Kryptografia i kryptoanaliza

## Laboratorium 3

Michał Łaskawski

# Zadanie 1

Zrealizować program implementujący podstawieniowy algorytm szyfrowania.

1. Wybrać dłuższy fragment tekstu w języku angielskim (np. akapit składający się z kilkunastu zdań).
2. Usunąć z niego wszystkie znaki niebędące literami (ograniczenie do 26 liter alfabetu łacińskiego).
3. Zaszyfrować tekst używając wybranego w sposób losowy klucza (tablicy podstawień): permutacji πˆ.

# Zadanie 2

Dokonać kryptoanalizy heurystycznej na zaimplementowany w ramach pierwszego zadania monoalfabetyczny kryptosystem podstawieniowy. Założenia ataku są następujące:

1. Znany jest szyfrogram.
2. Wiadomo jaki kryptosystem użyty został do zaszyfrowania wiadomości.
3. Należy odzyskać klucz i tekst jawny.

Do realizacji zadania kryptoanalizy heurystycznej, należy użyć algorytmu *Metropolis-Hastings*. Algorytm ten umożliwia realizację procedury poszukiwanie klucza (czyli mapowania znaków), który najlepiej dopasowuje się do rozkładu prawdopodobieństwa tekstu jawnego. Przebieg tego procesu wygląda następująco:

1. Wybieramy reprezentację klucza jako permutację πˆ:
   * Zakładamy losowy klucz początkowy, czyli permutację znaków klucza.
2. Definiujemy funkcję oceny klucza (funkcja celu / funkcja wiarygodności) pl(πˆ):
   * Funkcja ta jest miarą tego jak bardzo odszyfrowany tekst przypomina tekst naturalny.

Q

* + Na potrzeby zadania, funkcja ta zdefiniowana jest w następujący sposób: pl (πˆ) =

gdzie:

*i,j*

(M*i,j*

)*M*ˆ*i,j* .

* M to macierz bigramów utworzona na bazie tekstu referencyjnego, natomiast M*i,j* to liczba wystąpień pary (i, j) w tekście referencyjnym.
* Mˆ to macierz bigramów utworzona na bazie szyfrogramu, natomiast Mˆ*i,j* to liczba wystąpień

pary (i, j) w szyfrogramie.

* + Uwaga zdefiniowaną funkcję należy rozpatrywać w kategorii prawdopodobieństwa.

1. Losujemy nową permutację klucza πˆ′:
   * Zadanie to realizowane jest poprzez losową zamianę dwóch znaków z permutacji (klucza) πˆ.
2. Definiujemy kryterium akceptacji ρ(πˆ, πˆ′):
   * Algorytm *Metropolis-Hastings* akceptuje nowy klucz z pewnym prawdopodobieństwem, które za-

leży od stosunku funkcji oceny dla nowego i starego klucza. Jeśli nowy klucz πˆ′ prowadzi do

lepszego dopasowania, to akceptujemy go jako X*t*+1, jeśli nie, to zostajemy przy starym πˆ kluczu.

* + Dla rozważanego przypadku, kryterium akceptacji można zdefiniować w następujący sposób:

ρ(πˆ, πˆ′) = pl(*π*ˆ*′*) .

pl(*π*)

* + Dla rozważanego algorytmu, należy wylosować liczbę u należącą do rozkładu jednostajnego na przedziale [0, 1] a następnie dokonać porównania: u ≤ ρ(πˆ, πˆ′). Jeżeli warunek ten jest spełniony to akceptujemy nowy klucz (permutację), jeżeli nie to zostajemy przy starym kluczu.

1. Iteracja procesu:
   * Proces ten jest powtarzany, tworząc łańcuch kluczy {X*t* : t = 0, . . . , T }, które przybliżają opty- malne rozwiązanie.
   * W miarę postępu procesu iteracyjnego, algorytm koncentruje się na obszarach rozwiązań, które lepiej odtwarzają tekst jawny.

