LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

STABILIZATORU KODI KVANTU KĻŪDU KOREKCIJAI

BAKALAURA DARBS

Autors: **Boriss Benzerruki**
Studenta apliecības № - bb06008
Darba vadītājs: profesors Dr. dat. Andris Ambainis

RĪGA 2010

ANOTĀCIJA

Darbā pētītas kļūdu korekcijas metodes kvantu skaitļošanai. Tā kā viena no biežākajām kvantu kļūdu korekcijas metodēm ir stabilizatoru kodi, darba mērķis ir šādu kodu atrašana un teorētiska pētīšana. Darba teorētiskās izpētes rezultātā tika atrasti daži nosacījumi, pie kuriem stabilizatoru kodi var pastāvēt, un pie kuriem to pastāvēšana ir neiespējama. Darba ietvaros ir veikts datorpārlases mēģinājums dažiem vienkāršiem gadījumiem (izvēloties tos gadījumos, kuros, saskaņā ar teorētiskās izpētes rezultātiem, varētu pastāvēt stabilizatoru kodi).

Atslēgvārdi: *Kvantu skaitļošana, Kvantu kļūdu korekcija, Stabilizatoru kodi.*

ABSTRACT

This thesis studies the methods of error correction for quantum computing. One of most commonly used methods is stabilizer codes. Therefore, the goal of this thesis is studying and finding such codes. As a result of theoretical studies in this thesis, we have found several conditions for existence (or non-existence) of stabilizer codes. We have also conducted an attempt of computer search for stabilizer codes in several simple cases (choosing the cases, when, according to our theoretical study, such codes might exist).

Keywords: *Quantum Computing, Quantum Error Correction, Stabilizer Codes.*

SATURS

IEVADS

1965. gadā, sešus gadus pēc integrālas shēmas uzbūvēšanas, Gordons Mūrs (viens no Intel dibinātājiem), izsecināja tendenci, ka jaunie mikroshēmu modeli nomainīja vecos pēc aptuveni vienādiem laika periodiem – pēc 18—24 mēnesiem. Katru reizi to tranzistoru skaits apmēram divkāršojās. Jā tāda tendence turpināsies, teica Mūrs, tad elektronisko ierīču jauda diezgan īsā laika periodā eksponenciāli pieaugs. Šis novērojums tika nosaukts par „Mūra likumu". [3.1] Ir liela līdzīgu apgalvojumu kopa, kuri raksturo citus eksponenciālus pieaugumus un tos arī dažreiz sauc par Mūra likumiem. Piemēram, mazāk pazīstamais „Otrais Mūra likums" (ko Judžins Mejerāns ieviesa 1998. gadā) apgalvo, ka rūpnīcu cenas, kas ražo šīs mikroshēmas eksponenciāli pieaug atkarībā no to sarežģītības.

Cilvēka vajadzības pēc jaunām tehnoloģijām arī pieaug eksponenciāli un attiecība starp eksponencialām prasībām un eksponenciāliem piedāvājumiem pagaidām turpinās veidoties pēc tās pašas lineārās shēmas. [3.2] Bet eksistē svarīga problēma. Līdz kuram brīdim šī tendence ir spējīga turpināties? Galvenais ierobežojums ir tas, ka divreiz lielākais atmiņas mikroshēmu daudzums vai nu prasa sevīm vairāk vietas (kas, nav pieļaujams, jo gadījumā, ja neizdosies saglabāt prātigus skaitļošanas mašīnas processora izmēra standartus, sadzīves vajadzība pēc šiem datoriem strauji samazināsies), vai nu arī tranzistoru izmērs jāsamazina pēc aprakstīta plāna – divās reizēs katru divu gadu laikā, kas galu galā radīs nepieciešamību uzbūvēt atmiņas vienību, kuru izmērs bus jau tuvs vienam atomam.

Tajā pašā 1965. gadā Ričards Feinmans kļuva par Nobeļa prēmijas laureātu kvantu fīzikā. Viņš bija pirmais, kas argumentēja iespēju realizēt kvantu datorus [3.3]

Kvantu datori balstās uz kvantu mehāniku. Tā kā kvantu mehānikas likumi ir ļoti atšķirīgi no parastās fizikas likumiem, kvantu datori, no vienas puses prasa rūpīgu izpēti, attīstot jaunas tehnoloģijas, un no otras puses pavērš jaunas iespējas, piemēram dažu skaitļošanas uzdevumu paatrināšanu. Piemēram, Šora algoritma pielietošanai uz kvantu datora laika sarežģītības O-mērs būs: $O\left(n^3\right)$ [3.4], savukārt klasiskajos datoros tās sarežģītība $O\left(2^{n^{\frac{1}{3}}}\right)$. [2.1] Otrs piemērs ir Grovera meklēšanas algoritms masīvā. Lieliem n tas dod kvadrātisku priekšrocību salīdzinot ar parastajiem datoriem. [2.2] Vēl viena nozare, kur kvantu datori varētu būt lietderīgi ir Kriptogrāfija, jo tie ir spējīgi nodrošinat drošāku datu pārraidi. Jebkura kvantu stāvokļa mērīšana atstāj pēdas un ved pie iznākuma ziņojuma sagrozījumam. Tāpēc nelegāla pārraidāma signāla noklausīšanās nevar būt nepamanāma. Rezultātā iegūstam spēcīgu šifrēšanas sistēmu, kuras drošību garantē kvantu mehānika. [4.1]

Uz doto momentu, protams, kvanu skaitļošanas nozare attīstās ne tikai savā teorētiskā sfērā, bet arī kvantu datora fiziskās uzbūves realizācijā.

2010. gada sākumā zinātniekiem izdevās veiksmīgi izmantot 20 kubitu kvantu datoru precīzai molekulāra ūdeņraža enerģijas apreķināšanai. [1.1] Šis triumfs molekulāru pētījumu metodēs sola nozīmīgu potenciālu ne tikai kvantu ķīmijai, bet arī daudziem citiem zinātnes apgabaliem. Pēc pētnieku vārdiem, sistēma no 128 kubitiem varēs pārspēt klasiskus datorus, kā instrumentu sarežģītāku ķīmijas uzdevumu risinājumiem. Kriptogrāfijas uzdevumiem būs vajadzīgi tūkstoši kvantu bitu. [1.1]

Kvantu stāvokļiem mijiedarbojoties ar apkārtējo vidi var rasties kļūdas, kas traucē skaitļošanas procesu. Tāpēc nepieciešamas metodes kļūdu korekcijai kvantu datoriem. [2.3]

Pirmo kvantu kļūdu korekcijas algoritmu atklāja P. Šors 1995. gadā [2.4]. Populārākā kļūdu korekcijas metode ir stabilizatoru kodi [2.5].

Šīs Bakalaura darba mērķis ir apskatīt stabilizatoru pieeju kvantu kļūdas koriģējošiem kodiem un izanalizēt iespēju veikt efektīvu datoru pārlasi jaunu kodu atrašanai no ātrdarbības un atmiņas izmantošanas viedokļa. Ja veiksmīgu datoru pārlasi veikt neizdosies, tad mērķis varētu būt izmeklēt īpašības, kuri varēs pazemināt programmas izpildes laiku, pie sāmērā maziem ziņojuma garumiem un kļūdu skaitam, kurus būs tajā jākoriģē.

Šīs darbs ir veltīts kvantu kļūdu korekcijai, izmantojot stabilizatoru kodus. Ir uzbūvēta teorētiskā bāze, kuru ļāva efektīvi definēt un pierādīt svarīgas īpašības, kuras tika izmantotas formālai teoretiskai problēmas izpetei ar mērķi atrast paņemienus gan datoru pārlases efektivitātes palielināšanai un teorētiskas problēmas atrisināšanai visparīgā formā. Darba sākumā ir dots rūpīgs kļūdu koriģejoso kodu rupīgs apraksts, ka arī stabilizatoru kodu pielietojumu piemēri kvantu kļūdu korekcijai ar problēmas noformulējumu. 2. Nodaļa ir veltīta teorijas rūpīgai uzbūvei noderīgu īpašību izmeklēšanai, lai padarītu datorpārlasi efektivāku un virzīt uz priekšu pētamo teoretisko problēmas izpeti. Ka arī 2. nodaļā tika novērtēta datorpārlases laika sarežģītība un doti attiecīgi secinājumi. 3. nodaļā ir doti *Programmatūras Prasību Specifikācijas* un *Programmatūras Projektējuma Apraksta* fragmenti, izvirzīti mērķi datorpārlases programmai un datorpārlases sasniegtie rezultāti. Darba gala rezultāti un secinājumi ir aprakstīti attiecigās nodaļās.

1. TEORIJAS APRAKSTS

1.1. Kļūdu koriģējošie kodi

Glabājot, apstrādājot vai pārraidot jebkura tipa datus pa sakaru kanāliem, var rasties kļūdas. Pat vienas bita kļūdas rašanas var radīt nopietnas problēmas. Piemēram, ja pārraidītā informācija ir datorprogramma, tad tā var kļūt darboties nespējīga. Ja pārraidītā informācija ir arhivēti (saspiesti) dati, tad tos var būt neiespējams atarhivēt. Tāpēc jāmāk kļūdas atpazīt un koriģēt.

Šo problēmu var atrisināt izmantojot *kļūdas koriģējošus kodus*. Tad informācijai tiek atvēlēti vairāk bitu, nekā tā aizņem. „Liekajiem" bitiem ir sava funkcija – kontrolēt, vai nav kļūdu starp atlikušajiem paša kodēta *ziņojuma bitiem* ***X*** (daļēji dublējot ziņojuma informāciju noteiktā veidā). Šos „liekus" bitus mēs turpmāk sauksim par *pārbaudes* vai *nosacījuma bitiem* ***Y***. Visu ziņojuma virkni gan ar *ziņojuma bitiem*, gan ar *pārbaudes bitiem,* sauksim par *nokodēto bitu virkni* **M** vai par *nokodēto ziņojumu.*

Pārraidot *nokodētos bitus*, *pārbaudes biti* veic nosacījuma lomu, un katrs tāds bits spēj pārbaudīt ziņojuma sastāvdaļas korektīvu. Piemēram, tas var būt vienāds ar pāra bitu summas rezultātu pēc moduļa 2. Mūsu mērķis ir, lai mēs spētu noteikt kļūdu skaitu ***k*** un katras kļūdas vietu. Tas ļaus koriģēt kļūdas pēc nokodēta ziņojuma modificēšanas – ilgas glabāšanas, apstrādāšanas vai pārraides rezultātā.

Par **kļūdu koriģējošo** [***n***,***p***,***k***]-**kodu** sauksim *nokodēto ziņojuma* virkni ***M*** garumā ***n***, ar ***p*** *ziņojuma bitiem*, kurā var koriģēt jebkuru kombināciju no ne vairāk kā ***k*** kļūdām.

Tātad, sākumā mums ir kāda ziņojuma bitu virkne no kopas ***X*** elementiem garumā ***p***: $(x_1, x_2, ..., x_p)$; $kur: \ \forall i(x_i \in \{0;1\})$ - ziņojuma dati (p bitu virkne).

Pēc ziņojuma nokodēšanas dabūjam nokodēto ziņojuma bitu virkni ***M*** ar garumu ***n***:
$M = x_1, x_2, ..., x_p, y_1, y_2, ..., y_q$; $kur: \ \forall i(x_i, y_i \in \{0;1\})$ - nokodēta ziņojuma dati ($n = p + q$ bitu virkne ar q nosacījuma bitiem, vai vienkārši – ar q nosacījumiem).

