PROBLEM OF THE MONTH

POM 2019 - 2020

October, 2019

MATHEMATICS

5 points: Find all prime numbers p such that for each of them there exists a natural number n so that $\sqrt{n}+\sqrt{n+p}$ is a natural number.

Hint: Both $\sqrt{n}$ and $\sqrt{n+p}$ have to be natural numbers for their sum to be a natural number, and the only way that is possible is if n and $n+p$ are squares.

Answer: All prime numbers except 2

Solution:

Since the whole thing is a natural number, each square root must be a natural number, and thus both *n* and *n+p* are perfect squares, and *p* is the difference between two squares.
The difference between two adjacent square numbers is:
$(x+1)^2 - x^2 = x^2 + 2x + 1 - x^2 = 2x + 1$
$2x + 1$ is the standard notation for an odd number, therefore every positive odd number can be written as the difference between two adjacent squares, thus every odd prime can be written as the difference between two adjacent squares. 2 cannot be a difference of two squares because as you can see from the formula, the differences between squares only increase, and the smallest values are 1, then 3.

Note that in the problem we didn't say that n and $n+p$ had to be adjacent squares, but it is actually not possible for the difference of two non-adjacent squares to be prime. We'll leave the proof of that up to you if you're curious.

10 points:

a) 1 point: Find a solution to the equation $x^2 - xy - y^2 = 1$ where x and y are natural numbers. (Please write it as an ordered pair (x,y))

b) 7 points: Find all pairs of natural numbers (x,y) such that $x^2 - xy - y^2 = 1$.
c) 2 points: Prove that there are no other possible solutions.

Hint: Once you have found the pattern and some pair of numbers that works, you can use mathematical induction to prove that this pattern always works.

Answer:

a) (2, 1)
b) All pairs of Fibonacci numbers: (2, 1), (5, 3), (13, 8), etc.
c) See solution.

Solution 1:

1. Factor out $x - y = x'$ ($x' > 0$ because $x(x-y) = 1 + y^2 > 0$), extracting complete square, $(x^2 - xy - y^2 = 1) \Leftrightarrow ((x-y)^2 + (x-y)y - y^2 = 1) \Leftrightarrow (x'^2 + x'y - y^2 = 1)$
The pair (y, x') satisfies equation $x'^2 + x'y - y^2 = 1$.

2. Once again, factor out $y - x' = y'$ ($y' > 0$ because $(x' - y)(x' + y) = 1 - x'y < 0$, except for the special case, $x' = y = 1$, $x = 2$, considered separately below),
$(x'^2 + x'y - y^2 = 1) \Leftrightarrow (x'^2 - x'(y - x') - (y - x')^2 = 1) \Leftrightarrow (x'^2 - x'y' - y'^2 = 1)$
Hence, if a pair (x, y) of natural numbers satisfies the equation of the problem, the pair of smaller positive numbers, $(x', y') \equiv (x - y, y - (x - y))$, also satisfies the equation.

3. If the set of pairs satisfying the equation is not empty, then at least one pair (x, y) exists. From it, we obtain a sequence of pairs of natural numbers, (x_n, y_n), where numbers in each next pair are smaller than those in the previous pair. Hence, the sequence must end with a pair of smallest numbers, (x_m, y_m), $x_m > y_m$. If $x_m > y_m > 1$ and (x_m, y_m) satisfies the equation, then a smaller pair exists, which is in contradiction with (x_m, y_m) being the pair with the smallest numbers. Hence, $y_m = 1$ and the equation gives $x_m = 2$. We thus found the smallest pair, $(2, 1)$. Incidentally, this solves (a).

4. Now, re-order the pairs backwards, $(y_1, x_1), (y_2, x_2), (y_3, x_3), \ldots \equiv (1, 2), (y_2, x_2), (y_3, x_3), \ldots$. By construction, we have, $x_n = x_{n+1} - y_{n+1}$, $y_n = y_{n+1} 1 - (x_{n+1} - y_{n+1})$, wherefrom, $y_{n+1} = y_n + x_n, x_{n+1} = x_n + y_{n+1}$. We thus obtain a sequence of natural numbers where each number is the sum of the previous two (Fibonacci numbers), grouped in pairs, (y_n, x_n), starting with $(1, 2)$: $(1, 2), (3, 5), (8, 13), \ldots$.

5. The uniqueness of this solution is proven in #3: any pair that satisfies the equation generates a decreasing sequence that ends with $(1, 2)$.

Bonus. Note that from #1 and #5 it follows that pairs (y,x) of Fibonacci sequence regrouped with a shift of 1 compared to the solution found above, i.e. starting with $(1,1)$, $(1,1),(2,3),(5,8),(13,21),\ldots$, are the solutions of the equation $y^2+xy-x^2=1$.