Algorytm *Metropolis-Hastings* dla rozważanego problemu, przyjmuje następującą postać:

### Algorithm 1 MH

1: t ← 0

2: X0 ← πˆ0

3: **for** t = 1, . . . , T **do**

4: dla X*t* ← πˆ

5: wygeneruj i, j ∼ U({1, 2, . . . , 26}) ▷ ∼ znaczy ma rozkład

6: wygeneruj πˆ′ ▷ zamieniając znaki na pozycjach i oraz j w kluczu πˆ

7: ρ(πˆ, πˆ′) ← pl(*π*ˆ*′*)

pl(*π*)

8: wygeneruj u ∼ U([0, 1])

9: **if** u ≤ ρ(πˆ, πˆ′) **then**

10: X*t*+1 ← πˆ′

11: **else**

12: X*t*+1 ← πˆ

13: **end if**

▷ ρ - prawdopodobieństwo akceptacji

14: **end for**

### Uwagi:

* Wyznaczenie funkcji pl, może prowadzić do przekroczenia zakresu numerycznego. Aby uniknąć proble- mów związanych z precyzją numeryczną, można zastosować logarytmowanie funkcji pl.

log pl (πˆ) = Mˆ*i,j* · log M*i,j*

Σ

*i,j*

* Uwaga, przekształcenie to w konsekwencji poprowadzi do innego sposobu wyznaczenia współczynnika akceptacji poprzez obliczenie wartości funkcji wykładniczej, której argumentem jest różnica pomiędzy pl(πˆ′) a pl(πˆ).

Zamiast obliczać: ρ(πˆ, πˆ′) = pl(*π*ˆ*′*) należy obliczyć:

pl(*π*)

ρ πˆ, πˆ′ = exp log pl πˆ′ − log pl (πˆ)

Należy również zauważyć, iż współczynnik akceptacji nie powinien być większy do jedności.

* Jest to powszechny sposób stabilizacji obliczeń w algorytmach probabilistycznych.

# Zadanie 3

Dokonać kryptoanalizy heurystycznej na zaimplementowany w ramach pierwszego zadania monoalfabetyczny kryptosystem podstawieniowy. Założenia ataku są takie jak w zadaniu 2. Do ataku wykorzystać algorytm optymalizacji *Symulowanego Wyżarzania* (ang. *Simulated Annealing*).

## Algorytm symulowanego wyżarzania

Symulowane wyżarzanie to metoda optymalizacji inspirowana procesem fizycznym zwanym wyżarzaniem, który jest stosowany w metalurgii i krystalografii. Proces ten polega na podgrzewaniu materiału do wysokiej temperatury, a następnie stopniowym chłodzeniu w celu osiągnięcia stanu minimalnej energii. Algorytm

symulowanego wyżarzania wykorzystuje tę koncepcję, aby znaleźć rozwiązanie problemu optymalizacyjnego, które minimalizuje (lub maksymalizuje) pewną funkcję celu.

### Ogólny opis algorytmu

1. **Inicjalizacja**:
   * Algorytm zaczyna od pewnego początkowego rozwiązania (np. losowego lub opartego na pewnej heurystyce).
   * Ustalana jest początkowa temperatura, która kontroluje prawdopodobieństwo akceptacji gorszych rozwiązań.

### Główna pętla:

* + W każdej iteracji algorytm generuje nowe rozwiązanie poprzez niewielką modyfikację bieżącego rozwiązania (np. zamiana dwóch elementów w mapowaniu liter).
  + Obliczana jest różnica wyników między nowym a aktualnym rozwiązaniem.
    - Jeśli nowe rozwiązanie jest lepsze (ma wyższy wynik), to zostaje zaakceptowane.
    - Jeśli nowe rozwiązanie jest gorsze, to może ono zostać zaakceptowane z pewnym prawdopodo- bieństwem, które zależy od różnicy wyników i aktualnej temperatury. To prawdopodobieństwo jest obliczane jako:

score\_dif f temperature

P = exp

### Schładzanie:

* + Temperatura jest stopniowo zmniejszana zgodnie z określoną regułą chłodzenia (np. mnożenie przez współczynnik chłodzenia, który jest mniejszy od 1). W miarę spadku temperatury algorytm staje się mniej skłonny do akceptowania gorszych rozwiązań.

