VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Programų sistemų 3 kursas

Kodavimo teorijos dalyko

A11 užduoties

aprašas - dokumentacija

Atliko Evaldas Visockas

Vilnius – 2021

Turinys

Reikalavimų įgyvendinimas

Programoje įgyvendinti visi scenarijai. Visuose scenarijuose įgyvendinti visi reikalavimai. Kodas komentuotas paprastais bei „Javadoc“ stiliaus komentarais. Paaiškinta, už ką kiekviena klasė atsakinga, ir ką kiekvienas metodas atlieka, kokių parametrų įvesties reikalauja, bei ką apskaičiuoja. Atlikti eksperimentai. Papildomai pateikiami įrašai apie užduoties atlikimui skirtų valandų skaičių (pateikiama išsamiau: minutėmis, kadangi laikas buvo stebimas tiksliau), tačiau ataskaita yra kitokio formato, ir paaiškinimas kaip pasileisti programą (tačiau nėra paaiškinimo (nei vykdomojo failo), kaip ją sukompiliuoti).

Bibliotekų panaudojimas

Programai parašyti buvo naudojamos bibliotekos, kurių gavimui nereikalingas papildomas atsisiuntimas (visos iš „openjdk-16“).

Iš šių bibliotekų rinkinio pravertė:

- „java.util.Arrays“, kad galima būtų panaudoti funkciją „Arrays.copyOf“ siekiant sukurti naują vektorių, kuris išeis iš nepatikimo kanalo ir panašiems atvejams, kur reikia naujo masyvo.
- „java.util.Random“ nepatikimo kanalo klaidų generavimui bei atsitiktinės matricos (jei vartotojas pageidauja) sukūrimui.
- „java.util.HashMap“, kad galima būtų lengvai surišti sindromą ir jo lyderio svorį. Buvo galimas ir kitas variantas: nenaudoti „HashMap“ ir sukurti du masyvus, kur susiję elementai turės vienodas pozicijas. Tačiau perstatant elementus, tektų dirbtų su abejais masyvais ir padidėja tikimybė padaryti klaidą pametant sąryšį.

- „java.util.Scanner“ vartotojo tekstiniai įvesčiai gauti.
- „java.math.BigInteger“ žmogui perskaitomą tekstą paversti bitų eilute. Pavyzdžiui, bitų eilutė bus gaunama taip:
  ```
  new BigInteger(textData.toString().getBytes()).toString( radix: 2);
  ```
  O iš bitų eilutės tekstų eilutę gausime, panaudoję:
  ```
  new String(new BigInteger(corruptedBinary,  radix: 2).toByteArray());
  ```
- „java.awt.*“, kad pavyzdžiui būtų galima panaudoti „Desktop“ klasę, kuri, jeigu bus suteiktos privilegijos, atidarys paveiksliukus monitoriaus ekrane. Taip pat joje pasiekiama „Color“ klasė, į kurią lengvai sudedama rgb spalva.
- Klasės, kurių pavadinime yra žodis „image“ naudojamos paveiksliuko atidarymui programoje, siekiant juos apdoroti: gauti jų ilgį, plotį, ištraukti iš jų spalvas, kurios bus siunčiamos nepatikimu kanalu.
- „java.io.File“ naudojama paveiksliukų failų sukūrimui ir išsaugojimui laikiname „Temp“ aplanke.
- Paketą „Utils“ galima būtų vadinti mano sukurtų bibliotekų paketu, nes jame esančios klasės „CodeMath“ ir „TextUtils“ suteikia daugybę programos veikimui būtinų funkcijų.
- Taip pat buvo panaudotos „Unit“ testams skirtos bibliotekos. Tačiau jos naudojamos aplanke „tests“, o ne programoje.

Užduočiai skirti laiko pasiskirstymas

Pateikiu mano užfiksuotą darbotvarkę:

Nr.	Data	Laiko intervalas	Veiklos pavadinimas	Visas laikas	Pertraukos laikas	Darbo laikas
1	2021-09-13	17:47-18:26	Pradžia.	39 min.	10 min.	29 min.
2	2021-09-15	17:25-18:12	Matematikos kūrimas (sukurta vektoriaus ir matricos daugyba bei pradėtas kurti kontrolinės matricos radimas iš generuojančios matricos).	47 min.	3 min.	44 min.
3	2021-09-18	11:15-13:26	medžiagos skaitymas ir kartojimas.	2h 11 min.		2h 11 min.
4		13:56-15:59	pradinės tekstinės įvesties sąsajos kūrimas.	2 h 3 min.	5 min.	1h 58 min.
5		15:59-16:12	pradinės tekstinės įvesties sąsajos testavimas.	13 min.		13 min.
6		16:29-17:55	pirmojo scenarijaus užkodavimo programavimas.	1 h 26 min.	15 min.	1h 11 min.

7		17:55-17:59	pirmojo scenarijaus užkodavimo testavimas.	4 min.		4 min.
8		18:04-18:14	kontrolinės matricos radimas.	10 min.		10 min.
9		19:20-19:38	matricų daugybos bendru atveju užrašymas (vėliau tapo nebereikalingas).	18 min.		18 min.
10		19:38-19:44	matricų daugybos bendru atveju testavimas.	6 min.		6 min.
11		19:47-22:04	sindromų ir lyderių svorių susiejimas.	2h 17 min.	15 min.	2h 2 min.
12	2021-09-19	10:38-11:12	sindromų ir lyderių svorių susiejimo užbaigimas. Klaidų taisymas.	34 min.		34 min.
13		11:12-11:43	sindromų ir lyderių svorių susiejimo testavimas.	31 min.		31 min.
14		11:54-12:07	kanalo sukūrimas ir testavimas.	13 min.		13 min.
15		12:11-12:54	dekodavimas.	43 min.		43 min.
16		12:54-13:11	dekodavimo testavimas.	17 min.		17 min.
17		13:18-13:46	klaidų skaičiaus išvedimas ir jų pozicijų radimas, pirmojo scenarijaus išplėstinis testavimas.	28 min.		28 min.
18		14:00-14:09	papildomas I scenarijaus testavimas.	9 min.		9 min.
19		14:09-14:20	antro scenarijaus teksto įvedimas ir testavimas.	11 min.		11 min.
20		14:39-15:52	antro scenarijaus teksto siuntimo kanalu be kodavimo programavimas ir testavimas.	1h 13 min.	5 min.	1h 8 min.
21		16:04-16:34	antro scenarijaus teksto siuntimas kanalu, jį užkoduojant. Antrojo scenarijaus testavimas.	30 min.		30 min.
22		16:45-18:25	klaidų aptikimas antrame scenarijuje ir jų ištaisymas.	40 min.		40 min.
23		19:23-19:41	skaitymas apie „bmp“ formatą.	18 min.	5 min.	13 min.

24		19:55-23:59	trečiojo scenarijaus programavimas (neužkoduoto bitų srauto kanalu siuntimas).	4 h 4 min.		4h 4 min.
25	2021-09-20	12:22-13:14	trečiojo scenarijaus programavimo užbaigimas (užkoduoto bitų srauto kanalu siuntimas). Testavimas.	52 min.		52 min.
26		13:30-13:35	vartotojo įvesties, kai jis pasirenka scenarijų programavimas.	5 min.		5 min.
27	2021-09-21	11:48-12:00	„IntelliJ“ programos optimizavimo pasiūlymų priėmimas (pavyzdžiui String sujungimas padarytas su „StringBuilder“, priekinių nulių pridėjimas su „repeat“ funkcija, kai kurie for'ai pakeisti į „enhanced“ for'us).	12 min.		12 min.
28		12:04-12:42	sindromų lentelės sudarymo laiko testavimas.	38 min.		38 min.
29		12:42-14:14	teksto ir paveiksliukų siuntimo kanalu pavyzdžių rinkimas (vedžiau ir fotografavau pavyzdžius).	1h 32 min.	25 min.	1h 7 min.
30		15:44-18:16	paveiksliukų siuntimo kanalu pavyzdžių rinkimo baigimas. Papildomų „StringBuilder“ įvedimas, siekiant optimizuoti programos gebėjimą greitai nuskaityti spalvas.	2h 32 min.	1h 30 min.	1h 2 min.
31	2021-09-22	07:44-08:20	trečiojo scenarijaus laiko (įskaitant išskirtinai išreikštą laiką, kurį užtrunka programa užkoduodama, siųsdama kanalu, dekoduodama paveiksliuką) testavimas.	36 min.		36 min.
32	2021-09-26	10:33-12:26	„.jar“ failo sukūrimas ir aiškinimasis, kaip konsolėje spausdinti utf-8 formatu.	1 h 53 min.		1 h 53 min.

