwise, the red win in 1 step is not possible                          => X
"""
import re

class Connect4:
  m = 6 # rows
  n = 7 # columns
  # board = [['O','O','O','O','O','O','O'], # 0
  #          ['O','O','O','R','O','O','O'], # 1
  #          ['O','O','R','Y','O','O','O'], # 2
  #          ['O','R','Y','R','O','O','Y'], # 3
  #          ['O','R','Y','R','O','O','Y'], # 4
  #          ['O','Y','Y','R','O','O','Y']] # m-1
  board = []

  def input(self, fname):
    with open(fname) as in_file:
      for i in range(self.m):
        line = in_file.readline().strip()
        chars = [ch for ch in line]
        assert(len(chars) == self.n)
        assert(all(ch in ['R', 'Y', 'O'] for ch in chars))
        self.board.append(chars)

  def output(self, fname, answer):
    with open(fname, "w") as out_file:
      out_file.write(f"{answer}\n")

  # check if the disk at [i,j] is in a winning position
  def have_won(self, i, j) -> bool:
    symbol = self.board[i][j]
    assert(symbol=='R' or symbol=='Y')

    # horizontal
    k1 = j - 1
```

```
  while k1>=0 and self.board[i][k1]==symbol:
    k1 -= 1
  k2 = j + 1
  while k2<self.n and self.board[i][k2]==symbol:
    k2 += 1
  if k2-k1-1 >= 4:
    return True

  # vertical
  k1 = i - 1
  while k1>=0 and self.board[k1][j]==symbol:
    k1 -= 1
  k2 = i + 1
  while k2<self.m and self.board[k2][j]==symbol:
    k2 += 1
  if k2-k1-1 >= 4:
    return True

  # "left" diagonal
  i2 = i - 1
  j2 = j - 1
  while i2>=0 and j2>=0 and self.board[i2][j2]==symbol:
    i2 -= 1
    j2 -= 1
  i3 = i + 1
  j3 = j + 1
  while i3<self.m and j3<self.n and self.board[i3][j3]==symbol:
    i3 += 1
    j3 += 1
  if i3-i2-1 >= 4:
    return True

  # "right" diagonal
  i2 = i - 1
  j2 = j + 1
  while i2>=0 and j2<self.n and self.board[i2][j2]==symbol:
    i2 -= 1
    j2 += 1
  i3 = i + 1
  j3 = j - 1
  while i3<self.m and j3>=0 and self.board[i3][j3]==symbol:
    i3 += 1
    j3 -= 1
  if i3-i2-1 >= 4:
    return True

  return False

# get the row number of the topmost disk for each column
def find_top(self):
  top = [None]*self.n
  for j in range(self.n):
    i = 0
    while i<self.m and self.board[i][j] == 'O':
      i += 1
    if i < self.m:
      top[j] = i
```

```
  return top

# check if red or yellow has already won
# we assume a correct position, so the winning sequence must end at top
def check_won(self, top):
  for j in range(self.n):
    if top[j] is not None and self.have_won(top[j], j):
      return 'Z'

# return array of (i,j) of all possible next moves
def get_all_moves(self, top):
  a = []
  for j in range(self.n):
    if top[j] == 0:
      pass # full column
    elif top[j] is None:
      a.append((self.m-1, j))
    else:
      a.append((top[j]-1, j))
  return a

def move_1step(self, disk, top):
  # try all possible moves and see if it results in a win
  first_moves = self.get_all_moves(top)
  for i, j in first_moves:
    self.board[i][j] = disk # try move
    if self.have_won(i, j):
      self.board[i][j] = 'O' # undo
      return j, None
    self.board[i][j] = 'O' # undo

  # otherwise, the win is not reachable
  return 'X', first_moves

def check_1step(self, disk):
  # check if red or yellow has already won
  top = self.find_top()
  answer = self.check_won(top)
  if answer == 'Z':
    return answer

  # Red tries all possible moves and see if it results in a win in 1 step
  return self.move_1step(disk, top)

def move_2steps(self, disk):
  answer, first_red_moves = self.check_1step(disk)
  if type(answer) == int:
    return answer
  assert(answer == 'X')

  # if not then for each red move
  #   yellow tries all possible moves; if wins then this red move is not taken
  red_moves = [] # safe moves such that does not result in yellow's immediate win
  op_disk = 'Y' if disk == 'R' else 'R' # opponent's disk, yellow in our case
  for i, j in first_red_moves:
    self.board[i][j] = disk # try move
    top2 = self.find_top()
```

