Розрахунково-графічна робота

з дисципліни "Системи та технології кібернетичної безпеки"

Виконав: студент групи ФБ-12 Приходько Юрій

1. Завдання на розрахунково-графічну роботу

Постановка задачі:

Розробити програму автентифікації користувача по клавіатурному почерку.
Вимоги до програми:

1. Програма повинна працювати в двох режимах:
   - навчання (створення біометричного еталону)
   - ідентифікації (порівняння з біометричним еталоном).
2. На етапі навчання необхідно визначати еталонні статистичні параметри клавіатурного почерку – оцінки математичного чекання і дисперсії тривалості утримання клавіш. Параметри повинні записуватися у файл. Вчення виробляти по багатократному набору фіксованій контрольній фразі, символи якої рівномірно розподілені по клавіатурі.
3. На етапі ідентифікації необхідно визначити параметри введеної контрольної фрази і перевірити гіпотезу про те, що отримані оцінки математичного очікування і дисперсії належать тому ж розподілу, що і параметри біометричного еталону. На цьому етапі необхідно відображувати отримані оцінки і еталонні параметри. Рівень значущості критерію задавати в діалоговому вікні.
4. На етапах навчання і ідентифікації передбачити можливість відбракування грубих помилок (окремих вимірів)

2. Виконання завдання

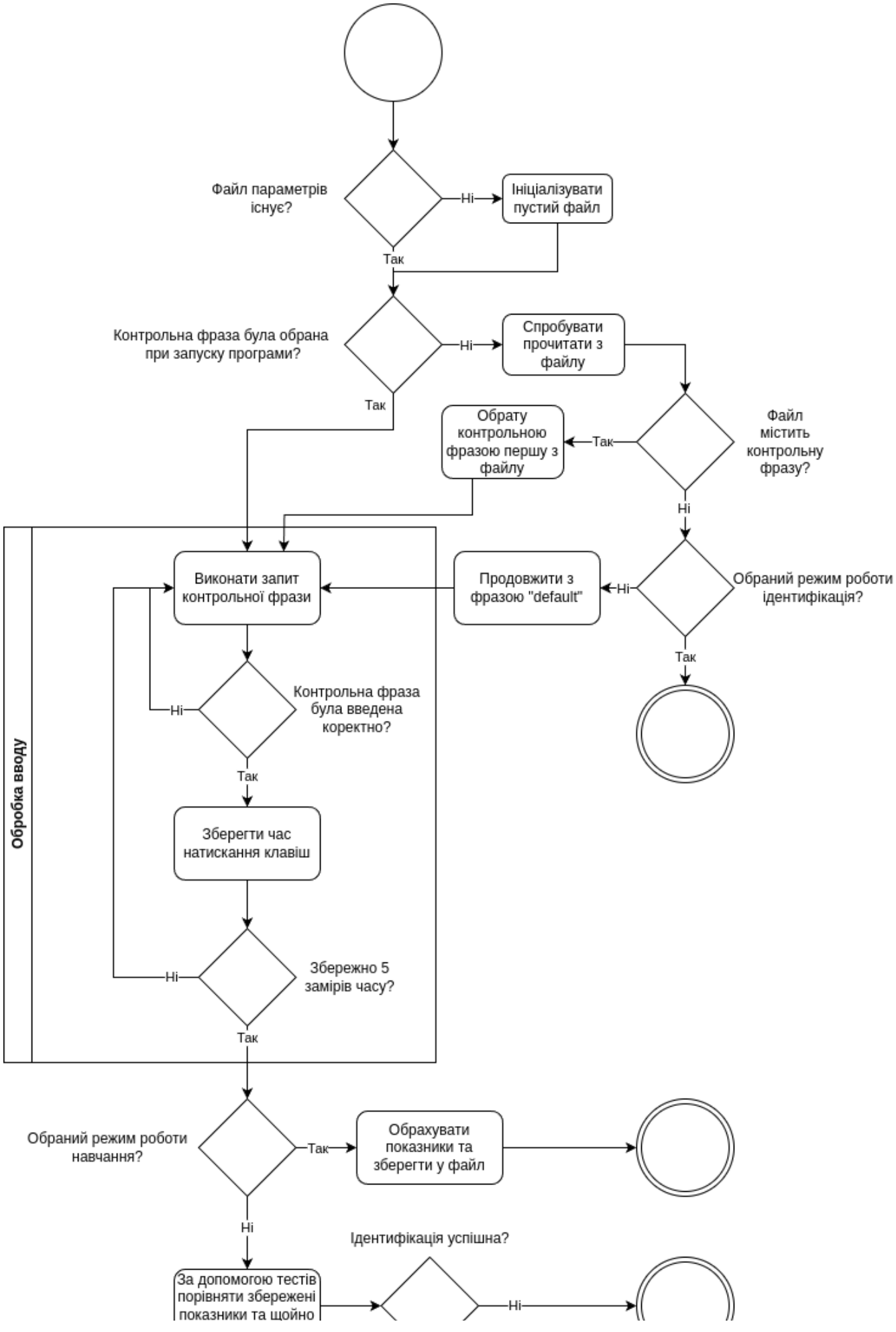
Аналіз поставленої задачі

В результаті аналізу поставленої задачі було визначено, що в результаті виконання роботи має бути отриманий застосунок що відповідає вимогам задачі, а саме:

- При запуску застосунку обирається режим роботи, та додаткові параметри (наприклад обрання контрольної фрази, режим "гучної" роботи програми)

- Дані отримані в режимі навчання мають зберігатись в окремому файлі, який ініціалізується під час виконання програми. Для зручності роботи було обрано збереження даних у форматі JSON.
- В обох режимах роботи користувач повинен вводити контрольну фразу декілька разів, для того аби створити вибірку значень. Була обрана кількість введень 5.
- В результаті обробки введених даних в режимі ідентифікації користувач має бути ідентифікований, або відмовлений.
- При обробці необхідно відбраковувати грубі помилки, провести F-test та t-test

Блок схема алгоритму роботи програми


Файл параметрів існує?
Ні
Ініціалізувати пустий файл
Так
Контрольна фраза була обрана при запуску програми?
Ні
Спробувати прочитати з файлу
Так
Обрату контрольною фразою першу з файлу
Так
Файл містить контрольну фразу?
Ні
Обраний режим роботи ідентифікація?
Ні
Продовжити з фразою "default"
Виконати запит контрольної фрази
Так
Обробка вводу
Ні
Контрольна фраза була введена коректно?
Так
Зберегти час натискання клавіш
Ні
Збережно 5 замірів часу?
Так
Обраний режим роботи навчання?
Так
Обрахувати показники та зберегти у файл
Ні
Ідентифікація успішна?
За допомогою тестів порівняти збережені показники та щойно
Ні

Математичний апарат

1. F-тест (тест Фішера) - це статистичний тест, який використовується для порівняння двох дисперсій або для аналізу загальних відмінностей між групами у множинному порівнянні. Назва тесту походить від Френсіса Фішера, який розробив цей метод.

F-тест застосовується в таких основних випадках:

- Перевірка рівності двох дисперсій (F-тест на рівність дисперсій):
  - Цей тест використовується для перевірки гіпотези про те, що дві вибірки мають однакові дисперсії.
- Аналіз варіацій (ANOVA):
  - Використовується для порівняння середніх значень трьох або більше груп.
  - ANOVA дозволяє визначити, чи існують значущі відмінності між середніми значеннями груп.

Нехай H_0 - нульова гіпотеза, "Дисперсії двох груп рівні", H_α альтернативна гіпотеза, "Дисперсії двох груп не рівні".
F-статистика обраховується наступним чином:

$$\mathrm{F} = \frac{s_1^2}{s_2^2}$$
$$\textit{Де } s_1^2 \textit{ та } s_2^2 \ - \ \textit{вибіркові дисперсії двох вибірок},$$
$$\textit{при тому } s_1^2 \ - \ \textit{більша дисперсія}, \ \textit{а } s_2^2 \textit{ менша}$$
$$\textit{обраховується як } s^2 = \frac{\sum (X_i - \bar{X})^2}{n-1}$$

Після цього порівнюємо F-статистику з критичним значенням з таблиці F-розподілу при заданому рівні значущості α та відповідних ступенях свободи.

2. t-тест - це статистичний тест, який використовується для порівняння середніх значень між вибірками або для порівняння середнього значення однієї вибірки з відомим значенням. Він підходить для даних, які приблизно відповідають нормальному розподілу (в нашому випадку ми вважаємо час натискань нормальним розподілом) і коли розмір вибірки невеликий.

Існують три основні типи t-тестів:

1. Одновибірковий t-тест (One-sample t-test):
    - Використовується для порівняння середнього значення вибірки з відомим значенням.
2. Двовибірковий t-тест (Two-sample t-test):
    - Використовується для порівняння середніх значень двох незалежних вибірок.
    - Може бути двох типів:
        - Припущення про рівність дисперсій (пулінг дисперсій).
        - Припущення про нерівність дисперсій (тест Уелча).
3. Парний t-тест (Paired t-test):
    - Використовується для порівняння середніх значень двох залежних вибірок (наприклад, до і після експерименту).

Нехай H_0 - нульова гіпотеза, "середні значення двох груп рівні", H_α альтернативна гіпотеза, "середні значення двох груп не рівні".

Формула для t-значення:

$$\mathrm{t} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Тут $\bar{X}_1$ і $\bar{X}_2$ - середні значення, s_1^2 і s_2^2 - дисперсії, n_1 і n_2 розміри вибірок.