33		12:34-13:23	programos paleidimo aprašymas dokumentacijoje.	49 min.		49 min.
34		13:24-15:25	kodo dokumentacijos rašymas (komentarais aprašomos funkcijos).	1 h 1 min.	15 min.	46 min.
35		15:54-16:38	bibliotekų panaudojimo aprašymas dokumentacijoje.	44 min.		44 min.
36		16:49-17:18	užduočiai skirto laiko ataskaitos apibendrinimo paruošimas.	29 min.		29 min.
37		17:37-18:29	įgyvendintų programinių sprendimų aprašymas.	52 min.	5 min.	47 min.
38	2021-09-27	10:58-11:52	įgyvendintų programinių sprendimų aprašymo užbaigimas.	54 min.		54 min.
39		11:59-13:08	vartotojo sąsajos aprašymas.	1h 9 min.	5 min.	1h 4 min.
40		13:20-14:02	Užduočiai skirto laiko ataskaitos patobulinimas (formatavimas).	42 min.	5 min.	37 min.
41		17:27-17:45	Užduočiai skirto laiko ataskaitos formatavimo užbaigimas.	18 min.		18 min.
42		17:46-17:55	Panaudotos literatūros surašymas.	9 min.		9 min.
43		18:07-18:30	Eksperimentų aprašymas.	23 min.		23 min.
44		19:22-22:07	Eksperimentų aprašymo užbaigimas.	2 h 35 min.	20 min.	2 h 15 min.
45		22:07-10:30	Užduočiai skirto laiko ataskaitos galutinis paruošimas. Darbo užbaigimas.	23 min.		23 min.
Viso				38h 53 min.	3h 43 min.	35h 10 min.

Užduoties reikalavimų skaitymas bei kodavimo aiškinimasis: iki 6h (įskaitant A11 užduoties aiškinamąjį įrašą, kuris truko apie 1h 48 min.).

Projektavimą dariau galvoje, kodo rašymo arba konkretaus laiko ataskaitoje užregistruoto punkto vykdymo metu. Kadangi atskiram projektavimui laiko neskyriau, jis nebuvo užregistruotas atskirai.

Programos paleidimas

Programos vykdomasis failas yra aplanke „programos paleidimas“ pavadinimu „A11EvaldasVisockas.jar“. Programos paleidimui palengvinti pridėtas papildomas „paleisti.bat“ failas:

```
1  chcp 65001
2  echo Įvedamas UTF-8 formatas.
3  java -Dfile.encoding=UTF-8 -jar A11EvaldasVisockas.jar
4  PAUSE
```

paleidimo parametrai 1

Pirma eilutė nurodo konsolei naudoti utf-8 formatą. 3 Eilutėje paleidžiamas failas su papildomu nurodymu naudoti utf-8 formatą. Nors programuota angliškai, tekstas spausdinamas lietuviškai, tad šio formato reikia.

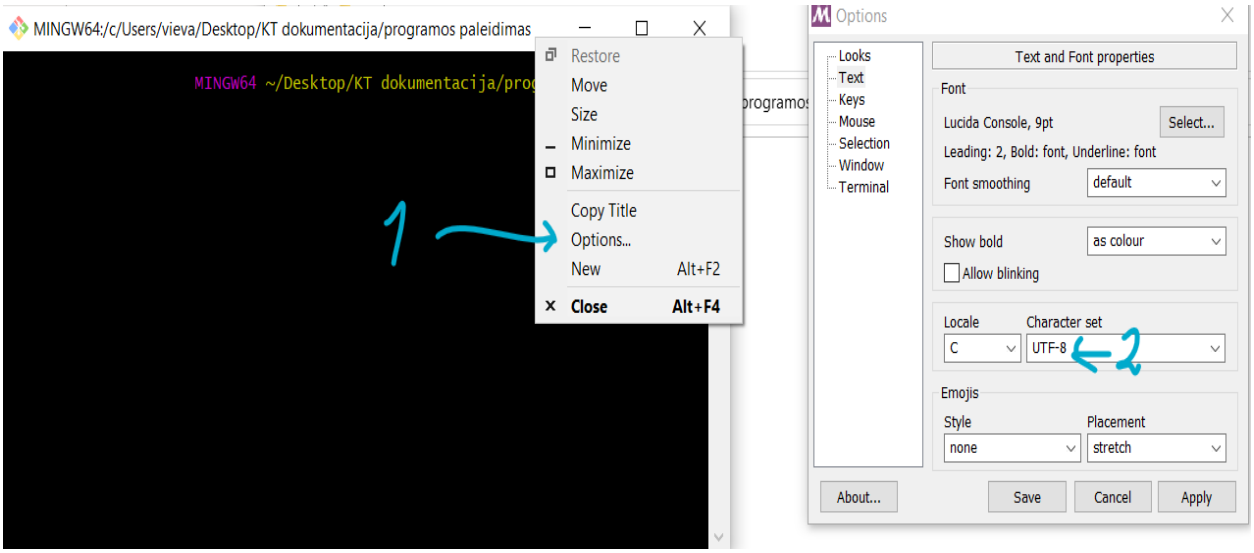
Vis dėlto, nors programos išvedamas tekstas spausdinamas lietuviškai, regis, įvedant lietuvišką tekstą, komandinė eilutė jo neatpažįsta:

```
Prašome įvesti tekstą. Tekstas gali būti sudarytas iš vienos daugiau eilučių. Kai norėsite baigti, spauskite „Enter“:
Bėgo kiškis ir pamatė lapiną. Lapinas sako:
-Labas, kiški!

Jūsų įvestą tekstą siunčiame neužkoduotą kanalu, kuriame tikimybė padaryti klaidą lygi 0.1.
Kanalo gale pasirodė štai toks pranešimas: J #▯ {▯Kcr▯Yr p`o▯d▯"hapi, >▯▯Axin▯s sakm:▯-Gab▯sl"▯▯ k)▯

Jūsų įvestą tekstą užkoduojame ir siunčiame tuo pačiu kanalu.
Kanalo gale pasirodė štai toks pranešimas: B go ki kis iv pamat  lapin . Lapinas sako:
-Labas, ki ki!
```

Todėl rekomenduojama įvedinėti tik lotyniškos abėcėlės raides.

Jei vis dėlto norite įvesti lietuvišką tekstą, siūlomi du sprendimai:

1) Jei turite, galite panaudoti „Git Bash Here“ komandinę eilutę. Nustatykite tokius parametrus:



Tada nusikopijuokite paleidimo parametrus iš failo „paleisti.bat“ (žr. paleidimo parametrai 1). Štai koks rezultatas:

```
Prašome įvesti tekstą. Tekstas gali būti sudarytas iš vienos daugiau eilučių. Ka
i norėsite baigti, spauskite „Enter“:
Bėgo kiškis ir sutiko lapiną. Lapinas sako:
- Labas, kiški!

Jūsų įvestą tekstą siunčiame neužkoduotą kanalu, kuriame tikimybė padaryti klaid
ą lygi 0.1.
Kanalo gale pasirodė štai toks pranešimas:
♦♦wo ciśkis`ap$w}vikk!la♦iř.!l!pincs♦{aJo:♦!biras<Okiűki)♦
Jūsų įvestą tekstą užkoduojame ir siunčiame tuo pačiu kanalu.
Kanalo gale pasirodė štai toks pranešimas: Bėgo kiškis ir sutiko lapiną. Lapinas
 sako:
- Labas, kiški!
```

Pastaba: nukopijavus paleidimo parametrus ir pirmą kartą įvedus kodo dimensijos reikšmę, tikėtina, kad ji nebus nuskaityta. Tokiu atveju prašome pakartoti įvedimą.

2) Jei turite, galite pasinaudoti „IntelliJ IDEA“. Šia programavimo aplinka buvo sukurta programa. Programos „.java“ failai patalpinti aplanke „programos kodo tekstai“.

Programos teksto failų („source-code“) aprašymas

Programos teksto failai yra aplanke „programos kodo tekstai/src“.

