МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №3**

по дисциплине: Теория информации

тема: «Исследование возможности применения методов энтропийного

кодирования для обработки двоичных последовательностей»

Выполнил: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил: Твердохлеб Виталий

Викторович

Белгород 2025 г.

Ход работы

**Задание 1.**

Открыть файл Лабораторная работа 3 (задание).txt. Рассмотреть возможность построения кода по методам Хаффмана и Шеннона-Фано для бинарной последовательности. Сделать выводы.

Бинарная последовательность уже ужата, в бинарной последовательности используются только два символа, значит для кодирования символа нужен один бит, это уже минимум.

**Задание 2.**

Рассмотреть варианты обработки цепочек символов, а именно:

- 2 символа;

- 4 символа;

- 8 символов.

Для этого разработать консольное приложение, разбивающее сплошной массив символов на цепочки заданной длины.

Код:

int main(){  
 FILE \*f = fopen("text.txt", "r");  
 int size;  
 scanf("%d", &size);  
 char letters[size+1];  
 char D[4120][size+1];  
 int count = 0;  
 while (fread(letters, 1, size, f) == size) { // Считываем по 2 символа  
 letters[size] = '\0';  
 for(int i = 0; i < size+1; i++){  
 D[count][i] = letters[i];  
 }  
 count++;  
 }  
  
 fclose(f);

for(int i = 0; i < count; i++){  
 printf("%s ", D[i]);  
 }

return 0;  
}

**Вывод программы:**

**Изображение выглядит как снимок экрана, текст, шаблон, Симметрия

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как снимок экрана, текст, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.**

**Задание 3, 4 и 5**

Рассматривая каждую цепочку (2, 4 и 8 символов длиной) как отдельный символ, построить коды по методу Хаффмана и Шеннона-Фано. Составить последовательности из полученных кодов символов для каждого случая. По результатам работы в п.3 сделать выводы по поводу полученных результатов для каждого из методов (простота, скорость, полученные результаты (рассчитать коэффициенты сжатия)).

**Код, метод Шеннона-Фано:**

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
typedef struct node{  
 int count;  
 char letter[15];  
 char code[30];  
}node;  
void sort(int n, node \*array[n]){  
 for(int i = 0; i < n; i++){  
 int is\_sort = 1;  
 for(int j = n - 1; j > i; j--){  
 if(array[j]->count > array[j-1]->count){  
 node \*temp = array[j];  
 array[j] = array[j-1];  
 array[j-1] = temp;  
 is\_sort = 0;  
 }  
 }  
 if(is\_sort){  
 break;  
 }  
 }  
}  
void putChar(char \*arr, char c){  
 char \*begin = arr;  
 while (\*begin!='\0'){  
 begin++;  
 }  
 \*begin = c;  
 \*(++begin) = '\0';  
}  
void fano(int size, node \*array[size]){  
 int sum1 = 0, sum2 = 0;  
 node \*part1[size];  
 node \*part2[size];  
 int size1 = 0;  
 int size2 = 0;  
 for(int i = 0; i < size; i++){  
 if(sum1 < sum2){  
 part1[size1++] = array[i];  
 sum1+=array[i]->count;  
 putChar(array[i]->code, '1');  
 }  
 else{  
 part2[size2++] = array[i];  
 sum2+=array[i]->count;  
 putChar(array[i]->code, '0');  
 }  
 }  
 if(size1 > 1){  
 fano(size1, part1);  
 }  
 if(size2 > 1){  
 fano(size2, part2);  
 }  
}  
  
int main(){  
 clock\_t start\_time = clock();  
 FILE \*f = fopen("text.txt", "r");  
 int size;  
 scanf("%d", &size);  
 char letters[size+1];  
 char D[4120][size+1];  
 int count = 0;  
 while (fread(letters, 1, size, f) == size) { // Считываем по 2 символа  
 letters[size] = '\0';  
 for(int i = 0; i < size+1; i++){  
 D[count][i] = letters[i];  
 }  
 count++;  
 }  
 fclose(f);  
 char \*begin;  
 node \*data[count];  
 int size\_data = 0;  
 for(int i = 0; i < count; i++){  
 int flag = 0;  
 for(int j = 0; j < size\_data; j++){  
 int flag\_isEq = 1;  
 begin = D[i];  
 char \*b = data[j]->letter;  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag\_isEq = 0;  
 break;  
 }  
 begin++;  
 b++;  
 }  
 if(flag\_isEq){  
 flag = 1;  
 data[j]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[size\_data] = (node \*)malloc(sizeof(node));  
 char \*b = data[size\_data]->letter;  
 begin = D[i];  
 while (\*begin!='\0'){  
 \*b = \*begin;  
 b++;  
 begin++;  
 }  
 \*b = '\0';  
 data[size\_data]->count = 1;  
 data[size\_data]->code[0] = '\0';  
 size\_data++;  
 }  
 }  
 sort(size\_data, data);  
 fano(size\_data, data);  
  
