Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Кафедра РТС

Отчет по лабораторной работе №3

«ИССЛЕДОВАНИЕ СВОЙСТВ КОДОВ БОУЗА – ЧОУДХУРИ – ХОКВИНГВЕМА»

Выполнил:

ст.гр.240102
shlom41k

Проверил:

Панькова В.В.

Минск 2015

Цель работы

1. Изучить методы помехоустойчивого кодирования и декодирования информации с помощью циклических блочных кодов Боуза – Чоудхури – Хоквингвема.

2. Исследовать свойства кодов Боуза – Чоудхури – Хоквингвема (БЧХ), изучить связь их корректирующей способности с конечными полями.

3. Приобрести навыки построения алгоритмов и программ кодирования и декодирования информации.

Выполнение работы

Алгоритм Питерсона

Алгебра вычислений:

```
with(linalg): with(LinearAlgebra):with(PolynomialTools): with(ArrayTools):
```

Расширенные поля

Рассмотрим случай, когда p = 2.

```
> p:=2;
> m:=6;
> n:=p^m-1;
```

Задание полинома расширенного поля

```
> polynoms := convert(Column(convert((Factors(x^n-1) mod p)[2], matrix), [1]), vector);
> lenght := Size(polynoms)[2];
> for i to lenght do
>  pg:=polynoms[i];
>  if degree(pg, x) = m then
>   if (Primitive(pg) mod p) = convert( true, 'truefalse' ) then
>    break;
>   fi:
>  fi:
> od:
> printf("примитивный неприводимый полином степени m = %g найден: pg =%s", m,
convert(pg,string));
> pg;
```

Проверка

```
> Primitive(pg) mod 2;
```

Определим alpha - корень полинома f(z) , m -> p^m

```
> alias(alpha=RootOf(pg) mod p):
> Q:=p^m;
```

Массив степеней **powers**

```
> powers:=vector(Q-1);
```

Формирование элементов поля через степени alpha^(i-1)

```
> for i to Q-1 do powers[i]:=evala(alpha^(i-1)) mod p od:
> print(powers);
> inverses:=vector(Q-1);
```

Формирование обратных элементов поля 1/powers[i]

```
> for i to Q-1 do inverses[i]:=evala(1/powers[i])mod p od:
> print(inverses);
```

Табличное задание

```
> read "MakeTable.m";
> read "Eform.m";# Оператор табличных вычислений
> FieldTable := Matrix(p^m,p):
> MakeTable(p^m,p,alpha);
```

Генераторный полином

Найдем генераторный полином БЧХ кода. Предварительно определим минимальный полином заданного поля. Предположим, что циклический код формируется по модулю (x^n - 1), т.е. кодовое слово имеет длину n= p^m-1. Код поддерживает расширенное поле GF(p^m). Проведем факторизацию полинома (x^n -1)

Чтение оператора, вычисляющего циклотомический класс s, для n и p

```
>  read "CyclotomicCoset.m";read"Eform.m";
```

Пример s=1,n=63,p=2

```
> CyclotomicCoset(1,n,p);
> read "AllCyclotomicCosets.m";
```

Чтение оператора, вычисляющего все циклотомические классы

```
> AllCyclotomicCosets(n,p);
```

Формирование минимальных многочленов через степени alpha^i, i - числа циклотомического класса

Программа **minpoly** использует результат вычислений **AllCyclotomicCosets(n,p)** и формулу для вычисления минимального полинома

```
> minpolynom:=proc(s,n,p)
>  local Ks,ml,m,l,a;
>  Ks:=CyclotomicCoset(s,n,p);
>  for l from 2 to nops(Ks) do
>   ml[1]:=(x-alpha^s);
>   a[l]:=op(l,Ks);
>   ml[l]:=sort(simplify(ml[l-1]*(x-alpha^a[l]))) mod p;
>  od:
>  m[s]:=ml[nops(Ks)];
>  return(sort(m[s]));
> end proc:
```

Пример

```
> Ks:=CyclotomicCoset(5,n,p);
> sort(minpolynom(5,n,p));
> sort(minpolynom(3,n,p));
```

БЧХ коды

Сформируем полином, исправляющий 3 ошибки

```
> t:=15;
> k:=n-m*t;
> r := n-k;
```

Сформируем требуемое количество 2t минимальных полиномов **mpl**

```
> for i2 from 1 to 2*t do mpl[i2]:=minpolynom(i2,n,p); od;
```

Используя оператор вычисления НОК ->Lcm() вычислим генераторный полином g3. Зададим список **pd** требуемых минимальных полиномов

```
> pd:=seq(mpl[i],i=1..2*t);
```

Скопируем полиномы списком и вставим этот список в оператор **Lcm**, который вычисляет НОК списка полиномов. Получим генераторный (порождающий) полином g3

```
> g := sort(Lcm(pd) mod p, x);
> k := n-degree(g,x);
```

Кодирование и моделирование канала

Кодирование

Сформируем информационный вектор. Случайное задание инф. полинома a

```
> a:= RandomTools[Generate](polynom(integer(range = 0..1), x, degree = k-1));
```

