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 4
по дисциплине: Теория информации
тема: «Исследование кода Гилберта-Мура»

Выполнил: ст. группы ПВ-211
Шамраев Александр Анатольевич

Проверил:
Твердохлеб Виталий Викторович

Белгород 2023 г.

СОДЕРЖАНИЕ ОТЧЕТА

ЗАДАНИЕ

1) Изучить принцип построения кода, используя пример в закрепленном файле «Пример. ЛБ4.pdf».

2) Построить обработчик, выполняющий компрессию по алгоритму Гилберта-Мура.

3) Создать генераторы данных, работающих как источники Хартли и Бернулли (в двоичном алфавите).

4) Сгенерировать 2 цепочки данных длиной 800 символов каждую (соответственно, порожденные по Бернуллевскому принципу и принципу Хартли).

5) Выбрать произвольную текстовую строку на русском языке длиной 100 символов, преобразовать в двоичный формат Unicode/ASCII (на выбор), пользуясь любым соответствующим онлайн-сервисом (на выходе имеем последовательность из 800 символов).

6) Построить коды Гилберта-Мура для каждой из последовательностей, для чего предварительно сегментировать каждую цепочку по 8 символов.

7) Вычислить полученные коэффициенты сжатия и величину дисперсии для каждой последовательности.

1 Построить обработчик, выполняющий компрессию по алгоритму Гилберта-Мура.

```
std::vector<bool> GetFloatNBits(
    float number,
    int n
) {
  std::vector<bool> res(n);
  for (int i = 0; i < n; i++) {
    number *= 2;
    res[i] = int(number);
    number -= int(number);
  }

  return res;
}

std::vector<FanoCode> GetMooreHilbertCode(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;
  std::vector<float> d;
  std::vector<float> sigma;
  size_t total = 0;

  for (auto &i : counters) {
    codes.push_back({i.first, i.second});
    total += i.second;
  }

  for (int i = 0; i < codes.size(); i++) {
    double p = static_cast<float>(codes[i].count) / total;

    if (i == 0) {
      d.push_back(0);
    } else {
      double prev_p = static_cast<float>(codes[i - 1].count) / total;
      d.push_back(d[i - 1] + prev_p);
    }

    sigma.push_back(d[i] + p / 2);
    int length = ceil(-std::log2(p)) + 1;
    codes[i].code = GetFloatNBits(sigma[i], length);
  }
```

```
  return codes;
}
```

2 Построить обработчик, выполняющий компрессию по алгоритму Гилберта-Мура.

```
std::wstring HartliGenerator(
    int n,
    int n_bits
) {
  std::string t;
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<unsigned char> d(0, std::min((1 <<
n_bits) - 1, 127));
  for (int i = 0; i < n; i++) {
    t.push_back(d(gen));
  }

  return utf8_to_utf16(t);
}
```

3 Создать генераторы данных, работающих как источники Хартли и Бернулли (в двоичном алфавите).

```
std::string BernoulliGenerator(
    int n,
    float p
) {
  std::string t;
  std::random_device rd;
  std::mt19937 gen(rd());
  std::bernoulli_distribution d(p);
  for (int i = 0; i < n; i++) {
    t.push_back(d(gen) + '0');
  }

  return t;
}
```

4 Построить коды Гилберта-Мура для каждой из последовательностей, для чего предварительно сегментировать каждую цепочку по 8 символов.

```
Алгоритм Гилберта-Мура для источника Хартли
Код:
<‼> = 11111110
<+> = 11111100
<↑> = 1111100
<4> = 11110100
<♂> = 11110001
<M> = 11101111
<↕> = 11101100
<♣> = 11101010
<{> = 1110010
<W> = 11011111
<6> = 11011101
<Q> = 11011010
<L> = 11011000
<,> = 1101010
<l> = 1100111
<V> = 01010000
<5> = 01001110
<p> = 01001011
<0> = 01001000
<$> = 0100010
<_> = 00001110
<x> = 01000001
<=> = 0011110
<n> = 00110111
<3> = 00110100
</> = 00110001
> = 00101010
<H> = 00100111
<S> = 00100101
<:> = 00100010
<u> = 01011101
<f> = 00000001
<<> = 00000011
<w> = 100111
<☺> = 10100101
<@> = 00000110
<♥> = 00101111
<y> = 00101100
<Z> = 00011101
<F> = 00001000
<~> = 00001011
> = 01111110
<C> = 1000001
<h> = 00010000
<-> = 00011010
<K> = 00010011
<♫> = 00010101
<\> = 00011000
<7> = 00011111
<r> = 011010
<s> = 01010011
<8> = 01010101
<[> = 0101100
<X> = 0110001
```

