PROBLEM OF THE MONTH POM 2019 - 2020 **November, 2019**

MATHEMATICS

5 points:

After a math class on the blackboard there remained the graph of $y = \frac{k}{x}$ and five lines, parallel to $y = 2kx \ (k \neq 0)$. Find the product of the *x*-coordinates of all 10 intersection points.

Answer: $-\frac{1}{32}$

Solution:

A line parallel to $y = 2kx \ (k \neq 0)$ is given by $y = 2kx + b$. The x-coordinate of its intersection point with the graph $y = \frac{k}{x}$ can be found from $\frac{k}{x} = 2kx + b$ or $2kx^2 + bx - k = 0$. There are two roots (two intersection points) of this equation. The product of those roots is given by Vieta's theorem and is equal to $-\frac{k}{2k} = -\frac{1}{2}$. Notice, that this product does not depend on b and is, therefore, the same for all 5 parallel lines. Therefore, for the product of all 10 x-coordinates we get $(-\frac{1}{2})^5 = -\frac{1}{32}$.

10 points:

Every bus has passengers, and not always the same amount. Every passenger has co-passengers. Two passengers are co-passengers if they are riding the same bus (every passenger is also his own co-passenger). At any given time, what is greater, the average number of passengers per bus or the average number of co-passengers per passenger?

You can start by considering 3 buses, with 3, 5, and 7 passengers. Then, consider the general case of *n* buses with $a_1, a_2, \ldots, a_n$ passengers.

Answer: the average number of passengers per bus is SMALLER than the average number of co-passengers per passenger.

Solution:

The average number of passengers per bus is given by $P = \frac{1}{n}(a_1 + a_2 + ... + a_n)$. The average number of co-passengers per passenger is given by $C = \frac{a_1^2+a_2^2+...+a_n^2}{a_1+a_2+...+a_n}$. We have to compare P and C. Let us start with an obvious inequality $2a_ia_j \le a_i^2 + a_j^2$ following from $(a_i - a_j)^2 \ge 0$. We sum both sides of this equation over i and j from 1 to n. We obtain

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 2a_ia_j \le \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^2 + a_j^2) = 2n(a_1^2 + a_2^2 + ... + a_n^2)$$

We can transform the left hand side of this equation as

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 2a_ia_j = 2(a_1^2 + a_2^2 + ... + a_n^2) + 2\sum_{i<j}^{n} 2a_ia_j = 2\,(a_1 + a_2 + ... + a_n)^2$$

Substituting this identity into the previous inequality we obtain

$$(a_1 + a_2 + ... + a_n)^2 \le n(a_1^2 + a_2^2 + ... + a_n^2)$$

The latter inequality is equivalent to $P \le C$. It is obvious than the equality can be achieved only if all a_i are the same which is not possible according to the statement of the problem. We conclude that $P < C$.

Remark: This problem explains Sod's law, or 'zakon podlosti' in Russian. We usually ride in buses that are more full than the average bus and wait in lines that are longer than the average line.

Remark: This problem is taken from Moscow math olympiad 2010.

PHYSICS

5 points:

Tokyo Skytree is the second tallest structure in the world. Shuttle elevators servicing its Tembo Observation Deck have a maximum speed of 10m/s, making them the fastest large-capacity elevators in Japan. It takes only 50 seconds for an elevator to travel 350m to the Observation Deck. Find the time during which the elevator moves with its maximum speed, assuming that it first reaches this speed by moving up with a constant acceleration, and that at the end of the trip it slows down with a constant deceleration.

Hint: The average velocity of the elevator during acceleration and deceleration is 5 m/s.

Answer: $T = 20$ s

Solution:

Let us denote the time of the motion with maximum speed T. Then the time of accelerated motion (including both acceleration and deceleration) is $(50 - T)$ (in seconds). The displacement of the elevator during that motion is $(10/2)(50 - T)$ (in meters) as the average velocity during the motion with constant acceleration is $10/2 = 5$ m/s. We have

$350 = 10T + 5 \times (50 - T)$,

$350 = 5T + 250$,

$T = 20$ s.

Notice, that if you assume that the time of acceleration is equal to the time of deceleration you get the same result but this assumption is not needed for the solution and is not stated in the problem.

10 points:

The UFO is moving in deep space with velocity V_0. The pilot of UFO would like to change the velocity of UFO so that it is the same in absolute value but turned by an angle of 90^0. The acceleration of the UFO cannot exceed a_0 in its absolute value. What is the shortest time needed for such a turn? Prove that this time is indeed the minimal. How far from its initial position the UFO will be after the completion of the maneuver?

