

Sprawozdanie z LAB 08

Zadaniem zajęć było kodowania korekcyjnego hamminga.

Oto kod źródłowy (modulatory i demodulatory):

```
static class Hamming74_SECDED {
    static std::vector<std::vector<int>> const G_matrix;
    static std::vector<std::vector<int>> const H_matrix;

    static std::vector<int> mod2(std::vector<int> in)
    {
        std::vector<int> a = {in.at(0) % 2};
        return a;
    }

    static std::string _code(bool d1, bool d2, bool d3, bool d4)
    {
        std::string wyn;
        std::vector<std::vector<int>> const d_matrix = { {d1},{d2},{d3},{d4} };
        auto coded_matrix = matrix::multiply(G_matrix, d_matrix);
        std::transform(coded_matrix.begin(), coded_matrix.end(), coded_matrix.begin(), mod2);
        int parzystosc = 0;
        for (std::vector<std::vector<int>>::iterator it = coded_matrix.begin(); it != coded_matrix.end(); it++)
        {
            parzystosc += it->at(0);
        }
        wyn += std::to_string(parzystosc % 2);
        for (std::vector<std::vector<int>>::iterator it = coded_matrix.begin(); it != coded_matrix.end(); it++)
        {
            wyn += std::to_string(it->at(0));
        }
        return wyn;
    }

    static std::string _decode(bool d1, bool d2, bool d3, bool d4, bool d5, bool d6, bool d7, bool d8)
    {
        std::vector<std::vector<int>> const d_matrix = { {d8},{d7},{d6},{d5},{d4},{d3},{d2} };
        auto decoded_matrix = matrix::multiply(H_matrix, d_matrix);
        std::transform(decoded_matrix.begin(), decoded_matrix.end(), decoded_matrix.begin(), mod2);
        int parzystosc = (d1 + d2 + d3 + d4 + d5 + d6 + d7 + d8) % 2;
        int blad = (decoded_matrix.at(2).at(0)*4) + (decoded_matrix.at(1).at(0) * 2) + decoded_matrix.at(0).at(0);
```

```

        if (parzystosc == 0 && blad == 0)
            return std::to_string(d4) + std::to_string(d6) + std::to_string(d7) + std::to_string(d8);
        else if (parzystosc == 1 && blad > 0)
        {
            switch (blad)
            {
                case 1:
                    d8 = !d8;
                    break;
                case 2:
                    d7 = !d7;
                    break;
                case 3:
                    d6 = !d6;
                    break;
                case 4:
                    d5 = !d5;
                    break;
                case 5:
                    d4 = !d4;
                    break;
                case 6:
                    d3 = !d3;
                    break;
                case 7:
                    d2 = !d2;
                    break;
                default:
                    break;
            }
            return std::to_string(d4) + std::to_string(d6) + std::to_string(d7) + std::to_string(d8);
        }
        else
            return "";
    }
public:
    static std::string code(std::string in)
    {
        if (in.size() % 4 != 0) return NULL;
        std::string wynik;
        for (int i = 0; i < in.size(); i+=4)
        {
            auto temp = _code(std::stoi(in.substr(i, 1)), std::stoi(in.substr(i + 1, 1)), std::stoi(in.substr(i + 2, 1)),
std::stoi(in.substr(i + 3, 1)));
            wynik.append(temp);
        }
    }

```

```

    }
    return wynik;
}
static std::string decode(std::string in)
{
    if (in.size() % 8 != 0) return NULL;
    std::string wynik;
    int wyciete = 0;
    for (int i = 0; i < in.size(); i += 8)
    {
        auto temp = _decode(std::stoi(in.substr(i, 1)), std::stoi(in.substr(i + 1, 1)), std::stoi(in.substr(i + 2, 1)),
std::stoi(in.substr(i + 3, 1)), std::stoi(in.substr(i + 4, 1)), std::stoi(in.substr(i + 5, 1)), std::stoi(in.substr(i + 6, 1)),
std::stoi(in.substr(i + 7, 1)));
        if (temp != "")
            wynik.append(temp);
        else
            wyciete++;
    }
    if (wyciete > 0) std::cout << "uszkodzonych ramek bylo" << wyciete << std::endl;
    return wynik;
}
static std::string destroy(std::string in, int bit, int ramka = 0)
{
    if (in.size() % 8 != 0) return NULL;
    if (in[((ramka+1) * 8) - bit] == '1')
        in[((ramka + 1) * 8) - bit] = '0';
    else
        in[((ramka + 1) * 8) - bit] = '1';
    return in;
}
};

std::vector<std::vector<int>> const Hamming74_SECDDED::G_matrix = {
{1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };
std::vector<std::vector<int>> const Hamming74_SECDDED::H_matrix = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };

int main()
{
    //auto proba = matrix::initialize(3, 5);
    //std::vector<std::vector<int>> mac1 = { {1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };
    //std::vector<std::vector<int>> mac2 = { {4},{6},{5},{3} };
    //matrix::print_matrix(mac1);
    //std::cout<< std::endl;
    //matrix::print_matrix(mac2);
    //auto wynik = matrix::multiply(mac1, mac2);

```

```
//matrix::print_matrix(wynik);

std::string proba = "1001";
std::cout << "dane: " << proba << std::endl;
auto zakodowany = Hamming74_SECEDED::code(proba);
std::cout << "zakodowany: " << zakodowany << std::endl;
auto popsuty = Hamming74_SECEDED::destroy(zakodowany, 5);
std::cout << "popsuty: " << popsuty << std::endl;
auto zdekodowany = Hamming74_SECEDED::decode(popsuty);
std::cout << "zdekodowany: " << zdekodowany << std::endl;
```

```
}
```

Wynik demodulacji:

```
Microsoft Visual Studio Debug Console  
dane: 1001  
zakodowany: 10011001  
popsuty: 10001001  
zdekodowany: 1001  
C:\Users\GSzwa\source\repos\TD_2020_4
```

LUB

```
Microsoft Visual Studio Debug Console  
dane: 1001  
zakodowany: 10011001  
popsuty: 10001001  
popsuty ponownie: 10001101  
uszkodzonych ramek bylo: 1  
zdekodowany:  
C:\Users\GSzwa\source\repos\TD_2020_4
```

Podsumowanie

Dzięki tym laboratoriom nauczyłem się przystosowywania binarnej informacji do przesyłu za pomocą analogowego ośrodka, co pozwala na zrozumienie w jaki sposób jest wysyłany sygnał za pomocą analogowego ośrodka. Wiedza to jest wykorzystywana w transmisji danych stanowiących rdzeń komunikacji pomiędzy urządzeniami cyfrowymi.

Wykonał Szwarc Grzegorz