### Zakończenie:

* + Algorytm kończy się po osiągnięciu określonej liczby iteracji lub po osiągnięciu bardzo niskiej temperatury, kiedy przestaje akceptować gorsze rozwiązania.
  + Ostateczne rozwiązanie jest uważane za przybliżone optimum globalne.

### Opis algorytmu w kontekście ataku na analizowany kryptosystem

Dane wejściowe:

* c - szyfrogram,
* π0 - początkowe mapowanie,
* g - n-gramy referencyjne (bigramy),
* N - maks. iteracje,
* T0 -początkowa temperatura,
* α współczynnik chłodzenia Opis Symboli:
* c: Szyfrogram, który chcemy odszyfrować.
* π: Permutacja, mapowanie liter (klucz szyfrujący).
* g: Referencyjne n-gramy (bigramy).
* Scurr: Aktualna wartość funkcji celu.

### Algorithm 2 SA

1: π ← π0

2: Scurr ← f*c,g* (π)

3: πbest ← π

4: Sbest ← Scurr

5: T ← T0

6: **for** k = 1 to N **do**

7: a ∼ U(A), b ∼ U(A \ {a}) ▷ A = {A, B, . . . , Z}

8: π′ ← σ*a,b*(π)

9: Snew ← f*c,g* (π′)

10: ∆S ← Snew − Scurr

11: **if** ∆S > 0 **or** exp ∆*S* > u dla u ∼ U(0, 1) **then**

*T*

12: π ← π′

13: Scurr ← Snew

14: **if** Scurr > Sbest **then**

15: πbest ← π

16: Sbest ← Scurr

17: **end if**

18: **end if**

19: T ← α × T

20: **end for**

21: **Return:** πbest

* Snew: Nowa wartość funkcji celu po zamianie liter.
* T : Temperatura, która kontroluje prawdopodobieństwo akceptacji gorszych rozwiązań.
* u ∼ U(0, 1): Liczba losowa z rozkładu jednostajnego na przedziale [0, 1].
* σ*a,b*(π): funkcja losowej zamiany miejscami liter w permutacji π.
* f*c,g*(π): Funkcja celu, może one być zdefiniowana w następujący sposób:

gdzie:

f*c,g*

*n*

(π) : ν*i*

Σ

*i*=1

* ϕ*i*

max Φ

* + ν*i*: oznacza częstość i-tego bigramu w analizowanym szyfrogramie c, który odszyfrowany został przy pomocy klucza: π.
  + ϕ*i*: oznacza częstość i-tego bigramu w rozkładzie referencyjnym (bigramy: g).
  + max Φ: to maksymalna wartość w rozkładzie referencyjnym bigramów Φ = {ϕ1, ϕ2, . . . , ϕ*n*}. Należy zwrócić uwagę na to, iż funkcja f*c,g*(π) implementuje algorytm normalizacji (dzielenie przez

max Φ), dzięki temu eliminuje się potencjalny problem z dokładnością numeryczną.

**Uwaga**. W celu poprawy stabilności uzyskiwanych wyników oraz unikania lokalnych maksimów, algorytm symulowanego wyżarzania należy uruchamiać wielokrotnie. Za każdym razem inicjując algorytm inną permu- tacją klucza. Następnie porównywać uzyskany wynik z poprzednio uzyskanym wynikiem i zachowując lepszy. Symbolicznie, podejście to można opisać w następujący sposób:

gdzie:

* Λmax: Najlepszy dotychczasowy wynik funkcji celu, reprezentujący maksymalną wartość oceny spośród wszystkich iteracji.
* Θopt: Najlepsze znalezione rozwiązanie (mapowanie liter - klucz), które maksymalizuje funkcję celu.
* k: Indeks iteracji, liczba całkowita biegnąca od 1 do N, gdzie N to liczba restartów algorytmu.

