## ITMO x Yandex 2025

**ФИО:** Кудряшов Георгий Андреевич
**Логин "Яндекс.Контеста":** Goshanenterprise@yandex.ru

### Задача 1. Fail2ban на коленке (6)

Предлагаю сначала рассмотреть содержимое Регулярного Выражения:

> `\[\d{4}-\d{2}-\d{2}\s+\d{2}:\d{2}:\d{2}\]`

**Блок проверяет соответствие метки Timestamp в записи журнала требуемому формату**.
Сам блок также состоит из двух частей: проверки даты и проверки времени. На счет использования "\s+", я долго думал, так как сначала рассчитывал на аномалии в логах. Однако когда оказалось что их нет, данный разделитель стал бесполезен. Однако если допустим будет такая ситуация что писателей логов станет несколько (и кто-то будет использовать вместо табуляции пробел) или слетит форматирование пробелов, то такой формат разделителей будет более гибким. Хотя все таки все зависит от конкретной итоговой задачи.

> `\s+WARNING\s+Request\s+from\s+`

Строго проверяет запись на наличие шаблона предупреждения о подозрительном запросе. Ничего особенного просто несколько слов разделенных пробелами. Хотя можно подчеркнуть как раз здесь используется табуляция и использование \s+ здесь не мешает правильно обрабатывать все записи и кроме этого придает некоторой гибкости шаблону (хотя эта гибкость здесь конечно не нужна, т.к. тестовые данные были полностью чистыми).

> `(?P<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})`

Первая именованная группа которая парсит IP подозрительной записи. Шаблон на 4 блока цифр (от 1 до 3), разделенных точками.

> \s+\("(?P<method>[A-Z]+)

Вторая именованная группа которая парсит метод запроса (в тестовых данных были только GET запросы, но и с другими я думаю шаблон неплохо справится). Представляет собой шаблон на несколько латинских букв в upper case.

> \s+(?P<path>.*)\s+HTTP/\d\.\d"\)

Третья именованная группа которая парсит адрес на который обращался запрос. Представляет собой мешанину из множества символов, которые при надобности можно тоже распарсить, однако в задании такого требования не заявлялось. Поэтому парсим все символы которые встречаются на пути до того как дойдем до протокола.

> \s+is\s+malicious\b

Обязательный для записи с предупреждением шаблон, который мы жестко закрепляем в РВ.

Ну и собственно мы читаем все строки файла, сравниваем их с регулярным выражением и если есть совпадение печатаем в консоль (ну или делаем что хотим):

```
import re

ans = []
s_method = set()
with open("server.log", "r") as inp:
    new_p = re.compile(r'\[\d{4}-\d{2}-
\d{2}\s+\d{2}:\d{2}:\d{2}\]\s+WARNING\s+Request\s+from\s+(?
P<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})\s+\("(?P<method>[A-Z]+)\s+(?
P<path>.*)\s+HTTP/\d\.\d"\)\s+is\s+malicious\b')
    for line in inp.readlines():
        x = new_p.match(line)
        if (x):
            ans.append(x)
            s_method.add(x.group("method"))
```

```
            print(line, end="")
            ip = x.group('ip')
            method = x.group('method')
            path = x.group('path')
            print(f"IP:     {ip}")
            print(f"Method: {method}")
            se = "=" * 100
            print(f"Path:   {path}", end=f"\n\n{se}\n\n")
```

**Пример вывода**

```
[2024-04-14 01:58:16] WARNING   Request from 25.180.8.182 ("GET
/api/index.php/v1/banners?public=true HTTP/1.1") is malicious
IP:     25.180.8.182
Method: GET
Path:   /api/index.php/v1/banners?public=true
```

### Задача 6. Преобразования метрик (12)

Задача довольно типичная уровня easy Литкод или Div 1 Codeforces. Так и намекает, что это НЕЯВНЫЙ граф и что суть состоит в том чтобы обойти граф в ширину и найти кратчайший путь до целевого значения метрики.

**Алгоритм:**

1. Считываем k.
2. K раз повторяем операции считывания новых значений A, B, C
3. Обходим граф в ширину и находим кратчайший путь

**Идея обхода:**

1. Давайте заведем массив расстояний **d** длины **n+1** (для учета того что мы стартуем с **1** и идем к **n**), и для каждого **d[i]** будем хранить минимальное кол-во операций чтобы добраться до данного **i**. Заполним массив **-1** как индикатор того что узел еще не посещен. Также устанавливаю расстояние до **1** как **0**, так как мы уже находимся в данной позиции.
2. Создадим очередь в которую будем помешать вершины которые нам нужно обойти. И сразу поместим туда **1**.
3. Затем мы входим в цикл и пока очередь не пуста, мы забираем из нее первую вершину.

4. После чего рассчитываем **4** следующих вершины (**x *A,* *x* B, x + 1, x + C**). Помещаем каждую из вершин в очередь и рассчитываем для каждой из вершин расстояние для них, как расстояние до значения x + 1. **d[x] + 1**
5. Также при расчете новых вершин внимательно следим за тем, чтобы не вылезть за границы массива **d**. Так как дальше значения метрики **n**, нам считать не требуется.
   После того как мы прошли все вершины и ни одной не осталось в очереди. Выходим из цикла и возвращаем как ответg посчитанное расстояние до вершины **n - d[n]**.

*Также если бы задача включала в себя вывод всех преобразований для достижения метрики, нужно было бы всего лишь завести дополнительный массив в котором мы бы хранили предков (для каждой вершины, номер вершины из которой мы попали в нее) для восстановления ответа.*