Solution 2:

a) No explanation was necessary, just an answer.

b) After finding several solutions that work, such as (1, 0), (2, 1), (5, 3), (13, 8), etc, we have a theory that the solution is pairs of Fibonacci numbers. (As a reminder, the Fibonacci sequence starts with 0,1,... and every number in the sequence is the sum of the two previous ones.)

In order to receive full credit for part b), it was not sufficient to say “all pairs of Fibonacci numbers.” (Unless you checked every pair by hand up to infinity, you didn’t actually show that all Fibonacci pairs are solutions.) This part requires a proof.

There are probably multiple ways to do this proof, but we will use a method called mathematical induction. (https://en.wikipedia.org/wiki/Mathematical_induction) It involves proving a base case, in this case, showing that (1,0) is a solution, and then proving the induction step, which involves showing that if some pair of Fibonacci numbers are a solution, then the next two Fibonacci numbers are also a solution.

Base case:
$1^2-0*1-0^2=1$
We have shown that (1, 0) is a solution.

Induction step:
Suppose *(a, b)* are a pair of successive Fibonacci numbers where *a* > *b*, and suppose *(a, b)* is a solution to the equation. Then
$a^2-ab-b^2=1$
If the Fibonacci sequence goes *..., b, a,* then the next two numbers are *b* + *a* and *(b + a)* + *a*, or *2a* + *b*.
We need to show that if *(a, b)* is a solution, so is *(2a + b, a + b)*.

$$
\begin{aligned}
&(2a+b)^2-(2a+b)(a+b)-(a+b)^2\\
&=(4a^2+4ab+b^2)-(2a^2+3ab+b^2)-(a^2+2ab+b^2)\\
&=(4a^2-2a^2-a^2)+(4ab-3ab-2ab)+(b^2-b^2-b^2)\\
&=a^2-ab-b^2\\
&=1.
\end{aligned}
$$

Now we have shown that if *(a,b)* is a solution to the equation, so is *(2a + b, a + b)*, and the proof by induction is complete.

c)
Let us assume that we found a solution (a, b). Then it is easy to show that the pair $(a', b') = (a - b, 2b - a)$ also satisfies the equation. For this solution to be a solution with natural numbers we have to require that $b' = 2b - a > 0$. Solving the equation $a^2 - ab - b^2 = 1$ we find $b = -\frac{a}{2} + \sqrt{\frac{5}{4}a^2 - 1}$ and $2b - a = -2a + \sqrt{5a^2 - 4} > 0$ is satisfied for all $a > 2$. Therefore, for any found pair (a, b) with $a > 2$ we can construct a legitimate "reduced" solution consisting of smaller numbers $(a', b') = (a - b, 2b - a)$. We can continue this process obtaining solutions made out of smaller and smaller numbers. This procedure can stop only at the solution (a, b) with $1 \leq b < a \leq 2$. The only such solution is $(2, 1)$ which is the pair of Fibonaccis. Therefore, we showed, that any found solution (a, b) is a member of the Fibonacci pairs found in part b).

PHYSICS

5 points:

When at the restaurant chopsticks come out of the dishwasher, they are not aligned -- roughly half the chopsticks have tips pointing one way, and half the other way. Propose a method to efficiently sort them based on their alignment. Explain the physical mechanism, and perform an experimental demonstration of your method. You can use (same-length) sharpened pencils to model chopsticks.

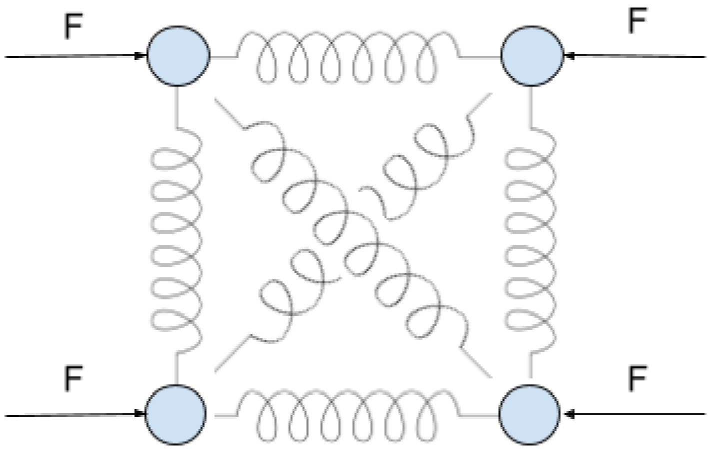
Hint: think about a single chopstick balancing on the edge of a table (perpendicular to that edge).

Answer: See below

Solution: One can line up all the chopsticks in a row perpendicular to the edge of a table, and then start pushing them all together with a ruler across the edge. The backward-facing chopsticks will tip over the edge before the forward-facing chopsticks because their center of mass is farther along.