Hint: Think of the velocity difference as of a vector. What should be the direction of the acceleration during the optimal maneuver?

Answer: $t = \sqrt{2}\, V_0/a_0$; $S = V_0^2/a_0$

Solution:

The velocity difference is the vector of the absolute value $\sqrt{2}\,V_0$. It is clear that the optical maneuver is to accelerate with the maximal acceleration directed along this difference. The minimal time required for that is $t = \sqrt{2}\,V_0/a_0$. (In velocity space any other “path” is longer than the straight line).

Let us now consider the motion corresponding to the optimal maneuver. One can decompose this motion onto the direction of acceleration (let us denote it x) and the direction orthogonal to the acceleration (y-direction). The initial x-velocity is $-V_0/\sqrt{2}$ and the final is $V_0/\sqrt{2}$. The displacement in the x direction is $S_x = (-V_0/\sqrt{2})t + a_0 t^2/2 = -V_0{}^2/a_0 + V_0{}^2/a_0 = 0$ (this is pretty obvious as the velocity changed sign during this motion). In the y direction the velocity is constant and equal $V_0/\sqrt{2}$. Therefore, the total displacement of the UFO is in y direction and is equal to $S = S_y = t\,V_0/\sqrt{2} = V_0{}^2/a_0$.

Remark: the motion of the UFO for the described maneuver is similar to the motion of the projectile thrown at 45 degrees to the horizon.

CHEMISTRY

5 points:

Experiments with oxygen are very spectacular, but its preparation require either expensive equipment or dangerous chemicals. Nevertheless, it is quite possible to prepare oxygen using only the materials that you can freely buy at Stop&Shop. Imagine you have $50. What should you buy at Stop&Shop (or another supermarket if you are living not in the US) for that money to prepare as much oxygen as possible?

Hint:

To obtain one of those chemicals, you should break apart some stuff that you can buy at every convenience store. Another reactant can be found in the medicine section.

Solution:

3% hydrogen peroxide is usually sold for $1.4 per 16 oz (roughly 0.5 L). Hydrogen peroxide decomposes spontaneously according to the equation:

$2\ H_2O_2 \rightarrow 2H_2O + O_2$

so two moles of hydrogen peroxide (i.e. 64 grams) form 22.4 L of oxygen. ~35 hydrogen peroxide bottles (18 L) can be bought for $50, which corresponds to 530 grams of H_2O_2, or 16 moles of it. 16 moles of H2O2 produce 8 moles of oxygen, or,roughly 180 liters of it.

The problem is that decomposition of hydrogen peroxide is very slow, and we need a catalyst for that. This process is catalyzed by, for example, manganese dioxide, which can be found in alkaline batteries. You need just one battery for that. Open it and drop a black stuff to hydrogen peroxide. A reaction will start immediately. Since MnO_2 is a catalyst, it is not consumed in the reaction. Another catalyst is sodium iodide, which is present in an iodized salt.

As you noticed, all these calculations are approximate, because hydrogen peroxide is unstable, so its actual concentration varies from bottle to bottle. Therefore, in this case, precise calculations are senseless.

10 points:

Imagine that by the 5th day of Sigma, the only remaining chemicals in Mark's lab were:

1. Sulfuric acid
2. Hydrogen peroxide
3. Copper sulfate
4. Manganese sulfate
5. Luminol
6. Potassium ferrocyanide
7. Ammonium thiocyanate
8. Malonic acid
9. Sodium hydroxide
10. Potassium iodate

11. Ferric chloride

Which spectacular chemical reactions our counselors would do using the chemicals from this list? What additional demonstrations will they be able to do if they come to the camp's kitchen and take some stuff there?

Hint:

Usually, one can find starch at any kitchen.
The name "luminol" has the same root as "lumos".

Solution:

1. Using luminol, hydrogen peroxide, copper sulfate (or potassium ferrocyanide, or ferric chloride) and sodium hydroxide, counselors can do a "glow-in-the-dark" reaction.
2. Using ferric chloride and ammonium thiocyanate, counselors can do a fake blood cut trick.

   And, finally, the most spectacular experiment, which also requires an additional chemical, starch, which can be found at any kitchen (or prepared from ordinary flour), is
3. Briggs-Raucher reaction, which requires manganese sulfate, hydrogen peroxide, sulfuric acid, malonic acid, potassium iodate, and starch, which is not in the list, but which can be obtained in a kitchen.

All these reactions are googleble, so no additional description is required.

