Zaštitni kodovi

Dragišić, Vanja

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Stručni studij

ZAŠTITNI KODOVI

Završni rad

Vanja Dragišić

Osijek, 2017

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 20.09.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Vanja Dragišić
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	a4020, 30.09.2014.
OIB studenta:	08335164829
Mentor:	Mr.sc. Anđelko Lišnjić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Vanja Mandrić-Radivojević
Član Povjerenstva:	Izv. prof.dr.sc. Slavko Rupčić
Naslov završnog rada:	Zaštitni kodovi
Znanstvena grana rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak završnog rada	Zadatak je teoretski obraditi zaštitne kodove; Hamming, Reed-Muller i CRC, te u jednom od programskih jezika napraviti program za kodiranje dekodiranje podataka u svakom od navedenih kodova.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	20.09.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 30.09.2017.

Ime i prezime studenta:	Vanja Dragišić
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	a4020, 30.09.2014.
Ephorus podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Zaštitni kodovi**

izrađen pod vodstvom mentora Mr.sc. Anđelko Lišnjić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Informacijsko i digitalno doba u kojem danas živimo donosi mnoge zahtjeve i potencijalne probleme koji u dobu začetaka informacijskog doba nisu bili toliko očiti iz vrlo jednostavnog shvaćanja informacija pa se tako razvitkom te teorije počela uviđati potreba za njihovim rješavanjem.

Potencijalni problem na koje treba obratiti pozornosti su sa koliko bitova treba predstaviti informaciju koju šaljemo, koji će biti omjer informacijskih i zaštitnih bitova, koja će biti brzina prijenosa odnosno kako informaciju prenijeti što brže, što točnije i sa što manje utrošene energije te na kraju kako možemo na siguran način tu informaciju prenijeti kroz komunikacijski kanal.

Digitalni prijenos postao je glavni način prijenosa podataka. Kada se govori o prenošenju informacije na ispravan i siguran način potrebno je definirati koja kodiranja to omogućuju pa je tako entropijsko kodiranje zaduženo za ispravnost poslane informacije a zaštitno kodiranje za sigurnost prijenosa informacije.

Prvenstveno je potrebno definirati što je to točno informacija te koja su njezina svojstva? Teorija koja stoji iza informacije započeo je Claude E. Shannon 1948. godine u svom djelu „Matematička teorija komunikacije“ („*A Mathematical Theory of Communication*“). U tom djelu informaciju se povezuje sa vjerojatnošću pojave pojedinih znakova u nizu riječi. Informaciju također možemo definirati kao svaki podatak koji smanjuje stupanj neizvjesnosti ili kao fundamentalnu veličinu koja nije svojstvo niti materije niti energije.

Neka od svojstava informacije koje također trebamo uzeti u obzir su spontanost, nepredvidivost, slojevitost, cirkularnost te povratnost.

Informacija koja se šalje mora biti zapisana u nekakvom fizičkom obliku koji je pogodan za slanje putem medija.

Kodiranje je jedan od tih koraka u kojem se uneseni podatci pretvaraju u niz znakova odgovarajućeg formata pazeći pri tome da je taj proces reverzibilan odnosno da primalac dobije iste podatke koji su poslani te se putem prijenosnog medija šalju ka primaocu, što u konačnici dovodi do koraka dekodiranja odnosno ponovnog pretvaranja niza znakova u razumljive podatke. U cijelom ovom procesu postoji vjerojatnost pogrešnog prijenosa i to je jedan od osnovnih zadataka zaštitnih kodova, svesti na minimum vjerojatnost pogreške. Ispravljanje pogrešaka je drugi zadatak zaštitnog kodiranja.

Ovaj rad obrađuje zaštitno kodiranje putem Hammingovog kodera i cikličke redundantne provjere (engl. *cycle redudancy check*). U prvom dijelu iznijete su teoretske osnove zaštitnog

kodiranja. Drugi dio rada opisuje primjenu zaštitnog kodiranja u praksi, izrađeni su Hammingovi koder i dekoder (7/4 te 15/11) te koder i dekoder cikličke redundantne provjere (engl. *cycle redudancy check*) u obliku mrežne aplikacije u .NET tehnologiji.

1.1. Zadatak završnog rada

Zadatak završnog rada je: *„ Teoretski obraditi temu zaštitnog kodiranja kroz obrađivanje teorije linearnog kodiranja, Hammingovog koda te cikličke redundantne provjere (engl. cycle redudancy check) sa svim pripadajućim potrebnim terminima. Izraditi koder i dekoder za Hammingovo kodiranje te koder i dekoder cikličke redundantne provjere u obliku mrežne aplikacije.“*
Tehnologije za izradu zadatka: *“.NET.“*

2. LINEARNI BLOK KODOVI

[1] Linearni blok kodovi su široko raširenи kodovi zbog lakoće implementacije i svoje jednostavnosti. Njihov oblik je [n, k] gdje je n dužina kodne riječi a k predstavlja dimenziju potprostora. [2] Ako je blok kod K potprostor vektorskog prostora V(n): $K \subset V(n)$, x i y su kodne riječi koda K i ako je $a \in \{0, 1\}$ te su za svaki x, y i a ispunjeni uvjeti:

1. $x + y \in K$
2. $a * x \in K$

Onda kažemo da je K linearan binarni blok kod, odnosno linearni blok kodovi definiraju se preko skupa vektora nad kojim su definirane određene operacije.
[1] Sva računanja obavljaju se u modulo 2 aritmetici i to kako je dano tablicom (Tab. 2.1.) :

Tablica 1. Modulo 2 aritmetika

x_1	x_2	$x_1 + x_2$	$x_1 * x_2$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Kako bi se moglo odraditi zaštitno kodiranje linearnim blok kodom potrebno je definirati njegovu generirajuću matricu G. [1] Ona se definira kao matrica k * n čiji se reci sastoje od vektora baze koda (n, M, d), gdje je n broj bitova u horizontalnom retku matrice, M broj bitova u vertikalnom retku matrice i d potprostor koji se koristi za generirajuću matricu. Ona nam olakšava zapis linearnog kod te pojednostavljuje operacije kodiranja i dekodiranja.
Primjerice, za linearni blok kod:

$K = (4, 3, 2)$

$$K = \begin{cases} 0\ 0\ 0\ 1 \\ 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{cases}$$

$$G = \begin{bmatrix} 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{bmatrix}$$

Tada se za svaku kombinaciju bitova računa kodna riječ. U ovom slučaju se računa za kombinaciju 2^2 bitova.

$0 * [0\ 1\ 0\ 0] + 0 * [1\ 1\ 1\ 1] = [0\ 0\ 0\ 0]$

$0 * [0\ 1\ 0\ 0] + 1 * [1\ 1\ 1\ 1] = [1\ 1\ 1\ 1]$

$1 * [0\ 1\ 0\ 0] + 0 * [1\ 1\ 1\ 1] = [0\ 1\ 0\ 0]$

$1 * [0\ 1\ 0\ 0] + 1 * [1\ 1\ 1\ 1] = [1\ 0\ 1\ 1]$

2.1. Kodiranje linearnim blok kodom

Linearni blok kod oblika [n, k] ima mogućnost kodiranja 2^k riječi koda. Generiranje poruke se u suštini svodi na umnožak vektor retka matrice i generirajuće matrice G a dana je izrazom.

$x = \sum_{i=1}^{k}(m * r) = m * G$ (2-1)

Primjer kodiranja linearnim blok kodom:

Zadana je generatorska matrica kao u prošlom primjeru

$G = \begin{bmatrix} 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{bmatrix}$

A poruka koja se šalje je „01“

Kodiranje se vrši prema formuli, odnosno:

$[0\ 1] * \begin{bmatrix} 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{bmatrix} = [0\ 1\ 1\ 1]$

Ovaj proces moguće je skratiti korištenjem standardnog oblika generirajuće matrice koda dano izrazom:

$m * [Ik \mid A] = \{m, m * A\}$ (2-2)

U ovom slučaju prvih k pozicija zauzima poruka a ostale pozicije zauzima umnožak m * A, gdje taj umnožak predstavlja dio za provjeru.

