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-211

Шамраев Александр Анатольевич

Проверил:

Твердохлеб Виталий Викторович

Белгород 2023 г.

содержание отчета

[Задание 3](#_Toc129610190)

[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. 4](#_Toc129610191)

[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. 12](#_Toc129610192)

[3 Получить кодовые представления сообщений из пунктов 1 и 2 задания по методу Хаффмана. Сравнить полученные результаты с методом Шеннона-Фано по показателям сжатия и дисперсии. Сделать соответствующие выводы. 14](#_Toc129610193)

[Вывод: 17](#_Toc129610194)

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.
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.
3. Получить кодовые представления сообщений из пунктов 1 и 2 задания по методу Хаффмана. Сравнить полученные результаты с методом Шеннона-Фано по показателям сжатия и дисперсии. Сделать соответствующие выводы.

# 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.

Для упрощения работы была написана программа:

#include <iostream>  
#include <vector>  
#include <array>  
#include <windows.h>  
#include <map>  
#include <cmath>  
#include <unordered\_map>  
  
using namespace std::literals;  
  
struct FanoCode {  
 wchar\_t symbol;  
 int count = 0;  
 std::vector<bool> code;  
};  
  
struct Tree {  
 wchar\_t symbol = '\_';  
 int count = -1;  
 Tree \*zero = nullptr;  
 Tree \*one = nullptr;  
};  
  
std::wostream &operator<<(  
 std::wostream &os,  
 const FanoCode &code  
) {  
 os << "<" << code.symbol << ">(" << code.count << ")" << " code: ";  
 for (const auto digit : code.code) {  
 os << digit;  
 }  
   
 return os;  
}  
  
template<class T>  
std::wostream &operator<<(  
 std::wostream &os,  
 const std::vector<T> &v  
) {  
 for (const auto &el : v) {  
 os << el;  
 }  
   
 return os;  
}  
  
template<class Iterator>  
Iterator SplitIntoSymmetricSums(  
 Iterator begin,  
 Iterator end  
) {  
 long long l\_sum = begin->count;  
 long long r\_sum = 0;  
   
 while (end - begin > 1) {  
 if (r\_sum > l\_sum) {  
 l\_sum += (++begin)->count;  
 } else {  
 r\_sum += (--end)->count;  
 }  
 }  
   
 return end;  
}  
  
template<class Iterator>  
void GetFanoCode(  
 const Iterator &begin,  
 const Iterator &end  
) {  
 if (end - begin < 2) { return; }  
   
 Iterator m = SplitIntoSymmetricSums(begin, end);  
   
 for (auto it = begin; it != end; ++it) {  
 it->code.push\_back(it < m);  
 }  
   
 GetFanoCode(begin, m);  
 GetFanoCode(m, end);  
}  
  
std::vector<FanoCode> GetFanoCode(const std::unordered\_map<wchar\_t, int> &counters) {  
 if (counters.empty()) {  
 return {};  
 }  
   
 if (counters.size() == 1) {  
 return **{**{counters.begin()->first, 1, std::vector**{**true**}**}**}**;  
 }  
   
 std::vector<FanoCode> codes;  
   
 for (auto &i : counters) {  
 codes.push\_back({i.first, i.second});  
 }  
   
 std::sort(codes.begin(),  
 codes.end(),  
 [](  
 FanoCode &a,  
 FanoCode &b  
 ) { return a.count > b.count; });  
 GetFanoCode(codes.begin(), codes.end());  
   
 return codes;  
}  
  
std::unordered\_map<wchar\_t, int> ParseString(const std::wstring &str) {  
 std::unordered\_map<wchar\_t, int> counters{};  
 for (const auto &symbol : str) {  
 counters[symbol]++;  
 }  
   
 return counters;  
}  
  
std::wstring ParseCode(const std::string &str, const size\_t n) {  
 std::wstring out;  
   
 size\_t l = str.length() / n;  
 for (::size\_t i = 0; i < l; i++) {  
 int id = 0;  
 for (size\_t j = 0; j < n; j++) {  
 id = id \* 2 + str[i \* n + j] - '0';  
 }  
   
 out.push\_back(id);  
 }  
   
 return out;  
}  
  
template<class Iterator>  
bool Merge(  
 Iterator &code\_begin,  
 Iterator &sum\_begin,  
 Iterator &sum\_end  
) {  
 auto cmp\_tree = [](  
 Tree &a,  
 Tree &b  
 ) { return static\_cast<unsigned>(a.count) <= static\_cast<unsigned>(b.count); };  
   