```
      answer, _ = self.move_1step(op_disk, top2)
      if answer == 'X':
        red_moves.append((i,j))
      self.board[i][j] = 'O' # undo

    # if remaining set of red moves is empty then yellow always wins
    if len(red_moves) == 0:
      return 'Y'

    # now we try all combinations if red could move twice
    # and find the winning combinations
    # if for any such 2 combinations the 1st move is the same but the 2nd is different
    # then yellow could not block both of them, therefore red wins anyway after 2nd step
    for i, j in red_moves:
      self.board[i][j] = disk # try move
      won = False
      top2 = self.find_top()
      moves = self.get_all_moves(top2)
      for i2, j2 in moves:
        self.board[i2][j2] = disk
        if self.have_won(i2, j2):
          if not won:
            won = True
          else: # 2nd win for different j2
            self.board[i2][j2] = 'O' # undo
            self.board[i][j] = 'O' # undo
            return j
        self.board[i2][j2] = 'O'
      self.board[i][j] = 'O' # undo

    # otherwise, the red win is not possible in 2 moves
    return 'X'

if __name__ == '__main__':
  connect4 = Connect4()
  connect4.input("input.txt")
  answer, _ = connect4.check_1step('R')
  if type(answer) == int:
    answer += 1
  print(f"answer = {answer}")
  connect4.output("output.txt", answer)
```

10 points:

Write a program that given a position in input.txt finds the move that either brings an immediate win to the Red or leads to a win in the next move, while avoiding a loss in the next move by Yellow.

Output should contain:

- number 1-7, corresponding to the move
- X - if the win for Red in one or two moves is not possible
- Y - if a loss for Red in unavoidable
- Z - if position is such that the one of the sides has already won

Solution:

Java:

```
/*
  Connect 4 - 10 pointer

This is a direct extension to the 5-pointer problem.

algorithm:
  1) check if red or yellow has already won                                    => Z
     again, we assume a correct position, so the winning sequence must end at top
  2) Red tries all possible moves and see if it results in a win in 1 step     => j
  3) if not then for each red move
       yellow tries all possible moves; if wins then this red move is not taken
  4) if remaining set of red moves is empty then yellow always wins            => Y
  5) now we try all combinations if red could move twice
     and find the winning combinations
     if for any such 2 combinations the 1st move is the same but the 2nd is different
     then yellow could not block both of them, therefore red wins anyway after 2nd step
  6) otherwise, the red win is not possible in 2 moves                         => X
*/

import java.io.*;
import java.util.*;
import java.util.stream.Stream;

public class Connect4 {
  static final int m = 6; // rows
  static final int n = 7; // columns
  static final Set<Character> validChars = new HashSet<>(Arrays.asList('R', 'Y', 'O'));
  char[][] board = {{'O','O','O','O','O','O','O'}, // 0
                    {'O','O','O','R','O','O','O'}, // 1
                    {'O','O','R','Y','O','O','O'}, // 2
                    {'O','R','Y','R','O','O','Y'}, // 3
                    {'O','R','Y','R','O','O','Y'}, // 4
                    {'O','Y','Y','R','O','O','Y'}}; // m-1

  void input(String fname) throws IOException {
    try(BufferedReader br = new BufferedReader(new FileReader(fname))) {
      for(int i=0; i<m; i++) {
        String line = br.readLine().trim();
        char[] chars = line.toCharArray();
        if(chars.length != n)
          throw new IllegalArgumentException(String.format("there must be %d columns", n));
        // Stream<Character> x = new String(chars).chars().mapToObj(ch->(char)ch);
        if(!(new String(chars).chars().mapToObj(ch->(char)ch)).allMatch(ch ->
validChars.contains(ch)))
        // if(!Stream.of(chars).allMatch(ch -> { System.out.println(ch); return
validChars.contains(ch); }))
          throw new IllegalArgumentException("only R, Y and O are allowed");
        board[i] = chars;
      }
    }
  }
```

```
void output(String fname, String answer) throws IOException {
  try(BufferedWriter bw = new BufferedWriter(new FileWriter(fname))) {
    bw.write(String.format("%s\n", answer));
  }
}

// check if the disk at [i,j] is in a winning position
boolean haveWon(int i, int j) {
  char symbol = board[i][j];
  if(symbol!='R' && symbol!='Y')
    throw new IllegalArgumentException("symbol must be either R or Y");