2.2. Dekodiranje linearnim blok kodom

Ako se promatra K = (4, 3, 2) sa kodnim riječima:

$$K = \begin{cases} 0\ 0\ 0\ 1 \\ 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{cases}$$

Pošto je udaljenost koda 2 kod može detektirati jednostruku pogrešku i ispraviti jednu pogrešku. Dekoder može za primljenu kodnu riječ odrediti najmanju Hammingovu distancu, koja se definira kao broj pozicija na kojima kodne riječi imaju različite simbole (stranica 131 …), i proglasiti ju kodnom riječi no ukoliko je ta udaljenost veća od 1 tada se utvrđuje dvostruka ili višestruka pogreška. Ovo je vrlo sporo rješenje pa se umjesto toga koristi tzv. sindromsko dekodiranje. Kako bi se sindromsko dekodiranje moglo izvršiti potrebno je definirati vektor pogreške, standardni niz te matricu provjere pariteta za koju je pak potrebno definirati dualni kod.

Vektor pogreške se definira na način da ako se na predajniku pošalje kombinacija bitova x = [1 0 1 1] a prijemna strana primi kombinaciju bitova y = [1 1 0 1] pomoću vektora pogreške koji je definiran kao razlika ta dva vektora e = y – x = [e_1 e_2 … e_n] može se detektirati do dvije pogreške,

$x = [1\ 0\ 1\ 1]$

$y = [1\ 1\ 0\ 1]$

$e = [0\ 1\ 1\ 0]$,

gdje nam „1“ određuje poziciju pogreške unutar vektora, a „0“ određuje poziciju ispravnog bita unutar vektora.

[1] Standardni niz kod K definira se kao tablica čiji prvi redak popunjavaju kodne riječi kod K, s tim da je prva kod na riječ 0. U svim ostalim recima nalaze se razredi standardnog niza K nastali na način da se kodnim riječima kod K dodavali svi vektori pogreške e, koje kod K može otkriti i ispraviti.

Ukoliko se uzima pretpostavka da nije nastala višestruka pogreška, dekodiranje se vrši na način da se primljena kombinacija bitova y traži unutar tablice standardnog niza i ukoliko ta kombinacija nije pronađena tada se detektira pogreška koja se ne može ispraviti. Ukoliko se

primljena kombinacija bitova y pronađe unutar tablice standardnog niza tada se ispravljanje pogreške vrši na način da se toj kombinaciji bitova x određuje i-ta kodna riječ koja zatim postaje primljena kodna riječ, postupak funkcionira na način pretpostavke da je to jedina kombinacija bitova koje je mogla biti poslana ukoliko se dogodila jednostruka pogreška.

Primjer dekodiranja jednostavnim standardnim nizom:
Ukoliko imamo tablicu standardnog niza (Tab. 2.2.):

Tablica 2. Tablica standardnog niza

0000	1100	0111	1011
0001	1101	0110	1010
0010	1110	0101	1001
0100	1000	0011	1111

Prvi red čine kodne riječi koda K dok svi ostali redove čine razredi kod K. Razredi koda K nastaju tako da se kodnim riječima koda K dodaju vektori pogreške e. Svi članovi nekog retka predstavljaju jedan razred koda K.

Poslana kombinacija bitova x = [1 1 0 0]
Primljena kombinacija bitova y = [1 1 1 0]
Primljena kombinacija bitova y = [1 1 1 0] pronalazi se u drugom stupcu, treći redak tablice standardnog niza K te se zaključuje jednostruka pogreška te se također zaključuje da je poslana kodna riječ kombinacija bitova x = [1 1 0 0] odnosno prva kombinacija bitova unutar stupca.

Matrica provjere pariteta uvedena je kako bi se riješio problem standardnog niza odnosno problem koji nastaje kada se radi sa matricama velikih dimenzija. Korištenjem metode standardnog niza nad matricama velikih dimenzija vrlo je procesorski zahtjevan zadatak pa se tako uvodi nova metoda odnosno metoda matrice provjere pariteta. Kako bi se definirala matrica provjere paritet potrebno je definirati dualni kod.

Dualni kod neka su x vektori kod K ($x \in K$). Skup svi vektora y vektorskog potprostora V(n), koji su ortogonalni na sve $x \in K$, čini dualni kod koda K i ima oznaku $K^{\perp}$:

$$K^{\perp} = \{y \in V(n) \mid \forall\, x \in K,\ y * x = 0\} \qquad (2\text{-}3)$$

Gdje je x * y skalarni produkt vektora:

$$x * y = \sum x_i * y_i \quad (2\text{-}4)$$

Za provjeru ispravnosti primljene kodne riječi može se koristiti skraćen postupak pošto je dualni kod ujedno i linearan. [1] Neka je K linearni blok kod [n, k], dualni kod $K^{\perp}$ koda K je blok kod [n, n-k].

Dokaz linearnosti dualnog koda $K^{\perp}$ leži u tome da ukoliko postoji matrica H koja se dobiva iz Dualnog kod $K^{\perp}$ da svi skalarni produkti između svih redaka generirajuće matrice G i generirajuće matrice dualnog kod H moraju biti 0 te prema tome zadovoljavaju jednadžbu $G * H^T = 0$. Prema tome zbog svojstava koje posjeduje za provjeru ispravnosti primljene kodne riječi ukoliko je poznata generirajuća matrica H svodi se na umnožak vektora i matrice. Ukoliko generirajuća matrica nije poznata ona se definira kao $H = [A^T \mid I_{n-k}]$ te mora zadovoljavati jednadžbu $G * H^T = 0$. Matrica H u ovom slučaju zbog svojih svojstava naziva se još i matrica provjere pariteta odnosno u svakom retku matrice H zbroj svih „1“ mora biti paran, tako se određuje da je primljena kodna riječ ispravna.

Kada se definiralo sve prethodno tada se može definirati sindromsko dekodiranje odnosno skraćeni oblik dekodiranja kod linearnih kodova.

Sindromsko dekodiranje je metoda koja nam omogućuje da složenost dekodiranja jedne riječi svedemo na razinu O(1). Temelj sindromskog dekodiranja je vektor koji se dobije množenjem primljene riječi i transponirane matrice provjere pariteta. Kako bi se sindromsko dekodiranje mogla izvesti potrebno je definirati što je to sindorm.

[2] Sindrom primljene kodne riječi y koda K s matricom provjere pariteta H je vektor dobiven umnoškom:

$$S(y) = y * H^T \quad (2\text{-}5)$$

Vrlo važno svojstvo sindroma je to da sve kodne riječi koje pripadaju unutar istog razreda koda K imaju isti sindrom, pa prema tome može se zaključiti da postoji preslikavanje jedan na jedan između vektora pogreške i sindroma. Dekodiranje tada postaje vrlo jednostavan postupak u kojem je potrebno odrediti sindrom S(y) primljene kodne riječi y te pomoću preslikavanja unutar dane tablice odrediti vektor pogreške e, potom se poslana kodna riječ dobije putem formule x = y – e. Ovom metodom određuje se jednostruka pogreška, no ukoliko unutar dane tablice ne postoji preslikavanje jedan na jedan tada se zaključuje da je nastala višestruka pogreška.

Primjer jednostavne tablice preslikavanja vektora pogreške e i sindroma kodne riječi S(y) (Tab. 2.3.):

Tablica 3. Tablice preslikavanja vektora pogreške e i sindroma kodne riječi S(y)

e	0000	0001	0010
S(y)	00	01	10

Definiranjem teorije linearnog blok koda vrlo lako se može iz njega uz korištenje stečenog znanja izlučiti te objasniti Hammingov i ciklički kod.