 char answer[count \* size \* 10];  
 int size\_ans = 0;  
 char \*ansBg = answer;  
 for(int i = 0; i < count; i++){  
 for(int j = 0; j < size\_data; j++){  
 int flag = 1;  
 char \*b = data[j]->letter;  
 begin = D[i];  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag = 0;  
 }  
 begin++;  
 b++;  
 }  
 if(flag){  
 begin = data[j]->code;  
 while (\*begin!='\0'){  
 \*ansBg = \*begin;  
 ansBg++;  
 size\_ans++;  
 begin++;  
 }  
 }  
 }  
 }  
 \*ansBg = '\0';  
 printf("%s", answer);  
 printf("\nK = %.1f", 4210.0/size\_ans);  
 for(int i = 0; i < size\_data; i++){  
 free(data[i]);  
 }  
 clock\_t end\_time = clock();  
 double time\_taken = (double)(end\_time - start\_time) / **CLOCKS\_PER\_SEC**;  
 printf("\nTime: %f\n", time\_taken);  
 return 0;  
}

**Выводы программы:Изображение выглядит как снимок экрана, текст, шаблон, шов

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана, шаблон, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.**

**Метод Хаффмана:**

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
typedef struct node{  
 int count;  
 char letter[15];  
 char code[30];  
 struct node\* left;  
 struct node\* right;  
}node;  
void sort(int n, node \*\*array){  
 for(int i = 0; i < n; i++){  
 int is\_sort = 1;  
 for(int j = n - 1; j > i; j--){  
 if(array[j]->count > array[j-1]->count){  
 node \*temp = array[j];  
 array[j] = array[j-1];  
 array[j-1] = temp;  
 is\_sort = 0;  
 }  
 }  
 if(is\_sort){  
 break;  
 }  
 }  
}  
void haffman(node \*let, char\* code, char \*endCode){  
 if(let->letter[0] == '\0'){  
 \*endCode = '0';  
 \*(endCode+1) = '\0';  
 haffman(let->left, code, endCode+1);  
 \*endCode = '1';  
 \*(endCode+1) = '\0';  
 haffman(let->right, code, endCode+1);  
 }  
 else{  
 char \*begin = let->code;  
 char \*begin\_code = code;  
 while (\*begin\_code!='\0'){  
 \*begin = \*begin\_code;  
 begin++;  
 begin\_code++;  
 }  
 \*begin = '\0';  
 }  
}  
void huffmanTreeToArray(node \*\*data, int\* count, node \*root){  
 if(root->letter[0] != '\0'){  
 data[\*count] = root;  
 (\*count)+=1;  
 }  
 else{  
 huffmanTreeToArray(data, count, root->left);  
 huffmanTreeToArray(data, count, root->right);  
 }  
}  
void free\_data(node \*root){  
 if(root->letter[0] == '\0'){  
 free\_data(root->left);  
 free\_data(root->right);  
 }  
 free(root);  
}  
  
  
int main(){  
 clock\_t start\_time = clock();  
 FILE \*f = fopen("text.txt", "r");  
 int size;  
 scanf("%d", &size);  
 char letters[size+1];  
 char D[4120][size+1];  
 int count = 0;  
 while (fread(letters, 1, size, f) == size) { // Считываем по 2 символа  
 letters[size] = '\0';  
 for(int i = 0; i < size+1; i++){  
 D[count][i] = letters[i];  
 }  
 count++;  
 }  
  
 fclose(f);  
 char \*begin;  
  
