



2016

Hemingov kod

Lazar Vasic 2013/0298

ELEKTROTEHNIČKI FAKULTET, BEOGRAD

Table of Contents

Tekst zadatka	2
Opis programa	2
Listing Programa	4
Test primeri I rezultati programa	10

Tekst zadatka

Implementacija Hemingovog koda:

Korisnik unosi:

- N (minimalno 7) – dužina kodne reči
- Da li želi detekciju dvostruke greške? Da li ima bit parnosti ili ne?
- K – dužinu informacione reči

Konstruiše se tabela (šablon za Hemingov kod), izračunavaju se zaštitni bitovi. Zatim se od korisnika traži da unese koliko gresaka se desilo u kodu, i pozicije grešaka. Pritom se izračunava sindrom i ispisuje komenar o uspešnosti prenete poruke. Program takođe određuje krajnju sekvencu kodne reči.

Opis programa

Fajlovi programa:

- I) main.cpp
- II) Heming.h
- III) Heming.cpp

- I) U main.cpp fajlu od korisnika se traži da unese odredjenje podatke (dužinu kodne reci – n , da li želi da hemingov kod radi sa bitom parnosti – detection, dužinu infomacione reči – k)
 Pri unosu svakog podatka vrši se provera da li su određeni podaci validni.
 Npr. Da li je n veće od 7, ako imamo k koje ne odgovara n.
 Od korisnika se traži ponovni unos tih podataka u slučaju pogresnog unosa.
 Detekcija dvosturke greske se unosi binarnim izborom korisnik bira izmedju 1 ili 0, Y (y) ili N(n).
 Zatim se pravi objekat klase Heming poziva se konsruktor i metoda code, koja izvrsava algortam.

```
Heming* H = new Heming(n, k, detection);
H->code();
```

II) Heming.h file sadrži deklaraciju metoda i polja.

Private (pomoćne) funkcije:

- 1) int calculateNumberOfSecurityBits(int k); - u zavisnosti od dužine informacione reči, ova funkcija vraća broj zaštitnih bita koji je potreban.
- 2) void insertCodeWord(); - ubacuje u niz infWord[] informacione bite koje korisnik unosi u binarnom obliku i pravi tabelu.
- 3) void initCodeWord(); - unosi informacione bite u kodnoj reči na odgovarajuće pozicije, ostavlja prostor za zaštitne bite.
- 4) void insertSecurityBits(); - izračunava zaštitne bitove i unosi ih u kodnoj reči. Ovde se izračunava i bit parnosti ako je to bilo zahtevano od strane korisnika
- 5) bitPermutation(int e); - invertuje bit. Ova funkcija se koristi kada korisnik unosi grešku/e. Takođe korisnik ovde unosi broj grešaka koji se javljaju.
- 6) void invertBit(int e, bool print = false); - pomoćna funkcija funkcije (5)
- 7) void findSindrom(); - izračunava sindrom.
- 8) void writeCommenar(); - ispisuje komentar u zavisnosti od rezultata.

Public funkcija code() – ova funkcija poziva predhodne metode kako bi izvršila celokupno kodiranje:

```
void Heming::code() {

    insertCodeWord();
    initCodeWord();
    insertSecurityBits();

    if (bitPermutation(-1)) {
        findSindrom();
        invertBit(sindrom, true);
        writeCommenar();
    }
}
```

III) Heming.cpp file sadrži definiciju metoda, klase Heming

Listing Programa

```

/*****HEMING.H*****/
#pragma once
#include <iostream>
#include <math.h>
using namespace std;
#define MAX_SECURITY 9
#define MAX LENGHT 100
#define INIT_ERROR 205
#define BIT LENGHT 8
#define ERRORS 2

class Heming {
private:
    int n, k, sindrom, sindromLastBit, numOfSecurSecurityBits;
    short errors[ERRORS];
    bool d, infWord[MAX LENGHT], securityBits[MAX_SECURITY] = { false };
    int codeWord[MAX LENGHT] = { 0 }, codeWordWithError[MAX LENGHT] = { 0 }
};

    void insertCodeWord();
    int calculateNumberOfSecurityBits(int k);
    void initCodeWord();
    void insertSecurityBits();
    bool bitPermutation(int e);
    void invertBit(int e, bool print = false);
    void findSindrom();
    void writeCommenar();

public:
    Heming(int n, int k, bool d);
    void code();
    ~Heming();
};

/*****HEMING.CPP*****/
#include <iostream>
#include <math.h>
#include <bitset>
#include "Heming.h"
using namespace std;