BIOLOGY

A sensory deprivation chamber is an environment, usually a large tub of saline water carefully calibrated to be at body temperature, and in a sealed room, in which all sensory inputs (ambient light, sound, tactile stimuli) are eliminated. In this environment, individuals report having no sensation of time and are often quite off in terms of their estimation of how much time has passed (they think hours have passed when they have been there only minutes, and vice versa).

5 points:

1. If they were in this environment long enough (e.g., weeks or months), how would their sleep-wake cycles be affected, and why?

2. Would slightly increasing or decreasing the ambient water temperature affect this process, and if so, how and why?

Answer:

Losing light inputs would cause circadian rhythms to default to non-entrained cycles, which are a little bit longer than 24 hours (24 hours and 11 minutes).

Increasing temperature would likewise increase metabolism, which normally would speed up all biological processes, including clocks. However the circadian period is designed to be temperature invariant, in order to compensate for these effects. This effect is called "temperature compensation," which is quite remarkable because temperature itself can entrain biological oscillators. For a potential resolution of this apparent paradox, see: *https://www.pnas.org/content/112/46/E6284*

10 points:

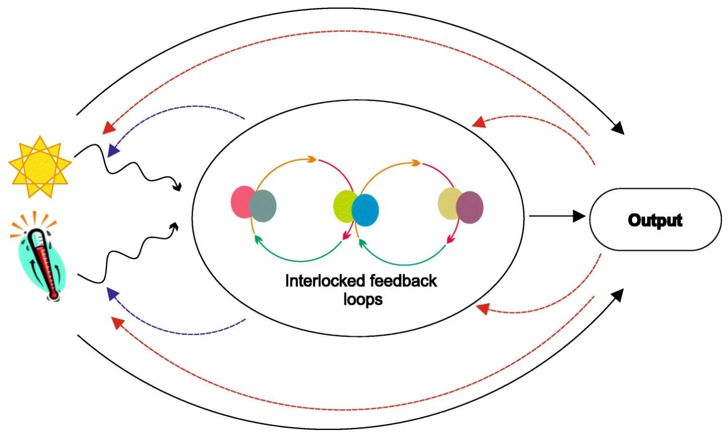
1. Besides measuring sleep-wake cycles, identify two other biological processes that one could monitor to determine the individual's endogenous circadian rhythm? Describe an experimental setup by which you could monitor them without disrupting the experiment.

2. In addition to the suprachiasmatic nuclei (SCN), which form the "master clock," other peripheral oscillators exist in other cells. Is it possible for different parts of the body to be oscillating at different frequencies? Why or why not?

Answer:

Some other diurnal processes that one could measure are: (1) fluctuations in core body temperature, (2) fluctuations in cortisol, measured by blood, saliva, hair, or urine (for cortisol, only blood would show a response with sufficient time-resolution to be useful for the above application). Heat sensors measuring changes in ambient water temperature, heated and cooled by occupant? Or sensors in contact lens https://advances.sciencemag.org/content/4/1/eaap9841? Or sensors in waterproof dermal patch https://www.medgadget.com/2018/07/wearable-patch-can-sense-cortisol-levels-in-sweat.html?

Biological oscillators are produced by nested control circuits (embedded negative feedback loops, see below). Conceptually, we can liken this relationship between master and slave clocks to a series of "gears" of different sizes, in which a central driving gear can cause other gears to cycle at other frequencies.



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!**

Intro:

Every year, after students are admitted to SigmaCamp, they get a list of semilabs to choose from, and specify, in order of preference, their top 4 choices. SigmaStaff then have to match students so that everyone gets semilabs they want.

For both problems, your program will read and process 2 input text files:
- The first file (named semilabs.txt) is a list of Sigma semilabs with the corresponding maximum number of students; the format is a table with N rows where each row consists of 2 integers: semilab_number, max_students. Here semilab_number is assumed to be non-repeating integers from 1 to N, one for each semilab. But max_students can be arbitrary positive integers. You can assume that N will not exceed 10.
Sample semilabs.txt file can look like the following:

```
1  4
2  4
3  4
4  3
5  3
6  5
```

- The second file (named students.txt) is a table of M rows (where M > N) where each row has 5 integers: student_id, choice_1, choice_2, choice_3, choice_4, where 1 <= choice_x <= N with each of the 4 choices distinct. You can assume that M will not exceed 15.

Sample students.txt file can look like the following:

```
111 3 2 4 1
3 1 3 4 2
4 2 1 6 3
223 6 1 2 3
211 3 4 1 6
98 6 3 1 2
44 1 2 4 3
52 6 1 4 2
67 2 3 4 1
21 2 4 1 6
```

5 points:

For this problem, your program should read the 2 input files and compute the popularity of each semilab. The output should be N rows, where each row contains the semilab number followed by 4 integers indicating the number of students who requested the semilab as their 1st to 4th choices, followed by the total “popularity” of the semilab.
For the sample input files, the following output should be produced:

```
1 2 3 3 2 10
2 3 2 1 3 9
3 2 3 0 3 8
4 0 2 5 0 7
5 0 0 0 0 0
6 3 0 1 2 6
```

Hint:

Using two-dimensional array makes the task easy.

