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 №2**

по дисциплине: Теория информации

тема: «Исследование кодов Шеннона-Фано»

Выполнил: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил: Твердохлеб Виталий

Викторович

Белгород 2025 г.

**Цель работы:** изучить способ кодирования Шеннона-Фано.

**Задание 1.**

Построить код для сообщения, содержащего строку панграммы **«в чащах юга жил бы цитрус? Да, но фальшивый экземпляр!».** Для полученного кода рассчитать показатели коэффициента сжатия и дисперсии.

Кодирование 1 буквы char 8 бит, значит размер 8\*54 = 432. Рассчитаем количество появлений каждого символа:



Теперь с помощью Шеннона-Фано получим кодовое предстовления каждого символа:  
Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Теперь закодируем предложение:  
Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Посчитаем коэффициент сжатия:  
изначально насчитали 432 бит, количество бит после сжатия: 260, в таком случае коэффициент сжатия равен = 1,66.

Рассчитаем среднюю длину символа и дисперсию, получим: 

**Задание 2.**

Построить код для сообщения, содержащего строку **«Victoria nulla est, Quam quae confessos animo quoque subjugat hostes»**

Для полученного кода рассчитать показатели коэффициента сжатия и дисперсии.

Закодированное предложение:

Изображение выглядит как снимок экрана, текст, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Кодирование 1 буквы char 8 бит, значит размер 8\*68 = 544. Размер сжатого равен 283, тогда коэффициент сжатия равен = 1,92.

Рассчитаем среднюю длину символа и дисперсию, получим:



**Задание 3.**

Построить консольное приложение, реализующее процесс кодирования по методу Шеннона-Фано (с возможностью расчета коэффициента сжатия и дисперсии).

#include <stdio.h>  
#include <windows.h>  
  
typedef struct node{  
 int count;  
 wchar\_t letter;  
 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() {  
  
 SetConsoleOutputCP(866);  
 SetConsoleCP(866);  
 wchar\_t str[1000000];  
 fgetws(str, sizeof(str) / sizeof(str[0]), **stdin**);  
  
 wchar\_t \*begin = str;  
 int size = 0;  
 while (\*begin!='\n'){  
 size++;  
 begin++;  
 }  
 \*begin = '\0';  
 begin = str;  
 int count = 0;  
 node \*data[size];  
 while (\*begin!='\0'){  
 int flag = 0;  
 for(int i = 0; i < count; i++){  
 if(data[i]->letter == \*begin){  
 flag = 1;  
 data[i]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[count] = (node \*)malloc(sizeof(node));  
 data[count]->letter = \*begin;  
 data[count]->count = 1;  
 data[count]->code[0] = '\0';   
 count++;  
 }  
 begin++;  
 }  
 sort(count, data);  
 fano(count, data);  
  
 char answer[size \* 8];  
 begin = str;  
 char \*ansBg = answer;  
 while (\*begin!='\0'){  
 for(int i = 0; i < count; i++){  
 if(data[i]->letter == \*begin){  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 \*ansBg=\*bg;  
 bg++;  
 ansBg++;  
 }  
 break;  
 }  
 }  
 begin++;  
 }  
 \*ansBg = '\0';  
 printf("\n%s", answer);  
 int s = 0;  
 ansBg = answer;  
 while (\*ansBg!='\0'){  
 ansBg++;  
 s++;  
 }  
 printf("\nsize: %d\n", s);  
  
 double K = 8\*size/(double)s;  
 printf("K = %f\n", K);  
  
 double sr = 0;  
 for(int i = 0; i < count; i++){  
 int len = 0;  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 bg++;  
 len++;  
 }  
 sr+= ((double)data[i]->count/size)\*len;  
 }  
 double d = 0;  
 for(int i = 0; i < count; i++){  
 int len = 0;  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 bg++;  
 len++;  
 }  
 d+= ((double)data[i]->count/size) \* (len - sr) \* (len - sr);  
 }  
 printf("D = %f\n", d);  
 for(int i = 0; i < count; i++){  
 free(data[i]);  
 }  
  
 return 0;  
}

Примеры:

Изображение выглядит как текст, снимок экрана, Шрифт

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

Контент, сгенерированный ИИ, может содержать ошибки.

**Задание 4.**

Программа для кодирования методом Хаффмана.

