МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА АЭРОКОСМИЧЕСКИХ КОМПЬЮТЕРНЫХ И ПРОГРАММНЫХ СИСТЕМ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассисстент		А.А. Бурков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

Использование циклических кодов для обнаружения ошибок в сетях передачи данных

по курсу: ОСНОВЫ ПОСТРОЕНИЯ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ И СЕТЕЙ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5722		Е. Д. Энс
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1. Цель работы: исследование типового алгоритма формирования контрольной суммы с использованием циклических кодов, разработка программы, наглядно демонстрирующей работу кодера и декодера для типового алгоритма формирования циклических кодов.

2. Описание моделируемой системы: к передаваемым данным добавляют контрольную сумму, которая вычисляется на основе этих же данных. По каналу передается сообщение, состоящее из данных и контрольной суммы. Использование контрольной суммы позволяет определить, по принятому сообщению, возникли ли ошибки при передаче данного сообщения по каналу.

Рис. 1. Структурная схема системы передачи данных:
$\bar{m}$ – информационное сообщение, К – блок кодера,
$\bar{a}$ –закодированное сообщение, $\bar{e}$ –вектор ошибок,
$\bar{b}$ – сообщение на выходе канала, Д – блок декодера,
E –принятое решение, $\bar{m}'$ – сообщение на выходе декодера

На вход кодера поступает некоторое информационное сообщение m, состоящее из нулей и единиц. Кодер по некоторому алгоритму вычисляет контрольную сумму, дописывает ее к передаваемому сообщению и таким образом формирует закодированное сообщение а так же состоящее из 0 и 1. В канале могут произойти ошибки, в результате которых некоторые биты сообщения инвертируются. Вектор ошибок показывает на каких позициях произошла ошибка, при этом канал может быть описан как операция XOR передаваемого сообщения и вектора ошибок. Декодер по некоторому алгоритму проверяет контрольную сумму в принятом сообщении и принимает одно из следующих решений:

$$E = \begin{cases} 1, \text{если были ошибки} \\ 0, \text{если не было ошибок} \end{cases}.$$

В работе рассматривается модель двоично-симметричного канала (ДСК) без памяти представленного на рис. 2.

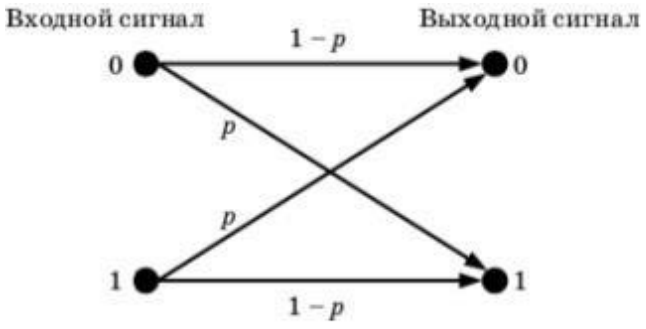


Рис. 2. Модель двоично-симметричного канала

3. Описание проводимого исследования: в данной работе необходимо разработать программу, наглядно демонстрирующую работу кодера и декодера для типового алгоритма формирования циклических кодов.

Кодер хранит порождающий многочлен *g(x)*. Степень многочлена обозначается как deg(g)x)) = *r* и определяет количество бит контрольной суммы в кодовом слове. *k* – число информационных символов передаваемого сообщения $\overline{m}$.

Передаваемое сообщение рассматривается как вектор длины *k*. Для каждого сообщения $(\overline{m})$ кодер выполняет следующие действия:

1) На основе вектора $\overline{m}$ формируется многочлен *m(x)*. Степень многочлена *m(x)* при этом меньше или равна *r – 1*;

2) Вычисляется многочлен $c(x) = m(x)x^r mod\ g(x)$. Степень многочлена *c(x)* при этом меньше или равна *r – 1*;

3) Вычисляется многочлен $a(x) = m(x)x^r + c(x)$;

4) На основе многочлена *a(x)* формируется вектор $\overline{a}$, длина которого *n* бит, где *n* = *k* + *r*.