Архітектура програми

Найзручніше для написання програми використати мову програмування `python3`, використання модулів цієї мови дозволить ефективно виконати поставлені задачі. Таким чином було вирішено для засікання часу набору використати модуль `keyboard`, для обрахунків табличних значень в тестах - модуль `scipy`.
Для нативно зрозумілої реалізації зручно розбити програму на компоненти, таким чином отримуємо наступну архітектуру застосунку:

- Файл `constants.py` - використовується для зберігання відомих значень що дозволяють налаштовувати роботу програми і часто в ній використовуються.
- Файл `collector.py` - містить клас `Collector` що відповідає на збір інформації при вводі користувача
- Файл `calc.py` - містить функції що відповідають за математичні операції і розрахування тестів.
- Файл `app.py` - основний файл програми, утилізує роботу всіх інших функцій імплементуючи тим самим логічний потік програми, виконаний у форматі консольного застосунку.

Доступ до вихідних кодів програми за посиланням:
https://github.com/gR4tiG0/program_sec_2024_CGW/

Далі буде розібрано основні функції застосунку, коментарі до коду є і тут, і в коді на репозиторії `github`.

`app.py`

```
#!/usr/bin/env python3
from constants import *
from collector import Collector
from calc import getStatistics, checkIdentification
from json import dump, load, JSONDecodeError
import argparse
import logging


logger = logging.getLogger("APP LOGGER")
logging.basicConfig(level=logging.INFO)


def getKeyStore() -> dict:
    #load keystore from file, if file not found or empty, return empty dict
    #stores mean and variance for each control phrase in json format
    try:
        with open(FILE_STORE, "r") as file:
            try:
                return load(file)
            except JSONDecodeError:
                logger.error(ERR_EMPTY_FILE)
                return dict()

    except FileNotFoundError:
        logger.error(ERR_FILE_NOT_FOUND)
        return dict()


def setKeyStore(keystore: dict) -> None:
    #save keystore to file
    with open(FILE_STORE, "w") as file:
        dump(keystore, file)


def appRoutine(mode: str, frase: str) -> None:
    #main routine of the application
```

```
    logger.debug(f"Starting application in {mode} mode.")

    #load keystore
    keystore = getKeyStore()

    #if control phrase not found in keystore, we can't run prod mode, return
    if mode == MODE_PRD:
        if frase not in keystore:
            logger.error(ERR_INVALID_PASSPHRASE)
            return


    #if control phrase not provided, use first key in keystore
    if not frase:
        logger.error(ERR_PASSPHRASE_NOT_FOUND)
        try:
            frase = keystore.keys()[0]
        except KeyError:
            logger.error(ERR_EMPTY_PASSPHRASE)
            #if keystore is empty, start with "default"
            frase = "default"

    logger.info(f"Control phrase: {frase}")

    #intialize collector and collect key press times
    collector = Collector(COLLECTION_ATTEMPTS, frase)
    key_press_times = collector.collect()

    logger.debug("Key press times collected")
    logger.debug(f"Key press times: {key_press_times}")

    #calculate mean and variance of key press times
    mt, vt = getStatistics(key_press_times, len(key_press_times[0]))

    logger.debug("Statistics calculated")
    logger.debug(f"Mean: {mt}")
    logger.debug(f"Variance: {vt}")

    #if learning mode, save mean and variance to keystore and exit
    if mode == MODE_LRN:
        keystore[frase] = {"mean": mt, "variance": vt}
        setKeyStore(keystore)
        logger.info("Keystore saved")
        return

    #if production mode, load mean and variance from keystore and check
```

```
identification
    mt_stored, vt_stored = keystore[frase]["mean"], keystore[frase]
["variance"]


    res = checkIdentification(key_press_times, (mt_stored, vt_stored))

    #print identification result

    if res:
        logger.info("User identified, access granted.")
    else:
        logger.error(ERR_USER_DEN)

def main() -> None:
    #parse arguments to determine mode
    parser = argparse.ArgumentParser(description="CGW program")
    parser.add_argument("-v", "--verbose", action="store_true",
help="increase output verbosity")
    parser.add_argument("-l", "--learning", action="store_true",
help="learning mode")
    parser.add_argument("-p", "--production", action="store_true",
help="production mode")
    parser.add_argument("-f", "--frase", type=str, help="control phrase")
    args = parser.parse_args()

    #verbose mode enabled
    if args.verbose:
        logger.setLevel(logging.DEBUG)
        logging.getLogger("COLLECTOR LOGGER").setLevel(logging.DEBUG)
        logging.getLogger("CALC LOGGER").setLevel(logging.DEBUG)

    #determine mode
    mode = MODE_PRD if args.production else MODE_LRN

    #launch main routine
    appRoutine(mode, args.frase)


if __name__ == "__main__":
    main()
```