10 points:

It is known that materials tend to expand in directions perpendicular to the direction of compression. To model this, consider a cell in which interatomic bonds are represented by springs as shown in the Figure below. The spring constants of all the springs are the same. Find the ratio of the expansion of the cell in the perpendicular (vertical) direction to its compression along the direction of the applied force (horizontal). Assume all the deformations to be small.



Hint: Consider the balance of forces acting on the “atom” in the upper left corner.

Answer: $\Delta y/\Delta x$ =1/3

Solution:

Let us denote the stiffness coefficient of the horizontal and vertical springs k_1 and of the diagonal springs k_2 . Consider the “atom” in the upper left corner. The equilibrium conditions for this atom are

$$F - k_1\Delta x - \frac{k_2\Delta L}{\sqrt{2}} = 0 \quad \text{(horizontal direction)}$$

$$\frac{k_2\Delta L}{\sqrt{2}} = k_1\Delta y \qquad \text{(vertical direction)}$$

Here Δx is the compression of the horizontal spring, Δy is the expansion of the vertical spring, and ΔL is the compression of the diagonal spring. We need only the second condition and the relation

$$\Delta L = \frac{\Delta x}{\sqrt{2}} - \frac{\Delta y}{\sqrt{2}} ,$$

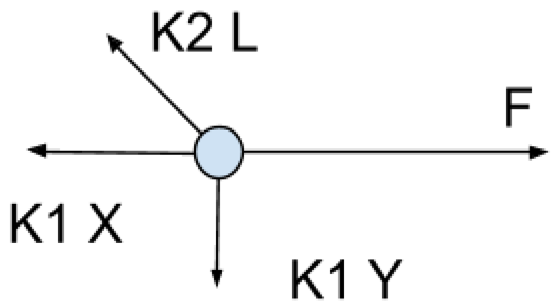
which can be obtained projecting horizontal and vertical displacements on the diagonal. We have

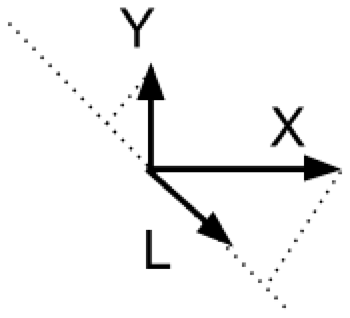
$$\frac{k_2}{\sqrt{2}}\left(\frac{\Delta x}{\sqrt{2}} - \frac{\Delta y}{\sqrt{2}}\right) = k_1\Delta y$$

And

$$\frac{\Delta y}{\Delta x} = \frac{k_2}{2k_1+k_2} = \frac{1}{3} ,$$

where we used that the stiffness of all springs is the same ($k_1 = k_2$).





CHEMISTRY

5 points:

During Sigma2019, the Mystery of chemical glassware semilab included the experiment called “extraction of caffeine from coffee”: coffee was extracted using an organic solvent called ethyl acetate, the organic phase is separated and evaporated, thereby yielding solid caffeine. Imagine that Mark forgot to bring ethyl acetate to Sigma. Would it still be possible to do that experiment if the only organic solvents available to Mark were: ethanol, hexane, acetone, dichloromethane, acetic acid, and all needed inorganic chemicals? What should Mark do in that situation?

Hint:

This type problem can be solved using two different approaches: we either prepare a solvent that we need using available chemicals, or we use another solvent with similar properties. Which approach is the best in our case? The first one? The second one? Maybe, both?

Solution:

First, Mark can prepare ethyl acetate from available chemicals. Usually, ethyl acetate is prepared from ethanol and acetic acid. A catalyst in this reaction is some strong acid (concentrated sulfuric acid works fine). Since Mark has all needed inorganic chemicals, sulfuric acid is definitely available. Mark has to *reflux* a mixture of acetic acid, ethanol and sulfuric acid for about an hour (“reflux” means to boil a reaction mixture in a flak with a condenser attached to its neck, so all condensed vapours drop back into the flask). After that, a large amount of water is added to the reaction mixture, and ethyl acetate, which is not mixable with water, forms a top layer, which is easy to separate.

Second, it is possible to select a solvent that can be used instead of ethyl acetate. Such a solvent has to meet two criteria: it should dissolve caffeine, and it should be not mixable with water. Among the solvent from the above list, dichloromethane meets both these criteria.

Both solutions will work.

10 points:

Imagine you have to prepare as much hydrogen as possible, and you can spend $40 for chemicals (you can buy whatever you want from Amazon; shipping cost is not included). What is the maximal volume of hydrogen that you can prepare? Shipping cost is not included.

Hint:

Obviously, one of the simplest ways is to use some metal that produces hydrogen in a reaction with some other chemical. The difference between this problem and similar problems of that kind is that you have to calculate the amount of hydrogen per $40, not the amount of hydrogen per one mole (or gram, etc) of chemicals.

