Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Petra Reberšek

Hadamardove matrike in njihova uporaba v kodah za popravljanje napak

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Bojan Orel

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00553 / 2013
Datum: 15.9.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PETRA REBERŠEK**

Naslov: **HADAMARDOVE MATRIKE IN NJIHOVA UPORABA V KODAH ZA POPRAVLJANJE NAPAK**
HADAMARD MATRICES AND ITS APPLICATIONS IN ERROR CORRECTING CODES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Preglejte načine za popravljanje napak pri prenosu podatkov. Pri tem posebno pozornost posvetite uporabi Hadamardovih matrik. Na primeru ilustrirajte uporabnost izbrane metode.

Mentor:

izr. prof. dr. Bojan Orel

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisana Petra Reberšek, z vpisno številko **63970275,** sem avtorica diplomskega dela z naslovom:

HADAMARDOVE MATRIKE IN NJIHOVA UPORABA V KODAH ZA POPRAVLJANJE NAPAK

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom prof. dr. Bojana Orla,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14. marec 2014

Podpis avtorja:

Reberšek

Najlepše se zahvaljujem vsem, ki so kakorkoli pripomogli k nastanku diplomskega dela. Še posebna zahvala gre mentorju prof. dr. Bojanu Orlu za potrpežljivost, usmerjanje in strokovno vodenje.

Zahvaljujem se tudi vsem prijateljem, sorodnikom in staršem, ki so me finančno in moralno podpirali na moji študijski poti.

Hvala tudi Gašperju za vse vzpodbudne besede, ki so mi pomagale, ko sem jih najbolj potrebovala.

Kazalo

Povzetek

Ključne besede . . . . . .

Abstract

Key words . . . . . .

1 Uvod **1**

1.1 Opredelitev problema 1

1.2 Zgradba diplomskega dela 2

2 Jacques Salamon Hadamard (biografija) **3**

3 Algeberske strukture **7**

3.1 Grupe 7

3.2 Podgrupe 10

3.3 Ciklične grupe 13

3.4 Komutativni obseg 14

4 Hadamardove matrike **17**

4.1 Hadamardove matrike 17

4.2 Kompleksne Hadamardove matrike 19

4.3 Hadamardova matrična domneva 20

4.4 Konstrukcije Hadamardovih matrik 20

4.4.1 Sylvestrova konstrukcija 20

4.4.2 Paleyjeve matrike 23

4.4.3 Druge konstrukcije 30

5 Osnove kodiranja **33**

5.1 Uvod 33

5.2 Kodiranje 34

6 Uporaba Hadamardovih matrik **37**

6.1 Hadamardove kode 37

6.2 Kodiranje in dekodiranje 39

6.3 Simulacija kodiranja in dekodiranja 42

7 Zaključek **47**

8 Dodatek **53**

Povzetek

Diplomsko delo opisuje konstrukcije Hadamardovih matrik in uporabo le teh v praksi.

V začetku diplomskega dela so opisane nekatere algeberske strukture (grupe, ciklične grupe in komutativni obsegi).

V nadaljevanju je opisana definicija Hadamardovih matrik in nekatere konstrukcije le teh (Sylvestrova in Paleyeva).

V zaključku diplomskega dela pa so podane osnove kodiranja in uporaba Hadamardovih matrik pri kodiranju in dekodiranju.

Ključne besede:

Hadamardove matrike, Kroneckerjev produkt, Paleyeve matrike, Hadamardove kode.

Abstract

In this work we describe construction of Hadamard matrices and use of this matrices in practice.

We start with basic algebraic structres (groups, cyclic groups, commutative fields), we continue with definition of Hadamard matrices and examples of construction of Hadamard matrices (Sylvester and Paley). We finish with basic description of error correcting codes and use of Hadamard matrices in encoding and decoding.

Key words:

Hadamard matrices, Kronecker product, Payley matrices, Hadamard codes.

Poglavje 1

Uvod

V času informacijske tehnologije je zelo pomembno, da pri prenosu informacij ne prihaja do napak. Zato je dobro poznati metode, ki popravijo napake, ki se pojavijo zaradi zunanjih vplivov (nevihte, električna napetost,...). Poznati je potrebno tudi, na kakšen način lahko določeno informacijo zakodiramo, da je nepoklicane osebe ne znajo razpoznati, in to informacijo dekodiramo, da jo bomo zopet lahko prepoznali.

1.1 Opredelitev problema

V diplomskem delu bom obravnavala Hadamardove matrike in Hadamardove kode. Hadamardove matrike so matrike s posebnimi lastnostmi, ki jih je Jacques Salamon Hadamard ugotovil že v koncu 19. stoletja. Te matrike so kasneje uporabili za kodiranje in dekodiranje informacij.
NASA je uporabila Hadamardove matrike pri prenosu podatkov iz vesoljske sonde, ki so jo poslali na Mars. Pri prenosu podatkov je bilo pomembno, da zaradi zunanjih vplivov ni prihajalo do napak. Signal je ob prihodu na Zemljo bil šibek in ga je bilo potrebno ojačati. Motnje iz vesolja in termične motnje pa so lahko povzročile napačno interpretacijo podatkov.
V tem primeru so pomemno vlogo imele Hadamardove matrike. Vsako infor-

macijo, ki so jo poslali, so najprej zakodirali s pomočjo Hadamardove matrike in jo nato na Zemlji dekodirali. Ta način pošiljanja informacij je zelo dobro popravljal tudi popačene informacije.

1.2 Zgradba diplomskega dela

Jacques Salamon Hadamard je zelo pomemben matematik, katerega dela so uporabili pri razvoju kodiranja in odpravljanja napak med prenosi. Zato sem se odločila, da drugo poglavje namenim njegovi biografiji.
V tretjem poglavju bom predstavila vse algeberske strukture, ki so potrebne v nadaljevanju in jih nismo uporabljali pri mojem študiju.
Četrto poglavje bom v celoti namenila Hadamardovim matrikam in konstrukcijam le-teh.
V petem in šestem poglavju pa bom predstavila teorijo kodiranja in uporabo Hadamardovih matrik pri kodiranju.

Poglavje 2

Jacques Salamon Hadamard (biografija)

Jacques Salamon Hadamard se je rodil 8. 10. 1865 v Versaillesu v Franciji, preminil pa je 17. 10. 1963 v Parizu.

Amédée Hadamard, Jacques Hadamardov oče, se je šestega junija 1864 poročil s Claire Marie Jeanne Picard. Amédée Hadamard je bil judovskega porekla in učitelj, poučeval je več predmetov, kot so: klasika, slovnica, zgodovina in geografija, medtem ko pa je Jacquesova mama poučevala klavir na domu. V času Jacquesovega rojstva je njegov oče poučeval na Lycée Impérial v Versaillesu. Družina se je preselila v Paris, ko je imel Jacques tri leta, takrat je njegov oče nastopil delo na *Lycée Charlemagne*.

Kmalu za tem, 19. julija 1870, se je pričela Francosko-Pruska vojna, ki je zelo negativno vplivala na Francijo. Pariz so napadli 19. septembra 1870, 28. januarja 1871 pa so se Parižani predali. Nato so 10. 5. 1871 ratificirali frankfurtsko pogodbo, ta ratifikacija je bila veliko ponižanje za Francijo. Med predajo in podpisom pogodbe je izbruhnila državljanska vojna v Parizu in hiša Hadamardov je pogorela do tal. To so bili zelo nesrečni časi za odraščanje otrok v Parizu. Med tem časom sta umrli tudi mlajši Jacquesovi sestri, Jeanne in Suzanne.

Jacques se je pričel šolati na *Lycée Charlemagne*, kjer je poučeval nje-

gov oče. V prvih letih šolanja je bil uspešen pri vseh predmetih, samo pri matematiki ne. Izstopal je predvsem pri grščini in latinščini.

V začetku je bil precej šibak v aritmetiki, v petem razredu pa je že zasedel drugo mesto v razredu. V tem času (1875) je prejemal različna priznanja pri številnih predmetih na nacionalnih tekmovanjih za učence. Takrat ga je poučeval zelo sposoben profesor, kateremu je uspelo, da ga je navdušil nad matematiko in naravoslovjem.

Jacques se je od leta 1876 izobraževal na *Lycée Louis-le-Grand*. Leta 1882 je diplomiral *Bachelier és lettres et és sciences*, v naslednjem letu je prejel naziv *Baccalauréat és sciences*. Leta 1883 je osvojil prvo mesto v algebri in mehaniki na *Concours Général*. Nato se je vpisal na *École Normale Superérieure*, kjer je diplomiral 30. 9. 1888.

Med raziskovanjem za doktorat je bil zaposlen kot učitelj. Najprej je bil zaposlen na *Lycée Caen*, kjer ni poučeval. Od junija 1889 je poučeval na *Lycée Saint-Louis*, kjer ni bil preveč uspešen, saj je bil za ta nivo prezahteven. Od septembra 1890 je bil zaposlen na *Lycée Buffon*, kjer je poučeval tri leta. Kljub poučevanju je njegova raziskovalna naloga potekala odlično. Leta 1892 je doktoriral, v doktoratu je raziskoval Taylorjevo vrsto. Njegovo delo o funkcijah kompleksne spremenljivke je bilo eno prvih na svetu, kjer je raziskoval splošno teorijo analitičnih funkcij.

Istega leta je Hadamard prejel *Grand Prix des Sciences Mathématiques* za delo z naslovom *Détermination du nombre de nombres premiers inférieure a un nombre donné* (Določitev števila praštevil manjših od danega števila). Leto 1892 je bilo za Hadamarda zelo pomembno. Poleg tega, da je doktoriral in prejel nagrado, se je junija tega leta poročil z Louiso-Anno Trénel. Naslednjega leta pa je bil imenovan za predavatelja na univerzi v Bordeuxu, kamor sta se z ženo preselila. Kot profesor na srednji šoli ni bil uspešen, tu pa je požel veliko zadovoljstva, hvale in uspehov. Svoje študente je zelo navdušil s svojim načinom raziskovanja in predavanja.

V štirih letih, ki jih je preživel v Bordeuxu, ni bil uspešen samo v privatnem življenju, rodila sta se mu dva sinova, ampak tudi profesionalno.

Objavil je 29 člankov. Članki so bili zelo poglobljena raziskovalna dela iz različnih vej matematike. Eno najpomembnejših del tistega časa je bil dokaz izreka o gostoti praštevil, ki ga je dokazal leta 1896. Domneva za ta izrek je bila predstavljena že v 18. stoletju. Zanimivo je, da sta neodvisno v tem času to domnevo dokazala Hadamard in Charles de la Valée Poussin z uporabo kompleksne analize.

Naslednji pomembni rezultat nejegovih raziskovanj je bila objava članka o njegovi matrični domnevi. Matrike, ki zadoščajo pogojem iz domneve, sedaj imenujemo Hadamardove matrike in so zelo pomembne v teoriji kodiranja, v teoriji integralskih enačbah in drugih področjih.

V letu 1897 se je z družino preselil v Paris, kjer je sprejel malo manj zahtevno delo na fakulteti *Collége de France*. Kmalu po prihodu je objavil *Leçons de Géometrie Elémentare*. V tem članku se je posvetil dvodimenzionalni geometriji, leta 1901 je izdal drugi del, kjer je dal poudarek na trodimenzionalni geometriji. To delo je imelo zelo velik vpliv na poučevanje matematike v Franciji.

Leta 1898 je prejel nagrado *Prix Poncelet* za raziskovalne uspehe v zadnjih desetih letih. Njegova raziskovanja so bila v večini na področju matematične fizike. Predvsem je delal na področju parcialnih diferencialnih enačb v matematični fiziki. Nato je bil leta 1912 imenovan za profesorja analize na *École polytechnique*. Konec tega leta je bil tudi imenovan v *l'Académie des sciences*.

Med drugo svetovno vojno sta oba njegova sinova umrla v boju. Nato se je še bolj posvetil raziskovanju, saj je le tam našel uteho. Zato je veliko potoval, gostoval je kot predavatelj v ZDA, Španiji, Češkoslovaški, Italiji, Švici, Braziliji, Argentini in Egiptu.

Do svoje smrti leta 1962 je izdal še veliko člankov in knjig. Lotil se je tudi popolnoma novega področja teorije verjetnosti, iz katerega je izdal precej člankov.

Poglavje 3

Algeberske strukture

V tem poglavju bomo opisali algeberske strukture, ki jih bomo uporabljali v nadaljevanju.

3.1 Grupe

Definicija 3.1 Grupa *je množica* $\mathbb{G}$*, v kateri je definirana operacija* $*$ *tako, da za vse* $a, b, c \in \mathbb{G}$*, veljajo naslednje lastnosti:*

1. *operacija* $*$ *je notranja:* $a * b \in \mathbb{G}$
2. *asociativnost:* $(a * b) * c = a * (b * c)$
3. *v množici* $\mathbb{G}$ *obstaja enota* (*označimo jo z* e)*, tako da velja:* $a * e = e * a = a$
4. *za vsak element* $a \in \mathbb{G}$ *obstaja* $a^{-1} \in \mathbb{G}$*, tako da velja:* $a * a^{-1} = a^{-1} * a = e$

V nadaljenavnju bomo pokazali nekaj trditev, ki veljajo v grupah. Te trditve bomo potrebovali v poglavju podgrup, saj bomo z njihovo pomočjo lažje in hitreje ugotovili ali je neka množica elementov iz grupe tudi podgrupa.