3. HAMMINGOV KOD

Hammingov kod je bilo koji linearni blok kod čija matrica provjere pariteta H ima r redaka, a u stupcima ima sve moguće vektore dimenzije r > 1 osim vektora 0. Stručno rečeno Hammingov kod se definira na slijedeći način.

Neka je r pozitivan cijeli broj i neka je H matrica dimenzija r * (2^r - 1) čije stupce sačinjavaju svi vektori dimenzije r različiti od 0 iz vektorskog prostora V(r). Matrica H je matrica provjere pariteta Hammingovog koda s oznakom Ham(r). (stranica 162.)

Svojstva Hammingovog koda:

1. linearni blok kod oblika [2^r – 1, 2' – 1 – r]
2. ukoliko ima najmanju udaljenost 3 otkriva dvostruku pogrešku te ispravlja jednostruku pogrešku
3. perfektan kod

3.1. Kodiranje Hammingovim kodom

Kodiranje Hammingovim kodom može se izvesti na dva načina, putem zadanih formula te pomoću generirajuće matrice. U nastavku biti će prikazana i objašnjenja oba načina.

Kodiranje pomoću zadanih formula:

Kako se unutar Hammingovog koda kodna riječ sastoji od informacijski bitova i zaštitnih bitova unaprijed određenih pozicija tako se ovaj način može prikazati putem formule: $c_1\ c_2\ i_1\ c_3\ i_2\ i_3\ i_4\ c_4\ i_5\ i_6\ i_7\ i_8\ i_9\ i_{10}\ i_{11}$ …, gdje oznaka c predstavlja poziciju zaštitnih bitova a oznaka i predstavlja poziciju informacijskih bitova.

Računanje zaštitnih bitova odvija se u modula 2 aritmetici i to pomoću sljedećih izvedenih formula:

$$c_1 = i_1 + i_2 + i_4 + i_5 + i_7 + i_9 + i_{11} \quad (3\text{-}1)$$

$$c_2 = i_1 + i_3 + i_4 + i_6 + i_7 + i_{10} + i_{11}$$

$$c_3 = i_2 + i_3 + i_4 + i_8 + i_9 + i_{10} + i_{11}$$

$$c_4 = i_5 + i_6 + i_7 + i_8 + i_9 + i_{10} + i_{11}$$

Provjera ispravnosti se vrši na način da zbroj svih zaštitnih bitova kodne riječi mora biti jednak 0.

Primjer kodiranja Hammingovim kodom pomoću zadane formule:
Kombinacija bitova koja se šalje na predajniku x = [1 0 1 0]
prema tome $i_1 = 1$, $i_2 = 0$, $i_3 = 1$, $i_4 = 0$. Računaju se zaštitni bitovi prema zadanim formulama
$c_1 = i_1 + i_2 + i_4 = 1 + 0 + 0 = 1$
$c_2 = i_1 + i_3 + i_4 = 1 + 1 + 0 = 0$
$c_3 = i_2 + i_3 + i_4 = 0 + 1 + 0 = 1$

Potom se putem zadane formule složi kodirana riječ koja glasi y = [1 0 1 1 0 1 0]. U ovom slučaju radi se o Hammingovom [7, 4] kodu gdje je ukupan broj bitova n = 7 a broj informacijskih bitova k = 4.
Ovaj način se može prikazati i u obliku tablice (Tab 3.1.):

Tablica 4. Prikaz kodiranja Hammingovim kodom pomoću tablice

c_1	c_2	i_1	c_3	i_2	i_3	i_4
1	2	3	4	5	6	7
x		x		x		x
	x	x			x	x
			x	x	x	x
1	0		1			

Iz tablice se iščitava pazeći pri tom na pozicije informacijskih i zaštitnih bitova kodna riječ y = [1 0 1 1 0 1 0]. Ova tablica predstavlja tablicu provjere paritet putem koje možemo dobiti generirajući matricu G.

Kodiranje pomoću generirajuće matrice:
Generirajuća matrica imati će 7 stupaca što je jednako dužini kodne riječi a pošto ovdje se radi o matrici [7, 4, 3] generirajuća matrica G će imati 4 retka.
Prvi bit kodne riječi y se dobiva na način da se poslana kombinacija bitova x množi sa prvim stupcem generirajuće matrice G, drugi bit kodne riječi y se dobiva na način da se poslana kombinacija bitova x množi sa drugim stupcem generirajuće matrice G itd. Ovim načinom dobije se generirajuća matrica oblika:

$$G = \begin{bmatrix} 1\,1\,1\,0\,0\,0\,0 \\ 1\,0\,0\,1\,1\,0\,0 \\ 0\,1\,0\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,0\,1 \end{bmatrix}$$

Provjera ispravnosti obavlja se na način da se poslana kombinacija bitova x množi sa generirajućom matricom G

$$[1\ 0\ 1\ 0] * \begin{bmatrix} 1\,1\,1\,0\,0\,0\,0 \\ 1\,0\,0\,1\,1\,0\,0 \\ 0\,1\,0\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,0\,1 \end{bmatrix} = [1\ 0\ 1\ 1\ 0\ 1\ 0]$$

3.2. Dekodiranje Hammingovim kodom

Dekodiranje Hammingovim kodom može se također napraviti na dva načina kao i kodiranje, putem zadanih formula te pomoću generirajuće matrice.

Dekodiranje pomoću zadanih formula:

Pošto se prema zadanim formula već zna na kojim pozicijama se nalaze informacijski odnosno zaštitni bitovi dekodiranje ovom metodom vrlo je lagano. Unaprijed definirana formula c_1 c_2 i_1 c_3 i_2 i_3 i_4 c_4 i_5 i_6 i_7 i_8 i_9 i_{10} i_{11} određuje te pozicije pa prema tome ukoliko je primljena kodna riječ y [1 0 1 1 0 1 0] zna se da c_1 iznosi „1“ c_2 iznosi „0“ i_1 iznosi „1“ c_3 iznosi „1“ i_2 iznosi „0“ i_3 iznosi „1“ te i_4 iznosi „0“ pa prema tome poslana kodna riječ je x [1 0 1 0] pod pretpostavkom da se pogreška nije dogodila.

Provjeru ispravnosti primljene kodne riječi može se odraditi također pomoću zadanih formula :

$c_1 = i_1 + i_2 + i_4$

$c_2 = i_1 + i_3 + i_4$

$c_3 = i_2 + i_3 + i_4$

I to na način da zbroj informacijskih bitova nam daje zaštitne bitove c_1 c_2 c_3 koji moraju računski odgovarati primljenim zaštitnim bitovima:

$c_1 = i_1 + i_2 + i_4 = 1 + 0 + 0 = 1$ – što odgovara primljenom c_1

$c_2 = i_1 + i_3 + i_4 = 1 + 1 + 0 = 0$ – što odgovara primljenom c_2

$c_3 = i_2 + i_3 + i_4 = 0 + 1 + 0 = 1$ – što odgovara primljenom c_3

To je dokaz da je primljena kodna riječ točna. S druge strane, kao pitanje se nameće što ako se primi kriva kodna riječ.

Neka je primljena kodna riječ y [1 0 0 1 0 1 0] sa greškom na poziciji i_1, ponovnim korištenjem formula može se odrediti točna pozicija pogreške

$c_1 = i_1 + i_2 + i_4 = 0 + 0 + 0 = 0$ – što ne odgovara primljenom c_1

$c_2 = i_1 + i_3 + i_4 = 0 + 1 + 0 = 1$ – što ne odgovara primljenom c_2

$c_3 = i_2 + i_3 + i_4 = 0 + 1 + 0 = 1$ – što odgovara primljenom c_3

Dakle greška je nastala ili na poziciji i_1 ili poziciji i_4 pošto su to zajedničke pozicije koje dijele zaštitni bitovi c_1 i c_2. Za poziciju i_4 može se zaključiti da je ispravno poslana pošto se ujedno nalazi unutar zaštitnog bita c_3 koji odgovara primljenom zaštitnom bitu c_3, pa se tako dedukcijom zaključuje da je pogreška nastala na poziciji i_1.

