VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Klaidas taisantys kodai

Ataskaita

Atliko: 4 kurso 1 grupės studentas
Tomas Kozakas (parašas)

Darbo vadovas:
Asist., Dr., Gintaras Skersys (parašas)

Vilnius
2025

Turinys

1. Užduoties aprašymas

Šios užduoties tikslas – sukurti programą, kuri modeliuotų klaidų taisymo kodo C veikimą virš baigtinio kūno F_q. Programa turi užkoduoti informaciją, perduoti ją kanalu su klaidos tikimybe p_e, ir dekoduoti, naudojant grandininį (step-by-step) dekodavimo algoritmą. Vartotojas gali pasirinkti kūno dydį q, kodo parametrus bei klaidos tikimybę.

Realizuojami trys scenarijai, kuriais vartotojas gali perduoti:

1. **Vektorių**
2. **Tekstą**
3. **Paveikslėlį**

Kūnas	**Kodas**	**Dekodavimo algoritmas**	**Kodo parametrai**
$q = 2$	tiesinis kodas $C[n,k]$	grandininis (step-by-step) dekodavimas (p. 79)	kodo ilgis n, dimensija k, generuojanti matrica G

1 lentelė. Kūno, kodo ir dekodavimo algoritmo parametrai

2. Programos veikimas

Programa (kodą galima surasti: https://github.com/tkozakas/coding-theory) atlieka tris pagrindines funkcijas: užkoduoja informaciją [3], perduoda ją per kanalą su klaidos tikimybe p_e [4] ir dekoduoja naudodama grandininį (step-by-step) dekodavimo algoritmą [6].

2.1. Matricu generavimas

Vartotojas turi įvesti arba automatiškai sugeneruoti generuojančios matricos G eilutėms ir stulpeliams nurodytus parametrus. Toliau programa automatiškai sugeneruoja tikrinimo (parity check) matrica H ir apskaičiuojami koset leader'iai su sindromais, kurie leidžia efektyviai identifikuoti klaidas perduotuose duomenų blokuose.

2.2. Koset lyderių paieška ir Sindromų skaičiavimas

Programa generuoja visus galimus klaidų šablonus ir jų atitinkamus sindromus, naudodama funkciją `findCosetLeaders`[5]. Kiekvienam galimam klaidos šablonui skaičiuojamas sindromas.

Programa naudoja koseto lyderių sąrašą (angl. *coset leaders*), kurį sudaro kiekvienam sindromui su mažiausiu galimu Hamingo svoriu priskirti klaidų vektoriai. Naudojant šiuos klaidų šablonus ir jų sindromus, programa pasirenka lyderį su mažiausia Hamingo svorio reikšme kiekvienam sindromui.

Funkcija `generateErrorPatterns` generuoja visus klaidų šablonus su tam tikru svoriu, o funkcija `updateCosetLeaders` atnaujina koseto lyderius, jei atranda mažesnio svorio klaidų vektorių.

Šis algoritmas veikia, kol visiems sindromams priskiriamas optimalus klaidos vektorius su mažiausia Hamingo svorio reikšme.

2.3. Bloko gavimas

Iš pradžių informacija (tekstą arba paveikslėlį) konvertuoja į dvejetainį bitų kodą, po to taikomas bitų sekos padalijimas į dalis arba blokus (angl. "blocks") pagal dydį k. Jei vaizdo duomenų seką suskaidžius negaunamas pilnas vektorius, jis yra užpildomas nuliniais bitais iki reikiamo dydžio k.

2.4. Užkodavimo procesas

Musu informacijos bloką (vektorius) $m = \begin{pmatrix} m_1 & m_2 & \cdots & m_k \end{pmatrix}$. programa užkoduoja, naudojant generuojanti matrica G, kurio parametrai yra kodo ilgis n, dimensija k. Generuojanti matrica gali būti arba įvesta vartotojo, arba sugeneruota programos automatiškai. Generuojanti matrica turi formą:

$$G = (I \mid A)$$

kur I yra vienetinė (standartinio pavidalo) $k \times k$ matrica, o A – atsitiktinių skaičių matrica, kurios dydis yra $k \times (n - k)$.

$$G = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{pmatrix}$$

$$c = (m \cdot G) = \begin{pmatrix} m_1 & m_2 & \cdots & m_k \end{pmatrix} \cdot \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{pmatrix} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix}$$

c - yra užkoduotas vektorius

2.5. Perdavimas kanalu

Užkoduota informacija siunčiama per kanalą [4], kuris modeliuoja klaidų atsiradimą su tikimybe p_e. Kanale kiekvienas siunčiamas kūno F_q simbolis yra iškraipomas su tikimybe p_e, nepriklausomai nuo kitų simbolių iškraipymo.

