Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт
по практической работе №5
«Кодирование и декодирование кодом Хэмминга»

Выполнил:
студент группы ИП-014
Бессонов А.О.

Работу проверил:
старший преподаватель
Дементьева К.И.

Новосибирск 2024 г.

Постановка задачи

Цель работы: Изучение кода Хэмминга.

Язык программирования: C, C++, C#, Python

Результат: программа, тестовые примеры, отчет.

Задание:

1. Для выполнения работы необходим закодированный побуквенным кодом файл из практической работы 2. В таком файле содержатся только 0 и 1.

2. Реализовать кодирование и декодирование файла кодом Хэмминга (длина кода 7 или 15)

3. После кодирования кодом Хэмминга в закодированном файле случайным образом с вероятностью p заменить 0 на 1, 1 на 0 (сделать ошибки при передаче файла в симметричном канале). Декодировать измененный файл. Сравнить исходный и декодированный файлы, подсчитать количество ошибок и заполнить таблицу.

Ход работы

Для кодирования кодом Хэмминга взят художественный текст на английском языке «Красавица и Чудовище», закодированный кодом Хаффмана. (Размер файла 149248 бит). Длина кода равна 7.

Результаты работы программы:

```
Probability = 0.0001
Error bits found during decoding = 22
Errors found when comparing files = 0

Probability = 0.001
Error bits found during decoding = 244
Errors found when comparing files = 2

Probability = 0.01
Error bits found during decoding = 2606
Errors found when comparing files = 139

Probability = 0.1
Error bits found during decoding = 19109
Errors found when comparing files = 9797
```

Вероятность ошибки	p = 0.0001	p = 0.001	p = 0.01	p = 0.1
Количество ошибок	0	2	139	9797
Количество исправленных ошибочных битов	22	244	2606	19109

Выводы

В практической работе было реализовано кодирование и декодирование кодом Хэмминга. Длина кода выбрана 7 бит, то есть в коде содержится 4 информационных бита и 3 проверочных бита.

Для кодирования был взят художественный текст на английском языке «Красавица и Чудовище», закодированный кодом Хаффмана. Размер файла составляет 149248 бит.

В результате кодирования Хэммингом файл стал размером 261184 бит. После чего подвергся изменениям с определенной вероятностью p и был декодирован.

С вероятностью $p = 0.0001$ в результате декодирования было исправлено 22 ошибочных бита и в итоге было найдено 0 ошибок. То есть можно заметить, что код Хэмминга безупречно выполнил свою работу, обеспечив идеальную передачу сообщения.

С вероятностью $p = 0.001$ в результате декодирования было исправлено 244 ошибочных бита и в итоге было найдено 2 ошибки. Количество исправленных битов увеличилось в 10 раз, как и вероятность по сравнению с прошлым результатом. Ошибок в декодированном файле всего 2, что означает, что код Хэмминга смог хорошо справиться со своей задачей.

С вероятностью $p = 0.01$ в результате декодирования было исправлено 2606 ошибочных бита и в итоге было найдено 139 ошибки. Опять можно заметить, что количество исправленных битов увеличилось в 10 раз, как и вероятность по сравнению с прошлым результатом. Но вот количество ошибок в декодированном файле увеличилось больше, чем в 10 раз, а значит код Хэмминга стал гораздо хуже справляться со своей задачей.

С вероятностью $p = 0.1$ в результате декодирования было исправлено 19109 ошибочных бита и в итоге было найдено 9797 ошибки. Количество исправленных битов опять же увеличилось примерно в 10 раз. Но вот количество ошибок увеличилось уже примерно в 100 раз, это показывает то,

что код Хэмминга уже не справляется с обнаружением и исправлением ошибок.

Таким образом можно заметить, что вероятность ошибки при передаче сообщения имеет сильное влияние на эффективность кода Хэмминга. При маленьких вероятностях он может полностью исправить все ошибки, но с ростом вероятности уменьшается его способность нахождения и исправления ошибок.