Dekodiranje pomoću generirajuće matrice:

Dekodiranje Hammingovim kodom putem generirajuće matrice obavlja se putem slijedećih koraka:

1. izračunavanje sindroma S(y) primljene kodne riječi y
2. ukoliko je S(y) različit od 0 potrebno je invertirati bit na odgovarajućoj poziciji decimalnom ekvivalentu sindroma

Primjer dekodiranja Hammingovim kodom pomoću matrice pariteta:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Pozicije unutar matrice odgovaraju matrici pariteta Hammingovog koda objašnjene u prethodnom poglavlju.

Ako se uzme kodna riječ y = [0 0 0 0 1 0 0] koja na 5. poziciji ima grešku te se preko nje računa sindrom S(y) zadanom formulom

$S(y) = y * H^T = [0\ 0\ 0\ 0\ 1\ 0\ 0] * H^T = [1\ 0\ 1]$

Dobiveni sindrom S(y) = [1 0 1] predstavlja u binarnom zapisu broj 5 te ujedno i poziciju pogreške unutar kodne riječi.

Kao problem ovog koda može se primjetiti to što jednostruka i dvostruka pogreška imaju sindrom jedne pogreške, koji je nastao kao posljedica perfektnosti koda. Ovaj problem ispravlja se proširenim Hammingovim kodom.

4. CIKLIČKA REDUNDANTNA PROVJERA (ENGL. *CYCLE REDUDANCY CHECK*)

Ciklička redundantna provjera (engl. *cycle redudancy check*) dio je ciklički kodovi koji su izvedenica linearnih blok kodova, za razliku od linearnih blok kodova kod cikličkih kodova generatorska i paritetna matrica se ne predstavljaju kao tablica već kao algoritam.

Ciklički kod kao i linearni blok kod je oblika [n, k] gdje je n ukupan broj bitova, a k broj informacijskih bitova te kako i ime kaže koristi metoda pomaka u lijevo odnosno desno umnoškom polinoma sa x kako bi se generirale druge kodne grupe. Ciklički kodovi su oblika c = (c_{n-1}, c_{n-2}, … c_1, c_0). Blok kod je cikličan ako je ujedno i linearni blok kod i ako bilo kakav pomak u bilo koju stranu ponovno daje kodnu riječ iz osnovnog blok koda.

Primjer cikličkog koda sa pomakom u lijevo:

c = (1 0 0 1)

c' = (0 0 1 1)

c'' = (0 1 1 0)

c'' = (1 1 0 0)

Kodiranje cikličkim kodom vrši se pomoću polinoma i to na način da se osnovnom obliku c dodaje umnožak x^n gdje n predstavlja komponentu kodne riječi.

$c = (c_{n-1}, c_{n-2}, \dots c_1, c_0)$ (4-1)

$c(x) = (c_{n-1} * x^{n-1}, c_{n-2} * x^{n-2}, \dots c_1 * x^1, c_0 * x^0)$

Ukoliko se taj postupak primjeni na kombinaciju bitova c = (1 0 0 1) te je n = 4 tada polinom ima oblik $c(x) = (1 * x^3 + 0 * x^2 + 0 * x^1 + 1 * x^0)$ odnosno $c(x) = (x^3 + 1)$ gdje polinom c(x) predstavlja kodnu riječ. Bilo koji x sa eksponentom pozicijski predstavlja „1“ a nedostatak x pozicijski predstavlja „0“ . Množenjem tog polinoma sa x dobijemo pomak u lijevo.

4.1. Kodiranje cikličkom redundantnom provjerom (engl. *cycle redudancy check*)

Da bi se kodiranje moglo izvršiti na ispravan način potreban je generirajući polinom koji se u pravilu dobije pomoću generirajuće matrice gdje su redci generirani cikličnim pomakom. U ovom slučaju pošto se radi o cikličnom kodiranju matrica se ne mora predstaviti kao tablica već se može predstaviti kao polinom što je velika prednost cikličnih kodova.

Pošto je u ovom slučaju broj informacijskih bitova 4 broj ukupnih bitova će biti 7 gdje njihova razlika 3 predstavlja paritetne bitove, prema tome radi se [7, 4] cikličnom kodu te je zadan generirajući polinom $g(x) = x^3 + x + 1$.

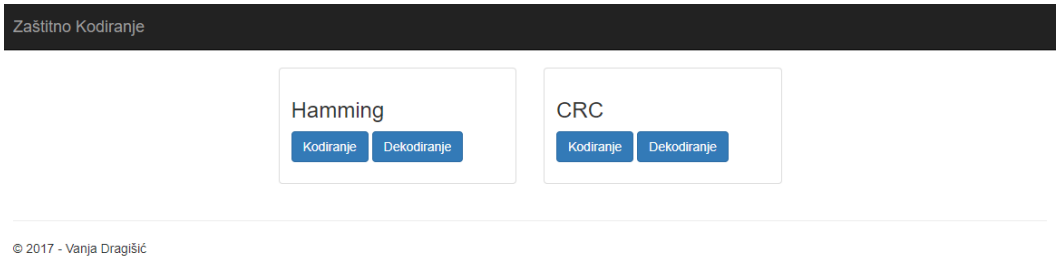
$c(x) = (x^3 + 1)$ koji predstavlja kodnu riječ množi se sa x^{n-k} odnosno sa x^3 te se dobije polinom $x^3c(x) = x^6 + x^3$, potom je potrebno odrediti polinom pariteta p(x) i to na način da se polinom $x^3c(x)$ dijeli sa generirajućim polinomom g(x) te se dobije paritetni polinom $p(x) = x^2 + x$. Nakon odrađenih tih operacija zbrajanjem paritetnog polinoma p(x) i polinoma $x^3c(x)$ dobije se polinom kodne riječi $U(x) = x^6 + x^3 + x^2 + x$ odnosno poslana riječ glasi 1 0 0 1 1 1 0.

4.2. Detekcija pogreške

Ciklička redundantna provjera (engl. *cycle redudancy check*) u svojoj osnovi nema mogućnost ispravljanja pogreške, pa se prema tome najviše koristi za detekciju pogreške. Detekcija pogreške obrnuti je proces nego proces kodiranja opisan u ovom radu, polinom koji primalac dobije sastoji se informacijskih bitova zbrojenih sa „CRC“ dijelom pa tako ako primljena kodna riječ ima 7 bitova ustvari se sastoji od 4 informacijska bita i 3 bita „CRC“ dijela. Obrnutim procesom od kodiranja odnosno dijeljenjem radi se provjera primljene kodne riječi na način da se provjerava ostatak, ukoliko postoji ostatak u ovom slučaju „1“ znamo da se na toj poziciji dogodila pogreška, ukoliko ostatak dijeljenja nema odnosno u ovom slučaju „0“ prijenos je ispravan. Prednost ovakvog sustava je u tome što se točno može odrediti pozicija nastale pogreške.

5. IZRADA APLIKACIJE ZA KODER I DEKODER ZAŠTITNOG KODIRANJA

Aplikacija je izrađena u tehnologiji .NET u obliku mrežne aplikacije. Tehnologija .NET pruža nam već gotovu radnu okolinu koja olakšava proces izrade aplikacije. Interaktivni početni zaslon aplikacije (Sl 5.1.) nudi opciju odabira Hammingovog kodera i dekodera ili CRC kodera i dekodera klikom na jedan od gumbova. Višak slobodnog mjesta namjerno je ostavljen zbog mogućnosti proširivanja aplikacije sa drugim koderima i dekoderima u budućnosti, opcija koju nam nudi tehnologija .NET.