V naslednji trditvi bomo pokazali, da lahko v vsaki grupi uporabimo pravilo krajšanja. S pomočjo tega pravila bomo v nadaljevanju pokazali, da sta v vsaki grupi enačbi $a * x = b$ in $y * a = b$ enolično rešljivi.

Trditev 3.1 *V grupi* $(G, *)$ *veljata pravili krajšanja:*

$$a * x = b * x \Rightarrow a = b$$

in

$$x * a = x * b \Rightarrow a = b.$$

Dokaz:
Dokazali bomo samo prvo pravilo, dokaz drugega je podoben:

$$\begin{aligned} a * x &= b * x \\ a * x * x^{-1} &= b * x * x^{-1} \\ a * e &= b * e \\ a &= b. \end{aligned}$$

□

Trditev 3.2 *V grupi* $(G, *)$ *sta enačbi* $a * x = b$ *in* $y * a = b$ *enolično rešljivi za vsak par* $a, b \in G$.

Dokaz:
Rešitev prve enačbe je $x = a^{-1} * b$. Preverimo,

$$a * x = a * (a^{-1} * b) = (a * a^{-1}) * b = e * b = b.$$

Pokažimo, da je to edina rešitev. Če imamo dve rešitvi x_1 in x_2, potem velja:

$$a * x_1 = b \quad \text{in} \quad a * x_2 = b.$$

Iz tega sledi

$$a * x_1 = a * x_2$$

in po pravilu krajšanja dobimo

$$x_1 = x_2.$$

Dokazali smo samo prvo enačbo, dokaz druge je podoben. □

V naslednjem poglavju bomo predstavili *Lagrangov izrek*, ki govori o razmerju med močjo končne grupe in njene podgupe. Za dokaz tega izreka potrebujemo spodaj definirano funkcijo, ki je bijektivna zaradi trditev 3.1 in 3.2.

Trditev 3.3 *Naj bo a element iz grupe $(G, *)$. Če funkcijo f_a definiramo takole:*

$$\text{za vsak } x \in G : f_a(x) = a * x.$$

Potem je f_a bijektivna funkcija.

Dokaz:
Iz trditve 3.1 sledi injektivnost in iz trditve 3.2 sledi surjektivnost. □

Naj bo a element iz grupe $(G, *)$ ter H (končna) podmnožica v G. Definirajmo naslednje množice:

$$aH = \{a * h : \text{ za vsak } h \in H\},$$
$$Ha = \{h * a : \text{ za vsak } h \in H\},$$
$$a^{-1}Ha = \{a^{-1} * h * a : \text{ za vsak } h \in H\}.$$

Trditev 3.4 *Množice definirane zgoraj imajo enako moč, se pravi:*

$$|aH| = |H| = |Ha| = |a^{-1}Ha|.$$

Dokaz:
Trditev sledi iz trditve 3.3. □

Trditev 3.5 *Naj bo $(G, *)$ grupa in $x, y \in G$, potem velja:*

$$(x * y)^{-1} = y^{-1} * x^{-1}$$

Dokaz:
Trditev sledi iz naslednjih dveh izpeljav:

$$(x * y) * (y^{-1} * x^{-1}) = (x * (y * y^{-1})) * x^{-1} = (x * e) * x^{-1} = x * x^{-1} = e$$

in

$$(y^{-1} * x^{-1}) * (x * y) = (y^{-1} * (x * x^{-1})) * y = (y^{-1} * e) * y = x^{-1} * x = e.$$

□

3.2 Podgrupe

Definicija 3.2 *$H \subseteq G$ je* podgupa *grupe $(G, *)$, če velja:*

1. $x, y \in H \Rightarrow x * y \in H$
2. $e \in H$
3. $x \in H \Rightarrow x^{-1} \in H$

Če je H podgrupa grupe G, pišemo $(H, *) \leq (G, *)$ oz. $H \leq G$.
V nasledjem izreku bomo pokazali, na kakšen način lahko vidimo, da je neka podmnožica grupe tudi podgrupa.

Izrek 3.1 *Naj bo H podmnožica v grupi G. H je podgrupa natanko takrat, ko je*

$$H \neq \emptyset \text{ in za vsak } x, y \in H \text{ velja: } x^{-1} * y \in H.$$

Dokaz:
($\Rightarrow$): Ker je $H \leq G$, je $e \in H$. Iz tega sledi: $H \neq \emptyset$. Naj bosta $x, y \in H$, potem sta tudi $x^{-1}, y \in H$, torej velja: $x^{-1} * y \in H$.
($\Leftarrow$): Dokazati moramo vse tri pogoje iz definicije o podgrupah.
Ker je $H \neq \emptyset$, obstaja element $x \in H$.
Naj bo $y = x$, potem velja:

$$x^{-1} * y \in H \Rightarrow x^{-1} * x \in H \Rightarrow e \in H.$$

Torej je izpolnjen pogoj 2.
Naj bo $y = x$ in $x = e$, potem velja:

$$x^{-1} * y \in H \Rightarrow e * x^{-1} \in H \Rightarrow x^{-1} \in H.$$

Torej je izpolnjen pogoj 3.
Naj bosta $x, y \in H \Rightarrow x^{-1}, y \in H \Rightarrow (x^{-1})^{-1} * y \in H \Rightarrow x * y \in H$.
Torej je izpolnjen tudi pogoj 1. □

Trditev 3.6 *Naj bo $H \leq G$ in $a \in G$. Potem je $a^{-1}Ha \leq G$.*

Dokaz:
Uporabili bomo izrek 3.1. Torej moramo dokazati, da je $a^{-1}Ha \neq \emptyset$ in za vsak $x, y \in a^{-1}Ha$ velja: $x^{-1} * y \in a^{-1}Ha$.
Ker $H \neq \emptyset$, je tudi $a^{-1}Ha \neq \emptyset$. Č sta $x, y \in a^{-1}Ha$, potem obstajata $h_1, h_2 \in H$, za katera velja:

$$x = a^{-1} * h_1 * a \text{ in } y = a^{-1} * h_2 * a.$$

Iz tega sledi:

$$x^{-1}*y = (a^{-1}*h_1*a)^{-1}*(a^{-1}*h_2*a) = a^{-1}*h_1^{-1}*a*a^{-1}*h_2*a = a^{-1}*(h_1^{-1}*h_2)*a.$$

Ker je H podgrupa v G, velja, da je $h_1^{-1} * h_2 \in H$. Iz tega sledi, da je $a^{-1} * (h_1^{-1} * h_2) * a \in a^{-1}Ha$. Torej je tudi $x^{-1} * y \in a^{-1}Ha$. □

S pomočjo naslednje definicije in trditve bomo pokazali, da je moč vsake končne grupe deljiva z močjo njene podgupe. To lastnost bomo potrebovali v poglavju cikličnih grup.

Definicija 3.3 *Naj bo $H \leq G$ in $x \in G$.* Levi odsek *elementa x po podgrupi H je*

$$xH = \{x * h : h \in H\}.$$

Trditev 3.7 *Naj bo $H \leq G$ in R relacija v G, definirana s predpisom:*

$$xRy \Leftrightarrow xH = yH.$$

Potem velja:

1. *R je ekvivalenčna relacija,*

2. *$xRy \Leftrightarrow x^{-1} * y \in H$.*

Dokaz:
Prva trditev:
refleksivnost: za vsak $a \in G$ velja: aRa (očitno)
simetričnost: za vsak $a, b \in G$ velja: $(aRb \Rightarrow bRa)$ (očitno)
tranzitivnost: za vsak $a, b, c \in G$ velja: $(aRb$ in $bRc) \Rightarrow aRc$ (očitno)

Druga trditev:
$(\Rightarrow) : xRy \Rightarrow xH = yH \Rightarrow x \in yH \Rightarrow \exists \quad h \in H : x = y * h \Rightarrow x^{-1} * y = h^{-1} \in H$
$(\Leftarrow) : x^{-1} * y \in H \Rightarrow \exists \quad h \in H : x^{-1} * y = h \Rightarrow y = x * h$ in $x = y * h^{-1} \Rightarrow yH \subseteq xH$ in $xH \subseteq yH$. □

Če je število levih odsekov po podgrupi H končno, ga imenujemo indeks podgrupe H v grupi G in ga označimo z $[G : H]$.

Izrek 3.2 (Lagrangev izrek) *Naj bo G končna grupa in $H \leq G$. Potem je moč grupe G deljiva z močjo podgrupe H in velja formula*

$$|G| = [G : H]|H|.$$

Dokaz:
Naj bo $a \in G$ poljuben element. Funcija f_a iz trditve (3.3) je bijektivna $\Rightarrow$ $|aH| = |H|$.
Vsi levi odseki po H imajo enako moč kot podgrupa H. Torej, vsi odseki po H imajo enako moč. Ker levi odseki sestavljajo razbitje grupe G, dobimo:

$$\begin{aligned} |G| = &\text{ vsota moči levih odsekov po } H = \\ &= (\text{število levih odsekov}) * (\text{moč odseka}) = \\ &= [G : H]|H|. \end{aligned}$$

□

3.3 Ciklične grupe

V nadaljevanju bomo definirali ciklične grupe, ki jih potrebujemo pri Paleyevi konstrukciji Hadamardovih matrik.

Definicija 3.4 *Naj bo G končna grupa in $a \in G$. Red elementa a je najmanjše naravno število n tako, da velja $a^n = e$.*

Trditev 3.8 *Naj bo G končna grupa moči n in $a \in G$. Potem velja:*

1. $a^{\mathrm{red}(a)} = e$
2. $\mathrm{red}(a) = 1 \Leftrightarrow a = e$
3. $\mathrm{red}(a) | n$
4. $a^n = e$.

Dokaz:
1. Sledi iz definicije.
2. $(\Rightarrow) : \mathrm{red}(a) = 1 \Rightarrow a^1 = e \Rightarrow a = e$
$(\Leftarrow) : e^1 = e \Rightarrow \mathrm{red}(e) = 1.$
3. $\{e, a, a^2, \ldots, a^{\mathrm{red}(a)-1}\}$ je podgrupa v G reda $\mathrm{red}(a)$. Zato po Langrangevem izreku sledi, da $\mathrm{red}(a) | n$.
4. Iz prejšnje trditve sledi, da je $n = \mathrm{red}(a)k$, za neko naravno število k. Iz tega sledi: $a^n = a^{\mathrm{red}(a)k} = (a^{\mathrm{red}(a)})^k = e^k = e$. □

Pri Paleyevi konstrukciji Hadamardovih matrik bomo definirali karakteristično funkcijo, za dokazovanje lastnosti te funkcije bo pomembna spodnja trditev.

Definicija 3.5 *Grupa G je* ciklična*, če obstaja $a \in G$ tako, da je vsak element grupe G enak neki potenci elementa a. Element a imenujemo generator grupe G.*

Trditev 3.9 *Če je moč grupe praštevilo, potem je grupa ciklična.*

Dokaz:
Naj bo $a \in G$ in naj bo $\text{red}(a) = p$. Zato je $\{e, a, a^2, \ldots, a^{p-2}, a^{p-1}\} = G$. Red enote je 1. Red vseh ostalih je p. □

3.4 Komutativni obseg

Ker v konstrukciji Hadamardovih matrik uporabljamo komutativne obsege, bomo v nadaljevanju le te definirali in pokazali nekaj primerov.

Definicija 3.6 *Komutativni obseg je množica $\mathbb{F}$, v kateri sta definirani operaciji $(\times, +)$ in če so $a, b, c \in \mathbb{F}$, veljajo naslednje lastnosti:*

1. *operaciji sta notranji: $a + b \in \mathbb{F}$ in $a \times b \in \mathbb{F}$,*
2. *komutativnost za seštevanje: $a + b = b + a$,*
3. *komutativnost za množenje: $a \times b = b \times a$,*
4. *asociativnost za seštevanje: $(a + b) + c = a + (b + c)$,*
5. *asociativnost za množenje: $(a \times b) \times c = a \times (b \times c)$,*
6. *distributivnost: $a \times (b+c) = a \times b + a \times c$ in $(a+b) \times c = a \times c + b \times c$,*
7. *v množici $\mathbb{F}$ obstaja nevtralni element za seštevanje (označimo ga z 0), tako da velja: $a + 0 = 0 + a = a$,*
8. *v množici $\mathbb{F}$ obstaja nevtralni element za množenje (označimo ga z 1), tako da velja: $a \times 1 = 1 \times a = a$ (1 imenujemo enota),*
9. *za vsak $a \in \mathbb{F}$ obstaja nasprotni element $-a \in \mathbb{F}$, tako da velja: $a + (-a) = (-a) + a = 0$,*
10. *za vsak $a \neq 0 \in \mathbb{F}$ obstaja inverzni element $a^{-1} \in \mathbb{F}$, tako da velja: $a \times a^{-1} = a^{-1} \times a = 1$.*

Primera:

(a) Množica racionalnih števil $\mathbb{Q}$ z opracijama seštevanje in množenje je komutativni obseg:

1. operaciji sta notranji,
2. komutativnost za seštevanje velja,
3. komutativnost za množenje velja,
4. asociativnost za seštevanje velja,
5. asociativnost za množenje velja,
6. distributivnost velja,
7. obstaja nevtralni element za seštevanje: 0,
8. obstaja enota za množenje: 1,
9. za vsak $a \in \mathbb{Q}$ obstaja nasprotni element $-a$,
10. za vsak $a \neq 0 \in \mathbb{Q}$ obstaja inverzni element $a^{-1} = \frac{1}{a}$.