<T> = 0111001
<)> = 01110111
<.> = 0111101
<|> = 10000110
<♠> = 10001000
<¶> = 10001011
<O> = 10001110
<R> = 10010000
<`> = 10010011
<z> = 10010101
<#> = 10011000
<(> = 1010101
<c> = 10101111
<'> = 1011001
<
> = 10110111
<E> = 10111001
<e> = 1011111
<> = 11000011
<B> = 1100011
<}> = 11001011
Исходное сообщение в символьном представлении:
uX♠rC.T|R`¶#)rxspV80[$z=3/r[$♥yHwn(X'
l,({↑}+‼rwS.:7Z-\e=♫BKTh_~F@<{↑f
Закодированное сообщение в двоичном представлении:
0101110101100011000100001101010000010111101011100110000110100100001001001110001011100
1100001110111011010010000010101001101001011010100000110001011111100101010101001000010
1100010001010010101001111000110100001100010110100101100010001000101111001011000010011
1100111001101111010101011000110110011011011110001110101111111000011110011110111001110
1010101111110100101110001110011111011000100111110110100100111011011101110111101111010111
1111001011101100001111010000011110101011110001101100111101111101010111110100001010101
1001111101010101010111100101111100110010111111110011111110011010100111001001010111101
0010001000011111000111010001101000011000101111100111100001010111000110001001101110010
0010000000011100000101100001000000001100000001111100101111100000000001

```
<x> = 01111100
<=> = 0111100
<f> = 00100101
<5> = 00011010
<▼> = 00100111
<m> = 00010000
<↕> = 00100010
<l> = 00010011
<1> = 0001011
<k> = 0001111
<>> = 0000110
<[> = 0111001
<a> = 00001000
<-> = 00000110
<g> = 00000011
<*> = 00000001
<U> = 00101010
<<> = 00101111
<w> = 00101100
<◄> = 00110001
<K> = 00111001
<.> = 00110111
<i> = 00110100
<⌂> = 0011110
<]> = 010100
<@> = 01000001
<{> = 010001
<_> = 0101100
<|> = 0110000
<∟> = 0110011
<W> = 0110110
Исходное сообщение в символьном представлении:
;7vu|o[W_\u]WoW{=G}OxsN?N♫n▬|{96znn~6s:=]←u⌂K3@?.In;o◄{i<o⌂∟U▼]←f{o}~‼O↕}kNs5v1ow[1|
m]C>l>a-∟kg7*~;_
Закодированное сообщение в двоичном представлении:
100101010100001001110100000101100001010100111001011011001011001000100010000101010001
101101010100110110010001011110010110100101110010111110111110011000111101001100011111 0
100111011010111000111010100110000010001110010111110111100010111110001110001111011111 0
111110001101111110011110001010011111101000010011110001110011101110101000001100011100
110111110110001110001001010101010001100010100010011010000101111101010001111001100 1100
101010001001110101001111110001001010100011010101011100111101111001110101111100100 0101
011100000111111010011100011000110101001110000101110101000101100011100100010110110 0000
001000001010011110001000011000010011000011000001000000001100110011000111100000011 1010
000000000011111011100101001011 00
```

```
Алгоритм Гилберта-Мура для сообщения
Код:
<о> = 11110
<ы> = 1110011
<ь> = 1110000
<л> = 11011101
<е> = 110101
<т> = 11000
<г> = 1011010
<р> = 10100
<,> = 10011010
< > = 10001
<д> = 01111001
<н> = 011100
<а> = 01100
<с> = 010100
```

<м> = 00000001
<щ> = 00000011
<й> = 00000110
<ц> = 00001000
<и> = 00010
<ю> = 0010001
<к> = 0010101
<у> = 0011000
<п> = 0011011
<в> = 010000
<я> = 00111100
<К> = 01001011
<з> = 01001000
Исходное сообщение в символьном представлении:
Картельные сговоры не допускают ситуации, при которой активно развивающиеся страны третьего мира при
ЗАкодированное сообщение в двоичном представлении:
0100101101100101001100011010111011101111000001110011100111101011000101010010110101111
0010000111101010011100111000101110011010110001011110011111000110110011000010100001010
1011000010001110001000101010000010110000011000011000000100000010000101001101010001001
1011101000001010001001010111110110001111010100111100000011010001011000010101110000001
0010000011100111101000110100011000100100001000000010010000011000010001000000110001011
0101010100001111001000101010011000101000110001110011100111000111000101001101011100011
1000011010110110101111010001000000100010101000110010001001101110100000010

5 Вычислить полученные коэффициенты сжатия и величину дисперсии для каждой последовательности.

```
Алгоритм Гилберта-Мура для источника Хартли
Длина исходного сообщения в битах: 800
Длина закодированого сообщения в битах: 749
Коэффициент сжатия: 1.06809
Средняя длина кодового слова: 7.49
Дисперсия: 0.4099
```

```
Алгоритм Гилберта-Мура для источника Бернулли
Длина исходного сообщения в битах: 800
Длина закодированого сообщения в битах: 712
Коэффициент сжатия: 1.1236
Средняя длина кодового слова: 7.12
Дисперсия: 0.4656
```

```
Алгоритм Гилберта-Мура для сообщения
Длина исходного сообщения в битах: 800
Длина закодированого сообщения в битах: 583
Коэффициент сжатия: 1.37221
Средняя длина кодового слова: 5.83
Дисперсия: 1.0811
```