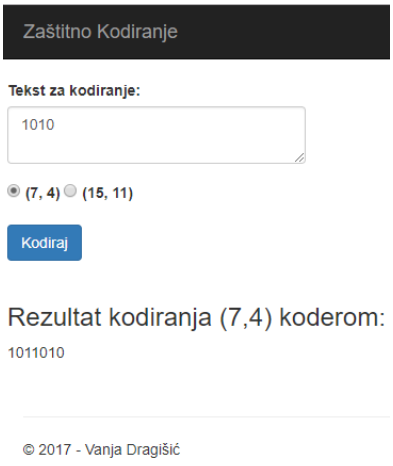
Slika 5.1. Interaktivni početni zaslon aplikacije

Odabirom opcije kodiranje kod Hammingovog koda aplikacija nas vodi u novi prozor (Sl 5.2.) u kojem imamo opciju unosa teksta za kodiranje te odabira u kojem koderu želimo kodirati taj tekst (Hammingov 7/4 koder ili Hammingov 15/11 koder).

Slika 5.2. Prozor za kodiranje teksta Hammingovim koderima

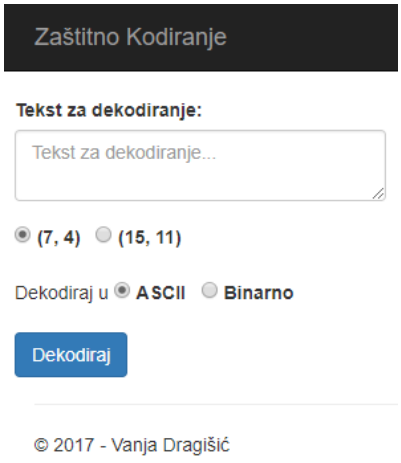
Aplikacija sama prepoznaje da li je uneseni tekst za kodiranje u binarnom ili znakovnom obliku te sukladno s time nastavlja daljnji proces kodiranja objašnjen u nastavku ovog poglavlja.

Kada je unesen tekst za kodiranje i kliknut gumb kodiraj pojavljuje nam se novi tekstualni prostor u kojem je ispisan kodirani tekst (Slika 5.3.).



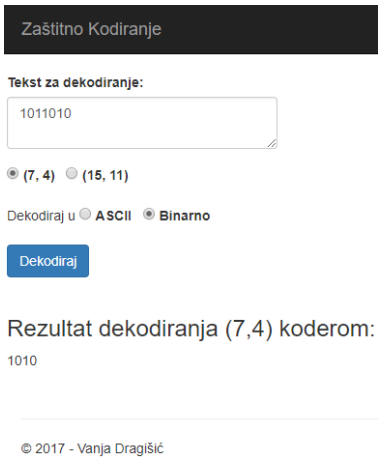
Slika 5.3. Rezultat kodiranja

Odabirom opcije dekodiranje kod Hammingovog koda aplikacija nas vodi u novi prozor (Sl 5.4.) u kojem imamo opciju unosa teksta za dekodiranje te odabira u kojem dekoderu želimo dekodirati taj tekst (Hammingov 7/4 dekoder ili Hammingov 15/11 dekoder) i u kojem obliku želimo dekodirati tekst (ASCII ili binarno).



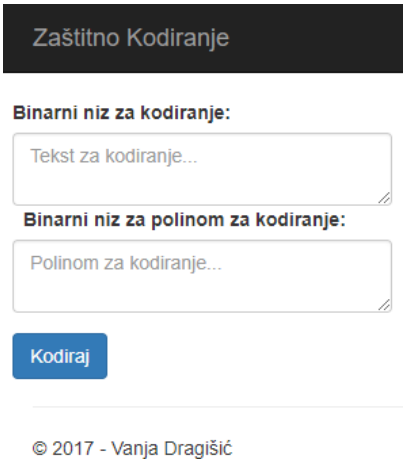
Slika 5.4. Prozor za dekodiranje teksta Hammingovim dekoderima

Aplikacija u ovom slučaju nema opciju prepoznavanja da li je uneseni tekst za dekodiranje u binarnom ili znakovnom obliku s obzirom da uneseni oblik za dekodiranje je uvijek u binarnom obliku. Kada je unesen tekst za dekodiranje i kliknut gumb dekodiraj pojavljuje nam se novi tekstualni prostor u kojem je ispisan dekodirani tekst (Sl 5.5.).



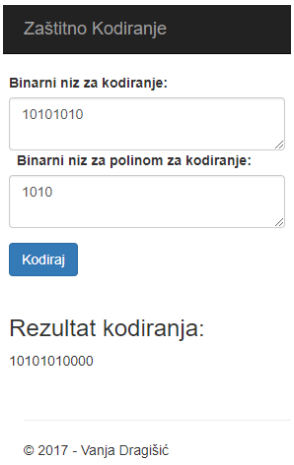
Slika 5.5. Rezultat dekodiranja

Odabirom opcije kodiranje kod CRC koda aplikacija nas vodi u novi prozor (Sl 5.6.) u kojem imamo opciju unosa teksta za kodiranje te opciju unosa binarnog niza koji će služiti kao polinom za kodiranje.



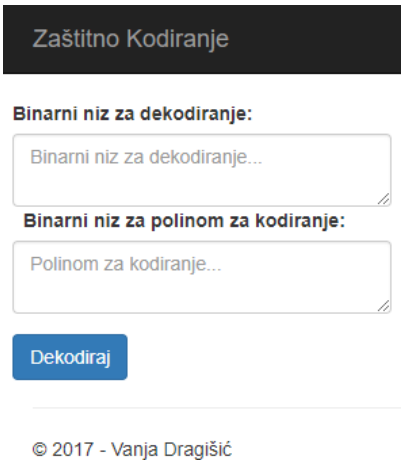
Slika 5.6. Prozor za kodiranje teksta CRC koderom

Aplikacija zahtjeva unos teksta u binarnom obliku. Kada je unesen tekst za kodiranje te binarni niz koji će služiti kao polinom za kodiranje i kliknut gumb kodiraj pojavljuje nam se novi tekstualni prostor u kojem je ispisan kodirani tekst (Sl 5.7.).



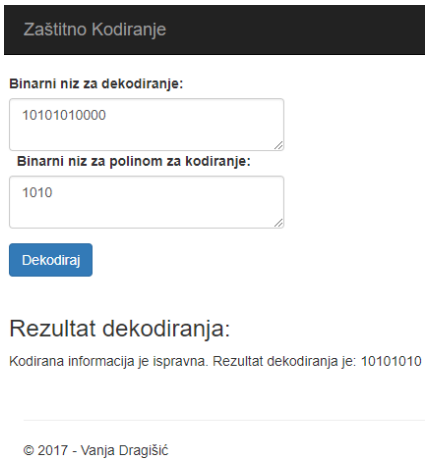
Slika 5.7. Rezultat kodiranja

Odabirom opcije dekodiranje kod CRC koda aplikacija nas vodi u novi prozor (Sl 5.8.) u kojem imamo opciju unosa teksta za dekodiranje te opciju unosa binarnog niza koji će služiti kao polinom za dekodiranje.



Slika 5.8. Prozor za dekodiranje teksta CRC koderom

Kada je unesen tekst za dekodiranje te binarni niz koji će služiti kao polinom za dekodiranje i kliknut gumb dekodiraj pojavljuje nam se novi tekstualni prostor u kojem je ispisan dekodirani tekst (Sl 5.9.) te da li je on ispravno dekodiran, ukoliko nije, aplikacija ispisuje pogrešku i na kojoj poziciji se pogreška nalazi.