Atsidarius aplanką matomi šie aplankai (o juose failai):

- channels
  - Channel.java – imituoja duomenų siuntimą nepatikimu kanalu, kuriuo iškraipomi siunčiami duomenis.
- data
  - CodeData.java – laikomi pradiniai vartotojo įvesti kodo parametrai, bei pagal juos išvesti duomenys (kontrolinė matrica, automatiškai sugeneruota generuojanti

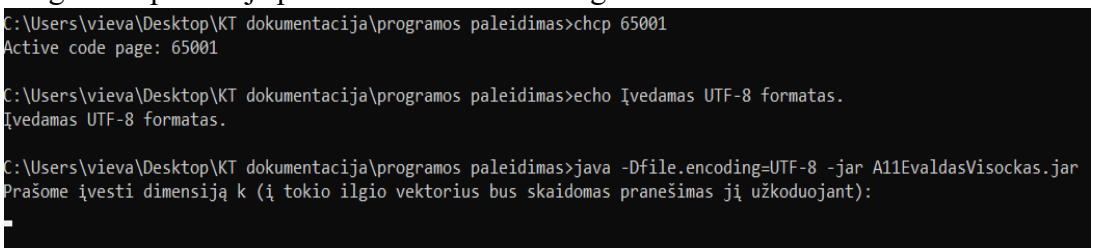
matrica). Klasė taip pat sugeba apskaičiuoti vektoriaus sindromą. Sindromų lentelė laikoma atskiroje klasėje.
    - SyndromeTable.java – sugeneruojama sindromų ir vektorių, iš kurių buvo gauti šie sindromai, svorių sąsaja. Naudojamas paieškos į gylį algoritmas.
- main
    - Input.java – priima pradinę vartotojo įvestį ir iš jos sugeneruoja „CodeData“ objektą.
    - Main.java – pradeda programos vykdymą ir priima vartotojo pasirinktą scenarijų. Programai pradėti argumentai nereikalingi.
- scenarios
    - FirstScenario.java – vykdo pirmąjį scenarijų.
    - SecondScenario.java – vykdo antrąjį scenarijų.
    - ThirdScenario.java – vykdo trečiąjį scenarijų.

- utils
    - CodeMath.java – atliekama didžioji dalis kodavimui reikalingų kodavimo operacijų.
    - TextUtils.java – teksto išvedimui ar įvesčiai formatuoti ir (ar) taisyti.

Vartotojo sąsajos aprašymas

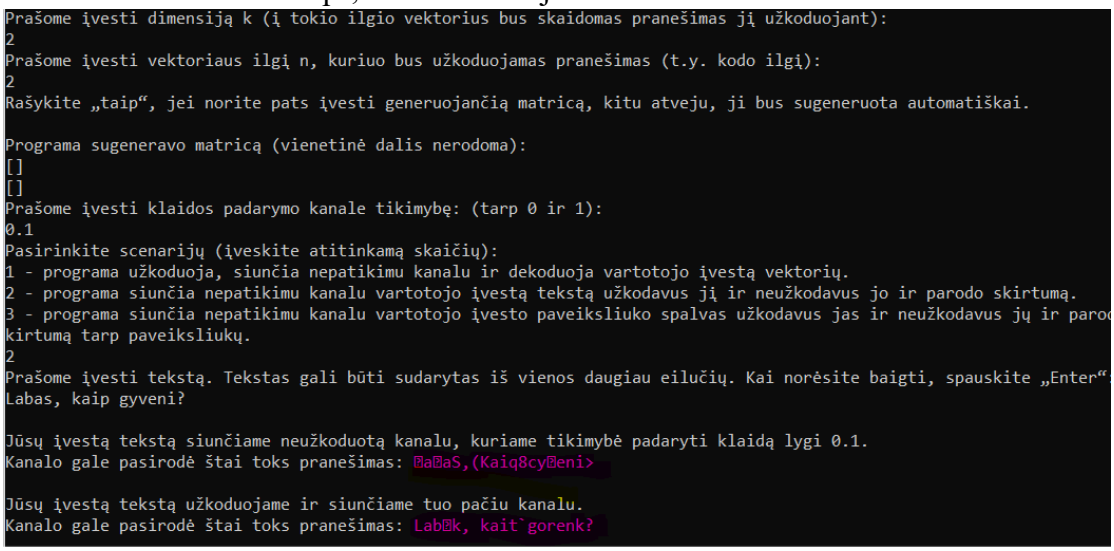
Programą buvo stengtasi daryti taip, kad ja naudojantis kiltų kuo mažiau neaiškumų. Programos vykdymo metu, į ekraną išvedami tekstai su paaiškinimais. Vis dėlto, kai kurie paaiškinimai gali būti per mažai išsamūs. Todėl šiame skyrelyje pateikiame keletą programos naudojimo pavyzdžių:

- Pradinių kodo parametrų įvedimas:
  Programos pradžioje pateikiamas štai toks langas:



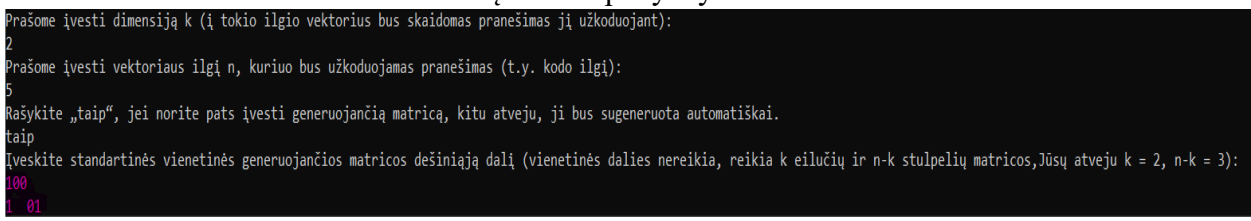
  Programa prašo įvesti skaičių k. K ilgio vektoriais bus skaidoma vartotojo įvesta informacija tolimesnėje programos eigoje (pavyzdžiui, tekstas pirmiausia bus išskaidytas į bitų srautą, o srautas padalintas į k ilgio vektorius). Skaičius k, turi būti sveikasis skaičius, didesnis už 0.
  Toliau programa prašo įvesti skaičių n. Tokio ilgio vektoriais programa koduos informaciją. Skaičius n turi būti sveikasis skaičius nemažesnis už prieš tai įvestą skaičių k. Programa nedraudžia įvesti skaičiaus n, lygaus skaičiui k. Tačiau tokiu

atveju, kodavimas netenka prasmės. Pastebėsime, kad jei k = n, iš kanalo išėjusi žinutė dažniausiai nesutaps, nes informacija nėra siunčiama vienu metu:



Įvedus skaičių n, programa paklaus, ar vartotojas nori pats įvesti generuojančią matricą. Jei nori, vartotojas turi užrašyti „taip“ (galima ir didžiosiomis), jei nenori, gali tiesiog spausti „Enter“.

Jei vartotojas užrašė „taip“, programa paprašys įvesti matricą, susidedančią iš k eilučių ir (n-k) stulpelių. Vienetinės dalies įvedinėti nereikia. Programa dirba tik su vienetinėmis matricomis. Matricos įvedimo pavyzdys:



Taigi, patogaus skaitymo dėlei, galite įvesti kiek norite tarpų. Tačiau tarpas ir bitų simboliai yra vienintelis priimtinas užrašymo būdas (negalima naudoti „tab“, „-0“ ir panašiai). Baigę užrašyti vieną matricos eilutę, pereisite prie kitos, paspausdami „Enter“.

Įvedus matricą, programa paprašys įvesti klaidos tikimybę. Klaidos tikimybė, tai realusis skaičius, didesnis už 0 ir mažesnis už 1.

Įvedus šiuos kodo parametrus, programa pradės generuoti sindromų lentelę. Tai gali užtrukti. Kad nereikėtų ilgai laukti, rekomenduojama parinkti skaičių n, nedidesnį nei 22.

Sugeneravus sindromų lentelę, programa leis pasirinkti vieną iš trijų scenarijų.