```

```

int Heming::calculateNumberOfSecurityBits(int k) {
    if (k >= 2 && k <= 4) {
        return 3;
    }
    else if (k >= 5 && k <= 11) {
        return 4;
    }
    else if (k >= 12 && k <= 26) {
        return 5;
    }
    else if (k >= 27 && k <= 57) {
        return 6;
    }
    else if (k == 1) {
        return 2;
    }
}

void Heming::insertCodeWord() {
    //Unos informacionih bita
    cout << "Unesi kodnu rec bit po bit:" << endl;
    for (int i = 0; i < k; ) {
        char temp;
        cin >> temp;

        if (temp == '0')         infWord[i++] = false;
        else if (temp == '1')    infWord[i++] = true;
        else                     cout << "Uneti ponvo " << i+1 << "-ti bit
informacione reci" << endl;
    }

    cout << endl << "====Sablon====" << endl;

    this->numOfSecurSecurityBits = calculateNumberOfSecurityBits(k);

    for (int i = 1, maska = 1, j = 1, p = 1; i <= n; i++) {
        string binary = bitset<BIT_LENGTH>(i).to_string();
        cout << i << "\t" << binary << "\t";
        if (i & maska) {
            if (j > numOfSecurSecurityBits)
                cout << "---Z" << endl;
            else
                cout << "Z" << j << endl;
            maska <<= 1;
            j++;
        }
        else {
            if (p > k)
                cout << "---" << endl;
            else
                cout << "I" << p << endl;
            p++;
        }
    }

    n = k + numOfSecurSecurityBits;
}

```

```

}

void Heming::initCodeWord() {
    int maska = 1, j = 0;

    for (int i = 0; i < n; i++) {
        if ((i + 1) & maska) {
            codeWord[i] = -1;
            maska <<= 1;
        }
        else
            codeWord[i] = infWord[j++];
    }

    cout << "Kodna rec bez zastinih bita je: " << endl;

    for (int i = 0, j = 0, maska = 1; i < n; i++) {
        if ((i + 1) & maska) {
            cout << "Z" << ++j << " ";
            maska <<= 1;
        }
        else
            if (codeWord[i] == INIT_ERROR)
                cout << "X" << " ";
            else
                cout << codeWord[i] << " ";
    }
    if (d)
        cout << "ZP";

    cout << endl;
}

void Heming::insertSecurityBits() {
    //Odredjivanje zastitnih bita (bez bita parnosti)
    for (int i = 0, maska = 1; i < numOfSecurSecurityBits; i++, maska <<=1)
    {
        for (int j = 0; j < n; j++) {
            if (codeWord[j] != -1 && (j+1)&maska ) {
                if (codeWord[j] == INIT_ERROR) continue;
                securityBits[i] ^= codeWord[j];
            }
        }
    }
    //Ubacivanje zastgitnih bita u poruku koju prenostimo (bez bita
    parnosti)
    for (int i = 0, j = 0; i < n; i++) {
        if (codeWord[i] == -1) {
            if (codeWord[i] != INIT_ERROR)
                codeWord[i] = securityBits[j++];
        }
    }

    //Odredjivanje bita parnosti i ubacivanje u poruku, ako je zahtevano!
    if (d) {

