1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 4**

По дисциплине «Основы информационной безопасности»

1. Выполнил
2. Студент гр. 13508/13 А.Э. Палёный

\_\_\_\_\_\_\_\_\_\_

1. Проверил
2. Преподаватель

\_\_\_\_\_\_\_\_\_\_ М.О.Калинин

1. Санкт-Петербург
2. 2016

Оглавление

[Цель работы: 3](#_Toc450574482)

[Решаемые задачи: 3](#_Toc450574483)

[Ход работы: 4](#_Toc450574484)

[Вывод: 6](#_Toc450574485)

[Ответы на контрольные вопросы: 18](#_Toc450574486)

# Цель работы:

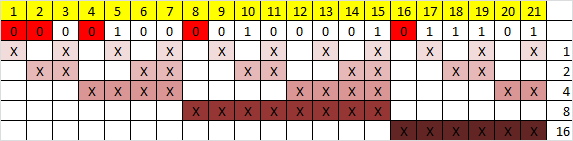
Ознакомление с методом Хемминга помехоустойчивости кодирования, позволяющим обнаруживать и автоматически исправлять ошибки, возникающие при хранении и передачи информации.

# Решаемые задачи:

1. Научиться шифровать сообщения кодами Хемминга
2. Создать программу кодер, реализующую шифрование кодами Хемминга
3. Реализовать программу, расшифровывающую зашифрованные сообщения
4. Протестировать программу

# Ход работы:

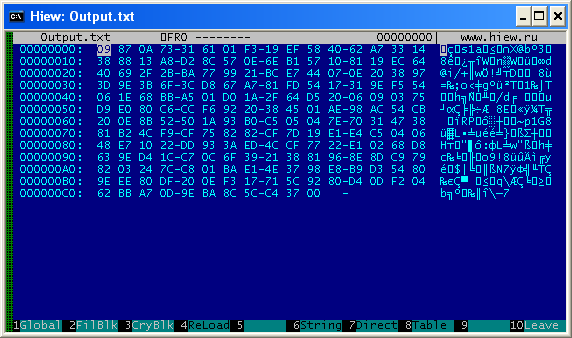
Теперь необходимо вычислить значение каждого контрольного бита. Значение каждого контрольного бита зависит от значений информационных бит (как неожиданно), но не от всех, а только от тех, которые этот контрольных бит контролирует. Для того, чтобы понять, за какие биты отвечает каждых контрольный бит необходимо понять очень простую закономерность: контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N. Не очень понятно, но по картинке, думаю, станет яснее:



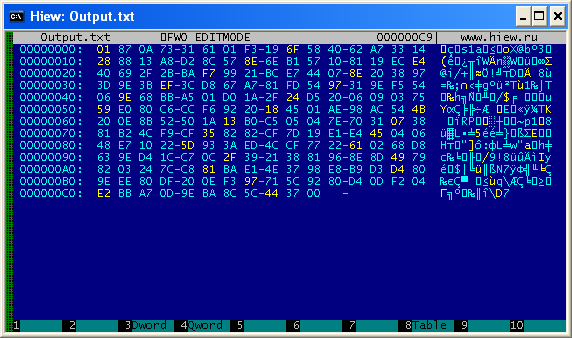
Здесь знаком «X» обозначены те биты, которые контролирует контрольный бит, номер которого справа. То есть, к примеру, бит номер 12 контролируется битами с номерами 4 и 8. Ясно, что чтобы узнать какими битами контролируется бит с номером N надо просто разложить N по степеням двойки.

А теперь перейдём к нашей программе.

Мы кодируем текст:  
«Have you ever dreamed of running a bookshop? You can have a go for a week at the Open Book Store in Wigton, Scotland. This is massage who need to encoded.»



Имитируем однобитные ошибки



В результате мы получили сообщение:

«Have you ever dreamed of running a bookshop? You can have a go for a week at the Open Book Store in Wigton, Scotland. This is massage who need to encoded.»

# Вывод:

В ходе работы, мы узнали, как кодировать сообщение, повышая при этом его надёжность. Также, если этот метод работает только для однобитных ошибок.

