Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

Практическая работа №2

«Маршрутные и подстановочные шифры»

|  |  |
| --- | --- |
| Выполнил: ст. гр. 910901 | Нехайчик И. И. |
| Проверил: | Давыдович К. И. |

Минск 2022

**Задание**

1. Открытый текст: *Please note that spaces and punctuation characters have been removed before encryption*. Шифротекст: *LMBEUDOPUASIIYUNEDDUDOENSPARTEYOPODGCTEDUASTIDZBCBDPUCNPZBACBKMTDZDPGWZOYOPO*. Найти ключ.
2. Необходимо зашифровать свою фамилию шифром маршрутной перестановки.
3. Придумать ключ. Написать код шифровки своих фамилии, имени, отчества. Написать код дешифровки своих фамилии, имени, отчества.

**Решение задания 1**



**Листинг кода к заданиям 2, 3**

const NAME\_STRING = 'Nekhaychik Irina Igorevna';

const TABLE\_WIDTH = 5;

const PLAYFAIR\_CIPHER\_KEY = 'TABLE';

function encryptWithRoutePermutationCipher (str, width) {

  const table = [];

  let strIndex = 0;

  let result = '';

  for (let i = 0; i < Math.ceil(str.length / width); i++) {

    table[i] = [];

    for (let j = 0; j < width; j++) {

      if (str[strIndex]) {

        table[i].push(str[strIndex++]);

      } else {

        table[i].push('\*');

      }

    }

  }

  for (let j = 0; j < width; j++) {

    for (let i = 0; i < table.length; i ++) {

      result += table[i][j];

    }

  }

  console.log(`Input string: ${NAME\_STRING}`);

  console.log(`Table: `, table);

  return result;

}

function createTableForPlayfairCipher (key) {

  const table = [];

  let charCode = 65;

  const keyArr = key.split('');

  for (let i = 0; i < 5; i++) {

    table[i] = [];

    for (let j = 0; j < 5; j++) {

      if (i === 0) {

        table[i].push(keyArr[j]);

      } else {

        while (keyArr.includes(String.fromCharCode(charCode)) || charCode === 74) {

          charCode++;

        }

        table[i].push(String.fromCharCode(charCode));

        charCode++;

      }

    }

  }

  return table;

}

function getBigramsFromString (str) {

  let cryptedStr = str.split(' ').join('').toUpperCase();

  const bigramsArr = [];

  for (let i = 0; i < cryptedStr.length + (cryptedStr.length % 2); i = i + 2) {

    if (i + 1 === cryptedStr.length + (cryptedStr.length % 2) - 1 && cryptedStr[i + 1] || cryptedStr[i] !== cryptedStr[i + 1] && cryptedStr[i + 1]) {

      bigramsArr.push(cryptedStr[i] + cryptedStr[i + 1]);

    } else if (i !== cryptedStr.length + (cryptedStr.length % 2)) {

      bigramsArr.push(cryptedStr[i] + 'X');

      if (cryptedStr[i + 1]) {

        i--;

      }

    }

  }

  return bigramsArr;

}

function cryptWithPlayfairCipher (str, key) {

  const table = createTableForPlayfairCipher(key);

  const bigramsArr = getBigramsFromString(str);

  let result = '';

  console.log('Table: ', table);

  console.log('Bigrams from input string: ', bigramsArr);

  bigramsArr.forEach((bigram) => {

    let i = 0;

    let a, b;

    while (i < 5) {

      if (table[i].includes(bigram[0])) {

        a = i;

        b = table[i].findIndex((el) => el === bigram[0]);

        i = 5;

      }

      i++;

    }

    let j = 0;

    let c, d;

    while (j < 5) {

      if (table[j].includes(bigram[1])) {

        c = j;

        d = table[j].findIndex((el) => el === bigram[1]);

        j = 5;

      }

      j++;

    }

    if (a === c) {

      if (b < TABLE\_WIDTH - 1) {

        b = b + 1;

      } else {

        b = 0

      }

      if (d < TABLE\_WIDTH - 1) {

        d = d + 1;

      } else {

        d = 0;

      }

      result += table[a][b] + table[c][d];

    } else if (b === d) {

      if (a < table.length - 1) {

        a = a + 1;

      } else {

        a = 0;

      }

      if (c < table.length - 1) {

        c = c + 1;

      } else {

        c = 0;