Slika 5.9. Rezultat dekodiranja

5.1. Koder i dekoder Hammingovog koda

Kako bi smo započeli sa kodiranjem Hammingovim kodom potrebno je prvo definirati osnovna pravila, u ovom slučaju radi se o Hammingovom 7/4 koderu pa tako uneseni tekst ili niz znakova za kodiranje po pravilu mora biti djeljiv sa 4 (Sl 5.1.).

```
public static string Kodiranje74(string text)
{
    int[] bitArray;

    if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
    {
        if (text.Length % 4 != 0)
        {
            return "Greška: Broj bitova nije djeljiv s 4!";
        }
```

Slika 5.1. Provjera djeljivosti unesenog teksta sa 4

Aplikacija za Hammingov 7/4 koder automatski prepoznaje da li je tekst za kodiranje unesen u znakovnom ili binarnom obliku te u skladu s time određuje daljnji način kodiranja, ukoliko je uneseni tekst u znakovnom obliku pretvara ga u ASCII sustav koji potom kodira kao binarni oblik, a ukoliko je uneseni tekst za kodiranje u binarnom obliku nema potrebe raditi pretvaranje već ga izravno kodira (Sl 5.2.).

```
            bitArray = new int[text.Length];

            for (int i = 0; i < bitArray.Length; i++)
            {
                bitArray[i] = text[i] == '0' ? 0 : 1;
            }
        }
        else
        {
            byte[] bytes = Encoding.ASCII.GetBytes(text);
            bitArray = new int[bytes.Length * 8];

            for (int i = 0; i < bytes.Length; i++)
            {
                int val = bytes[i];

                for (int j = 0; j < 8; j++)
                {
                    bitArray[i * 8 + j] = (val & 128) == 0 ? 0 : 1;
                    val <<= 1;
                }
            }
        }
```

Slika 5.2. Pretvaranje unesenog teksta u ASCII sustav

Kada su osnovne provjere pozitivno odrađene i da pri tome nije nastala greška nastavlja se proces kodiranja (Sl 5.3.) kako je opisan u prethodnim poglavljima.

```
        for (int i = 0; i < bitArray.Length; i += 4)
        {
            int C1 = bitArray[i] ^ bitArray[i + 1] ^ bitArray[i + 3];
            int C2 = bitArray[i] ^ bitArray[i + 2] ^ bitArray[i + 3];
            int C3 = bitArray[i + 1] ^ bitArray[i + 2] ^ bitArray[i + 3];

            result.Append(C1);
            result.Append(C2);
            result.Append(bitArray[i]);
            result.Append(C3);
            result.Append(bitArray[i + 1]);
            result.Append(bitArray[i + 2]);
            result.Append(bitArray[i + 3]);
        }

        return result.ToString();
    }
```

Slika 5.3. Proces kodiranja Hammingovim 7/4 koderom

Identičan proces odvija se i za Hammingov 15/11 koder (Sl 5.4.), no u ovom slučaju radi se provjera djeljivosti tako uneseni tekst ili niz znakova za kodiranje sa 11 te ukoliko nije nastala greška nastavlja se proces kodiranja kako je opisan u prethodnim poglavljima.

```
public static string Kodiranje1511(string text)
{
    int[] bitArray;

    if (String.IsNullOrWhiteSpace(text))
    {
        return "Greška: potrebno je upisati tekst za kodiranje!";
    }

    if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
    {
        if (text.Length % 11 != 0)
        {
            return "Greška: Broj bitova nije djeljiv s 11!";
        }

        bitArray = new int[text.Length];

        for (int i = 0; i < bitArray.Length; i++)
        {
            bitArray[i] = text[i] == '0' ? 0 : 1;
        }
    }
    else
    {
        byte[] bytes = Encoding.ASCII.GetBytes(text);
        bitArray = new int[bytes.Length * 8];

        if (bitArray.Length % 11 != 0)
        {
            return "Greška: Broj bitova u tekstu nije djeljiv s 11!";
        }

        for (int i = 0; i < bytes.Length; i++)
        {
            int val = bytes[i];

            for (int j = 0; j < 8; j++)
            {
                bitArray[i * 8 + j] = (val & 128) == 0 ? 0 : 1;
                val <<= 1;
            }
        }
    }

    StringBuilder result = new StringBuilder();

    for (int i = 0; i < bitArray.Length; i += 11)
    {
        int C1 = bitArray[i] ^ bitArray[i + 1] ^ bitArray[i + 3] ^ bitArray[i + 4] ^ bitArray[i + 6] ^ bitArray[i + 8] ^ bitArray[i + 10];
        int C2 = bitArray[i] ^ bitArray[i + 2] ^ bitArray[i + 3] ^ bitArray[i + 5] ^ bitArray[i + 6] ^ bitArray[i + 9] ^ bitArray[i + 10];
        int C3 = bitArray[i + 1] ^ bitArray[i + 2] ^ bitArray[i + 3] ^ bitArray[i + 7] ^ bitArray[i + 8] ^ bitArray[i + 9] ^ bitArray[i + 10];
        int C4 = bitArray[i + 4] ^ bitArray[i + 5] ^ bitArray[i + 6] ^ bitArray[i + 7] ^ bitArray[i + 8] ^ bitArray[i + 9] ^ bitArray[i + 10];

        result.Append(C1);
        result.Append(C2);
        result.Append(bitArray[i]);
        result.Append(C3);
        result.Append(bitArray[i + 1]);
        result.Append(bitArray[i + 2]);
        result.Append(bitArray[i + 3]);
        result.Append(C4);
        result.Append(bitArray[i + 4]);
        result.Append(bitArray[i + 5]);
        result.Append(bitArray[i + 6]);
        result.Append(bitArray[i + 7]);
        result.Append(bitArray[i + 8]);
        result.Append(bitArray[i + 9]);
        result.Append(bitArray[i + 10]);
    }

    return result.ToString();
}
```

Slika 5.4. Proces kodiranja Hammingovim 15/11 koderom

Proces dekodiranja Hammingovim 7/4 dekoderom započinje provjerom djeljivosti unesene riječi sa 4, ukoliko nije nastala greška proces se nastavlja. U ovom slučaju nema pretvorbe u ASCII sustav pošto će unesena informacija uvijek biti u binarnom zapisu a na korisniku je da svojim odabirom odabere želi li dekodiranu informaciju prikazati kao binarnu ili znakovnu vrijednost. Nakon odabira nastavlja se proces dekodiranja (Sl 5.5.) kako je opisan u prethodnim poglavljima.

```
public static string Dekodiranje74(string text, bool decodeToAscii)
{
    int[] bitArray;

    if (String.IsNullOrWhiteSpace(text))
    {
        return "Greška: potrebno je upisati tekst za dekodiranje!";
    }

    List<string> errors = new List<string>();

    if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
    {
        if (text.Length % 7 != 0)
        {
            return "Uneseni bitovi moraju biti djeljivi sa 7!";
        }

        bitArray = new int[text.Length];


        for (int i = 0; i < bitArray.Length; i++)
        {
            bitArray[i] = text[i] == '0' ? 0 : 1;
        }

        StringBuilder decodedBits = new StringBuilder();
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < bitArray.Length; i += 7)
        {
            int[] recieved = new int[7];

            recieved[0] = bitArray[i];
            recieved[1] = bitArray[i + 1];
            recieved[2] = bitArray[i + 2];
            recieved[3] = bitArray[i + 3];
            recieved[4] = bitArray[i + 4];
            recieved[5] = bitArray[i + 5];
            recieved[6] = bitArray[i + 6];

            int check0 = recieved[2] ^ recieved[4] ^ recieved[6];
            int check1 = recieved[2] ^ recieved[5] ^ recieved[6];
            int check3 = recieved[4] ^ recieved[5] ^ recieved[6];

            int checksum = 0;

            if (recieved[0] != check0) checksum += 1;
            if (recieved[1] != check1) checksum += 2;
            if (recieved[3] != check3) checksum += 4;

            if (checksum != 0)
            {
                result.Append($"Detektirana je pogreška na {checksum}. bitu informacije.<br />");

                if (recieved[checksum - 1] == 0)
                {
                    recieved[checksum - 1] = 1;
                }
                else
                {
                    recieved[checksum - 1] = 0;
                }
            }

            decodedBits.Append(recieved[2] == 0 ? "0" : "1");
            decodedBits.Append(recieved[4] == 0 ? "0" : "1");
            decodedBits.Append(recieved[5] == 0 ? "0" : "1");
            decodedBits.Append(recieved[6] == 0 ? "0" : "1");
        }

        if (result.Length > 0)
        {
            result.Append("Ispravljena informacija je: ");
        }

        if (!decodeToAscii)
        {
            return decodedBits.ToString();
        }
        else
        {
            for (int i = 0; i < decodedBits.Length / 8; i++)
            {
                result.Append(
                    Convert.ToChar(
                        Convert.ToByte(
                            decodedBits.ToString().Substring(i * 8, 8), 2
                            )
                            )
                            );
            }
        }

        return result.ToString();
    }
    else
    {
        return "Uneseni tekst treba sadržavati samo bitove (0 ili 1)!";
    }
}
```

