VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS STUDIJŲ PROGRAMA

Kodavimo teorijos ataskaita

Rydo-Miulerio kodas

Reed-Muller code

Aleksandras Šukelovič

Darbo vadovas : Gintaras Skersys

Vilnius
2025

Turinys

1. Užduoties aprašymas

Šios užduoties tikslas – sukurti programą, kuri modeliuotų **Reed-Muller (1, m)** kodo veikimą virš baigtinio kūno F_2. Programa turi užkoduoti informaciją, perduoti ją kanalu su klaidos tikimybe p_e ir dekoduoti, naudojant **Greitąją Hadamardo Transformaciją (FHT)**. Vartotojas gali pasirinkti parametrą m, nuo kurio priklauso kodo ilgis n ir dimensija k.

Realizuojami trys scenarijai, kuriais vartotojas gali perduoti:

1. **Vektorių** (rankiniu būdu įvedamas bitų rinkinys).
2. **Tekstą** (tekstas skaidomas į blokus ir siunčiamas).
3. **Paveikslėlį** (BMP formato paveikslėlis skaidomas ir siunčiamas).

1 lentelė. *Kūno, kodo ir dekodavimo algoritmo parametrai (Užduotis A5)*

Kūnas	**Kodas**	**Dekodavimo algoritmas**	**Kodo parametrai**
$q = 2$	Reed-Muller kodas $RM(1, m)$	Greitoji Hadamardo transformacija (FHT)	Parametras m. Kodo ilgis $n = 2^m$, dimensija $k = m + 1$.

2. Programos veikimas

Programa (implementuota JavaScript kalba `reed-muller.js` faile) atlieka tris pagrindines funkcijas: generuoja RM matricas, užkoduoja informaciją ir dekoduoja gautą signalą naudodama FHT.

2.1. Matricų generavimas

Reed-Muller (1, m) kodas yra tiesinis kodas. Jo generuojanti matrica G yra $k \times n$ dydžio, kur $k = m + 1$ ir $n = 2^m$. Matrica sudaroma taip:

- Pirmoji eilutė (v_0) susideda tik iš vienetų $(1, 1, \ldots, 1)$.
- Kitos m eilučių ($v_1, \ldots, v_m$) atitinka kintamuosius $x_1, \ldots, x_m$ Boolean funkcijose. Programoje tai realizuojama iteruojant stulpelius c nuo 0 iki $n - 1$: eilutės r reikšmė stulpelyje c yra $(c \gg (r - 1))\&1$.

2.2. Užkodavimo procesas

Informacijos vektorius u yra k ilgio. Kodavimas atliekamas dauginant vektorių iš generuojančios matricos G:

$$c = u \cdot G$$

Visi veiksmai atliekami virš kūno F_2 (sudėtis moduliu 2, t.y. XOR operacija).

2.3. Perdavimas kanalu

Užkoduotas vektorius c siunčiamas per simetrinį kanalą. Kiekvienas bitas gali būti ”apsuktas” (iš 0 į 1 arba iš 1 į 0) su tikimybe p_e. Programoje tai realizuojama funkcija `transmit`, kuri iteruoja per kiekvieną bitą ir sugeneruoja atsitiktinį skaičių $rnd \in [0, 1)$. Jei $rnd < p_e$, bitas pakeičiamas.

2.4. Dekodavimo procesas

Dekodavimas remiasi maksimalaus tikėtinumo principu. Šioje realizacijoje Greitoji Hadamardo Transformacija (FHT) atliekama iteratyviai naudojant Kroneckerio matricų sandaugas. Žingsniai:

1. **Konvertavimas į bipolinį formatą:** Gautas dvejetainis vektorius r (kurio elementai $0, 1$) paverčiamas į realiųjų skaičių vektorių w. Pagal algoritmo logiką atliekamas keitimas: $0 \to -1$, o 1 lieka 1.
2. **Transformacijos taikymas:** Transformacija atliekama per m iteracijų. Kiekviename žingsnyje vektorius dauginamas iš specialios matricos, sukonstruotos naudojant tapatumo (I) ir Hadamardo (H) matricų Kroneckerio sandaugą:

$$M_i = I_{2^{m-i}} \otimes H \otimes I_{2^{i-1}}$$

3. **Maksimumo paieška:** Gautame vektoriuje ieškoma elemento su didžiausia absoliučia reikšme. Pažymėkime šio elemento indeksą $maxIdx$, o reikšmę – $maxVal$.