Алгоритм декодирования выглядит следующим образом: декодер хранит порождающий многочлен *g(x)* и длину кодового слова *n*. Далее выполняются следующие действия:

1) Принятое сообщение $\overline{b} = a + e$ (где *e* – вектор ошибок) переводится в многочлен *b(x)*;

2) Вычисляется синдром: $s(x) = b(x) mod\ g(x)$;

3) Если *s(x)* ≠ 0, то декодер выносит решение, что произошли ошибки (*E* = 1, *E* – решение, принятое декодером), иначе декодер выносит решение, что ошибки не произошли (*E* = 0):

$$E = \begin{cases} 1, & s(x) \neq 0 \\ 0, & s(x) = 0 \end{cases}.$$

При этом стоит отметить следующее:

$$\text{Ошибка декодирования} = \begin{cases} \overline{e} \neq 0 \\ E = 0 \end{cases}.$$

Дополнительное задание: исследовать альтернативную реализацию алгоритма декодирования. Последовательность $\overline{b}$ на выходе из канала (см. рис. 3) делится на две части. Первая последовательность $\overline{m}_b$ содержит в себе символы, относящиеся к информационной части. Вторая $\overline{c}_b$ содержит в себе символы, относящиеся к контрольной сумме. Последовательность $\overline{m}_b$ вновь подаётся на вход кодера, в результате чего вычисляется контрольная сумма $\overline{c}_b'$. Если $\overline{c_b} \neq \overrightarrow{c_b}$, то принимается решение о наличии ошибок при передаче.

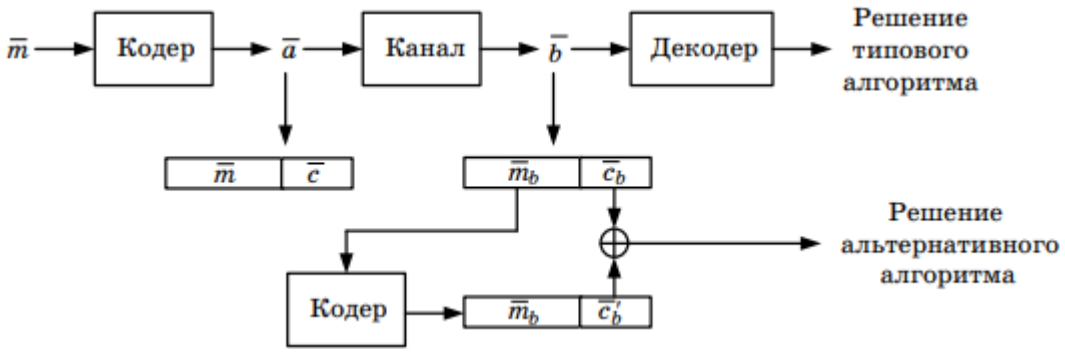
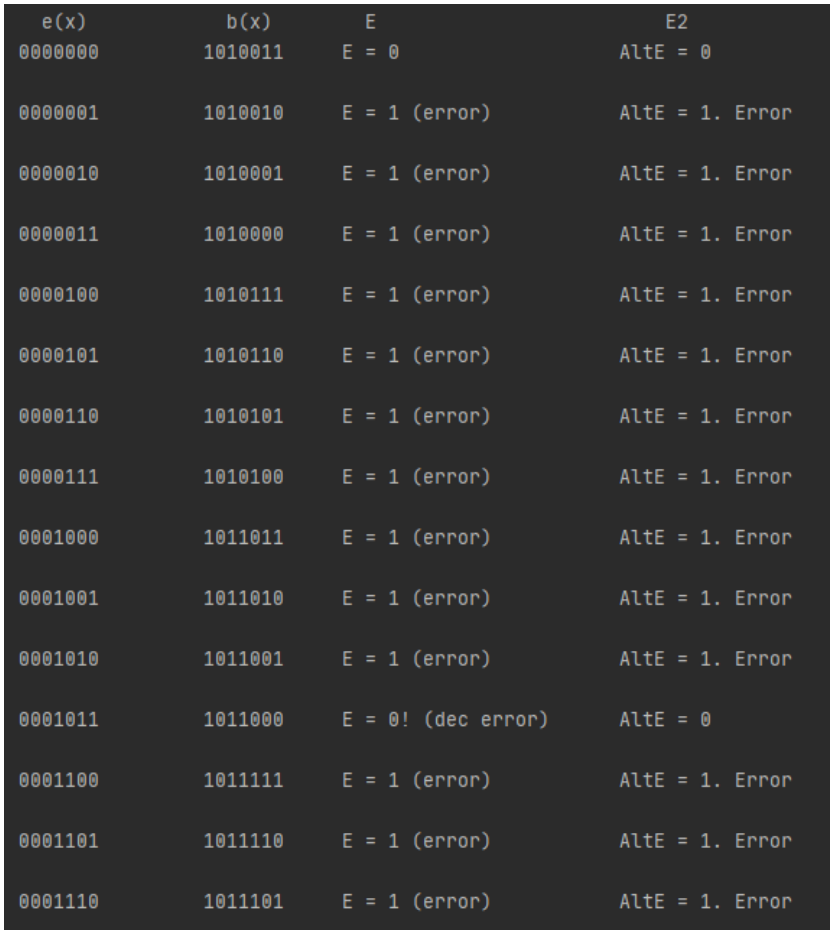


