

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Kodavimo teorijos A14 praktinė užduotis

Atliko: 4 kurso 5 grupės studentas
Aurelijus Banelis

Turinys

1. Programos paleidimas ir kompiliavimas.....	2
1.1. Kompiliavimas per komandinę eilutę.....	2
1.2. Paleidimas per komandinę eilutę.....	2
1.3. Java įdiegimas.....	2
2. Pradiniai tekstai.....	3
2.1. Katalogai ir failai.....	3
2.2. Klasės.....	3
3. Naudotojo sąsaja.....	4
3.1. Pagrindinės sąsajos dalys.....	4
3.2. Vektorių įvedimas.....	4
3.3. Kanalas ir triukšmai.....	5
3.4. Teksto įvedimas.....	5
3.5. Paveikslėlio įvedimas.....	6
3.6. Kiti naudotojo sąsajos elementai.....	7
4. Programiniai sprendimai.....	7
5. Testavimas.....	8
6. Nepilnai įgyvendintos dalys.....	9
7. Literatūra.....	9

1. Programos paleidimas ir kompiliavimas

Programos paleidimui yra skirtas **paleisti.bat**, o kompiliavimui **kompiliuoti.bat**. Jei bat failai nesuveiks (abiem atvejais turi pasileisti programa), galite bandyti paleisti ar kompiliuoti per komandinę eilutę. Dažniausiai pakanka į .bat failus įrašyti pilnus adresus iki kompiliatoriaus.

1.1. Kompiliavimas per komandinę eilutę

Programą galima sukompiliuoti naudojant *Java Development Kit* per komandinę eilutę:

```
javac -encoding utf8 -d dist/ src/lt/banelis/aurelijus/*.java  
src/lt/banelis/aurelijus/data/*.java src/lt/banelis/aurelijus/connectors/*.java  
jar cvfm AurelijusA14.jar MANIFEST.MF -C dist/ .
```

Jei *javac* arba *jar* nebus rasti, tai juos reikia pakeisti į pilną adresą iki JDK programų. Pavyzdžiui:

- *javac* → *c:\program files\java\jdk1.6.0_16\bin\javac.exe*
- *jar* → *c:\program files\java\jdk1.6.0_16\bin\jar.exe*

Kataloge *dist* bus laikomi sukompiliuoti *.class* failai. *MANIFEST.MF* nurodo, kurią klasę paleisti.

1.2. Paleidimas per komandinę eilutę

Programa parašyta Java kalba. Jei operacinė sistema **AurelijusA14.jar** neatpažįsta (pvz. neatidaro su *Java Platform SE binary*), programą reikia paleisti iš komandinės eilutės:

```
java -jar AurelijusA14.jar
```

Jei *java.exe* nebus rasti, tai žodį *java* reikia pakeisti į pilną adresą iki JRE programos, kažko panašaus į *C:\Program Files (x86)\Java\jre6\bin\java.exe*

1.3. Java įdiegimas

Jei kompiuteryje nepavyks rasti *java.exe*, *javac.exe* ir *jar.exe*. *Java Development Kit* ir *Java Runtime Environment*, juos reikia atsisiųsti ir įsidiegti:

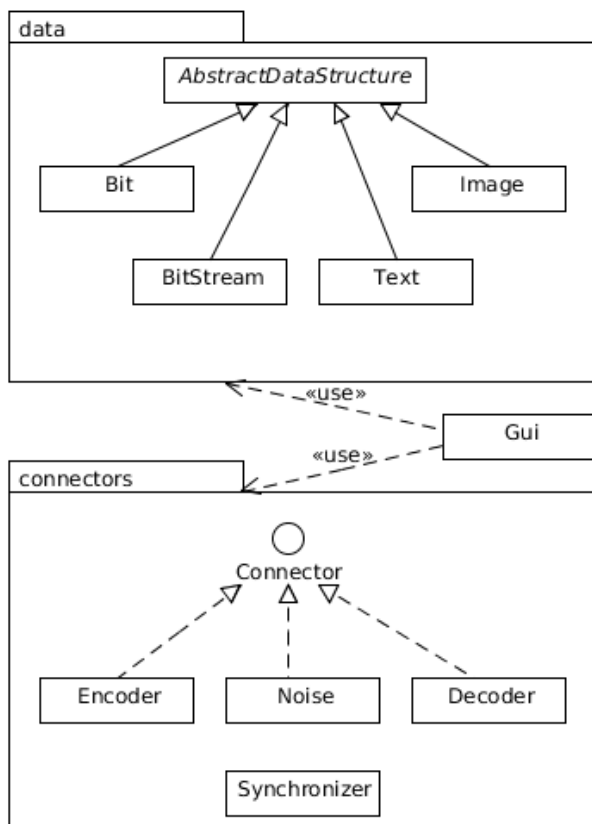
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

2. Pradiniai tekstai

2.1. Katalogai ir failai

- *src* – programos išeities tekstai (*.java failai)
 - *lt/banelis/aurelijus* – pagrindinis paketas
 - *lt/banelis/aurelijus/data* – duomenų struktūrų paketas
 - *lt/banelis/aurelijus/connectors* – duomenų transformacijų paketas
- *test* – JUnit automatiniam testavimui naudoti failai
- *dist* – tuščias katalogas, jame kompiliavimo metu bus sudėti .class failai
- *MANIFEST.MF* – konfigūracinis failas, skirtas nurodyti pagrindinę klasę
- *compile.bat* – kompiliavimui skirtas failas

2.2. Klasės



Kiekvienas .java failas atitinka klasę (pvz. Gui.java atitinka Gui), o katalogas – paketą.

Programa sudaryta iš 2 didelių paketų (*data* ir *connectors*) ir juos naudojančios **Gui** klasės. **Gui** klasė naudojama programos paleidimui ir grafinei sąsajai.

Data pakete yra įvairius duomenų tipus (įvedimą, pavaizdavimą, vertimą į dvejetainę formatą) realizuojančios klasės:

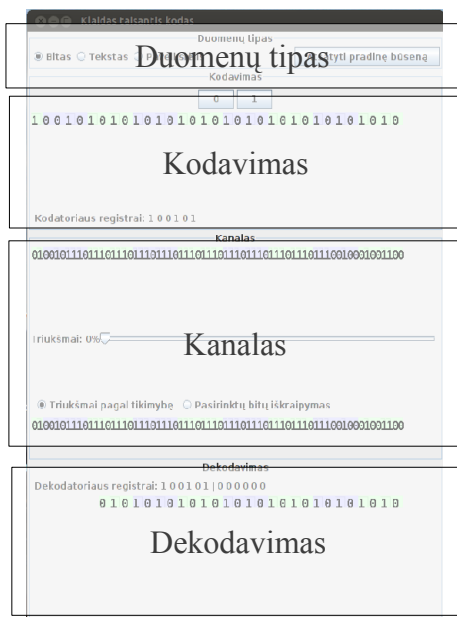
- **Bit** – bitų operacijoms ($q=2$ vektorių)
- **BitStream** – bitų sekos operacijas ir bitų iškraipymą, kai tai atlieka naudotojas.
- **Text** – su tekstu susijusias operacijas.
- **Image** – su paveikslėliu susijusias operacijas.

Connectors pakete sudėtos klasės skirtos susieti įvairius duomenų tipus:

- **Connector** – interfeisas skirtas sugrupuoti bitų sekų transformacijas atliekančias klases.
- **Encoder** – realizuoja užkodavimą .
- **Noise** – realizuoja triukšmus, priklausančius nuo tikimybės.
- **Decoder** – realizuoja dekodavimą.
- **Synchronizer** – realizuoja siuntėjo ir gavėjo duomenų palyginimą bei pranešimą apie einamą (kai operacijos reikalauja laiko) būseną.

