Temat C2. Algorytmy szyfrowania

1. W jakim celu stosujemy szyfry?
2. Podstawowe pojęcia dotyczące szyfrowania
3. Szyfrowanie tekstu metodą podstawieniową na przykładzie szyfru Cezara
4. Szyfrowanie tekstu metodą przestawieniową
5. Programowanie algorytmów szyfrowania w językach C++ i Python
   - 5.1. Miniszyfrowanie – tworzenie anagramów
   - 5.2. Programowanie algorytmu szyfrowania podstawieniowego
   - 5.3. Programowanie algorytmu szyfrowania przestawieniowego
6. Więcej o szyfrowaniu
   - 6.1. Szyfrowanie symetryczne i asymetryczne
   - 6.2. Maszyna Enigma
   - 6.3. Algorytm RSA

Warto powtórzyć

1. W jaki sposób można zadeklarować zmienne tekstowe w języku C++?
2. Jak używamy zmiennych tekstowych w języku Python?
3. W jaki sposób wprowadzamy dane tekstowe w języku C++ lub Python?
4. Jak odwołujemy się do pojedynczego znaku łańcucha w języku C++ lub Python?
5. Która funkcja umożliwia wyznaczenie długości łańcucha w języku C++ lub Python?

1. W jakim celu stosujemy szyfry?

{ Wyobraźmy sobie, że nie mamy dostępu do nowych technologii. Jak przekazać ważną wiadomość, aby jej treść nie trafiła w niepowołane ręce? }

Możemy zapisać wiadomość w taki sposób, aby tylko nadawca i odbiorca mogli ją zrozumieć – wykorzystamy w tym celu szyfr.

Na przykład wiadomość „JUTROIDZIEMYDOKINA" możemy zaszyfrować jako „QFGILRWARVNBWLPRMZ", wykorzystując schemat szyfrowania pokazany w tabeli 1.

S **Szyfr**

Algorytm szyfrowania i deszyfrowania.

A	Y	C	W	E	U	G	S	I	Q	K	O	M
Z	B	X	D	V	F	T	H	R	J	P	L	N

Tabela.1. *Przykład prostego schematu szyfrowania*

Ćwiczenie 1. Szyfrujemy wiadomości

1. Wyjaśnij, na czym polega schemat szyfrowania pokazany w tabeli 1.
2. Zaszyfruj wiadomość do koleżanki lub kolegi. Wykorzystaj schemat szyfrowania z tabeli 1.

Ćwiczenie 2. Deszyfrujemy wiadomość

1. Rozszyfruj wiadomość WLALYZXAVMRZLGIAVXRVQ, wykorzystując tabelę 1.
2. Wyjaśnij sposób rozszyfrowywania wiadomości.

Szyfrowanie S

Proces przetwarzania wiadomości na **szyfrogram**, pozwalający na ukrycie jej tekstu.

Deszyfrowanie D

Proces odtwarzania tekstu oryginalnego na podstawie treści szyfrogramu.

Szyfry stosuje się w celu utajnienia wiadomości, aby nie mogła zostać odczytana przez osobę postronną. Pierwsze szyfry wykorzystywano do przekazywania rozkazów i innych informacji wojennych, a czasami także... listów miłosnych. Z czasem sposoby szyfrowania stawały się coraz bardziej skomplikowane, aby utrudnić złamanie szyfru.

W dzisiejszych czasach szyfry wykorzystuje się zdecydowanie częściej niż kiedyś. Spotykamy się z nimi codziennie – służą na przykład do ochrony wrażliwych danych (dane osobowe, hasła dostępu, dane finansowe). Zupełnie nieświadomie korzystamy z szyfrowania, rozmawiając przez telefon komórkowy, na stronach internetowych czy wysyłając e-mail.

2. Podstawowe pojęcia dotyczące szyfrowania

Szyfrowanie to technika umożliwiająca bezpieczne przesyłanie wiadomości. **Nadawca wiadomości** chciałby mieć pewność, że nikt niepowołany nie pozna jej treści, ani też jej nie zmodyfikuje.

Niezaszyfrowaną wiadomość nazywamy **tekstem jawnym**, natomiast zaszyfrowaną – **szyfrogramem**.

Rys. 1. Schemat procesu szyfrowania i deszyfrowania

Nauka zajmująca się zabezpieczaniem wiadomości to **kryptografia**, natomiast **kryptoanaliza** to nauka o łamaniu tych zabezpieczeń. Razem tworzą one **kryptologię** – osobną dziedzinę matematyki. Szyfrowanie i deszyfrowanie wykonuje się za pomocą odpowiednich funkcji matematycznych, często przy użyciu komputera.

