VIGENERE CIPHER

I will first list all of the functions that I use in my code along with an explanation what each function does(some of the function for example the printing ones are just written for easier debugging):

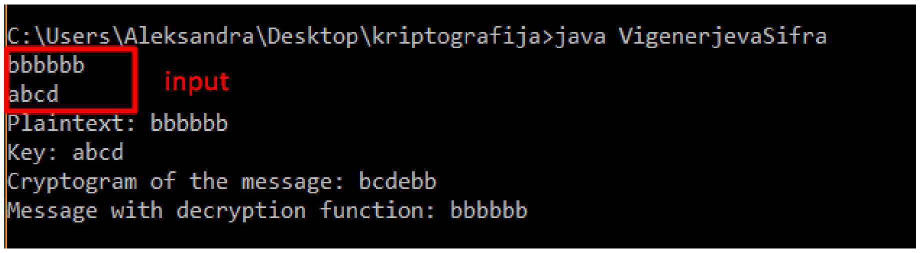
- public static int[] CharToInt(String b) -takes as string and returns an integer representation of it sotred in an array
- public static String IntToChar(int[] b) - takes integer representation of the letter and transforms it to a string
- public static void printString(String b) – prints a string
- public static void printArray(int[] b) – prints integer representation of a text
- public static int[] frequencies(String message) – returns an array of size 26 that contains the number of times each letter of the alphabet appears in the message
- public static double indexSovpadanja(int[] f, int[] m) – calculates the matching index using the formula: sum of the squares of the frequencies of each letter from the alphabet in a message, divided by the square of the length of the message
- public static int divisers(int n) – returns the number of divisors of the number n
- public static int minimal(double[] razliki) - it returns the index of the minimal difference in an array
- public static int maxFreq(int[] frequencies) - a function that returns the index of the maximal frequency in an array
- public static String Encrypt(String b, String k) – the encryption function. Given a plaintext and a key, it returns the cipher text as string. If the message is long l and the key is long r, then we apply the key l/r times consecutively. The key is consisted of r many letters and what we actually do is apply Caesar cipher to a pair of corresponding letters from the message and from the key. Each letter of the message is actually shifted to the right for the corresponding value in the key.

```java
public static String Encrypt(String b, String k){
    int[] word1=CharToInt(b);
    int[] word2=CharToInt(k);
    IntToChar(word1);
    int i=0;
    for(int l=0; l<b.length()/k.length(); l++)
        for(int j=0; j<k.length(); j++){
            word1[i]=(word1[i]+word2[j])%26;
            i++;
            if(i>=b.length())
                break;
        }
    return IntToChar(word1);
}
```

- public static String Decrypt(String b, String k) – the decryption function. The idea is the same as for the encrypting, only this time we shift to the left. All of the operations that are done are in Z_{26}

```
public static String Decrypt(String b, String k){
    int[] word1=CharToInt(b);
    int[] word2=CharToInt(k);
    IntToChar(word1);
    int i=0;
    for(int l=0; l<b.length()/k.length(); l++)
        for(int j=0; j<k.length(); j++){
            word1[i] = ((word1[i] - word2[j]) + 26) % 26;
            i++;
            if (i >= b.length())
                break;
        }
    return IntToChar(word1);
}
```

Example:



```
C:\Users\Aleksandra\Desktop\kriptografija>java VigenerjevaSifra
bbbbbb
abcd
Plaintext: bbbbbb
Key: abcd
Cryptogram of the message: bcdebb
Message with decryption function: bbbbbb
```

- public static int findLengthOfKey(String message, int[] frequencies)
  Idea: We try all possible lengths of the key (I said that it cant go above (lengths of message/10) just for the program to be faster, because if for example the key is as long as the message then it is impossible to be broken since we have only the ciphertext). Let's say that l is length of cipher and i is the current possible length of the key. The cipher can be divided into l/i many substrings. For one substring we take each i-th letter starting from the j-th letter. Then we calculate the matching index using the frequencies of the letters in our substring. For each possible value of I, we store the difference between its matching index and the matching index of the English language which on the lectures we said that it is around 0.065. In the end after calculating the whole array of differences, we take the minimal one, and the index of that result gives us the length of the key.

```
public static int findLengthOfKey(String message, int[] frequencies) {
    int[] text = CharToInt(message);
    int l = text.length;
    double[] razliki = new double[divisors(l)]; //contains the differences between
each sum and 0.065 (index sovpadanja of the English alphabet), we want the minimal
sum
    int counter = 0;
    //double[] errors=[l/9]
    for (int i = ; i < l/10; i++) {
        int[] y = new int[l / i];
        int k = 0;
        for (int j = 0; j < l; j+=i) {
            y[k] = text[j];
            k++;
            if (k >= l / i) //just for null pointer exception
                break;
        }
        int[] f = frequencies(new String(IntToChar(y)));
        double indexC = indexSovpadanja(f, y);
        System.out.println("Ic: " + indexC);
        razliki[counter] = Math.abs(indexC - 0.065);
        counter++;
    }
    return minimal(razliki);
}
```

- public static char breakingOneCezarjevaSifra(int[] cipher)- this function will return one letter of the key. Cypher is a row from int[][] y : a string where each letter is encrypted with the same Cesar cypher.
  Idea:
  Take the max frequency and say that that is the cipher for letter 'e' because that is the most frequent letter. Then we say that (letter in y1)-k1='e' and find k1 from this. We do the exact same for all yi.
  1. x - most frequent letter in yi (x is a number (0 instead of 'a',...))
  2. 'e' = 4 //integer value
  3. x-4=ki a.k.a. x+22=ki mod26 (22 is additive inverse of 4 //22=-4mod26)
  4. (char)(ki+'a') is the actual letter that we want
  5. rezultat[i]= ki; //rezultat is a char table that contains the letters of the key

```
public static char breakingOneCezarjevaSifra(int[] cipher){
    int[] f=frequencies(IntToChar(cipher));
    int x=maxFreq(f); //x is a number between 0 and 25 which is a cipher for the
letter 'e' in the english alphabet
    int ki=(x+22)%26;
    return (char)(ki+'a');
}
```

- public static String breakingCiphertext(String cipher) – Idea: divide the cipher into n-many subciphers that are encrypted with the same cesar cipher(There are cipher.length/n many substrings. For the j-th substring we take each n-th letter starting from the j-th letter.). We need to break n-many cesar ciphers, so for each subcipher separately and add them together into one word(the key). At the end we call our decryption function for the cipher and the key that we got represented as a string.

```java
public static String breakingCiphertext(String cipher){
    int n=findLengthOfKey(cipher, frequencies(cipher)); //length of the key
    int l=cipher.length();
    int[] ciphertext=CharToInt(cipher);
    int[][] y=new int[n][l/n]; //here each row will be the integer values of the letters of a substring that are encrypted with the same Caesar cipher
    for(int i=0; i<y.length; i++){
        int k=0; //index of a column for row i
        for(int j=i; j<l; j+=n){ //go through the whole ciphertext
            y[i][k]=ciphertext[j]; //for yi take each n-th letter
            k++;
            if(k>=l/n) //just for avoiding null pointer exception
                break;
        }
    }
    //We divided the cypher into n-many subcyphers that are encrypted with the same caesar cipher. Now we need to break n-many caesar ciphers, so for each subcipher separately
    char[] key=new char[n]; //each element of the char array key is one letter of the key that we are searching for
    for(int i=0; i<n; i++) //we go through all rows of y
        key[i]=breakingOneCezarjevaSifra(y[i]);

    System.out.println("Key with breaking the cipher: "+new String(key));
    return Decrypt(cipher, new String(key));
}
```