Kanalo veikimo principas yra toks: kiekvienam siunčiamam kūno F_q elementui atsitiktinai traukiamas skaičius a iš intervalo $[0, 1]$. Jei šis atsitiktinis skaičius a yra mažesnis už klaidos tikimybę p_e, siunčiamas elementas yra iškraipomas, priešingu atveju – paliekamas nepakitęs.

- Jei $q = 2$, kanalo veikimas yra paprastas: simbolis 0 pakeičiamas į 1, o 1 pakeičiamas į 0.
- Jei $q > 2$, iš likusių $q - 1$ kūno F_q elementų atsitiktinai parenkamas kitas elementas, kuriuo pakeičiamas siunčiamas simbolis. Tam naudojamas jau sugeneruotas atsitiktinis skaičius a. Intervalas $[0, p_e]$ padalijamas į $q - 1$ lygias dalis, ir nustatoma, kuriai daliai priklauso a, kad būtų galima pasirinkti naują elementą.

2.6. Dekodavimo procesas

Gautas vektorius po kanalo klaidų yra dekoduojamas naudojant grandininį dekodavimo algoritmą `decodeStepByStep`[6]. Dekodavimo metu naudojama pariteto tikrinimo matrica H ir koseto lyderiai, kad būtų pataisytos klaidos.

Dekodavimo algoritmas atliekamas taip:

1. Skaičiuojamas gauto vektoriaus r sindromas s, naudojant formulę:

$$s = H \cdot r^T$$

   kur H yra pariteto tikrinimo matrica, o r – gautas vektorius.

2. Pagal sindromą parenkamas koseto lyderis iš `cosetLeadersMap`, kuris nurodo minimalų klaidų skaičių, reikalingą taisymui.

3. Jei koseto lyderio svoris yra nulis ($w = 0$), tai reiškia, kad gautas vektorius yra teisingas kodinis žodis, ir dekodavimo procesas yra baigtas.

4. Jei koseto lyderio svoris nėra nulis, grandininis dekodavimas vyksta toliau, iteratyviai keičiant (apverčiant) po vieną bitą gautame vektoriuje.

5. Kiekvieną kartą apvertus bitą, perskaičiuojamas naujas sindromas ir tikrinama, ar naujojo koseto lyderio svoris sumažėjo. Jei naujas svoris yra mažesnis, pakeitimas išlaikomas, kitaip bitas grąžinamas į ankstesnę būseną.

6. Šis procesas kartojamas, kol bus pasiektas teisingas kodinis žodis arba nebus galima pataisyti daugiau klaidų.

3. Eksperimentai

3.1. Matricų skaičiavimas

Lentelėje 2, matome, kaip matricos generavimo sudėtingumas beveik nesikeičia didėjant n. Tai rodo, kad generavimo laikas išlieka stabilus, net kai didėja matricos dydis ar kontrolinių bitų skaičius. Tačiau, priešingai nei matricų generavimas, coset leader apskaičiavimo laikas auga itin sparčiai. Tai reiškia, kad nors didesnė n reikšmė nelemia žymaus generavimo laiko pailgėjimo, coset leader apskaičiavimas tampa vis sudėtingesnis. Jei reikia atlikti didesnį klaidų taisymą, n turėtų būti didesnis, tačiau tai reikš žymiai ilgesnį coset leader skaičiavimo laiką.

2 lentelė. Matricų generavimo rezultatai, kai $k = 16$, paleista 100 kartų

n	**Vid. generavimo laikas (ns)**	**Vid. pariteto tikrinimo laikas (ns)**	**Vid. coset leader laikas (ns)**	**Coset leaderių skaičius**
17	7455	3187	276947	2
18	3025	1520	440916	4
19	9847	2143	1069407	8
20	4056	2642	2805088	16
21	3427	2243	5105749	32
22	3698	2650	11851832	64
23	4303	4586	27307576	128
24	4568	2072	54633893	256
25	5204	2172	131037792	512
26	5655	2342	275484468	1024

3.2. Grandininis (step-by-step) dekodavimas sėkmės rodiklių rezultatai

Šiame skyriuje pateikiami grandininio algoritmo sėkmės rodikliai, kai duomenys buvo perduoti per kanalą, turintį skirtingą klaidos tikimybę (P_e). Rezultatai pateikiami trimis klaidos tikimybėmis: $P_e = 0.001$, $P_e = 0.01$ ir $P_e = 0.1$.