3. Szyfrowanie tekstu metodą podstawieniową na przykładzie szyfru Cezara

Szyfr podstawieniowy to taki, w którym każdy znak tekstu jawnego jest zastępowany innym znakiem. Najbardziej znanym przykładem szyfru podstawieniowego jest **szyfr Cezara**.

W ćwiczeniu 1. przedstawiliśmy algorytm szyfrowania oparty na metodzie podstawieniowej.

Autorstwo szyfru Cezara przypisuje się Juliuszowi Cezarowi. Szyfr ten polega na zastąpieniu każdej litery tekstu jawnego znakiem stojącym w alfabecie o trzy pozycje w prawo względem znaku źródłowego. I tak A zostaje zastąpione przez D, B przez E, ... W przez Z, X przez A, Y przez B i Z przez C. Na rysunku 2. pokazano szyfr Cezara dla alfabetu łacińskiego, który zawiera 26 liter.

Rys. 2. *Szyfr Cezara*

Szyfr Cezara jest szyfrem **przesuwającym**. W tej metodzie szyfrowania można zastosować liczbę inną niż 3, która wskaże, o ile pozycji należy się przesunąć względem znaku wyjściowego. W ten sposób dla tego samego tekstu jawnego możemy uzyskać różne szyfrogramy, w zależności od tego, jaką liczbę zastosujemy (tabela 2.). Aby odszyfrować tekst, odbiorca powinien znać liczbę pełniącą funkcję **klucza** szyfru (rys. 3.).

Tekst jawny	Klucz	Szyfrogram
KOTEK	3	NRWHN
KOTEK	5	PTYJP
KOTEK	13	XBGRX

K **Klucz**

Informacja służąca do szyfrowania lub deszyfrowania tekstu.

Tabela 2. *Przykład wiadomości zaszyfrowanych metodą podstawieniową (szyfr przesuwający) z różnymi kluczami*

Rys. 3. *Schemat procesu szyfrowania i deszyfrowania z kluczem*

Ćwiczenie 3. Szyfrujemy wyraz szyfrem Cezara

1. Zaszyfruj napis MATEMATYKA szyfrem Cezara.
2. Wyjaśnij zastosowany sposób szyfrowywania.

Ćwiczenie 4. Odszyfrowujemy słowa zapisane metodą Cezara

1. Na podstawie wskazówek przedstawionych na rysunku 2. rozszyfruj słowa:
   a. VCBIU
   b. LQIRUPDWBND
   c. NUBSWRJUDILD
2. Wyjaśnij zastosowany sposób rozszyfrowywania.

Ćwiczenie 5. Szyfrujemy zdanie szyfrem Cezara

Korzystając z szyfru Cezara, zaszyfruj zdanie: „Szyfr Cezara to najbardziej znany szyfr podstawieniowy".

4. Szyfrowanie tekstu metodą przestawieniową

Zanim poznamy inny sposób szyfrowania, wykonajmy ćwiczenie 6.

Ćwiczenie 6. Odszyfrowujemy cytaty

1. Przeanalizuj sposób szyfrowania pokazany w tabeli 3. Wyjaśnij, na czym polega ta metoda.
2. Korzystając z przykładów z tabeli 3., odszyfruj cytaty, posługując się podanym kluczem:
   a. NAWOEIWATJEERERMJAJZOŚZSEŻĆYAKNDDMI (klucz 5/7),
   b. NOAZSUIWMETSEOJRTŁCKEAWYHOSZYSSCZZCZŁHCUHĘ (klucz 6/7).

Wskazówka: Autorzy cytatów: a. Heraklit z Efezu, b. Adam Mickiewicz (*Pan Tadeusz*).

Tekst jawny	Klucz	Zapis wierszami	Szyfrogram
ALA MA KOTA	3/3	A L A M A K O T A	AMOLATAKA
SZKOŁA PONADPODSTAWOWA	3/7	S Z K O Ł A P O N A D P O D S T A W O W A	SOSZNTKAAODWŁPOAOWPDA

Tabela 3. *Przykłady wiadomości zaszyfrowanych metodą przestawieniową – ćwiczenie 6.*

W ćwiczeniu 6. posłużyliśmy się szyfrem przestawieniowym, który najprościej zrealizować, zapisując tekst jawny wierszami o ustalonej długości, a czytając go kolumnami (przy ustalonej długości wierszy powstają kolumny o określonej liczbie znaków). Kluczem może być liczba wierszy lub kolumn (w przykładach powyżej podane zostały obie te liczby) – wszystko zależy od ustaleń.

Szyfr przestawieniowy to taki, w którym w tekście zaszyfrowanym pojawiają się wszystkie znaki tekstu jawnego, ale w innej kolejności.