Solution:

Two packs of Reynolds aluminium foil (approximately 300 g) cost $9. One atom of aluminium produces 1.5 molecules of H_2 (aluminium is trivalent), so aluminium seems to be an optimal solution it terms of the volume of hydrogen per gram of metal, and per $1. To obtain hydrogen, we need some auxiliary reagent. Two reactions are the most promising candidates:

$2\ Al + 6\ HCl \rightarrow 2AlCl_3 + 3H_2$
$2\ Al + 6\ NaOH \rightarrow 2Al(OH)_3 + 3H_2$

The first reaction requires HCl, which costs $32 per gallon. Its concentration is ca 35%, so it is approximately 10 M. That means one gallon contains about 36 moles of HCl. 300 g of aluminium foil is 300/25= 12 moles of aluminium, which is exactly what is needed for a reaction with 36 moles of HCl (according to the equation, 3 moles of HCl are needed to reach with 1 mol of aluminium). 36 moles of HCl produce 18 moles of H_2. According to the Avogadro's law, one mole of any gas occupies 22.4 L at room temperature and atmospheric pressure, so the total volume will be 336 L.
The second reaction requires NaOH, which costs $30 per 5 pounds. To react with 12 moles of aluminium, 36 moles of NaOH are needed, which is 36×40=1440 grams, or 3.2 lb. The amount of hydrogen will be the same, but about 1.8 lb of NaOH will remain for your future experiments, so the second method is preferable.

BIOLOGY

5 points:

Mark has a dog, her name is Marta (on the right photo). She, like all laikas, is a passionate hunter. When she and Mark are walking in the woods, she loves to hunt chipmunks, and the hunt usually happens like this: Marta walks along the path; when the chipmunk sees her, it squeaks loudly and hides in the hole. Marta runs up to the hole, quickly digs it up (on the left photo), and, if Mark does not pull her away, she catches the chipmunk. It is clear that had the chipmunk not squeaked, Marta probably would not have noticed it. Obviously, chipmunk's squeaking is some instinct. Why is it needed?

Hint:

All chipmunks behave similarly, which means that is some inherited instinctive behaviour. That means that trait is beneficial for survival, although in this situation it leads to chipmunk's death in almost 100% cases (of course, we mean a hypothetical situation when Mark doesn't stop Marta). That seemingly obvious contradiction with the Darwin's "survival of the fittest" principle can be easily resolved if we remember that one word is missing. What exactly did Darwin meant under "survival of the fittest"?

Answer:

Darwin's core idea is "survival of the fittest *species"*. When a chipmunk is screaming, that most likely makes its own situation worse. However, by doing that, it warns all its neighbours, some of whom are its close relatives or even offspring. In other words, this self-sacrifice becomes beneficial for the population in general, so the chances of survival of the species as whole increase. That demonstrates that altruism can have a quite clear evolutionary origin, thereby debunking the claim made by Darwin's critics that his theory advocates egoism and individualism.

10 points:

In the science fiction novel "The Gods Themselves" by Isaac Asimov, the author describes some alien race with three sexes ("rational", "emotional", and "parental"). He explains the mechanism of their reproduction (all three sexes participate in reproduction, and always give birth to one "rational", one "emotional", and one "parental" child). Asimov tells nothing about the genomic organization of that alien species, but let's imagine that it is based on some nucleic acid, which is similar to our DNA, and inheritance rules are similar. Please explain how can chromosomes of such species be organized to produce three different sexes, as a result of sexual reproduction.

Hint:

Of course, the species described by Azimov are imaginary, so we can just speculate about their genome organization. However, we should keep in mind that the genome organization should look reasonable, and each of three sexes is supposed to have a chromosome set that will reliably lead to no loss of genetic information, and the ratio between the three sexes in the next generation must be exactly 1:1:1. Theoretically, a diploid genome can provide that. How exactly?

Answer:

The most obvious answer seems to be that this species has a diploid set of chromosomes, and the sex chromosomes are X and Y, but a distribution of chromosomes among different sexes is XX, YY, and XY. Formally that is correct, but it is not clear which mechanism of sexual reproduction can provide such sets. If each sex provides just one chromosome set (their sexual cells have a gaploid chromosome set), then six gametas are needed, which makes a sexual reproduction to complex and unreliable.
One possibility is that the reproduction is somewhat different, for example, the egg cell has a double set of chromosomes (e.g. two diploid sets, and, accordingly, a duplicated set of sex chromosomes, for example XY XY), and two sperm cells (from a second and a third parent) bear a gaploid set. When two gametes come, each with a haploid set of chromosomes, X and Y, during fertilization they mix, and the combined cell has a hexaploid chromosome set (three X and three Y chromosomes). Then the cell divides onto three parts, each of which has a double (diploid) set of chromosomes (XX, YY, and XY), and then each new part becomes a separate embryo. The most probably, parental is the sex that has an XY chromosome set.
In summary, the most plausible reproduction scheme should be as follows:

1. Parental has an XY set, and it produces an egg with a double diploid set (XY XY).
2. Rational and emotional have YY and XX sets, and they produce gametes with Y and X chromosomes, accordingly.
3. After fertilization, the "pre-embryo" cell has a hexadiploid set of chromosomes (XY XY from parental Y from rational and X from emotional), and it divides onto three parts with XX, YY and XY chromosomes, and each of them yields a separate embryo.