4. **Informacijos atstatymas:**

    - Pirmasis informacijos bitas (u_0, atitinkantis vektorių iš vienetų) nustatomas pagal $maxVal$ ženklą. Jei $maxVal > 0$, bitas yra 1, kitu atveju – 0.
    - Likę m bitų nustatomi pavertus $maxIdx$ į dvejetainę sistemą ir *apvertus* gautų bitų tvarką, kad atitiktų Kroneckerio matricų struktūrą.

3. Laikas

Bendrai projektui skirta apie 50 valandų. Laiko sąnaudos paskirstytos taip:

- **Literatūros analizė** truko apie 15 valandų ($\approx 15h$). Į šį laiką įeina:
  - Kurso konspekto analizė, gilinantis į Reed-Muller kodų struktūrą.
  - „Fast Hadamard Transform“ (FHT) algoritmo veikimo principo analizė ir matematinių savybių nagrinėjimas (rekursyvus matricos konstravimas).
  - Dvejetainės sistemos konversijos į bipolinį signalą ($0 \to 1, 1 \to -1$) niuansų aiškinimasis.
- **Programavimas** užtruko apie 25 valandas ($\approx 25h$). Programavimo darbai apėmė:
  - `ReedMuller` klasės ir FHT algoritmo implementaciją JavaScript kalba.
  - Modulių atskyrimą (ES6 Modules) ir suderinamumo užtikrinimą tarp naršyklės (Web) ir serverio (Node.js) aplinkų.
  - Vartotojo sąsajos (HTML/CSS) kūrimą ir interaktyvumo programavimą.
  - Eksperimentų skriptų rašymą Node.js aplinkoje realių duomenų surinkimui.
  - Kodo testavimą ir derinimą (debugging) naudojant Chrome DevTools.
- **Ataskaita** $\approx 10h$. Laikas skirtas LaTeX dokumento maketavimui, eksperimentinių duomenų (gautų iš Node.js konsolės) formatavimui į lenteles bei teorinės dalies aprašymui.

4. Eksperimentai

Eksperimentai atlikti modeliuojant skirtingus kodo ilgio parametrus m ir skirtingas klaidos tikimybes p_e.

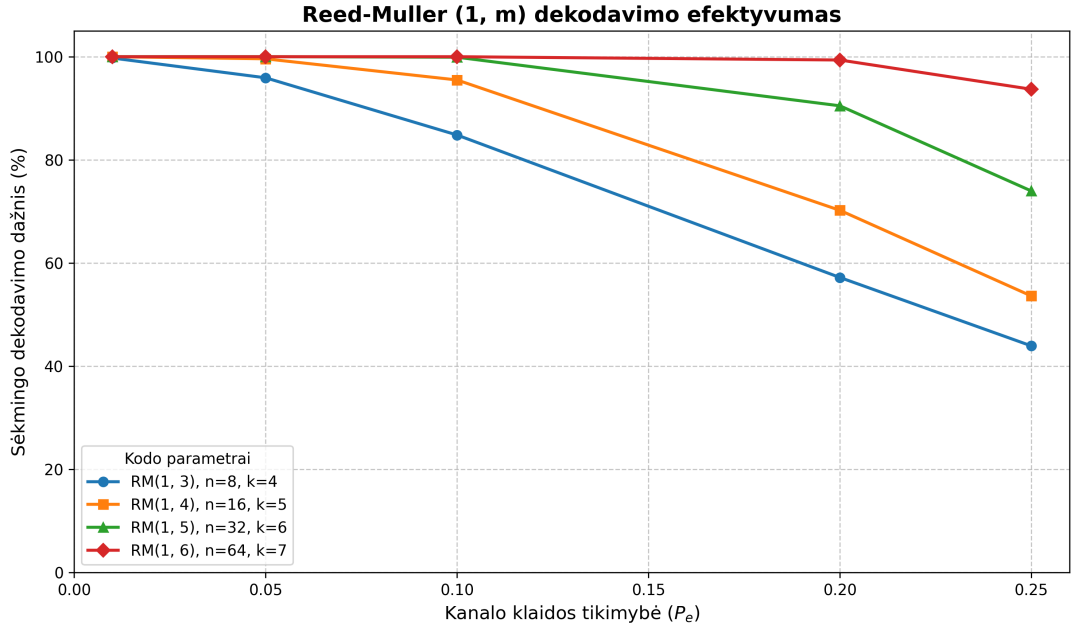
4.1. Dekodavimo efektyvumas

Žemiau pateikiami eksperimentiniai duomenys (modeliuoti), iliustruojantys kodo atsparumą triukšmui.