Ćwiczenie 7. Stosowanie szyfru przestawieniowego

Zaszyfruj cytat „To be or not to be – that is the question", stosując szyfrowanie przestawieniowe. Ustaw znaki tekstu jawnego w dziesięciu kolumnach.

Wskazówka: W szyfrowaniu przestawieniowym pomijamy spacje.
Tekst jawny: TOBEORNOTTOBETHATISTHEQUESTION

! Uwaga

„To be or not to be – that is the question"
– W. Szekspir, *Hamlet*, akt III, scena 1.

5. Programowanie algorytmów szyfrowania w językach C++ i Python

5.1. Miniszyfrowanie – tworzenie anagramów

Anagram tworzymy poprzez przestawienie liter lub sylab wyrazu bądź zdania, wykorzystując wszystkie litery (głoski lub sylaby) wyrazu bądź zdania źródłowego.

Najprostszy anagram można utworzyć, układając litery w odwrotnej kolejności, np. *baobab* – *baboab*. Można też przestawić tylko sylaby, co np. w wyrazie *ma·ra*, daje anagram: *ra·ma*.

Możemy w ten sposób utworzyć szyfr, którego działanie będzie polegało na przykład na wyprowadzeniu wyrazu (ciągu znaków) wspak, np. „poezja" – jako „ajzeop".

Ten rodzaj szyfru to szyfrowanie przestawieniowe – polega na zmianie kolejności znaków tekstu.

Ćwiczenie 8. Tworzymy anagramy i szukamy sposobu utworzenia anagramu

1. W kilku grupach twórzcie różne anagramy, np. jedna osoba wymyśla anagram słowa, a pozostałe osoby z grupy próbują odgadnąć, jakie to słowo oraz jaki zastosowano klucz (sposób, w jaki utworzono anagram).
2. Przedstawcie wybrane rozwiązania na forum klasy.

{ Chcemy napisać w wybranym języku program, który tworzy najprostszy anagram, czyli ustawia litery w odwrotnej kolejności. Jak to zrobić? }

Aby napisać program tworzący taki anagram, wystarczy wiedzieć, jak odwołać się do pojedynczego znaku napisu (co omówiliśmy w temacie C1), a także jak użyć instrukcji iteracyjnej do wyświetlania napisu od końca do początku.

Ćwiczenie 9. Piszemy program tworzący anagramy w wybranym języku programowania

1. Przypomnij, w jaki sposób można odwołać się do pojedynczego znaku tekstu w języku C++ lub w języku Python.
2. W wybranym języku programowania napisz program tworzący anagram słowa podanego z klawiatury. Anagram ma być przestawieniem liter w odwrotnej kolejności. Program ma umożliwiać tworzenie anagramów *n* napisów wprowadzanych z klawiatury.
3. Zapisz program w pliku pod nazwą *Miniszyfrowanie*.
4. Przetestuj program dla różnych danych.

Wskazówka: Instrukcja pętli może mieć postać:

C++

```
for(int i = napis.length() - 1; i >= 0; i--)
    cout << napis[i];
```

Python

W języku Python można użyć funkcji wbudowanej `reversed` do iteracji w odwrotnej kolejności po napisie lub liście.

```
for znak in reversed(napis):
    print(znak)
```

5.2. Programowanie algorytmu szyfrowania podstawieniowego

{ Chcemy napisać program realizujący algorytm szyfrowania metodą Cezara.
W jaki sposób napisać w wybranym języku programowania funkcję szyfrującą? }

Aby zaprogramować algorytm, w którym mamy zastąpić daną literę inną literą umiejscowioną w alfabecie o trzy pozycje w prawo względem znaku źródłowego, musimy wiedzieć, jak wyznaczyć tę pozycję litery w alfabecie. Ponadto działania musimy powtórzyć tyle razy, ile wynosi długość tekstu do zaszyfrowania – mamy więc do czynienia z iteracją.

Jak wiemy, każda litera ma swój odpowiednik liczbowy w kodzie ASCII, np. litera A ma kod 65, a Z – 90. W tabeli kodów ASCII litery alfabetu łacińskiego ustawiono po kolei, co możemy wykorzystać do szyfru. Chcemy jednak literom alfabetu przyporządkować kolejne liczby naturalne (od 0 do 25), czyli zamienić literę na liczbę określającą jej pozycję w alfabecie. Jak to zrobić? Można od kodu ASCII danej litery odjąć kod ASCII litery A równy 65. Na przykład:

dla B $66 - 65 = 1$,
dla L $76 - 65 = 11$,
dla Z $90 - 65 = 25$.