3. Naudotojo sąsaja

3.1. Pagrindinės sąsajos dalys



Programos langas susideda iš 4 dalių:

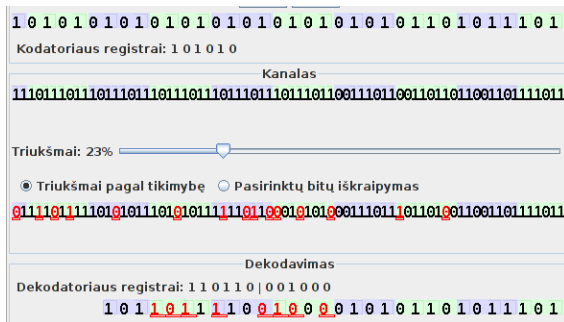
1. Duomenų tipo dalyje pasirenkama, ar bus siunčiami bitai ($q=2$ vektoriai), ar tekstas, ar paveikslėlis.
2. Kodavimo dalyje rodomi įvedimui skirti laukai ir mygtukai; koduotojo registrų reikšmės.
3. Kanale rodomi įeinantys (viršuje) ir išeinantys bitai (apačioje) bei parametrai triukšmams nuordyti.
4. Dekodavimo srityje rodomas iš kanalo dekodotas rezultatas bei dekodautojo registrai

3.2. Vektorių įvedimas

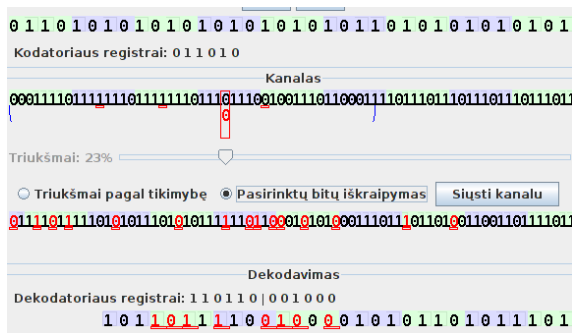


Bitus ($q=2$ vektorius) galima įvesti spaudžiant mygtukus arba naudojantis klavišais **1** ir **0**. Bus rodoma įvestų bitų istorija (bitų seka) visuose trijuose dalyse (kodavimo, kanalo ir dekodavimo). Du kartus bakstelėjus ant sekos, ji parodoma atskirame lange. Kadangi dekoduojant vėluojama 6 bitais, tai bitų seka dekodavimo skiltyje atsiranda ne iš karto.

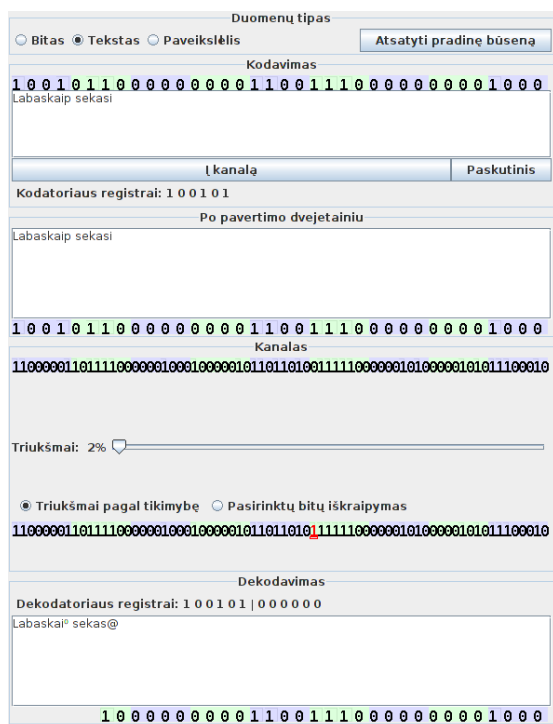
3.3. Kanalas ir triukšmai



juostos pagalba. Iškraipymai atliekami siunčiant kiekvieną naują bitą arba bitų seką (teksto, paveikslėlio atveju). Pakeista triukšmų tikimybė galioja tik naujai į kanalą atėjusiems vektoriams. Iškraipyti bitai pažymimi raudonai tiek išeinančiame kanale (užkoduotiems duomenims), tiek dekodavimo skiltyje (lyginant su pradiniais duomenimis).