 Tree \*min[2];  
   
 for (auto &i : min) {  
 if (cmp\_tree(\*code\_begin, \*sum\_begin)) {  
 i = &\*code\_begin;  
 code\_begin++;  
 } else {  
 i = &\*sum\_begin;  
 sum\_begin++;  
 }  
 }  
   
 if (min[0]->count == -1 || min[1]->count == -1) {  
 return false;  
 } else {  
 \*sum\_end = Tree{'\_', min[0]->count + min[1]->count, min[0], min[1]};  
 sum\_end++;  
 return true;  
 }  
}  
  
template<class Iterator>  
Tree GetHuffmanCode(  
 Iterator code\_begin,  
 Iterator sum\_begin  
) {  
 auto sum\_end = sum\_begin;  
 while (Merge(code\_begin, sum\_begin, sum\_end)) {}  
 auto res = \*std::prev(sum\_end);  
 return res;  
}  
  
Tree GetHuffmanCode(const std::unordered\_map<wchar\_t, int> &counters) {  
 if (counters.empty()) {  
 return Tree{};  
 }  
   
 if (counters.size() == 1) {  
 return Tree{counters.begin()->first};  
 }  
   
 static std::vector<Tree> sums;  
 static std::vector<Tree> codes;  
   
 sums.clear();  
 codes.clear();  
   
 for (auto &i : counters) {  
 codes.push\_back({i.first, i.second});  
 }  
   
 codes.push\_back({});  
 codes.push\_back({});  
   
 sums = std::vector<Tree>(codes.size());  
   
 std::sort(codes.begin(),  
 codes.end(),  
 [](  
 Tree &a,  
 Tree &b  
 ) { return static\_cast<unsigned>(a.count) < static\_cast<unsigned>(b.count); });  
 auto res = GetHuffmanCode(codes.begin(), sums.begin());  
 return res;  
}  
  
void GetTable(  
 const Tree &huffman,  
 std::unordered\_map<wchar\_t, std::vector<bool>> &codes  
) {  
 static std::vector<bool> dcode;  
   
 if (huffman.zero == nullptr && huffman.one == nullptr) {  
 codes.insert({  
 huffman.symbol, dcode.empty()  
 ? std::vector**{**true**}** : dcode  
 });  
 } else {  
 if (huffman.zero != nullptr) {  
 dcode.push\_back(false);  
 GetTable(\*huffman.zero, codes);  
 dcode.pop\_back();  
 }  
   
 if (huffman.one != nullptr) {  
 dcode.push\_back(true);  
 GetTable(\*huffman.one, codes);  
 dcode.pop\_back();  
 }  
 }  
}  
  
std::unordered\_map<wchar\_t, std::vector<bool>> GetTable(const std::vector<FanoCode> &fano) {  
 std::unordered\_map<wchar\_t, std::vector<bool>> codes;  
 for (auto &el : fano) {  
 codes.insert({el.symbol, el.code});  
 }  
 return codes;  
}  
  
std::unordered\_map<wchar\_t, std::vector<bool>> GetTable(const Tree &huffman) {  
 std::unordered\_map<wchar\_t, std::vector<bool>> codes;  
 GetTable(huffman, codes);  
 return codes;  
}  
  
std::wstring CodeToStr(const std::vector<bool> &code) {  
 std::wstring s;  
 for (auto x : code) {  
 s.push\_back(x + '0');  
 }  
   
 return s;  
}  
  
std::wstring CodeMessage(  
 const std::unordered\_map<wchar\_t, std::vector<bool>> &code,  
 const std::wstring &str  
) {  
 std::wstring out;  
   
 for (wchar\_t i : str) {  
 out += CodeToStr(code.find(i)->second);  
 }  
   
 return out;  
}  
  
std::unordered\_map<std::vector<bool>, wchar\_t> GetDecodeTable(const std::unordered\_map<wchar\_t, std::vector<bool>> &code) {  
 std::unordered\_map<std::vector<bool>, wchar\_t> out;  
   
 for (auto& el: code) {  
 out[el.second] = el.first;  
 }  
   
 return out;  
}  
  
std::wstring DecodeMessage(  
 const std::unordered\_map<std::vector<bool>, wchar\_t> &code,  
 const std::wstring &str  
) {  
 std::wstring out;  
 std::vector<bool> c;  
   
 int i = 0;  
 while (i < str.length()) {  
 c.clear();  
 while (!code.contains(c)) {  
 c.push\_back(str[i] - '0');  
 i++;  
 }  
   
