Coder Games 2025 Tasks

Deutsche Telekom IT Solutions

March - April 2025

Task 1

Anagram Pairs

In the heart of the country, nestled between the Tatra mountains, a brilliant young programmer named Janko dreamed of creating the fastest algorithm to predict the movements of the legendary treasure-hunting raven. One day, he received an urgent challenge from a secret society of historians who had discovered ancient encrypted maps hidden in Bratislava Castle. With only 30 minutes to decode the patterns, Janko had to write a program in Java, Python, or C++ to reveal the lost locations of medieval riches. As the clock ticked, his screen flashed with success—his algorithm had cracked the mystery, pinpointing a hidden cave near Spiš Castle. Celebrated as a national hero, Janko's code became a legend, proving that even the greatest secrets of the country could be unlocked with the right logic and a bit of ingenuity.

Problem Statement:
An anagram is a word or phrase formed by rearranging the letters of another word. Your task is to write a program that checks if two given words are anagrams of each other.

Input

- The program should read a single line containing two space-separated words.
- Words consist only of lowercase English letters (a-z).

Output

- Print "YES" if the words are anagrams of each other.
- Print "NO" otherwise.

Constraints:

- The solution should run efficiently for the given constraints.
- Sorting-based or dictionary-based approaches are acceptable.

Task 2

Stability of Universe

There is a two-dimensional array with N rows and M columns. The array defines a surface of sphere, so the [0] column is next to the [1] and the [M-1] columns, like the rows also (the [0] row is next to the [1] and the [N-1] rows). This array defines the Universe. Values at the array are separated with SPACE (what else could be the separator, then the SPACE character) and 0 <= values < 10.

Calculate the stability of the Universe like this:

1. **search for local maximum points**, where the local maximum is the P point, because the value at the P point is **bigger than** every other 16 values around the P point based on the next pattern. **A gravitational center will be the P point.** The other 8 values around the P point will be 0.

x2'	0	a2	0	y2
0	x1'	a1	y1	0
b2'	b1'	P	b1	b2
0	y1'	a1'	x1	0
y2'	0	a2'	0	x2

1. each center of gravity is clearly separated from the others
2. the value of P will be *bigger*, not *bigger or equal*, than the other 16 values
3. **calculate the stability of each gravitational center**, where the **P** point (as a gravitational center) is in stability when numbers are in balance around the **P** point:
   - the values on the [a,b,x,y] axes are located symmetrically around the point P exactly at the center (a: vertical, b: horizontal, x: diagonal, y: the other diagonal)
   - values are symmetrical when the **st** values are equal with the **st'** values, where $s \in a,b,x,y^{s \in \{a,b,x,y\}}$ and $t \in \{1,2\}^{t \in \{1,2\}}$ (fe.: a1=a1' **and** a2=a2' **and** ... y2=y2')
   - stability needs to be checked on all axes [a,b,x,y]
   - the numbers can be (most of the time *will be*) different on any two axes
     - in case of *Example #1*: around the value **8**: axis **a** has value **3** and **1**; axis **y** has value **4** and **1**, and the numbers are in symmetry around the value **8**
   - a gravitational center (**P** point) is unstable when any proper pair of the numbers on any axis from [a,b,x,y] are not equal around the **P** point (fe.: a1!=a1' **or** a2!=a2' **or** ... y2!=y2')
4. **count the stable and unstable gravitational centers**: after you found all gravitational centers, and you decided about a gravitational center is in balance or not, the questions are:
   - How many gravitational centers are stable, and
   - How many gravitational centers are unstable
5. **stability of the Universe can be described with the numbers of stable and unstable gravitational centers**

Input: a file with a grid representing a Universe. At the first line two numbers will be separated with a SPACE, as the dimension of grid. Fe.: "200 200" or like the example file below, "8 12". After that as many rows and columns as defined, values separated with SPACE also. The Universe is zero filled. The file doesn't contain empty lines between two lines of the universe and no empty line after the definition of the grid size.

Example input file:

```
8 12
4 0 0 3 0 2 0 2 0 3 0 0 4 3
0 2 0 0 5 5 3 5 0 0 1 0 1
0 0 0 1 2 7 2 1 0 0 0 0
0 0 0 0 5 3 5 0 0 0 0 0
0 0 0 3 0 2 0 3 0 0 0 0
0 1 0 0 0 0 0 0 0 2 0 1
4 0 0 0 0 0 0 0 0 4 3
6 3 0 0 0 0 0 0 0 3 6 8
```

Output: Two separated (with a SPACE) numbers. **The first is the number of stable center points and the second is the amount of unstable center points.**

Example #1: this universe has two gravitational centers, and both are stable. The brown **7** and the blue **8** are the local maximums according to the pattern above with the four axes. Numbers around the center points are the same on any axes and any in any distance.

4	0	0	3	0	2	0	3	0	0	4	3
0	2	0	0	5	3	5	0	0	1	0	1
0	0	0	1	2	7	2	1	0	0	0	0
0	0	0	0	5	3	5	0	0	0	0	0
0	0	0	3	0	2	0	3	0	0	0	0
0	1	0	0	0	0	0	0	0	2	0	1
4	0	0	0	0	0	0	0	0	0	4	3
6	3	0	0	0	0	0	0	0	3	6	8

The result should like: **2 0**

Example #2: this universe has two gravitational centers also, one stable and one unstable center point.