```

```

        for (int i = 0; i < n; i++)
            if (codeWord[i] != INIT_ERROR)
                securityBits[numOfSecurSecurityBits] ^= codeWord[i];

        codeWord[n] = securityBits[numOfSecurSecurityBits];
    }

    //Ispis zastitnih bitova
    cout << "Zastitni biti: " << endl;
    for (int i = 0; i < numOfSecurSecurityBits + d; i++)
        cout << "Z" << i + 1 << " = " << securityBits[i] << endl;
    cout << endl;

    //Ispis poruke koju trebamo da prenesemo (jos se nije desila greska)!
    n += d;
    cout << "Kodna rec od N = " << n << " bita:" << endl;

    for (int i = 0; i < n; i++) {
        cout << codeWord[i] << " ";
    }
    cout << endl;
}

void Heming::invertBit(int e, bool print) {
    for (int i = 0; i < n; i++) {
        if (i + 1 == e) {
            if (codeWordWithError[i] == 0)
                codeWordWithError[i] = 1;
            else
                codeWordWithError[i] = 0;
        }
    }

    if (print) {
        cout << "Kod posle korigovanje greske na mestu koje nam sindrom
pokazuje: " << endl;
        for (int i = 0; i < n; i++)
            cout << codeWordWithError[i] << " ";
        cout << endl;
    }
}

bool Heming::bitPermutation(int e) {

    for (int i = 0; i < n + d; i++)
        codeWordWithError[i] = codeWord[i];
    //Greska na bitu!
    if (e == -1) {
        short numOfError = 0;
        cout << "Unesi broj gresaka (1 ili 2)" << endl;
        cin >> numOfError;

        if (numOfError == 0) {
            cout << "Nema greske!" << endl;
            return false;
        }
        for (short i = 0; i < numOfError; i++) {

```



```

        cout << "Unesi poziciju " << i+1 << ". greske: " << endl;
        cin >> errors[i];

        while (errors[i] < 0 || errors[i] > n + d) {
            cout << "Pogresna vrednost je uneta za poziciju
greske, unesi ponovo" << endl;
            cin >> errors[i];
        }
        if(errors[i] != 0)
            invertBit(errors[i],false);
    }
}

//Ispis
cout << "Kod sa greskom: " << endl;
for (int i = 0; i < n; i++)
    cout << codeWordWithError[i] << " ";
cout << endl;

return true;
}

void Heming::findSindrom() {

    int lengthSyndrom = ceil(log2(n));

    sindrom = 0;

    for (int i=0,s= 0, maska = 1; i < lengthSyndrom;i++,s=0, maska <= 1) {
        for (int j = 0; j < n; j++) {
            if ((j + 1)&maska) && (codeWordWithError[j] != NIT_ERROR)){
                s ^= codeWordWithError[j];
            }
        }
        sindrom |= (s << i);
    }
    cout << "Sindorm: " << sindrom << endl;

    if (d) {
        sindromLastBit = 0;
        for (int i = 0;i < n + d;i++)
            sindromLastBit ^= codeWordWithError[i];
    }
}

void Heming::writeCommenar() {
    if (d) {
        cout << "Poslednji bit :" << codeWordWithError[n - 1] << endl;
        if (sindrom == 0) {
            if (sindromLastBit == 0)
                cout << "Nije bilo greske!" << endl;
            else
                cout << "Greska je na bitu parnosti!" << endl;
        }
    }
}

```

```

        else {
            if (sindromLastBit == 0)
                cout << "Paran broj gresaka! Ispravljen kod nije
validan, sindrom ne pokazuje poziciju greske" << endl;
            else
                cout << "Neparan broj gresaka! Sindrom pokazuje
gresku. Greska je bila na " << sindrom << "-bitu" << endl;
        }

    }
    else {
        if (sindrom == 0) {
            cout << "Greske nije bilo!" << endl;
        }
        else cout << "Greske je bilo na " << sindrom << "-bitu" << endl;
    }
}

Heming::Heming(int n, int k, bool d) {
    this->n = n;
    this->k = k;
    this->d = d;
}

void Heming::code() {

    insertCodeWord();
    initCodeWord();
    insertSecurityBits();

    //Ako vrati false => nije se desila greska, program se završava u
suprotnom nastavljamo sa algoritmom
    if (bitPermutation(-1)) {
        findSindrom();
        invertBit(sindrom, true);
        writeCommenar();
    }
}

Heming::~~Heming() {
    n = k = d = 0;

    delete infWord;
    delete securityBits;
    delete codeWord;
    delete codeWordWithError;
}

/*****MAIN.CPP*****/
#include<iostream>
#include "Heming.h"
using namespace std;

#define N 40