(b) Množica ostankov po modulu 3, to je $\mathbb{F}_3 = \{0, 1, 2\}$, z operacijama seštevanje in množenje je komutativni obseg:

1. operacija seštevnje je notranja: $0 + 0 = 0$, $0 + 1 = 1$, $0 + 2 = 2$, $1 + 0 = 1$, $1 + 1 = 2$, $1 + 2 = 0$, $2 + 0 = 2$, $2 + 1 = 0$, $2 + 2 = 1$,
2. operacija množenje je notranja: $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $0 \cdot 2 = 0$, $1 \cdot 0 = 0$, $1 \cdot 1 = 1$, $1 \cdot 2 = 2$, $2 \cdot 0 = 0$, $2 \cdot 1 = 2$, $2 \cdot 2 = 1$,
3. komutativnost za seštevanje in množenje velja,
4. asociativnost za seštevanje velja,
5. asociativnost za množenje velja,
6. distributivnost velja,
7. nevtralni element za seštevanje je 0,
8. enota za množenje je 1,

9. za vsak $a \in \mathbb{F}_3$ obstaja nasprotni element: $0 \to 0$, $1 \to 2$, $2 \to 1$,
10. za vsak $a \neq 0 \in \mathbb{F}_3$ obstaja inverzni element: $1 \to 1$, $2 \to 2$.

Poglavje 4

Hadamardove matrike

4.1 Hadamardove matrike

Definicija 4.1 *Kvadratni matriki H z n vrsticami in z elementi ± 1, za katero velja*

$$HH^T = nI_n, \tag{4.1}$$

pravimo Hadamardova matrika reda n.

Glede na to, da zgornji matrični produkt sestavljajo medsebojni produkti vrstic matrike H, lahko stolpce Hadamardove matrike poljubno premešamo ali pa pomnožimo z -1, pa bo nova matrika še vedno Hadamardova. Rekli bomo, da sta si taki Hadamardovi matriki *ekvivalentni*. Na prvi pogled se zdi, da identiteta (4.1) zagotavlja le pravokotnost vrstic. Poglejmo si še, kako je s stolpci:

$$\begin{aligned}
HH^T &= nI_n \mid \text{pomnožimo z desne s } H \\
HH^TH &= nH \mid \text{pomnožimo z leve s } H^{-1} \\
H^{-1}HH^TH &= H^{-1}Hn \mid \text{upoštevamo obrljivost Hadamardovih matrik} \\
I_nH^TH &= nI_n \mid \text{in dobimo} \\
H^TH &= nI_n.
\end{aligned}$$

Iz tega sledi med drugim tudi to, da so tudi stolpci paroma pravokotni. Primeri nekaj manjših Hadamardovih matrik:

$$n = 1 : H_1 = \begin{bmatrix} 1 \end{bmatrix}$$

$$n = 2 : H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$n = 4 : H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$n = 8 : H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

Vidimo, da sta v vseh predstavljenih primerih prvi stolpec in prva vrstica sestavljena iz samih enic. Same enice v prvem stolpcu oz. vrstici Hadamardove matrike dobimo tako, da vrstice oz. stolpce matrike H, ki se začne z negativnim številom, pomnožimo z -1. Temu postopku pravimo normalizacija.

Izrek 4.1 *Če je H_n Hadamardova matrika reda n, potem je $n = 1$ ali $n = 2$ ali $n \equiv 0 \pmod 4$.*

Dokaz:
Naj bo $n > 2$. Normaliziramo matriko H, nato lahko s permutacijo stolpcev uredimo matriko H tako, da dobimo:

$$\begin{array}{cccc} ++\cdots++ & ++\cdots++ & ++\cdots++ & ++\cdots++ \\ ++\cdots++ & ++\cdots++ & --\cdots-- & --\cdots-- \\ \underbrace{++\cdots++}_{a} & \underbrace{--\cdots--}_{b} & \underbrace{++\cdots++}_{c} & \underbrace{--\cdots--}_{d} \end{array}$$

Če upoštevamo definicijo Hadamardovih matrik, da je $HH^T = nI_n$, potem za a, b, c, $d \in \mathbb{N}_0$ velja:

$$\begin{aligned} &a+b+c+d = n, \\ &a-b+c-d = 0 \quad \text{(produkt tretje vrstice s prvo vrstico)}, \\ &a+b-c-d = 0 \quad \text{(produkt druge vrstice s prvo vrstico) in} \\ &a-b-c+d = 0 \quad \text{(produkt druge vrstice s tretjo vrstico)}, \end{aligned}$$

vsota zgornjih enačb pa nam da $4a = n$. □

4.2 Kompleksne Hadamardove matrike

Če namesto $H_{ij} = \pm 1$ zahtevamo pri definiciji Hadamardove matrike raje $|H_{ij}| = 1$, transponiranje pa pomeni Hermitsko transponiranje (konjugiranje in navadno transponiranje), dobimo kompleksne Hadamardove matrike. Podobno kot pri Hadamardovih matrikah opisanih v prejšnjem razdelku imamo v bistvu natanko eno kompleksno Hadamardovo matriko reda 3 (t.i. Butson-Hadamardovo matriko) in jo sestavimo s primitivnim kubičnim korenom števila 3, ki ga označimo z w:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & w & w^2 \\ 1 & w^2 & w \end{pmatrix}.$$

Kompleksne Hadamardove matrike obstajajo za vsako naravno število n (za razliko od realnih). Npr. Fourierove matrike $(F_n)_{jk} := e^{2\pi i(j-1)(k-1)/n}$, za $j, k = 1, 2, \ldots, n$, pripadajo temu razredu.

Vrstice matrike H_n so med sabo pravokotne ter vse imajo normo $||\cdot||_2$ natanko $\sqrt{n}$ (enako velja za stolpce).

Trditev 4.1 (Hadamard, 1893) *Naj bo A kompleksna matrika velikosti $n \times n$ z linearno neodvisnimi stolpci $z_1, z_2, \ldots, z_n$. Potem velja*

$$|det(A)|^2 \leq \prod_{k=1}^{n} ||z_k||^2.$$

Dokaz:
Dokaz se nahaja v [20]. □

Zgornja trditev se imenuje tudi Hadamardov problem največje determinante. Tako sledi, da je H_n rešitev Hadamardovega problema največje determinante oziroma natančneje, matrika H_n ima največjo determinanto med vsemi kompleksnimi $n \times n$-dimenzionalnimi matrikami A, za katere velja $|(A)_{i,j}| \leq 1$ (Hadamard, 1893)

$$|\det(H_n)| = n^{n/2}. \qquad \text{[20, stran 2]}$$

4.3 Hadamardova matrična domneva

Hadamard je 1893 zapisal naslednjo domnevo [7]: Naj bo $s \in \mathbb{N}$. Tedaj obstaja Hadamardova matrika reda n = 4s.

Domneva ni bila nikoli dokazana, kljub temu pa poznamo kar nekaj konstrukcij Hadamardovih matrik, s pomočjo katerih dobimo vse matrike za $n < 668$ oz. $s < 167$ in tudi nekatere večje. Hadamardova matrika reda 668 trenutno velja za najmanjšo matriko, ki je še ne znamo konstruirati. Nekatere konstrukcije Hadamardovih matrik so opisane v naslednjem razdelku.

4.4 Konstrukcije Hadamardovih matrik

4.4.1 Sylvestrova konstrukcija

Najenostavneša konstrukcija nove Hadamardove matrike iz že poznane Hadamardove matrike je Kroneckerjev produkt.

Kroneckerjev produkt

Kroneckerjev produkt je operacija, ki se izvaja na dveh matrikah poljubne velikosti.

Definicija 4.2 *Naj bo A matrika reda $p \times q$ in naj bo B matrika reda $m \times n$, potem je* Kroneckerjev produkt $A \otimes B$ *bločna matrika reda $pm \times qn$:*

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ \vdots & & & \vdots \\ a_{p1}B & a_{p2}B & \cdots & a_{pq}B \end{bmatrix}$$

Primer:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} & 2 \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \\ 3 \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} & 4 \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \otimes \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} & 2 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} & 3 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} \\ 4 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} & 5 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} & 6 \cdot \begin{bmatrix} 5 \\ 6 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 5 & 10 & 15 \\ 6 & 12 & 18 \\ 20 & 25 & 30 \\ 24 & 30 & 36 \end{bmatrix}$$

Naj bodo A, B in C poljubne matrike in k poljuben skalar, potem veljajo neslednje lastnosti [16, stran 140]:

1. $A \otimes (B + C) = A \otimes B + A \otimes C$,
2. $(A + B) \otimes C = A \otimes C + B \otimes C$,
3. $(kA) \otimes B = A \otimes (kB) = k(A \otimes B)$,
4. $(A \otimes B) \otimes C = A \otimes (B \otimes C)$,
5. $(A \otimes B) \cdot (C \otimes D) = (AC) \otimes (BD)$,
6. $(A \otimes B)^T = A^T \otimes B^T$.

Z naslednjim izrekom bomo pokazali, da za vsak $n \in \mathbb{N}$ obstaja Hadamardova matrika reda 2^n.

Izrek 4.2 *Kroneckerjev produkt Hadamardovih matrik je Hadamardova matrika.*

Dokaz:
Naj bosta matriki H_n in H_m Hadamardovi matriki reda n oziroma m, potem velja:

$$\begin{aligned}(H_n \otimes H_m) \cdot (H_n \otimes H_m)^T &= \\ &= (H_n \otimes H_m) \cdot (H_n^T \otimes H_m^T) = \\ &= (H_n H_n^T) \otimes (H_m H_m^T) = \\ &= (nI_n) \otimes (mI_m) = \\ &= nm(I_n \otimes I_m) = \\ &= nmI_{nm}\end{aligned}$$

Iz zgornjega računa je razvidno, da smo s Kroneckerjevim produktom Hadamardovih matrik dobili zopet Hadamardovo matriko. □

Sylvestrova matrika $S(k)$ reda 2^k je k-ta iteracija Kroneckerjevega produkta Hadamardove matrike H_2.

$$k = 1 : S(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H_2$$

$$k = 2 : S(2) = H_2 \otimes S(1) = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$k = 3 : S(3) = H_2 \otimes S(2)$$

$$k = 4 : S(4) = H_2 \otimes S(3)$$

$$\vdots$$

$k = n : S(n) = H_2 \otimes S(n-1)$

S tem dobimo Hadamardove matrike reda 2^n za vsak $n \in \mathbb{N}$.

4.4.2 Paleyjeve matrike

V tem razdelku bomo predstavili konferenčne matrike, ki jih je leta 1933 konstruiral Paley [9] in uporabo teh matrik pri konstrukciji Hadamardovih matrik.

Definicija 4.3 *Kvadratna matrika C reda n je* simetrična *natanko takrat, ko zadošča pogoju*

$$C = C^T.$$

Primer:

$$A = \begin{bmatrix} 0 & 2 & -3 \\ 2 & 0 & 5 \\ -3 & 5 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 0 \end{bmatrix}$$

Definicija 4.4 *Kvadratna matrika C reda n je* antisimetrična *natanko takrat, ko zadošča pogoju*

$$C = -C^T.$$

Primer:

$$C = \begin{bmatrix} 0 & 2 & -3 \\ -2 & 0 & -5 \\ 3 & 5 & 0 \end{bmatrix}, \qquad C = \begin{bmatrix} 0 & 1 & -1 & -1 \\ -1 & 0 & -1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 0 \end{bmatrix}$$

Definicija 4.5 *Kvadratno matriko C reda n z elementi*

$$c_{ij} = \begin{cases} 0 & \text{če je } i = j, \\ +1 \text{ ali } -1 & \text{če je } i \neq j \end{cases}$$

imenujemo konferenčna *matrika, če zadošča pogoju*

$$CC^T = (n-1)I.$$

Naj bo C konferenčna matrika [5: stran: 200-203] reda $n \neq 1$, potem je n sodo število. S permutacijo vrstic in stolpcev ter z množenjem vrstic in stolpcev z -1 pa dobimo:

- če je n=2, ekvivalentno simetrično ali antisimetrično matriko,
- sicer pa:
  - če je $n \equiv 2 \pmod 4$, ekvivalentno simetrično konferečno matriko in
  - če je $n \equiv 0 \pmod 4$, ekvivalentno antisimetrično konferenčno matriko.

Primer:

$n = 2$ in C je simetrična: $C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ ali $C = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$

$n = 2$ in C je antisimetrična: $C = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ ali $C = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

$n = 4$ in C je antisimetrična: $C = \begin{bmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$

$n = 6$ in C je simetrična: $C = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 & -1 & 1 \\ 1 & 1 & 0 & 1 & -1 & -1 \\ 1 & -1 & 1 & 0 & 1 & -1 \\ 1 & -1 & -1 & 1 & 0 & 1 \\ 1 & 1 & -1 & -1 & 1 & 0 \end{bmatrix}$

V nadaljevanju bomo pokazali, kako lahko iz konferenčne matrike skonstruiramo Hadamardovo matriko.

Izrek 4.3 *Če je C antisimetrična konferenčna matrika, potem je $I + C$ Hadamardova matrika.*

Dokaz:
$(I+C)(I+C)^T = (I+C)(I^T+C^T) = I+C+C^T+CC^T = I+C-C+CC^T = I + (n-1)I = nI$ □

Izrek 4.4 *Če je C simetrična konferenčna matrika reda n, potem je*

$$H = \begin{bmatrix} I+C & -I+C \\ -I+C & -I-C \end{bmatrix}$$

Hadamardova matrika reda $2n$.