4	0	0	3	0	2	0	3	0	0	4	3
0	2	0	0	5	3	5	0	0	1	0	1
0	0	0	1	2	7	2	1	0	0	0	0
0	0	0	0	5	3	5	0	0	0	0	0
0	0	0	1	0	2	0	3	0	0	0	0
0	1	0	0	0	0	0	0	0	2	0	1
4	0	0	0	0	0	0	0	0	0	4	3
6	3	0	0	0	0	0	0	0	3	6	8

The diagonal line with red values is not in balance around this center point. Red values are not equal, this diagonal axis is not in balance, so the center point (with value **7**) is not in balance.

The result should like: **1 1**

There is 1 input file. The grid size is 200x200. It will contain some gravity centers. The separator is the SPACE, and the grid is zero filled.

Task 3

The Lunar Codex Treasure

Story: Deep within the archives of an ancient Minoan civilization, an encrypted message has been discovered. Scholars believe it contains instructions leading to a powerful artifact. However, the message has been deeply encoded using an ancient technique, and additional layers of obscurity were added to deter intruders. Legends speak of a forgotten order that used multiple shifts within the same message, complicating decryption.
Your task is to find all keys and uncover the secret hidden in the text itself. Goals:

- Write a function that reads an encoded message from a file and deciphers it.
- Within the challenge description itself lies a hidden clue that will help uncover the pattern.

Input:

- The input.txt file containing an encoded message. The message may span multiple lines.
- The length of the message does not exceed 104 characters.

- Each decrypted message is used as an input in the next round.

Output:

- The decrypted message.
- The responses should have some readable clue.

Example: phtajvbuavutl $->$ iamtcountonme

Hints:

- The first value corresponds to an important lunar cycle (between 1 and 30).
- After the initial value, subsequent characters may have increasing or decreasing values in a pattern.
- Each message found may contain a clue for the next value.
- Consider frequency analysis if the pattern is not immediately obvious.
- Keys are numerical.

Solution Format:

- The solution should be a function that reads the file, applies the appropriate keys, and prints the decoded text.
- It should prompt the user to continue uncovering clues until it reveals the true message.

Resolution:

- Find all values that decrypt the file and print them each one in a new line.
- Copy the values and past it to the appropriate textbox in submission page.

Task 4

Bike Path in Valencia During Las Fallas

Every year in Valencia, the Las Fallas festival brings beautiful sculptures and fireworks - but also causes temporary street closures across the city. You're using a bike-sharing system and want to find the shortest route from station 1 to station N, avoiding roads that are closed during your journey.

City Details:

- There are N bike stations and M roads
- Each road is bidirectional and takes a specific number of minutes to travel.
- Some roads are closed during specific time intervals (in minutes after midnight).
- You start your trip at station 1 at time T (in minutes after midnight).

Rules:

- If a road is closed at your intended time of travel, you must wait until it reopens.
- There is no upper limit on waiting, but your goal is to minimize total arrival time.
- You may pass through the same station more than once.

Input Format:

N M T

u1 v1 t1 k1 s11 e11 s12 e12 ...

...

uM vM tM kM sM1 eM1 ...

Where:

- N - number of stations (2 <= N <= 1000)

- M - number of roads

- T - starting time (0 <= T < 1440)

- Each road connects stations ui and vi, takes ti minutes to travel

- ki - number of closure intervals for that road

- Each interval [s, e) defines a closed time period (inclusive of s, exclusive of e)

Output Format:

A single integer: the **minimum arrival time** at station N starting from station 1 at time T, or -1 if it's impossible to reach.

Example Input:

5 5 100

1 2 20 1 110 130

2 3 15 0

3 5 30 1 120 150

1 4 25 0

4 5 20 0

Example Output:

145

Explanation:

- Start at station 1 at time 100
- Travel: 1 → 4 (25 min) → arrive at 125
- Travel: 4 → 5 (20 min) → arrive at 145
  (No closures on these roads)

Even though 1 → 2 is shorter, the closure from 110 to 130 would force a wait.

Task 5

The Flawed Autobahn Speed Tests

Problem Statement:
A country famous for its high-speed highways has introduced automated speed monitoring systems. Before deployment, developers wrote unit tests to verify the system's correctness. However, some test cases contain logical errors, leading to false positives or false negatives. Your task is to analyze and identify faulty unit tests where the system's violation flag is incorrect based on the given speed rules. **Input:**

You receive a list of unit tests in the following format:

TEST_ID;CAR_SPEED (km/h);ROAD_TYPE;WEATHER;VIOLATION

- **TEST_ID:** Unique identifier for the test case
- **CAR_SPEED:** The recorded speed of the vehicle
- **ROAD_TYPE:** Either HIGHWAY, CITY, or RURAL
- **WEATHER:** Either CLEAR, RAIN, or FOG
- **VIOLATION:** TRUE if the system flags a violation, FALSE otherwise

The speed rules are:

- **CITY:** 50 km/h (40 km/h in rain, 30 km/h in fog)
- **RURAL:** 100 km/h (80 km/h in rain, 60 km/h in fog)
- **HIGHWAY:** No limit in clear conditions, but 130 km/h in rain, 100 km/h in fog

Output:
Your program should detect the IDs of the faulty unit tests where the VIOLATION flag does not correctly reflect whether a speeding violation occurred according to the speed rules.

Example Input:

```
T001;55;CITY;CLEAR;FALSE
T002;120;HIGHWAY;RAIN;FALSE
T003;90;RURAL;FOG;TRUE
T004;130;HIGHWAY;FOG;TRUE
T005;50;CITY;RAIN;TRUE
```

Example Output:

```
T001

T005
```