Sėkmės rodiklis (*success rate*) apskaičiuojamas pagal šią formulę:

$$\text{Sėkmės rodiklis} = \begin{cases} 100\%, & \text{jei totalIntroducedErrors} = 0 \\ \left(\frac{\text{totalFixedErrors}}{\text{totalIntroducedErrors}}\right) \times 100\%, & \text{jei totalIntroducedErrors} > 0 \end{cases}$$

Čia:

- **totalIntroducedErrors** – bendras įvestų klaidų skaičius.
- **totalFixedErrors** – ištaisytų klaidų skaičius.

Lentelėse pateikti skaičiai 9, 10, 11, 12, 13 ir t. t. atitinka mūsų naudojamų generuojančių matricų eilučių skaičių (n). Generuojanti matrica yra naudojama kodavimui ir klaidų taisymui, o n nurodo kiekvienos matricos dimensijas.

Tuo tarpu *Bloko dydis* nurodo mūsų bloko dydį, tai yra bitų seką, kurią mes norime perduoti per kanalą. Bloko dydis gali svyruoti nuo mažesnių (pvz., 8, 16 bitų) iki didesnių (pvz., 128, 256 bitų)

3 Rezultatų lentelė. Sėkmės rodiklis, kai $P_e = 0.001$

Bloko dydis	9	10	11	12	13	14	15	16	17	18
8	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
16	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
32	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
64	100.00	100.00	100.00	100.00	0.00	100.00	100.00	0.00	100.00	100.00
128	0.00	100.00	100.00	0.00	100.00	100.00	9.52	100.00	100.00	100.00
256	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	100.00	100.00

4 Rezultatų lentelė. Sėkmės rodiklis, kai $P_e = 0.01$

Bloko dydis	9	10	11	12	13	14	15	16	17	18
8	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
16	100.00	100.00	100.00	100.00	100.00	100.00	100.00	50.00	100.00	100.00
32	0.00	100.00	0.00	100.00	100.00	100.00	100.00	25.00	100.00	100.00
64	100.00	100.00	14.29	100.00	0.00	0.00	100.00	0.00	0.00	100.00
128	0.00	100.00	3.45	100.00	0.00	4.35	0.00	6.67	7.41	0.00
256	3.85	0.00	1.74	1.30	2.21	4.35	6.12	4.50	1.94	6.38

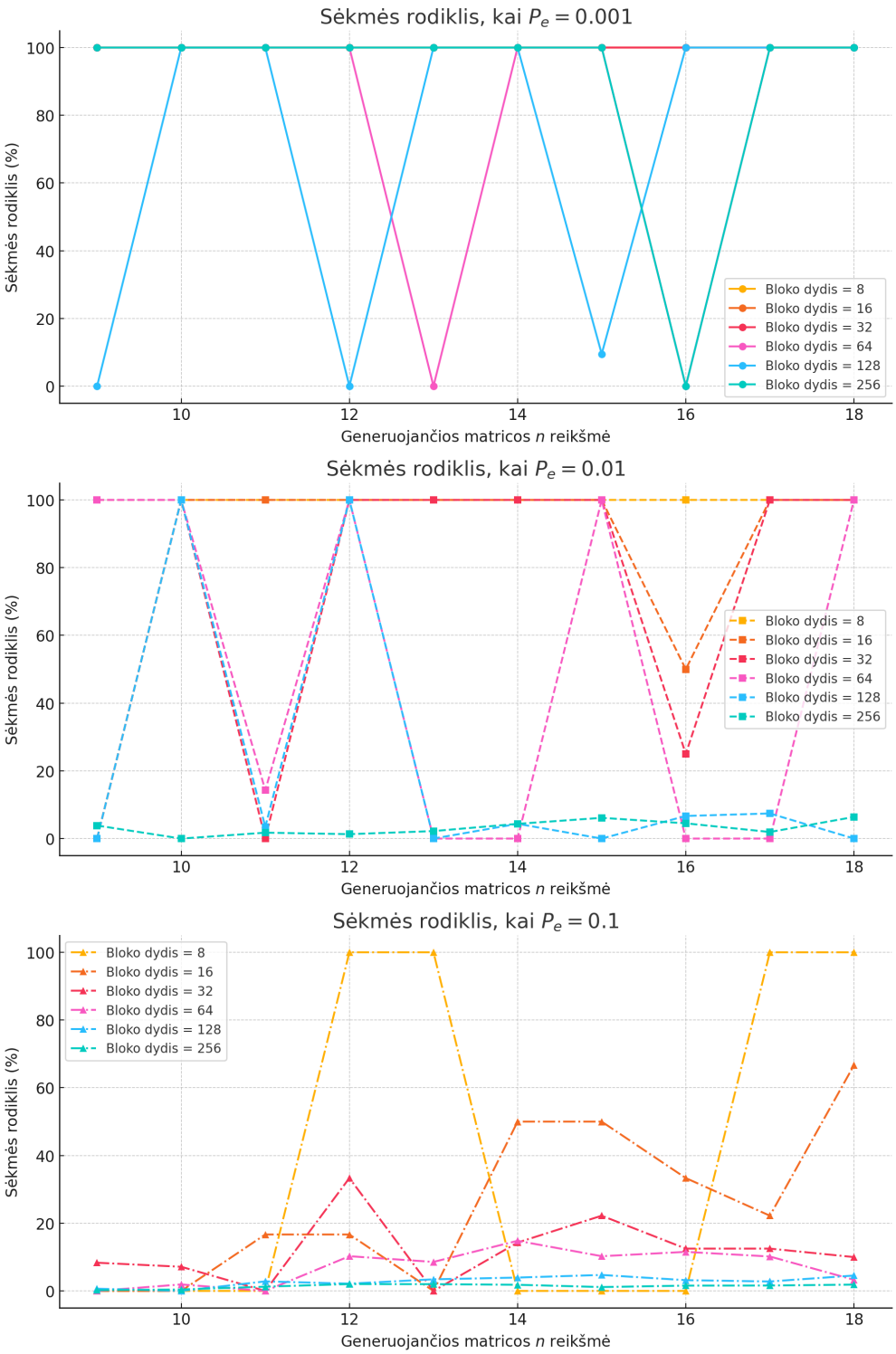
5 Rezultatų lentelė. Sėkmės rodiklis, kai $P_e = 0.1$