Dokaz:

$$\begin{aligned}
&HH^T = \\
&= \begin{bmatrix} (I+C)(I+C)+(-I+C)(-I+C) & (I+C)(-I+C)+(-I+C)(-I-C) \\ (-I+C)(I+C)+(-I+C)(-I+C) & (-I+C)(-I+C)+(-I+C)(-I-C) \end{bmatrix} = \\
&= \begin{bmatrix} 2I+2CC & 0 \\ 0 & 2I+2CC \end{bmatrix} = \\
&= \begin{bmatrix} 2I+2(n-1)I & 0 \\ 0 & 2I+2(n-1)I \end{bmatrix} = \\
&= \begin{bmatrix} 2nI & 0 \\ 0 & 2nI \end{bmatrix} = \\
&= 2nI_{2n}
\end{aligned}$$

□

Spodaj bomo konstruirali posebne konferenčne matrike, ki jih je leta 1933 predstavil Paley [9]. Pri sami konstrukciji je uporabil karakteristično funkcijo [5: stran: od 199 do 204] nad komutativnim obsegom.

Definicija 4.6 *Število q je* kvadratni ostanek *po modulu p, če obstaja tako število $r \in \mathbb{N}$, da velja*

$$q \equiv r^2 \pmod{p}.$$

Definicija 4.7 *Naj bo q potenca lihega praštevila. V komutativnem obsegu $\mathbb{F}_q$ ostankov po modulu q definiramo* karakteristično funkcijo

$$\chi(x) = \begin{cases} 0 & \text{če je } x \text{ enak } 0, \\ 1 & \text{če je } x \text{ kvadratni ostanek po modulu } q, \\ -1 & \text{sicer.} \end{cases}$$

Za vsak x in y iz $\mathbb{F}_q$ velja:

$$\chi(x)\chi(y) = \chi(xy). \qquad \text{[5: stran 202]}$$

Naj bo q potenca lihega praštevila. Množica $\mathbb{F}_q$ je grupa za množenje in seštevanje in reda q, zato je $\mathbb{F}_q$ ciklična grupa za množenje. Ker je v množici $\mathbb{F}_q$ polovica elementov kvadratnih ostankov po modulu q in polovica elementov kvadratnih ne-ostankov, velja tudi enačba:

$$\sum_{x \in \mathbb{F}_q} \chi(x) = 0.$$

Naj bo $0 \neq c \in \mathbb{F}_q$, potem velja:

$$\sum_{b \in \mathbb{F}_q} \chi(b)\chi(b+c) = -1. \tag{4.2}$$

To trditev dokažemo z naslednjim razmislekom:

- če je $b = 0$, potem je $\chi(b)\chi(b+c) = 0$,
- sicer pa je $\chi(b)\chi(b+c) = \chi(b)\chi(b(1+cb^{-1})) = \chi(b)\chi(b)\chi(1+cb^{-1}) = 1 \cdot \chi(1+cb^{-1})$. Če b teče po vseh elementih polja $\mathbb{F}_q$, ki so različni od 0, potem izraz $1 + cb^{-1}$ zavzame vse vrednosti razen 1. V množici $\mathbb{F}_q$ je polovica elementov kvadratnih ostankov po modulu q in polovica elementov kvadratnih ne-ostankov. Torej je zgornja vsota enaka -1

Definicija 4.8 *Označimo elemente komutativnega obsega $\mathbb{F}_q$ na naslednji način:*

$$0 = a_0, a_1, a_2, \ldots, a_{q-1}.$$

Kvadratno matriko Q reda q definiramo kot:

$$q_{ij} = \chi(a_i - a_j), \quad 0 \leq i, j < q.$$

Matrika Q je simetrična, če je $q \equiv 1 \pmod 4$ in antisimetrična, če je $q \equiv 3 \pmod 4$.

Z uporabo zgoraj definirane matrike lahko sedaj skonstruiramo konferenčno matriko, ki jo bomo uporabili v Paleyevi konstrukciji Hadamardove matrike.

Definicija 4.9 *Matriko C velikosti $(q+1)\times(q+1)$ definirajmo na naslednji način:*

$$C = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ \pm 1 & & & & \\ \vdots & & & Q & \\ \pm 1 & & & & \end{bmatrix},$$

kjer predznak ± 1 izberemo tako, da je matrika C simetrična ali antisimetrična.

Iz definicije in lastnosti matrike Q sledi, da je matrika C konferenčna matrika reda $q+1$. Tej konstrukciji pravimo Palayeva konstrukcija, matrikam te oblike pa Paleyeve matrike.

Izrek 4.5 (Paley, 1933) *Naj bo q potenca lihega praštevila, potem obstaja Hadamardova matrika reda $q+1$, če je $q \equiv 3 \pmod 4$, in Hadamardova matrika reda $2(q+1)$, če je $q \equiv 1 \pmod 4$.* [9]

Dokaz:
Naj bodo $a_1, a_2, \ldots, a_q$ elementi polja $\mathbb{F}_q$ in matrika $B = \{b_{ij}\}_{1\le i,j\le q}$, tako da je $b_{ij} = \chi(a_i - a_j)$. Za matriko B velja:

$$\begin{aligned} (BB^T)_{ij} &= \textstyle\sum_{1\le k\le q} b_{ik}b_{jk} = \\ &= \textstyle\sum_{1\le k\le q} \chi(a_i - a_k)\chi(a_j - a_k) = \\ &= \textstyle\sum_{a\in\mathbb{F}_q} \chi(a)\chi(a_j - a_i + a) = \\ &= \begin{cases} -1 & \text{če je } i \ne j \\ q-1 & \text{sicer.} \end{cases} \end{aligned}$$

Za vsak $1 \le i \le q$ imamo

$$\sum_{1\le k\le q} b_{ik} = \sum_{1\le k\le q} \chi(a_i - a_k) = \sum_{c\in\mathbb{F}_q} \chi(c) = 0$$

$$b_{ij} = \chi(a_i - a_j) = \chi(-1)\chi(a_j - a_i) = \chi(-1)b_{ji} =$$
$$= \begin{cases} b_{ji} & \text{če je } q \equiv 1 \pmod 4 \\ -b_{ji} & \text{če je } q \equiv 3 \pmod 4 \end{cases}$$

Iz zgornje enakosti vidimo, da če je $q \equiv 1 \pmod 4$, je matrika B simetrična in lahko definiramo matriko

$$C = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & & & & \\ \vdots & & & B & \\ 1 & & & & \end{bmatrix}$$

in če je $q \equiv 3 \pmod 4$, je matrika B antisimetrična in lahko definiramo matriko C kot:

$$C = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ -1 & & & & \\ \vdots & & & B & \\ -1 & & & & \end{bmatrix}.$$

Vidimo, da sta tako dobljeni matriki C konferenčni matriki, zanju velja $CC^T = qI$. Z uporabo izreka (2.3) in (2.4) dobimo ustrezni Hadamardovi matriki reda $q+1$ in reda $2(q+1)$. □

Primer Hadamardove matrike reda 12 s konstrukcijo Paleyeve matrike reda 5+1:
Najprej naredimo konstrukcijo matrike Q z uporabo definicije (4.8), pri čemer je $q = 5$:

$$Q = \begin{bmatrix} 0 & -1 & -1 & +1 & +1 \\ +1 & 0 & +1 & -1 & -1 \\ -1 & +1 & 0 & +1 & -1 \\ -1 & -1 & +1 & 0 & +1 \\ +1 & -1 & -1 & +1 & 0 \end{bmatrix}.$$

Nato naredimo simetrično konferečno matriko C z uporabo definicije (4.9):

$$C = \begin{bmatrix} 0 & +1 & +1 & +1 & +1 & +1 \\ +1 & 0 & -1 & -1 & +1 & +1 \\ +1 & +1 & 0 & +1 & -1 & -1 \\ +1 & -1 & +1 & 0 & +1 & -1 \\ +1 & -1 & -1 & +1 & 0 & +1 \\ +1 & +1 & -1 & -1 & +1 & 0 \end{bmatrix}.$$

In na koncu še uporabimo izrek (4.4) in dobimo Hadamardovo matriko:

+1	+1	+1	+1	+1	+1	-1	+1	+1	+1	+1	+1
+1	+1	+1	−1	−1	+1	+1	−1	+1	−1	−1	+1
+1	+1	+1	+1	−1	−1	+1	+1	−1	+1	−1	−1
+1	−1	+1	+1	+1	−1	+1	−1	+1	−1	+1	−1
+1	−1	−1	+1	+1	+1	+1	−1	−1	+1	−1	+1
+1	+1	−1	−1	+1	+1	+1	+1	−1	−1	+1	−1
−1	+1	+1	+1	+1	+1	−1	−1	−1	−1	−1	−1
+1	−1	+1	−1	−1	+1	−1	−1	−1	+1	+1	−1
+1	+1	−1	+1	−1	−1	−1	−1	−1	−1	+1	+1
+1	−1	+1	−1	+1	−1	−1	+1	−1	−1	−1	+1
+1	−1	−1	+1	−1	+1	−1	+1	+1	−1	−1	−1
+1	+1	−1	−1	+1	−1	−1	−1	+1	+1	−1	−1

Primer Hadamardove matrike reda 12 s konstrukcijo Paleyeve matrike reda 11+1:

+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
−1	+1	+1	−1	+1	+1	+1	−1	−1	−1	+1	−1
−1	−1	+1	+1	−1	+1	+1	+1	−1	−1	−1	+1
−1	+1	−1	+1	+1	−1	+1	+1	+1	−1	−1	−1
−1	−1	+1	−1	+1	+1	−1	+1	+1	+1	−1	−1
−1	−1	−1	+1	−1	+1	+1	−1	+1	+1	+1	−1
−1	−1	−1	−1	+1	−1	+1	+1	−1	+1	+1	+1
−1	+1	−1	−1	−1	+1	−1	+1	+1	−1	+1	+1
−1	+1	+1	−1	−1	−1	+1	−1	+1	+1	−1	+1
−1	+1	+1	+1	−1	−1	−1	+1	−1	+1	+1	−1
−1	−1	+1	+1	+1	−1	−1	−1	+1	−1	+1	+1
−1	+1	−1	+1	+1	+1	−1	−1	−1	+1	−1	+1

Z uporabo Paleyevih konstrukcij in Sylvestrove konstrukcije lahko skonstruiramo zelo veliko družino Hadamardovih matrik. Obstajajo pa tudi take Hadamardove matrike, ki jih te konstrukcije ne pokrijejo. Najmanjša med njimi je H_{92}.

4.4.3 Druge konstrukcije

Poleg omenjenih obstaja še veliko drugih konstrukcij. Med najbolj znanimi sta konstrukcija z Baumert-Hallovimi tabelami in Williamsova konstrukcija. Hadamardove matrike lahko konstruiramo tudi iz latinskih kvadratov, Steinerjevih trojk itd.

Že samo opisane konstrukcije v prejšnjih poglavjih so dovolj, da skonstruiramo Hadamardove matrike do reda 100, z izjemo $n = 92$. S Paleyjevim izrekom znamo skonstruirati Hadamardovo matriko reda n, ko $2^e | n$ za $e > 1$ in je $\frac{n}{2^e} - 1$ potenca lihega praštevila. Za $n < 1000$ nam potem ostanejo še naslednje velikosti: 92, 116, 156, 172, 184, 188, 232, 236, 260, 268, 292, 324, 356, 372, 376, 404, 412, 428, 436, 452, 472, 476, 508, 520, 532, 536, 584, 596, 604, 612, 652, 668, 712, 716, 732, 756, 764, 772, 808, 836, 852, 856, 872, 876, 892, 904, 932, 940, 944, 952, 956, 964, 980, 988, 996.

L. Baumert, S. W. Golomb in M. Hall leta 1962 so, poleg metode J. Williamsona, ki je prvi skonstruiral Hadamardovo matriko reda 172, že uporabili računalnike. Slavna Hadamardova matrična domneva iz leta 1893 pravi, da obstaja Hadamardova matrika reda $4n$ za vsako naravno število n. Leta 2004 sta iranska matematika H. Kharaghani in B. Tayfeh-Rezaie konstruirala Hadamardovo matriko reda 428. Najmanjši odprti primeri obstoja Hadamardove matrike so sedaj matrike reda 668, 716, 876, 892. Najnovejšo konstrukcijo (Hadamardova matrika reda 764) pa je leta 2008 odkril Djoković [13].

Poglavje 5

Osnove kodiranja

5.1 Uvod

V času informacijske tehnologije (zgoščenke, mobilni telefoni, bančne kartice, internet) se vsi dobro zavedamo pomena hitrega in natančnega prenosa, obdelovanja in hranjenja informacij. Še tako popolne naprave delajo napake, le-te pa lahko hitro spremenijo sicer izredno koristno programsko in strojno opremo v ničvredno ali celo nevarno orodje. Dolgo časa so se ljudje trudili izdelati računalnike in pomnilnike, ki bodo vsebovali kar se da malo napak. Seveda so bile zato cene takih izdelkov vedno višje. Potem pa so se domislili, da bi raje računalnike same naučili iskati in odpravljati napake. Za povečanje zanesljivosti prenosa in obdelave informacij smo dolgo časa uporabljali kontrolne bite. Ko je matematik Richard Hamming vnašal v računalnik programe s pomočjo luknjača kartic in mu je nato računalnik večkrat zavrnil paket kartic zaradi napak, se je zamislil: Če zna računalnik sam odkriti napako, zakaj ne zna najti tudi njenega mesta in je odpraviti?"