- Scenarijų pasirinkimas:
  Norėdami pasirinkti scenarijų, įveskite vieną iš skaičių, atitinkantį Jūsų norimą scenarijų. Scenarijų naudojimosi aprašymai:
  - Pirmasis scenarijus:
    Programa paprašys jūsų įvesti vektorių ilgio k, kurį nurodėte programos pradžioje. Įvedus vektorių, programa jį užkoduoja vektoriumi ilgio n (kurį

taip pat nurodėte programos pradžioje) ir parodo. Užkoduotas vektorius siunčiamas nepatikimu kanalu, kurio iškraipymo tikimybė nurodoma (Jūsų įvesta). Atvaizduojamas kanalo gale pasirodęs vektorius ir akcentuojama, kiek klaidų buvo padaryta siunčiant jį kanalu ir kuriose pozicijose (pradinė pozicija yra vienetas). Prieš programai dekoduojant vektorių, vartotojas gali įvesti naują, tokiu atveju jis ir bus dekoduojamas. Jei vartotojui tinka iš kanalo išėjus vektorius, tereikia spustelti „Enter“. Vartotojui įvedus arba pasirinkus iš kanalo išėjusį vektorių, kaip tinkamą, programa dekoduoja vektorių. Jeigu kanale (arba vartotojo įvestame naujame vektoriuje) yra lygiai ar mažiau klaidų, nei dekodavimo algoritmas gali ištaisyti, programa išves pirmojo scenarijaus pradžioje vartotojo įvestą vektorių. Jei klaidų yra daugiau, klaidų ištaisymas nėra garantuojamas ir gali būti, kad algoritmas betaisydamas paliks daugiau klaidų, negu įvyko kanale.

- Antrasis scenarijus:
  Antrame scenarijuje tereikia įvesti tekstą. Tekstas gali būti ilgas arba trumpas, vienos ar daugiau eilučių. Pavyzdžiui:

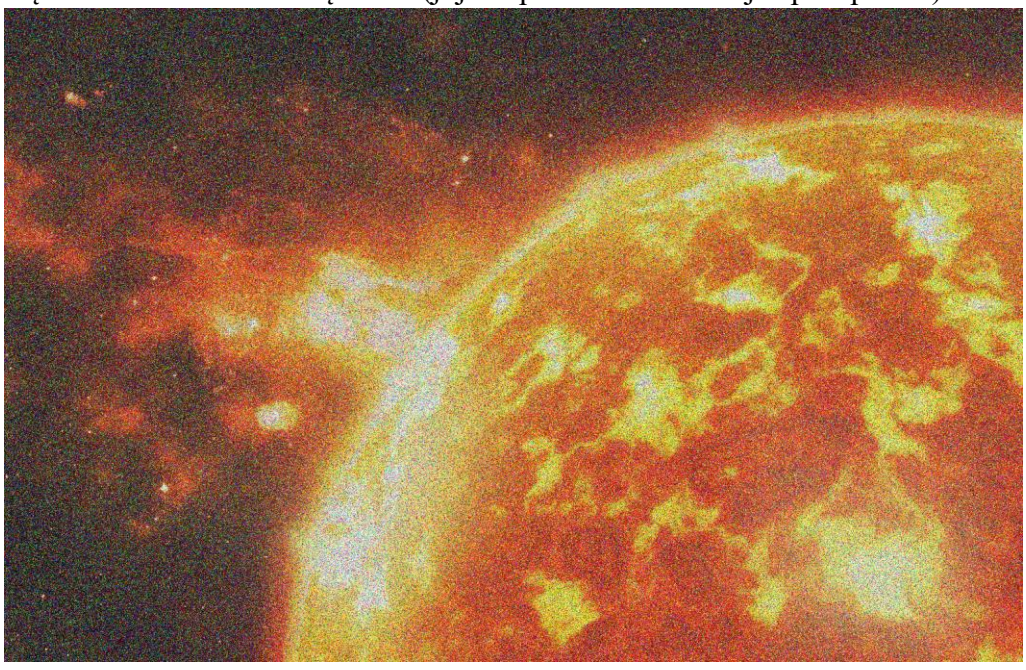
```
Prašome įvesti tekstą. Tekstas gali būti sudarytas iš vienos daugiau eilučių. Ka
i norėsite baigti, spauskite „Enter“:
Bėgo kiškis ir pamatė lapę.
Lapė pasisvei-
kino.
Kiškis
išsigando ir
pabėgo.
```

  Jei norite pradėti naują eilutę, paprasčiausiai įvesto teksto eilutės pabaigoje paspauskite „Enter“. Jei baigėte rašyti tekstą, spauskite „Enter“ tuščioje eilutėje, kitaip programa manys, kad tiesiog norite pradėti naują teksto eilutę.
  Įvedus tekstą, bus priminta Jūsų programos pradžioje įvesta kanalo klaidos tikimybė, bei išspausdinti du tekstai. Pirmasis sukonstruotas iš kanalo gale pasirodžiusios neužkoduotos bitų sekos, antrasis iš užkoduotos bitų sekos.

```
Jūsų įvestą tekstą siunčiame neužkoduotą kanalu, kuriame tikimybė padaryti klaid
ą lygi 0.1.
Kanalo gale pasirodė štai toks pranešimas: z♦♦gl Kh♦kiz"hB ptmqvćl`♦ę.♦n!x♦♦(pas
isvea-k)no.JI♦škxs♦i♦♦s♦g9jdo -z
uaJ 'o.

Jūsų įvestą tekstą užkoduojame ir siunčiame tuo pačiu kanalu.
Kanalo gale pasirodė štai toks pranešimas: Bėgo kiškis ir!pamatė lapę.
Lapė pasisvei-
kino.
Kiškis
išsigando ir
pabėgo.
```

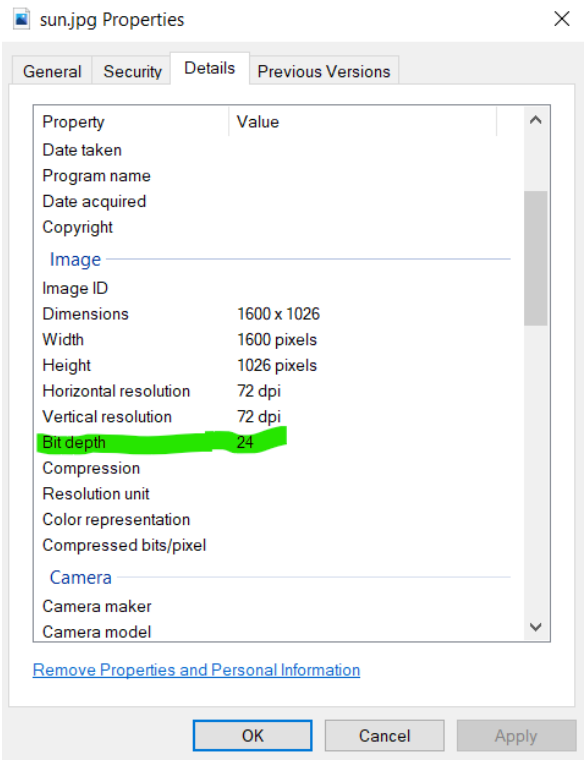
- Trečiasis scenarijus:
  Jums tereikia įvesti kelią iki paveiksliuko, kurį norite siųsti nepatikimu kanalu. Įvedus kelią iki paveiksliuko programa jį atidarys.

Jeigu atidaryti nepavyko, programa baigs darbą. Kitu atveju primenama vartotojo įvesta klaidos tikimybė ir iš karto pradedamas paveiksliuko spalvų išgavimas ir siuntimas kanalu. Šis procesas gali užtrukti. Iš pradžių jums bus atidarytas paveiksliukas, sukonstruotas iš nepatikimu kanalu siųstos neužkoduotos bitų sekos (joje talpinama informacija apie spalvas).