Otrzymaliśmy w ten sposób dla każdej litery jej pozycję w alfabecie (przyjęto numerowanie od zera, aby ułatwić wykonywanie operacji dodawania modulo 26).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

T	U	V	W	X	Y	Z
19	20	21	22	23	24	25

Tabela 4. *Przyporządkowanie literom kolejnych liczb naturalnych*

Aby zamieniać literę na jej pozycję w alfabecie (czyli znak na kod) i odwrotnie (kod na znak), zdefiniujemy dwie funkcje: *znak_na_kod()* i *kod_na_znak()* – rysunki 4a (C++) i 4b (Python).

W języku C++ znak jest tożsamy z jego kodem ASCII, dlatego nie musimy w programie zamieniać liczby na jej kod ASCII – zrobi to automatycznie kompilator. Użyjemy tylko wbudowanej funkcji `toupper()`, która zwraca znak przekazany jako parametr funkcji zamieniony z małej litery na wielką (wielkie litery nie są zmieniane).

C++

```
unsigned int znak_na_kod(char znak)
{
    return toupper(znak) - 'A';
}

unsigned char kod_na_znak(int kod_znaku)
{
    return kod_znaku + 'A';
}
```

Rys. 4a. *Przykładowe funkcje zmieniające literę na jej pozycję w alfabecie i odwrotnie (C++)*

W języku Python, inaczej niż w języku C++, znak nie jest tożsamy z liczbą, dlatego do zamiany litery na jej kod Unicode (i odwrotnie) musimy użyć wbudowanych funkcji:

- `ord()` – zamienia znak na kod Unicode,
- `chr()` – zamienia kod Unicode na odpowiadający mu znak.

Użyjemy również metody `upper()`, która zwraca znak zamieniony z małej litery na wielką.

Uwaga

W przypadku liter alfabetu łacińskiego, cyfr i znaków interpunkcyjnych dostępnych na klawiaturze, kod Unicode tych znaków jest taki sam, jak ich kod ASCII. Zbiór kodów ASCII jest podzbiorem zbioru kodów Unicode.

Python

```
def znak_na_kod(znak):
    return ord(znak.upper()) - ord('A')

def kod_na_znak(kod_znaku):
    return chr(kod_znaku + ord('A'))
```

Rys. 4b. *Przykładowe funkcje zmieniające literę na jej pozycję w alfabecie i odwrotnie (Python)*

Ćwiczenie 10. Testujemy działanie funkcji

1. Przetestuj działanie funkcji pokazanych na rysunkach 4a lub 4b. Napisz program, definiując te funkcje i wywołując je w programie głównym z parametrem aktualnym *litera*. Znak ma być wprowadzany z klawiatury.
2. Zapisz program w pliku pod nazwą *Zamiany*.
3. Sprawdź działanie programu dla liter E oraz e. Co zauważasz? Uzasadnij odpowiedź.

Wskazówka: (Python) Pamiętaj, że w języku Python możesz w trybie interaktywnym (w oknie powłoki Pythona) sprawdzać działanie różnych instrukcji.

Zastanówmy się teraz, jak otrzymać pozycję zakodowanego znaku. Czy wystarczy do liczby określającej pozycję litery dodać wartość klucza, czyli 3?

Dla litery L: 11 + 3 = 14 po zakodowaniu otrzymamy literę O,
ale dla litery Z: 25 + 3 = 28 po zakodowaniu powinniśmy otrzymać literę C, która w alfabecie ma pozycję 2.

Obliczenie samej sumy nie wystarczy. Aby otrzymywać zawsze poprawną pozycję zaszyfrowanego znaku, musimy obliczać resztę z dzielenia sumy pozycji znaku tekstu jawnego i klucza przez 26.

Zatem dla litery Z poprawne działanie to: (25 + 3) % 26 = 28 % 26 = 2 (gdzie operator % oznacza resztę z dzielenia).

Dlatego w funkcji *zaszyfruj()* realizującej algorytm szyfrowania metodą Cezara (rys. 5a i 5b) należy umieścić odpowiednio przypisanie:

C++

```
unsigned int kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26;
```

Python

```
kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26
```

Funkcję *zaszyfruj()* zdefiniujemy z jednym parametrem *napis*, określającym tekst jawny. Zastosujemy pętlę **for**, w której przeglądamy kolejne znaki tekstu, począwszy od 0. Zanim napiszemy cały program, spróbujmy przeanalizować działanie tej funkcji (rys. 5a i 5b).