3.4. Teksto įvedimas



Kanalą sudaro įeinanti (viršutinė), išeinanti (apatinė) bitų sekos ir triukšmams nustatyti skirti elementai. Kanalo triukšmams galima pasirinkti vieną iš variantų: „Triukšmai pagal tikimybę“ arba „Pasirinktų bitų iškraipymas“.

Triukšmai pagal tikimybę nustatomi slinkties

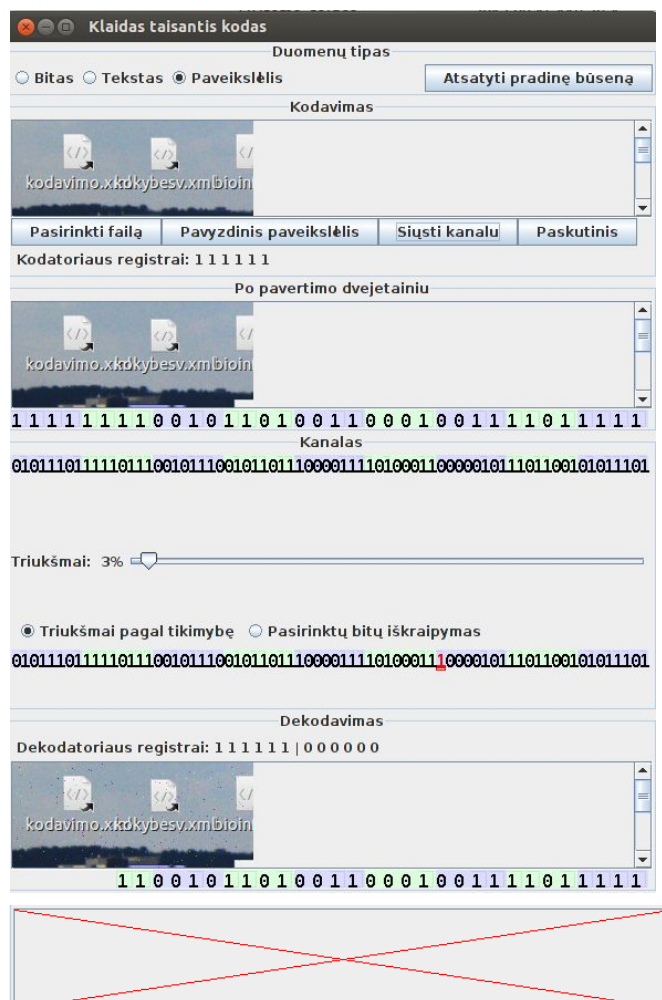
Norint koreguoti pasirinktus bitus, reikia pažymėti tam skirtą opciją, užkoduoti keletą vektorių (keisis tik į kanalą naujai įeinantis duomenų srautas) ir tada su pelės pagalba paspausti ant norimų pakeisti bitų. Kai norimi bitai pasirinkti, mygtuku „Siųsti kanalą“ bus persiųsti pakeisti bitai gavėjui.

Tekstas įvedamas viršuje pasirinkus duomenų tipą „Tekstas“. Įvedus norimą tekstą, reikia paspausti mygtuką „Į kanalą“. Tekstas bus paverstas į dvejetainį kodą, bei atverstas atgal ir parodytas skiltyje „Po pavertimo dvejetainiu“. Šioje skiltyje bus rodomas ne tik paskutinis tekstas, bet ir anksčiau siūsta informacija (taip patogiau palyginti dekodavimą, nes duomenų persiuntimas vėluoja). Po teksto laukeliu bus rodomas į dvejetainį srautą paverstas teksto variantas.