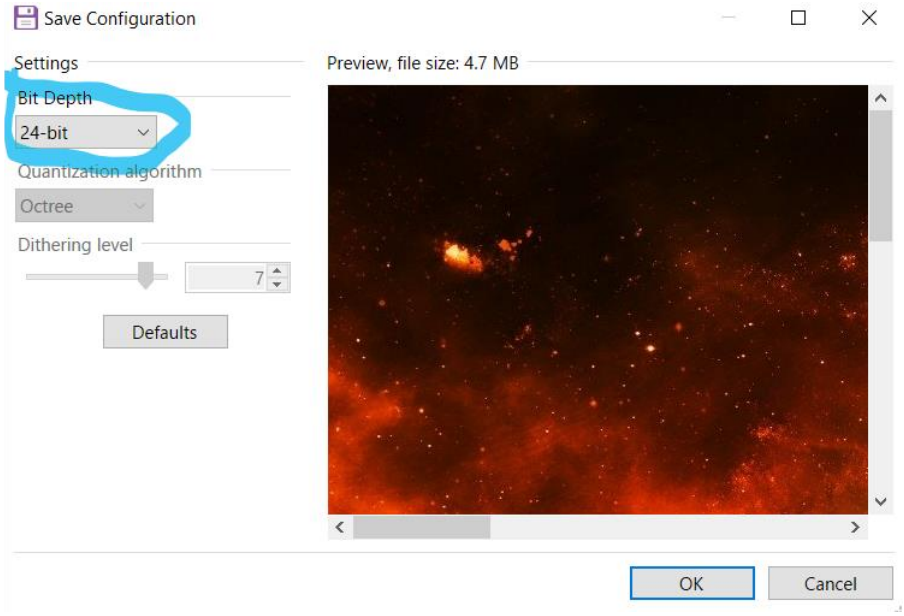


Po kurio laiko bus atidarytas kitas paveiksliukas, sukonstruotas iš užkoduotos bitų sekos.

Pastebėkime, kad neužkoduotas paveiksliukas aiškiai sugadintas, o koduoto paveiksliuko iškraipymai tokio dydžio paveiksliuke atrodo kaip mažos žvaigždelės ir niekam neturėtų kelti įtarimo.
Sukurti paveiksliukai išsaugomi laikiname „Temp“ aplanke su priešdėliu „tempCodeTheoryFiles“ ir galūne „bekodo.bmp“ arba „koduotas.bmp“ atitinkamai. Jeigu šiame scenarijuje įvyksta klaida, programa baigia darbą. Todėl vertėtų įsitikinta, ar „bmp“ paveiksliukas, kurį nurodote, yra užkoduotas spalvomis, kurių gylis 3 baitai (rgb). Tą galite įsitikinti paspausdami dešinįjį klavišą ant paveiksliuko, pasirinkdami „Properties“ tada „Details“ ir patikrinę reikšmę atitinkamame lauke:



Laukelio reikšmė turi būti 24. Jei ji tokia nėra, galite atidaryti paveikslėlį piešimo programėle „paint.net“ (jei turite), spausti „Save As“ ir pasirinkus bmp formatą, nustatyti apibraukto laukelio reikšmę 24:



Įgyvendinti programiniai sprendimai

Siunčiant vektorių kanalu, jis privalo būti atitinkamo ilgio (kaip nurodo skaičius k). Tačiau į programą įvedamas tekstas, konvertuotas į bitų seką, dažnai gali nesidalinti į k ilgio

vektorius po lygiai. Galima būtų neimti paskutiniojo vektoriaus, tačiau jeigu k didelis, bus prarandamas nemažas informacijos kiekis. Todėl matematinių skaičiavimų klasėje įvedama formulė:

```
int vectorCount = binaryText.length() / codeData.getK();
int additionalBitsToAdd = codeData.getK() - binaryText.length() % codeData.getK();
if(additionalBitsToAdd == codeData.getK()) additionalBitsToAdd = 0;
if(additionalBitsToAdd != 0) vectorCount+=1;
```

Kai duomenų neišeina padalinti į k ilgio vektorius po lygiai, pridedamas papildomas vektorius.

Antrojo ir trečiojo scenarijaus klasės papildomų bitų skaičių išsisaugo:

```
additionalBitsToAdd = codeData.getK() - binaryText.length() % codeData.getK();
if(additionalBitsToAdd == codeData.getK()) additionalBitsToAdd = 0;
```

Pasinaudojus šia informacija, galima apskaičiuoti, kiek paskutinio vektoriaus bitų priklauso siunčiamai informacijai, o kiek yra iš jų yra pridėtiniai. Jeigu bitų sekos ilgis yra skaičiaus k kartotinis, tai „int“ tipo „additionalBitsToAdd“ kintamasis įgaus reikšmę k. Tokiu atveju viskas išsidalino po lygiai ir kintamojo reikšmę verčiame į nulį.

Jeigu bitų sekos ilgis nėra skaičiaus k kartotinis, tai „additionalBitsToAdd“ reikšmė taps apskaičiuotas perteklius. Apskaičiavę atkoduotą bitų eilutę su pertekliumi, galėsime atmesti papildomus bitus ir gauti pradinę bitų eilutę (jeigu klaidų buvo padaryta ne daugiau, nei dekodavimo algoritmas gali ištaisyti):

```
corruptedBinary = new StringBuilder(corruptedBinary.substring(0, corruptedBinary.length() - additionalBitsToAdd));
return corruptedBinary.toString();
```

Nors papildomi bitai yra nuliniai ir, regis, prilygtų eilutės pabaigos simboliui „\0“, šis atmetimas gali būti naudingas situacijose, kur atminties taupymas yra labai svarbus.

Iš tiesų programą stengtasi parašyti taip, kad ji taupytų kuo daugiau atminties. Naudodami programą, pastebėsite, kad neprašoma įvesti generuojančios matricos vienetinės dalies ir ši dalis niekada nėra spausdinama. Taip padaryta ne tik dėl to, kad vartotojui būtų patogiau įvesti duomenis. Programa vienetinės dalies nesaugo atmintyje. Vienetinė dalis turi dėsningumą, visur nuliai išskyrus pagrindinėje įstrižainėje. Tuo galima pasinaudoti ir užkoduojant kodas c bus pradinė žinutė ir prie jos pridėta matricų m ir Glikusi sandauga: c = m + m x Glikusi, kur Glikusi yra generuojančios matricos dešinioji pusė (t.y. G be vienetinės dalies). Tokiu atveju nereikia apkrauti atminties, nes galime šią logiką įprogramuoti skaičiavimuose.

Vienetinės dalies nereikia saugoti, net jeigu norime rasti vektoriaus sindromą. Žinome, kad jeigu vektoriai užkoduojame tik bitais, tai $G = (I \mid A) \Leftrightarrow H = (A^t \mid I)$. Pavyzdžiui:

$$G = \begin{array}{ccc|ccc} 1 & 0 & | & 1 & 1 & 0 \\ 0 & 1 & | & 0 & 1 & 1 \end{array} \Leftrightarrow H = \begin{array}{ccc|ccc} 1 & 0 & | & 1 & 0 & 0 \\ 1 & 1 & | & 0 & 1 & 0 \\ 0 & 1 & | & 0 & 0 & 1 \end{array}$$

Tačiau pastebėkime, kad norėsime surasti sindromą: $s(y) = H \times y^t$, galime poromis su y^t atitinkamomis pozicijos dauginti tik kontrolinės matricos kairiąją dalį. Tačiau dešiniosios dalies daugybą irgi privaloma realizuoti. Žinome, kurią kontrolinės matricos eilutę dabar dauginame. Jeigu dauginame pirmąją, prie sandaugos pridėsime tik (pirmąjį + k) transponuoto vektoriaus y^t stulpelį. Kai dauginsime antrąjį kontrolinės matricos stulpelį, pridėsime (antrąjį + k) ir taip nelaikydami vienetinės matricos dalies atmintyje, galėsime apskaičiuoti vektoriaus y sindromą s(y).

```
public int[] calculateSyndrome(int[] vector) {
    int[] syndrome = new int[n-k];
    for(int i = 0; i < (n-k); i++) {
        for(int r = 0; r < k; r++) {
            syndrome[i] += (parityMatrix[i][r] * vector[r]); // Transpoziciją atlieku tiesiog čia ir dabar.
        }
        syndrome[i] += vector[k + i]; // vienetinės dalies pasirodymas.
        syndrome[i] %= 2; // Mūsų kūnas yra dvejetainis.
    }
    return syndrome;
}
```