C++

```
17  string zaszyfruj(string napis)
18  {
19      string wynik = "";
20      for(int i = 0; i < napis.length(); i++)
21      {
22          unsigned int kod_znaku = znak_na_kod(napis[i]);
23          unsigned int kod_znaku_zaszyfrowany =  (kod_znaku + 3) % 26;
24          wynik += kod_na_znak(kod_znaku_zaszyfrowany);
25      }
26      return wynik;
27  }
```

Rys. 5a. *Przykładowa funkcja realizująca algorytm szyfrowania szyfrem Cezara (C++)*

Python

```
def zaszyfruj(napis):
    wynik = ""
    for znak in napis:
        kod_znaku = znak_na_kod(znak)
        kod_znaku_zaszyfrowany = (kod_znaku + 3) % 26
        wynik += kod_na_znak(kod_znaku_zaszyfrowany)
    return wynik
```

Rys. 5b. *Przykładowa funkcja realizująca algorytm szyfrowania szyfrem Cezara (Python)*

Ćwiczenie 11. Piszemy program realizujący algorytm szyfrowania szyfrem Cezara

1. Napisz w wybranym języku programowania program umożliwiający zaszyfrowanie szyfrem Cezara tekstu wprowadzonego przez użytkownika. Wykorzystaj w programie funkcje zapisane w pliku *Zamiany* w ćwiczeniu 10. oraz funkcje szyfrujące pokazane na rysunkach 5a (C++) lub 5b (Python). Wyjaśnij znaczenie poszczególnych wierszy wybranej funkcji szyfrującej.
2. Wywołaj funkcję w programie głównym z parametrem aktualnym.
3. Zapisz program w pliku pod nazwą *Szyfr_Cezara*.
4. Przetestuj program dla różnych danych.

Ćwiczenie 12. Formułujemy algorytm deszyfrowania

1. Napis „SLHV" został zaszyfrowany metodą Cezara. Na przykładzie odszyfrowywania tego słowa uzasadnij, dlaczego kod znaku odszyfrowanego tekstu obliczamy tak, jak pokazano w poniższych instrukcjach przypisania.

   C++

   ```
   unsigned int kod_znaku_odszyfrowany = (kod_znaku - 3 + 26) % 26;
   ```

   Python

   ```
   kod_znaku_odszyfrowany = (kod_znaku - 3 + 26) % 26
   ```

2. Sformułuj algorytm odszyfrowywania tekstu zaszyfrowanego metodą Cezara.

Ćwiczenie 13. Piszemy funkcję deszyfrowania podstawieniowego

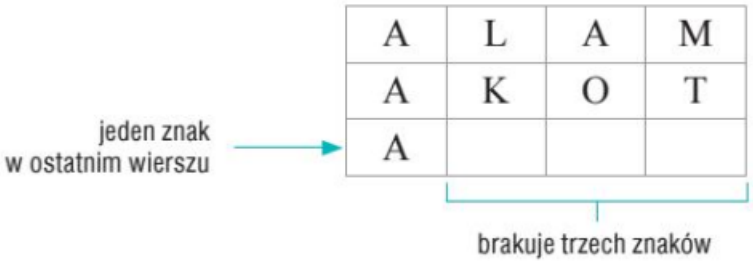
1. Uzupełnij program zapisany w ćwiczeniu 11. o funkcję deszyfrującą tekst wprowadzony przez użytkownika.
2. Wywołaj funkcję w programie głównym.
3. Zapisz plik pod tą samą nazwą i przetestuj działanie programu dla różnych danych.

5.3. Programowanie algorytmu szyfrowania przestawieniowego

{ Chcemy napisać program realizujący algorytm szyfrowania przestawieniowego.
W jaki sposób możemy napisać funkcję szyfrującą w wybranym języku programowania? }

Aby zaszyfrować dany tekst metodą przestawieniową z użyciem kolumn, musimy zadbać o to, by w każdej kolumnie było tyle samo znaków. W przypadku tekstu, którego długość nie jest podzielna przez liczbę kolumn, musimy taki tekst dopełnić wybranym znakiem.

Na przykład, aby zaszyfrować tekst jawny ALAMAKOTA z użyciem czterech kolumn, trzeba będzie dopełnić tekst trzema znakami do dwunastu znaków (rys. 6.).



Rys. 6. *Przykład tekstu jawnego ALAMAKOTA do zaszyfrowania z użyciem czterech kolumn*

Zdefiniujemy funkcję pomocniczą *dopelnij()* z trzema parametrami: *napis*, *znak* i *liczba_znakow*, która zwraca tekst *napis* dopełniony znakami do długości równej *liczba_znakow*.

C++

```
 7  string dopelnij(string napis, char znak, int liczba_znakow)
 8  {
 9      string wynik = napis;
10      for(int i = napis.length(); i < liczba_znakow; i++)
11          wynik += znak;
12      return wynik;
13  }
```

Rys. 7a. *Przykładowa pierwsza funkcja pomocnicza do szyfrowania przestawieniowego (C++)*

Python

```
def dopelnij(napis, znak, liczba_znakow):
    wynik = napis
    for i in range (len(napis), liczba_znakow):
        wynik += znak
    return wynik
```

Rys. 7b. *Przykładowa pierwsza funkcja pomocnicza do szyfrowania przestawieniowego (Python)*

Aby skorzystać z funkcji *dopelnij()*, musimy wiedzieć, jaka powinna być długość tekstu z dopełnieniem. W tym celu obliczamy resztę z dzielenia długości tekstu przez liczbę kolumn, która to wartość odpowiadać będzie liczbie znaków w ostatnim wierszu. Jeżeli będzie ona niezerowa – długość zwiększamy o brakującą liczbę znaków.

