МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Эффективное кодирование в отсутствии помех

Лабораторная работа №1

по дисциплине

«Прикладная теория информации»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ломакин Д. В.

(подпись)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сачков А.Д.

(подпись)

13-АС-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2016

**Введение**

***Кодирование*** - отображение состояний некоторой системы (источника сообщений) с помощью состояний сложного сигнала, который представляет собой последовательность из *n* элементарных сигналов.

***Алфавит кода*** – множество состояний элементарного сигнала. Алфавит кода имеет размер . При элементарный сигнал имеет состояния ( и ).

***Кодовое слово*** – последовательность элементов алфавита кода, которая описывает состояние сложного сигнала.

Код называется равномерным, если кодовые слова имеют одинаковую длину, и неравномерным в противном случае.

В случае равномерного кода эффективное кодирование невозможно. Если, код неравномерный и известна априорная информация, например, вероятность появления кодовых слов или символов, то возможно провести эффективное кодирование.

Если источник сообщений вырабатывает последовательность из символов, каждому символу будет соответствовать кодовое слово определенной длины , то длина соответствующей последовательности из кодовых слов:

– размер алфавита и количество состояний источника,

- количество вхождений -ого символа в последовательность.

Умножив и разделив на и учитывая, что при неограниченном увеличении относительная частота стремится к вероятности , можно представить эту формулу в виде:

- средняя длина кодового слова

Качество кодирования определяется соотношением , чем оно меньше, тем эффективнее кодирование, а значит, для получения оптимального кодирования нужно уменьшать .

Таким образом, задача оптимального кодирования заключается в определении однозначно декодируемых кодовых слов с такими длинами, при которых их средняя длина минимальна.

Однако существует предельная возможность эффективного кодирования, которая определяется энтропией источника. ***Энтропия*** - это среднее количество информации, приходящееся на одно сообщение источника.

**Алгоритмы эффективного кодирования**

**Метод Шеннона-Фано**

Алгоритм: Все символы алфавита источника сообщений располагают в порядке убывания вероятностей их появления. Затем символы алфавита делятся на две группы приблизительно равной суммарной вероятности их появления. Все символы первой группы получают «» в качестве первого элемента кодового символа, а все символы второй группы — «». Далее группы делятся на подгруппы, по тому же правилу примерно равных вероятностей, и в каждой подгруппе присваивается вторая позиция кодовых символов. Процесс повторяется до кодирования всех символов алфавита источника сообщений.

Пусть, имеется последовательность из и , причем вероятность появления и известны.

Объединим разряды в блоки по в каждом. Изменяя от до , произведем кодирование методом Шенонна-Фано.

Средняя длина блока рассчитывается по формуле:

Энтропия определяется по формуле:

Пусть вероятность появления : , тогда вероятность появления : q.

Вычислим энтропию источника:

Для количества цифр в блоке , число блоков равно :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Кодовое слово |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Рассчитаем кодовые слова и запишем их в таблицу:

Рассчитаем среднюю длину блока:

Вычислим отношение:

Для количества цифр в блоке , число блоков равно :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Кодовое слово |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Рассчитаем кодовые слова и запишем их в таблицу:

Рассчитаем среднюю длину блока:

Вычислим отношение:

Для количества цифр в блоке , число блоков равно :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Кодовое слово |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1

0

Рассчитаем среднюю длину блока:

Вычислим отношение:

Во всех случаях , следовательно, кодирование эффективно.

**Метод Хаффмана**

Алгоритм: Все символы алфавита источника сообщений располагают в порядке убывания вероятностей их появления. Два события с минимальными вероятностями отмечаем и и объединяем в одно, вероятность которого определяется суммой вероятностей этих двух событий. Новое событие располагаем так, чтобы не нарушить порядок убывания вероятностей. Повторяем объединение событий, пока в списке не останется одно событие. В результате, код каждого символа получим чтением полученных последовательностей и в обратном порядке.

**Программная реализация алгоритма Хаффмана на языке С++**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <map>

#include <algorithm>

using namespace std;

void print\_usage(const string &);

enum alg\_t { HUFFMAN, SHANNON };

void huffman\_alg(const string &, ostream &);

void shannon\_alg(const string &, ostream &);

struct letter

{

char c;

int next\_id;

bool binary\_code;

double probability;

letter():

c(0),

next\_id(-1),

binary\_code(false),

probability(0)

{ };

letter(char \_c):

c(\_c),

next\_id(-1),

binary\_code(false),

probability(0)

{ };

letter(char \_c, double \_probability):

c(\_c),

next\_id(-1),

binary\_code(false),

probability(\_probability)

{ };

letter(double \_probability):

c(0),

next\_id(-1),

binary\_code(false),

probability(\_probability)

{ };

letter(char \_c, int \_next\_id, bool \_binary\_code, double \_probability):

c(\_c),

next\_id(\_next\_id),

binary\_code(\_binary\_code),

probability(\_probability)

{ };

};

int main(int argc, char\* argv[])

{

vector<string> args(argv, argv + argc);

if (find(args.begin(), args.end(), "-h") != args.end()

||

find(args.begin(), args.end(), "--help") != args.end())

{

print\_usage(args[0]);