Dažreiz varētu būt arī pietiekami ar ***k*** kļūdas atpazīšanu nevis to labošanu, jā tādi nejauši parādās. Piemēram datu pārraidē, ja kļūda parādās, tad kļūdu koriģēšana nav nepieciešama, jo visu ziņojumu var atsūtīt atkārtoti, tikreiz, kamēr kļūdu nebūs.

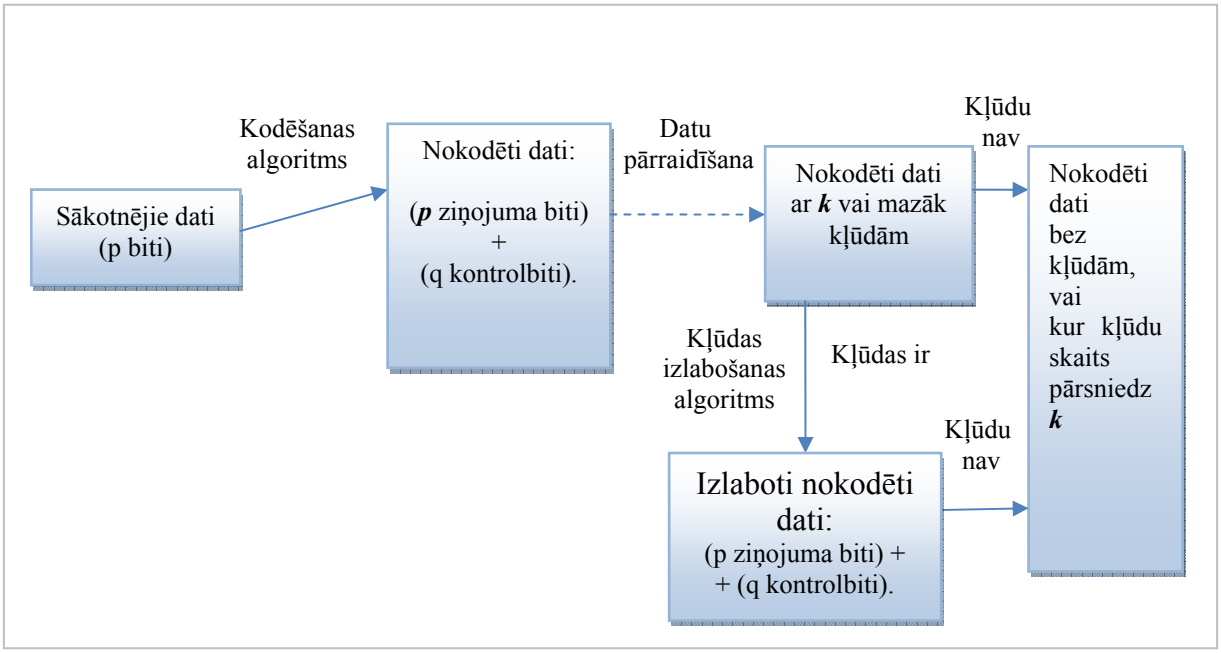
Lai saprast kā tas strādā, apskatīsim vienas kļūdas gadījumu. Tātad nofiksēsim ***k:*** *k=1***.** Manuprāt, vienkāršākais veids, kā veikt vienu ziņojuma kļūdu konstatēšanu varētu būt šāds:

Pārraidām nokodēto ziņojumu ***M*** sekojošā veidā:

$M = x_1, x_2, ..., x_p, y_1;$ *kur*: $y_1 = (x_1 + x_2 + ... + x_p) \bmod 2$. Nav grūti saredzēt, ka pēc nokodēšanas jebkuram nokodētam ziņojumam piemīt šāda īpašība: Ja tajā nav nevienas kļūdas tad šeit visu nokodēto ziņojumu ***p***+*1* bitu paritāte būs vienāda ar *0*. Savukārt ja ziņojumā parādīsies viena kļūda – paritāte būs *1*, un mēs kļūdas eksistenci spēsim detektēt. Detektēt vairāk par vienu kļūdu, ka arī izlabot ziņojumu mēs nespēsim, bet tas šajā piemērā arī nebija prasīts, tātad kodējuma mērķis šeit ir sasniegts.

Ko darīt, gadījumā ja ziņojuma atkārtota pārraide būs neiespējama, vai arī, ja mūsu dati, varētu tikt sabojāti nekārtīgas glabāšanas vai apstrādes dēļ, un kuriem nav paredzēta rezerves kopija?

Lai atbildētu uz šo pamatjautājumu jārealizē ziņojuma kodēšanas, un ***k*** kļūdas detektēšanas algoritmus izmantojot kontrolbitus.



*1.1. att. K***ļūdu koriģējošo kodu izmantošanas principi kļūdu labošanai.**

Šeit, kad ***k***=*1*, paša triviālākā metode, kuru var izdomāt uzreiz ir datu atkārtošana. Ja pirmais ziņojuma bits x_1 ir nulle, tad pārraidām aiz viņa vēl divas kontrolbitus nulles, pretējā gadījumā – ja pārraidāmais bits ir viens, tad pārraidām kopā trīs vieniniekus. Lai izlabotu kļūdu, šeit algoritms ir šāds: skatāmies, kādu bitu ir vairāk – nuļļu vai vieninieku. Šis kods spēj koriģēt tikai vienu kļūdu! Ja ***p*** ir *2*, tad tas iznāk **kļūdu koriģējošais** [***6***,***2***,***1***]**-kods**. To kodēšanu veicam tā:

$$X = \{x_1 x_2\} => (x_1, x_2) => \begin{cases} 00 \xrightarrow{kodejam\ ka} 000000 = x_1 y_1 y_2 x_2 y_3 y_4 = M_{00} \\ 01 \xrightarrow{kodejam\ ka} 000111 = x_1 y_1 y_2 x_2 y_3 y_4 = M_{01} \\ 10 \xrightarrow{kodejam\ ka} 111000 = x_1 y_1 y_2 x_2 y_3 y_4 = M_{10} \\ 11 \xrightarrow{kodejam\ ka} 111111 = x_1 y_1 y_2 x_2 y_3 y_4 = M_{11} \end{cases}$$

Un dekodējam pēc sekojošas shēmas, sadalot nokodēto ziņojumu divās daļās pa trim bitiem katrā:

$$[6,2,1] = (x_1 y_1 y_2 x_2 y_3 y_4) = \begin{cases} \text{Atkode vispirms pirma trijnieka dalu lai izlabotu } x_1: \\ 000... \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka kluda neparadijas un } x_1 = 0. \\ 001... \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } y_2 \text{ ir kludains bits un } x_1 = 0. \\ 010... \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } y_1 \text{ ir kludains bits un } x_1 = 0. \\ 011... \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } x_1 \text{ ir kludains bits un } x_1 = 1. \\ 100... \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } x_1 \text{ ir kludains bits un } x_1 = 0. \\ 101... \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } y_1 \text{ ir kludains bits un } x_1 = 1. \\ 110... \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } y_2 \text{ ir kludains bits un } x_1 = 1. \\ 111... \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka kluda neparadijas un } x_1 = 0. \\ \text{Pec tam atkode otro trijnieka dalu, lai izlabotu } x_2: \\ ...000 \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka kluda neparadijas un } x_2 = 0. \\ ...001 \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } y_4 \text{ ir kludains bits un } x_2 = 0. \\ ...010 \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } y_3 \text{ ir kludains bits un } x_2 = 0. \\ ...011 \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } x_2 \text{ ir kludains bits un } x_2 = 1. \\ ...100 \xrightarrow{dekodejam\ ka} \underline{000}, \text{konstatejot, ka } x_2 \text{ ir kludains bits un } x_2 = 0. \\ ...101 \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } y_3 \text{ ir kludains bits un } x_2 = 1. \\ ...110 \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka } y_4 \text{ ir kludains bits un } x_2 = 1. \\ ...111 \xrightarrow{dekodejam\ ka} \underline{111}, \text{konstatejot, ka kluda neparadijas un } x_2 = 0. \end{cases}$$

Eksistē daži citi kodēšanas paņēmieni, kuri nokodē ziņojumus kļūdu labošanai daudz ietilpīgāk. Par piemēru optimālākam nokodējumam varam apskatīt *Haminga **[7,4,1]-kodu**,* kurš, nokodējot četrus ziņojuma bitus, ir optimālāka garumā – 7. Tātad kods ir ar 3-im nosacījuma kontrolbitiem un šis kods spēj koriģēt sevī vienu kļūdu *(**k**=1)*:

$$M_{[7,4,1]} = x_1 x_2 x_3 y_1 x_4 y_2 y_3 \quad , kur: \begin{cases} y_1\ ir\ tads, ka: & x_1 + x_2 + x_3 + y_1 = 0 \pmod 2 \\ y_2\ ir\ tads, ka: & x_1 + x_2 + x_4 + y_2 = 0 \pmod 2 \\ y_3\ ir\ tads, ka: & x_1 + x_3 + x_4 + y_3 = 0 \pmod 2 \end{cases}$$

1.2. Stabilizatoru kodu pielietojumi un piemēri kvantu kļūdu korekcijai

Tagad pāriesim no klasiskās uz kvantu kodējumu.

Par **stabilizatoru** vai **kvantu kļūdu koriģējošo** [***n***,***p***,***k***]**-kodu** vai par ***M[n,p,k] virkni*** sauksim kvantu bitu *nokodēta ziņojuma* virkni garumā ***n***, ar ***p*** *ziņojuma bitiem*, kurā var koriģēt jebkuru kombināciju no ne vairāk kā ***k*** kļūdām.

Šeit, kvantu kodā, savukārt, katram bitam ir iespējamas vairāki kļūdas tipi. Tā kā klasiskā koda bitu ziņojumā pārraidījām tikai *nulles* un *vieniniekus*, tad uzzinot, ka pozīcijā parādās kļūda, vienkārši aizvietojam *0* ar *1* un *1* ar *0*, līdz ar ko izlabojam attiecīgo kļūdu. Tā kā kvantu bits vienmēr ir 2^p dimensiju vienības vektors (garumā viens, kur ***p*** *– ziņojuma bitu* skaits), tad tam ir iespējami vairāki stāvokļi. Piemēram, divdimensiju kvantu bits (vektors) ar vienu brīvo ziņojuma bitu ir attēlojams šādā veidā:

$$\alpha|0\rangle + \beta|1\rangle, \quad kur: \quad \begin{cases} \alpha, \beta \in [0;1]; \\ |\alpha|^2 + |\beta|^2 = 1. \end{cases}$$

No Kvantu Skaitļošanas kursā *[1.1.]* mēs jau pieradījām, ka jebkuru kvantu bitu pēc pārraides var sadalīt četros saskaitāmos:

- Koeficients reiz saskaitāmais, kurā ir notikusi ***I*** transformācijas tipa kļūda;
- Koeficients reiz saskaitāmais, kurā ir notikusi ***X*** transformācijas tipa kļūda;
- Koeficients reiz saskaitāmais, kurā ir notikusi ***Y*** transformācijas tipa kļūda;
- Koeficients reiz saskaitāmais, kurā ir notikusi ***Z*** transformācijas tipa kļūda;

Pierādījums šim apgalvojumam ir diezgan netriviāls, bet viņš eksistē, un to mēs pilnvaroti varam pieņemt ka patiesu. Lai saprastu principu, kā tas darbojas – apskatīsim un precīzi aprakstīsim īsu un skaistu piemēru *(lai uzreiz saprast piemērā izmanotas **I** un **Z** tranformācijas jēdzienus, uzsaku vispirms izlasīt tās definīcijas kuras šajā apakšnodaļā tīšuprāt ies mazliet tālāk)*, lai tas būtu gaiši saredzams:

> Dots kvantu stāvoklis $|\psi\rangle = \frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$. Pārraides rezultātā ir notikusi kļūda: spontāna bita mērījums uz 1. kvantu bita.
>
> *Pierādīt, ka šo kļūdu var izteikt, ka **I** un **Z** tipa kļūdas lineāro kombināciju.*

Pierādījums:

Mērot $|\psi\rangle = \frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$ kvantu stāvokli uz pirmā kvantu bita, iegūstot $|0\rangle$ dabūjam stāvokli $|\psi_0\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|010\rangle$, iegūstot $|1\rangle$ dabūjam kvantu stāvokli $|\psi_1\rangle = |100\rangle$.