Рис. 3. Схема альтернативного алгоритма декодирования

4. Результаты проводимых исследований:

```
g(x): x^3 + x^1 + x^0
1011

m(x): x^3 + x^1
1010

m(x) * x^ 3 =x^6 + x^4
1010000

c(x): x^1 + x^0
11

a(x): x^6 + x^4 + x^1 + x^0
1010011

----------------------------
e(x): 0

0000000

b(x): x^6 + x^4 + x^1 + x^0
1010011

s(x): 0



E = 0
```

Рис. 4. Результат работы программы с типовым декодером



e(x)	b(x)	E	E2
0000000	1010011	E = 0	AltE = 0
0000001	1010010	E = 1 (error)	AltE = 1. Error
0000010	1010001	E = 1 (error)	AltE = 1. Error
0000011	1010000	E = 1 (error)	AltE = 1. Error
0000100	1010111	E = 1 (error)	AltE = 1. Error
0000101	1010110	E = 1 (error)	AltE = 1. Error
0000110	1010101	E = 1 (error)	AltE = 1. Error
0000111	1010100	E = 1 (error)	AltE = 1. Error
0001000	1011011	E = 1 (error)	AltE = 1. Error
0001001	1011010	E = 1 (error)	AltE = 1. Error
0001010	1011001	E = 1 (error)	AltE = 1. Error
0001011	1011000	E = 0! (dec error)	AltE = 0
0001100	1011111	E = 1 (error)	AltE = 1. Error
0001101	1011110	E = 1 (error)	AltE = 1. Error
0001110	1011101	E = 1 (error)	AltE = 1. Error

Рис. 5. Результат работы программы с альтернативным декодером

5. Выводы: в ходе данной работы был исследован типовой алгоритм формирования контрольной суммы с использованием циклических кодов для обнаружения ошибок при передаче данных по каналу, была разработана программа, наглядно демонстрирующая работу кодера и декодера для типового алгоритма.

Так же был реализован альтернативный алгоритм работы декодера. При сравнении результатов работы обоих видов декодера обнаружено совпадение решения о наличии или отсутствии ошибок. Таким образом, результаты моделирования работы типового и альтернативного декодеров совпадают с утверждением об эквивалентности обоих алгоритмов с точки зрения обнаружения ошибок, данным в лекционном материале.

6. Листинг программы:

Main.java

```
import java.util.ArrayList;

public class main {
    public static void main(String[] args) {



        String gS = "1011";
        ArrayList <Integer> gA = toArray(gS);
        Polynomial gx = new Polynomial(gA, gA.size());
        String mS = "1010";
        ArrayList <Integer> mA = toArray(mS);
        Polynomial m = new Polynomial(mA, mA.size());
        Polynomial a = Encode(gx,m);
        String eS = "0000000";
        ArrayList <Integer> eA = toArray(eS);
        Polynomial ex = new Polynomial(eA, eA.size());
        System.out.print("---------------------------\n");
        Polynomial bx = Decode(gx, ex, a);

        for (int i = 0; i < 10; i++) {
            System.out.println(i + ") \n");
            ArrayList<Integer> tmpM = Rand( size: 4);
            Polynomial tempMx = new Polynomial(tmpM, tmpM.size());
            Polynomial aTmp = Encode(gx,tempMx);
            ArrayList <Integer> tempE = Rand(a.coefficients.size());
            Polynomial <Integer> tempEx = new Polynomial<>(tempE, tempE.size());
            System.out.println("Decoder: ");
            Polynomial b = Decode(gx, tempEx, aTmp);
            System.out.println("Alternative decoder: ");
            Polynomial alt_b = AlternativeDecode( gx, tempEx, aTmp, m);
            System.out.print("-------------------------------------\n");
        }
    }
```

```
public static Polynomial Encode (Polynomial g, Polynomial m){
    System.out.println("g(x): " + g.toString());
    System.out.println("m(x): " + m.toString());
    Polynomial mxr = m.pow(g.degMax);
    System.out.println("m(x) * x^ " + g.degMax + " =" + mxr.toString());
    Polynomial cx = mxr.modgx(g);
    System.out.println("c(x): " + cx.toString());
    Polynomial ax = mxr.sum(cx);
    System.out.println("a(x): " + ax.toString());
    return ax;
}
```

```java
public static Polynomial Decode (Polynomial g, Polynomial e, Polynomial a){
    if (e.length != a.length){
        throw new RuntimeException("E has incorrect size\n");
    }
    System.out.println("e(x): " + e.toString());
    Polynomial bx = a.sum(e);
    System.out.println("b(x): " + bx.toString());
    Polynomial sx = bx.modgx(g);
    System.out.println("s(x): " + sx.toString());
    if(sx.coefficients.contains(1)){
        System.out.println("E = 1 (error)");
    }
    else{
        if (e.degMax != 0){
            System.out.println("E = 0! Decoding error\n");
        }
        else{
            System.out.print("E = 0\n");
        }
    }
    return bx;
}
```

```
public static Polynomial AlternativeDecode (Polynomial g, Polynomial e, Polynomial a, Polynomial m){
    if (e.length != a.length){
        throw new RuntimeException("E has incorrect size\n");
    }
    Polynomial bx = a.sum(e);
    ArrayList <Integer> mA = new ArrayList();
    int i = 0;
    for (i = 0; i < m.coefficients.size(); i++) {
        mA.add((Integer) bx.coefficients.get(i) % 2);
    }
    Polynomial mb = new Polynomial(mA, mA.size());
    Polynomial aAfterEncode = Encode(g, mb);
    System.out.println("aAfterEncode = " + aAfterEncode.toString());
    Polynomial cbSH = aAfterEncode.sub(mb.pow(g.degMax));
    ArrayList <Integer> cA = new ArrayList();
    int l = i;
    while ((Integer)bx.coefficients.get(l) == 0 && l < bx.coefficients.size()-1){
        l++;
    }
    for (int j = l; j < bx.coefficients.size(); j++) {...}
    Polynomial cb = new Polynomial(cA, cA.size());
    Polynomial c = a.sub(m.pow(g.degMax));

    boolean eq = true;
    while (cb.coefficients.size() != cbSH.coefficients.size()){
        if(cb.coefficients.size() > cbSH.coefficients.size()){
            cbSH.coefficients.add( index: 0, element: 0);
        }
        else{
            cb.coefficients.add( index: 0, element: 0);
        }
    }
    for (int k = 0; k < cb.coefficients.size() ; k++) {
        if (cb.coefficients.size() != cbSH.coefficients.size()){
            eq = false;
            break;
        }
        if (cb.coefficients.get(k) != cbSH.coefficients.get(k)){
            eq = false;
        }
    }
```

```
if (eq == true){
    System.out.println("AltE = 0\n");
}
else{
    System.out.println("AltE = 1. Error\n");
}
System.out.println("cb' =  " + cbSH.toString());
System.out.println("cb = " + cb.toString());
Polynomial res = new Polynomial();
return  res;
```

```java
public static ArrayList Rand(int size){
    String nums = "01";
    StringBuilder sb = new StringBuilder(size);
    for (int i = 0; i < size ; i++) {
        int index = (int) (nums.length() * Math.random());
        sb.append(nums.charAt(index));

    }
    String str = sb.toString();
    return toArray(str);
}

public static ArrayList toArray(String g){
    ArrayList<Integer> res = new ArrayList();
    for (int i = 0; i < g.length(); i++) {
        res.add(Integer.parseInt(String.valueOf(g.charAt(i))));
    }
    return res;
}
```