```

```

int main() {

    cout << "=====Hemingov kod===== " << endl << endl;

    /*****USER INSERT*****/
    int n;
    cout << "Uneti duzinu kodne reci (minimalno 7)" << endl;
    cin >> n;

    while (n < 7) {
        cout << "Broj n je manji od 7, uneti ponovo" << endl;
        cin >> n;
    }

    char yn;
    bool detection;

    cout << "Da li zelite detekciju 2 greske? y/n (Y/N) (1/0)" << endl;

    while (true) {
        cin >> yn;

        if (yn == 'y' || yn == 'Y' || yn == '1')
            detection = true;
        else if (yn == 'n' || yn == 'N' || yn == '0')
            detection = false;

        if (yn == 'y' || yn == 'Y' || yn == 'n' || yn == 'N') break;

        cout << "Uneta je pogresna opcija! Unesi ponovo y/n(Y/N)" <<
endl;

    }

    int k;
    cout << "Uneti duzinu informacionih bita" << endl;
    cin >> k;

    int uslovK = k + ceil(log2(k + 1)) + detection;
    cout << "Uslov za K: " << uslovK << endl;
    while (k < 1 || uslovK > n) {
        cout << "Unesi ponovo K" << endl;
        cin >> k;
        uslovK = k + ceil(log2(k + 1)) + detection;
    }

    /*****END USERR INSERT*****/

    Heming* H = new Heming(n, k, detection);
    H->code();

    system("PAUSE");
    return 0;
}

```

Test primeri I rezultati programa

(7, 4) – bez bita parnosti, jedna greška

=====Hemíngov kod=====

Uneti dužinu kodne reci (minimalno 7)

7

Da li želite detekciju 2 greske? y/n (Y/N)(1/0)

n

Uneti dužinu informacionih bita

4

Uslov za K: 7

Unesi kodnu rec bit po bit:

1100

=====Sablon=====

1	00000001	Z1
2	00000010	Z2
3	00000011	I1
4	00000100	Z3
5	00000101	I2
6	00000110	I3
7	00000111	I4

Kodna rec bez zastinih bita je:

Z1 Z2 1 Z3 1 0 0

Zastitni biti:

Z1 = 0

Z2 = 1

Z3 = 1

Kodna rec koja se sastoji od N = 7 bita:

0 1 1 1 1 0 0

Unesi broj gresaka (1 ili 2)

1

Unesi poziciju 1. greske:

5

Kod sa greskom:

0 1 1 1 0 0 0

Sindorm: 5

Kod posle korigovanje greske na mestu koje nam sindrom pokazuje:

0 1 1 1 1 0 0

KOMENTAR:

Greske je bilo na 5-bitu

(8,4) – sa bitom parnosti, 2 greške

=====Hemingov kod=====

Uneti duzinu kodne reci (minimalno 7)

8

Da li zelite detekciju 2 greske? y/n (Y/N)(1/0)

y

Uneti duzinu informacionih bita

4

Uslov za K: 8

Unesi kodnu rec bit po bit:

1011

=====Sablon=====

1	00000001	Z1
2	00000010	Z2
3	00000011	I1
4	00000100	Z3
5	00000101	I2
6	00000110	I3
7	00000111	I4
8	00001000	---Z

Kodna rec bez zastinih bita je:

Z1 Z2 1 Z3 0 1 1 ZP

Zastitni biti:

Z1 = 0

Z2 = 1

Z3 = 0

Z4 = 0

Kodna rec koja se sastoji od N = 8 bita:

0 1 1 0 0 1 1 0

Unesi broj gresaka (1 ili 2)

2

Unesi poziciju 1. greske:

4

Unesi poziciju 2. greske:

8

Kod sa greskom:

0 1 1 1 0 1 1 1

Sindorm: 4

Kod posle korigovanje greske na mestu koje nam sindrom pokazuje:

0 1 1 0 0 1 1 1

KOMENTAR:

Paran broj gresaka! Ispravljen kod nije validan, sindrom ne pokazuje poziciju greske

