Szyfrowanie danych cz. 1 - szyfry i ich łamanie

Cel ćwiczenia

Celem ćwiczenia jest poznanie różnych klasycznych technik szyfrowania oraz własności algorytmów szyfrowania. Dodatkowo na zajęciach będzie można zapoznać się z pojęciem entropii i zbadać jej własności.

Materiały powstały w oparciu o książkę Al Sweigart **„Złam ten kod z Pythonem. Jak tworzyć, testować i łamać szyfry**", Helion 2021.

Przebieg ćwiczenia

Entropia

Entropia w teorii informacji jest to średnia ilość informacji niesiona przez określoną wiadomość pochodzącą ze źródła wiadomości. Im większa entropia tym prawdopodobieństwo wystąpienia danych informacji w wiadomości jest bardziej równomierne.

Entropia w teorii informacji zapisuje się jako entropię Shannona:

$$H(X) = -\sum_{i=1}^{n} p(i) log_r p(i),$$

gdzie:
- n to liczba wszystkich zdarzeń w przestrzeni
- $p(i)$ to prawdopodobieństwo zajścia zdarzenia i
- r to podstawa logarytmu (np. 2 dla bitów)

Entropię można interpretować jako niepewność wystąpienia danego zdarzenia w następnej chwili. Jeżeli określone zdarzenie wystąpi z prawdopodobieństwem równym 1 to wtedy entropia będzie wynosiła 0 – z góry wiemy co się stanie.

Entropia metryczna – jest to entropia w teorii informacji znormalizowana przez długość wiadomości. Wyraża się ją następująco:

$$H(X) = -\frac{1}{L}\sum_{i=1}^{n} p(i) log_r p(i),$$

gdzie:
- L to długość wiadomości
- n to liczba wszystkich zdarzeń w przestrzeni
- $p(i)$ to prawdopodobieństwo zajścia zdarzenia i
- r to podstawa logarytmu (np. 2 dla bitów)

Entropia metryczna **spada do zera** przy jednolitym ciągu znaków w wiadomości i **rośnie** monotonicznie wraz **ze wzrostem różnorodności** wiadomości osiągając wartość 1 dla wiadomości maksymalnie zróżnicowanej.

Entropia ważna jest w kryptografii ze względu na bezpieczeństwo generowanych kluczy kryptograficznych. Im bardziej są przewidywalne tym łatwiej jest złamać szyfr.

Zadania

Zadanie 1. Badanie entropii

- Przygotuj środowisko programistyczne, np. Visual Studio Code, PyCharm, IDEA dla języka Python.
- Napisz poniższy kod:

```
import random

random.seed(42)
numbers = []
for i in range(30):
    numbers.append(random.randint( a: 0,  b: 1))
print(numbers)
```

Generuje on zbiór 30. liczb pseudolosowych. Zapoznaj się z funkcją `seed`. Dlaczego za każdym uruchomieniem programu wyświetla się identyczny zbiór wartości? Czy po zmianie ziarna (seed) zmienia się wynikowy zbiór?

- Uruchom teraz następujący kod:

```
random.seed(42)
numbers = []
for i in range(30):
    numbers.append(random.SystemRandom().randint( a: 0,  b: 1))
print(numbers)
```

Czemu po użyciu funkcji `SystemRandom().randint()` generowane są zawsze inne wartości? W jaki sposób generowane są wartości przez funkcję `SystemRandom()`?

- Napisz kod, który obliczać będzie entropię dla ciągu bitów 101111011:

```
def entropia(bity):  1 usage
    wartosci, licznosci = np.unique(bity, return_counts=True)
    print(wartosci, licznosci)

    prawdopodobienstwa = licznosci / licznosci.sum()
    print(prawdopodobienstwa)

    entropy = -np.sum(prawdopodobienstwa * np.log2(prawdopodobienstwa))
    return entropy


ciag_bitow = [1, 0, 1, 1, 1, 1, 0, 1, 1]
print("Entropia: ", "{0:.5f}".format(entropia(ciag_bitow)))
```

Dodatkowo zaimportuj bibliotekę numpy:

```
import numpy as np
```

Po uruchomieniu powinieneś zobaczyć następujący wynik:

```
[0 1] [2 7]
[0.22222222 0.77777778]
Entropia:  0.76420
```

Porównaj ten wynik z wynikiem otrzymanym na witrynie pod adresem: Shannon Entropy CalculatorI.

- Przeprowadź test i wykaż jakie wartości entropii będą otrzymywane dla zbioru losowo wygenerowanych 1000 bitów za pomocą funkcji `SystemRandom().randint()` oraz `randint()`. Czy wartości te różnią się? Jakie wnioski można wysunąć z tej obserwacji?

  Teraz samodzielnie napisz kod, który obliczy entropię metryczną. Wygeneruj wartość entropii dla ciągów „abc123”, „abccba”, „Klucz kryptograficzny”. Który z tych ciągów ma najwyższą entropię metryczną? Porównaj wyniki z wynikami otrzymanymi na witrynie pod adresem: Shannon Entropy Calculator.