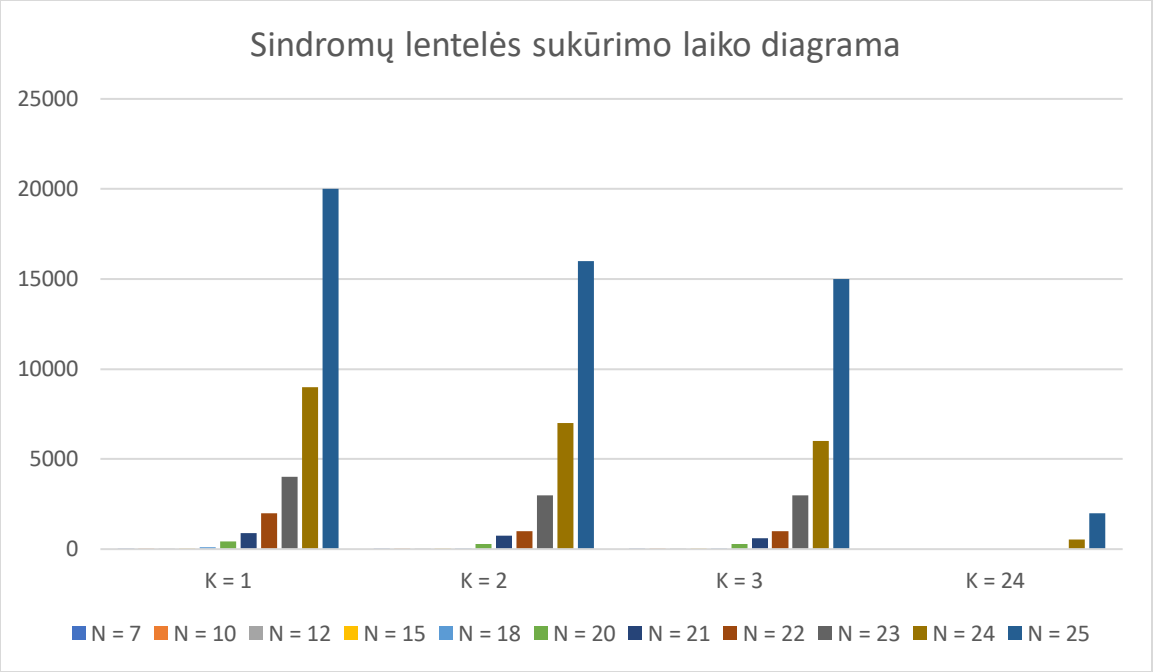
„syndrome[i] += vector[k + i]“, kur i yra kontrolinės matricos eilutės stulpelio rodiklis, ir atlieka šį triuką.

Prie atminties saugojimo prisideda ir paieškos į gylį algoritmas. Sindromų apskaičiavimo taisyklės sako, kad turime pradėti nuo mažiausio svorio vektorių ir tik išnagrinėjus visus to svorio variantus, padidinti kandidatų svorį, ieškant sindromų lyderių. Trumpai tariant, tokiam ieškojimui trumpiausias yra paieškos į plotį algoritmas. Tačiau, nors jis yra greitesnis, atminties atžvilgiu yra brangus, nes jo sudėtingumas atminties atžvilgiu yra O(didžiausias plotis). Iš viso yra 2^{n-1} variantų, ir kadangi paieškos į plotį algoritmas naudoja „backtracking“ technologiją, visus lapus teks saugoti atmintyje, kad prie jų būtų galima sugrįžti ir pratęsti. Paieškai į gylį tiek atminties nereikia, užtenka O(didžiausias gylis). Medžio gylis yra kodo žodžio ilgis n+1. Pastebėkime, kad $n + 1 \leq 2^{n-1}$, jei tik n >= 3. Taigi atminties taupymo atžvilgiu, paieškos į gylį algoritmas laimi. Nors ir nukenčia greitis, kadangi nesaugome matricos vienetinės dalies, naudodami šiuos kelis programavimo sprendimus kartu, tam tikrose situacijose, išlošiame nemažai atminties. Ir ten, kur atmintis ribota, tas išties gali praversti.

Eksperimentai

Eksperimentų aprašymo logiką galite rasti aplanke „programos kodo tekstai/tests“. Tai nėra tikrieji testai, kadangi nepanaudota „Assert“ dalis, o tik būdas izoliuoti programą, kai kurias dalis perkopijuojant į testų klases, kad beeksperimentuojant nebūtų palikta klaidų.

Panagrinėkime, kaip greitai programa sugeba sugeneruoti sindromų lentelę, parinkus tam tikrus k ir n parametrus. Vertikalioje ašyje pateiktas laikas milisekundėmis:



Šiuos duomenis taip pat galite pamatyti nuėję į aplanką „eksperimentai“ bei pasirinkę „sindromų lentelės sudarymo laikas.txt“ dokumentą. Šį (ar panašų) eksperimentą galite pasileisti nuėję į aplanką „programos kodo tekstai/tests/data“ ir pasileidę „SyndromeTableTests.java“ viduje esantį testą.

Iš karto gali kilti bent 2 klausimai:

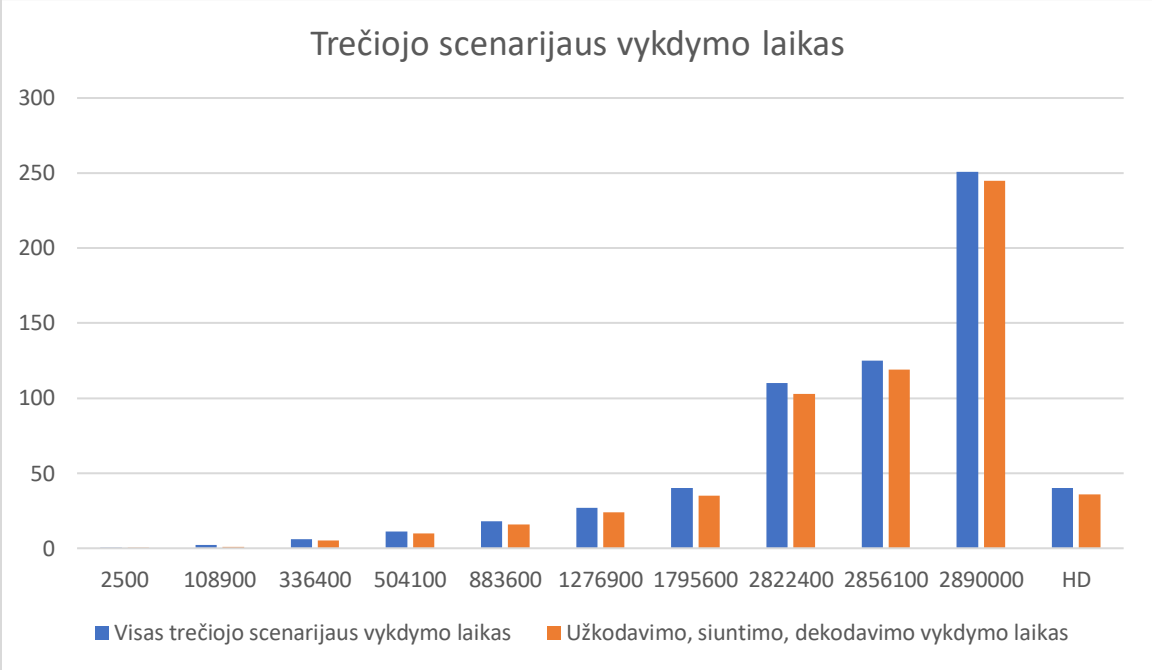
1) Kodėl kai k = 24, o n = 24 arba n = 25, programa užtrunka net apie 2 sekundes? Juk galimų sindromų skaičius apskaičiuojamas pagal formulę q^(n-k), o mūsų atveju q = 2? Ši formulė tinka tik paieškos į plotį atveju. Mes ieškome į gylį siekdami sutaupyti atminties, tačiau atmintis kainuoja laiko. Kadangi ieškodami į gylį dažnai pirmiau aptinkame sunkesnius vektorius, juos tenka pakeisti vėliau, todėl surinkę maksimalų galimų sindromų skaičių negalime naudoti „Java“ raktažodžio „break“ norėdami sustabdyti ciklą. Kitaip gausime neteisingai sindromą atspindinčius lyderių svorius.
2) Jeigu paieška į gylį negali stabdyti ciklo, kai visi sindromai yra lentelėje, kodėl tada laikas, vis dėlto, mažėja, kai n ir k skirtumas mažas, bei n pakankamai didelis? Nepamirškime, skaičiuojant sindromą, padeda tai, kad vienetinė matrica yra įprogramuota, bet nesaugoma atmintyje, o tai jau optimizavimas:

```
for(int i = 0; i < (n-k); i++) {
    for(int r = 0; r < k; r++) {
        syndrome[i] += (parityMatrix[i][r] * vector[r]);
    }
```