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!**

Credit card companies tend to collect all purchases made by a particular client in a given month and mail a report to the client's home in a batch known as the "monthly statement". The clients then pay for their purchases with cash from their bank accounts.
As an incentive for customers to use their credit cards, there is also a reward mechanism: clients earn "points" on all/most purchases they make using their credit cards, typically in proportion to the dollar amount of the purchases. The points can then be redeemed and used to "cover"/pay for some of the purchases in lieu of using cash. We will assume that i) all available points are to be redeemed at once for a combination of charges and that ii) no partial covering of charges is allowed: a particular charge is either fully covered or not covered at all with the points. Any number of points that are left without covering charges in full will disappear and are effectively wasted.

5 points:

Write a program that takes as input a list of positive integer valued (not necessarily sorted in any manner) charges and finds out if there exists two pairs of charges that have the same total amount. If there exists a list of 4 separate charges a, b, c, and d, such that a+b=c+d, your program should print "YES" and list the charges. Otherwise the program should print "NO". Assume that charges are all positive integers. If the input contains less than 4 elements or if the input cannot be interpreted as integers, the program should indicate so.

Hint:

It’s convenient to use a map of all sums and a tuple of their constituents. Alternatively, you can use an array and sort it.

Solution:

Python:

```
"""
We can always try to do the brute force approach. Let n be the length of input array.
1) Let "a" be a number from 1st in the list till last-2            - approximately n choices
2) Let "b" be a number from "a" till last                          - approximately n/2 choices
3) Let "c" be a number from "a" till last-1                        - approximately n/2 choices
4) Let "d" be a number from "c" till last                          - approximately n/2 choices
Also make sure you pick different numbers for a, b, c and d. I may be slightly off with
indexing but you get the idea.
So, in total, we'll get O(n^4) operations (the division by 2 doesn't count as we're interested
in
how the complexity grows with n, instead of exactly counting number of operations).
We can try to improve this by sorting the array and aborting a loop when we get c+d that is
already > a+b.
Now let's try to improve this even further. Let's represent the numbers as a table.
    x1 x2 x3 x4 ... xn
x1  0  +  +  +      +
x2  0  0  +  +      +
x3  0  0  0  +      +
...
xn  0  0  0         0
'+' represents a sum of corresponding numbers. We can ignore the main diagonal (a,b,c,d should
refer
to different numbers) and below (the matrix is symmetrical as a+b=b+a). We'll need n*(n/2)
operations
for this. Then all we need is to find a pair of '+' that are the same but located on different
rows
and columns (so we don't count the same numbers). The easiest way to do this is to sort them
and
scan sequentially until we find 2 same sums.
"""
import numpy as np

numbers_str = input("Enter numbers: ").strip().split()
# the following will croak if the elements are not all integers
numbers = [int(x) for x in numbers_str]
# also verify that they all > 0
assert(all(x > 0 for x in numbers))
# and at least 4 numbers
n = len(numbers)
assert(n >= 4)

# construct the table
table = np.zeros([n,n], dtype="int")
for j in range(1,n):
  for i in range(0,j):
    table[i,j] = numbers[i] + numbers[j]

# collect the sums with their indices
```

```
a = []
for j in range(1,n):
  for i in range(0,j):
    a.append((table[i,j], i, j))

# sort by the field "sum"
a.sort(key = lambda x: x[0])

for i in range(1,len(a)):
  # check if "sums" are the same but indices different
   if a[i-1][0] == a[i][0] and a[i-1][1] != a[i][1] and a[i-1][2] != a[i][2] and a[i-1][1] !=
a[i][2] and a[i-1][2] != a[i][1]:
         print("YES", numbers[a[i-1][1]], numbers[a[i-1][2]], " and ", numbers[a[i][1]],
numbers[a[i][2]])
    exit(0) # stop after the 1st one is found
print("NO")
exit(1)
```