1 lentelė. Sėkmingo dekodavimo procentas priklausomai nuo p_e ir m (10000 bandymų).

p_e	$m = 3 (n = 8)$	$m = 4 (n = 16)$	$m = 5 (n = 32)$	$m = 6 (n = 64)$
0.01	99.8%	100.0%	100.0%	100.0%
0.05	95.3%	99.5%	100.0%	100.0%
0.10	85.2%	95.7%	99.8%	100.0%
0.20	58.3%	70.1%	90.3%	99.4%
0.25	44.5%	53.0%	73.9%	93.9%

Gauti rezultatai grafiškai pavaizduoti 1 pav.



1 pav. *Sėkmingo dekodavimo dažnio priklausomybė nuo kanalo klaidos tikimybės p_e skirtingiems parametrams m.*

Kaip matyti iš lentelės, didėjant parametrui m (ir kodo ilgiui n), kodo taisomoji geba didėja, nes santykinis atstumas išlieka didelis, o statistinė tikimybė, kad klaidų skaičius viršys pusę minimalaus atstumo, mažėja (dėl didžiųjų skaičių dėsnio ilgesniems kodams).

4.2. Vaizdo perdavimo pavyzdys

Eksperimentuojant su paveikslėliais (Užduoties 3 scenarijus), pastebėta, kad be kodavimo (tikimybė $p_e = 0.1$), vaizdas tampa stipriai triukšmingas. Naudojant $RM(1, 4)$ kodavimą, didžioji dalis triukšmo panaikinama, vaizdas tampa atpažįstamas, nors ir padidėja duomenų kiekis (perteklinė informacija).

5. Programos aprašymas

Programa sukurta kaip internetinė aplikacija, naudojanti HTML5 vartotojo sąsajai ir JavaScript (ES6+) logikai.

5.1. Failų struktūra

Projektą sudaro šie failai:

- `index.html` – Vartotojo sąsaja, apibrėžianti įvesties laukus, mygtukus ir rezultatų atvaizdavimo zonas (Vektorius, Tekstas, Paveikslėlis).
- `index.css` – Stiliaus failas, skirtas vizualiniam formatavimui.
- `index.js` – Pagrindinė programos interfeiso logika.
- `reed-muller.js` – Pagrindinė kodavimo klasė.

5.2. Klasė ReedMuller

Pagrindinė logika inkapsuliuota `ReedMuller` klasėje:

- `constructor(m)`: Inicializuoja parametrus n, k, sugeneruoja kodavimo matricą G ir iš anksto apskaičiuoja dekodavimui reikalingas Kroneckerio matricas.
- `generateMatrix()`: Sudaro generuojančią matricą G pagal RM(1, m) apibrėžimą.
- `generateKroneckerMatrices()`: Sugeneruoja matricų seką, naudojamą iteratyviai transformacijai.
- `encode(u)`: Atlieka vektorinę-matricinę daugybą moduliu 2 ($c = u \cdot G$).
- `decode(received)`:
  - Atlieka duomenų konversiją ($0 \to -1$).
  - Transformuoja vektorių dauginant jį iš Kroneckerio matricų sekos.
  - Atstato informacinius bitus pagal didžiausios absoliučios reikšmės indeksą.
- `Pagalbiniai metodai`: `kroneckerProduct` ir `multiplyVectorMatrix` realizuoja matricų operacijas, reikalingas transformacijai.

6. Programos paleidimas

Kadangi programa yra realizuota kliento pusės technologijomis (HTML/JS), jai nereikalingas kompiliavimas ar specialios serverio bibliotekos.