Tereikia perbėgti (n-k) * k variantų ir kiekviename cikle pridėti įsivaizduojamą vienetinę dalį bei apskaičiuoti modulį (žinoma, modulis būtinas bet kurio atveju):

```
syndrome[i] += vector[k + i]; // vienetinės dalies pasirodymas.
syndrome[i] %= 2; // Mūsų kūnas yra dvejetainis.
```

Toliau nagrinėsime programos gebą nuskaityti paveiksliukus. Vertikalioje ašyje nurodomas laikas sekundėmis, horizontalioje ašyje – paveiksliuko plotais pikseliais. Paimkite vidutinio ilgumo kodavimą: k = 2, n = 15:



Tai tik ištrauka iš eksperimento, daugiau duomenų galite pamatyti nuėję į aplanką „eksperimentai“ ir pasirinkę „paveiksliukų laikas.txt“ bei „paveiksliukų laikas 2.txt“ dokumentus. Tik įdėmiau peržvelgę „paveiksliukų laikas.txt“, pastebėsite tokią ir panašias klaidas:

```
Visas trečiojo scenarijaus vykdymo laikas:17 (sekundės).
Visas trečiojo scenarijaus vykdymo laikas:16 (sekundės).
```

Iš tiesų turi būti:

```
Visas trečiojo scenarijaus vykdymo laikas:34 (milisekundės).
Užkodavimo, siuntimo, dekodavimo vykdymo laikas:27 (milisekundės).
```

ir panašiai.

Eksperimento vykdomasis failas guli aplanke „programos kodo tekstai/tests/scenarios“ pavadinimu „ThirdScenarioTests.java“.

Diagramoje matosi, kad programai nekyla sunkumų atidaryti paveiksliuką ir nuskaityti iš jo duomenis. Žinoma, esant didesniam paveiksliukui ilgėja nuskaitymo laikas, tačiau didžioji dalis trečiojo scenarijaus vykdymo laiko atitenka užkodavimui, siuntimui ir dekodavimui. Pasinaudoję programa pastebėsite, kad sugadintas paveiksliukas atidaromas gana greitai, o užkoduoto tenka palaukti ilgiau. Taip yra todėl, kad ne tik padidėja pats paveiksliuko dydis, juk kiekvienam papildomam pikseliui tenka 3 baitai ir programa kiekvieną jį turės užkoduoti. Užkodavimo laiką galima sutrumpinti imant didesniais intervalais, bet tai sumažins pakankamai gero paveikslėlio gavimo atkodavus tikimybę.

Trečiasis eksperimentas nagrinės klaidų aptikimo statistiką, siunčiant tekstą nepatikimu kanalu užkodavus ir be kodo. Turime štai tokį tekstą (ilgis 560 simbolių, 4903 bitų):

```
Gyveno neturtingas žmogus. Nuėjo į bažnyčią ir meldžiasi atsiklaupęs:
– Kad dievas duotų man pinigų ir trūktų nors vieno rublio iki šimto, neimčiau.
Kunigas, išgirdęs tą jo kalbą, liepė zakristijonui pamesti prie to žmogelio devyniasdešimt devynis rublius.
 Zakristijonas numetė. Žmogus pasiėmė tuos pinigus ir sako:
– Tikiu, dieve, jeigu jau davei devyniasdešimt devynis rublius, tai atiduosi ir tą vieną.
Tada kunigas prišokęs ėmė šaukti:
– Atiduok pinigus!
O žmogus kunigui – su lazda.
– Kokius pinigus? – sako jis. – Man dievas davė! –ir išėjo patenkintas.""";
```

Pritaikysime jam eksperimentą, kuris aprašytas aplanke „programos kodo tekstai/tests/scenarios“ pavadinimu „SecondScenarioTests.java“. Testavimą pradėsime imdami klaidos tikimybę 0.01 ir baigdami 0.99. Kad skaitytojui diagrama būtų aiški, pateiksime tik eksperimento ištraukas. Viso eksperimento informacija pasiekiame aplanke „eksperimentai/klaidų statistikos“ (klaidos tikimybė buvo didinama kas 0.01).

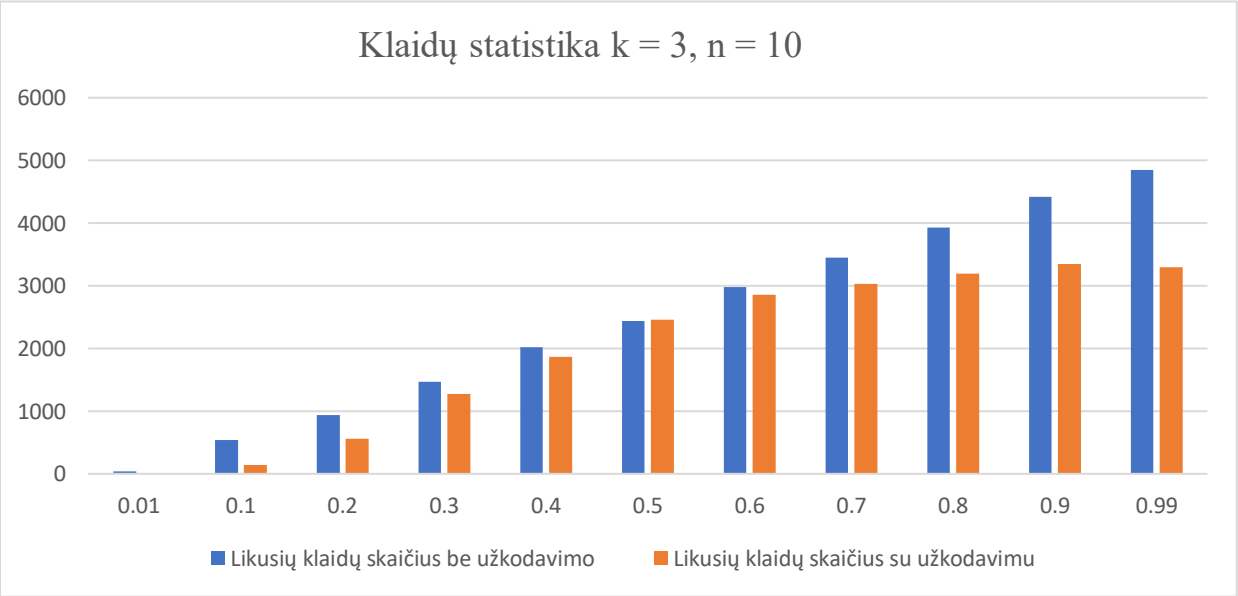
Vertikaliose diagramos ašyse bus padarytų klaidų skaičius, horizontaliose klaidos tikimybė.

Imkime k = 3, n = 10. Programos sugeneruota matrica (be vienetinės dalies):

[0, 0, 0, 1, 1, 1, 0]

[1, 1, 0, 1, 1, 0, 0]

[1, 1, 1, 1, 0, 0, 1]