Коментарі в коді.

constants.py

```
#define modes
MODE_LRN = "Learning"           #identifier for learning mode
MODE_PRD = "Prodcution"         #identifier for production mode

#define keystore
FILE_STORE = "keystore.json"    #file to store key press times

#define error messages

ERR_FILE_NOT_FOUND = "DB File not found."
#error message occurs when file not found
ERR_EMPTY_FILE = "DB File is empty. Initializing empty keystore."
#error message occurs when file is empty
ERR_INVALID_PASSPHRASE = "Invalid passphrase, please try again."
#error message occurs when passphrase is invalid
ERR_PASSPHRASE_NOT_FOUND = "Passphrase not provided. Trying to read from
file." #error message occurs when passphrase is not provided
ERR_EMPTY_PASSPHRASE = "Empty passphrase, starting with 'default'."
#error message occurs when passphrase is empty
ERR_USER_DEN = "User not identified, access denied."
#error message occurs when user did not pass identification

#constants
COLLECTION_ATTEMPTS = 5         #number of attempts to collect key press
times
SIG_LEVEL = 0.05                #significance level for hypothesis testing,
basicly inferential statistic
```

Коментарі в коді.

`collector.py`

```
from constants import *
from typing import Optional
import logging
import keyboard

logger = logging.getLogger("COLLECTOR LOGGER")
logger.setLevel(logging.INFO)

class Collector:
    """
    Class to collect key press timings from user
    """
```

```python
    def __init__(self, collection_attempts:int, control_phrase:
Optional[str] = None):
        """
        Initialize the collector
        :param collection_attempts: Number of collection attempts
        :param control_phrase: Control phrase to be used for verification
        """
        self.control_phrase = control_phrase
        self.collection_attempts = collection_attempts
        self.key_press_times = []

    def prompt(self) -> tuple[str, list[float]]:
        """
        Prompt the user to enter the control phrase
        :return: Tuple of control phrase and list of timings
        """

        logger.debug("Prompting user")
        logger.info("Enter passphrase: ")

        #start recording keyboard events
        keyboard.start_recording()
        inp_phrase = input("> ")
        events = keyboard.stop_recording()
        #stop recording keyboard events

        logger.debug(f"User entered events: {events}")
        logger.debug("Recording stopped")

        #parse the events to get the timings
        #initialize unacked dict to keep track of unacknowledged key presses
        unacked = {i: c for i, c in enumerate(self.control_phrase)}
        timings_l = [0.0] * len(self.control_phrase)
        key_down_time = {}



        for event in events:
            if isinstance(event, keyboard.KeyboardEvent):
                #checl if the event is a key press
                if event.event_type == keyboard.KEY_DOWN:
                    #if key pressed down, we save it's time
                    #we also save keyname here, because it is possible
                    #that next key will be released before this one
                    key_down_time[event.name] = event.time
                elif event.event_type == keyboard.KEY_UP:
```

```
                    #if key is released, we calculate the time it was
pressed
                    try:
                        #check if this key was pressed during this session
                        if key_down_time[event.name]:
                            logger.debug(f"Key {event.name} found. Was
pressed for {event.time - key_down_time[event.name]} seconds")

                            #calculate the time key was pressed
                            timing = event.time - key_down_time[event.name]

                            #clear it from the dict
                            del key_down_time[event.name]

                            #check if this key is part of the control phrase
                            for key, char in unacked.items():
                                if char == event.name:
                                    timings_l[key] = timing
                                    del unacked[key]
                                    break
                    except KeyError:
                        continue

                #sometimes user can press enter before releasing last key of
the phrase
                #in that case, we need to check if any key of phrase is
still unacked
                #and calculate the time it was pressed with time of last
button (always enter)
                if event.name == "enter" and event.event_type ==
keyboard.KEY_DOWN and unacked:
                    for key in unacked:
                        logger.debug(f"Key {unacked[key]} UP was not
recorded.")
                        if unacked[key] in key_down_time: timings_l[key] =
event.time - key_down_time[unacked[key]]

                    #clear the unacked dict
                    key_down_time, unacked = {}, {}

        logger.debug(f"Timings: {timings_l}")

        return (inp_phrase, timings_l)


    def collect(self) -> list[list[float]]:
```

```
        logger.debug("Starting collector")
        att = 0
        while att < self.collection_attempts:
            logger.info(f"Attempt {att + 1}")
            #prompt the user to enter the control phrase
            phrase, timings_l = self.prompt()

            #we want to collect info only for valid control phrase
            if phrase != self.control_phrase:
                #skip if control phrase is invalid
                logger.error(ERR_INVALID_PASSPHRASE)
                continue

            #if control phrase is valid, save the timings
            self.key_press_times += [timings_l]
            att += 1

        return self.key_press_times
```

Додаткові пояснення:
Вимагає запуску програми в адміністративному моді, для зчитування подій клавіатури.