 out.push\_back(code.find(c)->second);  
 }  
   
 return out;  
}  
  
std::wostream &operator<<(  
 std::wostream &os,  
 const std::unordered\_map<wchar\_t, std::vector<bool>> &v  
) {  
 for (const auto &el : v) {  
 os << "<" << el.first << "> = " << el.second << "\n";  
 }  
   
 return os;  
}  
  
  
int GetCodeWeight(const std::wstring &str) {  
 std::unordered\_map<wchar\_t, int> counters{};  
 for (const auto &symbol : str) {  
 counters[symbol]++;  
 }  
   
 int i = ceil(log(counters.size()) / log(2));  
   
 return i \* str.length();  
}  
  
float GetDispersion(const std::unordered\_map<wchar\_t, std::vector<bool>> &v, const std::wstring& msg) {  
 auto r = ParseString(msg);  
   
 int sum = 0;  
 for (auto &el : v) {  
 sum += el.second.size();  
 }  
   
 float avg = float(sum) / v.size();  
   
 float dispersion = 0;  
 for (auto &el : v) {  
 float pi = float(r[el.first]) / msg.length();  
 dispersion += pi \* (el.second.size() - avg) \* (el.second.size() - avg);  
 }  
   
 return dispersion;  
}

int main() {  
 setlocale(LC\_ALL, "");  
   
 std::vector<std::wstring> msgs **{**L"в чащах юга жил бы цитрус? да но фальшивый экземпляр!",  
 L"Victoria nulla est, Quam quae confessos animo quoque subjugat hostes"**}**;  
 for (auto& msg : msgs) {  
 auto table = GetTable(GetFanoCode(ParseString(msg)));  
 auto coded = CodeMessage(table, msg);  
 std::wcout << "[FANO]\n" << "Table:\n" << table << "Message:\n" << msg << "\nCoded:\n" << coded <<"\nInitial length: " << GetCodeWeight(msg) << " --- Coded length: " << coded.length() << " --- Coefficient: " << float(GetCodeWeight(msg)) / coded.length() << " --- Dispersion: " << GetDispersion(table, msg) << "\n";  
 std::wcout << "Decoded\n" << DecodeMessage(GetDecodeTable(table), coded) << "\n";  
 }  
   
 for (auto& msg : msgs) {  
 auto table = GetTable(GetHuffmanCode(ParseString(msg)));  
 auto coded = CodeMessage(table, msg);  
 std::wcout << "[HUFFMAN]\n" << "Table:\n" << table << "Message:\n" << msg << "\nCoded:\n" << coded <<"\nInitial length: " << GetCodeWeight(msg) << " --- Coded length: " << coded.length() << " --- Coefficient: " << float(GetCodeWeight(msg)) / coded.length() << " --- Dispersion: " << GetDispersion(table, msg) << "\n";  
 std::wcout << "Decoded\n" << DecodeMessage(GetDecodeTable(table), coded) << "\n";  
 }  
 return 0;  
}

Результат работы:

[FANO]

Table:

<п> = 000000

<м> = 000001

<е> = 000010

<б> = 000011

<ю> = 000100

<с> = 011001

<у> = 01101

<т> = 011100

<э> = 011000

<а> = 110

<р> = 10000

<ы> = 10001

<в> = 1001

<я> = 001101

<и> = 1010

<л> = 1011

<н> = 01111

<ц> = 011101

< > = 111

<д> = 00111

<о> = 01011

<ф> = 010101

<ь> = 010100

<ш> = 010011

<й> = 010010

<к> = 010001

<з> = 010000

<ч> = 001100

<!> = 001011

<щ> = 001010

<х> = 001001

<г> = 001000

<?> = 00011

<ж> = 000101

Message:

в чащах юга жил бы цитрус? да но фальшивый экземпляр!

Coded:

10011110011001100010101100010011110001000010001101110001011010101111100001110001111011101101001110010000011010110010001111100111110111011110101111101010111010110101000100111010100110001010010111011000010001010000000010000001000000101100110110000001011

Initial length: 318 --- Coded length: 251 --- Coefficient: 1.26693 --- Average code length: 4.73585 --- Dispersion: 1.55287

Decoded:

в чащах юга жил бы цитрус? да но фальшивый экземпляр!