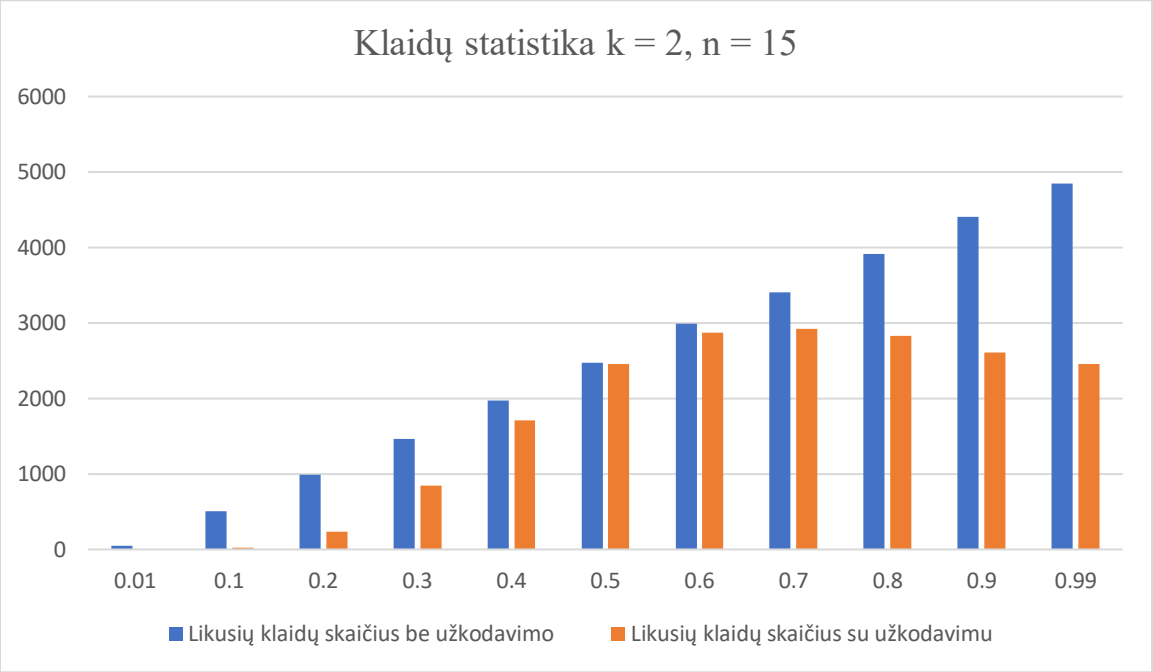
Iš diagramos matyti, kad esant 50% tikimybei, tokios generuojančios matricos geriau nenaudoti, nes tikėtina, kad atkodavus bus padaryta daugiau klaidų, negu siunčiant informaciją neužkoduotą. Tačiau matosi, kylant klaidos tikimybei virš 50 procentų, nors klaidų vis didėja, klaidų skirtumas tarp užkoduoto ir neužkoduoto vektoriaus taip pat didėja. Negalima sakyti, kad

generuojanti matrica yra prasta, nes esant mažoms tikimybėms atkodavus klaidų lieka mažiau nei siunčiant pranešimą nekoduotą. Iš diagramos matome, kad ši matrica yra atsparesnė, kai klaidos labiau koncentruotas viena prie kitos (t.y. kai tikimybė didelė). Vėliau pamatysime kodą, kuris esant didelei klaidos tikimybei geba tik dar labiau sugandinti vektorių.

Imkime kitą pavyzdį: tegu k = 2, n = 15. Programos sugeneruota matrica (be vienetinės dalies):

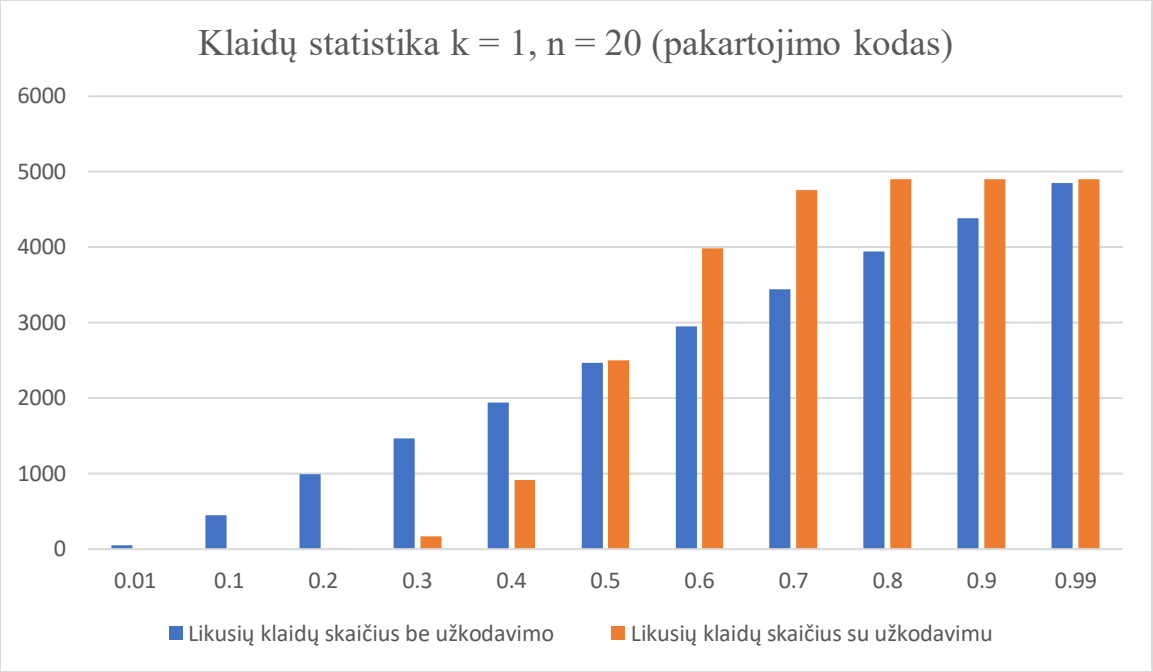
[1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]

[0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1]



Iš diagramos matome, kad kodui neverta taisyti klaidų, kai tikimybė yra arti 50%. Bet nieko blogo ir ištaisyti klaidas. Taip pat matome, kad didėjant klaidų skaičiui, klaidų skirtumas didėja. Tiesą sakant, kodas, su 80% tikimybę padaro mažiau klaidų nei tas pats kodas su 70% tikimybę. Jeigu kanalo klaidos tikimybė 99%, kodas, tikėtina, ištaisys daugiau klaidų, negu su 80% klaidų darančiu kanalu.

Pateikime ir pakartojimo kodo pavyzdį. Imkime R_{20}:



Iš diagramos matome, kad pakartojimo kodas klaidas taiso žymiai geriau už kitus du anksčiau pristatytus kodus, net kai ir klaidos tikimybė yra pakankamai didelė (pavyzdžiui, 40%). Tačiau priėjus 50% ribą viskas apsiverčia aukštyn kojom. Jei kiti kodai sugeba kovoti su klaidomis net esant dideliam jų procentui, pakartojimo kodas ne tik nesugeba klaidų ištaisyti, bet betaisydamas dar labiau sugadina esamą informaciją. Galima pastebėti, kad ties 80-90 procentų riba ir toliau, visi dekoduojami bitai yra klaidingi. Kodėl taip yra? Kadangi naudojamas pakartojimo kodas R_{20}, kad būtų atkoduota teisingai, viename vektoriuje turi būti padaryta ne daugiau devynių klaidų. Jei padaroma vienuolika ar daugiau klaidų, pakartojimo kodas bitą būtinai atkoduos neteisingai. Tikimybė, kad iš 20 bitų bent 11 bus neteisingų, kai klaidos tikimybė yra 80 procentų ar daugiau yra labai didelė. Todėl viskas ir dekoduojama klaidingai. Tačiau galima pažvelgti ir iš geros pusės. Ką mes pastebėjome? Jeigu žinome, kad kanalo tikimybė yra didelė, galime prieš algoritmui pradedant dekoduoti iš kanalo gautą vektorių vienetus keisti į nulius, nulius į vienetus. Tada 99% klaidos tikimybės atveju šis pakartojimo kodas turėtų visą tekstą ištaisyti teisingai. Taigi apibendrinus, jeigu turime kodą, kuris pavyzdžiui, prie 50% ribos pradeda duoti daugiau žalos negu naudos, apkeiskime jo bitus prieš pradedant dekodavimą. Šiuo atveju, jeigu klaidos tikimybė yra 99%, pakeisdami bitus galima skelbti, kad klaidos tikimybė yra 100%-99%=1%. O kadangi pakartojimo kodas yra stiprus ir sugeba rasti daug klaidų, praktiškai ji yra nulinė. Programoje šis būdas nėra įgyvendintas.

Pabaigai verta paminėti, kad yra ir daugiau eksperimentų. Vaizdiniai antrojo ir trečiojo scenarijaus eksperimentai (pavyzdžiai su skirtingais tekstais ir nuotraukomis) pateikti aplanke „eksperimentai/2scenarijus“ ir „eksperimentai/3scenarijus“.

Literatūra

1) Pagrindinis informacijos šaltinis rašant klaidas taisančius kodus: Gintaro Skersio „Klaidas taisančių kodų teorija“ paskaitų konspektas, 2021.
2) Panaudoto „grandininio“ („step-by-step“) dekodavimo algoritmo aprašymas: S.A.Vanstone, P.C. van Oorschot. An introduction to error correcting codes with applications. Kluwer Academic Publishers, Boston, 1989.