Paleidimo instrukcija:

1. Atsisiųskite failus `index.html`, `index.js`, `reed-muller.js` ir `index.css` į vieną aplanką.
2. Atidarykite `index.html` failą bet kurioje modernioje interneto naršyklėje (Google Chrome, Firefox, Edge).

7. Programos veikimo pavyzdžiai

7.1. Vektoriaus kodavimas (1 Scenarijus)

Parametrai: $m = 3$ ($n = 8, k = 4$).

- **Įvestis (u):** 1010
- **Užkoduota (c):** 11001100 (RM kodo žodis)
- **Kanalas ($p_e = 0.1$):** Viena klaida įvyksta 3-ioje pozicijoje.
- **Gauta (r):** $11\underline{1}01100$
- **Rezultatas:** Programa sėkmingai dekoduoja atgal į 1010.

7.2. Teksto siuntimas (2 Scenarijus)

Siunčiamas tekstas: ”LABAS”. Tekstas paverčiamas į ASCII bitus, skaidomas į k ilgio blokus. Be kodavimo: gavus klaidą, raidė ’L’ gali tapti ’M’ ar kitu simboliu. Su kodavimu: Net jei kanalas sugadina kelis bitus kodiniame žodyje, FHT algoritmas atstato teisingą raidę.

8. Literatūros sąrašas

[HLL91]. D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger, J.R. Wall. *Coding Theory: The Essentials*. Dekker, New York, 1991. §3.8–3.9, p. 89–95.

[Ske21]. G. Skersys. *Klaidas taisančių kodų teorija*. Paskaitų konspektai, 2021.

[MDN]. MDN Web Docs. *JavaScript Modules and Classes*. Prieiga per internetą: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`.