Solution:

Java

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Sched5 {
  private Map<Integer, Integer>   labs       = new HashMap<>(); // lab id -> max number of
students
  private Map<Integer, Integer[]> students   = new HashMap<>(); // student id —> [choice_1,
choice_2, choice_3, choice_4]
```

```
  private Map<Integer, Integer[]> popularity = new HashMap<>(); // lab id —> [total_1st_choice,
total_2nd_choice, total_3rd_choice, total_4th_choice, total_choice]

  public void readSemilabs(String fname) throws IOException {
    try(Stream<String> stream = Files.lines(Paths.get(fname))) {
      stream.forEach(line -> {
        // System.out.println(line);
        String[] nums = line.trim().split("\\s*[\\s,]\\s*");
        // the following will croak if the elements are not all integers
        int[] numbers = Arrays.stream(nums).map(s ->
Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
        if(numbers.length != 2)
          throw new AssertionError("must have 2 numbers");
        if(numbers[1] <= 0)
          throw new AssertionError("must be > 0");
        labs.put(numbers[0], numbers[1]);
      });
    }
  }

  public void readStudents(String fname) throws IOException {
    try(Stream<String> stream = Files.lines(Paths.get(fname))) {
      stream.forEach(line -> {
        // System.out.println(line);
        String[] nums = line.trim().split("\\s*[\\s,]\\s*");
        // the following will croak if the elements are not all integers
        int[] numbers = Arrays.stream(nums).map(s ->
Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
        if(numbers.length != 5)
          throw new AssertionError("must have 5 numbers");
        Arrays.stream(numbers).skip(1).forEach(x -> {
          if(!labs.containsKey(x))
            throw new AssertionError("invalid lab");
        });
        Integer[] ns = Arrays.stream(numbers).skip(1).boxed().toArray(Integer[]::new);
        students.put(numbers[0], ns);
      });
    }
  }

  public void calcPopularity() {
    labs.forEach((k,v) -> popularity.put(k, IntStream.range(0,5).mapToObj(i ->
0).toArray(Integer[]::new))); // init to all 0
    students.forEach((studentId, labChoices) -> {
      IntStream.range(0, labChoices.length).forEach(i ->{
        int lab = labChoices[i];
        popularity.get(lab)[i]++;
        popularity.get(lab)[4]++;
      });
    });
  }

  public void printPopularity() {
    popularity.forEach((lab, choices) -> {
      System.out.printf("%d ", lab);
      Arrays.stream(choices).forEach(choice -> System.out.printf("%d ", choice));
      System.out.println();
    });
  }

  public static void main(String[] args) throws IOException {
    Sched5 sched = new Sched5();
    sched.readSemilabs("/data1/proj/TestP/semilabs.txt");
    sched.readStudents("/data1/proj/TestP/students.txt");
    sched.calcPopularity();
    sched.printPopularity();
    System.out.println("end.");
  }
}
```

Python-3

```
# lab id -> max number of students
def read_semilabs(fname):
  labs = {}
  with open(fname, "r") as f:
    for line in f:
      numbers_str = line.strip().split()
      numbers = [int(x) for x in numbers_str]
      assert(len(numbers) == 2)
      assert(numbers[1] > 0)
      labs[numbers[0]] = numbers[1]
  return labs

# student id —> [choice_1, choice_2, choice_3, choice_4]
def read_students(fname, labs):
  students = {}
  with open(fname, "r") as f:
    for line in f:
      numbers_str = line.strip().split()
      numbers = [int(x) for x in numbers_str]
      assert(len(numbers) == 5)
      assert(all(x in labs for x in numbers[1:5]))
      students[numbers[0]] = numbers[1:5]
  return students

# lab id —> [total_1st_choice, total_2nd_choice, total_3rd_choice, total_4th_choice,
total_choice]
def calc_popularity(labs, students):
  popularity = {}
  for lab in labs:
    popularity[lab] = [0]*5
  for student_id, lab_choices in students.items():
    for i in range(len(lab_choices)):
      lab = lab_choices[i]
      popularity[lab][i] += 1
      popularity[lab][4] += 1
  return popularity

def print_popularity(popularity):
  for lab in sorted(popularity.keys()):
    print("%d " % lab, end='')
    [print("%d " % x, end='') for x in popularity[lab]]
    print()
```