  // horizontal
  int k1 = j - 1;
  while(k1>=0 && board[i][k1]==symbol)
    k1--;
  int k2 = j + 1;
  while(k2<n && board[i][k2]==symbol)
    k2++;
  if(k2-k1-1 >= 4)
    return true;

  // vertical
  k1 = i - 1;
  while(k1>=0 && board[k1][j]==symbol)
    k1--;
  k2 = i + 1;
  while(k2<m && board[k2][j]==symbol)
    k2++;
  if(k2-k1-1 >= 4)
    return true;

  // "left" diagonal
  int i2 = i - 1;
  int j2 = j - 1;
  while(i2>=0 && j2>=0 && board[i2][j2]==symbol) {
    i2--;
    j2--;
  }
  int i3 = i + 1;
  int j3 = j + 1;
  while(i3<m && j3<n && board[i3][j3]==symbol) {
    i3++;
    j3++;
  }
  if(i3-i2-1 >= 4)
    return true;

  // "right" diagonal
  i2 = i - 1;
  j2 = j + 1;
  while(i2>=0 && j2<n && board[i2][j2]==symbol) {
    i2--;
    j2++;
  }
  i3 = i + 1;
```

```
  j3 = j - 1;
  while(i3<m && j3>=0 && board[i3][j3]==symbol) {
    i3++;
    j3--;
  }
  if(i3-i2-1 >= 4)
    return true;

  return false;
}

// get the row number of the topmost disk for each column
int[] findTop() {
  int[] top = new int[n];
  Arrays.fill(top, -1); // we'll use -1 as an "empty" marker
  for(int j=0; j<n; j++) {
    int i = 0;
    while(i<m && board[i][j]=='O')
      i++;
    if(i < m)
      top[j] = i;
  }
  return top;
}

// check if red or yellow has already won
// we assume a correct position, so the winning sequence must end at top
char checkWon(int[] top) {
  for(int j=0; j<n; j++) {
    if(top[j]>=0 && haveWon(top[j], j))
      return 'Z';
  }
  return '_';
}

// return array of (i,j) of all possible next moves
ArrayList<ArrayList<Integer>> getAllMoves(int[] top) {
  ArrayList<ArrayList<Integer>> a = new ArrayList<>();
  for(int j=0; j<n; j++) {
    if(top[j] == 0)
      ; // full column
    else if(top[j] < 0)
      a.add(new ArrayList<>(Arrays.asList(m-1, j)));
    else
      a.add(new ArrayList<>(Arrays.asList(top[j]-1, j)));
  }
  return a;
}

ArrayList<Object> move1step(char disk, int[] top) {
  // try all possible moves and see if it results in a win
  ArrayList<ArrayList<Integer>> firstMoves = getAllMoves(top);
  for(ArrayList<Integer> item : firstMoves) {
    int i = item.get(0);
    int j = item.get(1);
    board[i][j] = disk; // try move
    if(haveWon(i, j)) {
```

```
        board[i][j] = 'O'; // undo
        return new ArrayList<>(){{ add(j); }};
      }
      board[i][j] = 'O'; // undo
    }

    // otherwise, the win is not reachable
    return new ArrayList<>(Arrays.asList('X', firstMoves));
  }

  ArrayList<Object> check1step(char disk) {
    // check if red or yellow has already won
    int[] top = findTop();
    char answer = checkWon(top);
    if(answer == 'Z')
      return new ArrayList<>(){{ add(answer); }};

    // Red tries all possible moves and see if it results in a win in 1 step
    return move1step(disk, top);
  }

  Object move2steps(char disk) {
    ArrayList<Object> x = check1step(disk);
    Object answer = x.get(0);
    ArrayList<ArrayList<Integer>> firstRedMoves = (ArrayList<ArrayList<Integer>>)x.get(1);
    if(answer instanceof Integer)
      return answer;
    assert(answer instanceof Character && (char)answer=='X');