W naszym przykładzie tekst jawny ALAMAKOTA ma długość 9, czyli:

$9 \% 4 = 1$, czyli w ostatnim wierszu brakuje: $4 - 1 = 3$ znaków.

Do tekstu jawnego dodajemy brakujące znaki, czyli długość tekstu z dopełnieniem wynosi: $9 + 3 = 12$ znaków.

C++

```
15  int liczba_znakow_z_dopelnieniem(string napis, int liczba_kolumn)
16  {
17      int dlugosc = napis.length();
18      int liczba_znakow_ostatni_wiersz = dlugosc % liczba_kolumn;
19      if(liczba_znakow_ostatni_wiersz != 0)
20          dlugosc += (liczba_kolumn - liczba_znakow_ostatni_wiersz);
21      return dlugosc;
22  }
```

Rys. 8a. *Przykładowa druga funkcja pomocnicza do szyfrowania przestawieniowego (C++)*

Python

```python
def liczba_znakow_z_dopelnieniem(napis, liczba_kolumn):
    dlugosc = len(napis)
    liczba_znakow_ostatni_wiersz = dlugosc % liczba_kolumn
    if liczba_znakow_ostatni_wiersz != 0:
        dlugosc += (liczba_kolumn - liczba_znakow_ostatni_wiersz)
    return dlugosc
```

Rys. 8b. *Przykładowa druga funkcja pomocnicza do szyfrowania przestawieniowego (Python)*

Zapisując algorytm szyfrowania metodą przestawieniową, możemy zauważyć, że czytanie tekstu kolumnami polega w istocie na odczytywaniu co *n*-tego znaku, gdzie *n* odpowiada kluczowi szyfru określonemu jako liczba kolumn. Dla powyższego przykładu, przyjmując, że znaki numerowane są od zera, powinniśmy zatem odczytać znaki o numerach: 0, 4, 8 (pierwsza kolumna), 1, 5, 9 (druga kolumna) itd. W naszym przykładzie użyjemy znaku podkreślenia „_" do dopełnienia tekstu (rys. 9.). Tego samego znaku użyjemy również w funkcji szyfrującej (rys. 10a i 10b).

Znak	A	L	A	M	A	K	O	T	A	_	_	_
Numer znaku	0	1	2	3	4	5	6	7	8	9	10	11

Rys. 9. *Znaki tekstu z dopełnieniem do dwunastu znaków*

Ćwiczenie 14. Szyfrujemy tekst

Zanim napiszesz program, zaszyfruj tekst ALAMAKOTA metodą przestawieniową z użyciem czterech kolumn.

Do szyfrowania tekstu utworzymy funkcję *zaszyfruj()*. Aby wybrać z tekstu co *n*-ty znak, wykorzystujemy zmienną *akt_pozycja*, w której przechowywana jest aktualna pozycja znaku tekstu niezaszyfrowanego. Pozycję tę zwiększamy o parametr *liczba_kolumn*. Jeżeli wartość zmiennej *akt_pozycja* będzie większa lub równa długości tekstu, zmniejszamy ją o długość tekstu i zwiększamy o jeden. Proces ten powtarzamy tak długo, aż wykorzystamy wszystkie znaki tekstu niezaszyfrowanego.

C++

```
24  string zaszyfruj(string napis, int liczba_kolumn)
25  {
26      string wynik = "";
27      napis = dopelnij(napis, '_', liczba_znakow_z_dopelnieniem(napis, liczba_kolumn));
28      int dlugosc = napis.length();
29      int akt_pozycja = 0;
30      for(int i = 0; i < dlugosc; i++)
31      {
32          wynik += napis[akt_pozycja];
33
34
35
36      }
37      return wynik;
38  }
```

Rys. 10a. *Przykładowa funkcja realizująca algorytm szyfrowania przestawieniowego (C++)*

Python

```
def zaszyfruj(napis, liczba_kolumn):
    wynik = ""
    napis = dopelnij(napis, '_', liczba_znakow_z_dopelnieniem(napis, liczba_kolumn))
    dlugosc = len(napis)
    akt_pozycja = 0
    for i in range(dlugosc):
        wynik += napis[akt_pozycja]
        [redacted]
    return wynik
```