```
labs = read_semilabs("semilabs.txt")
students = read_students("students.txt", labs)
popularity = calc_popularity(labs, students)
print_popularity(popularity)
print("end.")
```

10 points:

For this problem, your program should read the 2 input files and determine if there is an assignment that gives each student his/her 1st choice as well as another semilab from the other 3 semilabs he/she indicated. If there is no such assignment possible, then output "no satisfactory match". If such an assignment is possible, print:
- M rows, one for each student, containing student_id, assigned_semilab_1, assigned_semilab_2;
and
- a list of 4 integers: number of students who got their 1st choice, 2nd choice, 3rd choice and 4th choice.
For the sample input files, the program should produce:

```
111 3 4
3 1 2
4 2 6
223 6 3
211 3 6
98 6 3
44 1 4
52 6 4
67 2 1
21 2 1

10 1 5 4
```

Hint:

You can use a depth first search using the two-dimensional array of choices. Don't forget to keep occupancy counts in an array.

Solution:

Java

```
/*
The problem asks explicitly about the 1st choice, so we'll check it separately.

To find the other 3 choices it's convenient to construct a table:

               1st student   2nd student   3rd student ...
2nd choice ->     lab A         lab D         lab G    ...
```

```
3rd choice ->     lab B         lab E         lab H    ...
4th choice ->     lab C         lab F         lab I    ...

Each cell in a column (containing a lab id) is connected to every cell on the column to the
right of it.
Now, if you imagine a virtual start node to the left of the table and an end node to the right,
all we need to do is to find a path from start to finish that satisfies the labs capacity
condition.
For example, performing a depth first search we can keep a running occupancy array:
going "down" a node we increment the students' count for the lab, and compare it with the max
allowed,
aborting this branch if necessary, and when back—tracking "up" a node we decrement the count.
If we reached the final node then we found an assignment. If we exhausted the search then there
is
"no satisfactory match". We should also keep a list of current nodes as we need it later
to calculate the summary of choices.

Although it is not required for this problem, in real life we'd want to find the most desirable
solution, i.e.,
ideally, the 2nd choice for all; if not, then the least number of 3rd choices; and if there's
no solution in
the space of 2nd and 3rd choices, only then use the 4th choice. Actually, a solution to this,
imho, is not
more complicated as to the original problem if we use an auxiliary array for a path:
the length is the number of students (number of columns in our table), and the element is a
choice for the
current student. For example, we start with all zeros - corresponding to the 2nd choice for all
- then we
successfully add ones (imagine filling up a binary number with "1" bits). When we explored all
paths in the
first two rows then we switch to ternary system and start adding the 4th choice. This will give
us the most
optimal solution. However, for illustration purposes, I’d like to code a Genetic Algorithm
which is often
helps when the search space is too large to try all the potential solutions. We will randomly
change the choices;
these are our "mutations". We'll also select more desirable choices with more probability.
*/

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.util.stream.Stream;

public class Sched10 {
  private Map<Integer, Integer>   labs       = new HashMap<>(); // lab id -> max number of
students
  private Map<Integer, Integer[]> students   = new HashMap<>(); // student id —> [choice_1,
choice_2, choice_3, choice_4]
  private Map<Integer, Integer>   chosenLabs = new HashMap<>(); // lab id -> count
  private Random rand = new Random();

  public void readSemilabs(String fname) throws IOException {
    try(Stream<String> stream = Files.lines(Paths.get(fname))) {
      stream.forEach(line -> {
        // System.out.println(line);
        String[] nums = line.trim().split("\\s*[\\s,]\\s*");
        // the following will croak if the elements are not all integers
        int[] numbers = Arrays.stream(nums).map(s ->
Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
        if(numbers.length != 2)
          throw new AssertionError("must have 2 numbers");
        if(numbers[1] <= 0)
          throw new AssertionError("must be > 0");
        labs.put(numbers[0], numbers[1]);
      });
    }
  }

  public void readStudents(String fname) throws IOException {
    try(Stream<String> stream = Files.lines(Paths.get(fname))) {
      stream.forEach(line -> {
        // System.out.println(line);
        String[] nums = line.trim().split("\\s*[\\s,]\\s*");
        // the following will croak if the elements are not all integers
        int[] numbers = Arrays.stream(nums).map(s ->
Integer.valueOf(s)).mapToInt(Integer::intValue).toArray();
        if(numbers.length != 5)
          throw new AssertionError("must have 5 numbers");
        Arrays.stream(numbers).skip(1).forEach(x -> {
          if(!labs.containsKey(x))
            throw new AssertionError("invalid lab");
        });
        Integer[] ns = Arrays.stream(numbers).skip(1).boxed().toArray(Integer[]::new);
        students.put(numbers[0], ns);
      });
    }
  }