Java:

```
/*
We can always try to do the brute force approach. Let n be the length of input array.
1) Let "a" be a number from 1st in the list till last-2             - approximately n choices
2) Let "b" be a number from "a" till last                           - approximately n/2 choices
3) Let "c" be a number from "a" till last-1                         - approximately n/2 choices
4) Let "d" be a number from "c" till last                           - approximately n/2 choices
Also make sure you pick different numbers for a, b, c and d. I may be slightly off with
indexing but you get the idea.
So, in total, we'll get O(n^4) operations (the division by 2 doesn't count as we're interested
in
how the complexity grows with n, instead of exactly counting number of operations).
We can try to improve this by sorting the array and aborting a loop when we get c+d that is
already > a+b.
Now let's try to improve this even further. Let's represent the numbers as a table.
    x1 x2 x3 x4 ... xn
x1  0  +  +  +      +
x2  0  0  +  +      +
x3  0  0  0  +      +
...
xn  0  0  0         0
'+' represents a sum of corresponding numbers. We can ignore the main diagonal (a,b,c,d should
refer
to different numbers) and below (the matrix is symmetrical as a+b=b+a). We'll need n*(n/2)
operations
for this. Then all we need is to find a pair of '+' that are the same but located on different
rows
and columns (so we don't count the same numbers). The easiest way to do this is to sort them
and
scan sequentially until we find 2 same sums.
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;

public class AmEx5 {
  int[] numbers; // = {1, 2};
  int n;
  int[][] table;

  void input() throws IOException {
    System.out.print("Enter numbers: ");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    String line = reader.readLine();
    String[] nums = line.trim().split("\\s*[\\s,]\\s*");
    // the following will croak if the elements are not all integers
    numbers = Arrays.stream(nums).map(s ->
Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
    // also verify that they all > 0
    if(!Arrays.stream(numbers).allMatch(i -> i > 0))
      throw new AssertionError("all numbers must be positive");
    n = numbers.length;
    if(n < 4)
      throw new AssertionError("must have at least 4 numbers");
  }

  void solve() {
    // construct the table
    table = new int[n][n];
    for(int j = 1; j < n; j++) {
      for(int i = 0; i < j; i++) {
        table[i][j] = numbers[i] + numbers[j];
      }
    }

    // collect the sums with their indices
    ArrayList<Integer[]> a = new ArrayList<>();
    for(int j = 1; j < n; j++) {
      for(int i = 0; i < j; i++) {
        a.add(new Integer[]{table[i][j], i, j});
      }
    }

    // sort by the field "sum‟
    a.sort((x, y) -> x[0] - y[0]);

    for(int i = 1; i < a.size(); i++) {
      // check if "sums" are the same but indices different
      if(a.get(i - 1)[0] == a.get(i)[0] && a.get(i - 1)[1] != a.get(i)[1] && a.get(i - 1)[2] !=
a.get(i)[2] &&
          a.get(i - 1)[1] != a.get(i)[2] && a.get(i - 1)[2] != a.get(i)[1]) {
         System.out.printf("YES %d, %d and %d, %d\n", numbers[a.get(i - 1)[1]], numbers[a.get(i
- 1)[2]], numbers[a.get(i)[1]], numbers[a.get(i)[2]]);
        return; // stop after the 1st one is found
      }
    }
    System.out.println("NO");
  }

  public static void main(String[] args) throws IOException {
    AmEx5 amex = new AmEx5();
    amex.input();
    amex.solve();
  }
}
```

10 points:

Write a program that takes as input a list of positive integer valued charges and a positive integer point value earned, and outputs the combination of charges that wastes the least number of points when redeemed and the actual point value wasted.
For example, input of charges [3, 4, 1, 9, 100, 4, 1, 6, 1] and point value of 111 would result in the output of charges [1, 1, 9, 100] as the best combination with the wasted point value equal to 111-(1+1+9+100)=0. Input of charges [3, 4, 1, 9, 100] and point value of 106 would result in the output of charges [1, 4, 100] with 106-(1+4+100)=1 point wasted.
In case the inputs are non-numerical/non-integer/non-positive, or if not a single charge can be covered by the point value provided, then the program should indicate so.
If more than one combination of charges results in the same minimum amount of points wasted (there is a tie), it suffices to output only one of the combinations along with the points wasted.

Hint:

Use principles of dynamic programming: consider increasing in size sets of charges and remember accumulated totals in a table as you go.

Solution:

Python:

```
"""
This is the classical knapsack 0/1 problem (https://en.wikipedia.org/wiki/Knapsack_problem).
You can find many videos on youtube that explain the algorithm.
I only want to point out the difference between the polynomial runtime and pseudo-polynomial
(this problem). In the polynomial format the runtime depends only on the number of elements,
but in pseudo-polynomial problems it also depends on the values of the elements themselves.
For example, for a practical knapsack for 10 elements you may have 20 rows in the memoization
table
(or however many kilograms you can lift) but for the same credit card charges, if they are
of the order of $100 (= 10,000 cents) you'll get 10,000 rows!
Note: in our problem the value = the weight.
"""
import re
import numpy as np

numbers_str = re.split(r"[\s,]\s*", input("Enter charges in cents: ").strip())
# the following will croak if the elements are not all integers
numbers = [int(x) for x in numbers_str]
# also verify that they all > 0
```

```
assert(all(x > 0 for x in numbers))

points_str = input("Enter number of points: ").strip()
# the following will croak if the elements are not all integers
W = int(points_str) # knapsack capacity
# also verify that they all > 0
assert(W > 0)
# verify that the points can cover at least 1 charge
assert(W >= min(numbers))

n = len(numbers) # number of charges

v = [0] + numbers # values with starting index = 1
w = v # weights

dp = np.empty([n+1, W+1], dtype="int") # our DP memoization table
dp.fill(-1)

# define function m so that it represents the maximum value we can get under the condition
# use first i items, total weight limit is j
def m(i, j):
  if i == 0 or j <= 0:
    return 0

  if dp[i-1,j] == -1: # m(i-1, j) has not been calculated, we have to call function m()
    dp[i-1,j] = m(i-1, j)

  if w[i] > j: # item cannot fit in the bag
    dp[i,j] = dp[i-1,j]
  else:
      if  dp[i-1,  j-w[i]]  ==  -1:  #  m(i-1,  j-w[i])  has  not  been  calculated,  we  have  to  call
function m()
      dp[i-1, j-w[i]] = m(i-1, j-w[i])
    dp[i,j] = max(dp[i-1,j], dp[i-1, j-w[i]] + v[i])

  return dp[i,j]

res = m(n, W)
print("wasted points = %d" % (W-res))

# trace back the table to find charges
print("charges: ", end='')
j = W
for i in range(n, 0, -1):
  if res <= 0:
    break
  # either the result comes from the top (dp[i-1,j]) or from (v[i-1] + dp[i-1, j-w[i]])
  # as in memoization table. If it comes from the latter one it means the item is included.
  if res == dp[i-1,j]:
    continue
  else:
    # this item is included
    print(w[i], ", ", end='')
    # since this weight is included its value is deducted
    res = res - v[i]
    j = j - w[i]
print()
```

```
exit(0)
```

Java:

```
/*
This is the classical knapsack 0/1 problem (https://en.wikipedia.org/wiki/Knapsack_problem).
You can find many videos on youtube that explain the algorithm.
I only want to point out the difference between the polynomial runtime and pseudo-polynomial
(this problem). In the polynomial format the runtime depends only on the number of elements,
but in pseudo-polynomial problems it also depends on the values of the elements themselves.
For example, for a practical knapsack for 10 elements you may have 20 rows in the memoization
table
(or however many kilograms you can lift) but for the same credit card charges, if they are
of the order of $100 (= 10,000 cents) you'll get 10,000 rows!
Note: in our problem the value = the weight.
*/

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

public class AmEx10 {
  int[] numbers; // charges
  int n;         // number of charges
  int W;         // knapsack capacity
  int[] v;       // values
  int[] w;       // weights
  int[][] dp;    // our DP memoization table

  void input() throws IOException {
    System.out.print("Enter charges: ");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    String line = reader.readLine();
    String[] nums = line.trim().split("\\s*[\\s,]\\s*");
    // the following will croak if the elements are not all integers
    numbers = Arrays.stream(nums).map(s ->
       Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
    // also verify that they all > O
    if(!Arrays.stream(numbers).allMatch(i -> i > 0))
      throw new AssertionError("all numbers must be positive");
    n = numbers.length;
    if(n <= 0)
      throw new AssertionError("provide at least 1 charge");

    System.out.print("Enter number of points: ");
    line = reader.readLine();
    // the following will croak if the elements are not all integers
    W = Integer.valueOf(line.trim());
    // also verify that they all > 0
    if(W <= 0)
      throw new AssertionError("all numbers must be positive");
    // verify that the points can cover at least 1 charge
    if(W < Arrays.stream(numbers).min().getAsInt())
      throw new AssertionError("the points must cover at least 1 charge");

    v = new int[n + 1]; // values with starting index = 1
    v[0] = 0;
    System.arraycopy(numbers, 0, v, 1, n);
    w = v;
    dp = new int[n + 1][W + 1];
    Arrays.stream(dp).forEach(row -> Arrays.fill(row, -1));
  }

  // Define function m so that it represents the maximum value we can get under the condition:
  // use first i items, total weight limit is j.
  int m(int i, int j) {
    if(i == 0 || j <= 0)
      return 0;

    if(dp[i-1][j] == -1) // m(i—1, j) has not been calculated, we have to call function m()
      dp[i-1][j] = m(i-1, j);

    if(w[i] > j) // item cannot fit in the bag
      dp[i][j] = dp[i-1][j];
    else {
      if(dp[i-1][j-w[i]] == -1) // m(i-1, j-w[i]) has not been calculated, we have to call
function m()
        dp[i-1][j-w[i]] = m(i-1, j-w[i]);
      dp[i][j] = Math.max(dp[i-1][j], dp[i-1][j-w[i]]+v[i]);
    }
    return dp[i][j];
  }

  void solve() {
    int res = m(n, W);
    System.out.printf("wasted points = %d\n", W - res);

    // trace back the table to find the charges
    System.out.print("charges: ");
    int j = W;
    for(int i = n; n > 0; i--) {
      if(res <= 0)
        break;
      // Either the result comes from the top (dp[i-1][j]) or from (v[i-1] + dp[i-1][j-w[i]])
      // as in memoization table. If it comes from the latter one it means the item is
included.
      if(res == dp[i - 1][j])
        continue;
      else {
        // this item is included
        System.out.printf("%d, ", w[i]);
        // since this weight is included its value is deducted
        res -= v[i];
        j -= w[i];
      }
    }
    System.out.println();
  }

  public static void main(String[] args) throws IOException {
    AmEx10 amex = new AmEx10();
    amex.input();
    amex.solve();
  }
```

```
}
```