  
  
 node \*data[count];  
 int size\_data = 0;  
 for(int i = 0; i < count; i++){  
 int flag = 0;  
 for(int j = 0; j < size\_data; j++){  
 int flag\_isEq = 1;  
 begin = D[i];  
 char \*b = data[j]->letter;  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag\_isEq = 0;  
 break;  
 }  
 begin++;  
 b++;  
 }  
 if(flag\_isEq){  
 flag = 1;  
 data[j]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[size\_data] = (node \*)malloc(sizeof(node));  
 char \*b = data[size\_data]->letter;  
 begin = D[i];  
 while (\*begin!='\0'){  
 \*b = \*begin;  
 b++;  
 begin++;  
 }  
 \*b = '\0';  
 data[size\_data]->count = 1;  
 data[size\_data]->code[0] = '\0';  
 size\_data++;  
 }  
 }  
 sort(size\_data, data);  
 while (size\_data != 1){  
 node \*temp = malloc(sizeof(node));  
 temp->count = data[size\_data-1]->count + data[size\_data-2]->count;  
 temp->letter[0] = '\0';  
 temp->code[0] = '\0';  
 temp->left = data[size\_data-1];  
 temp->right = data[size\_data-2];  
 data[size\_data-2] = temp;  
 size\_data--;  
 sort(size\_data, data);  
 }  
 node \*root = data[0];  
 char code[300];  
 haffman(root, code, code);  
 size\_data = 0;  
 huffmanTreeToArray(data, &size\_data, root);  
  
  
 char answer[count \* size \* 10];  
 int size\_ans = 0;  
 char \*ansBg = answer;  
 for(int i = 0; i < count; i++){  
 for(int j = 0; j < size\_data; j++){  
 int flag = 1;  
 char \*b = data[j]->letter;  
 begin = D[i];  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag = 0;  
 }  
 begin++;  
 b++;  
 }  
 if(flag){  
 begin = data[j]->code;  
 while (\*begin!='\0'){  
 \*ansBg = \*begin;  
 ansBg++;  
 size\_ans++;  
 begin++;  
 }  
 }  
 }  
 }  
  
 \*ansBg = '\0';  
 printf("%s", answer);  
  
 printf("\nK = %.1f", 4210.0/size\_ans);  
 free\_data(root);  
 clock\_t end\_time = clock();  
 double time\_taken = (double)(end\_time - start\_time) / **CLOCKS\_PER\_SEC**;  
  
 printf("\ntime: %f\n", time\_taken);  
 return 0;  
}

**Выводы программы:**Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

По результатам видно, что алгоритм Хаффмана справился с задачей сжатия лучше, таже скорость больше у алгоритма Хаффмана, но алгоритм Шеннона-Фано проще в реализации.

**Задание 6.**

Написать программу, восстанавливающую последовательности, полученные в п.3 в исходный вид согласно вариантам, приведенным в п.2.

Код, кодирование Шеннона-Фано:

#include <stdio.h>  
#include <stdlib.h>  
  
  
typedef struct node{  
 int count;  
 char letter[15];  
 char code[30];  
}node;  
void sort(int n, node \*array[n]){  
 for(int i = 0; i < n; i++){  
 int is\_sort = 1;  
 for(int j = n - 1; j > i; j--){  
 if(array[j]->count > array[j-1]->count){  
 node \*temp = array[j];  
 array[j] = array[j-1];  
 array[j-1] = temp;  
 is\_sort = 0;  
 }  
 }  
 if(is\_sort){  
 break;  
 }  
 }  
}  
void putChar(char \*arr, char c){  
 char \*begin = arr;  
 while (\*begin!='\0'){  
 begin++;  
 }  
 \*begin = c;  
 \*(++begin) = '\0';  
}  
void fano(int size, node \*array[size]){  
 int sum1 = 0, sum2 = 0;  
 node \*part1[size];  
 node \*part2[size];  
 int size1 = 0;  
 int size2 = 0;  
 for(int i = 0; i < size; i++){  
 if(sum1 < sum2){  
 part1[size1++] = array[i];  
 sum1+=array[i]->count;  
 putChar(array[i]->code, '1');  
 }  
 else{  
 part2[size2++] = array[i];  
 sum2+=array[i]->count;  
 putChar(array[i]->code, '0');  
 }  
 }  
 if(size1 > 1){  
 fano(size1, part1);  
 }  
 if(size2 > 1){  
 fano(size2, part2);  
 }  
}  
  
  
  
int main(){  
 FILE \*f = fopen("text.txt", "r");  
 int size;  
 scanf("%d", &size);  
 char letters[size+1];  
 char D[4120][size+1];  
 int count = 0;  
 while (fread(letters, 1, size, f) == size) { // Считываем по 2 символа  
 letters[size] = '\0';  
 for(int i = 0; i < size+1; i++){  
 D[count][i] = letters[i];  
 }  
 count++;  
 }  
 fclose(f);  
 char \*begin;  
 node \*data[count];  
 int size\_data = 0;  
 for(int i = 0; i < count; i++){  
 int flag = 0;  
 for(int j = 0; j < size\_data; j++){  
 int flag\_isEq = 1;  
 begin = D[i];  
 char \*b = data[j]->letter;  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag\_isEq = 0;  
 break;  
 }  
 begin++;  
 b++;  
 }  
 if(flag\_isEq){  
 flag = 1;  
 data[j]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[size\_data] = (node \*)malloc(sizeof(node));  
 char \*b = data[size\_data]->letter;  
 begin = D[i];  
 while (\*begin!='\0'){  
 \*b = \*begin;  
 b++;  
 begin++;  
 }  
 \*b = '\0';  
 data[size\_data]->count = 1;  
 data[size\_data]->code[0] = '\0';  
 size\_data++;  
 }  
 }  
 sort(size\_data, data);  
 fano(size\_data, data);  
  