  public boolean canMake1stChoice() {
    for(Map.Entry<Integer,Integer[]> entry : students.entrySet()) {
      Integer studentId = entry.getKey();
      Integer[] labChoices = entry.getValue();
      chosenLabs.compute(labChoices[0], (k, v) -> 1 + (v == null ? 0 : v));
      if(chosenLabs.get(labChoices[0]) > labs.get(labChoices[0]))
        return false;
    }
    return true;
  }

  // true on 1st val in dict2 which is over corresponding value in dict1
  // assumes the keys are the same in both
  public boolean diffDict(Map<Integer, Integer> dict1, Map<Integer, Integer> dict2) {
    for(Map.Entry<Integer,Integer> entry : dict2.entrySet()) {
      if(dict1.get(entry.getKey()) - entry.getValue() < 0)
        return true;
    }
    return false;
  }

  // m - number of choices
  public Integer[] mutatePath(Integer[] path, int m) {
    int i = rand.nextInt(path.length); // random index in path
    int j;
    do {
      double x = rand.nextDouble();
      j = (int)((Math.exp(x)-1) * m / (Math.E-1)); // more preferable choice gets exponentially
more hits
    } while(path[i] == j);
    // random choice
    Integer[] path2 = Arrays.copyOf(path, path.length);
    path2[i] = j;
    return path2;
  }

  // return { student id -> choice number (from 2nd till 4th }
  public Integer[] canMakeAnotherChoice() {
    int n = students.size();
    Integer[] a = new Integer[0];
    Integer[] studentIds = students.keySet().toArray(a);
    int[][] graph = new int[3][n];
    // fill our table
    for(int j=0; j<n; j++) {
      for(int i=0; i<3; i++)
        graph[i][j] = students.get(studentIds[j])[i+1];
    }

    // subtract 1st choice labs from capacity
    for(int student : studentIds) {
      int lab = students.get(student)[0];
      labs.compute(lab, (k, v) -> v - 1);
    }

    Set<List<Integer>> visited = new HashSet<>(); // not to repeat ourselves
    Integer[] path = Arrays.stream(new int[n]).boxed().toArray(Integer[]::new); // init to 2nd
choice
    visited.add(Arrays.asList(path));

    for(int i=0; i<100000; i++) {
      System.out.printf("trying [%d]: ", i);  System.out.println(Arrays.toString(path));

      // calc occupancy for current path
      Map<Integer, Integer> occupancy = new HashMap<>(labs);
      occupancy.replaceAll((k,v) -> v = 0);
      for(int j=0; j<n; j++) {
        int lab = graph[path[j]][j];
        occupancy.compute(lab, (k,v) -> v + 1);
      }

      // see if there is any overbooking
      if(diffDict(labs, occupancy)) {
        // try next path
        for(int j=0; j<1000; j++) {
          Integer[] path2 = mutatePath(path, 3);
          if(!visited.contains(Arrays.asList(path2))) {
            path = path2;
            visited.add(Arrays.asList(path));
            break;
          }
        }
      }
      else // found a path
        return path;
    }
    return null;
  }

  public void printChoices(Integer[] choices) {
    int i = 0;
    for(Map.Entry<Integer,Integer[]> entry : students.entrySet()) {
      Integer studentId = entry.getKey();
      Integer[] labChoices = entry.getValue();
      System.out.printf("%3d %d %d\n", studentId, labChoices[0], labChoices[choices[i++] + 1]);
    }
    int[] satisfaction = new int[4];
    satisfaction[0] += students.size(); // by this point everybody got their 1st choice
    Arrays.stream(choices).forEach(choice -> satisfaction[choice+1]++);
    System.out.println(Arrays.toString(satisfaction));
  }

  public static void main(String[] args) throws IOException {
    Sched10 sched = new Sched10();
    sched.readSemilabs("/data1/proj/TestP/semilabs.txt");
    sched.readStudents("/data1/proj/TestP/students.txt");
    if(sched.canMake1stChoice()) {
      Integer[] choice = sched.canMakeAnotherChoice();
      if(choice != null)
        sched.printChoices(choice);
      else
        System.out.println("no satisfactory match");
    }
    else
      System.out.println("no satisfactory match");
    System.out.println("end.");
  }
}
```

Python-3

```
"""
The problem asks explicitly about the 1st choice, so we'll check it separately.