#include <stdio.h>  
#include <windows.h>  
  
typedef struct node{  
 int count;  
 wchar\_t letter;  
 char code[300];  
 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'){  
 \*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->left==**NULL**){  
 data[\*count] = root;  
 (\*count)+=1;  
 }  
 else{  
 huffmanTreeToArray(data, count, root->left);  
 huffmanTreeToArray(data, count, root->right);  
 }  
}  
void free\_data(node \*root){  
 if(root->left!=**NULL**){  
 free\_data(root->left);  
 free\_data(root->right);  
 }  
 free(root);  
}  
  
int main() {  
  
 SetConsoleOutputCP(866);  
 SetConsoleCP(866);  
 wchar\_t str[1000000];  
 fgetws(str, sizeof(str) / sizeof(str[0]), **stdin**);  
  
 wchar\_t \*begin = str;  
 int size = 0;  
 while (\*begin!='\n'){  
 size++;  
 begin++;  
 }  
 \*begin = '\0';  
 begin = str;  
 int count = 0;  
 node \*data[size];  
 while (\*begin!='\0'){  
 int flag = 0;  
 for(int i = 0; i < count; i++){  
 if(data[i]->letter == \*begin){  
 flag = 1;  
 data[i]->count++;  
 break;  
 }  
 }  
 if(flag == 0){  
 data[count] = (node \*)malloc(sizeof(node));  
 data[count]->letter = \*begin;  
 data[count]->count = 1;  
 data[count]->code[0] = '\0';  
 data[count]->left = **NULL**;  
 data[count]->right = **NULL**;  
 count++;  
 }  
 begin++;  
 }  
 sort(count, data);  
 while (count != 1){  
 node \*temp = malloc(sizeof(node));  
 temp->count = data[count-1]->count + data[count-2]->count;  
 temp->letter = '\0';  
 temp->code[0] = '\0';  
 temp->left = data[count-1];  
 temp->right = data[count-2];  
 data[count-2] = temp;  
 count--;  
 sort(count, data);  
 }  
  
 node \*root = data[0];  
 char code[300];  
 haffman(root, code, code);  
  
 count = 0;  
 huffmanTreeToArray(data, &count, root);  
 char answer[size \* 100];  
 begin = str;  
 char \*ansBg = answer;  
 while (\*begin!='\0'){  
 for(int i = 0; i < count; i++){  
 if(data[i]->letter == \*begin){  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 \*ansBg=\*bg;  
 bg++;  
 ansBg++;  
 }  
 break;  
 }  
 }  
 begin++;  
 }  
 \*ansBg = '\0';  
 printf("\n%s", answer);  
 int s = 0;  
 ansBg = answer;  
 while (\*ansBg!='\0'){  
 ansBg++;  
 s++;  
 }  
 printf("\nsize: %d\n", s);  
 double K = 8\*size/(double)s;  
 printf("K = %f\n", K);  
 double sr = 0;  
 for(int i = 0; i < count; i++){  
 int len = 0;  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 bg++;  
 len++;  
 }  
 sr+= ((double)data[i]->count/size)\*len;  
 }  
 double d = 0;  
 for(int i = 0; i < count; i++){  
 int len = 0;  
 char \*bg = data[i]->code;  
 while (\*bg!='\0'){  
 bg++;  
 len++;  
 }  
 d+= ((double)data[i]->count/size) \* (len - sr) \* (len - sr);  
 }  
 printf("D = %f\n", d);  
 free\_data(root);  
 return 0;  
}

Примеры:  
Изображение выглядит как текст, снимок экрана, Шрифт

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

Контент, сгенерированный ИИ, может содержать ошибки.

Размер закодированного сообщения 1 методом Хаффмана на 2 бита меньше, коэффициент сжатия соответственно больше, однако дисперсия у метода Шеннона-Фано меньше, это говорит о том, что символы, закодированные методом Шеннона-Фано более приближённые по размерам, но это не помогло получить более сжатый размер.

Размер закодированного сообщения 2 методом Хаффмана также меньше, на 5 бит, коэффициент сжатия также больше, но вновь дисперсия меньше у метода Шеннона-Фано, значит коды символов более сбалансированные.

**Вывод:** в ходе проделанной работы я реализовал алгоритм кодирования методом Шеннона-Фано и Хаффмана, после сравнения результатов заметил, что коэффициент сжатия у метода Хаффмана больше, значит сжатие прошло лучше, получили более короткое сообщение, но у метода Шеннона-Фано меньше дисперсия, значит таким методом получаются более сбалансированные коды символов, нет больших скачков.