      }

      result += table[a][b] + table[c][d];

    } else {

      result += table[a][d] + table[c][b];

    }

  });

  return result;

}

function encryptPlayfairCipher (str, key) {

  const table = createTableForPlayfairCipher(key);

  const bigramsArr = getBigramsFromString(str);

  let result = '';

  console.log('Table: ', table);

  console.log('Bigrams from crypted string: ', bigramsArr);

  bigramsArr.forEach((bigram) => {

    let i = 0;

    let a, b;

    while (i < 5) {

      if (table[i].includes(bigram[0])) {

        a = i;

        b = table[i].findIndex((el) => el === bigram[0]);

        i = 5;

      }

      i++;

    }

    let j = 0;

    let c, d;

    while (j < 5) {

      if (table[j].includes(bigram[1])) {

        c = j;

        d = table[j].findIndex((el) => el === bigram[1]);

        j = 5;

      }

      j++;

    }

    if (a === c) {

      if (b !== 0) {

        b = b - 1;

      } else {

        b = TABLE\_WIDTH - 1;

      }

      if (d !== 0) {

        d = d - 1;

      } else {

        d = TABLE\_WIDTH - 1;

      }

      result += table[a][b] + table[c][d];

    } else if (b === d) {

      if (a !== 0) {

        a = a - 1;

      } else {

        a = table.length - 1;

      }

      if (c !== 0) {

        c = c - 1;

      } else {

        c = table.length - 1;

      }

      result += table[a][b] + table[c][d];

    } else {

      result += table[a][d] + table[c][b];

    }

  });

  return result;

}

console.log('ROUTE PERMUTATION CIPHER');

console.log(`Crypted string: ${encryptWithRoutePermutationCipher(NAME\_STRING, TABLE\_WIDTH)}\n\n`);

console.log('CRYPT WITH PLAYFAIR CIPHER');

const CRYPTED\_WITH\_PLAYFAIR\_CIPHER\_STRING = cryptWithPlayfairCipher(NAME\_STRING, PLAYFAIR\_CIPHER\_KEY);

console.log(`Crypted string: ${CRYPTED\_WITH\_PLAYFAIR\_CIPHER\_STRING}\n\n`);

console.log('ENCRYPT PLAYFAIR CIPHER');

console.log(`Encrypted string: ${encryptPlayfairCipher(CRYPTED\_WITH\_PLAYFAIR\_CIPHER\_STRING, PLAYFAIR\_CIPHER\_KEY)}`);

**Вывод в консоли**

ROUTE PERMUTATION CIPHER

Input string: Nekhaychik Irina Igorevna

Table:  [

  [ 'N', 'e', 'k', 'h', 'a' ],

  [ 'y', 'c', 'h', 'i', 'k' ],

  [ ' ', 'I', 'r', 'i', 'n' ],

  [ 'a', ' ', 'I', 'g', 'o' ],

  [ 'r', 'e', 'v', 'n', 'a' ]

]

Crypted string: Ny arecI ekhrIvhiignaknoa

CRYPT WITH PLAYFAIR CIPHER

Table:  [

  [ 'T', 'A', 'B', 'L', 'E' ],

  [ 'C', 'D', 'F', 'G', 'H' ],

  [ 'I', 'K', 'M', 'N', 'O' ],

  [ 'P', 'Q', 'R', 'S', 'U' ],

  [ 'V', 'W', 'X', 'Y', 'Z' ]

]

Bigrams from input string:  [

  'NE', 'KH', 'AY',

  'CH', 'IK', 'IR',

  'IN', 'AI', 'GO',

  'RE', 'VN', 'AX'

]

Crypted string: OLODLWDCKMMPKOTKHNUBYIBW

ENCRYPT PLAYFAIR CIPHER

Table:  [

  [ 'T', 'A', 'B', 'L', 'E' ],

  [ 'C', 'D', 'F', 'G', 'H' ],

  [ 'I', 'K', 'M', 'N', 'O' ],

  [ 'P', 'Q', 'R', 'S', 'U' ],

  [ 'V', 'W', 'X', 'Y', 'Z' ]

]

Bigrams from crypted string:  [

  'OL', 'OD', 'LW',

  'DC', 'KM', 'MP',

  'KO', 'TK', 'HN',

  'UB', 'YI', 'BW'

]

Encrypted string: NEKHAYCHIKIRINAIGOREVNAX