Slika 5.5. Proces dekodiranja Hammingovim 7/4 dekoderom

Proces dekodiranja Hammingovim 15/11 dekoder identičan je procesu dekodiranja Hammingovim 7/4 dekoderom uz iznimku da se u ovom slučaju radi provjera djeljivosti unesenog teksta ili niz znakova za kodiranje sa 11 te ukoliko nije nastala greška nastavlja se proces dekodiranja (Sl 5.6.) kako je opisan u prethodnim poglavljima.

```
        public static string Dekodiranje1511(string text, bool decodeToAscii)
        {
            int[] bitArray;

            if (String.IsNullOrWhiteSpace(text))
            {
                return "Greška: potrebno je upisati tekst za dekodiranje!";
            }

            List<string> errors = new List<string>();

            if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
            {
                if (text.Length % 15 != 0)
                {
                    return "Uneseni bitovi moraju biti djeljivi sa 15!";
                }

                bitArray = new int[text.Length];

                for (int i = 0; i < bitArray.Length; i++)
                {
                    bitArray[i] = text[i] == '0' ? 0 : 1;
                }

                StringBuilder decodedBits = new StringBuilder();
                StringBuilder result = new StringBuilder();

                for (int i = 0; i < bitArray.Length; i += 15)
                {
                    int[] recieved = new int[15];

                    recieved[0] = bitArray[i];
                    recieved[1] = bitArray[i + 1];
                    recieved[2] = bitArray[i + 2];
                    recieved[3] = bitArray[i + 3];
                    recieved[4] = bitArray[i + 4];
                    recieved[5] = bitArray[i + 5];
                    recieved[6] = bitArray[i + 6];
                    recieved[7] = bitArray[i + 7];
                    recieved[8] = bitArray[i + 8];
                    recieved[9] = bitArray[i + 9];
                    recieved[11] = bitArray[i + 11];
                    recieved[10] = bitArray[i + 10];
                    recieved[12] = bitArray[i + 12];
                    recieved[13] = bitArray[i + 13];
                    recieved[14] = bitArray[i + 14];

                    int check0 = recieved[2] ^ recieved[4] ^ recieved[6] ^ recieved[8] ^ recieved[10] ^ recieved[12] ^ recieved[14];
                    int check1 = recieved[2] ^ recieved[5] ^ recieved[6] ^ recieved[9] ^ recieved[10] ^ recieved[13] ^ recieved[14];
                    int check3 = recieved[4] ^ recieved[5] ^ recieved[6] ^ recieved[11] ^ recieved[12] ^ recieved[13] ^ recieved[14];
                    int check7 = recieved[8] ^ recieved[9] ^ recieved[10] ^ recieved[11] ^ recieved[12] ^ recieved[13] ^ recieved[14];

                    int checksum = 0;

                    if (recieved[0] != check0) checksum += 1;
                    if (recieved[1] != check1) checksum += 2;
                    if (recieved[3] != check3) checksum += 4;
                    if (recieved[7] != check7) checksum += 8;

                    if (checksum != 0)
                    {
                        result.Append($"Detektirana je pogreška na {checksum}. bitu informacije.<br />");

                        if (recieved[checksum - 1] == 0)
                        {
                            recieved[checksum - 1] = 1;
                        }
                        else
                        {
                            recieved[checksum - 1] = 0;
                        }
                    }

                    decodedBits.Append(recieved[2] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[4] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[5] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[6] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[8] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[9] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[10] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[11] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[12] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[13] == 0 ? "0" : "1");
                    decodedBits.Append(recieved[14] == 0 ? "0" : "1");

                }

                //ako je rezultatni string > 0 to znač da smo upisivali greške
                if (result.Length > 0)
                {
                    result.Append("Ispravljena informacija je: ");
                }

                if (!decodeToAscii)
                {
                    return decodedBits.ToString();
                }
                else
                {
                    for (int i = 0; i < decodedBits.Length / 8; i++)
                    {
                        result.Append(
                            Convert.ToChar(
                                Convert.ToByte(
                                    decodedBits.ToString().Substring(i * 8, 8), 2
                                    )
                                    )
                                    );
                    }

                    return result.ToString();
                }
            }
            else
            {
                return "Uneseni tekst treba sadržavati samo bitove (0 ili 1)!";
            }
        }
    }
}
```

Slika 5.6. Proces dekodiranja Hammingovim 15/11 dekoderom

5.2. Koder i dekoder Cikličke redundantne provjere (engl. *cycle redudancy check*)

CRC koder jednostavniji je od Hammingovog kodera u tome što CRC koder radi samo sa binarnim oblikom informacije pa prema tome zahtjeva samo jednu provjeru, a to je da li su uneseni tekst za kodiranje i polinom za kodiranje u binarnom obliku (Sl 5.7.).

```
if (polynomial.Count(c => c == '0') + polynomial.Count(c => c == '1') == polynomial.Length)
{
    polyArray = new int[polynomial.Length];

    for (int i = 0; i < polyArray.Length; i++)
    {
        polyArray[i] = polynomial[i] == '0' ? 0 : 1;
    }
}
```

Slika 5.7. Provjera binarnog oblika unesenog teksta za kodiranje

Ako nije nastala greška nastavlja se proces kodiranja (Sl. 5.6.) koji je opisan u prethodnim poglavljima.

```
public static string Kodiranje(string text, string polynomial)
{
    int[] bitArray;
    int[] polyArray;

    if (String.IsNullOrWhiteSpace(text))
    {
        return "Greška: potrebno je upisati tekst za kodiranje!";
    }

    if (String.IsNullOrWhiteSpace(polynomial))
    {
        return "Greška: potrebno je upisati polinom za kodiranje!";
    }

    if (polynomial.Count(c => c == '0') + polynomial.Count(c => c == '1') == polynomial.Length)
    {
        polyArray = new int[polynomial.Length];

        for (int i = 0; i < polyArray.Length; i++)
        {
            polyArray[i] = polynomial[i] == '0' ? 0 : 1;
        }
    }
    else
    {
        return "Greška: polinom mora biti u binarnom obliku!";
    }

    if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
    {
        bitArray = new int[text.Length];

        for (int i = 0; i < bitArray.Length; i++)
        {
            bitArray[i] = text[i] == '0' ? 0 : 1;
        }
    }
    else
    {
        return "Greška: informacija mora biti u binarnom obliku!";
    }

    int totalLength = bitArray.Length + polyArray.Length - 1;

    int[] divison = new int[totalLength];
    int[] remainder = new int[totalLength];
    //int[] crc = new int[totalLength];

    for (int i = 0; i<bitArray.Length; i++)
    {
        divison[i] = bitArray[i];
    }

    for (int i = 0; i < divison.Length; i++)
    {
        remainder[i] = divison[i];
    }

    remainder = Division(divison, polyArray, remainder);

    StringBuilder result = new StringBuilder();

    for (int i = 0; i < divison.Length; i++)
    {
        result.Append(Convert.ToString(divison[i] ^ remainder[i]));
    }

    return result.ToString();
}
```