- Функція `collect` починає цикл збору часових інтервалів для зазначеної контрольної фрази, перевіряє чи фраза введена правильно, та повертає масив масивів з часовими відмітками.
- Функція `prompt` запитує у користувача ввід контрольної фрази, зчитує події клавіатури, та обраховує час натиснення кожної клавіші для контрольної фрази.

`calc.py`

```
from scipy.stats import t, f, f_oneway, ttest_ind
from math import sqrt
from constants import *
import logging

logger = logging.getLogger("CALC LOGGER")
logger.setLevel(logging.INFO)

def getStatistics(key_press_times:list, ph_len: int) -> tuple[list[float],
list[float]]:
    """
    Get statistics for key press times
    :param key_press_times: List of key press times
    :param ph_len: Length of the passphrase
```

```
    :return: Tuple of mean and variance of key press times

    in general data would look like this:
    [
        [0.1799740791, 0.1698138713, 0.1883199214, 0.1741979122, ....,
0.1050920486], - timimngs for letters in first attempt
        [0.1711528301, 0.1699109077, 0.1331260204, 0.1427738666, ....,
0.1071650981], - timings for letters in second attempt

.....................................................................
,
        [0.1162168979, 0.1681020259, 0.1548230648, 0.1233329772, ....,
0.1316030025], - timings for letters in N attempt
    ]         ^             ^              ^            ^
^
        for letter 1   for letter 2  for letter 3  for letter 4        for
letter M

    """

    #initialize mean and variance lists
    mt, vt = [0.0] * ph_len, [0.0] * ph_len

    #calculate mean and variance for each letter
    for i in range(ph_len):
        #iterate through phrase length
        lc = 0
        for j in range(len(key_press_times)):
            #iterate through collection attempts

            if key_press_times[j][i] != -1:
                #ignore insuficient data
                mt[i] += key_press_times[j][i]
                lc += 1
        #mean
        mt[i] /= lc



    for i in range(ph_len):
        #iterate through phrase length
        lc = 0
        for j in range(len(key_press_times)):
            #iterate through collection attempts
            if key_press_times[j][i] != -1:
                #ignore insuficient data
```

```
                vt[i] += pow(key_press_times[j][i] - mt[i], 2)
                lc += 1
        #variance
        vt[i] /= lc

    #return tuple of mean and variance
    return (mt, vt)


def sieve(item: list) -> list:
    """
    Sieve the data for outliers
    :param item: List of key press times
    :return: List of key press times with outliers marked as -1
    """
    res = list(item)
    for i in range(len(res)):
        item_ = item[:i] + item[i + 1:]
        mean = sum(item_) / len(item_)
        Si = sqrt(sum(pow(x - mean, 2) for x in item_) / (len(item_) - 1))
        tp = abs((item[i] - mean) / Si)
        tT = t.ppf(1 - SIG_LEVEL / 2, df = len(item_) - 1)
        if tp > tT:
            res[i] = -1
    return res

def checkIdentification(user_stats: list[list[float]], stored_stats:
tuple[list[float], list[float]]) -> bool:
    """
    Check identification of user
    :param user_stats: List of key press times for user
    :param stored_stats: Tuple of mean and variance of key press times
stored in keystore
    :return: True if user is identified, False otherwise
    """

    #get mean and variance for stored user stats, get length of passphrase
    mt_stored, vt_stored = stored_stats
    ph_len = len(mt_stored)

    logger.debug(f"Stored stats: {mt_stored = }, {vt_stored = }")
    logger.debug(f"User stats: {user_stats}")

    #sieve the data for outliers
    upd_user_stats = [[0.0]*ph_len for i in range(COLLECTION_ATTEMPTS)]
    for i in range(ph_len):
```

```
        tmp = sieve([j[i] for j in user_stats])
        for j in range(COLLECTION_ATTEMPTS):
            upd_user_stats[j][i] = tmp[j]


    logger.debug(f"Updated user stats: {upd_user_stats}")

    #calculate degrees of freedom
    df = COLLECTION_ATTEMPTS - 1

    #calculate critical values
    #table value for t-test,

    # H0: there is no statisticly significant difference between user and
stored stats
    tT = t.ppf(1 - SIG_LEVEL / 2, df = df*2) #two tailed, two-sampled t-test

    #short explanation:
    # if we are testing whether the means of two samples are different,
    # a two-tailed test will consider both the possibility that the first
mean
    # is greater than the second mean and the possibility that the first
mean
    # is less than the second mean. We use one-sample t-test determine
whether
    # the mean of a single sample is significantly different from a known
mean



    #table values for f-test
    f_critical_low = f.ppf(SIG_LEVEL / 2, df, df)
    f_critical_high = f.ppf(1 - SIG_LEVEL / 2, df, df)


    logger.info(f"Table values: {tT = :.6f}, {f_critical_low = :.6f},
{f_critical_high = :.6f}")

    #get fresh statistics for user
    mt_user, vt_user = getStatistics(upd_user_stats, ph_len)

    logger.debug(f"User stats: {mt_user = }, {vt_user = }")
    logger.debug(f"Stored stats: {mt_stored = }, {vt_stored = }")

    #perform f-test and t-test for each letter (bcs I am honestly not sure
how to do it for the whole passphrase)
    for letter in range(ph_len):
```