Bloko dydis	9	10	11	12	13	14	15	16	17	18
8	0.00	0.00	0.00	100.00	100.00	0.00	0.00	0.00	100.00	100.00
16	0.00	0.00	16.67	16.67	0.00	50.00	50.00	33.33	22.22	66.67
32	8.33	7.14	0.00	33.33	0.00	14.29	22.22	12.50	12.50	10.00
64	0.00	1.92	0.00	10.26	8.57	14.81	10.26	11.54	10.17	3.28
128	0.65	0.00	2.82	2.15	3.43	3.95	4.72	3.18	2.79	4.55
256	0.22	0.44	1.26	2.03	2.00	1.82	1.15	1.57	1.60	1.87

Kaip matome iš lentelės 3, kai $P_e = 0.001$, sėkmės rodiklis yra arti 100% daugumai įėjimo dydžių ir n verčių, rodančių, kad sistema puikiai taiso klaidas esant mažai klaidų tikimybei. Tai rodo, kad algoritmas efektyviai veikia esant labai nedidelei klaidų tikimybei. Šią tendenciją taip pat galime pastebėti grafike 1, kuriame sėkmės rodiklis išlieka stabilus ir aukštas, nepriklausomai nuo didėjančių generuojančios matricos n verčių.

Tuo tarpu lentelėje 4 matome, kad esant $P_e = 0.01$, didesni įėjimo dydžiai turi mažesnį sėkmės rodiklį, nors mažesnių įėjimo dydžių atvejais rodiklis išlieka aukštas. Grafike 1 galime pastebėti, kad esant šiai klaidų tikimybei, sėkmės rodikliai tampa ne tokie stabilūs — atsiranda tam tikrų „šuolių“ tarp skirtingų n verčių, o mažesni įėjimo dydžiai vis dar rodo geresnius rezultatus.

Galiausiai, lentelėje 5 (kai $P_e = 0.1$) matome, kad didėjant klaidų tikimybei, sėkmės rodikliai žymiai krinta, ypač didesnėms įėjimo dydžių vertėms. Šį didelį sėkmės rodiklio kritimą taip pat galime matyti grafike 1, kur didesnis klaidų kiekis sukelia didelius šuolius sėkmės rodikliuose tarp skirtingų n verčių. Nepaisant to, esant didesniems n, kai kurios vertės rodo tam tikrą stabilizaciją, tačiau bendrai sėkmės rodiklis yra žymiai mažesnis.



1 pav. Sėkmės rodiklis

4. Programos Aprašymas

Programa palaiko vektorių, teksto ir paveikslėlių apdorojimą bei leidžia vartotojui koreguoti klaidų tikimybę ir kitus parametrus. Programa turi du režimus.