**Algorithm 3** Optymalizacja z wieloma restartami

1: Λmax ← −∞

2: Θopt ← ∅

3: **for** k ← 1 **to** N **do**

4: π*k* ∼ U(A26) ▷ Losowa permutacja z A26, zbioru 26 liter

5: Ψ*k*, Λ*k* ← Optimize(π*k*)

6: **if** Λ*k* > Λmax **then**

7: Λmax ← Λ*k*

8: Θopt ← Ψ*k*

9: **end if**

10: **end for**

11: **return** Θopt, Λmax

* π*k*: Losowa permutacja 26 liter alfabetu łacińskiego, wybrana z rozkładu jednostajnego A26.
* Ψ*k*: Rozwiązanie wygenerowane w k - tej iteracji, bazujące na permutacji π*k*.
* Λ*k*: Wartość funkcji celu dla rozwiązania Ψ*k*.
* A26: Zbiór wszystkich możliwych permutacji 26-literowego alfabetu.
* U(A26): Rozkład jednostajny na zbiorze A26, z którego losowana jest permutacja π*k*.

# Zadanie 4

Dokonać kryptoanalizy heurystycznej na zaimplementowany w ramach pierwszego zadania monoalfabetyczny kryptosystem podstawieniowy. Założenia ataku są takie jak w zadaniach 2 i 3. Do ataku wykorzystać gene- tyczny algorytm optymalizacji.

## Algorytm genetyczny

Algorytm genetyczny (AG) to metoda optymalizacji inspirowana zasadami ewolucji biologicznej. Wykorzy- stuje pojęcia takie jak selekcja naturalna, krzyżowanie (rekombinacja) oraz mutacja. Jego celem jest znale- zienie rozwiązania problemu, które będzie jak najlepsze według określonej funkcji celu (fitness). Algorytmy genetyczne są często stosowane do rozwiązywania problemów trudnych obliczeniowo, takich jak optymalizacja w przestrzeni wielowymiarowej, planowanie, projektowanie, identyfikacja parametryczna modeli matematycz- nych czy też kryptoanaliza.

### Ogólny opis algorytmu

Podstawowe komponenty algorytmu genetycznego:

* **Populacja**: Zbiór możliwych rozwiązań, zwanych osobnikami lub chromosomami. Populacja ewoluuje z pokolenia na pokolenie.
* **Chromosom**: Reprezentacja kandydata na rozwiązanie. Zwykle jest przedstawiany jako ciąg znaków lub cyfr binarnych, na które zamieniane są inne struktury danych, zależnie od charakteru problemu.
* **Funkcja dopasowania** (fitness function): Funkcja, która ocenia jakość każdego osobnika. Określa, jak dobrze dany chromosom rozwiązuje problem.

Operatory genetyczne:

* **Selekcja** lub inaczej **Reprodukcja**: Proces wybierania najlepszych osobników, które będą rodzicami przyszłych generacji.
  + Popularne metody selekcji obejmują selekcję ruletkową, turniejową lub rankingową.
* **Krzyżowanie** (crossover): Proces łączenia dwóch osobników (rodziców) w celu wygenerowania nowych osobników (potomków).
  + Celem jest wymiana genów między rodzicami, co może prowadzić do tworzenia lepszych rozwiązań.
* **Mutacja**: Proces wprowadzania niewielkich zmian w genotypie osobnika, aby zapewnić różnorodność genetyczną w populacji i uniknąć zbieżności do lokalnych minimów.

Parametry algorytmu:

* **Rozmiar populacji**: m - liczba osobników w każdej generacji (w rozpatrywanym przypadku jest to liczba kluczy).
* **Prawdopodobieństwo krzyżowania**: p*c* - procent rodziców który ulega krzyżowaniu.
* **Prawdopodobieństwo mutacji**: p*m* - prawdopodobieństwo modyfikacji genów (w rozpatrywanym przypadku jest prawdopodobieństwo zmian liter w kluczu).
* **Maksymalne odchylenie standardowe** *funkcji dopasowania*: max s*f* - kryterium zbieżności.
* **Maksymalna liczba generacji**: imax. Ogólny schemat działania algorytmu genetycznego:

### Inicjalizacja:

* + Algorytm zaczyna od stworzenia początkowej populacji m losowych kluczy szyfrujących.
    - Klucz to permutacja alfabetu, która reprezentuje potencjalne rozwiązanie.