Polynomial.java

```java
import java.util.ArrayList;

public class Polynomial<vector> {

    public ArrayList <Integer> coefficients = new ArrayList<>();
    public int degMax;
    public int length;

    public Polynomial() {
        this.coefficients = new ArrayList<>();
        this.degMax = 0;
        this.length = 0;
    }

    public Polynomial(ArrayList<Integer> coefficients, int k){
        int temp_k = k;
        if (k <= 0){
            throw new RuntimeException("Does not exist\n");
        }
        while (temp_k > coefficients.size()){
            coefficients.add( index: 0,  element: 1);
            if((temp_k == coefficients.size()) && (coefficients.get(0) != 1)){
                coefficients.set(0, 1);
            }
        }
        this.length = k;
            if(coefficients.indexOf(1) == -1){
                this.degMax = 0;
            }
            else {
                this.degMax = coefficients.size() - coefficients.indexOf(1) - 1;
            }
        for (int i = 0; i < coefficients.size() ; i++) {
            if (coefficients.get(i) != 1 && coefficients.get(i) != 0){
                throw new RuntimeException("Error: coefficients must contain only 0 and 1 ");
            }
            if(coefficients.get(i) == 0){
                this.coefficients.add(0);
            }
            else{
                this.coefficients.add(1);
            }
        }
```

```java
public Polynomial pow (int deg){
    Polynomial result = new Polynomial();
    result.degMax = this.degMax + deg;
    for(int i = 0; i < this.coefficients.size(); i++){
        result.coefficients.add(this.coefficients.get(i));
    }
    for (int i = 0; i < deg; i++){
        result.coefficients.add(0);
    }
    result.length = result.coefficients.size();
    return result;
}
```

```java
public Polynomial sub (Polynomial b){
    boolean eq = true;
    for (int i = 0; i < b.coefficients.size() ; i++) {
        if (this.coefficients.get(i) != b.coefficients.get(i)){
            eq = false;
        }
    }
    Polynomial result = new Polynomial();
    if (eq == true){
        return result;
    }
    for (int i = 0; i < this.coefficients.size() ; i++) {
        if (i < (this.coefficients.size() - b.coefficients.size())){
            result.coefficients.add(this.coefficients.get(i));
        }
        else{
            result.coefficients.add(Math.abs(this.coefficients.get(i) -
                    (int)b.coefficients.get(i -(this.coefficients.size() - b.coefficients.size())) %2));
        }
    }
    for (int i = 0; i < result.coefficients.size(); i++){
        if ((int)result.coefficients.get(i) != 0){
            eq = false;
        }
    }
    if (eq == true){
        return  result;
    }
    int k = 0;

    while ((int)result.coefficients.get(k) == 0){
        k++;
    }
    ArrayList<Integer> tmp = new ArrayList<>();
    if (k < result.coefficients.size()){
        for (int i = k; i < result.coefficients.size(); i++) {
            tmp.add((Integer) result.coefficients.get(i));
        }
        result.coefficients = tmp;
    }
    result.length = tmp.size();
    result.degMax = tmp.size() - 1;
    return result;
```

```java
public Polynomial sum (Polynomial b){
    Polynomial result = new Polynomial();
    boolean eq = true;
    result.degMax = Math.max(this.degMax, b.degMax);
    for (int i = 0; i < this.coefficients.size() ; i++) {
        if (i < (this.coefficients.size() - b.coefficients.size())){
            result.coefficients.add(this.coefficients.get(i));
        }
        else{
            result.coefficients.add(Math.abs(this.coefficients.get(i) +
                    (int)b.coefficients.get(i -(this.coefficients.size() - b.coefficients.size())) %2));
        }
    }
    result.length = result.coefficients.size();
    for (int i = 0; i < result.coefficients.size() ; i++) {
        if ((int)result.coefficients.get(i) != 0){
            eq = false;
        }
    }
    if (eq == true){
        result.degMax = 0;
    }
    return result;
}

public Polynomial modgx(Polynomial b){
    Polynomial result = new Polynomial();
    Polynomial a_tmp = this;
    while(a_tmp.degMax >= b.degMax){
        int deg = a_tmp.degMax - b.degMax;
        Polynomial tmp = b.pow(deg);
        result = a_tmp.sub(tmp);
        a_tmp = result;
    }
    return result;
}
```

```java
@Override
public String toString (){

    StringBuilder result = new StringBuilder();

    if (!this.coefficients.contains(1)){
        result.append("0\n");
    }
    int p = 0;
    for (int i = 0; i < this.coefficients.size() ; i++) {
        if (this.coefficients.get(i) % 2 != 0){
            if (i == 0 || p == 0){
                result.append("x^" + (this.length - i - 1));
                p++;
            }
            else{
                result.append(" + x^" + (this.length - i - 1));
            }
        }
    }
    result.append("\n");
    for (int i = 0; i < this.coefficients.size(); i++) {
        result.append(this.coefficients.get(i) % 2);

    }
    result.append("\n");
    return result.toString();

}
```