Вычисленная дисперсия = 1.55287

Коэффициент сжатия = 1.26693 (длина кодового слова исходного сообщения вычислялась через мощность алфавита (равна 6))

# 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.

Используя тот же код получаем:

[FANO]

Table:

<f> = 0000000

<,> = 0000001

<h> = 000001

<r> = 000010

<b> = 000011

<j> = 000101

<g> = 000111

< > = 111

<a> = 1000

<Q> = 000100

<n> = 00111

<V> = 000110

<s> = 110

<u> = 101

<o> = 1001

<e> = 0111

<t> = 0110

<i> = 0101

<q> = 0100

<m> = 00110

<c> = 00101

<l> = 00100

Message:

Victoria nulla est, Quam quae confessos animo quoque subjugat hostes

Coded:

00011001010010101101001000010010110001110011110100100001001000111011111001100000001111000100101100000110111010010110000111111001011001001110000000011111011010011101111000001110101001101001111010010110010100101011111111010100001100010110100011110000110111000001100111001100111110

Initial length: 340 --- Coded length: 278 --- Coefficient: 1.22302 --- Average code length: 4.08823 --- Dispersion: 1.13927

Decoded:

Victoria nulla est, Quam quae confessos animo quoque subjugat hostes

Вычисленная дисперсия = 1.13927

Коэффициент сжатия = 1.22302 (длина кодового слова исходного сообщения вычислялась через мощность алфавита (равна 5))

# Получить кодовые представления сообщений из пунктов 1 и 2 задания по методу Хаффмана. Сравнить полученные результаты с методом Шеннона-Фано по показателям сжатия и дисперсии. Сделать соответствующие выводы.

[HUFFMAN]

Table:

<ю> = 110111

<б> = 110110

<е> = 110101

<м> = 110100

<в> = 10000

<я> = 00011

<р> = 01111

<л> = 0110

<ы> = 10001

<ч> = 01110

<!> = 01001

<щ> = 01000

<а> = 001

<э> = 110010

<и> = 0101

<х> = 00010

<?> = 00001

<ж> = 00000

<п> = 100100

<з> = 100101

<к> = 100110

<й> = 100111

<ш> = 101000

<ь> = 101001

<ф> = 101010

<о> = 101011

<н> = 101100

<с> = 101101

<у> = 101110

<т> = 101111

< > = 111

<ц> = 110000

<д> = 110001

<г> = 110011

Message:

в чащах юга жил бы цитрус? да но фальшивый экземпляр!

Coded:

100001110111000101000001000101111101111100110011110000001010110111110110100011111100000101101111011111011101011010000111

111000100111110110010101111110101000101101010011010000101100001000110011111111001010011010010111010111010010010001100001

10111101001

Initial length: 318 --- Coded length: 251 --- Coefficient: 1.26693 --- Average code length: 4.73585 --- Dispersion: 1.47739

Decoded

в чащах юга жил бы цитрус? да но фальшивый экземпляр!

[HUFFMAN]

Table:

<i> = 11111

<q> = 11110

<a> = 1110

<o> = 1101

<e> = 1100

< > = 101

<j> = 100111

<b> = 100110

<h> = 00010

<u> = 001

<c> = 00011

<V> = 100001

<s> = 010

<t> = 0110

<g> = 100000

<m> = 01110

<f> = 100101

<,> = 100100

<l> = 01111

<n> = 0000

<Q> = 100010

<r> = 100011

Message:

Victoria nulla est, Quam quae confessos animo quoque subjugat hostes

Coded:

100001111110001101101101100011111111110101000000101111011111110101110001001101001001011000100011110011101011111000111101

100101000111101000010010111000100101101010101111000001111101110110110111110001110111110001110010101000110011010011100110

00001110011010100010110101001101100010

Initial length: 340 --- Coded length: 278 --- Coefficient: 1.22302 --- Average code length: 4.08824 --- Dispersion: 0.992215

Decoded

Victoria nulla est, Quam quae confessos animo quoque subjugat hostes

Вычисленная дисперсия первого сообщения = 1.47739

Коэффициент сжатия первого сообщения = 1.26693

Вычисленная дисперсия второго сообщения = 0.992215

Коэффициент сжатия второго сообщения = 1.22302

Вывод:

Вывод: коды Хаффмана и Шеннона-Фано – префиксные неравномерные коды. Часто встречающийся символ кодируется меньшим числом бит, редко – большим. Код Хаффмана и Шеннона-Фано в большинстве случаев дают одинаковый коэффициент сжатия, однако дисперсия кода Хаффмана стабильно ниже, чем у кода Шеннона-Фано.