***Rys. 10b**. Przykładowa funkcja realizująca algorytm szyfrowania przestawieniowego (Python)*

Ćwiczenie 15. Piszemy program realizujący algorytm szyfrowania przestawieniowego

1. W wybranym języku programowania napisz program szyfrujący szyfrem przestawieniowym kolumnowym tekst wprowadzony przez użytkownika. Liczba kolumn powinna być podawana przez użytkownika.
2. Przykładowe funkcje pomocnicze i szyfrujące pokazano na rysunkach 7a, 8a, 10a (C++) i 7b, 8b, 10b (Python). Możesz się wzorować na tych rozwiązaniach, przy czym w funkcjach szyfrujących pokazanych na rysunkach 10a i 10b brakuje po trzy wiersze programu. Uzupełnij je, korzystając z opisu umieszczonego przed rysunkami.
3. Wywołaj funkcję szyfrującą w programie głównym.
4. Zapisz program w pliku pod nazwą *Szyfrowanie_przestawieniowe*.
5. Przetestuj program dla różnych danych, w tym dla tekstu „ALAMAKOTA".

Deszyfrowanie tekstu wykorzystuje ten sam algorytm, co szyfrowanie. Z tekstu zaszyfrowanego odczytujemy jednak nie co n-ty znak, lecz co m-ty znak, gdzie m odpowiada liczbie wierszy ($m \cdot n = liczba\ znaków\ z\ dopełnieniem$).

Ćwiczenie 16. Piszemy funkcję deszyfrowania przestawieniowego

1. Uzupełnij program zapisany w ćwiczeniu 15. o funkcję deszyfrującą tekst wprowadzony przez użytkownika. Liczba kolumn powinna być podawana przez użytkownika. Wywołaj funkcję w programie głównym.
2. Zapisz plik pod tą samą nazwą.
3. Przetestuj program dla różnych danych.

6. Więcej o szyfrowaniu

6.1. Szyfrowanie symetryczne i asymetryczne

Podczas szyfrowania danych z kluczem możemy spotkać się z dwoma rodzajami algorytmów szyfrujących: **symetrycznym** i **asymetrycznym**.

Szyfrowanie symetryczne wykorzystuje jeden klucz – ten sam zarówno do szyfrowania, jak i deszyfrowania wiadomości, np. dla szyfru przestawieniowego kluczem jest liczba kolumn.

W **algorytmach asymetrycznych** wiadomość dla danego adresata zaszyfrować może każdy, używając **klucza publicznego** tego adresata. Natomiast odczytanie wiadomości możliwe jest tylko przy użyciu innego klucza – **klucza prywatnego**, który zna jedynie odbiorca. Z algorytmów asymetrycznych korzysta się np. przy uwierzytelnianiu transakcji bankowych czy w certyfikatach bezpieczeństwa stron HTTPS. Popularne protokoły szyfrowania, które zapobiegają przechwytywaniu i zmienianiu przesyłanych danych, wykorzystywane m. in. na stronach internetowych, to SSL lub TLS.

Pamiętaj, aby sprawdzać czy strona, na której podajesz dane wrażliwe, tj. hasło lub dane karty kredytowej, np. strona płatności w sklepie internetowym, wykorzystuje aktualny certyfikat szyfrowania HTTPS (znak kłódki lub *https://* w adresie strony). Jeśli strona go nie ma, twoje dane mogą zostać przechwycone podczas ich przesyłania.

6.2. Maszyna Enigma

Metodę podstawieniową wykorzystano w słynnej maszynie szyfrującej Enigma, która w sposób mechaniczny dokonuje kolejnych podstawień alfabetycznych. Podstawowy model maszyny Enigma wynalazł Arthur Scherbius w Berlinie w 1918 roku. Szyfrowała ona wiadomość, wykonując kolejno pewną liczbę podmian znaków i wykorzystując przy tym połączenia elektryczne. Jej najważniejszymi elementami były obracające się na jednej osi bębny szyfrujące.

Rys. 11. *Maszyna Enigma*

Jeszcze przed rozpoczęciem II wojny światowej Enigmą zainteresowało się niemieckie wojsko, które po zmodyfikowaniu jej konstrukcji i zasad działania wprowadziło maszynę do własnego użytku.

W kolejnych latach w różnych krajach trwały prace nad złamaniem niemieckich szyfrów. Panowało przekonanie, że najlepsi w tej dziedzinie będą językoznawcy i lingwiści. Polacy jako pierwsi zatrudnili w tym celu profesjonalnych matematyków, byli to Jerzy Różycki, Henryk Zygalski oraz Marian Rejewski, którym w 1932 roku udało się rozwiązać szyfr Enigmy. Polacy mogli odczytywać korespondencję niemiecką, jednak nie było to proste, bowiem Niemcy stale udoskonalali maszynę i procedury szyfrowania. Podczas II wojny światowej łamaniem szyfrów Enigmy zajmował się w Wielkiej Brytanii zespół matematyków kierowany przez Alana Turinga, jednego z pionierów informatyki.