Dekodavimo dalyje bus rodomas dekodotas tekstas. Kanalo triukšmų pridėjimas analogiškas darbui su bitais. Klaidos žymimos tik bitų sekoje

(tekste ir taip lengvai pastebimos, o paryškimo realizavimas pakankamai sudėtingas). Pabaigus paskutinį pranešimą reikia paspausti „Paskutinis“, kad dekodavime iš registrų išeitų paskutinės reikšmės.

3.5. Paveikslėlio įvedimas



Paprastumo dėlei yra mygtukas „Pavyzdinis paveikslėlis“, kuris nupaišo paprastą paveikslėlį. Tiek įkrautą, tiek pavyzdinį paveikslėlį galima „Siųsti kanalu“ paspaudus tokio pat pavadinimo mygtuką.

Dėl kodavimo vėlavimo, paskutiniai 6 bitai nėra persiunčiami, bet paveikslėlio vaizdui jie daro labai mažą įtaką. Norint nuskaityti likusius iš dekodavimo registrų, reikia paspausti „Paskutinis“. Persiuntimas yra suprogramuotas taip, kad jei siunčiama didelė bitų seka, būtų atmesti 6 bitai ir iš karto skaičiuojamas paveikslėlio dydis (laikant senus paveikslėlius bitų pavidalu atmintyje labai greitai susinaudoja darbinė – RAM – kompiuterio atmintis).

Klaidos, kaip ir teksto atveju, pažymimos tik bitų sekoje (kairiniai bitai yra paskutiniai išsiųsti).

Paveikslėlį persiųsti galima prie „Duomenų tipas“ pasirinkus „Paveikslėlis“. Testuota su .jpg, .png ir .bmp failais. Dideliems paveikslėliams programa nėra optimizuota ir kodavimas gali užtrukti nemažai laiko ir suvartoti daug atminties. Bandytas 200x200 taškų paveikslėlio persiuntimas užtruko maždaug 10 s. 100x100 paveikslėlis yra optimaliausias. Dekodavimui yra uždėtas 400x400 taškų limitas, kad programa nenulaužtų kompiuterio, kai neteisingai dekoduojamas dydis ir paveikslėlio plotis arba ilgis tampa kelias dešimt tūkstančių.

Paveikslėlį įkrauti galima paspaudus ant „Pasirinkti failą“. Netinkamiems ar nenuskaitomiems failams paveikslėlio vietoje bus rodomas raudonas X-as.

3.6. Kiti naudotojo sąsajos elementai

Koduojant ir dekoduojant taip pat rodomos ir kodatoriaus ir dekodatoriaus registrų reikšmės (eilės tvarka atitinka literatūroje pateiktos būsenų diagramos tvarką, kuri taip pat atitinka išsaugojimo tvarką).

Išvalyti kanalus ir buvusias bitų sekas galima pasinaudojus mygtuku „Atstatyti pradinę būseną“.

4. Programiniai sprendimai

Kadangi užduotyje nurodyta, kad galima įvesti tik $q=2$ vektorius, todėl įvedamo formato tikrinimas buvo pakeistas tiesiog dviem mygtukais: „1“ ir „0“. Su pele didesniems vektoriams mygtukus nebūtų patogu spaudžioti, tačiau suprogramavus sparčiuosius klavišus (skaičių klavišai „1“ ir „0“ atitinka mygtukus) atitiko vektorių tikrinimą (kitų skaičių klavišai tiesiog nieko nedaro) ir sutaupė papildomą veiksmą (nuspausti mygtuką, kad vektorių seka būtų išsiųsta). Įvedant bitus po vieną, labai gerai matosi, kaip pereina kodavimo ir dekodavimo registrai, taip įsitikinant algoritmo korektiškumu ir bendru veikimo principu.

Žmogui palyginti bitų seką yra pakankamai sudėtinga, ypač, kai dekodavimas vėluoja. Todėl bitų grupavimas (skirtingos spalvos fonu) ir grafinis sulygiavimas (gavėjo bitai rodomi 6 pozicijom toliau nuo kairio krašto) buvo suprogramuoti jau pačioje pradžioje. Tai leido daug lengviau aptikti įgyvendinimo klaidas arba įsitikinti korektišku veikimu. Vėliau buvo sulygiuotos ir originali ir užkoduota seka. 2 užkoduoti bitai atitiko 1 originalų, todėl buvo pakeistas jų plotis.