}

else

{

string filename\_for\_input;

string filename\_for\_output = "-";

alg\_t alg = HUFFMAN;

ifstream input\_file;

ofstream output\_file\_stream;

streambuf \* output\_file;

if (args.size() < 2)

{

print\_usage(args[0]);

}

else

{

filename\_for\_input = args.back();

input\_file.open(filename\_for\_input);

if (!input\_file.is\_open())

{

cout << "Cannot open file " << filename\_for\_input

<< " for reading input!" << endl;

return -1;

}

vector<string>::iterator it;

it = find(args.begin(), args.end(), "-m");

if (it == args.end())

{

it = find(args.begin(), args.end(), "--method");

}

if (it != args.end())

{

it = next(it);

if (it == args.end())

{

cout << "Cannot find value for " << \*(--it)

<< " argument!" << endl;

return -1;

}

else

{

if (\*it == "huffman")

{

alg = HUFFMAN;

}

else if (\*it == "shannon")

{

alg = SHANNON;

}

else

{

cout << "Cannot parse value for " << \*(--it)

<< " argument!" << endl;

return -1;

}

}

}

it = find(args.begin(), args.end(), "-o");

if (it == args.end())

{

it = find(args.begin(), args.end(), "--output");

}

if (it != args.end())

{

it = next(it);

if (it == args.end())

{

cout << "Cannot find value for " << \*(--it)

<< " argument!" << endl;

return -1;

}

else

{

filename\_for\_output = \*it;

if (filename\_for\_output == "-")

{

output\_file = cout.rdbuf();

}

output\_file\_stream.open(filename\_for\_output);

if (!output\_file\_stream.is\_open())

{

cout << "Cannot open file " << filename\_for\_output

<< " for print results!" << endl;

return -1;

}

else

{

output\_file = output\_file\_stream.rdbuf();

}

}

}

string input((std::istreambuf\_iterator<char>(input\_file)),

std::istreambuf\_iterator<char>());

ostream out(output\_file);

switch (alg)

{

case HUFFMAN:

huffman\_alg(input, out);

break;

case SHANNON:

shannon\_alg(input, out);

break;

}

}

}

return 0;

}

void huffman\_alg(const string &input, ostream &out)

{

map<char, int> letterCounts;

map<char, double> probability;

out << "Source text (" << input.size() << " symbols): "

<< endl << input << endl;

for (size\_t i = 0; i < input.size(); ++i)

{

letterCounts[input[i]]++;

}

vector<letter> v(letterCounts.size());

out << "Letters info: " << endl;

size\_t pos = 0;

for (map<char, int>::iterator it = letterCounts.begin(); it != letterCounts.end(); ++it, ++pos)

{

letter l(it->first, (double)it->second / (double)input.size());

v[pos] = l;

out << "Symbol '" << l.c << "' presents " << it->second

<< " times. Probability: " << l.probability << endl;

}

vector<pair<double, int>> w;

for (size\_t i = 0; i < v.size(); ++i)

{

w.push\_back(make\_pair(v[i].probability, i));

}

while (w.size() > 1)

{

sort(w.rbegin(), w.rend());

pair<double, int> a = \*(--w.end());

pair<double, int> b = \*(--(--w.end()));

int next\_id = v.size();

v[a.second].binary\_code = 0;

v[b.second].binary\_code = 1;

v[a.second].next\_id = v[b.second].next\_id = next\_id;

v.push\_back(letter(v[a.second].probability + v[b.second].probability));

w.pop\_back();

w.pop\_back();

w.push\_back(make\_pair(v[a.second].probability + v[b.second].probability,

next\_id));

}

size\_t new\_file\_size = 0;

for (size\_t i = 0; i < letterCounts.size(); ++i)

{

out << "Code for '" << v[i].c << "' symbol is: ";

letter temp = v[i];

string code = temp.binary\_code ? "1" : "0";

while (temp.next\_id >= 0)

{

temp = v[temp.next\_id];

code += temp.binary\_code ? '1' : '0';

}

reverse(code.begin(), code.end());

new\_file\_size += code.size() \* letterCounts[v[i].c];

out << code << endl;

}

out << endl;

out << "Previous file size: " << input.size() \* 8 << " bits" << endl;

out << "New file size: " << new\_file\_size << " bits" << endl;

}

void shannon\_alg(const string &input, ostream &out)

{

out << "Not implemented yet" << endl;

}

void print\_usage(const string &executable)

{

cout << "Usage: " << endl;

cout << "\t" << executable << "[options] input\_file" << endl;

cout << "Arguments: " << endl;

cout << "\t input\_file" << endl;

cout << "\t\tFile which will be encoded" << endl;

cout << "\t --method, -m" << endl;

cout << "\t\t Available methods: huffman, shannon. Default: huffman " << endl;

cout << "\t --output, -o" << endl;

cout << "\t\t File in which results will be printed. Default: stdout " << endl;

cout << "\t Example: " << endl;

cout << "\t\t " << executable << " -m shannon -o result.txt text.txt" << endl;

}

**Руководство по запуску программы**

Управление работой программы осуществляется через аргументы командной строки.

Единственный обязательный аргумент - имя входного файла. Указывается в самом конце списка аргументов.

Также можно указать имя файла, куда будет выведен результат с помощью параметра *­-o*. По умолчанию результаты распечатываются на экран