(16, 11) sa bitom parnosti, greška na bitu parnosti

```

=====Hemingov kod=====

Uneti duzinu kodne reci (minimalno 7)
16

Da li zelite detekciju 2 greske? y/n (Y/N)(1/0)
y

Uneti duzinu informacionih bita
11
Uslov za K: 16

Unesi kodnu rec bit po bit:
1011 0101 010

=====Sablon=====
1      00000001      Z1
2      00000010      Z2
3      00000011      I1
4      00000100      Z3
5      00000101      I2
6      00000110      I3
7      00000111      I4
8      00001000      Z4
9      00001001      I5
10     00001010      I6
11     00001011      I7
12     00001100      I8
13     00001101      I9
14     00001110      I10
15     00001111      I11
16     00010000      ---Z

Kodna rec bez zastinih bita je:
Z1 Z2 1  Z3 0  1  1  Z4 0  1  0  1  0  1  0  ZP

Zastitni biti:
Z1 = 0
Z2 = 1
Z3 = 0
Z4 = 1
Z5 = 0

Kodna rec koja se sastoji od N = 16 bita:
0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0

Unesi broj gresaka (1 ili 2)
1
Unesi poziciju 1. greske:
16

Kod sa greskom:
0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 1
Sindorm: 0

Kod posle korigovanje greske na mestu koje nam sindrom pokazuje:
0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 1

KOMENTAR:
Greska je na bitu parnosti!

```

(32, 20) bez bita parnosti, jedna greška

=====Hemingov kod=====

Uneti duzinu kodne reci (minimalno 7)

31

Da li zelite detekciju 2 greske? y/n (Y/N)(1/0)

n

Uneti duzinu informacionih bita

20

Uslov za K: 25

Unesi kodnu rec bit po bit:

1010 1111 0100 1000 0110

=====Sablon=====

1	00000001	Z1
2	00000010	Z2
3	00000011	I1
4	00000100	Z3
5	00000101	I2
6	00000110	I3
7	00000111	I4
8	00001000	Z4
9	00001001	I5
10	00001010	I6
11	00001011	I7
12	00001100	I8
13	00001101	I9
14	00001110	I10
15	00001111	I11
16	00010000	Z5
17	00010001	I12
18	00010010	I13
19	00010011	I14
20	00010100	I15
21	00010101	I16
22	00010110	I17
23	00010111	I18
24	00011000	I19
25	00011001	I20
26	00011010	---
27	00011011	---
28	00011100	---
29	00011101	---
30	00011110	---
31	00011111	---

Kodna rec bez zastinih bita je:

Z1 Z2 1 Z3 0 1 0 Z4 1 1 1 1 0 1 0 Z5 0 1 0 0 0 0 1 1 0

Zastitni biti:

Z1 = 0

Z2 = 1

Z3 = 0

Z4 = 0

Z5 = 1

Kodna rec koja se sastoji od N = 25 bita:

0 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0

Unesi broj gresaka (1 ili 2)

1

Unesi poziciju 1. greske:

18

Kod sa greskom:

0 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 1 0

Sindorm: 18

Kod posle korigovanje greske na mestu koje nam sindrom pokazuje:

0 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 1 0

KOMENTAR:

Greske je bilo na 18-bitu

(15, 7) bez bita parnosti, bez greške

=====Hemíngov kod=====

Uneti duzinu kodne reci (minimalno 7)
15

Da li zelite detekciju 2 greske? y/n (Y/N)(1/0)
n

Uneti duzinu informacionih bita
7
Uslov za K: 10

Unesi kodnu rec bit po bit:
101 1110

=====Sablon=====

1	00000001	Z1
2	00000010	Z2
3	00000011	I1
4	00000100	Z3
5	00000101	I2
6	00000110	I3
7	00000111	I4
8	00001000	Z4
9	00001001	I5
10	00001010	I6
11	00001011	I7
12	00001100	---
13	00001101	---
14	00001110	---
15	00001111	---

Kodna rec bez zastininih bita je:
Z1 Z2 1 Z3 0 1 1 Z4 1 1 0

Zastitni biti:
Z1 = 1
Z2 = 0
Z3 = 0
Z4 = 0

Kodna rec koja se sastoji od N = 11 bita:
1 0 1 0 0 1 1 0 1 1 0

Unesi broj gresaka (1 ili 2)
1
Unesi poziciju 1. greske:
0

Kod sa greskom:
1 0 1 0 0 1 1 0 1 1 0
Sindorm: 0

Kod posle korigovanje greske na mestu koje nam sindrom pokazuje:
1 0 1 0 0 1 1 0 1 1 0

KOMENTAR:
Greske nije bilo!