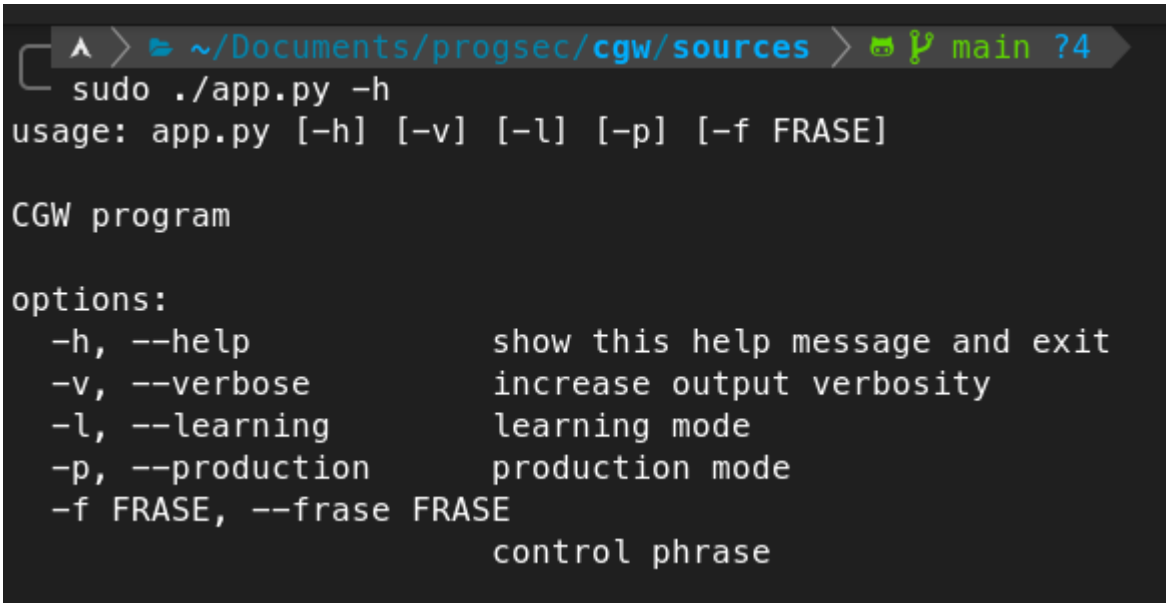
```
        logger.info(f"Checking letter {letter + 1}")
        logger.debug(f"S1 {max(vt_user[letter], vt_stored[letter]):.6f}")
        logger.debug(f"S2 {min(vt_user[letter], vt_stored[letter]):.6f}")

        #f-test
        # H0: there is no statisticly significant difference between user
and stored stats (comparing variances)
        fp = vt_user[letter] / vt_stored[letter] if vt_user[letter] >
vt_stored[letter] else vt_stored[letter] / vt_user[letter]

        #p-value
        p_value = 2 * min(f.cdf(fp, df, df), 1 - f.cdf(fp, df, df))
        logger.info(f"P-value: {p_value:.6f}")
        logger.info(f"F-test Statistic: {fp:.6f}")
        if fp < f_critical_low or fp > f_critical_high:
            #if f-test fails, we can't perform t-test
            logger.info(f"F-test failed for letter {letter + 1}")
            return False

        #t-test
        # H0: there is no statisticly significant difference between user
and stored stats (comparing means)
        tp = abs((mt_user[letter] - mt_stored[letter]) /
sqrt(vt_user[letter] / COLLECTION_ATTEMPTS + vt_stored[letter] /
COLLECTION_ATTEMPTS))
        logger.info(f"t-test Statistic: {tp:.6f}")

        if tp > tT:
            #if t-test fails, we can't identify user
            logger.info(f"t-test failed for letter {letter + 1}")
            return False

    #if all tests pass for each letter, user is identified
    return True
```

Додаткові пояснення:

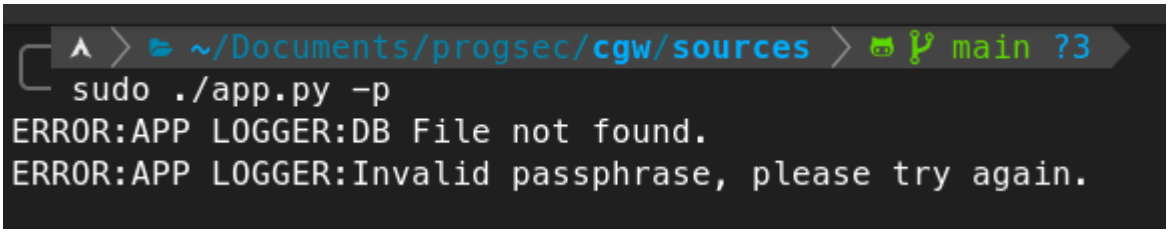
- функція `getStatistics` - приймає на вхід масив масивів з часом натиснення клавіш, повертає масив середніх значень та дисперсій для кожної літери.
- функція `sieve` - просіює значення прибираючи аномальні, можна помітити що вона замінює їх числом `-1` , у відповідній функції `getStatistics` врахований варіант отримання на вхід числа `-1` що просто ігнорує такий переданий показник.
- функція `checkIdentification`, приймає на вхід масив масивів з часовими відмітками користувача, та збережені статистичні значення. Саме тут відбувається перевірка двох тестів зазначених в завданні.

Робота програми

Розпочнемо роботу з написаною програмою, перевіряючи всі крайні випадки її використання.
Довідка по роботі:



Спробуємо запустити в режимі ідентифікації без існуючого файлу з параметрами.



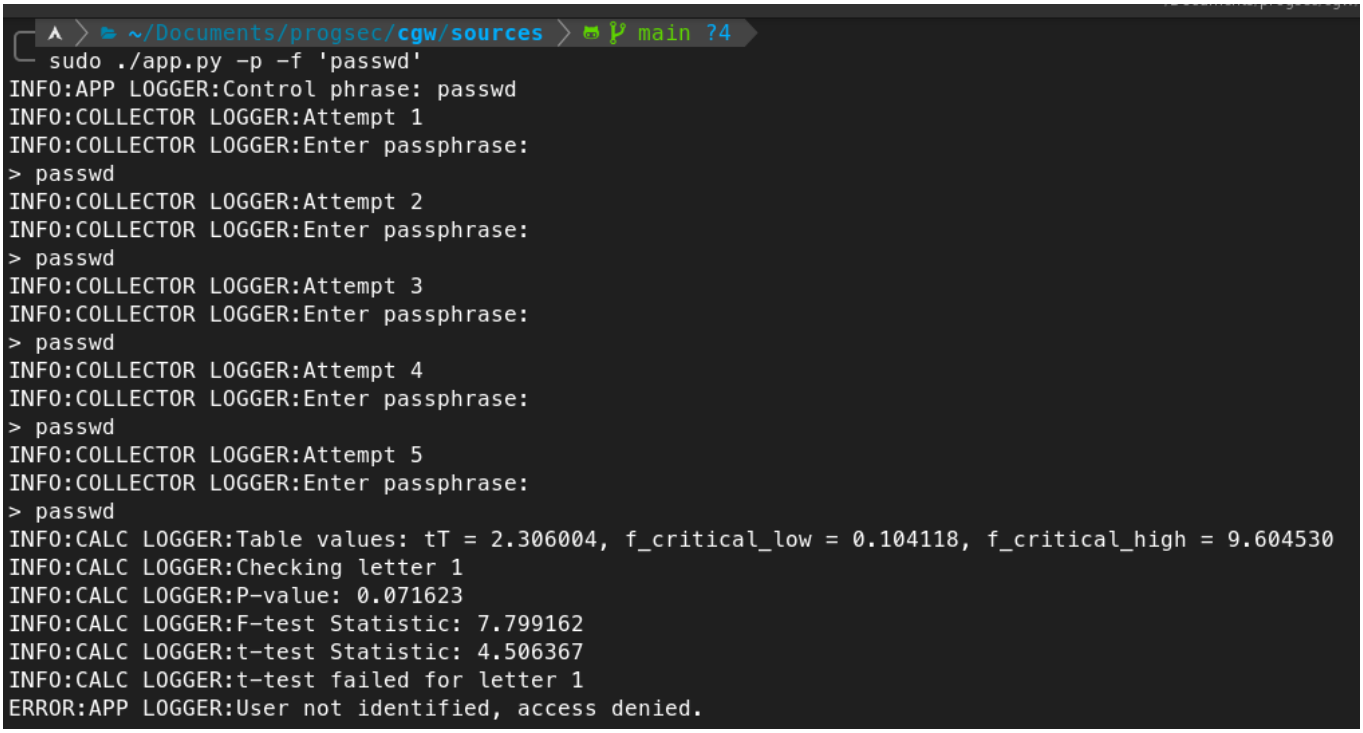
Запустимо в режимі навчання передавши коротку фразу `passwd`. Введемо її необхідну кількість раз.

```
~/Documents/progsec/cgw/sources main ?4
sudo ./app.py -l -f 'passwd'
INFO:APP LOGGER:Control phrase: passwd
INFO:COLLECTOR LOGGER:Attempt 1
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 2
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 3
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 4
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 5
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:APP LOGGER:Keystore saved

~/Documents/progsec/cgw/sources main ?4
ls -la keystore.json
-rw-r--r-- 1 root root 304 Jun  8 16:32 keystore.json

~/Documents/progsec/cgw/sources main ?4
cat keystore.json | jq
{
  "passwd": {
    "mean": [
      0.10249314308166504,
      0.13306040763854982,
      0.11058001518249512,
      0.08934054374694825,
      0.12719178199768066,
      0.09036259651184082
    ],
    "variance": [
      0.00007137923077152664,
      0.00003439183431964920 5,
      0.00020432503104984788,
      0.00019464899520698964,
      0.00009601377284980117,
      0.00023019382791517276
    ]
  }
}
```