Листинг кодера:  
#include <stdio.h>

#include <Windows.h>

#include <malloc.h>

#include <math.h>

void GetText(char \*\*text, int \*length)

{

FILE \*Input=fopen("Input.txt","rb+");

if(!Input) exit(0);

fgetc(Input);

fseek(Input, 0, SEEK\_END);

\*length=ftell(Input);

fseek(Input, 0, SEEK\_SET);

\*text=(char\*)malloc(sizeof(char)\*(\*length+1));

fread(\*text, sizeof(char), \*length, Input);

\*(\*text+\*length)=0;

fclose(Input);

}

void ByteToBit(char \*\*res, char data, int pos)

{

int i;

for (i=0; i < 8; i++)

{

if ((data >> i & 1) == 1) res[0][pos+7-i]=1;

else res[0][pos+7-i]=0;

}

}

void ChangePosition(char \*Temp, char \*Dump, int max)

{

int i,step=1, service=1;

int valid, dis;

int number=0;

int dump, trash;

for(i=0; i<max; i++)

{

if((i+1)==step)

{

Temp[i]=0;

step\*=2;

service++;

}else{ Temp[i]=Dump[number]; number++;}

}

}

void ChangeValue(char \*Temp, int max)

{

int i,step=1, service=1;

int valid, dis;

int number=0;

int dump, trash;

for(i=0; i<max; i++)

{

if((i+1)==step)

{

dump=0;

for(valid=i; valid < max;)

{

for(trash=0; trash<i+1; trash++)

{

if((valid+trash) > (max-1)) break;

dump+=Temp[valid+trash];

}

valid+=2\*(i+1);

}

Temp[i]=(dump % 2) ? 1:0;

step\*=2;

service++;

}

}

}

char BitToByte(char \*data)

{

int i;

char res=0;

for (i=0; i < 8; i++)

if(data[7-i] == 1)

res+=pow(2.0, i);

return res;

}

void WriteToFile(char \*Temp, int length)

{

char \*data=(char\*)malloc(8);

char str;

int i, bug;

FILE \*Output=fopen("Output.txt", "wb+");

if(!Output) exit(0);

for(i=0; i<length;)

{

for(bug=0; bug<8; bug++)

{

if((i+bug)>length) data[bug]=0;

else data[bug]=Temp[i+bug];

printf("%d",data[bug]);

}

printf("\n");

str=BitToByte(data);

fwrite(&str, 1, 1, Output);

i+=8;

}

fclose(Output);

}

void Encoding(char \*text, int length, int CountBit, int ReadBit)

{

int i,pos=0, GlobalPos, ser;

int max=CountBit+ReadBit\*8;

char \*Dump=(char\*)malloc(ReadBit\*8);

char \*Temp=(char\*)malloc(max);

char \*Output=(char\*)malloc(max);

for(GlobalPos=0; GlobalPos < length; GlobalPos+=ReadBit)

{

for(i=0; i!=ReadBit; i++)

{

ByteToBit(&Dump, text[i+GlobalPos], pos);

pos+=8;

}

pos=0;

ChangePosition(Temp, Dump, max);

ChangeValue(Temp, max);

Output=(char\*)realloc(Output, (GlobalPos+ReadBit)/ReadBit\*max+1);

ser=(GlobalPos/ReadBit)\*max;

for(i=0; i<max; i++)

{

Output[ser+i]=Temp[i];

printf("%d",Temp[i]);

}

printf("\n");

Output[ser+i]=5;

}

printf("\n");

for(i=0; i<GlobalPos/ReadBit\*max; i++)

printf("%d",Output[i]);

WriteToFile(Output, GlobalPos/ReadBit\*max);

}

int main()

{

int CountBit,ReadBit,Step;

char \*text;

int length;

printf("Enter the count of bits\n");

scanf("%d",&CountBit);

for(Step=1; 1!=CountBit; Step++) CountBit/=2;

CountBit=Step;

Step=0;

for(ReadBit=4; ReadBit!=CountBit; ReadBit++) Step++;

ReadBit=pow(2,Step);

GetText(&text, &length);

Encoding(text, length, CountBit, ReadBit);

getchar();

}Листинг декодера:

#include <stdio.h>

#include <Windows.h>

#include <malloc.h>

#include <math.h>

void GetText(char \*\*text, int \*length)

{

FILE \*Input=fopen("Output.txt","rb+");

if(!Input) exit(0);

fgetc(Input);

fseek(Input, 0, SEEK\_END);

\*length=ftell(Input);

fseek(Input, 0, SEEK\_SET);

\*text=(char\*)malloc(sizeof(char)\*(\*length+1));

fread(\*text, sizeof(char), \*length, Input);

\*(\*text+\*length)=0;

fclose(Input);

}

int FindError(char \*Temp, int max, int val)

{

int i,trash,dump=0, flag=0;

for(i=val-1; i<max; i+=2\*val)

{

for(trash=0; trash<val; trash++)

{

if((trash+i)>=max) break;

if(flag==0){ dump+=0; flag++;}

else dump+=Temp[i+trash];

}

}

dump=(dump % 2) ? 1:0;

if(Temp[val-1] == dump) return 0;

else return 1;

}

void Proverka(char \*Temp, int max, int CountBit)

