Ministerul Educaţiei şi Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică

Laboratory work 1

Subject: Cryptography methods for information protection

Done by: Gitlan Gabriel
st. gr. FAF-213

Verified by: Aureliu ZGUREANU
asist. univ.

Chişinău - 2023

Subiectul: Cifrul lui Cesar

Sarcini:

1. De implementat algoritmul Cezar pentru alfabetul limbii engleze în unul din limbajele de programare. Utilizați doar codificarea literelor cum este arătat în tabelul 1 (nu se permite de folosit modificările specificate în limbajul de programare, de ex. ASCII sau Unicode). Valorile cheii vor fi cuprinse între 1 și 25 inclusiv și nu se permit alte valori. Valorile caracterelor textului sunt cuprinse între 'A' și 'Z', 'a' și 'z' și nu sunt premise alte valori. În cazul în care utilizatorul introduce alte valori - i se va sugera diapazonul corect. Înainte de criptare textul va fi transformat în majuscule și vor fi eliminate spațiile. Utilizatorul va putea alege operația - criptare sau decriptare, va putea introduce cheia, mesajul sau criptograma și va obține respectiv criptograma sau mesajul decriptat.
2. De implementat algoritmul Cezar cu 2 chei, cu păstrarea condițiilor exprimate în Sarcina 1.1. În plus, cheia 2 trebuie să conțină doar litere ale alfabetului latin, și să aibă o lungime nu mai mică de 7.

Cifrul lui Cesar

Cifrul lui Cesar (sau Cezar). În acest cifru fiecare literă a textului clar este înlocuită cu o nouă literă obţinută printr-o deplasare alfabetică. Cheia secretă k, care este aceeaşi la criptare cât şi la decriptare, constă în numărul care indică deplasarea alfabetică, adică k{1, 2, 3,..., n–1}, unde n este lungimea alfabetului. Criptarea şi decriptarea mesajului cu cifrul Cezar poate fi definită de formulele c = ek(x) = x + k (mod n), m = dk(y) = y – k (mod n), unde x și y sunt reprezentarea numerică a caracterului respectiv din textul clar m și din criptograma c. Funcţia numită Modulo (a mod b) returnează restul împărţirii numărului întreg a la numărul întreg b. Această metodă de criptare este numită aşa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi, folosind cheia k = 3.

Rezultatul efectuării sarcinilor:

```java
public static void main(String[] args) {
    System.out.println("Caesar");
    cesarEncryption(word:"CARTE", key:3);
    System.out.println();
    cesarDecryption(word:"FDUWH", key:3);

    System.out.println("\n\nBrut");
    cesarBrutEncryption(word:"cary", key:3, keyWord:"SDFGS");
    System.out.println();
    cesarBrutDecryption(word:"IEVD", key:3, keyWord:"SDFGS");
}
```

Figure 1 – Main method and some test cases

```java
private static void keyVerification(int key) {
    if ((key > 25) || (key ≤ 0)) {
        throw new NoSuchElementException("Introduce 0 < K < 26");
    }
}
```

Figure 2 – Function that check whether key is valid

```java
public static void cesarEncryption(String word, int key) {
    keyVerification(key);
    List<Character> chars = new ArrayList<>();
    for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {
        chars.add(ch);
    }
    List<Character> alphabetList = getAlphabetList();
    chars.stream().map(character → {
        int i = indexOfElement(alphabetList, character);
        if ((i + key) ≥ alphabetList.size()) {
            return alphabetList.get(indexOfElement(alphabetList, character) - 26 + key);
        }
        return alphabetList.get(indexOfElement(alphabetList, character) + key);
    }).forEach(System.out::print);
}

public static void cesarDecryption(String word, int key) {
    keyVerification(key);
    List<Character> chars = new ArrayList<>();
    for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {
        chars.add(ch);
    }
    List<Character> alphabetList = getAlphabetList();
    chars.stream().map(character → {
        int i = indexOfElement(alphabetList, character);
        if ((i - key) < 0) {
            return alphabetList.get(i + 26 - key);
        }
        return alphabetList.get(i - key);
    }).forEach(System.out::print);
}
```

Figure 3 – Caesar Encryption and Decryption

```java
public static void cesarBrutEncryption(String word, int key, String keyWord) {
    keyVerification(key);
    List<Character> chars = new ArrayList<>();
    for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {
        chars.add(ch);
    }
    List<Character> alphabetList = getAlphabetList();
    List<Character> keyWordList = new ArrayList<>();
    for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {
        keyWordList.add(ch);
    }
    keyWordList.forEach(alphabetList::remove);
    alphabetList.addAll(0, keyWordList.stream().distinct().toList());
    chars.stream().map(character -> {
        int i = indexOfElement(alphabetList, character);
        if ((i + key) >= alphabetList.size()) {
            return alphabetList.get(indexOfElement(alphabetList, character) - 26 + key);
        }
        return alphabetList.get(indexOfElement(alphabetList, character) + key);
    }).forEach(System.out::print);
}

public static void cesarBrutDecryption(String word, int key, String keyWord) {
    keyVerification(key);
    List<Character> chars = new ArrayList<>();
    for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {
        chars.add(ch);
    }
    List<Character> alphabetList = getAlphabetList();
    List<Character> keyWordList = new ArrayList<>();
    for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {
        keyWordList.add(ch);
    }
    keyWordList.forEach(alphabetList::remove);
    alphabetList.addAll(0, keyWordList.stream().distinct().toList());
    chars.stream().map(character -> {
        int i = indexOfElement(alphabetList, character);
        if ((i - key) < 0) {
            return alphabetList.get(i + 26 - key);
        }
        return alphabetList.get(i - key);
    }).forEach(System.out::print);
}
```

Figure 4 – Caesar Brut Encryption and Decryption

```java
public static int indexOfElement(List<Character> characterList, Character elementToFind) {
    int index = 0;
    for (Character element : characterList) {
        if (element.equals(elementToFind)) {
            return index;
        }
        index++;
    }
    return -1;
}
```

Figure 5 – Function that finds the index of an element in a list

```java
private static List<Character> getAlphabetList() {
    List<Character> alphabet = new ArrayList<>();
    for (char letter = 'A'; letter <= 'Z'; letter++) {
        alphabet.add(letter);
    }
    return alphabet;
```

Figure 6 – Function to declare a list with alphabet characters (without ASCII)

Figure 7 – Program output

Conclusion: During my recent laboratory work, I explored the operations of both Caesar and Caesar Brut encryption and decryption methods. I developed an application capable of encrypting and decrypting words using a key that corresponds to the word's position on a list. It's worth noting that the second program enhances the security of the code significantly, making it more challenging for unauthorized individuals to decipher the sender's intended message. This is because the attacker would need to search through a staggering number of 26! * 25 possible word versions.

Git repo: https://github.com/5anji/CS_Labs