Tiešām, tie abi varētu būt izsakāmi caur $|\psi\rangle$ kvantu stāvokļa $I_1|\psi\rangle = \frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$ un $Z_1|\psi\rangle = \frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle - \frac{1}{\sqrt{3}}|100\rangle$ kļūdu transformāciju lineāro kombināciju:

$$\begin{cases} |\psi_0\rangle = \alpha \cdot I_1|\psi\rangle + \beta \cdot Z_1|\psi\rangle = \alpha \cdot |\psi\rangle + \beta \cdot Z_1|\psi\rangle \\ |\psi_1\rangle = \gamma \cdot I_1|\psi\rangle + \delta \cdot Z_1|\psi\rangle = \gamma \cdot |\psi\rangle + \delta \cdot Z_1|\psi\rangle \end{cases}, \text{ kur: } \begin{cases} \alpha = \frac{\sqrt{3}}{2\sqrt{2}}; \\ \beta = \frac{\sqrt{3}}{2\sqrt{2}}; \\ \gamma = \frac{\sqrt{3}}{2}; \\ \delta = \left(-\frac{\sqrt{3}}{2}\right). \end{cases} \text{, jo:}$$

1) $|\psi_0\rangle = \alpha \cdot |\psi\rangle + \beta \cdot Z_1|\psi\rangle = \frac{\sqrt{3}}{2\sqrt{2}} \cdot \left(\frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle\right) + \frac{\sqrt{3}}{2\sqrt{2}} \cdot$

$\cdot \left(\frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle - \frac{1}{\sqrt{3}}|100\rangle\right) = \frac{1}{2\sqrt{2}}|000\rangle + \frac{1}{2\sqrt{2}}|010\rangle + \frac{1}{2\sqrt{2}}|100\rangle +$

$+ \frac{1}{2\sqrt{2}}|000\rangle + \frac{1}{2\sqrt{2}}|010\rangle - \frac{1}{2\sqrt{2}}|100\rangle = 2 \cdot \frac{1}{2\sqrt{2}}|000\rangle + 2 \cdot \frac{1}{2\sqrt{2}}|010\rangle =$

$= \underline{\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|010\rangle}$ (+);

2) $|\psi_1\rangle = \gamma \cdot |\psi\rangle + \delta \cdot Z_1|\psi\rangle = \frac{\sqrt{3}}{2} \cdot \left(\frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle\right) - \frac{\sqrt{3}}{2} \cdot$

$\cdot \left(\frac{1}{\sqrt{3}}|000\rangle + \frac{1}{\sqrt{3}}|010\rangle - \frac{1}{\sqrt{3}}|100\rangle\right) = \frac{1}{2}|000\rangle + \frac{1}{2}|010\rangle + \frac{1}{2}|100\rangle -$

$- \frac{1}{2}|000\rangle - \frac{1}{2}|010\rangle + \frac{1}{2}|100\rangle = 2 \cdot \frac{1}{2}|100\rangle = |100\rangle$ (+).

k.b.j. ■

Tagad formāli definēsim tās četras transformācijas, un skaidrības pēc, parādīsim kādā veidā tās ir pielietojamas kādam vektoram, un par ko šīs vektors transformējās gan ģeometriskā gan dimensiju komponenšu pāreju interpretācijā.

Dots kvantu stāvoklis $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$.

_I_ transformācija:

$\Rightarrow$ ***I*** *unitāra* transformācija nedara ar doto vektoru neko (mēs uzskatīsim, ka ja pēc kvantu bita pārraides pa sakaru kanālu viņā neierādījās kļūda, tad pārraides procesā bitam bija pielietota ***I*** transformācija).

Formāli transformāciju $I: \begin{cases} |0\rangle \to |0\rangle \\ |1\rangle \to |1\rangle \end{cases}$ ir jāpielieto šādi:

$$I|\varphi\rangle = I\left(\alpha|0\rangle + \beta|1\rangle\right) = \alpha|0\rangle + \beta|1\rangle = |\varphi\rangle.$$

_X_ transformācija:

$\Rightarrow$ ***X*** *unitāra* transformācija atspoguļo vektoru ap $|0\rangle = |1\rangle$ taisni, vai ja gribat, ap vienu no vektoriem $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ vai $-\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, kuri arī pieder šai taisnei.

Formāli transformāciju $X: \begin{cases} |0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle \end{cases}$ ir jāpielieto šādi:

$$X|\varphi\rangle = X\left(\alpha|0\rangle + \beta|1\rangle\right) = \beta|0\rangle + \alpha|1\rangle.$$

_Z_ transformācija:

$\Rightarrow$ ***Z*** *unitāra* transformācija atspoguļo vektoru ap $|1\rangle = 0$ taisni, vai ja gribat, ap vienu no vektoriem $1|0\rangle + 0|1\rangle = |0\rangle$ vai $-1|0\rangle + 0|1\rangle = -|0\rangle$, kuri arī pieder šai taisnei.

Formāli transformāciju $Z: \begin{cases} |0\rangle \to |0\rangle \\ |1\rangle \to -|1\rangle \end{cases}$ ir jāpielieto šādi:

$$Z|\varphi\rangle = Z\left(\alpha|0\rangle + \beta|1\rangle\right) = \alpha|0\rangle - \beta|1\rangle.$$

***Y* transformācija:**

Parasti Y transformāciju definē, ka: $Y:\begin{cases}|0\rangle \to -i|1\rangle \\ |1\rangle \to i|0\rangle\end{cases}$, bet lai nedarbotos ar kompleksiem skaitļiem, labāk to definēt nedaudz citādāk, jo uz kļūdas korekciju īpašībām tās neietekmēs.

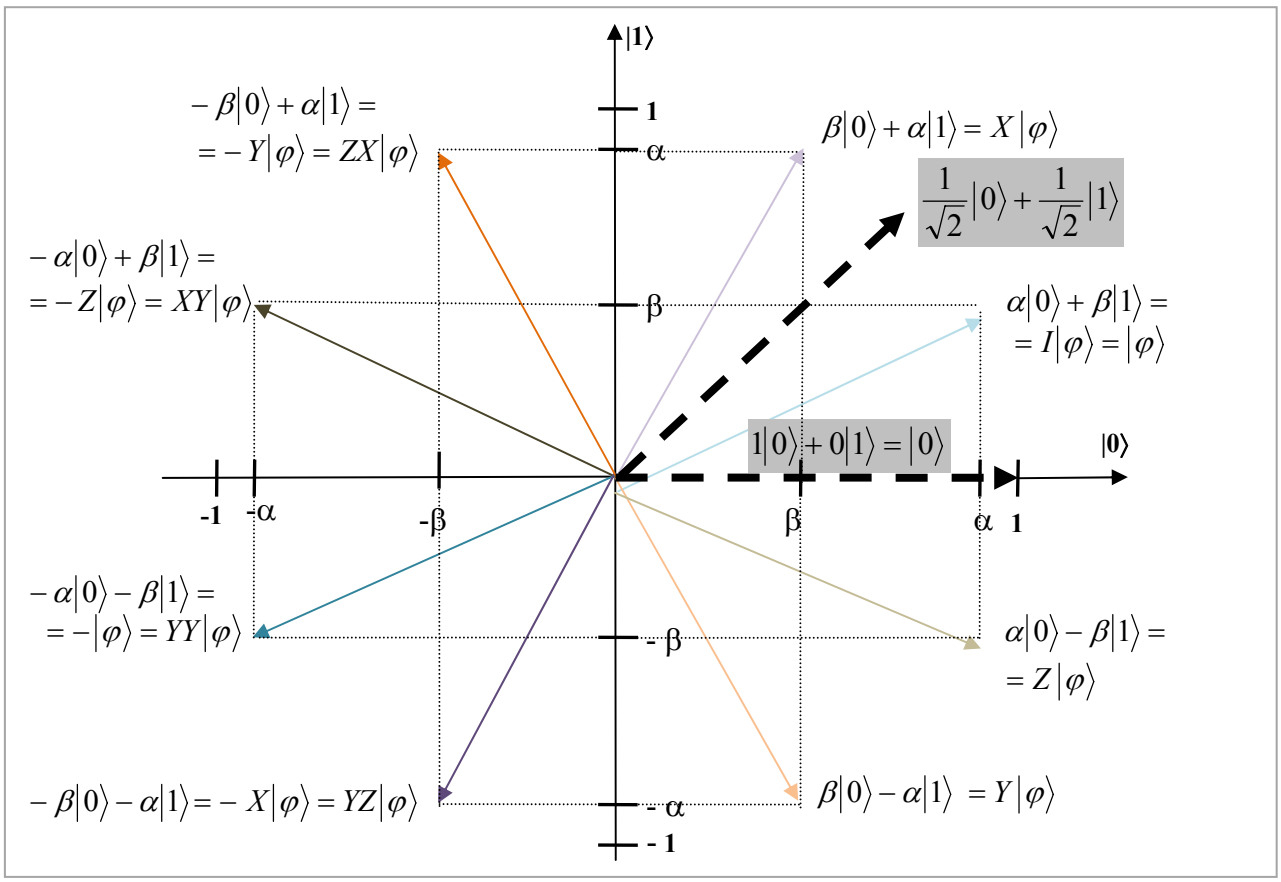
$\Rightarrow$ ***Y*** *unitāra* transformācija ir ***X*** un ***Z*** transformācijas secīga izpilde.