### Ewaluacja funkcji dopasowania:

* + Obliczyć wartość *funkcji dopasowania* korzystając funkcji logarytmicznej wiarygodności dla każ- dego klucza.
    - Porównując częstości bigramów w tekście zaszyfrowanym po zdekodowaniu przy użyciu klucza z częstościami referencyjnymi.

### Selekcja:

* + Wykorzystać algorytm *ruletkowej selekcji*, w której osobniki są wybierane z populacji z prawdo- podobieństwem proporcjonalnym do ich wartości funkcji dopasowania.
    - To oznacza, że lepiej dopasowane osobniki mają większą szansę na wybór.
  + Dla każdego osobnika (klucza) π*i* w populacji, prawdopodobieństwo wyboru wynosi:

S (π*i*)

Σ

p*i* = *m*

*j*=1

S (π*j*)

gdzie S (π*i*) jest funkcją wartości dopasowania dla osobnika π*i*.

**Algorithm 4** Selekcja Ruletkowa

**Require:** P = {π1, π2, . . . , π*m*}, S(π*i*)∀π*i* ∈ P

**Ensure:** πselected

1: F ← Σ

*m*

*i*=1

2: p*i* ← *S*(*πi*)

*F*

S(π*i*) ▷ Sumaryczna wartość funkcji dopasowania

∀i ∈ {1, 2, . . . , m} ▷ Prawdopodobieństwa wyboru

3: r ∼ U(0, 1) ▷ Losowa liczba z rozkładu jednostajnego

4: C ← 0 ▷ Suma skumulowana

5: **for** i ← 1 **to** m **do**

6: C ← C + p*i*

7: **if** r ≤ C **then**

8: **return** π*i* ▷ Zwróć wybrany osobnik

9: **end if**

10: **end for**

### Operatory genetyczne:

* + Dokonać operacji *krzyżowania* (crossover):
    - Dwa rodzice są łączeni w celu stworzenia dwóch potomków.
    - Wykorzystać algorytm *krzyżowania jednopunktowego*, co oznacza, że losowo wybierany jest punkt cięcia, po którym geny (litery) są wymieniane między rodzicami.

Rodzice: π1 = [x1, . . . , x*k*, . . . , x26], π2 = [y1, . . . , y*k*, . . . , y26]

Dzieci: π1 = [x1, . . . , x*k*, y*k*+1, . . . , y26], π2 = [y1, . . . , y*k*, x*k*+1, . . . , x26]

**Algorithm 5** Krzyżowanie

**Require:** π1, π2 ▷ Rodzice

**Ensure:** πchild1, πchild2 ▷ Potomkowie

1: k ∼ U{1, . . . , 25} ▷ Losowy punkt cięcia

2: πchild1 ← [π1[1 : k] ∪ π2[k + 1 : 26]] 3: πchild2 ← [π2[1 : k] ∪ π1[k + 1 : 26]] 4: **return** πchild1, πchild2

* + Dokonać operacji *mutacji* :
    - Z niewielkim prawdopodobieństwem zamienić dwa znaki w kluczu miejscami (geny w chro- mosomie).

∗ Zapewnia to różnorodność genetyczną i zapobiega zbieżności do lokalnych minimów.