To find the other 3 choices it's convenient to construct a table:

               1st student   2nd student   3rd student ...
2nd choice ->     lab A         lab D         lab G    ...
3rd choice ->     lab B         lab E         lab H    ...
4th choice ->     lab C         lab F         lab I    ...

Each cell in a column (containing a lab id) is connected to every cell on the column to the
right of it.
Now, if you imagine a virtual start node to the left of the table and an end node to the right,
all we need to do is to find a path from start to finish that satisfies the labs capacity
condition.
For example, performing a depth first search we can keep a running occupancy array:
going "down" a node we increment the students' count for the lab, and compare it with the max
allowed,
aborting this branch if necessary, and when back—tracking "up" a node we decrement the count.
```

```
If we reached the final node then we found an assignment. If we exhausted the search then there
is
"no satisfactory match". We should also keep a list of current nodes as we need it later
to calculate the summary of choices.

Although it is not required for this problem, in real life we'd want to find the most desirable
solution, i.e.,
ideally, the 2nd choice for all; if not, then the least number of 3rd choices; and if there's
no solution in
the space of 2nd and 3rd choices, only then use the 4th choice. Actually, a solution to this,
imho, is not
more complicated as to the original problem if we use an auxiliary array for a path:
the length is the number of students (number of columns in our table), and the element is a
choice for the
current student. For example, we start with all zeros - corresponding to the 2nd choice for all
- then we
successfully add ones (imagine filling up a binary number with "1" bits). When we explored all
paths in the
first two rows then we switch to ternary system and start adding the 4th choice. This will give
us the most
optimal solution. However, for illustration purposes, I'd like to code a Genetic Algorithm
which is often
helps when the search space is too large to try all the potential solutions. We will randomly
change the choices;
these are our "mutations". We'll also select more desirable choices with more probability.
"""

import numpy as np

# lab id -> max number of students
def read_labs(fname):
  labs = {}
  with open(fname, "r") as f:
    for line in f:
      numbers_str = line.strip().split()
      numbers = [int(x) for x in numbers_str]
      assert(len(numbers) == 2)
      assert(numbers[1] > 0)
      labs[numbers[0]] = numbers[1]
  return labs

# student id -> [choice_1, choice_2, choice_3, choice_4]
def read_students(fname, labs):
  students = {}
  with open(fname, "r") as f:
    for line in f:
      numbers_str = line.strip().split()
      numbers = [int(x) for x in numbers_str]
      assert(len(numbers) == 5)
      assert(all(x in labs for x in numbers[1:5]))
      students[numbers[0]] = numbers[1:5]
  return students

def can_make_1st_choice(labs, students):
  chosen_labs = {}
  for lab in labs:
    chosen_labs[lab] = 0
  for student_id, lab_choices in students.items():
    chosen_labs[lab_choices[0]] += 1
    if chosen_labs[lab_choices[0]] > labs[lab_choices[0]]:
      return False
  return True

# true on 1st val in dict2 which is over corresponding value in dict1
# assumes the keys are the same in both
def diff_dict(dict1, dict2):
  for key,val in dict2.items():
    if dict1[key]-val < 0:
      return True
  return False

# m - number of choices
def mutate_path(path, m):
  i = np.random.randint(0, len(path)) # random index in path
  while True:
    x = np.random.rand()
    j = ((np.exp(x)-1)*m/(np.e-1)).astype(int) # more preferable choice gets exponentially more
hits
    if path[i] != j:
      break
  # random choice
  path2 = path.copy()
  path2[i] = j
  return path2

# return { student id -> choice number (from 2nd till 4th }
def can_make_another_choice(labs, students):
  n = len(students)
  students_id = list(students.keys())
  graph = np.zeros([3,n], dtype=int)
  # fill our table
  for j in range(n):
    for i in range(3):
      graph[i,j] = students[students_id[j]][i+1]

  # subtract 1st choice labs from capacity
  for student in students_id:
    lab = students[student][0]
    labs[lab] -= 1

  visited = set() # not to repeat ourselves
  path = np.zeros(n, dtype=int) # init to 2nd choice
  visited.add(tuple(path))

  for i in range(1000):
    print("trying: [%d] " % i, path)

    # calc occupancy for current path
    occupancy = dict.fromkeys(labs, 0)
    for j in range(n):
      lab = graph[path[j],j]
      occupancy[lab] += 1