Pirmasis režimas, naudojant grafinę sąsają 6, galima lengvai koreguoti parametrus ir atlikti generuojančiosios matricos ir pariteto tikrinimo matricos (*Parity Check Matrix*) vizualizavimą bei jų koregavimą

Antrasis režimas yra komandų eilutės pagrindu veikiantis režimas, kuris leidžia vartotojui atlikti visas tas pačias operacijas, išskyrus galimybę interaktyviai koreguoti iš kanalo išeinantį vektorių.

4.1. Programos failų struktūra

Programos kodas yra viduj src -> main -> java

```
java/
 model/
    CosetLeader.java
 processor/
    Data.java
    EncoderDecoder.java
    Processor.java
 ui/
    FxTableInterface.java
    FxUserInterface.java
    UserInterface.java
 Main.java
```

- **model/CosetLeader.java**: koseto lyderių duomenų laikymo modelis.
- **processor/Data.java**: duomenų apdorojimo ir saugojimo klasė.
- **processor/EncoderDecoder.java**: užkodavimo ir dekodavimo proceso realizacija.
- **processor/Processor.java**: bitų suskaidymo į dalis ir valdymo logika.
- **ui/FxTableInterface.java**: lentelės sąsaja su grafinės vartotojo sąsajos (GUI) vizualizavimu.
- **ui/FxUserInterface.java**: pagrindinė grafinė vartotojo sąsaja (GUI), naudojama interaktyviems veiksmams.
- **ui/UserInterface.java**: sąsaja komandų eilutės režimui.
- **Main.java**: pagrindinis programos paleidimo taškas.

5. Programos paleidimas

Programa buvo parašyta naudojant Java programavimo kalbą (Java 21) ir naudoja Lombok biblioteką bei JavaFX grafinei vartotojo sąsajai. Programą galima paleisti tiek su grafine vartotojo sąsaja, tiek be jos, priklausomai nuo poreikio. Programa buvo sukompiliuota ir jos '.jar' failas yra 'target' direktorijoje.

Norėdami paleisti programą su grafine vartotojo sąsaja (žr. 6), naudokite pateiktą komandą. Atkreipkite dėmesį, kad kopijuojant komandą iš PDF failo, ji gali būti nukopijuota neteisingai!):

```
java -jar target/coding-theory-1.0.jar ui
```

Kodas 1: Programos paleidimas su grafine vartotojo sąsaja

Jei grafinė vartotojo sąsaja nereikalinga, programą galima paleisti be papildomų parametrų:

```
java -jar target/coding-theory-1.0.jar
```

Kodas 2: Programos paleidimas be grafinės vartotojo sąsajos

Tai pat galima surasti informacija README.md faile.

5.1. Naudojami įrankiai ir bibliotekos

- **Java 21** – Java versija, naudojama programos logikai ir skaičiavimams.
- **JavaFX 21** – naudojama kuriant grafinę vartotojo sąsają (GUI).
- **Lombok** – naudojama automatizuoti kodo rašymą, pavyzdžiui, generuojant getterius, setterius ir kitus šabloninius metodus.

6. Programos įvedimo pavyzdžiai

Šiame skyriuje pateikiami vartotojo sąsajos įvedimo pavyzdžiai, iliustruojantys, kaip galima naudotis programa. Vartotojas gali įvesti vektorius, tekstą ar paveikslėlius, nurodyti generuojančią matricą ir keisti programos parametrus, tokius kaip klaidos tikimybė.

6.1. Pagrindinis meniu

Pagrindiniame meniu vartotojas gali pasirinkti, kokią veiksmą atlikti, pavyzdžiui, įvesti duomenis ar generuoti matricą.

Pavyzdys:

```
Probability of error: 0.05000
Number of symbols in the alphabet: 2
Generator matrix: Empty
Input vector length: 0
Input vector: Empty

Choose an option:
1. Input (vector, text, image) to process
2. Enter generating matrix
3. Generate generating matrix
4. Change probability of error
5. Change number of symbols in the alphabet
6. Debug mode (currently OFF)
Choice: 1
```

6.2. Generuojančios matricos įvedimas

Vartotojas gali rankiniu būdu įvesti generuojančią matricą, nurodydamas eilutes su dvejetainėmis reikšmėmis (tik 0 ir 1).