Тепер спробуємо перейти до режиму ідентифікації, та ввести контрольну фразу так само як ми її вводили при навчанні (мова йде про стиль друку).

```
~/Documents/progsec/cgw/sources main ?4
sudo ./app.py -p -f 'passwd'
INFO:APP LOGGER:Control phrase: passwd
INFO:COLLECTOR LOGGER:Attempt 1
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 2
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 3
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 4
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:COLLECTOR LOGGER:Attempt 5
INFO:COLLECTOR LOGGER:Enter passphrase:
> passwd
INFO:CALC LOGGER:Table values: tT = 2.306004, f_critical_low = 0.104118, f_critical_high = 9.604530
INFO:CALC LOGGER:Checking letter 1
INFO:CALC LOGGER:P-value: 0.491104
INFO:CALC LOGGER:F-test Statistic: 2.096258
INFO:CALC LOGGER:t-test Statistic: 2.286842
INFO:CALC LOGGER:Checking letter 2
INFO:CALC LOGGER:P-value: 0.322967
INFO:CALC LOGGER:F-test Statistic: 2.927366
INFO:CALC LOGGER:t-test Statistic: 1.043913
INFO:CALC LOGGER:Checking letter 3
INFO:CALC LOGGER:P-value: 0.111676
INFO:CALC LOGGER:F-test Statistic: 5.970553
INFO:CALC LOGGER:t-test Statistic: 0.923230
INFO:CALC LOGGER:Checking letter 4
INFO:CALC LOGGER:P-value: 0.726467
INFO:CALC LOGGER:F-test Statistic: 1.452324
INFO:CALC LOGGER:t-test Statistic: 0.581756
INFO:CALC LOGGER:Checking letter 5
INFO:CALC LOGGER:P-value: 0.788509
INFO:CALC LOGGER:F-test Statistic: 1.330860
INFO:CALC LOGGER:t-test Statistic: 0.160844
INFO:CALC LOGGER:Checking letter 6
INFO:CALC LOGGER:P-value: 0.198171
INFO:CALC LOGGER:F-test Statistic: 4.132668
INFO:CALC LOGGER:t-test Statistic: 1.175724
INFO:APP LOGGER:User identified, access granted.
```

Як бачимо доступ дозволено, перевірка пройдена успішно. Тепер спробуємо надрукувати, навмисно змінивши стиль друку, роблячи паузи і затримки на певних літерах.



Бачимо що на першій же літері t-test виявив розбіжність, що призвело до заборони доступу на основі не проходження ідентифікації.

Розрахунок помилок 1-го та 2-го роду.

Нехай кількість загальних спроб $N_0 = 10$
Спробуємо пройти перевірку 10 раз, з власним клавіатурним почерком.
Отриманий результат 6 ідентифікацій, 4 заборони, отже кількість невдалих спроб $N_1 = 4$

Тоді помилка першого роду $P_1 = \frac{N_1}{N_0} = \frac{4}{10} = 0.4$

При проведенні 10 експериментів за нелегітимного користувача, всі 10 раз нелегітимний користувач не зміг ввійти в систему. Таким чином кількість вдалих спроб
$N_2 = 0$
Тоді помилка другого роду $P_2 = \frac{N_2}{N_0} = \frac{0}{10} = 0$

Висновки

В результаті виконання роботи я розібрався в застосуваннях t-тесту та F-тесту, навчився використовувати їх на практиці. Був розроблений CLI застосунок, що дозволяє перевірити клавіатурний почерк користувача, по збереженим даним. Застосунок має доволі велику помилку першого роду, що свідчить про те що варто спробувати змінити значення α, що за замовчуванням рівне `0.05`.