Formāli transformāciju $Y:\begin{cases}|0\rangle \to -|1\rangle \\ |1\rangle \to |0\rangle\end{cases}$ ir jāpielieto šādi:

$$Y|\varphi\rangle = ZX|\varphi\rangle = ZX\left(\alpha|0\rangle + \beta|1\rangle\right) = Z\left(X\left(\alpha|0\rangle + \beta|1\rangle\right)\right) = Z(\beta|0\rangle + \alpha|1\rangle) = \underline{\beta|0\rangle - \alpha|1\rangle}.$$

Attēlā *1.2.* ir parādīts kā mainās vektora izskats pēc transformācijas vai transformāciju kombinācijas secīgas pielietošanas. No šī zīmējuma saskatīsim dažas svarīgas īpašības:

$$XX|\varphi\rangle = I|\varphi\rangle = |\varphi\rangle;$$
$$XY|\varphi\rangle = XZX|\varphi\rangle = -ZXX|\varphi\rangle = -Z|\varphi\rangle;$$
$$XZ|\varphi\rangle = -ZX|\varphi\rangle = -Y|\varphi\rangle;$$
$$YX|\varphi\rangle = ZXX|\varphi\rangle = Z|\varphi\rangle$$
$$YY|\varphi\rangle = ZXY|\varphi\rangle = Z\left(-Z|\varphi\rangle\right) = -I|\varphi\rangle = -|\varphi\rangle;$$
$$YZ|\varphi\rangle = ZXZ|\varphi\rangle = -XZZ|\varphi\rangle = -X|\varphi\rangle;$$
$$ZX|\varphi\rangle = Y|\varphi\rangle;$$
$$ZY|\varphi\rangle = ZZX|\varphi\rangle = X|\varphi\rangle;$$
$$ZZ|\varphi\rangle = I|\varphi\rangle = |\varphi\rangle.$$



1.2. att. **Kvantu bita $|\varphi\rangle=\alpha|0\rangle+\beta|1\rangle$ unitāras transformācijas ģeometriska interpretācija. Visas transformācijas ir attēloti ka vienības vektori (garumā 1).**

Tātad, ja mēs pārraidām ***n*** kvantu bitus, tad visas iespējamās kļūdas ir attēlojamas, ka vektors, vai arī ir aprakstāmas vienas secīgās virknes veidā: $A_{M[n,p,k]}=A_1,A_2,...,A_n;\quad kur:\quad A_i\in\{I,X,Y,Z\}$. Pie tām, līdzīgi klasiskajam kodējumam (*skaties apakšnodaļu 1.1.: „Kļūdu koriģējošie kodi"*) var dažus bitus izmantot, ka nosacījuma bitus, atstāšot tiem tikai pārbaudes funkciju kvantu kļūdu meklēšanai. Arī līdzīgā veidā *pārbaudes veikšanai* katrs kvantu bits atsevišķi veic vienu y_i nosacījuma lomu, saistot ar sevi visus pārejos kvantu bitus. Pārbaudes virkne arī ir attēlojama virknes veidā: $B(y_i)=\left(B(y_i)_1,B(y_i)_2,...,B(y_i)_n\right);\quad kur:\quad B(y_i)_j\in\{I,X,Y,Z\}$.

Kvantu kontrolbitu *nosacījumu uzbūves pamatkoncepcija* ir šāda:

a) $B(y_i)_j$ var noķert tikai $A_{[n,p,k]j}$ kļūdu, pie tam tikai tad, ja $A_{[n,p,k]j} \neq B(y_i)_j$ un ja gan $A_{[n,p,k]j} \neq I$, gan $B(y_i)_j \neq I$.

b) Pārbaude $B(y_i)$ atgriež ***1***, ja pārbaudē noķertais kļūdas kopskaits ir pāra, un ***0***, ja nepāra.

c) Pārbaudes $B(y_i)$ un $B(y_t)$ ir savietojamas, ja, savā starpā, pārbaudot viens otru, abi atgriež ***0***.

d) Bitu pārbaudei ir jābūt savietojamai ar katru citu nosacījuma bita pārbaudi šajā ziņojumā.

Varam sīkāk apskatīt, ka piemēru, šādu ***1.2a uzdevumu***:

> Mums ir dots divu (***p***=*2*) kvantu bitu ziņojums kuru ir jāpārraida pa sakaru kanāliem. Mērķis pārraidīt nokodēto ziņojumu tā, lai gadījumā kad pēc pārraides ir parādījās viena kļūda (***k***=*1*), būtu iespēja **noķert kļūdas tipu**, ka arī pretējā gadījuma – atpazīt, ka kļūda neparādījās. Arī šī uzdevumu formulējumā ir svarīgi uzsvērt to, ka mēs nerūpēsimies par tiem gadījumiem, kad kļūdu skaits būs lielāks par ***k***.

Šī piemēra risinājums varētu būt sekojošs.

$X = \{x_1, x_2\}$; *Informācijas bitiem* pieliksim klāt vel no *nosacījumu bitu kopas* Y divus nosacījumu bitu elementus $Y = \{y_1, y_2\}$, līdz ar to, palielinot nokodēta ziņojuma elementu kopas garumu līdz četriem bitiem (***n***=*4*). No tiem elementiem veidosim nokodēto ziņojumu virkni tā:

$$M_{[4,2,0]} = x_1 x_2 y_1 y_2, \quad kurā: \begin{cases} y_1 \; ir \; parbaudes \; B(y_1) = XXXX \; atgriezta \; vertiba. \\ y_2 \; ir \; parbaudes \; B(y_2) = YYYY \; atgriezta \; vertiba. \end{cases}$$

Tiešām, šis nokodētais ziņojums noķer vienu kļūdas tipu ja tā parādās, ka arī pārbaudes $B(y_1)$ un $B(y_2)$ ir veiksmīgi savietojamas pēc *nosacījumu uzbūves pamatkoncepcijas*

definīcijas. Pēc y_1 un y_2 vērtību kombinācijas var atšķirt visus prasītus gadījumus. To nav grūti pierādīt ar pilno gadījumu pārlasi. Pārlases visi rezultāti ir apkopoti *tabulā 1.2a.*

1.2. tabula

1.2a uzdevuma visu kļūdu gadījumu pārlase ar B un C tipa pārbaužu atgriežamiem rezultātiem

Kļūdas gadījums	*$B(y_1)$=XXXX pārbaudes atgriežamā vērtība*	*$B(y_2)$=YYYY pārbaudes atgriežamā vērtība*	*Rezultāti ar secinājumiem.*
IIII (kļūdas nav)	*0*	*0*	$M_{[4,2,0]} = x_1x_2 00$: Ziņojumā nav kļūdu, vai kļūdu skaits ir >1.
XIII, IXII, IIXI, *vai* IIIX	*0*	*1*	$M_{[4,2,0]} = x_1x_2 01$: Ziņojumā ir viena X tipa kļūda, vai kļūdu skaits ir >1.
YIII, IYII, IIYI, *vai* IIIY	*1*	*0*	$M_{[4,2,0]} = x_1x_2 10$: Ziņojumā ir viena Y tipa kļūda, vai kļūdu skaits ir >1.
ZIII, IZII, IIZI, *vai* IIIZ	*1*	*1*	$M_{[4,2,0]} = x_1x_2 11$: Ziņojumā ir viena Z tipa kļūda, vai kļūdu skaits ir >1.
Citas permutācijas (XXII, XIXI, XIIX, IXXI, IXIX, IIXX, XYII, XIYI, XIIY, IXYI, IXIY, IIXY, XZII, XIZI, XIIZ, IXZI, IXIZ, IIXZ, YXII, YIXI,YIIX, IYXI, IYIX, IIYX, YYII, YIYI,YIIY, IYYI, IYIY, IIYY, YZII, YIZI,YIIZ, IYZI, IYIZ, IIYZ, ZXII, ZIXI, ZIIX, IZXI, IZIX, IIZX, ZYII, ZIYI, ZIIY, IZYI, IZIY, IIZY, ZZII, ZIZI, ZIIZ, IZZI, IZIZ, IIZZ, XXXI, ... , ZZZZ.) ($4^n - 3n - 1 = 243$ gadījumi)	*Nav svarīgi (jo **k>1**)*	*Nav svarīgi (jo **k>1**)*	*Ir pieļauts jebkurš no iepriekšminētiem rezultātiem.*

Noteikt kļūdas tipu, vai vispār kļūdas eksistenci ir ļoti noderīgi, ja ziņojumu atkārtota pārraide nav sarežģīta. Atkārtojot iepriekšējās nodaļas pamatjautājumu (***Ko darīt, gadījumā ja***

ziņojuma atkārtota pārraide būs neiespējama, vai arī, ja mūsu dati, varētu tikt sabojāti nekārtīgas glabāšanas vai apstrādes dēļ, un kuriem nav paredzēta rezerves kopija?) ir jānoformulē vēl vienu citu tipa uzdevumu ar dziļaku mērķi – ne tikai atrast kļūdas tipu, bet arī vietu, lai to varētu izlabot jau pēc pirmās ziņojuma pārraides.

Noformulēsim, ka piemēru, šādu ***1.2b uzdevumu***:

> Mums ir dots vienu (*p=1*) kvantu bitu ziņojums kuru ir jāpārraida pa sakaru kanāliem. Mērķis pārraidīt nokodēto ziņojumu tā, lai gadījumā kad pēc pārraides ir parādījās viena kļūda (*k=1*), būtu iespēja **noķert kļūdas tipu un vietu**, ka arī pretējā gadījuma – atpazīt, ka kļūda neparādījās. Arī šī uzdevumu formulējumā ir svarīgi uzsvērt to, ka mēs nerūpēsimies par tiem gadījumiem, kad kļūdu skaits būs lielāks par *k*.

Šī piemēra risinājums varētu būt sekojošs:

Realizējot ziņojuma kodēšanu, pamēģināsim izmantot iepriekšējā nodaļā aprakstītu *Haminga* metodi, kura viltīgi izmantojot kontrolbitus detektēja kļūdas vietu.

Tātad mums ir dots vienu ($p = 1$) kvantu bitu ziņojums $X = \{x_1\}$. Uzbūvēsim nokodēto ziņojumu M ar sešiem ($q = 6$) kontrolbitiem garumā septiņi $\left(|M| = n = 7\right)$ tādu ka:

$$M_{[7,1,1]} = x_1 y_1 y_2 y_3 y_4 y_5 y_6, \text{ kur: } \begin{cases} y_1 \; ir \; parbaudes \; B(y_1) = XXXXIII \; atgriezta \; vertiba \\ y_2 \; ir \; parbaudes \; B(y_2) = XXIIXXI \; atgriezta \; vertiba. \\ y_3 \; ir \; parbaudes \; B(y_3) = XIXIXIX \; atgriezta \; vertiba. \\ y_4 \; ir \; parbaudes \; B(y_4) = YYYYIII \; atgriezta \; vertiba. \\ y_5 \; ir \; parbaudes \; B(y_5) = YYIIYYI \; atgriezta \; vertiba. \\ y_6 \; ir \; parbaudes \; B(y_6) = YIYIYIY \; atgriezta \; vertiba. \end{cases}$$

Tiešām, šis nokodētais ziņojums noķer kļūdas tipu un vietu ja tā parādās, balstoties tikai uz pārraidāma ziņojuma kontrolbitu vērtībām. Visas 6 pārbaudes ir savietojamas, jo katrs no $\frac{q^2 - q}{2} = 15$ atbilstošiem pāriem:

1) $(B(y_1); B(y_2))$,

2) $(B(y_1); B(y_3))$,

3) $(B(y_1); B(y_4))$,

4) $(B(y_1); B(y_5)$,

5) $(B(y_1); B(y_6)$,

6) $(B(y_2); B(y_3)$,

7) $(B(y_2); B(y_4)$,

8) $(B(y_2); B(y_5)$,

9) $(B(y_2); B(y_6)$,

10) $(B(y_3); B(y_4)$,

11) $(B(y_3); B(y_5)$,

12) $(B(y_3); B(y_6)$,

13) $(B(y_4); B(y_5)$,

14) $(B(y_4); B(y_6)$,

15) $(B(y_5); B(y_6)$;

pēc *nosacījumu uzbūves pamatkoncepcijas* definīcijas, pārbaudot viens otru rezultātā atgriež *0*.

Pēc y_1, y_2, y_3, y_4, y_5 un y_6 vērtību kombinācijas var atšķirt visus vienas kļūdas gadījumus. To arī nav grūti pierādīt ar pilno gadījumu pārlasi. Pārlases visi rezultāti ir apkopoti attiecīgā *tabulā 1.2b.*

1.2b uzdevuma visu kļūdu gadījumu pārlases rezultāti.