Несистематический код c(x)=a(x)g(x)

```
> cn:=sort(expand(simplify(g*a)) mod p);
```

Систематический код . Программа вычислений

words:=a(x)x^(n-k)

```
> words:=sort(expand(x^(n-k)*a));
```

Вычисление остатка **ww:= rem{a(x)x^(n-k)/g(x))**

```
> ww:=Rem(words,g,x) mod p;
```

Формирование кодового слова

```
> cs:=words+ww mod p;
```

Выбор систематического или несистематического кода

```
> c := cn;
```

Модель канала - через полином ошибок err(x)

Генератор случайных полиномов ошибок

```
> for j from 1 to 10 do  er[j]:=randpoly(x,terms=t+3,degree=(p^m-1)) mod 2;od;
```

Выбор вектора ошибок

```
> err:= er[1];
```

Аддитивная смесь кода и ошибок

```
> rws:=sort(c+err mod p);
```

Принимаемый сигнал

```
> y:=sort(c+err mod p);
```

Декодирование БЧХ кода

Алгоритм Питерсона-Горенштейна-Цирлера (ПГЦ)

1. Вычисляются компоненты синдрома. Задается величина v = t.
2. Составляется матрица синдромов.
3. Вычисляется определитель матрицы. Если $\Delta = 0$, то матрица является сингулярной, и обратной матрицы не существует. В этом случае изменяется размер матрицы v:= (v - 1) до тех пор, пока определитель не станет отличным от нуля. После чего повторяется второй шаг.
4. Определяются коэффициенты полинома локаторов ошибок
5. Определяются корни $\sigma(x) = 0$ и через их инверсию вычисляют значения X_i .
6. Оцениваются величины ошибок $\{Q_i\}$.
7. Корректируют принятый вектор, подставляя вычисленные значения Qi

Вычисление вектора синдромов

Значения синдрома вычисляются путем подстановки в полином y(x) значения соответствующего элемента поля

```
> sx:=array(1..2*t);
> for i from 1 to 2*t do sx[i]:=Eval(y,x=alpha^i) mod p;od;
```

Нормализация путем табличного сложения элементов поля

```
> for i from 1 to 2*t do sxh[i]:=Eform(Normal(sx[i]) mod p);od;
```

Вычисление синдрома по схеме Горнера

Maple-оператор имеет обозначение **horner**
Моделирование схемы Горнера

```
> hor:=convert(y,horner,x);
```

Табличные вычисления по правилам поля для получения компактного результата

```
> for i from 1 to 2*t do shor[i]:= Eform(Normal(expand(eval(hor, x=alpha^i))) mod p);od;
```

Сравниваем полученные результаты

<u>Полином локаторов ошибок</u>

Составление матрицы сдвигов компонент синдрома ошибок A

```
> for j from 0 to t-1 do rw[j]:=seq(sx[i+j],i=1..t); od:
> M:=matrix(t,t,[seq(rw[i],i=0..t-1)]);
```

Определение количества реальных ошибок ***v*** и формирование матрицы сдвигов компонент синдромов ***Ag*** с отличным от нуля определителем ***Dg***. Предварительное вычисление определителя по модулю полиномы поля **pg**

```
>  Dg := Rem(det(M),pg,x) mod p;
```

Определитель без модуля

```
> det(M) mod p;
```

Задание исходных данных программе определения числа реальных ошибок

```
> v := t:
Ag:=M:
```

Вычисление числа реальных ошибок и матрицы ***Ag*** путем последовательного выбора подматрицы ***Ag*** матрицы **M**. Подматрица должна иметь отличный от нуля определитель, что гарантирует существование обратной матрицы. Размер подматрицы ***Ag*** определяет число реальных ошибок в канале ***v*** .

```
> while Dg = 0 and v > 1 do
>  Ag := delshlomchikrows(Ag,v..v);
>  Ag := delcols(Ag,v..v);
>  Dg:= Rem(det(Ag),pg,x) mod p;
>  v := v-1:
>  print(Ag);
> od:
> if Dg <> 0 then printf("Количество ошибок равно %g", v) fi:
```

Итоговая форма матрицы A

```
> print(Ag);
```

Задание вектора значений синдрома для правой части системы уравнений

```
> hs:=vector(v,[seq(sx[v+i],i=1..v)]);
```

Решение системы линейных уравнений алгоритма Питерсона. Решение дает коэффициенты полинома локаторов ошибок. Используется оператор Maple **Linsolve**

```
> co:=Linsolve(Ag,hs) mod p;
```

Задание полинома позиций ошибок **sig**

```
> sig:=sort(z^v+sum(z^i1*co[i1+1],i1=0..v-1));
```

Факторизация полинома **sig** для нахождения его корней

```
> Factor(sig) mod p;
```

Отсюда корни **alpha^3** и **alpha^3+alpha^2+1.**
Другой способ нахоэдения корней полиномы. Воспользуемся оператором Maple **Roots**

```
> ro:=Roots(sig) mod p;
```

Результаты вычислений по двум способам должны совпасть

Нахождение корней полинома позиций ошибок. Зададим вектор **erp**

```
> erp:=vector(v);
```