Resnično nas pogosto bolj zanima, kaj lahko storimo, če pride do tovrstnih napak, saj so računalniki začeli prevzemati večino dela na področju obdelovanja informacij in pri telekomunikacijah. Na začetku so bili računalniški programi dovolj enostavni, tako da so tehnične napake hitro postale očitne. Toda z razvojem strojne opreme so postajali programi vse obsežnejši in bolj

zapleteni, s tem pa je postalo upanje, da bi lahko hitro opazili majhne napake, ki spremenijo delovanje naprave, zanemarljivo, in zato tudi resna skrb. Možnost, da se nam izmuzne kakšna napaka, je vse večja tudi zato, ker so elektronska vezja iz dneva v dan manjša, računalniki pa vse hitrejši. Tudi če je možnost napake ena sama milijardinka, se bo računalnik, ki opravi 2 milijardi osnovnih operacij na sekundo, zmotil približno dvakrat na sekundo.

Odgovor za odpravljanje napak je v kodah. Koda je skupina simbolov, ki predstavlja informacijo. Kode obstajajo že tisočletja. To so npr. hieroglifi, grška abeceda, rimske številke ali pa genetska koda za sestavljanje ribonukleinskih kislin. Nastale so za različne potrebe: za zapis govora ali glasbe, Morsejeva abeceda za prenos informacij, za shranjevanje podatkov itd. Matematična teorija kod, ki se je razvila v zadnjih petdesetih letih, je računalnikarjem in inženirjem omogočila, da sestavijo sisteme, ki so kar se da zanesljivi. Tehnologija kod za popravljanje napak je danes tako razširjena kot zgoščenke. Omogoča nam, da poslušamo glasbo brez kakršnih koli motenj, četudi je zgoščenka pošteno spraskana.

5.2 Kodiranje

Leta 1948 je Richard Hamming izumil metodo za popravljanje ene napake in odkrivanje dveh napak. Bistvo vseh metod za odpravljanje napak je dodajanje kontrolnih bitov. Najenostavnejša koda za odpravljanje napak je zasnovana na ponavljanju. Če pričakujemo, da pri prenosu podatkov ne bo prišlo do več kot ene same napake, potem je dovolj, da vsak bit ponovimo trikrat in pri sprejemu uporabimo tako imenovano "večinsko pravilo". Simbol 1101 zakodiramo v 111 111 000 111. Če prejmemo 111 011 000 111, popravimo sporočilo v 111 111 000 111 in ga nazadnje še odkodiramo v 1101. V splošnem lahko odpravimo n napak z $(2n + 1)$-kratnim ponavljanjem in uporabo večinskega pravila. Toda ta metoda je preveč potratna. V času, ko si želimo hitrega prenosa čim večje količine podatkov, je takšno napihovanje večinoma nesprejemljivo. Namesto tega si želimo dodati manjše število

kontrolnih bitov, ki pa naj bodo ravno tako, ali pa morda še bolj, učinkoviti.

Koda je podmnožica nekega prostora, v katerem je definirana razdalja. Elementom kode pravimo tudi kodne besede. Kodiranje je prirejanje kodnih besed posameznim informacijam. Če pri prenosu kodne besede ni prišlo do napak, je odkodiranje obratno prirejanju. V nasprotnem primeru pa pri odkodiranju običajno prejeto besedo popravimo tako, da izberemo kodno besedo, ki je prejeti besedi najbližja. Ta postopek imenujemo tudi princip najbližjega soseda. To si lahko tudi predstavljamo kot prostor večnadstropne stanovanjske hiše (blok), kodne besede pa so središča izbranih sob. Informacijo predstavimo (zakodiramo) tako, da v središča nekaterih izbranih sob postavimo žoge. Prenos informacij je potem podoben potresu, ki žoge prestavlja. Odkodirni proces pa poskuša žoge vrniti nazaj v središča. Če je potres zelo močan in žoga konča v drugi sobi, potem pride do napačnega popravka, kadar pa žoga ne zapusti sobe, vsaj na prvi pogled ni problema.

Za prostor si izberemo množico vseh n-teric s simboli iz neke končne množice F, imenovane tudi abeceda:

$$F^n = \{(a_0, a_1, a_2, \ldots, a_{n-1}) | a_i \in F, \quad i = 0, 1, 2, \ldots, n-1\}.$$

Definicija 5.1 *Če je $x \in F^n$ in $y \in F^n$, potem je* razdalja $d(x, y)$ *med x in y definirana kot:*

$$d(x, y) = |\{i | 1 \leq i \leq n, x_i \neq y_i\}|.$$

Primer razdalje: F naj bo množica ostankov pri deljenju s 5 in $n = 6$. Če je $x = (0, 3, 2, 1, 4, 3) \in F^6$ in $y = (2, 3, 2, 4, 4, 4) \in F^6$, potem je razdalja med x in y enaka $d(x, y) = |\{1, 4, 6\}| = 3$.

Razdalja kode je najmanjša razdalja med različnimi kodnimi besedami. V tem primeru je razdalja med dvema n-tericama število mest, na katerih se razlikujeta. Tej razdalji pravimo tudi *Hammingova razdalja.* Kadar je koda podmnožica takega prostora, rečemo, da gre za bločne kode dolžine n. Običajno razbijemo dano sporočilo na bloke fiksne dolžine k, kjer je $0 < k \leq n$, ki jih nato povežemo s kodnimi besedami z neko bijektivno

korespondenco. Taka korespondenca je naravna v primeru, da kodo predstavlja k-razsežen linearen podprostor n-razsežnega vektorskega prostora. V tem primeru rečemo, da gre za linearno (n, k)-kodo.

Če privzamemo odkodirni princip najbližjega soseda, ima koda, ki odpravi (do) t napak, razdaljo $d \geq 2t+1$, saj morajo biti krogle s središčem v kodnih besedah in radijem t disjunktne. Se pravi, da mora biti presek dveh različnih krogel prazen. Pri kodi nas najbolj zanima, koliko napak lahko odpravimo glede na to, koliko kontrolnih bitov dodamo osnovni informaciji.

Lema 5.1 (Singletonova meja) [11, stran 132] Naj bo C bločna koda dolžine n nad abecedo s q elementi in d njena Hammingova razdalja. Potem je število elementov kode C kvečjemu q^{n-d+1}.

Dokaz:
Naj bo C' koda, ki jo konstruiramo iz kode C tako, da izberemo $d-1$ koordinat in jih zbrišemo v vseh kodnih besedah. Ker je razdalja kode C enaka d, imata obe kodi enako število elementov (iz različnih elementov kode C nismo mogli dobiti istega elementa kode C'). Koda C' ima dolžino $n-d+1$, zato ima (in s tem tudi koda C) največ q^{n-d+1} elementov. □

Če izberemo za sporočila vse možne k-terice nad abecedo s q elementi ter obstaja bijekcija med sporočili ter kodnimi besedami iz $C \subseteq F^n$, ima koda C q^k elementov in pravimo, da gre za (n, k)-kodo. V tem primeru se Singletonova meja prevede v zgornjo mejo za razdaljo kode:

$$d \leq n - k + 1.$$

Od tod in neenakosti $2t + 1 \leq d$ sledi, da ima koda vsaj $2t$ kontrolnih bitov:

$$t \leq \left\lfloor \frac{n-k}{2} \right\rfloor.$$

Torej lahko (n, k)-koda odpravi kvečjemu $\lfloor (n-k)/2 \rfloor$ napak.

Poglavje 6

Uporaba Hadamardovih matrik

6.1 Hadamardove kode

Naj bo H Hadamardova matrika reda $4m$. Vzemimo vrstice matrike H in matrike $-H$ ter zamenjajmo vse "-1" z "0". Na ta način dobimo $8m$ vektorjev s $4m$ komponentami nad binarnim obsegom $\mathbb{Z}_2$. Neposredno iz definicije Hadamardove matrike sledi, da je Hammingova razdalja med poljubnima različnima vektorjema bodisi $2m$ ali $4m$ (razdalja do svojega negativa je $4m$, do vseh drugih pa $2m$). Te vrstice generirajo vektorski podprostor v $(\mathbb{Z}_2)^{4m}$, ki predstavlja kodo. Potem je njena najmanjša razdalja $2m$, koda pa lahko popravi do $(m-1)$-napak. Tako dobljenim kodam pravimo Hadamardove kode.

Proces uporabe kod bomo preverili na aplikaciji iz realnega sveta. Mariner 9 je bila vesoljska sonda iz leta 1971, katere namen je bil leteti do Marsa in pošiljati črno-bele slike na Zemljo (prva sonda, ki je letela v orbiti drugega planeta). Čez vsako sliko je bila nameščena podrobna mreža in za vsak kvadratek v tej mreži (piksel) je bila izmerjena sivina na skali od 0 do 63 (tj. 2^6 možnosti oziroma 6 bitov informacije). Ta števila, zapisana v binarni obliki, so bili podatki, ki so bili poslani na Zemljo (bolj natančno v laboratorij kalifornijskega inštituta za tehnologijo v Pasadeni). Ob prihodu je bil signal šibek in je moral biti ojačan. Motnje iz vesolja ter termične motnje iz

ojačevalca povzročijo, da včasih poslano enico preberemo kot ničlo (in obratno ničlo kot enico). Že če je verjetnost napake le 5%, bo ob predpostavki, da ne bomo uporabili nobenih kod, kvaliteta slik izredno slaba (le 26% pravilna, velja namreč $1 - 0,95^6 \approx 0,26$). Torej ni dvoma, da moramo uporabiti kode za odpravljanje napak. Vprašamo pa se lahko tudi, katere kode naj uporabimo. Vsaka koda bo povečala velikost podatkov, ki jih moramo poslati. Mariner 9 je bilo majhno vozilo, ki ni moglo nositi velikega oddajnika, tako da je moral biti signal usmerjen, vendar je na velikih razdaljah signal težko uspešno usmeriti. Poleg tega pa je bila omejena tudi maksimalna velikost podatkov, ki se jih da poslati v danem trenutku (ko je oddajnik naravnan). Le-ta je bila enaka petkratni velikosti originalnih podatkov. Ker so bili podatki sestavljeni iz šestih bitov, so bile lahko kodne besede sestavljene iz tridesetih bitov.

Če vsak simbol ponovimo m-krat, lahko z večinskim pravilom odkodiramo pravilno, če je pri prenosu prišlo do manj kot $m/2$ napak (kode s ponavljanjem). Koda s petimi ponovitvami je bila ena izmed možnosti, saj jo je enostavno implementirati, vendar pa le-ta lahko popravi le 2 napaki. Hadamardova koda, ki je zasnovana na Hadamardovi matriki reda 32, pa lahko popravi 7 napak, tako da je bila vredna nekoliko bolj zapletene implementacije. Z uporabo te kode je verjetnost napake na sliki zreducirana na samo 0,01%. V primeru kode s petimi ponovitvami pa bi bila verjetnost približno 1%.

Predpostavimo, da je verjetnost, da se spremeni en bit, enaka $p = 0,01$. Potem je verjetnost, da je sprejeti piksel napačen $1-0,99^6 \approx 0,06$. Verjetnost v primeru kode s ponavljanjem pa je

$$1 - \left[\binom{5}{0}(1-p)^5 + \binom{5}{1}p(1-p)^4 + \binom{5}{2}p^2(1-p)^3\right]^6$$

in v našem primeru verjetnost $6 \cdot 10^{-5}$, da bo prejeti piksel napačen. V primeru Hadamardove kode pa imamo:

$$1 - \sum_{i=0}^{7}\binom{32}{i}p^i(1-p)^{32-i} = \sum_{i=8}^{32}\binom{32}{i}p^i(1-p)^{32-i},$$

kar nam da bistveno boljšo verjetnost $8 \cdot 10^{-10}$.

6.2 Kodiranje in dekodiranje

V nadaljevanju bomo predstavili problem kodiranja in dekodiranja z uporabo Hadamardove kode. Na prvi pogled kodiranje ne sme predstavljati težav. Imamo namreč 64 različnih podatkov in 64 kodnih besed, kar pomeni, da bi morala delovati že poljubna bijekcija. Težava pa je v tem, da je Mariner 9 majhna naprava, ta pristop pa bi zahteval hranjenje vseh 64 32-bitnih kodnih besed. Veliko bolj ekonomično v smislu prostora in lažje je bilo narediti strojno podporo, ki dejansko izračuna kodno besedo, namesto da jo prebere iz pomnilnika.

S pravilno izbiro Hadamardove matrike postane Hadamardova koda linearna, tako da je ta izračun v resnici množenje podatkov z generatorsko matriko kode. Prava izbira za Hadamardovo matriko je tista, ki jo dobimo iz Kroneckerjevega produkta Hadamardove matrike reda 2. Z matematično indukcijo lahko preverimo, da je taka koda linearna. Sedaj pa si oglejmo pobliže še problem dekodiranja. Prejeti signal, zaporedje 32-ih ničel in enic, je najprej spremenjen v obliko ± 1 (tako da zamenjamo "0" z "-1"). Tako dobimo vektor x in če ni bilo napak, potem je xH^T , kjer je H originalna Hadamardova matrika, vektor z 31-imi koordinatami enakimi 0 in eno koordinato enako ± 32, če pa nastanejo napake, potem se ta števila spremenijo. Vendar pa največ 7 napak poveča vrednost z 0 na največ 14, vrednost 32 pa se zmanjša kvečjemu do 18. V tem primeru prejeta beseda spominja na poslano kodo bolj kot katera koli izmed preostalih 63 kodnih besed. Torej nam mesto, na katerem se pojavi po absolutni vrednosti največja vrednost vektorja xH^T, pove, katera vrstica matrike H (oziroma $-H$, če je originalna vrednost negativna) je bila poslana. Ker je bil originalni algoritem za dekodiranje signalov sonde Mariner 9 počasen (potreboval je 322 množenj in ustrezna seštevanja za vsako kodno besedo), so na Zemlji uporabili številne računske trike in na ta način zreducirali računanje na vsega eno tretjino.