Daug svarstymų sukėlė klaidų pažymėjimo ir kanalo redagavimo reikalavimas. Redagavimas dažniausiai įgyvendinamas tekstiniu laukeliu, tačiau klaidų pažymėjimo įgyvendinimas yra daug sudėtingesnis. Klaidas bitų sekose pamatyti akimi yra sudėtinga, o paprastas parašymas, kad klaidos buvo n -tojoje ar k -tojoje pozicijoje būtų neefektyvus, nes klaidų kiekis gali būti ir 100%. Todėl buvo pasirinkta apsirašyti primitivų bitų sekos piešimą (klaidų paryškinimui) ir pagal pelės poziciją apskaičiuojamą duomenų struktūros poziciją (bitų pakeitimo interaktyvumą).

Įgyvendinus bito kodavimą ir dekodavimą, programą reikėjo pritaikyti ir tekstui bei paveikslėliui. Pagal objektinę paradigmą, funkcijos skirtos darbui objektui turėtų būti aprašytos prie pačio objekto. Todėl buvo nuspręsta, kad kiekvienas duomenų tipas ar duomenų transformacijos (kodavimas, dekodavimas ir triukšmų pridėjimas realizacijos požiūriu yra duomenų transformacijos) kartu būtų ir grafiniai elementai (realizuoti kaip JPanel vaikai). Tokiu

atveju, kodavimo klasė gali pati atsinaujinti registrų pavaizdavimą, teksto klasė gali realizuoti teksto įvedimą ir t. t.

Tekstas ir paveikslėlis buvo persiuntimui paverčiami į bitų seką. Kadangi tekstui vėluojantys bitai turi įtakos paskutiniam simboliui, todėl dekodavimui saugoma visa persiuntimo istorija. Paveikslėlis koduojamas 32 bitų blokais, nes toks yra Java kalbos sveiko bitų skaičiaus dydis. Pirmi 2 žodžiai (blokai po 32 bitus) užkoduoja ilgį ir plotį, o likę spalvas einant per kiekvieną stulpelį ir eilutę. Vėluojantys bitai paveikslėlio vaizdui daro labai mažą įtaką (vizualiai beveik nesiskiria), tačiau siunčiant naują paketą buvusio paveikslėlio pabaiga gali būti interpretuota, kaip naujo paveikslėlio dydį nusakantys žodžiai. Todėl gavus pranešimą, patikrinama, ar dalis paveikslėlio jau buvo siųsta, ar dydį nusakantys bitai yra sekos pradžioje.

Dėl galimybės keisti kanalo turinį, jis sudaryta iš 2 dalių: įeinančios ir išeinančios. Kai naudojami triukšmai pagal tikimybę, duomenys iš vienos funkcijos į kitą perduodami vienu paspaudimu. Dvi duomenų struktūras sujungia duomenų transformaciją atliekanti klasė, pavyzdžiui, siuntėją ir kanalo įėjimą sujungia kodavimo klasė. Taip susidaro grandinė nuo siuntėjo iki gavėjo. Kai klaidos įvedamos naudotojo, kanalo įėjimas ir išėjimas nebūna sujungtas tol, kol naudotojas nepaspaudžia mygtuko „Siųsti kanalą“. Po šio veiksmo, į kanalą atėjusi nauja ir galbūt pakeista informacija patenka į kanalo išėjimą atitinkantį objektą ir taip grandine iki gavėjo. Dėl duomenų vėlavimo naudotojui yra svarbu, atsiminti ir buvusius 6 bitus, todėl kiekviena duomenų struktūra (t. y. jų tėvinė klasė) saugo ir buvusias reikšmes. Duomenų srautas nuo pirmo iki paskutinio bito, leidžia grafiškai pamatyti, ar klaidos atsiranda pagal nustatytą tikimybę.