    # see if there is any overbooking
    if diff_dict(labs, occupancy):
      # try next path
      for j in range(1000):
        path2 = mutate_path(path, 3)
        if tuple(path2) not in visited:
          path = path2
          visited.add(tuple(path))
          break
    else: # found a path
      return True, path
  return False, []

def print_choices(students, choices):
  i = 0
  for student_id, lab_choices in students.items():
    print("%3d %d %d" % (student_id, lab_choices[0], lab_choices[choices[i]+1]))
    i += 1
  satisfaction = np.zeros(4, dtype=int)
  satisfaction[0] += len(students) # by this point everybody got their 1st choice
  for choice in choices:
    satisfaction[choice+1] += 1
  print(satisfaction)

labs = read_labs("semilabs.txt")
students = read_students("students.txt", labs)
if can_make_1st_choice(labs, students):
  res, choices = can_make_another_choice(labs, students)
  if res:
    print_choices(students, choices)
  else:
    print("no satisfactory match")
else:
  print("no satisfactory match")
print("end.")
```

LINGUISTICS

5 points:

Consider the following words/phrases from a language spoken in the northwest of Russia:

kujŋətenək	‘near the glass’
raralqək	‘on top of the roof’
raraɣiŋəŋ	‘into the basement’
aŋqakin	‘from the sea’
aŋqan	‘sea’
keŋən	‘bear’
keŋəlqəkin	‘from the bear’
raralqən	‘roof’
kujŋəŋ	‘into the glass’
keŋək	‘inside the bear’
aŋqatenək	‘on the shore’

Problem: translate into this language the following words/phrases and explain your reasoning.

1. basement
2. inside the house
3. glass
4. from the roof
5. to the bear

Solution:

Matching the words from the language and their translations, we can deduce that *kujŋə* means “glass”, *aŋqa* means “sea”, and *keŋə* means “bear”.

Also, -*n* corresponds to nominative case, -*tenək* means “near”, -*ŋ* corresponds to “into”, -*kin* stands for “from”, -*k* means “in, on”. Therefore, *aŋqatenək* actually means “near the sea”. *keŋəlqəkin* can be deconstructed into *keŋə* “bear”, *kin* “from”, and an unknown morpheme -*lqə-.* We can hypothesize that it means “space above” or something like that: it can also be seen in *raralqək* “on top of the roof” — which is literally translated as “in the space above the house”, and therefore *rara* means “house”. Similarly, *raraɣiŋəŋ* is literally “into the space under the house”, and therefore *ɣiŋə* stands for “space under”. We can also observe that -*tenək* can be decomposed into -*tenə*- and -*k,* where *tenə* stands for “space near”.

Therefore, the words in the problem have the following structure: 1). Object; 2). Part of the space relative to the object, 3). Case ending.

The translations are therefore the following:

1. basement *rara-γiŋə-n*
2. inside the house *rara-k*
3. glass *kujŋə-n*
4. from the roof *rara-lqə-kin*
5. to the bear *keŋə-teŋə-ŋ*

10 points:

Consider the following sentences in a Uralic language:

Am os am ürtum Pet'a men as'agamen eruptijagamen.
'I and my friend Petja love our fathers.'

Man nang samyn pusmaltiluv.
'We heal your eye.'

Nang ürtyn eruptilyn.
'You love your friend.'

Am samagum pantijagum.
'I close eyes.'

Am man luvuv os nang ampanyn pusmaltijanum.
'I heal our horse and your dogs.'

Problem: Translate the following and explain your reasoning.

Into English: *Man üratanuv eruptijanuv.*
From English: You heal my father and my friend.

Solution:

erupti - "to love"
pusmalti - "to heal"
panti - "to close"
ürt - "friend"
as' - "father"
sam - "eye"

luv - “horse”
amp - “dog”

The verb endings correspond to the pronouns: I *-um*; you.singular *-yn*; we *-uv/-amen*. Same endings can be found on nouns, and they indicate possession: my *-um*, your *-yn*; our *-uv/-amen*.

We can also notice that words *am*, *nang*, *man* can be both personal pronouns and possessive adjectives: “I”, “you”, “we” and “my”, “your”, “our” — they double endings, as in, for example, ***nang*** *sam-**yn**.*

Between the stem and the ending there are morphemes such as *-an-/-jan-* and *-ag-/-jag-*. They indicate whether the noun is plural or whether the number of the verbal object is plural. Singular nouns don’t have any marking, but verbs with singular objects have morpheme *-l-*. We can observe that in reality there is a difference between *-an-* and *-ag-*: *-an-* means plural number, and *-ag-* means dual number (where there are two of something).

Now knowing about the existence of dual, we can understand that *-amen* is the ending corresponding to “we.dual”, as well as “our.dual”.

The translations of the phrases are therefore the following:

1. We love our friends.
2. *Nang am as’um os am ürtum pusmaltijagyn.*