Код программы

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <random>

using namespace std;

void encode_Hamming(string filename)
{
    ifstream input(filename);
    ofstream output("encoded_Hamming_" + filename);

    char c;
    vector <int> buffer;
    while (input.get(c)) {
        buffer.push_back(c - '0');

        if (buffer.size() == 4) {
            int encoded_buffer[7]{};
            encoded_buffer[2] = buffer[0];
            encoded_buffer[4] = buffer[1];
            encoded_buffer[5] = buffer[2];
            encoded_buffer[6] = buffer[3];

            encoded_buffer[0] = encoded_buffer[2] ^ encoded_buffer[4]
^ encoded_buffer[6];
            encoded_buffer[1] = encoded_buffer[2] ^ encoded_buffer[5]
^ encoded_buffer[6];
            encoded_buffer[3] = encoded_buffer[4] ^ encoded_buffer[5]
^ encoded_buffer[6];

            for (int i = 0; i < 7; i++)
                output.put(encoded_buffer[i] + '0');

            buffer.clear();
        }
    }

    input.close();
    output.close();
}

void change_bits_with_probability(string filename, double
probability)
{
    ifstream input(filename);
    ofstream output("probability_" + to_string(probability) + "_" +
filename);
```

```
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> distribution(0.0, 1.0);

    char c;
    while (input.get(c)) {
        if (distribution(gen) <= probability)
            c = c == '0' ? '1' : '0';
        output.put(c);
    }

    input.close();
    output.close();
}

void decode_Hamming(string filename)
{
    ifstream input(filename);
    ofstream output("decoded_Hamming_" + filename);

    long error_count = 0;

    char c;
    vector <int> buffer;
    while (input.get(c)) {
        buffer.push_back(c - '0');

        if (buffer.size() == 7) {
            int data[4]{};

            int p1 = buffer[0] ^ buffer[2] ^ buffer[4] ^ buffer[6];
            int p2 = buffer[1] ^ buffer[2] ^ buffer[5] ^ buffer[6];
            int p3 = buffer[3] ^ buffer[4] ^ buffer[5] ^ buffer[6];

            int error_bit = p1 + 2 * p2 + 4 * p3;
            if (error_bit != 0) {
                buffer[error_bit - 1] = buffer[error_bit - 1] == 0 ?
1 : 0;
                error_count++;
            }

            data[0] = buffer[2];
            data[1] = buffer[4];
            data[2] = buffer[5];
            data[3] = buffer[6];

            for (int i = 0; i < 4; i++)
                output.put(data[i] + '0');

            buffer.clear();
        }
    }
```

```
    cout << "Error bits found during decoding = " << error_count <<
"\n";

    input.close();
    output.close();
}

void compare_original_and_decoded_files(string original_filename,
string decoded_filename)
{
    ifstream original(original_filename);
    ifstream decoded(decoded_filename);

    long error_count = 0;

    char original_char, decoded_char;
    while (original.get(original_char) && decoded.get(decoded_char))
        if (original_char != decoded_char)
            error_count++;

    cout << "Errors found when comparing files = " << error_count <<
"\n";

    original.close();
    decoded.close();
}

int main()
{
    string filename = "encoded_Huffman_Beauty_and_the_Beast.txt";
    // orignal file = 149248 bits
    string encoded_filename = "encoded_Hamming_" + filename;

    encode_Hamming(filename);

    vector<double> probabilities = { 0.0001, 0.001, 0.01, 0.1 };

    for (double probability : probabilities) {
        cout << "Probability = " << probability << "\n";
        change_bits_with_probability(encoded_filename, probability);

        string probability_encoded_filename = "probability_" +
to_string(probability) + "_" + encoded_filename;
        decode_Hamming(probability_encoded_filename);

        string decoded_probability_encoded_filename =
"decoded_Hamming_" + probability_encoded_filename;
        compare_original_and_decoded_files(filename,
decoded_probability_encoded_filename);
        cout << "\n";
    }
```

```
    return 0;
}
```