Poglejmo si še algoritem za dekodiranje prejetega signala $\hat{x}$ dolžine $4m$ s pomočjo Hadamardove matrike:

1. korak: spremenimo vse ničle v -1: $x = 2\hat{x} - e$, kje je e vektor samih 1 dolžine $4m$.

2. korak: Izračunamo $s = x \cdot H_{4m}$

3. korak: Če obstaja $k \in \mathbb{N}$, da je $s = \pm 4m \cdot e_k$, kjer je vektor e_k sestavljen iz samih ničel, na k-tem mestu pa ima 1, potem pri prenosu ni prišlo do napake. (Konec)

4. korak: V vektorju s poiščemo največjo komponento po absolutni vrednosti, ki se nahaja na k-tem mestu. Če je največja absolutna vrednost komponent vektorja s manjša od $2m+4$ in če absolutne vrednosti ostalih komponent niso večje od $2m-2$, potem je bila poslana k-ta vrstica matrike H_{4m}. (Konec)

5. korak: Poslan signal ima preveč napak, da bi ga lahko dekodirali.

Primeri:

Naj bo $m = 2$ in $H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$.

(a) Naj bo prejeti signal $\hat{x} = 11000011$, potem je:

1. korak: $x = 2\hat{x} - e = (1, 1, -1, -1, -1, -1, 1, 1)$

2. korak: $s = x \cdot H_8 = (0, 0, 0, 0, 0, 0, 8, 0)$

3. korak: Sedma komponenta vektorja s je največja, ostale komponente so pa eneke 0, torej je dekodirani signal enak prejetemu.

(b) Naj bo prejeti signal $\hat{x} = 10111010$, potem je:

1. korak: $x = 2\hat{x} - e = (1, -1, 1, 1, 1, -1, 1, -1)$
2. korak: $s = x \cdot H_8 = (2, 6, -2, 2, 2, -2, -2, 2)$
3. korak: Preskočimo, ker ne obstaja k, da je $s = \pm 8 \cdot e_k$
4. korak: Največja absolutna vrednost vektorja s je 6 in se nahaja na drugem mestu ($k = 2$) in je manjša od $2 \cdot 2 + 4$, absolutne vrednosti ostalih komponent pa niso večje od $2 \cdot 2 - 2$, torej je rešitev 2. vrstica matrike H_8 $(1, -1, 1, -1, 1, -1, 1, -1) \Rightarrow$ dekodirani signal je 10101010

(c) Naj bo prejeti signal $\hat{x} = 00011010$, potem je:

1. korak: $x = 2\hat{x} - e = (-1, -1, -1, 1, 1, -1, 1, -1)$
2. korak: $s = x \cdot H_8 = (2, 2, -2, -2, -6, 2, -2, -2)$
3. korak: Preskočimo, ker ne obstaja k, da je $s = \pm 8 \cdot e_k$
4. korak: Največja absolutna vrednost vektorja s je 6 in se nahaja na petem mestu ($k = 5$) in je manjša od $2 \cdot 2 + 4$, absolutne vrednosti ostalih komponent pa niso večje od $2 \cdot 2 - 2$, torej je rešitev 5. vrstica matrike $-H_8$ $(-1, -1, -1, -1, 1, 1, 1, 1) \Rightarrow$ dekodirani signal je 00001111

(d) Naj bo prejeti signal $\hat{x} = 11111100$, potem je:

1. korak: $x = 2\hat{x} - e = (1, 1, 1, 1, 1, 1, -1, -1)$
2. korak: $s = x \cdot H_8 = (4, 0, 0, 4, 4, 0, 0, -4)$
3. korak: Preskočimo, ker ne obstaja k, da je $s = \pm 8 \cdot e_k$
4. korak: Preskočimo, ker je $4 > 2 \cdot 2 - 2$.
5. korak: Poslan signal ima preveč napak in ga ne moremo dekodirati.

6.3 Simulacija kodiranja in dekodiranja

V tem poglavju bomo predstavili simulacijo kodiranja s Hadamardovo matriko. Program, ki ga uporabljamo za simulacijo, je napisan v programskem jeziku C# v okolju Visual Studio 2012 in je priložen na CD, glavni del programa je v poglavju Dodatek. Program je namenjen simulaciji napak pri prenosu slik.

Kot vhodni podatek program sprejme 8 bitno sivinsko sliko.

Vir: http://incisors.files.wordpress.com/2008/06/gs.jpg

Iz vhodnega podatka pridobimo binaren zapis vseh pixlov (avtor tega dela kode je Morten Nielsen [22]). Binaren zapis nato skrčimo, tako da 8 bitov premaknemo za dva bita v desno. S tem se izgubita dva bita informacij (dva odtenka sivine, bela in črna barva se ohranita). Tako dobimo 6 bitni zapis. To naredimo zato, ker bomo za kodiranje in dekodiranje uporabili Hadamardovo matriko reda 32. Za prikaz skrčene slike 6 bitov razširimo na 8 bitov s premikom za 2 mesti v levo (za prikazovanje potrebujemo 8 bitov). Pri tem postopku se slika bistveno ne pokvari, kar je vidno spodaj.

Pri pošiljanju in simulaciji napak naredimo naslednje:

6 bitni zapis slike zakodiramo tako, da vsaki 6 bitni vrednosti priredimo kodno besedo iz matrike $C = [H_{32}; -H_{32}]$, kjer je H_{32} Hadamardova matrika reda 32. Za prirejanje uporabimo enostavno bijekcijo, kjer 6 bitni besedi priredimo ustrezno vrstico v matriki C (6 bitno besedo $x_{[2]}$, zapisano v dvojiškem sistemu, pretvorimo v desetiški sistem in dobim vrednost $x_{[10]}$, nato vzamemo iz matrike C vrstico $x_{[10]} + 1$). Na kodiranih besedah in na 6 bitnem zapisu simuliramo napako z verjetnostjo 0,05 (verjetnost je parameter, ki ga lahko spreminjamo, privzeta je 0,05). Spodaj je prikazana nekodirana pokvarjena slika.

Na pokvarjenih kodiranih besedah nato naredimo dekodiranje, kot je opisano v prejšnjem poglavju, in dobim spodnjo sliko.

V primeru, da večamo verjetnost napake, se dobro vidi, da tudi dekodiranje ne more popraviti vseh napak.

Slika, ki jo bomo prenesli:

Vir: http://www.cosy.sbg.ac.at/∼pmeerw/Watermarking/lena_gray.gif

Pokvarjena slika (verjetnost napake je 0,2):

Dekodirana (popravljena) slika:

Poglavje 7

Zaključek

V samem diplomskem delu sem predstavila zame nove algeberske strukture, ki sem jih kasneje potrebovala za razumevanje samih matrik, Hadamardovih matrik in teorije kodiranja. Spoznala sem, da je vsako informacijo potrebno zakodirati na tak način, da ne bo prihajalo do izgube le-teh, saj če je ne znamo dekodirati, je informacija izgubljena.
Pri Hadamardovih kodah je potrebno veliko število operacij, da neko informacijo zakodiramo in nato še dekodiramo, saj so matrike precej velike (po navadi so reda vsaj 32), torej je potrebno veliko logičnih operacij, kar pa je zelo zamudno in potratno. Z uporabo Hadamardovih kod se je razvoj kodiranja začel in se razvija še danes.

Literatura

[1] I. Vidav, *Algebra*, Mladinska knjiga, Ljubljana, 1972.

[2] F. Cajori, *A History of Mathematics*, Madmillan & co, LTD, London, 1909.

[3] S. Lang, *Linear Algebra, 3rd ed.*, Springer, New York, 1987.

[4] J. H. van Lint, *Introduction to Coding Theory, 3rd ed.*, Springer, Berlin, 1999.

[5] J. H. van Lint & R. M. Wilson, *A course in combinatorics*, Cambridge University Press, Cambridge, 2004.

[6] R. A. Horn, C. R. Johnson *Matrix analysis*, Cambridge University Press, Cambridge, 2005.

[7] K. J. Horadam, *Hadamard Matrices and Their Applications*, Princeton University Press, New Jersey, 2007.

[8] P. Nose, *Učinkovita aritmetika na eliptičnih krivuljah nad praštevilskimi obsegi*, diplomsko delo, FRI, FMF, Ljubljana, 2008.

[9] R. E. A. C. Paley, *On orthogonal matrices*, J. Math. Phys., 12, (1933), str. 311-320.

[10] A. Jurišić, *Hadamardove matrike in misija Mariner 9*, Obzornik za matematiko in fiziko, 56, (2009) str. 121-235.

[11] A. Jurišić, A. Žitnik, *Reed-Solomonove kode*, Obzornik za matematiko in fiziko, 51, (2004) str. 129-143.

[12] S. Klavžar, *O teoriji kodiranja, linearnih kodah in slikah z Marsa*, Obzornik za matematiko in fiziko, 45, (1998) str. 97-106.

[13] D. Ž. Djoković, *Hadamard matrics of order 764 exist*, Combinatorica 28 (2008), str. 487-489.

[14] J. J. O'Connor, E. F. Robertson (2003), *Hadamard Biography*. Dostopno na:
http://www-history.mcs.st-andrews.ac.uk/Biographies/Hadamard.html.

[15] M. Malek, *Coding Theory: Hadamard Codes*. Dostopno na:
http://www.mcs.csueastbay.edu/∼malek/TeX/Hadamard.pdf.

[16] Alan J. Laub, *Kronecker Products*. Dostopno na:
http://www.siam.org/books/textbooks/OT91sample.pdf.

[17] E. Oklapi, S. Numanović, *Teorija kodiranja. Hamingov kod i njegova definicija*. Dostopno na:
http://www.dms.rs/petnicamat/data/radovi/2007/letnji/
HamingovKod.pdf.

[18] P. J. Cameron (2011), *Conference matrices*. Dostopno na:
http://www.maths.qmul.ac.uk/∼pjc/csgnotes/conftalk.pdf.

[19] M. Polajnar (2006), *Algebra*. Dostopno na:
http://www.fmf.uni-lj.si/∼skreko/Pouk/ds2/Zapiski/Polajnar-DS2-3.pdf.

[20] *Hadamard's Maximum Determinant Problem*. Dostopno na:
http://www2.fiu.edu/∼draghici/pastcourses/appla-fa11/Hadam_handout.pdf.

[21] Aleksandar Jurišić (2003/2004), *Tečaj iz kriptografije in teorije kodiranja*. Dostopno na:
http://lkrv.fri.uni-lj.si/∼ajurisic/tec3/folije/p21.pdf.

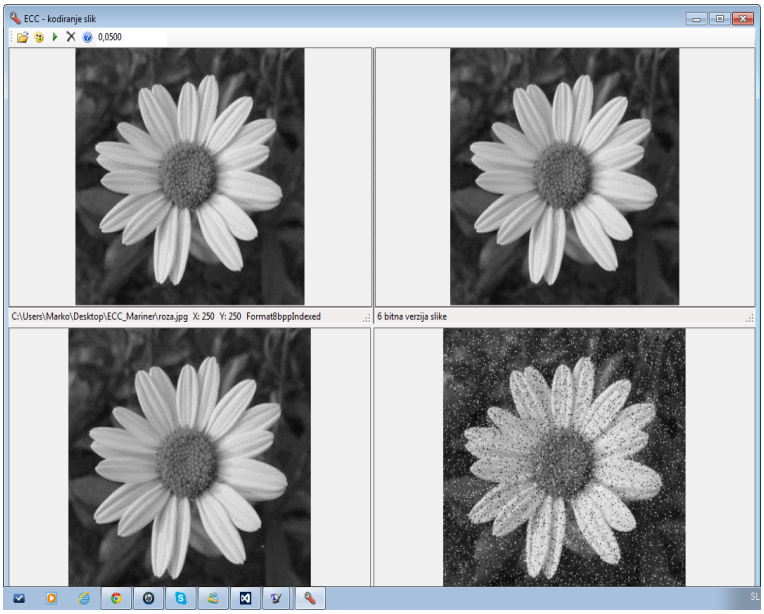
[22] Morten Nielsen (2006), *Working with 8bit images in .NET* Dostopno na:
http://www.sharpgis.net/post/2006/04/05/Working-with-8bit-images-in-NET.aspx.

Poglavje 8

Dodatek

Program, ki ga uporabljam za simulacijo, je napisan v programskem jeziku C# v okolju Visual Studio 2012 in je priložen na CD.

Vnosna maska:



Najprej pritisnemo na prvi gumb v orodni vrstici, da vstavimo 8 bitno sivinsko sliko. Nato pritisnemo na drugi gumb, da jo pretvorimo v 6 bitno. Potem pritisnemo naslednji gumb, ki sliko zakodira in odpošlje. Pri pošiljanju se pokvarijo biti z vrejetnostjo napake, ki jo lahko vnesemo

v vnosnem polju. Istočasno se z enako verjetnostjo pokvarijo tudi biti originalne slike. Nato dekodiramo (popravimo) prejeto sliko in obe sliki prikažemo v spodnji vrstici.