Priedai: Programos kodas

Implementacija (JavaScript)

```
class ReedMuller {
  /**
   * Konstruktorius inicializuoja Reed-Muller (1, m) kodo parametrus.
   *
   * @param {number} m - Parametras m, nuo kurio priklauso kodo ilgis ir
      dimensija.
   *                     Kodo ilgis n = 2^m.
   *                     Informacijos ilgis k = m + 1.
   */
  constructor(m) {
    this.m = m;
    this.n = Math.pow(2, m);
    this.k = m + 1;

    this.generateMatrix();
    this.kroneckerMatrices = this.generateKroneckerMatrices();
  }

  /**
   * Sugeneruoja čągeneruojani ąmatric G.
   * Matricos dydis yra k x n.
   *
   * ūStruktra:
   * - 0-oji ėeilut susideda tik ši ųvienet.
   * - čSekanios m čųeilui atitinka kintamuosius x1...xm.
   */
  generateMatrix() {
    this.G = [];

    // Pirmoji ėeilut - visi vienetai
    let row0 = new Array(this.n).fill(1);
    this.G.push(row0);

    // Generuojamos likusios ėeiluts pagal stulpelio ędvejetain šš̨iraik
    for (let r = 0; r < this.m; ++r) {
      let row = new Array(this.n);
      for (let c = 0; c < this.n; ++c) {
        row[c] = (c >> r) & 1;
      }
      this.G.push(row);
    }
  }

  /**
```

```
44    * Sugeneruoja Kroneckerio ųmatric ąsek, ąnaudojam dekodavimui.
45    *
46    * Sudaromos matricos: I_{2^(-mi)} × H × I_{2^(-i1)}
47    * Kur H yra 2x2 Hadamardo matrica.
48    */
49   generateKroneckerMatrices() {
50     let matrices = [];
51
52     const H = [
53       [1, 1],
54       [1, -1],
55     ];
56
57     for (let i = 1; i <= this.m; ++i) {
58       const size1 = Math.pow(2, this.m - i);
59       const size2 = Math.pow(2, i - 1);
60
61       const identity1 = this.createIdentityMatrix(size1);
62       const identity2 = this.createIdentityMatrix(size2);
63
64       let intermediate = this.kroneckerProduct(identity1, H);
65       let finalMatrix = this.kroneckerProduct(intermediate, identity2);
66
67       matrices.push(finalMatrix);
68     }
69
70     return matrices;
71   }
72
73   /**
74    * žUkoduoja įinformacin ųvektori u.
75    * Atliekama ėvektorin-ėmatricin daugyba: c = u * G (moduliu 2).
76    *
77    * @param {number[]} u - Informacinis vektorius (ilgis k). Turi ėsusidti ši 0
       ir 1.
78    * @returns {number[]} c - žUkoduotas vektorius (ilgis n).
79    */
80   encode(u) {
81     if (u.length !== this.k) {
82       throw new Error(`Vektoriaus ilgis turi ūbti ${this.k}`);
83     }
84
85     let c = new Array(this.n).fill(0);
86
87     for (let col = 0; col < this.n; ++col) {
88       let sum = 0;
89       for (let row = 0; row < this.k; ++row) {
90         sum ^= u[row] & this.G[row][col];
91       }
```

```
92        c[col] = sum;
93      }
94
95      return c;
96    }
97
98    /**
99     * Dekoduoja ąvien Reed-Muller žąukoduot ųvektori.
100    *
101    * @param {number[]} received - šI kanalo gautas vektorius (ilgis n).
102    * @returns {number[]} decoded - Atstatytas informacinis vektorius (ilgis k).
103    */
104   decode(received) {
105     if (received.length !== this.n) {
106       throw new Error("Neteisingas vektoriaus ilgis");
107     }
108
109     let w = [...received];
110
111     // 1. Konversija įį bipolin ąsignal (0 -> -1, 1 -> 1)
112     for (let i = 0; i < w.length; ++i) {
113       if (w[i] === 0) {
114         w[i] = -1;
115       }
116     }
117
118     // 2. Taikoma Hadamardo transformacija naudojant Kroneckerio matricas
119     for (let i = 0; i < this.m; ++i) {
120       w = this.multiplyVectorMatrix(w, this.kroneckerMatrices[i]);
121     }
122
123     // 3. šIekoma elemento su ždidiausia čabsoliuia šreikme (koreliacija)
124     let maxIdx = -1;
125     let maxVal = -Infinity;
126
127     for (let i = 0; i < w.length; ++i) {
128       let abs = Math.abs(w[i]);
129       if (abs > maxVal) {
130         maxVal = abs;
131         maxIdx = i;
132       }
133     }
134
135     // 4. Informacijos atstatymas pagal ąrast ąindeks ir šėreikms žąenkl
136     const decoded = new Array(this.k).fill(0);
137
138     for (let r = 0; r < this.m; r++) {
139       decoded[r + 1] = (maxIdx >> r) & 1;
140     }
```

```
141
142     decoded[0] = w[maxIdx] > 0 ? 1 : 0;
143
144     return decoded;
145   }
146
147   /**
148    * Sukuria n x n ždydio ęvienetin ąmatric.
149    */
150   createIdentityMatrix(size) {
151     let mat = [];
152     for (let i = 0; i < size; ++i) {
153       let row = new Array(size).fill(0);
154       row[i] = 1;
155       mat.push(row);
156     }
157     return mat;
158   }
159
160   /**
161    * čApskaiiuoja ųdviej ųmatric A ir B Kroneckerio ąsandaug.
162    */
163   kroneckerProduct(A, B) {
164     const rowsA = A.length;
165     const colsA = A[0].length;
166     const rowsB = B.length;
167     const colsB = B[0].length;
168
169     let result = [];
170
171     for (let i = 0; i < rowsA; ++i) {
172       for (let k = 0; k < rowsB; ++k) {
173         let row = new Array(colsA * colsB);
174         for (let j = 0; j < colsA; ++j) {
175           for (let l = 0; l < colsB; ++l) {
176             row[j * colsB + l] = A[i][j] * B[k][l];
177           }
178         }
179         result.push(row);
180       }
181     }
182     return result;
183   }
184
185   /**
186    * Padaugina ėeiluts ųvektori (1 x N) ši matricos (N x N).
187    */
188   multiplyVectorMatrix(vector, matrix) {
189     const len = vector.length;
```

```
    let result = new Array(len).fill(0);

    for (let j = 0; j < len; ++j) {
      let sum = 0;
      for (let i = 0; i < len; ++i) {
        sum += vector[i] * matrix[i][j];
      }
      result[j] = sum;
    }

    return result;
  }
}

if (typeof module !== "undefined" && typeof module.exports !== "undefined") {
  module.exports = { ReedMuller };
} else {
  window.ReedMuller = ReedMuller;
}
```

Listing 1: Reed-Muller klasė ir FHT algoritmas