Pavyzdys:

```
Enter the number of columns (n): 5
Enter the number of rows (k): 3

Enter the matrix row by row (only 0s and 1s):
1 0 1 0 1
0 1 0 1 0
1 1 0 0 1
```

6.3. Vektoriaus apdorojimas

Vartotojas gali pasirinkti apdoroti dvejetainį vektorių, kurį reikia įvesti konsolėje. Programa užkoduoja vektorių, įveda klaidas ir vėliau iškoduoja. n = 6, k = 5

Pavyzdys:

```
Choose an option:
1. Process Vector
2. Process Text
3. Process Image
4. Back to main menu
Choice: 1

Enter the vector to encode:
1, 0, 1, 1, 0

Processing...

Decoded blocks: [[1, 0, 1, 1, 0]]
```

6.4. Teksto apdorojimas

Teksto apdorojimo atveju, vartotojas įveda tekstą, kurį programa koduos ir vėliau ištaisys klaidas, atsiradusias siunčiant per kanalą. n = 12 ir k = 8

Pavyzdys:

```
Choose an option:
1. Process Vector
2. Process Text
3. Process Image
4. Back to main menu
Choice: 2

Enter the text to encode:
Hello World

Processing...
Decoded text: Hello World
Decoded blocks: [[0, 1, 0, 0, 1, 0, 0, 0], [0, 1, 1, 0, 0, 1, 0, 1], [0, 1, 1, 0, 1
```

6.5. Paveikslėlio apdorojimas

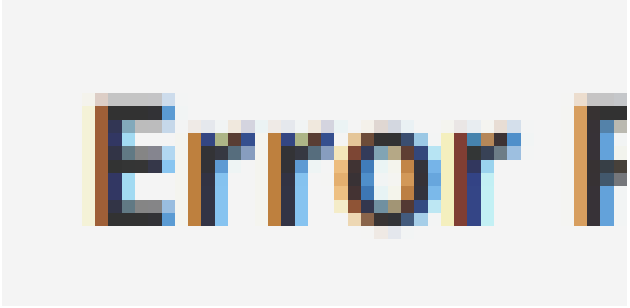
Vartotojas gali nurodyti paveikslėlio failo kelią. Programa užkoduoja paveikslėlį, įveda klaidas ir dekoduoja, išsaugodama rezultatą. n = 12, k = 8

Pavyzdys:

```
Choose an option:
1. Process Vector
2. Process Text
3. Process Image
4. Back to main menu
Choice: 3

Enter the path to the image file:
img/original_image.png

Processing...
Decoded image saved as img/decoded.png
```