№	*Saņemtais nokodētais ziņojums* $M_{[7,1,1]} = x_1 y_1 y_2 y_3 y_4 y_5 y_6$.	*Secinājumi.*
1	$x_1$000000	Ir **IIIIIII** tipa kļūda (ziņojumā nav kļūdu) vai kļūdu skaits ir >1.
2	$x_1$000111	Ir **XIIIIII** tipa kļūda vai kļūdu skaits ir >1.
3	$x_1$111000	Ir **YIIIIII** tipa kļūda vai kļūdu skaits ir >1.
4	$x_1$111111	Ir **ZIIIIII** tipa kļūda vai kļūdu skaits ir >1.
5	$x_1$000110	Ir **IXIIIII** tipa kļūda vai kļūdu skaits ir >1.
6	$x_1$110000	Ir **IYIIIII** tipa kļūda vai kļūdu skaits ir >1.
7	$x_1$110110	Ir **IZIIIII** tipa kļūda vai kļūdu skaits ir >1.
8	$x_1$000101	Ir **IIXIIII** tipa kļūda vai kļūdu skaits ir >1.
9	$x_1$101000	Ir **IIYIIII** tipa kļūda vai kļūdu skaits ir >1.
10	$x_1$101101	Ir **IIZIIII** tipa kļūda vai kļūdu skaits ir >1.
11	$x_1$000100	Ir **IIIXIII** tipa kļūda vai kļūdu skaits ir >1.
12	$x_1$100000	Ir **IIIYIII** tipa kļūda vai kļūdu skaits ir >1.
13	$x_1$100100	Ir **IIIZIII** tipa kļūda vai kļūdu skaits ir >1.
14	$x_1$000011	Ir **IIIIXII** tipa kļūda vai kļūdu skaits ir >1.
15	$x_1$011000	Ir **IIIIYII** tipa kļūda vai kļūdu skaits ir >1.
16	$x_1$011011	Ir **IIIIZII** tipa kļūda vai kļūdu skaits ir >1.
17	$x_1$000010	Ir **IIIIIXI** tipa kļūda vai kļūdu skaits ir >1.
18	$x_1$010000	Ir **IIIIIYI** tipa kļūda vai kļūdu skaits ir >1.
19	$x_1$010010	Ir **IIIIIZI** tipa kļūda vai kļūdu skaits ir >1.
20	$x_1$000001	Ir **IIIIIIX** tipa kļūda vai kļūdu skaits ir >1.
21	$x_1$001000	Ir **IIIIIIY** tipa kļūda vai kļūdu skaits ir >1.
22	$x_1$001001	Ir **IIIIIIZ** tipa kļūda vai kļūdu skaits ir >1.
23	Kāda cita vērtība	Kļūdu skaits ir >1.

1.3. Secinājumi, problēmas noformulējums un tālāki mērķī darbam

Uz šo brīdi ir izsecinātas šādas problēmas:

- Acīmredzami meklēt gan kļūdas pozīciju, gan kļūdas tipu smagi apgrūtina uzdevumu. To izdarīt ir daudz grūtāk nekā vienkārši detektēt kļūdas, ja tie vispār parādās ziņojumā.
- *Haminga* metode kvantu variantā jau strādā ne īpaši efektīvi, viņu var izmantot, bet tā nav spējīga dot optimālāko risinājumu, jo tā nav paredzēta vairākiem kļūdu tipiem.
- Nav skaidrs kādā veidā būvēt veiksmīgu savstarpēji savietojamu pārbaužu sistēmu, pēc kuras varētu risināt gan 1.2a, gan 1.2b tipa problēmas.

Ir nepieciešams teorētiskās izpētes mēģinājums. Jāpierāda kādas teorēmas un jāizmeklē labas īpašības stabilizatoru *[n,p,k]*-kodu veidošanai un pastāvībai. Teorētiska izpēte varētu palīdzēt izdomāt sistēmu, pēc kuras varētu atrisināt 1.2a un 1.2b tipa problēmas teorētisko daļu.

Ļoti iespējams, ka būs vērts izmantot šai problēmai datoru pārlases programmas meklēšanas iespēju, lai izpētīt un saskatīt kādas sakarības starp eksistējošiem *[n,p,k]*-kodiem, kad *n* ir samērā mazs. Tas palīdzēs produktīvāk un dziļāk veikt tālāku problēmas teorētisko izpēti vispārīgā *[n,p,k]* formā.

2. PROBLĒMAS TEORĒTISKA IZPĒTE

2.1 Teorijas uzbūvē

Nodefinēsim visus jēdzienus, kurus izmantosim:

Definīcija 1.1:

$|objekts| \Leftrightarrow$ *Objekta* garums.

Piemērs: $|\{1,3,5,10\}| = 4$

Definīcija 1.2:

$\langle objekts \rangle \Leftrightarrow$ Kopa ar visām iespējamam vērtībām, kurus var pieņemt šis *objekts*.

Piemērs: Ja ***a*** *ir definēts, ka kāds naturāls pāra skaitlis mazāks par 7, tad* $\langle a \rangle = \{0,2,4,6\}$.

Definīcija 2.1:

$K = \{I, X, Y, Z\}$ // K_I – Tās kvantu bitu transformācijas, kuras var nejauši izpildīties // bitam ziņojuma pārraides rezultātā.

Definīcija 2.2:

$K_I = \{I\}$. // K_I – Tās transformācijas, kuras rašanās nevarēs izraisīt kļūdu.

Definīcija 2.3:

$K_T = \{X, Y, Z\}$ // K_T – Tās transformācijas, kuras rašanās varēs izraisīt kļūdu.

Definīcija 3.1:

x_i // *i*-tais brīvs kvantu ziņojuma bits, no tiem, kurus mums ir // jāpārraida.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$ *un* $i = 1$, *tad* $x_i = x_1$.

Definīcija 3.2:

p // Visu brīvo kvantu bitu daudzums ziņojumā, kuru mērķis ir // pārraidīt pa sakaru kanālu.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$, *tad* $p = |X| = |\{x_1\}| = 1$.

Definīcija 3.3:

$X = \{x_1, x_2, ..., x_p\}$ // Kopa, kuru elementi ir brīvi ziņojuma biti; kurus mums ir
// jāpārraida.

Piemērs: Ja mums ir viens ziņojuma bits, kuru ir jāparaida $|\varphi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ *, tad* $X = \{x_1\}$, kur $x_1 = |\varphi\rangle$.

Definīcija 4.1:

y_i // ***i***-tais nokodēta ziņojuma kvantu kontrolbits, šo bitu vērtība
//vienmēr būs atkarīga no dažiem citiem bitiem nokodētā ziņojumā.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$ *un* $i = 4$, *tad* $y_i = y_4$.

Definīcija 4.2:

q // Visu kontrolbitu daudzums nokodētā ziņojumā.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$, *tad* $q = |Y| = |(y_1, y_2, y_3, y_4, y_5, y_6)| = 6$.

Definīcija 4.3:

$Y = \{y_1, y_2, ..., y_q\}$ // Kopa, kuru elementi ir visi nokodēta ziņojuma kontrolbiti.

Piemērs: Ja mums ir viens ziņojuma kontrolbits y_1*, kurš daļēji dublē ziņojuma pārējo bitu informāciju, un ja pārējie nokodēta ziņojuma biti ir brīvie biti, tad* $Y = \{y_1\}$.

Definīcija 5.1:

$M_{[n,p,k]} = (m_1, m_2, ..., m_n)$
// Nokodētais ziņojums, kas ir secīga elementu virkne garumā ***n***, ar ***p***
// *ziņojuma bitiem*, kurā var koriģēt ne vairāk par ***k*** kļūdām.
// Visi elementi kopā *M* ir pa vienu reizi ņemti no visiem kopas
// $X \cup Y$ elementiem.

Piemērs: $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$ *– ir nokodētais ziņojums, kurš spēj koriģēt vienu kļūdu.*

Definīcija 5.2:

m_i // ***i***-tais nokodēta ziņojuma virknes kvantu bits.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$ *un* $i = 6$, *tad* $m_i = y_5$.

Definīcija 5.3:

n // Visu nokodēta ziņojuma virknes kvantu bitu daudzums.

Piemērs: Ja $M_{[n,p,k]} = M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$, *tad* $n = |M_{[7,1,1]}| = |(x_1 y_1 y_2 y_3 y_4 y_5 y_6)| = 7$.

Definīcija 6.1.1:

$B(y_i)$ // Pārbaužu virkne ar kļūdas tipiem nokodētām ziņojumam. Šo
// pārbaudi realizē *pārbaudes bita* y_i nosacījums.

Piemērs: Ja $M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$, *un* y_4 *atgriež YYYYIII pārbaudes vērtību, tad* $B(y_4) = (Y, Y, Y, Y, I, I, I)$.

Definīcija 6.1.2:

$B(y_i)_j$ // ***j***-tais $B(y_i)$ virknes elements.

Piemērs: Ja $B(y_4) = (Y, X, Z, Y, I, Z, I)$, $i = 4$ *un* $j = 2$, *tad* $B(y_i)_j = B(y_4)_2 = X$.

Definīcija 6.2.1:

$B(Y)$ // Virkņu virkne, kuras elementi ir visas nokodēta ziņojuma tās
// pārbaužu virknes, kurus realizē šī nokodēta ziņojuma *pārbaudes*
// *bitu* y_i nosacījumi no kopas ***Y***, virknes ir sakārtotas pēc ***i*** augošā
// secībā.

Piemērs: Ja

$$M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6), \text{kur: } \begin{cases} y_1 \text{ ir parbaudes } B(y_1) = XXXXIII \text{ atgriezta vertiba} \\ y_2 \text{ ir parbaudes } B(y_2) = XXIIXXI \text{ atgriezta vertiba.} \\ y_3 \text{ ir parbaudes } B(y_3) = XIXIXIX \text{ atgriezta vertiba.} \\ y_4 \text{ ir parbaudes } B(y_4) = YYYYIII \text{ atgriezta vertiba.} \\ y_5 \text{ ir parbaudes } B(y_5) = YYIIYYI \text{ atgriezta vertiba.} \\ y_6 \text{ ir parbaudes } B(y_6) = YIYIYIY \text{ atgriezta vertiba.} \end{cases},$$

tad:

$$B(Y) = (B(y_1), B(y_2), B(y_3), B(y_4), B(y_5), B(y_6)) = ((X, X, X, X, I, I, I), (X, X, I, I, X, X, I),$$
$$(X, I, X, I, X, I, X), (Y, Y, Y, Y, I, I, I), (Y, Y, I, I, Y, Y, I), (Y, I, Y, I, Y, I, Y)).$$

Definīcija 6.2.2:

$B(Y)_i$ // ***i***-tā virknes $B(Y)$ virkne.

Piemērs: Ja $B(Y) = (B(y_1), B(y_2), B(y_3), B(y_4), B(y_5), B(y_6))$ *un* $i = 2$, *tad* $B(Y)_i = B(y_2)$.

Definīcija 7.1:

k // *Kļūdu skaits, kas pēc nokodēta ziņojuma pārraides varētu būt*
// ***izlabots*** (Gadījumā, ja kļūdas skaits nokodētā ziņojuma
// nepārsniedz ***k***, tad paša šī nokodēta ziņojuma virkne *M* varēs
// detektēt sevī, kļūdas skaitu, ka arī katru attiecīgo kļūdas gan vietu,
// gan tipu).