 char answer[count \* size \* 10];  
 int size\_ans = 0;  
 char \*ansBg = answer;  
 for(int i = 0; i < count; i++){  
 for(int j = 0; j < size\_data; j++){  
 int flag = 1;  
 char \*b = data[j]->letter;  
 begin = D[i];  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag = 0;  
 }  
 begin++;  
 b++;  
 }  
 if(flag){  
 begin = data[j]->code;  
 while (\*begin!='\0'){  
 \*ansBg = \*begin;  
 ansBg++;  
 size\_ans++;  
 begin++;  
 }  
 }  
 }  
 }  
 \*ansBg = '\0';  
  
 char string[4121];  
 begin = string;  
 ansBg = answer;  
 int shift = 0;  
 while (\*ansBg!='\0'){  
 int flag = 0;  
 for(int i = 0; i < size\_data; i++){  
 int flag\_isEq = 1;  
 for(int j = 0; j < shift + 1; j++){  
 if(ansBg[j] != data[i]->code[j]){  
 flag\_isEq = 0;  
 break;  
 }  
 if(j == shift && data[i]->code[j+1] != '\0'){  
 flag\_isEq = 0;  
 }  
 }  
 if(flag\_isEq){  
 ansBg+=shift+1;  
 flag = 1;  
 char \*b = data[i]->letter;  
 while (\*b != '\0'){  
 \*begin = \*b;  
 b++;  
 begin++;  
 }  
 }  
 }  
 if(flag){  
 shift = 0;  
 }  
 else{  
 shift++;  
 }  
 }  
 \*begin = '\0';  
 printf("%s\n",string);  
 for(int i = 0; i < size\_data; i++){  
 free(data[i]);  
 }  
 return 0;  
}

**Выводы программы:  
Изображение выглядит как снимок экрана, текст, шаблон, прямоугольный

Контент, сгенерированный ИИ, может содержать ошибки.**

Изображение выглядит как текст, снимок экрана, шаблон

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как снимок экрана, текст, шаблон, шов

Контент, сгенерированный ИИ, может содержать ошибки.

**Код, кодирование Хаффмана:**

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct node{  
 int count;  
 char letter[15];  
 char code[30];  
 struct node\* left;  
 struct node\* right;  
}node;  
void sort(int n, node \*\*array){  
 for(int i = 0; i < n; i++){  
 int is\_sort = 1;  
 for(int j = n - 1; j > i; j--){  
 if(array[j]->count > array[j-1]->count){  
 node \*temp = array[j];  
 array[j] = array[j-1];  
 array[j-1] = temp;  
 is\_sort = 0;  
 }  
 }  
 if(is\_sort){  
 break;  
 }  
 }  
}  
void haffman(node \*let, char\* code, char \*endCode){  
 if(let->letter[0] == '\0'){  
 \*endCode = '0';  
 \*(endCode+1) = '\0';  
 haffman(let->left, code, endCode+1);  
 \*endCode = '1';  
 \*(endCode+1) = '\0';  
 haffman(let->right, code, endCode+1);  
 }  
 else{  
 char \*begin = let->code;  
 char \*begin\_code = code;  
 while (\*begin\_code!='\0'){  
 \*begin = \*begin\_code;  
 begin++;  
 begin\_code++;  
 }  
 \*begin = '\0';  
 }  
}  
void huffmanTreeToArray(node \*\*data, int\* count, node \*root){  
 if(root->letter[0] != '\0'){  
 data[\*count] = root;  
 (\*count)+=1;  
 }  
 else{  
 huffmanTreeToArray(data, count, root->left);  
 huffmanTreeToArray(data, count, root->right);  
 }  
}  
void free\_data(node \*root){  
 if(root->letter[0] == '\0'){  
 free\_data(root->left);  
 free\_data(root->right);  
 }  
 free(root);  
}  
  