π = [x1, . . . , x*a*, . . . , x*b*, . . . , x26] ⇒ π′ = [x1, . . . , x*b*, . . . , x*a*, . . . , x26]

**Algorithm 6** Mutacja

**Require:** π ▷ Chromosom

**Ensure:** π′ ▷ Zmutowany chromosom

1: a, b ∼ U{1, . . . , 26}, a ̸= b ▷ Losowe indeksy

2: π′ ← σ*a,b*(π) ▷ Zamiana miejsc elementów π[a] i π[b]

3: **return** π′

### Warunki zakończenia:

* + Algorytm powinien sprawdzać, czy populacja osiągnęła zbieżność, to znaczy czy odchylenie stan- dardowe wartości dopasowania jest mniejsze niż przyjęte *maksymalne odchylenie standardowe*.
  + Jeśli nie osiągnięto zbieżności, to należy kontynuować proces iteracyjny aż osiągnięcia *maksymalnej liczby generacji*

### Zwrócenie najlepszego rozwiązania:

* + Po zakończeniu algorytmu wybierany powinien być klucz, który miał najwyższą wartość funkcji dopasowania.
    - Jest to klucz, który najlepiej odtwarza oryginalny tekst.

### Opis algorytmu w kontekście ataku na analizowany kryptosystem

Dane wejściowe:

* c - szyfrogram.
* g - n-gramy referencyjne (bigramy).
* m - rozmiar populacji.
* p*c* - prawdopodbieństwo krzyżowania.
* p*m* - prawdopodobieństwo mutacji.
* imax - maksymalna liczba generacji.
* s*f* - odchylenie standardowe funkcji dopasowania do wszystkich osobników z populacji P. Opis użytych symboli:
* π - Pojedynczy chromosom (klucz), permutacja liter alfabetu łacińskiego.
* πbest - Najlepsza permutacja (klucz szyfrujący) znaleziona podczas ewolucji.
* S(π) - Wartość funkcji dopasowania dla permutacji π.
* P - Populacja permutacji kluczy szyfrujących.
* arg max - Oznacza wybór klucza z największą wartością funkcji dopasowania.
* smax - Maksymalna dopuszczalna wartość odchylenia standardowego.

*f*

* SRW(P, S) - Selekcja ruletkowa, która wybiera osobnika π z populacji P proporcjonalnie do wartości funkcji dopasowania S(π).
* C - Operator krzyżowania.
* M - Operator mutacji.

*f*

### Algorithm 7 GA

**Require:** c, g, m, p*c*, p*m*, imax, smax

**Ensure:** πbest

1: P ← {π*i* ∼ Perm(A) | i = 1, . . . , m} ▷ Losowe permutacje alfabetu

2: S(π) ← f*c,g*(π) ∀π ∈ P

3: πbest ← arg max*π*∈P S(π)

4: **for** i ← 1 **to** imax **do**

5: s*f* ← q 1 Σ*π*∈P S(π) −  1 Σ*π*∈P S(π) 2 ▷ Oblicz odchylenie standardowe

*m*

*m*

6: **if** s*f* ≤ smax **then**

*f*

7: **break** ▷ Zakończ, jeśli populacja jest zbieżna

8: **end if**

9: P′ ← ∅

10: **for** j ← 1 **to** *m* **do**

2

11: π1 ← SRW(P, S) ▷ Selekcja ruletkowa

12: π2 ← SRW(P \ {π1}, S) ▷ Selekcja ruletkowa bez π1

13: **if** u1 ∼ U(0, 1) < p*c* **then**

14: (πchild1, πchild2) ← C(π1, π2) ▷ Krzyżowanie

15: **else**

16: (πchild1, πchild2) ← (π1, π2)

17: **end if**

18: P′ ← P′ ∪ {πchild1, πchild2}

19: **end for**

20: **for all** π ∈ P′ **do**

21: **if** u2 ∼ U(0, 1) < p*m* **then**

22: π ← M(π) ▷ Mutacja

23: **end if**

24: **end for**

25: S(π) ← f*c,g*(π) ∀π ∈ P′

26: πbest ← arg max (S(πbest), max*π*∈P*′* S(π))

27: P ← P′

28: **end for**

29: **return** πbest

# Zadanie 5

Dokonać analizy pracy zaimplementowanych algorytmów, porównując ich wydajność w ataku na analizowany kryptosystem.