Piemērs: Ja

$M_{[7,1,1]} = (x_1, y_1, y_2, y_3, y_4, y_5, y_6)$, *kur:*

$$\begin{cases} y_1 \text{ ir parbaudes } B(y_1) = XXXXIII \text{ atgriezta vertiba} \\ y_2 \text{ ir parbaudes } B(y_2) = XXIIXXI \text{ atgriezta vertiba.} \\ y_3 \text{ ir parbaudes } B(y_3) = XIXIXIX \text{ atgriezta vertiba.} \\ y_4 \text{ ir parbaudes } B(y_4) = YYYYIII \text{ atgriezta vertiba.} \\ y_5 \text{ ir parbaudes } B(y_5) = YYIIYYI \text{ atgriezta vertiba.} \\ y_6 \text{ ir parbaudes } B(y_6) = YIYIYIY \text{ atgriezta vertiba.} \end{cases}, \text{ tad } M_{[n,p,k]} \text{ var koriģēt } k = 1 \text{ kļūdu.}$$

Definīcija 7.2:

d // *Kļūdu skaits, kas pēc nokodēta ziņojuma pārraides varētu būt*
// ***atpazīts*** (Paša virkne *M* ir spējīga detektēt savu kļūdu skaitu,
// ja tas nepārsniedz ***d***).

Piemērs:

Ja $M_{[m,p,k]} = M_{[4,2,0]} = (x_1, x_2, y_1, y_2)$, *kurā:*

$$\begin{cases} y_1 \text{ ir parbaudes } B(y_1) = XXXX \text{ atgriezta vertiba.} \\ y_2 \text{ ir parbaudes } B(y_2) = YYYY \text{ atgriezta vertiba.} \end{cases}, \text{ tad } M_{[n,p,k]} \text{ var atpazīt } d = 1 \text{ kļūdu.}$$

Definīcija 8.1:

A_i // Elements no kopas ***K***. Kļūdas tips ***i***-tam nokodēta ziņojuma bitam
// pēc viņa pārraides.

Piemērs:

Ja $i = 2$ *un pēc ziņojuma* $M_{[7,1,1]}$ *pārraides, otram bitam ir notikusi Y tipa kļūda, tad* $A_i = A_2 = Y$, kur $Y \in K$.

Definīcija 8.2:

$A_{M[n,p,k]}$ // Virkne ar kļūdas tipiem, kas secīgi apraksta visus nokodēta
// ziņojuma ***M*** kļūdas pēc viņa pārraides.

Piemērs: $A_{M[n,p,k]} = (I, I, I, X, I, I, Y)$ *nozīmē, ka n=7, un ka pēc ziņojuma* $M_{[7,1,1]}$ *pārraides, četrām bitam ir notikusi X tipa kļūda un pēdējām bitam ir notikusi Y tipa kļūda.*

Definīcija 8.3:

$A_{M[n,p,k]i}$ // Elements no kopas ***K***, kas ir ***i***-tais virknes $A_{M[n,p,k]}$ elements.

Piemērs: Ja $A_{M[n,p,k]} = (I, I, I, X, I, I, Y)$ *un* $i = 2$, *tad* $A_{M[n,p,k]i} = I$.

Definīcija 8.4:

$A_{M[n,p,k]}(r)$ // (Lai uzreiz saprast ***r*** jēdzienu, uzsaku vispirms izlasīt *12.Definīciju*
// kas šajā apakšnodaļā tīšuprāt ies tālāk) Virkne ar kļūdas tipiem,
// kas secīgi apraksta visus nokodēta ziņojuma ***M*** kļūdas pēc viņa
// pārraides. Kļūdu skaits ir nofiksēts un ir vienāds ar ***r***.

Piemērs: $A_{M[n,p,k]} = (I,I,I,X,I,I,Y)$ *nozīmē, ka n=7, un ka pēc ziņojuma* $M_{[7,1,1]}$ *pārraides ir notikušas tieši r=2 kļūdas – četrām bitam ir notikusi X tipa kļūda un pēdējām bitam ir notikusi Y tipa kļūda.*

Definīcija 8.5:

$A_{M[n,p,k]i}(r)$ // Elements no kopas ***K***, kas ir ***i***-tais virknes $A_{M[n,p,k]}(r)$ elements.

Piemērs: Ja $A_{M[n,p,k]}(2) = (I,I,I,X,I,I,Y)$ *un* $i = 5$, *tad* $A_{M[n,p,k]i}(r) = I$.

Definīcija 9.1:

$C_{M[n,p,k]}$ // Virkne ar $\boldsymbol{K_T}$ kļūdas tipa secīgu sarakstu nokodētā ziņojumā **M** pēc
// tās pārraides.

Piemērs: Ja $A_{M[n,p,k]} = (I,I,I,X,I,I,Y)$, *tad* $C_{M[n,p,k]} = (X,Y)$.

Definīcija 9.2:

$C_{M[n,p,k]_i}$ // Virknes $C_{M[n,p,k]}$ ***i***-tais elements.

Piemērs: Ja $C_{M[n,p,k]} = (X,Y)$ *un* $i = 2$, *tad* $C_{M[n,p,k]i} = Y$.

Definīcija 10.1:

$D_{M[n,p,k]}$ // Kopa, kuras elementi ir pozīcijas, kurās pēc nokodētā ziņojumā **M**
// pārraides ir notikusi kāda no $\boldsymbol{K_T}$ tipa kļūdām.

Piemērs: Ja $A_{M[n,p,k]} = (I,I,I,X,I,I,Y)$, *tad* $D_{M[n,p,k]} = \{4,7\} = \{7,4\}$.

Definīcija 10.2:

$D_{M[n,p,k]_i}$ // Kāds no $D_{M[n,p,k]}$ kopas elementiem.

Piemērs: Ja $D_{M[n,p,k]} = \{4,7\}$, *tad* $\begin{bmatrix} D_{M[n,p,k]_i} = 7 \\ D_{M[n,p,k]_i} = 4 \end{bmatrix}$.

Definīcija 11.1.1:

$A_{M[n,p,k]}(B(y_i))$ // Kopa, kura sastāv no tiem $\left\langle A_{M[n,p,k]} \right\rangle$ kopas elementiem, kurus var
// saķert $B(y_i)$ pārbaude.

Piemērs: Ja $B(y_i) = (I, X, Y)$, *tad*

$A_{M[n,p,k]}(B(y_i)) = \{$ *(I,I,X),(I,I,Z),(I,X,X),(I,X,Z),(I,Y,I),(I,Y,Y),(I,Z,I),(I,Z,Y),*
(X,I,I),(X,I,Y),(X,X,I),(X,X,Y),(X,Y,X),(X,Y,Z),(X,Z,X),(X,Z,Z),
(Y,I,I),(Y,I,Y),(Y,X,I),(Y,X,Y),(Y,Y,X),(Y,Y,Z),(Y,Z,X),(Y,Z,Z),
(Z,I,I),(Z,I,Y),(Z,X,I),(Z,X,Y),(Z,Y,X),(Z,Y,Z),(Z,Z,X),(Z,Z,Z) $\}$.

Definīcija 11.1.2:

$A_{M[n,p,k]}(B(y_i))_j$ // Jebkāds no $A_{M[n,p,k]}(B(y_i))$ kopas elementiem.

Definīcija 11.2.1:

$A_{M[n,p,k]}(\sim B(y_i))$ // Kopa, kura sastāv no tiem $\langle A_{M[n,p,k]} \rangle$ kopas elementiem, kurus
// nevar saķert $B(y_i)$ pārbaude.

Piemērs: Ja $B(y_i) = (I, X, Y)$, *tad*

$A_{M[n,p,k]}(\sim B(y_i)) = \{$ *(I,I,I),(I,I,Y),(I,X,I),(I,X,Y),(I,Y,X),(I,Y,Z),(I,Z,X),(I,Z,Z),*
(X,I,X),(X,I,Z),(X,X,X),(X,X,Z),(X,Y,I),(X,Y,Y),(X,Z,I),(X,Z,Y),
(Y,I,X),(Y,I,Z),(Y,X,X),(Y,X,Z),(Y,Y,I),(Y,Y,Y),(Y,Z,I),(Y,Z,Y),
(Z,I,X),(Z,I,Z),(Z,X,X),(Z,X,Z),(Z,Y,I),(Z,Y,Y),(Z,Z,I),(Z,Z,Y) $\}$.

Definīcija 11.2.2:

$A_{M[n,p,k]}(\sim B(y_i))_j$ *Definīcija 11.1.2:*

$A_{M[n,p,k]}(B(y_i))_j$ // Jebkāds no $A_{M[n,p,k]}(\sim B(y_i))$ kopas elementiem.

Definīcija 12:

r // K_T tipa kļūdu skaits, kas faktiski parādījās ziņojumā $M_{[m,p,k]}$ pēc
// pārraides.

Apskatīsim dažas īpašības, kurus var izvest tieši no definīcijām, lai tiešā veidā izmantot tos teorēmas pierādījumiem:

Īpašība 1:

$K_I \cap K_T = \{\}$. // Mēs pieņemsim, ka transformācijas never vienlaikus izraisīt
// un neizraisīt kļūdas.

Īpašība 2:

$K = K_I \cup K_T$. // Mēs pieņemsim, ka transformācijas ir tikai tādas, kuri vai nu
// izraisa vai nu neizraisa kļūdas.

Īpašība 2.1:

$\left|K\right| = 4.$ // Seko no 1.1. un 2.1. definīcijām.

Īpašība 2.2:

$\left|K_I\right| = 1.$ // Seko no 1.1. un 2.2. definīcijām.

Īpašība 2.3:

$\left|K_T\right| = 3.$ // Seko no 1.1. un 2.3. definīcijām.

Īpašība 3.1:

$\left|X\right| = p.$ // Seko no 1.1., 3.2. un 3.3. definīcijām.

Īpašība 3.2:

$\left|Y\right| = q.$ // Seko no 1.1., 4.3. un 4.2. definīcijām.

Īpašība 4:

$\left|M_{[\mathrm{m,p,k}]}\right| = n.$ // Seko no 1.1., 5.1. un 5.3. definīcijām.

Īpašība 5.1:

$\left|A_i\right| = 1.$ // Seko no 1.1 un 8.1. definīcijas.

Īpašība 5.2.1:

$\left|A_{M[n,p,k]i}\right| = 1.$ // Seko no 1.1. un 8.3. definīcijas.

Īpašība 5.2.2:

$\left|A_{M[n,p,k]}\right| = n.$ // Seko no 1.1., 5.3. un 8.2. definīcijām.

Īpašība 5.3:

$\left\langle A_{M[n,p,k]i}\right\rangle = K = \{I, X, Y, Z\}.$

// Seko no 1.2., 2.1. un 8.3. definīcijām.

Īpašība 5.4.1:

$\left|A_{M[n,p,k]i}(r)\right| = 1.$ // Seko no 1.1. un 8.5. definīcijas.

Īpašība 5.4.2:

$$\left|A_{M[n,p,k]}(r)\right| = n. \qquad \text{// Seko no 1.1., 5.3. un 8.4. definīcijām.}$$

Īpašība 5.5:

$$\left\langle A_{M[n,p,k]i}(r)\right\rangle = \begin{cases} \{I\} = K_I, & kad\ r = 0; \\ \{I, X, Y, Z\} = K, & kad\ r \in (0; n); \\ \{X, Y, Z\} = K_T, & kad\ r = n; \end{cases}$$

// Seko no 1.2., 2.1., 2.2., 2.3., un 8.5. definīcijām.