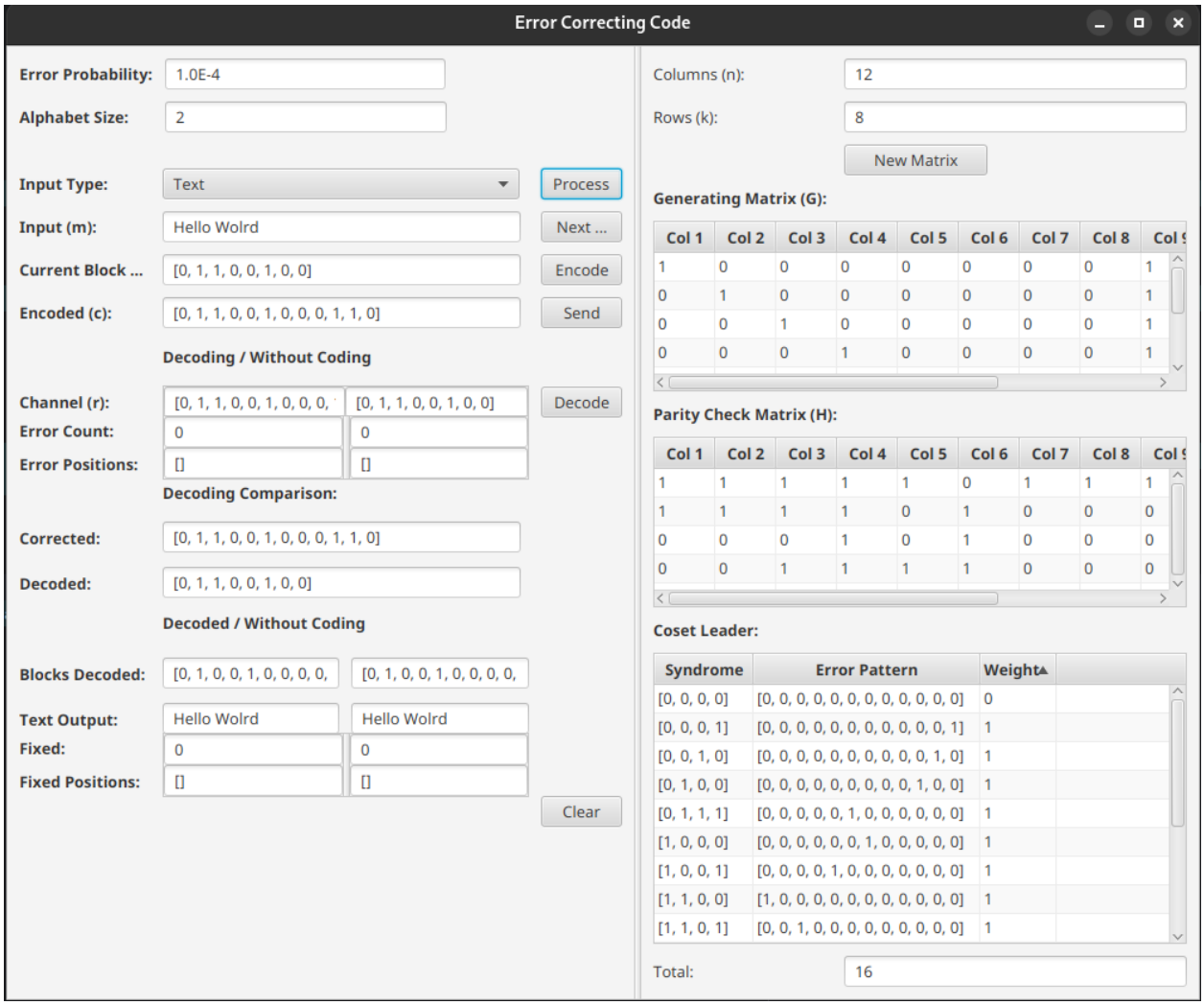
2 pav. Originalus paveikslėlis

3 pav. Dekoduotas paveikslėlis

4 pav. Paveikslėlis be kodo

5 pav. Palyginimas: originalus, dekoduotas ir paveikslėlis be kodo.

Priedai



6 pav. Grafinė sąsaja

```java
/**
     * Encodes a message using the generator matrix G.
     *
     * @param m message to encode
     * @param G generator matrix
     * @return encoded message
     */
    public int[] encode(int[] m, int[][] G) {
        int[] c = matrixVectorMultiply(m, G);
        if (debug) {
            StringBuilder message = Processor.getStringFromBits(m);
            System.out.println("\n========================================\n");
            System.out.printf("Encoding message (m): %s (%s)%n", Arrays.toString(m), message);
            System.out.println("\nGenerating matrix (G):");
            printMatrix(G);
            System.out.println("\nEncoded message (c): " + Arrays.toString(c));
        }
        return c;
    }

    /**
     * Multiplies a matrix by a vector.
     * @param m vector
     * @param G matrix
     * @return result of the multiplication
     */
    private int[] matrixVectorMultiply(int[] m, int[][] G) {
        int[] c = new int[G[0].length];
        for (int i = 0; i < G[0].length; i++) {
            for (int j = 0; j < m.length; j++) {
                c[i] += m[j] * G[j][i];
            }
            c[i] %= 2;
        }
        return c;
    }
```

Kodas 3: Pranešimo kodavimas

```java
/**
     * Introduces errors in a codeword.
     * @param c codeword
     * @param pe probability of error
     * @param q number of symbols in the alphabet
     * @return codeword with errors
     */
    public int[] introduceErrors(int[] c, double pe, int q) {
        int[] r = Arrays.copyOf(c, c.length);

        if (debug) {
            System.out.println("\n=== Introducing Errors ===");
            System.out.println("Original codeword (c): " + Arrays.toString(c));
        }

        for (int i = 0; i < r.length; i++) {
            double a = random.nextDouble();
            if (a < pe) {
                if (q == 2) {
                    r[i] ^= 1; // Flip the bit
                } else if (q > 2) {
                    // Change the symbol to a random one
                    int currentSymbol = r[i];
                    int newSymbol;
                    do {
                        newSymbol = random.nextInt(q);
                    } while (newSymbol == currentSymbol);
                    r[i] = newSymbol;
                } else {
                    throw new IllegalArgumentException("Invalid value for q: " + q);
                }

                if (debug) {
                    System.out.printf("Error introduced at position %d: new symbol = %d%n%n", i, r[i]);
                }
            }
        }

        if (debug) {
            System.out.println("Transmitted vector with errors (r): " + Arrays.toString(r));
        }
        return r;
    }
```