    // if not then for each red move
    //   yellow tries all possible moves; if wins then this red move is not taken
    ArrayList<ArrayList<Integer>> redMoves = new ArrayList<>(); // safe moves such that does
not result in yellow's immediate win
    char opDisk = 'Y'; // opponent's disk, yellow in our case
    if(disk == 'Y')
      opDisk = 'R';
    for(ArrayList<Integer> item : firstRedMoves) {
      int i = item.get(0);
      int j = item.get(1);
      board[i][j] = disk; // try move
      int[] top2 = findTop();
      x = move1step(opDisk, top2);
      answer = x.get(0);
      if(answer instanceof Character && (char)answer == 'X')
        redMoves.add(new ArrayList<>(Arrays.asList(i,j)));
      board[i][j] = 'O'; // undo
    }

    // if remaining set of red moves is empty then yellow always wins
    if(redMoves.isEmpty())
      return 'Y';

    // now we try all combinations if red could move twice
    // and find the winning combinations
    // if for any such 2 combinations the 1st move is the same but the 2nd is different
    // then yellow could not block both of them, therefore red wins anyway after 2nd step
    for(ArrayList<Integer> item : redMoves) {
```

```
      int i = item.get(0);
      int j = item.get(1);
      board[i][j] = disk; // try move
      boolean won = false;
      int[] top2 = findTop();
      ArrayList<ArrayList<Integer>> moves = getAllMoves(top2);
      for(ArrayList<Integer> item2 : moves) {
        int i2 = item2.get(0);
        int j2 = item2.get(1);
        board[i2][j2] = disk;
        if(haveWon(i2, j2)) {
          if(!won)
            won = true;
          else { // 2nd win for different j2
            board[i2][j2] = 'O'; // undo
            board[i][j] = 'O'; // undo
            return j;
          }
        }
        board[i2][j2] = 'O';
      }
      board[i][j] = 'O'; // undo
    }

    // otherwise, the red win is not possible in 2 moves
    return 'X';
  }
}

public class Connect4_10 {
  public static void main(String[] args) throws Exception {
    Connect4 connect4 = new Connect4();
    connect4.input("input.txt");
    Object answer = connect4.move2steps('R');
    if(answer instanceof Character) {
      System.out.println("answer = " + (char)answer);
      connect4.output("output.txt", String.valueOf(answer));
    }
    else if(answer instanceof Integer) {
      int x = (int)answer + 1;
      System.out.println("answer = " + x);
      connect4.output("output.txt", String.valueOf(x));
    }
    else
      throw new Exception("bug: answer should have been either char or int");
  }
}
```

Python:

```
"""
  Connect 4 - 10 pointer

This is a direct extension to the 5-pointer problem.

algorithm:
  1) check if red or yellow has already won                                    => Z
     again, we assume a correct position, so the winning sequence must end at top
```

```
  2) Red tries all possible moves and see if it results in a win in 1 step         => j
  3) if not then for each red move
       yellow tries all possible moves; if wins then this red move is not taken
  4) if remaining set of red moves is empty then yellow always wins                => Y
  5) now we try all combinations if red could move twice
     and find the winning combinations
     if for any such 2 combinations the 1st move is the same but the 2nd is different
     then yellow could not block both of them, therefore red wins anyway after 2nd step
  6) otherwise, the red win is not possible in 2 moves                             => X
"""

import re

class Connect4:
  m = 6 # rows
  n = 7 # columns
  # board = [['O','O','O','O','O','O','O'], # 0
  #          ['O','O','O','R','O','O','O'], # 1
  #          ['O','O','R','Y','O','O','O'], # 2
  #          ['O','R','Y','R','O','O','Y'], # 3
  #          ['O','R','Y','R','O','O','Y'], # 4
  #          ['O','Y','Y','R','O','O','Y']] # m-1
  board = []

  def input(self, fname):
    with open(fname) as in_file:
      for i in range(self.m):
        line = in_file.readline().strip()
        chars = [ch for ch in line]
        assert(len(chars) == self.n)
        assert(all(ch in ['R', 'Y', 'O'] for ch in chars))
        self.board.append(chars)

  def output(self, fname, answer):
    with open(fname, "w") as out_file:
      out_file.write(f"{answer}\n")

  # check if the disk at [i,j] is in a winning position
  def have_won(self, i, j) -> bool:
    symbol = self.board[i][j]
    assert(symbol=='R' or symbol=='Y')

    # horizontal
    k1 = j - 1
    while k1>=0 and self.board[i][k1]==symbol:
      k1 -= 1
    k2 = j + 1
    while k2<self.n and self.board[i][k2]==symbol:
      k2 += 1
    if k2-k1-1 >= 4:
      return True