Īpašība 6.1:

$$\left|C_{M[n,p,k]i}\right| = 1. \qquad \text{// Seko no 1.1. un 9.2. definīcijām.}$$

Īpašība 6.2:

$$\left|C_{M[n,p,k]}\right| = r. \qquad \text{// Seko no 1.1., 9.1. un 12. definīcijām.}$$

Īpašība 6.3:

$$\left\langle C_{M[n,p,k]i}\right\rangle = K_T = \{X, Y, Z\}.$$

// Seko no 1.2., 2.3., 9.1. un 9.2. definīcijām.

Īpašība 7.1:

$$\left|D_{M[n,p,k]i}\right| = 1. \qquad \text{// Seko no 1.1. un 10.2. definīcijām.}$$

Īpašība 7.2:

$$\left|D_{M[n,p,k]}\right| = r \qquad \text{// Seko no 1.1., 10.1. un 12. definīcijām.}$$

Īpašība 7.2:

$$\left\langle D_{M[n,p,k]i}\right\rangle = \{1,2,3,...,n\}.$$

// Seko no 1.2., 10.1. un 10.2. definīcijām.

Īpašība 8.1.1:

$$\left|B(y_i)\right|_j = 1. \qquad \text{// Seko no 1.1. un 6.1.2. definīcijām.}$$

Īpašība 8.1.2:

$$\left|B(y_i)\right| = n. \qquad \text{// Seko no 1.1., 5.3. un 6.1.1. definīcijām.}$$

Īpašība 8.2.1:

$\left|B(Y)\right| = q$ // Seko no 1.1., 4.2. un 6.2.1. definīcijām.

Noderīgas īpašības un pierādījumi, kuri balstās uz definīcijām un uz citām īpašībām:

Īpašība 8.2.2:

$\left|B(Y)_i\right| = \left|B(y_i)\right| = n$ // Seko no 1.1., 6.2.2. definīcijām un 8.1.2 īpašības.

Īpašība 9.1:

$\left|\left\langle A_{M[n,p,k]i}\right\rangle\right| = \left|K\right| = 4.$ // Seko no 2.1. un 5.3. īpašībām.

Īpašība 9.2:

$\left|\left\langle B(y_i)\right\rangle\right| = \left|\left\langle A_{M[n,p,k]}\right\rangle\right| = \left|\left\langle A_{M[n,p,k]j}\right\rangle\right|^{\left|A_{M[n,p,k]}\right|} = \left|K\right|^n = 4^n$

// Seko no 1.1., 6.1.1, 8.2. definīcijām un 2.1., 5.2.2., 9.1. īpašībām,
// ka arī izmantojot kombinatorikas kursa materiālus, varam
// izsecināt, ka: ja visur definēta attēlojuma tips ir ***funkcija***, un ir
// jāatrod: cik iespējamos variantos var izvietot ***atšķiramas***
// $\left|A_{M[n,p,k]}\right|$ elementus, starp $\left|\left\langle A_{M[n,p,k]j}\right\rangle\right|$ ***atšķiramām*** pozīcijām,
// tad to skaits ir definēts, ka $\left|\left\langle A_{M[n,p,k]j}\right\rangle\right|^{\left|A_{M[n,p,k]}\right|}$.

.

Īpašība 10.1:

$\left|\left\langle C_{M[n,p,k]i}\right\rangle\right| = \left|K_T\right| = 3$. // Seko no 2.3. un 6.3. īpašībām.

Īpašība 10.2:

$\left|\left\langle C_{M[n,p,k]}\right\rangle\right| = \left|\left\langle C_{M[n,p,k]i}\right\rangle\right|^{\left|C_{M[n,p,k]}\right|} = \left|K_T\right|^r = 3^r$

// Seko no 1.1., 9.1. definīcijām un 2.3., 6.2., 10.1. īpašībām,
// ka arī izmantojot kombinatorikas kursa materiālus, varam
// izsecināt, ka: ja visur definēta attēlojuma tips ir ***funkcija***, un ir
// jāatrod: cik iespējamos variantos var izvietot ***atšķiramas***
// $\left|C_{M[n,p,k]}\right|$ elementus, starp $\left|\left\langle C_{M[n,p,k]i}\right\rangle\right|$ ***atšķiramām*** pozīcijām,
// tad to skaits ir definēts, ka $\left|\left\langle C_{M[n,p,k]i}\right\rangle\right|^{\left|C_{M[n,p,k]}\right|}$.

Īpašība 11.1:

$$\left|\left\langle D_{M[n,p,k]_i}\right\rangle\right| = \left|\{1,2,3,...,n\}\right| = n$$

// Seko no 7.2. īpašības.

Īpašība 11.2:

$$\left|\left\langle D_{M[n,p,k]}\right\rangle\right| = \binom{n}{r}.$$

// Seko no 1.1., 1.2., 10.1., 5.3., 12. definīcijām,
// ka arī izmantojot kombinatorikas kursa materiālus, varam
// izsecināt, ka: ja visur definēta attēlojuma tips ir ***injekcija***, un ir
// jāatrod: cik iespējamos variantos var izvietot ***r***, šajā gadījumā –
// ***neatšķiramas*** kļūdas, starp ***n atšķiramām*** pozīcijām. To
// skaits ir definēts, ka $\binom{n}{r} = \frac{n!}{(n-r)!r!}$.

Īpašība 12.1:

$$\left|\left\langle A_{M[n,p,k]_i}(r)\right\rangle\right| = \begin{cases} |K_I| = 1, & kad\ r = 0; \\ |K| = 4, & kad\ r \in (0;n); \\ |K_T| = 3, & kad\ r = n. \end{cases}.$$

// Seko no 2.1., 2.2., 2.3., 5.5. īpašībām.

Īpašība 12.2:

$$\left|\left\langle A_{M[n,p,k]}(r)\right\rangle\right| = \left|\left\langle D_{M[n,p,k]}\right\rangle\right| \cdot \left|\left\langle C_{M[n,p,k]}\right\rangle\right| = \binom{n}{r} \cdot 3^r$$

// Seko no 1.1., 1.2., 8.4. definīcijām un 11.2., 10.2. īpašībām,
// ka arī izmantojot kombinatorikas kursa materiālus, atceramies,
// ja lielumi $\left|\left\langle D_{M[n,p,k]}\right\rangle\right|$ un $\left|\left\langle C_{M[n,p,k]}\right\rangle\right|$ ir neatkarīgi viens no otra,
// tad, kombinējot tos kopā šajā tipa situācijās ir jāizmanto
// reizināšanas likumu.

2.2 Izmeklētas īpašības datorpārlasei un tās efektivitātes novērtējumi

No iepriekšējas nodaļas varam minēt, ka datoru pārlases ieguldījums šīs problēmas atrisināšanai varētu būt pavisam niecīgs. Es mēģināšu novērtēt nepieciešamu atmiņas apjomu un programmas sarežģītību novērtējot to, ka O-mēru caur ***n*** un ***k*** vērtībām. Ka arī nepieciešamu laiku, kas aizņems šī pārlase, izmantojot procesora resursus, kurus piedāvā mūsdienu vidējas statistikas datora mašīna.

Pieņemsim, ka mūsu programma meklējot ***[n,p,k]-kodus*** strādās pēc paša triviāla algoritma un apskatīsim šīs datoru pārlases shēmas pirmo soli:

- Izej cauri $\langle B(y_i)\rangle$ kopai un katram $B(y_i)$ saglabājam bināro masīvu garumā $n-i$, caur kuru mēs zināsim kādas pārbaudes ir savietojamas, un kuras nē. Lai to vērtību aprēķināt jāsalīdzina katru $B(y_i)_j$ pozīciju ar $B(y_{i+k})_j$ pozīciju ($i<k\le n$). Ja tās abas nav „I", vai atšķīrās, tad mainām masīva $i+k$ elementu uz pretējo.
  a. No tā, ka $|\langle B(y_i)\rangle| = 4^n$ [īpašība 9.2], un no tā, ka $|B(y_i)| = n$, izdarām secinājumu, ka atmiņas apjoms tas būs vajadzīgs ir $4^n \cdot n$ char-tipa simbolus.

  b. Pēc ātrdarbības šo daļu varam novērtēt tā: $\frac{1}{4}+\frac{2}{4}+\frac{3}{4}+\frac{5}{16}+\frac{6}{16}+\frac{7}{16}+\frac{9}{64}+\ldots=\sum_{i=1}^{3n}\left(\left\lceil\frac{4i-1}{3}\right\rceil-\frac{1}{4^{\left\lceil\frac{i}{3}\right\rceil}}\right)\approx 3{,}33$ rekursīvi salīdzinājumi vidējam $B(y_i)$ gadījumam lai mēs varētu pariet pie nākamā sekojoša kopas $\langle B(y_i)\rangle$ elementa (attiecīgas funkcijas koda piemērs c# valodā ir piedāvāts zemāk):

```
string increaseAVector(string givenVector, string largestVector)
{
      if (givenVector[givenVector.Length - 1] == 'I')
      {
            return givenVector.Remove(givenVector.Length - 1) + "X";
      }
      if (givenVector[givenVector.Length - 1] == 'X')
      {
            return givenVector.Remove(givenVector.Length - 1) + "Y";
      }
      if (givenVector[givenVector.Length - 1] == 'Y')
      {
            return givenVector.Remove(givenVector.Length - 1) + "Z";
      }
      if (givenVector != largestVector)
            return increaseAVector(givenVector.Remove(givenVector.Length-1),
                  largestVector) + "I";
      else
            return largestVector;
}
```

Pie tām, vēl katram no $|\langle B(y_i)\rangle| = 4^n$ ir jāizdara $|B(y_i)| = B(y_i)_j = n$ salīdzinājumi ar attiecīgu $B(y_{i+k})_j$ simbolu. Nav viegli saredzēt, ka vidējais

n salīdzinājumu skaits katram no 4^n elementiem ir $\frac{(n-1)+(n-2)+...+0}{n} = \frac{\left((n-1)^2-(n-1)\right)}{2n} = \frac{n^2-3n+2}{2n} \approx \frac{n^2-3n}{2n} =$ $= \frac{n-3}{2} \approx \frac{n}{2}$. Rezultātā dabujam $\frac{4^n \cdot n^2}{2}$ char tipa salīdzinājumus ar O-mēru: $O\left(n^2 \cdot 2^{2n-1}\right)$. Rūpji pieņēmot, ka salīdzinājuma operācija aizņem vienu procesora taktu frekvenci dabūjam, ka 2.7 GHz processoram tas aizņems $\frac{4^n \cdot n^2}{2 \cdot 2{,}7 \cdot 10^9}$ sekundes. Kas piemēram nozīmē, ka pie $n = 15$, programmas pirmais solis aizņem aptuveni 44 sekundes. Bet ar $n = 20$ programma darbosies 81445 sekundes, kas aptuveni mūsdienu datoram aizņems 23 stundas. Un tas ir tikai lai sagatavotu savietojamu pārbaužu grupas pirms koda meklēšanai! Šīs bēdīgs fakts liecina par to, ka datoru pārlase ir noderīga tikai samērā maziem nokodēta zinojuma garumiem.

Tālāk šajā nodaļā galvenokārt ies runa par to, kādus izmeklējumus es taisīju, lai pazemināt datoru pārlases darba laiku, iespējams dažos gadījumos caur izmantotas atmiņas palielināšanu. Vispirms ies īpašības, kas ekonomēs pārlases lauku par velti, kam sekos nopietnāki, kuri atmiņā glabās papildus datu strukturas.