int main(){  
 FILE \*f = fopen("text.txt", "r");  
 int size;  
 scanf("%d", &size);  
 char letters[size+1];  
 char D[4120][size+1];  
 int count = 0;  
 while (fread(letters, 1, size, f) == size) { // Считываем по 2 символа  
 letters[size] = '\0';  
 for(int i = 0; i < size+1; i++){  
 D[count][i] = letters[i];  
 }  
 count++;  
 }  
 fclose(f);  
 char \*begin;  
 node \*data[count];  
 int size\_data = 0;  
 for(int i = 0; i < count; i++){  
 int flag = 0;  
 for(int j = 0; j < size\_data; j++){  
 int flag\_isEq = 1;  
 begin = D[i];  
 char \*b = data[j]->letter;  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag\_isEq = 0;  
 break;  
 }  
 begin++;  
 b++;  
 }  
 if(flag\_isEq){  
 flag = 1;  
 data[j]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[size\_data] = (node \*)malloc(sizeof(node));  
 char \*b = data[size\_data]->letter;  
 begin = D[i];  
 while (\*begin!='\0'){  
 \*b = \*begin;  
 b++;  
 begin++;  
 }  
 \*b = '\0';  
 data[size\_data]->count = 1;  
 data[size\_data]->code[0] = '\0';  
 size\_data++;  
 }  
 }  
 sort(size\_data, data);  
 while (size\_data != 1){  
 node \*temp = malloc(sizeof(node));  
 temp->count = data[size\_data-1]->count + data[size\_data-2]->count;  
 temp->letter[0] = '\0';  
 temp->code[0] = '\0';  
 temp->left = data[size\_data-1];  
 temp->right = data[size\_data-2];  
 data[size\_data-2] = temp;  
 size\_data--;  
 sort(size\_data, data);  
 }  
 node \*root = data[0];  
 char code[300];  
 haffman(root, code, code);  
 size\_data = 0;  
 huffmanTreeToArray(data, &size\_data, root);  
 char answer[count \* size \* 10];  
 int size\_ans = 0;  
 char \*ansBg = answer;  
 for(int i = 0; i < count; i++){  
 for(int j = 0; j < size\_data; j++){  
 int flag = 1;  
 char \*b = data[j]->letter;  
 begin = D[i];  
 while (\*begin != '\0'){  
 if(\*begin != \*b){  
 flag = 0;  
 }  
 begin++;  
 b++;  
 }  
 if(flag){  
 begin = data[j]->code;  
 while (\*begin!='\0'){  
 \*ansBg = \*begin;  
 ansBg++;  
 size\_ans++;  
 begin++;  
 }  
 }  
 }  
 }  
  
 \*ansBg = '\0';  
 char string[4121];  
 begin = string;  
 ansBg = answer;  
 int shift = 0;  
 while (\*ansBg!='\0'){  
 int flag = 0;  
 for(int i = 0; i < size\_data; i++){  
 int flag\_isEq = 1;  
 for(int j = 0; j < shift + 1; j++){  
 if(ansBg[j] != data[i]->code[j]){  
 flag\_isEq = 0;  
 break;  
 }  
 if(j == shift && data[i]->code[j+1] != '\0'){  
 flag\_isEq = 0;  
 }  
 }  
 if(flag\_isEq){  
 ansBg+=shift+1;  
 flag = 1;  
 char \*b = data[i]->letter;  
 while (\*b != '\0'){  
 \*begin = \*b;  
 b++;  
 begin++;  
 }  
 }  
 }  
 if(flag){  
 shift = 0;  
 }  
 else{  
 shift++;  
 }  
 }  
 \*begin = '\0';  
 printf("%s\n",string);  
  
  
 free\_data(root);  
  
 return 0;  
}

**Выводы программы:**

**Изображение выглядит как снимок экрана, текст, шаблон, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, шаблон, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.** **Изображение выглядит как текст, снимок экрана, шаблон

Контент, сгенерированный ИИ, может содержать ошибки.**

**Задание 7.**

Восстановить исходный текст из полученных последовательностей, пользуясь сервисом <https://onlineutf8tools.com/convert-binary-to-utf8>.

**Полученное сообщение:**

Ветер свистел, визжал, кряхтел и гудел на разные лады. То жалобным тоненьким голоском, то грубым басовым раскатом распевал он свою боевую песенку. Фонари чуть заметно мигали сквозь огромные белые хлопья снега, обильно сыпавшиеся на тротуары, на улицу, на экипажи, лошадей и прохожих.

**Вывод:** в ходе проделанной работы я научился кодировать и декодировать двоичную последовательность.