{

int i,var=1,Restore=0;

for(i=0; i<CountBit; i++)

{

if(FindError(Temp, max, var)){

printf("There are error in %d\n",i+1);

Restore+=var;

}

var\*=2;

}

if(Restore!=0) Temp[Restore-1]=Temp[Restore-1] ^ 1;

}

void ByteToBit(char \*\*res, char data, int pos)

{

int i;

for (i=0; i < 8; i++)

{

if ((data >> i & 1) == 1) res[0][pos+7-i]=1;

else res[0][pos+7-i]=0;

}

}

char BitToByte(char \*data)

{

int i;

char res=0;

for (i=0; i < 8; i++)

if(data[7-i] == 1)

res+=pow(2.0, i);

return res;

}

void ChangePosition(char \*Temp, char \*Dump, int max)

{

int i,step=1;

int number=0;

for(i=0; i<max; i++)

{

if((i+1)==step) step\*=2;

else { Dump[number]=Temp[i]; number++;}

}

}

void WriteToFile(char \*Temp, int length)

{

char \*data=(char\*)malloc(8);

char str;

int i, bug;

FILE \*Output=fopen("Input.txt", "wb+");

if(!Output) exit(0);

for(i=0; i<length;)

{

for(bug=0; bug<8; bug++)

{

if((i+bug)>length) data[bug]=0;

else data[bug]=Temp[i+bug];

printf("%d",data[bug]);

}

str=BitToByte(data);

fwrite(&str, 1, 1, Output);

i+=8;

}

fclose(Output);

}

void Encoding(char \*text, int length, int CountBit, int ReadBit)

{

int i,pos=0, GlobalPos, ser;

int max=CountBit+ReadBit\*8;

char \*Dump=(char\*)malloc(ReadBit\*length\*8);

char \*Back=(char\*)malloc(ReadBit\*8);

char \*Temp=(char\*)malloc(max);

char \*Output=(char\*)malloc(max);

for(i=0; i!=ReadBit\*length; i++)

{

ByteToBit(&Dump, text[i], pos);

pos+=8;

}

pos=0;

length=length\*8/max\*ReadBit;

for(GlobalPos=0; GlobalPos < length; GlobalPos+=ReadBit)

{

for(i=0; i<max; i++)

{

Temp[i]=Dump[i+pos];

printf("%d",Dump[i+pos]);

}

printf("\n");

pos+=max;

Proverka(Temp, max, CountBit);

ChangePosition(Temp, Back, max);

Output=(char\*)realloc(Output, (GlobalPos+ReadBit)\*8);

ser=GlobalPos\*8;

for(i=0; i<8\*ReadBit; i++)

{

Output[ser+i]=Back[i];

printf("%d",Back[i]);

}

printf("\n");

Output[ser+i]=5;

}

WriteToFile(Output, GlobalPos\*8);

}

int main()

{

int CountBit,ReadBit,Step;

char \*text;

int length;

printf("Enter the count of bits\n");

scanf("%d",&CountBit);

for(Step=1; 1!=CountBit; Step++) CountBit/=2;

CountBit=Step;

Step=0;

for(ReadBit=4; ReadBit!=CountBit; ReadBit++) Step++;

ReadBit=pow(2,Step);

GetText(&text, &length);

Encoding(text, length, CountBit, ReadBit);

getchar();

}

# Ответы на контрольные вопросы:

1. Расстоянием Хэмминга двух строк a и b одинаковой длины называется количество позиций, в которых эти строки различаются
2. Они позволяют исправлять многобитные ошибки. Но только до 25 процентов ошибок
3. Положим  a_{i} \in GF(q) , (i = 1,2,\ldots,k-1) , \alpha \in GF(q) — примитивный элемент поля GF(q), и пусть 
   a = (a_0,a_1,\ldots,a_{k-1})
    — вектор информационных символов, а значит a(x) = a_0 + a_{1}x + \ldots + a_{k-1}x^{k-1} — информационный многочлен. Тогда вектор u = (a(1),a(\alpha),\ldots,a(\alpha^{q-2})) есть вектор кода Рида — Соломона, соответствующий информационному вектору a. Этот способ кодирования показывает, что для кода РС вообще не нужно знать порождающего многочлена и порождающей матрицы кода, достаточно знать разложение[поля](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) GF(q) по примитивному элементу \alpha и размерность кода k (длина кода в этом случае определяется как n = q - 1). Все дело в том, что за разностью n-k полностью скрывается порождающий многочлен g(x) и кодовое расстояние.
4. На надёжность передачи. Т.е. чем меньше размер кодового слова, тем надёжнее передача. Но, правда увеличивается размер сообщения.