Slika 5.8. Proces kodiranja CRC koderom

Dekodiranje CRC dekoderom kako je opisano u prethodnim poglavljima funkcionira na obrnuti način od CRC kodera uz jednu identičnu stvar a to je provjera da li su uneseni tekst za dekodiranje i polinom za dekodiranje u binarnom obliku. ukoliko nije nastala greška nastavlja se proces kodiranja (Sl 5.9.) kako je opisan u prethodnim poglavljima.

```
public static string Dekodiranje(string text, string polynomial)
{
    int[] bitArray;
    int[] polyArray;

    if (String.IsNullOrWhiteSpace(text))
    {
        return "Greška: potrebno je upisati tekst za dekodiranje!";
    }

    if (String.IsNullOrWhiteSpace(polynomial))
    {
        return "Greška: potrebno je upisati polinom za dekodiranje!";
    }

    if (polynomial.Count(c => c == '0') + polynomial.Count(c => c == '1') == polynomial.Length)
    {
        polyArray = new int[polynomial.Length];

        for (int i = 0; i < polyArray.Length; i++)
        {
            polyArray[i] = polynomial[i] == '0' ? 0 : 1;
        }
    }
    else
    {
        return "Greška: polinom mora biti u binarnom obliku!";
    }

    if (text.Count(c => c == '0') + text.Count(c => c == '1') == text.Length)
    {
        bitArray = new int[text.Length];

        for (int i = 0; i < bitArray.Length; i++)
        {
            bitArray[i] = text[i] == '0' ? 0 : 1;
        }
    }
    else
    {
        return "Greška: informacija mora biti u binarnom obliku!";
    }

    int totalLength = bitArray.Length;

    int[] divison = new int[totalLength];
    int[] remainder = new int[totalLength];
    //int[] crc = new int[totalLength];

    for (int i = 0; i < bitArray.Length; i++)
    {
        divison[i] = bitArray[i];
    }

    for (int i = 0; i < divison.Length; i++)
    {
        remainder[i] = divison[i];
    }

    remainder = Division(divison, polyArray, remainder);

    StringBuilder result = new StringBuilder();

    bool ok = true;

    for (int i = 0; i < remainder.Length; i++)
    {
        if (remainder[i] != 0)
        {
            ok = false;
            break;
        }
    }

    if (ok)
    {
        result.Append("Kodirana informacija je ispravna. Rezultat dekodiranja je: ");

        for (int i = 0; i < divison.Length - polyArray.Length + 1; i++)
        {
            result.Append(Convert.ToString(divison[i] ^ remainder[i]));
        }

        return result.ToString();
    }
    else
    {
        return "Greška: neispravno je primljena kodirana informacija!";
    }
}
```

Slika 5.9. Proces dekodiranja CRC koderom

6. ZAKLJUČAK

Hammingov koder i dekoder vrlo su jednostavni primjeri zaštitnog kodiranja, čija primjena u današnjim sustavima je ograničena zbog količine informacija koja se prenosi te potrebne brzine prijenosa koje ubrzano rastu iz dana u dan. Današnje informacijsko doba zahtjeva veći stupanj ispravljanja i detekcije pogreške od onoga koje Hammingov koder i dekoder pružaju. Ciklička redundantna provjera (engl. *cycle redudancy check*) koristi se u današnjim računalnim mrežama i kao takva je najkorištenija metoda za detekciju pogrešaka. Računalne mreže koriste cikličku redundantnu provjeru (engl. *cycle redudancy check*) zbog jednostavnosti detekcije pogreške metodom pomaka, kao i jednostavniji zapis u obliku algoritma naspram matrica kao kod ostalih metoda.

Ova mrežna aplikacija može se koristiti kao svojevrsni kalkulator za provjeru ispravnosti poslanih informacija što potencijalno olakšava rad u školstvu odnosno predmetima blisko vezanim za područje informacijskih znanosti. Ima mogućnost nadogradnje sa ostalim koderima i dekoderima pa tako može postati cijeli sustav za kodiranje i dekodiranje informacija.

Aplikacija je mogla biti izrađena u više oblika te u više programskih jezika no izrađena je u .NET tehnologiji te zamišljena kao mrežna aplikacija. .NET u svojoj osnovi pruža infrastrukturu za izradu mrežna aplikacija pa tako nudi gotova rješenja i funkcionalnosti koje ubrzavaju proces izradu mrežnih aplikacija. Prednosti mrežne aplikacije su što je dostupna svim uređajima koji imaju internet preglednik, vrlo lako se može promijeniti ili nadograditi sa novom verzijom pošto na internetu svi vide istu verziju za razliku od programa koji mora biti instaliran na računalu.

LITERATURA:

[1] Igor S. Pandžić, Željko Ilić, Mlade Kos, Alen Bažant, Zdenko Vrdoljak, Vjekoslav Sinković - Uvod u teoriju informacije i kodiranje, Zagreb, 2007.

[2] Zdenko Vrdoljak – Zaštitno kodiranje, Zagreb, 2006.

[3] C.E.Shannon, W.Weaver - The Mathematical Theory of Communication, Urbana and Chicago, 1998.

[4] https://www.microsoft.com/net/

[5] https://www.visualstudio.com/

Popis upotrijebljenih oznaka:

CRC - Ciklička redundantna provjera

ASCII - American Standard Code for Information Interchange

HTML - Hyper Text Markup Language

CSS - Cascading Style Sheets

SAŽETAK

Rad obrađuje temu zaštitnog kodiranja. Objasnio je potencijalne probleme slanja informacije, koji se pojavljuju svakodnevno u informacijskim zanostima, kroz jedan od koraka u cjelokupnom procesu slanja informacije, a to je zaštitno kodiranje. Zaštitno kodiranje odnosno Linearni blok kodovi, detaljnio su opisani ...Hammingovi kodovi te ciklički kodovi objašnjeni su teoretski uz osnovne teoreme, teoretski prikaz njihovih kodera, dekodera, pomoću jednostavnih primjera te praktično kroz izradu kodera i dekodera unutar programskog jezika C# gdje je prikazan i iskorišten pun potencijal slanja informacija digitalnim putem.

Pobliže su pojašnjeni principi rada Hammingovog kodera i dekodera te kodera i dekodera specifičnog cikličkog koda naziva Ciklička redundantna provjera (engl. *Cyclic redundancy check*) kao i izrada njihovih kodera i dekodera u tehnologiji .NET.

Ključne riječi:
informacija, zaštitno kodiranje, koder, dekoder, linearni kodovi, Hammingovi kodovi, ciklički kodovi

ABSTRACT

The paper deals with the theme of encryption. This final paper explained the potential problems of sending information, which occur daily in the information science, through one of the steps in the overall process of sending information, which is encryption. Encryption or rather Linear Codes, Hamming Codes and Cyclic Codes are explained theoretically along with theoretical theorems, theoretical presentation of their encoders, decoders, by simple examples and practically through the creation of encoders and decoders within the programming language C # where the full potential of sending information digitally displayed and utilized .

The more detailed principles of Hamming's encoder and decoder, the encoder and decoder of a specific cyclic code called Cyclic redundancy check, and the creation of their encoders and decoders in the C # programming language.

Ključne riječi na engleskom jeziku:

information, encryption, coder, decoder, linear codes, Hammings codes, cyclic codes

ŽIVOTOPIS:

Vanja Dragišić rođen je 05.08.1991. u Osijeku, gdje je završio osnovnu i Elektrotehničku i prometnu srednju školu . Student je stručnog studija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, smjer Informatika. Od stranih jezika zna engleski gdje je dobar u čitanju, pisanju i govoru.