5. Testavimas

Automatiniai testai kodavimui ir dekodavimui bei jų efektyvumui nebuvo parašyti, nes jau programos kūrimo pradžioje didelis dėmesys buvo skirtas grafinei naudotojo sąsajai ir testavimui naudojantis ja. Empiriniais bandymais buvo pastebėta, kad su pavienėmis klaidomis (triukšmų tikimybė 4%) kodavimas visas klaidas ištaisydavo. Taip pat būdavo ištaisomos ir 2 gretimos klaidos, jei aplinkui kitų klaidų nebūdavo (pvz. 18 bitų atstumu). Klaidas nepavykdavo ištaisyti, kai pasitaikydavo 3 ir daugiau šalia esančių klaidų (pvz, 16 bitų gabale).

Tiek tekste, tiek paveikslėliuose klaidos turėdavo didelę įtaką. Jau su 5% triukšmų tikimybe būdavo sugadinama kas antra arba kas trečia raidė, todėl atspėti prasmę tapdavo sudėtinga. Paveikslėliui didžiausią įtaką turėdavo klaidos pločio ir ilgio aprašyme – tada dydžiai tapdavo milijoniniai arba neigiami, todėl nebūdavo įmanoma atkurti paveikslėlio vizualiai panašaus į buvusį. Spalvų klaidos ne visada būdavo tokios ryškios, nes klaida galėdavo atsirasti ir mažus ir

didelius skaičius atitinkančiuose bituose. Bitų arba teksto atveju, persiunčiama santykinai mažai duomenų, todėl dekodavimas vyksta akimirksniu. Paveikslėliams reikia daug daugiau duomenų persiųsti ir apdoroti. 100x100 paveikslėlį persiųsdavo maždaug per 4 sekundes, kai 200x200 užtrukdavo 10 ir daugiau sekundžių (dėl šios priežasties kodavimo metu lango antraštėje rodoma atlikto darbo dalis).

Skaičių, teksto ir paveikslėlio vertimo į bitų srautą ir atvirkščiai testavimui buvo pasirašyti JUnit testai (*kodas/test* kataloge). Vertimą į ir iš dvejetainio kodo galima patikrinti per grafinę sąsają (pvz. nustačius triukšmų lygį 0 arba pagal skiltį „Po pavertimo dvejetainiu“), o modulių testavimas be programavimo aplinkos yra sudėtingas. Todėl šie automatinių testų failai paliekami tik kaip iliustracija lengvesniam visos programos veikimui suvokti, o ne algoritmų teisingumui ištestuoti.

6. Nepilnai įgyvendintos dalys

- **Nerodomas klaidų skaičius**, tačiau pačios klaidos paryškinamos.
- **Persiuntimo būseną kartais „mirkčioja“**: Linux operacinėje sistemoje lango antraštėje procentai keičiasi tolygiai, o Windows – kažkodėl mirkčioja įvairios būsenos.

7. Literatūra

Daugiausia remtasi prie užduoties pateikta literatūra [Ber84, §15.61–15.63, p. 388–391]¹.

Sąsukos kodų bendriesiems principams suvokti naudota paskaitų medžiaga [Ske08]² bei keli pavyzdžiai iš interneto [Han04]³ ir [Zol07]⁴

1 E.R. Berlekamp. *Algebraic Coding Theory*. Revised 1984 Edition. Aegean Park Press, Laguna Hills, CA, 1984.

2 <http://uosis.mif.vu.lt/~skersys/11r/ktkt/KTKT1-2.pdf>

3 Yungshiang S. Han. *Introduction to Binary Convolutional Codes*. National Taipei University, Taiwan, 2004.

4 V.V.Zolotarev, R.R.Nazirov, I.V.Chulkov. *The quick almost optimal multithreshold decoders for noisy gaussian channels*. RSCGSO International Conference ESA in Moscow, June, 2007.