Kodas 4: Perdavimo kanalas

```java
/**
     * Finds the coset leaders for each syndrome without a maximum weight constraint.
     *
     * @param H parity-check matrix
     * @return map of coset leaders
     */
    public Map<String, CosetLeader> findCosetLeaders(int[][] H) {
        int n = H[0].length; // Length of the codeword
        int m = H.length; // Number of syndromes

        Map<String, CosetLeader> cosetLeadersMap = new LinkedHashMap<>();
        initializeCosetLeaders(cosetLeadersMap, m, n);

        if (debug) {
            System.out.println("\n=== Finding All Optimal Coset Leaders ===");
            System.out.println("Syndrome | Error Pattern | Hamming Weight");
        }

        int weight = 0;
        int unassignedSyndromes = cosetLeadersMap.size();

        // Continue generating error patterns until all coset leaders are assigned with optimal weights
        while (unassignedSyndromes > 0) {
            List<int[]> errorPatterns = generateErrorPatterns(n, weight);
            unassignedSyndromes -= updateCosetLeaders(cosetLeadersMap, H, errorPatterns, weight);
            weight++;
        }

        if (debug) {
            System.out.println("Total coset leaders found: " + cosetLeadersMap.size());
        }

        return cosetLeadersMap;
    }

    private void initializeCosetLeaders(Map<String, CosetLeader> cosetLeadersMap, int m, int n) {
        int totalSyndromes = (int) Math.pow(2, m);
        for (int i = 0; i < totalSyndromes; i++) {
            int[] syndrome = new int[m];
            for (int j = 0; j < m; j++) {
                syndrome[j] = (i >> (m - 1 - j)) & 1;
            }
            String syndromeStr = Arrays.toString(syndrome);
            cosetLeadersMap.put(syndromeStr, new CosetLeader(syndrome, null, n + 1));
        }
    }









    /**
     * Updates coset leaders map with new error patterns and returns the count of updated syndromes.
     *
```

```
60      * @param cosetLeadersMap map of coset leaders
61      * @param H               parity-check matrix
62      * @param errorPatterns   list of error patterns of the current weight
63      * @param weight          current weight of error patterns
64      * @return count of updated syndromes
65      */
66     private int updateCosetLeaders(Map<String, CosetLeader> cosetLeadersMap, int[][] H, List<int[]>
        errorPatterns, int weight) {
67         int updatedCount = 0;
68         for (int[] errorPattern : errorPatterns) {
69             int[] syndrome = computeSyndrome(H, errorPattern);
70             String syndromeStr = Arrays.toString(syndrome);
71 
72             CosetLeader existingLeader = cosetLeadersMap.get(syndromeStr);
73             if (existingLeader != null && weight < existingLeader.weight()) {
74                 // Update the coset leader with the new error pattern if it has a lower weight
75                 cosetLeadersMap.put(syndromeStr, new CosetLeader(syndrome, errorPattern, weight));
76                 updatedCount++;
77                 if (debug) {
78                     System.out.println(syndromeStr + " | " + Arrays.toString(errorPattern) + " | " + weight);
79                 }
80             }
81         }
82         return updatedCount;
83     }
```

Kodas 5: Coset lyderių paieška ir Sindromų skaičiavimas

```java
/**
    * Decodes a received vector using the parity-check matrix H and coset leaders.
    * @param r received vector
    * @param H parity-check matrix
    * @param cosetLeadersMap map of coset leaders
    * @return decoded vector
    */
   public int[] decodeStepByStep(int[] r, int[][] H, Map<String, CosetLeader> cosetLeadersMap) {
       int n = r.length;
       int i = 0;
       int[] rCopy = Arrays.copyOf(r, n);

       while (true) {
           int[] syndrome = computeSyndrome(H, rCopy);
           String syndromeStr = Arrays.toString(syndrome);
           CosetLeader cosetLeader = cosetLeadersMap.get(syndromeStr);
           int w = (cosetLeader != null) ? cosetLeader.weight() : n + 1;

           if (w == 0) {
               if (debug) {
                   System.out.println("Corrected codeword: " + Arrays.toString(rCopy));
               }
               return rCopy;
           }

           if (i >= n) {
               System.out.println("Error: Cannot decode the received vector.");
               return rCopy;
           }

           // Flip bit i
           rCopy[i] ^= 1;

           // Compute new syndrome and coset leader weight
           int[] syndromeNew = computeSyndrome(H, rCopy);
           String syndromeNewStr = Arrays.toString(syndromeNew);
           CosetLeader cosetLeaderNew = cosetLeadersMap.get(syndromeNewStr);
           int wNew = (cosetLeaderNew != null) ? cosetLeaderNew.weight() : n + 1;

           // If the new weight is less than the previous weight, keep the bit flipped
           if (wNew < w) {
               if (debug) {
                   System.out.printf("Flipped bit %d, new coset leader weight %d < %d%n", i + 1, wNew, w);
               }
           } else {
               rCopy[i] ^= 1;
           }
           i++;
       }
   }
```

Kodas 6: Dekodavimo procesas