    # vertical
    k1 = i - 1
    while k1>=0 and self.board[k1][j]==symbol:
      k1 -= 1
    k2 = i + 1
```

```
  while k2<self.m and self.board[k2][j]==symbol:
    k2 += 1
  if k2-k1-1 >= 4:
    return True

  # "left" diagonal
  i2 = i - 1
  j2 = j - 1
  while i2>=0 and j2>=0 and self.board[i2][j2]==symbol:
    i2 -= 1
    j2 -= 1
  i3 = i + 1
  j3 = j + 1
  while i3<self.m and j3<self.n and self.board[i3][j3]==symbol:
    i3 += 1
    j3 += 1
  if i3-i2-1 >= 4:
    return True

  # "right" diagonal
  i2 = i - 1
  j2 = j + 1
  while i2>=0 and j2<self.n and self.board[i2][j2]==symbol:
    i2 -= 1
    j2 += 1
  i3 = i + 1
  j3 = j - 1
  while i3<self.m and j3>=0 and self.board[i3][j3]==symbol:
    i3 += 1
    j3 -= 1
  if i3-i2-1 >= 4:
    return True

  return False

# get the row number of the topmost disk for each column
def find_top(self):
  top = [None]*self.n
  for j in range(self.n):
    i = 0
    while i<self.m and self.board[i][j] == 'O':
      i += 1
    if i < self.m:
      top[j] = i
  return top

# check if red or yellow has already won
# we assume a correct position, so the winning sequence must end at top
def check_won(self, top):
  for j in range(self.n):
    if top[j] is not None and self.have_won(top[j], j):
      return 'Z'

# return array of (i,j) of all possible next moves
def get_all_moves(self, top):
  a = []
  for j in range(self.n):
```

```
    if top[j] == 0:
      pass # full column
    elif top[j] is None:
      a.append((self.m-1, j))
    else:
      a.append((top[j]-1, j))
  return a

def move_1step(self, disk, top):
  # try all possible moves and see if it results in a win
  first_moves = self.get_all_moves(top)
  for i, j in first_moves:
    self.board[i][j] = disk # try move
    if self.have_won(i, j):
      self.board[i][j] = 'O' # undo
      return j, None
    self.board[i][j] = 'O' # undo

  # otherwise, the win is not reachable
  return 'X', first_moves

def check_1step(self, disk):
  # check if red or yellow has already won
  top = self.find_top()
  answer = self.check_won(top)
  if answer == 'Z':
    return answer

  # Red tries all possible moves and see if it results in a win in 1 step
  return self.move_1step(disk, top)

def move_2steps(self, disk):
  answer, first_red_moves = self.check_1step(disk)
  if type(answer) == int:
    return answer
  assert(answer == 'X')

  # if not then for each red move
  #   yellow tries all possible moves; if wins then this red move is not taken
  red_moves = [] # safe moves such that does not result in yellow's immediate win
  op_disk = 'Y' if disk == 'R' else 'R' # opponent's disk, yellow in our case
  for i, j in first_red_moves:
    self.board[i][j] = disk # try move
    top2 = self.find_top()
    answer, _ = self.move_1step(op_disk, top2)
    if answer == 'X':
      red_moves.append((i,j))
    self.board[i][j] = 'O' # undo

  # if remaining set of red moves is empty then yellow always wins
  if len(red_moves) == 0:
    return 'Y'

  # now we try all combinations if red could move twice
  # and find the winning combinations
  # if for any such 2 combinations the 1st move is the same but the 2nd is different
  # then yellow could not block both of them, therefore red wins anyway after 2nd step
```

```
    for i, j in red_moves:
      self.board[i][j] = disk # try move
      won = False
      top2 = self.find_top()
      moves = self.get_all_moves(top2)
      for i2, j2 in moves:
        self.board[i2][j2] = disk
        if self.have_won(i2, j2):
          if not won:
            won = True
          else: # 2nd win for different j2
            self.board[i2][j2] = 'O' # undo
            self.board[i][j] = 'O' # undo
            return j
        self.board[i2][j2] = 'O'
      self.board[i][j] = 'O' # undo

    # otherwise, the red win is not possible in 2 moves
    return 'X'

if __name__ == '__main__':
  connect4 = Connect4()
  connect4.input("input.txt")
  answer = connect4.move_2steps('R')
  if type(answer) == int:
    answer += 1
  print(f"answer = {answer}")
  connect4.output("output.txt", answer)
```