6.3. Algorytm RSA

Jednym z aktualnie najpopularniejszych asymetrycznych algorytmów szyfrowania jest algorytm RSA. Jego nazwa pochodzi od pierwszych liter nazwisk jego twórców: Rona Rivesta, Adiego Shamira oraz Leonarda Adlemana, którzy zaprojektowali go w 1977 roku.

Jest to pierwszy algorytm, który może być stosowany zarówno do szyfrowania, jak i do podpisów elektronicznych.

W celu wygenerowania pary kluczy prywatnego i publicznego w algorytmie stosuje się działania i funkcje matematyczne na dużych liczbach pierwszych.

Obecnie RSA stosuje się na przykład w systemie międzynarodowych przekazów bankowych SWIFT oraz w przeglądarkach internetowych. Również wspomniane wcześniej certyfikaty bezpieczeństwa stron HTTPS bazują na tego typu algorytmie.

Warto zapamiętać

- Aby zabezpieczać informacje, można posługiwać się różnymi algorytmami szyfrującymi, m.in. algorytmem podstawieniowym i przestawieniowym.
- Szyfr, w którym każdy znak tekstu jawnego jest zastępowany innym znakiem, nazywamy szyfrem podstawieniowym. Najbardziej znany przykład szyfru podstawieniowego to szyfr Cezara.
- Szyfr, w którym w tekście zaszyfrowanym pojawiają się wszystkie znaki tekstu jawnego, ale w innej kolejności, nazywamy szyfrem przestawieniowym.
- Gdy w wyrazie lub zdaniu przestawimy litery lub sylaby, wykorzystując wszystkie litery (głoski lub sylaby) tego wyrazu lub zdania – powstaje anagram.
- Szyfrowanie symetryczne wykorzystuje jeden klucz – ten sam zarówno do szyfrowania, jak i deszyfrowania wiadomości. W algorytmach asymetrycznych wiadomość zaszyfrować może każdy, używając klucza publicznego. Wiadomość można odczytać przy użyciu klucza prywatnego, który zna jedynie odbiorca.

Pytania i polecenia

1. Czym jest szyfr?
2. Czym różni się tekst jawny od szyfrogramu?
3. Czym zajmuje się kryptografia, a czym kryptoanaliza?
4. W jaki sposób działają techniki szyfrowania wiadomości? Podaj kilka przykładów.
5. Wyjaśnij na wymyślonym przez siebie przykładzie, na czym polega szyfrowanie podstawieniowe.
6. Wyjaśnij na wymyślonym przez siebie przykładzie, na czym polega szyfrowanie przestawieniowe.
7. Czym jest anagram?
8. Czym się różni szyfrowanie symetryczne od asymetrycznego?
9. Do czego służyła maszyna Enigma? Która metoda szyfrowania była w niej wykorzystywana?

Zadania

1. Odszyfruj tekst: „VCBIUFHCDUDMHVWSURVWB".
2. Korzystając z metod szyfrowania omówionych w tym temacie, zaszyfruj na dwa sposoby swoje dane osobowe (imię, nazwisko, miejsce zamieszkania).
3. Zadanie do wykonania w grupie dwuosobowej. Każda z osób wymyśla własny algorytm szyfrowania, następnie przekazuje zaszyfrowany tekst drugiej osobie do odszyfrowania.
4. Do programu zapisanego w ćwiczeniu 13. dodaj możliwość wprowadzania z klawiatury dowolnej wartości klucza. Zapisz plik pod tą samą nazwą.
5. Do programu zapisanego w zadaniu 4. dodaj możliwość wyboru szyfrowania lub deszyfrowania bądź zakończenia programu.
6. Do programu zapisanego w ćwiczeniu 16. dodaj możliwość wyboru szyfrowania lub deszyfrowania bądź zakończenia programu.

Dla zainteresowanych

7. Napisz program wyświetlający wszystkie anagramy utworzone na podstawie wyrazu „pora".
8. Steganografia to nauka o ukrywaniu informacji wewnątrz innej informacji. Zapoznaj się z tym zagadnieniem, korzystając z dodatkowej literatury i źródeł internetowych.
9. Korzystając z Internetu i dodatkowej literatury, znajdź informacje na temat Enigmy. **Wskazówka**: Zadanie można wykonać w grupie kilku osób, które następnie mogą w dyskusji podzielić się wiedzą.
10. Korzystając z Internetu i dodatkowej literatury, dowiedz się więcej na temat algorytmu z kluczem jawnym RSA.