Пересчет значений корней полинома позиций ошибок в позиции ошибок

```
> for i from 1 to v do
  for j from 1 to n do
    if (ro[i,1]=powers[j]) then erp[i]:=j-1 fi;
  od;od;
```

Вывод значений позиций ошибок

```
> print(erp);
```

Формируем полином оценок ошибок **erre**

```
> erre:=sort(sum(x^erp[i1+1],i1=0..v-1));
```

Корректируем принятый сигнал **y**, путем его сложения по mod 2 с оценкой ошибок **erre**. Вычисляем оценку переданного кодового слова **ec**

```
> ec:=y+erre mod p;
```

Проверяем - декодированное кодовое слова должно делится на порождающий полином без остатка

```
> rem(ec,g,x) mod p;
```

Сравнение декодированного и исходных кодов

```
> c;
```

Программа сравнения

```
> if (c=ec) then print(YES) else print(NO);fi;
```

Вычисление синдрома декодированного кодового слова

```
> sxe:=array(1..2*(t+3));
```

```
> for i from 1 to 2*t do sxe[i]:=Eval(ec,x=alpha^i) mod p;od;
```

Алгоритм Берлекемпа-Месси

```
> S:=convert(sx, vector);
> print(S);
```

Функция вычисления переменной T

```
> Tproc:=proc(B, A, dr, p)
  local Tx;
  Tx:=(A-dr*x*B) mod p;
end proc:
```

Функция вычисления невязки

```
> Neviazka:=proc(r, S, A, L, p, m)
local dr, j, a, d;
  dr:=S[r];
  for j from 1 to L do
    a:=coeff(A,x,j);
    dr:=(dr+S[r-j]*a) mod p;
  od:
  dr:=evala(dr) mod p;
end proc:
```

Проверка условий выхода из алгоритма

```
> iffer:=proc(B, dr, p, A, L, r)
local list, T, Bx, Ax, Lr, dri;
  Ax:=A;
  Bx:=B;
  Lr:=L;
  if dr <> 0 then
    dri:=evala(1/dr) mod p;
  fi:
  if dr=0 then
    Bx:=(Bx*x) mod p;
  else
    T:=Tproc(Bx, Ax, dr, p);
    if ((2*Lr)<=(r-1)) then
      Bx:=(Ax*dri) mod p;
      Ax:=T;
      Lr:=r-Lr;
    else
      Ax:=T;
      Bx:=(Bx*x) mod p;
    end if;
  end if;
  list:=[sort(simplify(Ax),x), sort(simplify(Bx),x), Lr];
end proc:
```

Задание начальных значений переменных

```
> A:=1;
```

```
> L:=0;
> B:=1;
```

Вычисление полинома локаторов ошибок при помощи алгоритма Берлекемпа-Месси

```
> for r from 1 to 2*t do
  printf("Номер итерации r = %d\n", r);
  dr:=sort(Neviazka(r, S, A, L, p, m), alpha):
  printf("Невязка dr = %s\n", convert(dr, string));
  N:=iffer(B, dr, p, A, L, r):
  A:=N[1] mod p:
  d:=degree(A,x):
  Ax:=0:
  al:=0:
  for i from 0 to d do
    for j from 1 to Q-1 do
      if (powers[j]=coeff(A,x,i)) then
        Ax:=Ax + (evala(alpha(j-1)) mod p)*x^i;
      fi:
    od:
  od:
  printf("Текущее значение полинома A = %s\n", convert(Ax, string));
  B:=N[2] mod p:
  d:=degree(B,x):
  Bx:=0:
  al:=0:
  for i from 0 to d do
    for j from 1 to Q-1 do
      if (powers[j]=coeff(B,x,i)) then
        Bx:=Bx + (evala(alpha^(j-1)) mod p)*x^i;
      fi:
    od:
  od:
  printf("Значение переменной B = %s\n", convert(Bx, string));
  L:=N[3]:
  printf("Текущая длина регистров сдвига L = %d\n",L);
  print("---------------------------");
od:

> print(A);
> Factor(A) mod p;
> t:=degree(A,x);
> rbma:=Roots(A) mod p;
> erpbma:=vector(t);
> for i from 1 to t do
    for j from 1 to Q-1 do
      if (rbma[i,1]=inverses[j]) then erpbma[i]:=j-1; j:=Q fi;
    od;od;
> print(erpbma);
```

Определение полинома ошибок

```
> errebma:=sort(sum(x^erpbma[i1+1],i1=0..v-1));
```

Полином ошибок вычисленный методом ПГЦ

```
> erre;
```

Сравнение результатов

```
> if ((errebma - erre) mod p) = 0 then
>  printf("\tРезултаты совпадают\n");
> else
>  printf("\tРезултаты не совпадают\n");
> fi:
```

Вывод

В данной лабораторной работе были изучены методы помехоустойчивого кодирования и декодирования информации с помощью циклических блочных кодов Боуза – Чоудхури – Хоквингвема, исследованы свойства кодов (БЧХ), связь их корректирующей способности с конечными полями. Приобрели навыки построения алгоритмов и программ кодирования и декодирования информации.