LINGUISTICS

5 points:

Consider the following words from a North-East Caucasian language. Apart from translations, you are given the sample sentences where the particular form would be used.

	Gloss	**Sample sentence**
q'atluvu	in the house	I'm in the house
q'atluxux	behind the house	I pass behind the house
q'atluvatu	from inside the house	I go from inside the house
q'atlulu	under the house	I am under the house
q'atluj	on top of the house	I am on top of the house
q'atluvun	into the house	I enter into the house
q'atluxatu	from behind the house	I come from behind the house
q'atlulun	under the house	I go under the house
q'atlujx	over the house	I walk over the house (over the roof)

Translate the following phrases (as used in the corresponding sentences) into this language and explain your reasoning.

1.	from under the house	I exit from under the house
2.	through the house	I pass through the house
3.	on top of the house	I enter on top of the house (its roof)

Solution:

First, the part of the words corresponding to "house" is *q'atlu*, and from now on we are only going to consider the endings. We can create a table using the meanings given in the problem; the cells of the table for forms we need to find in the problem are marked with a question mark.

	(1)	**(2)**	**(3)**	**(4)**
	location	**moving to**	**moving from**	**passing**
Inside	-vu	-vun	-vatu	**? - 2.**
under	-lu	-lun	**? - 1.**	
on	-j	**? - 3.**		-jx
behind			-xatu	-xux

Examining the table allows us to understand the pattern: the ending from column (1) is modified to get an ending from other columns. Column (2) is obtained by adding *-n* to the ending in column (1). Column (4) is obtained by addition of *-x*. It is not completely obvious how to obtain endings in column (3), so it is possible to have at least two alternative hypotheses. The first possibility is to just use the consonant from the first column and add *-atu*. The second possibility is to insert *-at* inside the ending in column (1). Luckily, for the purposes of the problem we don't need to decide which of these hypotheses is correct and how to deal with the ending *-j*.

The answer is therefore the following:

1. *from under the house* (as in *I exit from under the house*): *q'atlu-latu*
2. *through the house* (as in *I pass through the house*): *q'atlu-vux*
3. *on top of the house* (as in *I enter on top of the house*): *q'atlu-jn*

10 points:

Consider the following sentences from a North-East Caucasian language and their translations:

1.	dis hibat:ur došdur di	I have a good sister
2.	wel x:allu ušdu wak:urši wi	We see a bad brother
3.	bolo x:allub xoʕn bi	We have a bad cow
4.	ez jamut q'onq' erxinši i	I forget this book
5.	bez jamub c'ai bokorši bi	I hear this goat
6.	el hibat:ut aqx' ak:urši i	We see good meat

Translate into this language:

1. We hear this sister.
2. We have a bad book.
3. I forget a good brother.
4. I have a good goat.

Solution:

Establishing correspondence between the words from the problem and their translations leads to the following structure:

1). 1st place: pronoun (*dis*, *ez*, *bez* 'I', *bolo*, *wel*, *el* 'we')
2). 2nd place: adjective or demonstrative pronoun (*hibat:ur*, *hibat:ut* 'good', *x:allu*, *x:allub* 'bad', *jamut*, *jamub* 'this')

3). 3rd place: noun
4). 4th place: verb (*wak:urši*, *ak:urši* ‘see’, *erxinši* ‘forget’, *bokorši* ‘hear’)
5). Last place: auxiliary verb ‘to be’ (*di*, *wi*, *bi*, *i*)

All parts of speech but nouns change and take prefixes (*d-*, *w-*, *b-*) and suffixes (*-r*, *-b*, *-t*).

The suffixes and prefixes depend on the noun:

- sister: *d-; -r*
- brother: *w-; -∅*
- cow, goat: *b-; -b*
- book, meat: *∅-; -t*

The answer to the question is therefore the following:

1. We hear this sister: *del jamur došdur dokorši di.*
2. We have a bad book: *olo x:allut q’onq’ i.*
3. I forget a good brother: *wez hibat:u ušdu werxinši wi.*
4. I have a good goat: *bis hibat:ub c’ai bi.*