Zadanie 2. Szyfr Cezara

- Napisz następujący kod reprezentujący szyfr Cezara:

```python
# Każdy możliwy symbol, który można zaszyfrować:
SYMBOLE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?.'

# Ciąg znaków do zaszyfrowania/odszyfrowania:
wiadomosc = 'To jest moja wiadomosc'

# Klucz szyfrowania/odszyfrowania:
klucz = 7

# Tryb pracy:
tryb = 'deszyfruj'

# Miejsce przechowania zaszyfrowanej/odszyfrowanej wiadomości:
ciag = ''

for symbol in wiadomosc:
    # Uwaga: Tylko symbole w ciągu SYMBOLE mogą być szyfrowane/odszyfrowane
    if symbol in SYMBOLE:

        symbolIndex = SYMBOLE.find(symbol)

        if tryb == 'szyfruj':
            ciagIndex = symbolIndex + klucz
        elif tryb == 'deszyfruj':
            ciagIndex = symbolIndex - klucz

        # W razie potrzeby obsłuż wraparound:
        if ciagIndex >= len(SYMBOLE):
            ciagIndex = ciagIndex - len(SYMBOLE)
        elif ciagIndex < 0:
            ciagIndex = ciagIndex + len(SYMBOLE)

        ciag = ciag + SYMBOLE[ciagIndex]
    else:
        ciag = ciag + symbol

# Wyświetla przetłumaczony ciąg:
print("Otrzymany ciąg: ",ciag)
```

- Za pomocą tego programu zaszyfruj wiadomości z wykorzystaniem następujących kluczy:
  - „1234567890”, klucz 21,
  - „Alicja”, klucz 5,
  - „Szyfrowanie szyfrem Cezara”, klucz 8.

- Za pomocą tego programu deszyfruj wiadomości z wykorzystaniem następujących kluczy:
  - „j9Az8uQ38z9!7uw4u”, klucz 20,
  - „b98o1FLn9j1j”, klucz 9,
  - „j5Mzu90M0u190Mzq?4A”, klucz 16.

- Teraz samodzielnie napisz kod, który **złamie szyfr Cezara metodą siłową** (brute force). Wykorzystaj poniższe konstrukcje:

  - Zainicjalizuj dwie zmienne:

```python
wiadomosc = 'guv6Jv6Jz!J6rp5r7Jzr66ntrM'
SYMBOLE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?.'
```

  - Iteruj po każdym kluczu, który mógł wystąpić:

```
for klucz in range(len(SYMBOLE)):
```

- Iteruj po każdym symbolu w wiadomości i odnajdź indeks danego znaku:

```
for symbol in wiadomosc:
    if symbol in SYMBOLE:
        symbolIndex = SYMBOLE.find(symbol)
```

Pamiętaj o owinięciu (jeżeli dotrzemy do końca tablicy).

- Wyświetl wszystkie możliwe ciągi i odnajdź tekst zdający się jako czytelny dla szyfrogramu podanego w zmiennej `wiadomosc`:

```
# Wyświetl wszystkie możliwe zdeszyfrowane ciągi:
print('Klucz #%s: %s' % (klucz, ciag))
```

Zadanie 3. Szyfr przestawieniowy

- Napisz następujący kod reprezentujący szyfr przestawieniowy (kolumnowy):

```python
def main():
    mojaWiadomosc = 'Zdrowy rozsądek nie jest aż tak powszechny.'
    mojKlucz = 4

    szyfrogram = szyfrujWiadomosc(mojKlucz, mojaWiadomosc)

    # Wyświetl tekst jawny i zaszyfrowany ciąg na ekranie, ze znakiem | ("potok") po nim,
    # jeśli na końcu zaszyfrowanej wiadomości znajdują się spacje:
    print("Tekst jawny:", mojaWiadomosc)
    print("Szyfrogram:", szyfrogram, '|')


def szyfrujWiadomosc(klucz, wiadomosc):  1 usage
    # Każdy ciąg w szyfrogramie reprezentuje kolumnę w siatce:
    szyfrogram = [''] * klucz

    # Pętla przez każdą kolumnę w tekście zaszyfrowanym:
    for kolumna in range(klucz):
        biezacyIndex = kolumna

        # Kontynuuj pętlę, aż biezacyIndex przekroczy długość wiadomości:
        while biezacyIndex < len(wiadomosc):
            # Umieść znak w bieżącym indeksie w wiadomości na
            # końcu bieżącej kolumny na liście szyfrogramów:
            szyfrogram[kolumna] += wiadomosc[biezacyIndex]


            biezacyIndex += klucz

    # Konwertuj listę szyfrogramu na pojedynczą wartość ciągu i zwróć ją:
    return ''.join(szyfrogram)

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

- Za pomocą tego programu zaszyfruj wiadomości z wykorzystaniem następujących kluczy:
  - „To jest kolumna”, klucz 14,
  - „Algorytm AES”, klucz 7,
  - „Prosty szyfr przestawieniowy”, klucz 2.

- Przeanalizuj kod funkcji `szyfrujWiadomosc` i napisz własną funkcję o nazwie `deszyfrujWiadomosc`, która będzie deszyfrowała podane przez użytkownika szyfrogramy. Pamiętaj o podaniu klucza.

- Teraz za pomocą napisanej funkcji przeprowadź deszyfrowanie poniższych wiadomości z wykorzystaniem następujących kluczy:
  - „BeyKoeńsowzstmyptepciwmuheoótc wezS r”, klucz 8,
  - „Znp 1aoros z zsekyzslwytuffacrrwzoenuwmy a m1”, klucz 11,
  - „Gtu agćcn rujunowzi!alęde iea”, klucz 3.