Če se z miško postavimo na sliko in pritisnemo desni gumb na miški, lahko izberemo Pixels. S tem vidimo binarni zapis slike.

Na naslednjih straneh je izpisana koda programa.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace ECC_Mariner
{


    /// <summary>
    /// Programček za simulacijo napak pri prenosih 6 bitnih slik.
    /// 8 bitne sivinske (grayscale) slike pretvori v 6 bitne (6 najpomebnejših bitov).
    /// Na pretvorjenih slika naredi kodiranje s s pomočjo hadamardove matrike H32.
    /// Na kodirani sliki in na nekodirani sliki naredi napake (ju pokvari) z verjetnostjo p (verjetnost napake/bit)
    /// Kodirano sliko z napakami odkodira in prikaže obe, odkodirano z napakami in nekodirano z napakami.
    /// Avtor: Petra Reberšek, 2013
    /// </summary>
    public partial class Form1 : Form
    {
        private Bitmap original;    // originalna slika
        private byte[] originalPixels;  // pixli originalne slike
        private Bitmap original6bit;    // originalna slika 6 bitna
        private byte[] originalPixels6bit;  // pixli originalne slike 6 bitna
        private byte[] originalPixels6bitOf8bit;  // pixli originalne slike 6 bitne pretvorjeno v 8 bitno

        private Bitmap kodirana;     // slika po prenosu kodiranih pixlov
        private byte[] kodiranaPixels;  // pixli prenesene slike z ECC
        private byte[] kodiranaPixels6bitIf8Bit; // pixli prenesene slike zapisani za 8 bitni grayscale
        private byte[][] kodiranaVSE;     // pixli zapakirani v kodne besede (4 byte dolga kodna beseda)
        private byte[][] napakeKodiraneVse; // napake na kodirianh besedah



        private Bitmap nekodirana;   // slika po prenosu nekodiranih pixklov (pokvarjena slika)
        private byte[] nekodiranaPixels;       // pixkli nekodirane slike (pokrvarjeni pixli)
        private byte[] nekodiranaPixels6bitIf8Bit; // pixli prenesene slike zapisani za 8 bitni grayscale


        private float verjetnostNapake; // verjetnos napake (parameter)
        private byte[][] H32, C;     // hadamardvoa matrika in matrika s kodnimi besedami (-1 pri hadamrdovi matriki
    zamenjamo z 0)
        private Random rndGen;      // generator naključnih števil
        private byte[] tabelVrednosti;      // tabela vseh števil od 0 do 64 v binarni obliki
        private int[][] stevilo1in0;    // stevilo 1 in 0 za določeno vrednost (prvi del je vrednost, drugi 1 ali 0)




        public Form1()
        {

            // incializacija vrednosti, privzetih in ostalih, ki jih potrebujemo v programu vedno
            this.verjetnostNapake = 0.05F;
            // inicializacija matrike (H32 = H2*H2*H2*H2*H2, * je kroneckejrev produkt)
            // potrebujemo matriki H32 (hadamardova matrika reda 32) in C = [H;-H] (64 vrstic, 32 stolpcev, vsaka vrstica
    je kodna beseda)
            // matrika ima 32 vrstic, vsaka vrstica vsebuje tabelo 4 byte po 8 bitov, torej 32 bitov
            // matriko C dobimo tako, da negiramo po bitih vrstice od 32 do 63
            // decimalen zapis je bil dobljen s excela
            H32 = new byte[32][];
            for (int i = 0; i < 32; i++)
                H32[i] = new byte[4];
            H32[0][0] = 255; H32[0][1] = 255; H32[0][2] = 255; H32[0][3] = 255;
            H32[1][0] = 170; H32[1][1] = 170; H32[1][2] = 170; H32[1][3] = 170;
            H32[2][0] = 204; H32[2][1] = 204; H32[2][2] = 204; H32[2][3] = 204;
            H32[3][0] = 153; H32[3][1] = 153; H32[3][2] = 153; H32[3][3] = 153;
            H32[4][0] = 240; H32[4][1] = 240; H32[4][2] = 240; H32[4][3] = 240;
            H32[5][0] = 165; H32[5][1] = 165; H32[5][2] = 165; H32[5][3] = 165;
            H32[6][0] = 195; H32[6][1] = 195; H32[6][2] = 195; H32[6][3] = 195;
            H32[7][0] = 150; H32[7][1] = 150; H32[7][2] = 150; H32[7][3] = 150;
            H32[8][0] = 255; H32[8][1] = 0; H32[8][2] = 255; H32[8][3] = 0;
            H32[9][0] = 170; H32[9][1] = 85; H32[9][2] = 170; H32[9][3] = 85;
            H32[10][0] = 204; H32[10][1] = 51; H32[10][2] = 204; H32[10][3] = 51;
            H32[11][0] = 153; H32[11][1] = 102; H32[11][2] = 153; H32[11][3] = 102;
            H32[12][0] = 240; H32[12][1] = 15; H32[12][2] = 240; H32[12][3] = 15;
```

```
83            H32[13][0] = 165; H32[13][1] = 90; H32[13][2] = 165; H32[13][3] = 90;
84            H32[14][0] = 195; H32[14][1] = 60; H32[14][2] = 195; H32[14][3] = 60;
85            H32[15][0] = 150; H32[15][1] = 105; H32[15][2] = 150; H32[15][3] = 105;
86            H32[16][0] = 255; H32[16][1] = 255; H32[16][2] = 0; H32[16][3] = 0;
87            H32[17][0] = 170; H32[17][1] = 170; H32[17][2] = 85; H32[17][3] = 85;
88            H32[18][0] = 204; H32[18][1] = 204; H32[18][2] = 51; H32[18][3] = 51;
89            H32[19][0] = 153; H32[19][1] = 153; H32[19][2] = 102; H32[19][3] = 102;
90            H32[20][0] = 240; H32[20][1] = 240; H32[20][2] = 15; H32[20][3] = 15;
91            H32[21][0] = 165; H32[21][1] = 165; H32[21][2] = 90; H32[21][3] = 90;
92            H32[22][0] = 195; H32[22][1] = 195; H32[22][2] = 60; H32[22][3] = 60;
93            H32[23][0] = 150; H32[23][1] = 150; H32[23][2] = 105; H32[23][3] = 105;
94            H32[24][0] = 255; H32[24][1] = 0; H32[24][2] = 0; H32[24][3] = 255;
95            H32[25][0] = 170; H32[25][1] = 85; H32[25][2] = 85; H32[25][3] = 170;
96            H32[26][0] = 204; H32[26][1] = 51; H32[26][2] = 51; H32[26][3] = 204;
97            H32[27][0] = 153; H32[27][1] = 102; H32[27][2] = 102; H32[27][3] = 153;
98            H32[28][0] = 240; H32[28][1] = 15; H32[28][2] = 15; H32[28][3] = 240;
99            H32[29][0] = 165; H32[29][1] = 90; H32[29][2] = 90; H32[29][3] = 165;
100           H32[30][0] = 195; H32[30][1] = 60; H32[30][2] = 60; H32[30][3] = 195;
101           H32[31][0] = 150; H32[31][1] = 105; H32[31][2] = 105; H32[31][3] = 150;
102
103           C = new byte[64][];
104           for (int i = 0; i < 64; i++)
105               C[i] = new byte[4];
106           for (int i = 0; i < 32; i++)
107           {
108               for (int j = 0; j < 4; j++)
109               {
110                   C[i][j] = H32[i][j];
111                   C[i + 32][j] = (byte)((~(H32[i][j])) & 255);
112               }
113           }
114           this.rndGen = new Random();
115
116           // inicliazicaija tabele vrednosti
117           tabelVrednosti = new byte[8];
118           tabelVrednosti[0] = 1;
119           for (int i = 1; i < 8; i++)
120               tabelVrednosti[i] = (byte)(tabelVrednosti[0] << i);
121           this.stevilo1in0 = new int[256][];
122           for (int i = 0; i < 256; i++)
123           {
124               byte ii = (byte)i;
125               stevilo1in0[i] = new int[2];
126               // ni najbolj ucinkovito, amapak se izvaja samo enkrat in je za nase potrebe dobro, drugace bi naredili s
    shift operatorji
127               stevilo1in0[i][0] =  Convert.ToString(i,2).PadLeft(8, '0').ToCharArray().Count(c => c=='0');
128               stevilo1in0[i][1] = Convert.ToString(i, 2).PadLeft(8, '0').ToCharArray().Count(c => c == '1');
129           }
130
131               InitializeComponent();
132
133       }
134
135       private void toolStripButton1_Click(object sender, EventArgs e)
136       {
137           try
138           {
139               DialogResult resko = this.openFileDialog1.ShowDialog();
140               if (resko == System.Windows.Forms.DialogResult.OK)
141               {
142
143                   System.IO.FileInfo fi = new System.IO.FileInfo(this.openFileDialog1.FileName);
144                   if (!fi.Exists) throw new Exception("Izbrana datoteka (" + this.openFileDialog1.FileName + ") ne
    obstaja.");
145                   Bitmap xx = (Bitmap)Image.FromFile(this.openFileDialog1.FileName);
146                   // zanimajo nas samo 8-bitne grayscale slike, na drugih predstavitve ne delamo
147                   if (xx.PixelFormat != PixelFormat.Format8bppIndexed) throw new Exception("Izberite 8 bitno grayscale
    sliko.");
148                   this.original = xx;
149                   // bitni zapis slike delamo na kopiji, ker objekt Image8Bit zaklene vsebino slike, ki jo obdelujemo,
    dokler ni disposed
150                   Bitmap kopija = (Bitmap)original.Clone();
151                   this.originalPixels = this.ChangeFromBitmapToByte(kopija);
152                   this.pictureBox1.Image = this.original;
153                   this.toolStripStatusLabel3.Text = fi.FullName;
154                   this.toolStripStatusLabel4.Text = "X: " + original.Width.ToString();
155                   this.toolStripStatusLabel5.Text = "Y: " + original.Height.ToString();
156                   this.toolStripStatusLabel6.Text = this.original.PixelFormat.ToString();
157
158
159
160               }
161           }
162           catch (Exception eexx)
```

```
163             {
164                 MessageBox.Show("Napaka pri odpiranju slike: \n" + eexx.Message, "Napaka", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
165             }
166         }
167 
168         private byte[] ChangeFromBitmapToByte(Bitmap picture)
169         {
170             byte[] result = null;
171             try
172             {
173                 if (picture != null)
174                 {
175                     result = new byte[(picture.Height) * (picture.Width)];
176                     // tukaj zaklenemo sliko, zato pri pridobivanju podatkov o bitih (sploh ce jih hranimo, kot je primer
    pri original), delamo na kopijah
177                     Image8Bit bitsForImage = new Image8Bit(picture);
178                     bitsForImage.MakeGrayscale();
179                     int indexB = -1;
180                     for (int y = 0; y < picture.Height; y++)
181                     {
182                         for (int x = 0; x < picture.Width; x++)
183                         {
184                             indexB++;
185                             if (indexB > result.Length) throw new Exception("Trenutni indeks " + indexB.ToString() + "
    večji od produkta širine " + picture.Width.ToString() + " X višine " + picture.Height.ToString());
186                             byte p = bitsForImage.GetByte(x, y);
187                             result[indexB] = p;
188                         }
189 
190                     }
191                 }
192                 else throw new Exception("Parameter  je null.");
193             }
194             catch(Exception eexx)
195             {
196                 MessageBox.Show("Napaka pri pridobivanje binarnega zapisa slike:\n" + eexx.Message,"Napaka",
    MessageBoxButtons.OK,MessageBoxIcon.Error);
197                 result = null;
198             }
199             return result;
200         }
201 
202         private string FromByteToStringBinary(byte[] data)
203         {
204             string result = "";
205             try
206             {
207                 // lambd izraz, ki vsakemu elementu tabele priredi string, pretvorjen v ustrezen zapis (2 - binaren, 10 -
    destiski,...),
208                 // pad left pa na levo dopolni z 0 do 8 bitov (3 - binarno = 11, pad left naredi 00000011)
209                 result = string.Join(" ", data.Select(x => Convert.ToString(x, 2).PadLeft(8, '0')));
210             }
211             catch(Exception eexx)
212             {
213                 MessageBox.Show("Napaka pri predstavitvi binarnega zapisa slike:\n" + eexx.Message,"Napaka",
    MessageBoxButtons.OK,MessageBoxIcon.Error);
214                 result = "";
215             }
216             return result;
217         }
218 
219         private  void pixelsToolStripMenuItem_Click(object sender, EventArgs e)
220         {
221             this.ShowBinaryPicture(this.originalPixels,"Originalna slika (8 bit grayscale)");
222         }
223 
224         private void imageToolStripMenuItem_Click(object sender, EventArgs e)
225         {
226             this.ShowBiggerPicture(this.original, "Originalna slika (8 bit grayscale)");
227         }
228 
229         private void ShowBiggerPicture(Bitmap pic, string ime)
230         {
231             try
232             {
233                 PictureDisplay pd = new PictureDisplay();
234                 pd.Text = ime;
235                 pd.picture = pic;
236                 pd.Show();
237             }
238             catch (Exception eexx)
239             {
240                 MessageBox.Show("Napaka pri prikazu slike:\n" + eexx.Message, "Napaka", MessageBoxButtons.OK,
```

```
MessageBoxIcon.Error);
        }
    }

    private void ShowBinaryPicture(byte[] pic, string ime)
    {
        try
        {









            PixelDisplay prikaz = new PixelDisplay();
            prikaz.Text = ime;
            prikaz.niz = FromByteToStringBinary(pic);
            if (prikaz.niz.Length > 0)
                prikaz.Show();
            else
                prikaz.Dispose();
        }
        catch (Exception eexx)
        {
            MessageBox.Show("Napaka pri prikazu pixlov: \n" + eexx.Message, "Napaka", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }

    private void toolStripButton4_Click(object sender, EventArgs e)
    {
        try
        {
            Bitmap kopija = (Bitmap)original.Clone();
            Image8Bit bitsForImage = new Image8Bit(kopija);
            bitsForImage.MakeGrayscale();
            originalPixels6bit = new byte[(kopija.Height) * (kopija.Width)];
            int bitsToShift = 2;
            int mask = 255 >> bitsToShift;
            int mask1 = 63 << bitsToShift;

            int indexB = -1;
            for (int y = 0; y < kopija.Height; y++)
            {
                for (int x = 0; x < kopija.Width; x++)
                {
                    indexB++;
                    if (indexB > originalPixels6bit.Length) throw new Exception("Trenutni indeks " + indexB.ToString()
+ " večji od produkta širine " + kopija.Width.ToString() + " X višine " + kopija.Height.ToString());

                    byte p = bitsForImage.GetByte(x, y);
                    originalPixels6bit[indexB] = (byte)((int)p >> bitsToShift);
                    bitsForImage.SetPixel(x, y,
                        (byte)((int)originalPixels6bit[indexB] << bitsToShift));
                }

            }
            original6bit = (Bitmap)kopija.Clone();
            this.pictureBox4.Image = original6bit;
            Bitmap kopija2 = (Bitmap)original6bit.Clone();
            originalPixels6bitOf8bit = this.ChangeFromBitmapToByte(kopija2);

        }
        catch (Exception eexx)
        {
            MessageBox.Show("Napaka pri kodiranju slike v 6 bit grayscale:\n" + eexx.Message, "Napaka",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void toolStripMenuItem11_Click(object sender, EventArgs e)
    {
        this.ShowBinaryPicture(this.originalPixels6bit, "Originalna slika (6 bit grayscale)");
    }

    private void pixels8bitFrom6BitToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.ShowBinaryPicture(this.originalPixels6bitOf8bit, "Originalna slika (8 bit grayscale from 6 bit)");
    }

    private void toolStripMenuItem12_Click(object sender, EventArgs e)
    {
        this.ShowBiggerPicture(this.original6bit, "Originalna slika (8 bit grayscale from 6 bit)");

    }

```

```
321         private void Form1_Load(object sender, EventArgs e)
322         {
323             // default vrednost
324             this.toolStripTextBox1.Text = this.verjetnostNapake.ToString("0.0000");
325 
326 
327 
328 
329 
330 
331 
332         }
333 
334         private void toolStripButton3_Click(object sender, EventArgs e)
335         {
336             Application.Exit();
337         }
338 
339         private void toolStripButton2_Click(object sender, EventArgs e)
340         {
341 
342                 this.SendOriginalPicture();
343         }
344 
345         private void SendOriginalPicture()
346         {
347 
348             try
349             {
350                 // najprej preverimo verjetnost napake, ki mora biti med 0 in 1
351                 if (this.toolStripTextBox1.Text.Length <= 0)
352                     throw new Exception("Verjetnost napake ni vnešena.");
353                 this.verjetnostNapake = float.Parse(this.toolStripTextBox1.Text);
354                 if (this.verjetnostNapake >= 1 || this.verjetnostNapake <= 0)
355                     throw new Exception("Verjetnost napake mora biti >0 in <1");
356                 // verjetnost napake pomnožimo z 10000, da dobimo celo število
357                 int verjetnost_napake_int = (int)(this.verjetnostNapake * 10000);
358 
359 
360 
361                 // kodira originalne 6 bitne pixle
362                 // vrendosti (od 0 do 64 na 6 bitih) priredi vrstico v matriki C
363                 // nekodirano sliko prepise
364                 this.kodiranaVSE = new byte[this.originalPixels6bit.Length][];
365                 this.napakeKodiraneVse = new byte[this.originalPixels6bit.Length][];
366                 this.nekodiranaPixels = new byte[this.originalPixels6bit.Length];
367                 for (int i = 0; i < this.originalPixels6bit.Length; i++)
368                 {
369 
370                     this.kodiranaVSE[i] = new byte[this.C[0].Length];
371                     this.napakeKodiraneVse[i] = new byte[this.C[0].Length];
372                     this.nekodiranaPixels[i] = this.originalPixels6bit[i];
373                     // 6  naključnih števil, če so manjša ali enaka od verjetnosti napake, pokvari k ti bit (1,6) na
    nekodirani
374                     // samo 6 bitov, ker nekodirano posljemo v 6 bitnih pixlih
375                     // kvarimo z XOR, kjer je maska byte z enko na mestu, ki ga zelimo pokvariti
376                     for (int k = 0; k < 6; k++)
377                     {
378                         // pokvarimo XOR s tabelvrednosti
379                         if (this.rndGen.Next(0, 10000) <= verjetnost_napake_int)
380                             this.nekodiranaPixels[i] = (byte)(this.nekodiranaPixels[i] ^ tabelVrednosti[k]);
381                     }
382                     if (this.originalPixels6bit[i] < 64)
383                     {
384                         for (int j = 0; j < this.C[0].Length; j++)
385                         {
386                             this.kodiranaVSE[i][j] = C[this.originalPixels6bit[i]][j];
387                             this.napakeKodiraneVse[i][j] = this.kodiranaVSE[i][j];
388                             // kodirane pa so 4 8 bitne besede = 32 bitov in kvarimo vseh 8 bitov posameznega byte-a
389                             // kvarimo z XOR, kjer je maska byte z enko na mestu, ki ga zelimo pokvariti
390                             for (int k = 0; k < 8; k++)
391                             {
392                                 if (this.rndGen.Next(0, 10000) <= verjetnost_napake_int)
393                                     this.napakeKodiraneVse[i][j] = (byte)(this.napakeKodiraneVse[i][j] ^ tabelVrednosti[k])
    ;
394                             }
395                         }
396                     }
397                     else
398                         throw new Exception("Napaka pri kodiranju, vrednost pixla je večja od 64 (ni 6 bitni).");
399 
400 
401                 }
402 
```

```
403             // imamo kodirane vrednosti nepokvarjene -> kodiranevse
404             // imamo kodirane vrednosti pokvarjene -> napakekodiranevse
405             // ima nekodirane pokvarjene -> nekodiranepixels
406             // zdaj dekodiramo pokvarjene kodirane
407             // dekodiranje poteka takole: kodiranevse po pixlih (vrstice) pomnozimo z matriko H^T
408             // mnozenje je xor po bytih in nato negacijo (npr. 1101 * 1010 = ~(1101 XOR 1010 = 0111) = 1000
409             // (0 so -1, 1 so 1 in če od št. 1 odštejemo št. 0 dobimo -3, kar je isto kot če bi originalno H matriko
    pomožili z vektorjem, kjer bi 0 spremenili v -1)
410             // za vsako vrstico kodirane vse dobimo en vektor dolžine 32 z vrednostmi koordinat med 0 in +-31.
411             // kjer je največja abs. vrednost, tisto je originalna kodna beseda -> v matriki C, vrednost spremenimo v
    indkes med 0, 63 in temu indeksu pripišemo 6 bitno vrednost -> dekodiran
412             // 6 bitni pixel. Če je več kot 7 napak -> bosta več kot en enaki abs. max. vrednosti -> izberemo prvo ->
    tu je moznost, da javimo napako, ki pa ni implementirana!!
413             this.kodiranaPixels = new byte[this.napakeKodiraneVse.Length];
414             for (int i = 0; i < this.napakeKodiraneVse.Length; i++)
415             {
416                 // branje po vrsticah
417                 byte[] vrstica = this.napakeKodiraneVse[i];
418                 int[] xHt = new int[H32.Length];
419                 for (int j = 0; j < H32.Length; j++)
420                 {
421
422                     xHt[j] = 0;
423                     for (int k = 0; k < vrstica.Length; k++)
424                     {
425
426                         byte produkt = (byte)((~((vrstica[k] ^ H32[j][k])) & 255));
427
428                         if (produkt > this.stevilo1in0.Length) throw new Exception("Produkt " + produkt.ToString() + "
     je vecji od " + this.stevilo1in0.Length.ToString());
429                         xHt[j] = xHt[j] + this.stevilo1in0[produkt][1] - this.stevilo1in0[produkt][0];
430                     }
431                 }
432                 // pogledamo indeks, kjer so xht najvecje vrednosti (absolutno)
433                 int indeksMax = 0;
434                 for (int j = 1; j < xHt.Length; j++)
435                 {
436                     if (Math.Abs(xHt[j]) > Math.Abs(xHt[indeksMax]))
437                         indeksMax = j;
438                 }
439                 // napaka !!!!!!
440                 // indksMax je indeks vrstice v matriki C -> hkrati je to zaradi zgornje preslikave pri kodiranju tudi
    6 bitni zapis števila, ki ga iščemo
441                 if (xHt[indeksMax] < 0) indeksMax = 31 + indeksMax;
442                 this.kodiranaPixels[i] = (byte)indeksMax;
443             }
444             // imam dekodiranje pokvarjene pixle -> kodiranepixels
445             // iz kodiranipixels in nekodiranipixels generiramo sliko in ju prikažemo v vnosni maski
446
447             Bitmap kopija = (Bitmap)original.Clone();
448             Bitmap kopija1 = (Bitmap)original.Clone();
449             Image8Bit bitsForImage = new Image8Bit(kopija);
450             Image8Bit bitsForImage1 = new Image8Bit(kopija1);
451             bitsForImage.MakeGrayscale();
452             bitsForImage1.MakeGrayscale();
453             int bitsToShift = 2;
454             int mask = 255 >> bitsToShift;
455             int mask1 = 63 << bitsToShift;
456
457             int indexB = -1;
458             for (int y = 0; y < kopija.Height; y++)
459             {
460                 for (int x = 0; x < kopija.Width; x++)
461                 {
462                     indexB++;
463                     if (indexB > kodiranaPixels.Length) throw new Exception("Trenutni indeks " + indexB.ToString() + "
    večji od produkta širine " + kopija.Width.ToString() + " X višine " + kopija.Height.ToString());

464                     bitsForImage.SetPixel(x, y,
465                         (byte)((int)kodiranaPixels[indexB] << bitsToShift));
466                     if (indexB > nekodiranaPixels.Length) throw new Exception("Trenutni indeks " + indexB.ToString() +
    " večji od produkta širine " + kopija1.Width.ToString() + " X višine " + kopija1.Height.ToString());
467                     bitsForImage1.SetPixel(x, y,
468                         (byte)((int)nekodiranaPixels[indexB] << bitsToShift));
469                 }
470
471             }
472             this.kodirana = (Bitmap)kopija.Clone();
473             this.pictureBox2.Image = this.kodirana;
474             Bitmap kopijaX = (Bitmap)kodirana.Clone();
475             kodiranaPixels6bitIf8Bit = this.ChangeFromBitmapToByte(kopijaX);
476
477             this.nekodirana = (Bitmap)kopija1.Clone();
478             this.pictureBox3.Image = this.nekodirana;
```

```
479                 Bitmap kopijaX1 = (Bitmap)nekodirana.Clone();
480                 nekodiranaPixels6bitIf8Bit = this.ChangeFromBitmapToByte(kopijaX1);
481
482
483             }
484             catch (Exception eexx)
485             {
486                 MessageBox.Show("Napaka pri prenosu in simulaciji kodiranja/dekoridranja in napak:\n" + eexx.Message,
        "Napaka", MessageBoxButtons.OK, MessageBoxIcon.Error);
487             }
488
489
490         }
491
492         private void toolStripMenuItem1_Click(object sender, EventArgs e)
493         {
494             this.ShowBinaryPicture(this.kodiranaPixels, "Kodirana slika (6 bit grayscale)");
495         }
496
497         private void toolStripMenuItem2_Click(object sender, EventArgs e)
498         {
499             this.ShowBiggerPicture(this.kodirana, "Kodirana slika (8 bit grayscale from 6 bit)");
500         }
501
502         private void toolStripMenuItem7_Click(object sender, EventArgs e)
503         {
504             this.ShowBiggerPicture(this.original6bit, "Nekodirana slika (8 bit grayscale from 6 bit)");
505         }
506
507         private void toolStripMenuItem6_Click(object sender, EventArgs e)
508         {
509             this.ShowBinaryPicture(this.nekodiranaPixels, "Nekodirana slika (6 bit grayscale)");
510         }
511
512         private void pixels8BtToolStripMenuItem_Click(object sender, EventArgs e)
513         {
514             this.ShowBinaryPicture(this.nekodiranaPixels6bitIf8Bit, "Nekodirana slika (8 bit grayscale from 6 bit)");
515         }
516
517         private void pixels8bitFrom6BitToolStripMenuItem1_Click(object sender, EventArgs e)
518         {
519             this.ShowBinaryPicture(this.kodiranaPixels6bitIf8Bit, "Kodirana slika (8 bit grayscale from 6 bit)");
520         }
521
522
523     }
524 }
525
```