Lai pamēģinātu paaugstināt datorpārlas efektivitāti, varam noformulēt dažas labas īpašības:

1. Apstrādājot $\langle B(y_i) \rangle$ kopas elementus noteikti ir vērts glabāt tos nevis, ka STRING tipa objektus, bet, ka skaitļus četrinieku skaitļošanas sistēmā. Tas uzlabos laiku vairāk par trīs reizēm, jo $\sum_{i=1}^{3n} \left(\left\lceil \frac{4i-1}{3} \right\rceil - \frac{1}{4^{\left\lceil \frac{i}{3} \right\rceil}} \right) \approx 3{,}3$ salīdzināšanu vietā būs jaizmanto tikai vienu aritmetisko summas operāciju – pieskaitot vieninieku dabūsim nākamo skaitli, kurš reprezentē $B(y_i)$.

   Piemērs: $B(y_i) = XXIYZIZ \rightarrow 1102303_4$.
   $B(y_{i+1}) \rightarrow 1102303_4 + 1_4 = 1102310_4 \rightarrow XXIYZXI$.

   (attiecīgas modificētas funkcijas koda piemērs c# valodā ir piedāvāts zemāk):

```
long increaseAVector2(long givenVector, long largestVector)
{
      if (givenVector != largestVector)
            return givenVector + 1;
      else
            return largestVector;
}
```

2.
   a. Nav jēgas pārlasīt tās pārbaudes sadalījumu grupas, kuri sastāv no ***n***, vai vairāku parbaužu skaita. Jo katra pārbaude $B(y_i)$ izmanto savu

unikālo y_i nosacījuma bitu nokodētā ziņojumā. Ja to skaits šajā ziņojumā ir ***n*** vai vairāk, tad izpildās $\forall j\left(j \in (1; p+q) \middle| (m_j \notin X) \cap (m_j \in Y)\right)$, kas liecina par to, ka nokodētā ziņojuma nebūs vietu nevienam no informāciju (brīvu) x_j bitam, un to pārraidei nebūs jēgas.

b. Nav jēgas pārlasīt tās pārbaudes sadalījumu grupas, kur sastāv no tik maza parbaužu skaita, ka visas tās atgriežamu rezultātu kombināciju daudzveidība $\left|\{0,1\}\right|^{|B(Y)|} = 2^q$ (*Īpašība 8.2.1*) ir nepietiekama (q bitus nepietiek), lai ar to nokodētu $\sum_{i=0}^{k}\left|\left\langle A_{M[n,p,k]}(i)\right\rangle\right| = \sum_{i=0}^{k}\left(\binom{n}{i}\cdot 3^i\right) = 1+\sum_{i=1}^{k}\left(\binom{n}{i}\cdot 3^i\right)$ (*Īpašība 8.2.1*), tātad kad kļūdas skaits ir no 0 līdz k.

Apvienojot 2.a. ar 2.b. dabūjam, ka $M_{[n,p,k]}$ neeksistē, ja neeksistē tāda q, ka:

$\begin{cases} q \le n-1 \\ 2^q \ge 1+\sum_{i=1}^{k}\left(\binom{n}{i}\cdot 3^i\right) \end{cases}$ No tā, kā $y = 2^x$ ir augoša funkcija, dabūjam, ka:

$\begin{cases} 2^q \le 2^{n-1} \\ 2^q \ge 1+\sum_{i=1}^{k}\left(\binom{n}{i}\cdot 3^i\right) \end{cases}$ vai: $2^{n-1} \ge 2^q \ge 1+\sum_{i=1}^{k}\left(\binom{n}{i}\cdot 3^i\right)$.

Dabujam gala nosacījumu caur n un k:

$2^{n-1} \ge 1+\sum_{i=1}^{k}\left(\binom{n}{i}\cdot 3^i\right)$ vai $2^{n-1}-1-\sum_{i=1}^{k}\frac{n!\cdot 3^i}{(n-i)!\,i!} \ge 0.$

Šīs svarīgas nevienadības rezultāti, dažiem pirmiem nofiksētiem ***n***, kurus varētu būt spējīga apstrādāt datorpārlase adekvātā laikā, ir apkopoti tabulā 2.2.

2.2. tabula

$2^{n-1}-1-\sum_{i=1}^{k}\frac{n!\cdot 3^i}{(n-i)!\,i!}$ **vērtības visiem *n*, kurus varētu būt apstrādātas ar datoru pārlasi uz mūsdienu mašīnas racionālā laikā** *(*$n=29$ *un* $k=1$ *gadījuma pārlases ilgums pārsniedz 3 gadus).*

n\k	1	2	3	4	5	6
1	-3	-∞	-∞	-∞	-∞	-∞
2	-5	-26	-∞	-∞	-∞	-∞
3	-6	-51	-186	-∞	-∞	-∞
4	-5	-83	-371	-1208	-∞	-∞
5	0	-120	-660	-2415	-7518	-∞
6	13	-158	-1076	-4505	-15035	-45896
7	42	-189	-1638	-7875	-28854	-91791
8	103	-197	-2357	-12995	-52847	-179207
9	228	-150	-3228	-20400	-92328	-338730
10	481	16	-4214	-30674	-154280	-616952
11	990	429	-5214	-44418	-247566	-1081542
12	2011	1345	-5999	-62186	-383108	-1827014
13	4056	3276	-6084	-84357	-574002	-2981160
14	8149	7246	-4472	-110879	-835505	-4712084
15	16338	15303	858	-140757	-1184766	-7235709
16	32719	31543	13975	-171029	-1640045	-10823501
17	65484	64158	43044	-194718	-2218908	-15809898
18	131017	129532	104422	-196736	-2934374	-22598420
19	262086	260433	230850	-145692	-3786966	-31664412
20	524227	522397	487837	22492	-4748570	-43550324
21	1048512	1046496	1006428	437349	-5729910	-58845336
22	2097085	2094874	2048740	1359403	-6515255	-78132944
23	4194234	4191819	4139034	3311241	-6631590	-101873739
24	8388535	8385907	8325859	7339657	-5086715	-130157843
25	16777140	16774290	16706340	15539940	154800	-162195930
26	33554353	33551272	33474754	32104450	13219948	-195286688
27	67108782	67105461	67019682	65419770	42424194	-222736434
28	134217643	134214073	134118313	132261010	104465536	-229682306
29	268435368	268431540	268325052	266180415	232812870	-184721880
30	536870821	536866726	536748736	534284581	494482963	-23154908
31	1073741730	1073737359	1073607066	1070788887	1023594642	386551431
32	2147483551	2147478895	2147335471	2144126359	2088477415	1309850983

3.

a. Definēsim aizvietojamas pārbaudes virknes jēdzienu:

*Par **savstarpēji aizvietojamajām pārbaužu virknēm** vai par **aizvietojamajām pārbaudēm** mēs sauksim* $B(y_i)$ *un* $B(y_j)$ *pārbaudes, ja eksistē bijekcija* $G(k_1) = k_2$, *kurā* $Dom(k_1) = Ran(G(k_1)) = K$ *un kura apraksta kā* K *kopa attēlojas uz sevi, tadā veidā, ka:*

$$\begin{cases} \forall a\left(a \in [1;n] \middle| G(B(y_i)_a) = B(y_j)_a\right); \\ G(I) = I. \end{cases}$$

Tātad nav jēgas pārlasīt sadalījumu grupas pārbaudei, ja kaut viena *savstarpēji aizvietojuma pārbaude* jau bija pārlasīta. Apskatīsim piemēru:

Piemērs:
n=5, k=1.
Pārbaudes $B(y_i) = (X, I, X, Z, Y)$ un $B(y_j) = (Z, I, Z, Y, X)$ ir *savstarpēji aizvietojamās*, un pārlasīt abus sadalījumus nav vērts, jo pārlasot $B(y_i)$ sadalījumu (skaties att. 2.2a), un uzbūvējot bijekciju G, tādu, ka: $\begin{cases} G(I) = I; \\ G(X) = Z; \\ G(Z) = Y; \\ G(Y) = X; \end{cases}$, varam secināt, ka, kamēr izpildās: $\begin{cases} \forall a\left(a \in [1;n] \middle| G(B(y_i)_a) = B(y_j)_a\right); \\ G(I) = I. \end{cases}$ mums ir iespēja nevis pārlasīt no jaunas $B(y_j) = (Z, I, Z, Y, X)$ pārbaudes sadalījumu, bet vienkārši katru $B(y_i) = (X, I, X, Z, Y)$ pārbaudes sadalījumā $B(y_i)_a$ elementu aizvietot ar $G(B(y_i)_a)$, dabūšot $B(y_j) = (Z, I, Z, Y, X)$ sadalījumu (skaties att. 2.2b).

$B(y_i)=(X,I,X,Z,Y)$	
$A_{M[n,p,k]}(B(y_i))$ kopas elementi	$A_{M[n,p,k]}(\sim B(y_i))$ kopas elementi
IIIIX	IIIII
IIIIZ	IIIIY
IIIXI	IIIZI
IIIYI	IIXII
IIYII	IXIII
IIZII	IYIII
YIIII	IZIII
ZIIII	XIIII

2.2a. att. **Sadalījums $B(y_i)=(X,I,X,Z,Y)$ pārbaudei uz tiem $k=1$ kļūdas gadījumiem, kurus tā spēj un kurus nespēj noķert.**

b. Definēsim simetriskas pārbaudes virknes jēdzienu:

Par ***savstarpēji simetrisku pārbaužu virknēm*** *vai par* ***simetriskiem pārbaudēm*** *mēs sauksim* $B(y_i)$ *un* $B(y_j)$ *pārbaudes, ja* $\forall a\left(a\in[1;n]\middle|B(y_i)_a=B(y_j)_{n-a}\right)$.

Tātad nav jēgas pārlasīt sadalījumu grupas pārbaudei, ja kaut viena *savstarpēji simetriska pārbaude* jau bija pārlasīta. Apskatīsim piemēru:

Piemērs:
n=5, k=1.
Pārbaudes $B(y_i)=(X,I,X,Z,Y)$ un $B(y_j)=(Y,Z,X,I,X)$ ir *savstarpēji simetriskas*, un pārlasīt abus sadalījumus nav vērts, jo pārlasot $B(y_i)$ sadalijumu (skaties att. 2.2a), varam secināt, ka, kamēr izpildās: $\forall a\left(a\in[1;n]\middle|B(y_i)_a=B(y_j)_{n-a}\right)$ mums ir iespēja nevis pārlasīt no jaunas $B(y_j)=(Y,Z,X,I,X)$ pārbaudes sadalījumu, bet vienkārši katru $B(y_i)=(X,I,X,Z,Y)$ pārbaudes sadalījumā $B(y_i)_a$ elementu aizvietot ar $B(y_i)_{n-a}$, dabūšot $B(y_j)=(Y,Z,X,I,X)$ sadalījumu (skaties att. 2.2c).

$B(y_j)=(Z,I,Z,Y,X)$	
$A_{M[n,p,k]}(B(y_j))$ kopas elementi	$A_{M[n,p,k]}(\sim B(y_j))$ kopas elementi
IIIIZ	IIIII
IIIIY	IIIIX
IIIZI	IIIYI
IIIXI	IIZII
IIXII	IZIII
IIYII	IXIII
XIIII	IYIII
YIIII	ZIIII

2.2b. att. **Sadalījums** $B(y_j)=(X,I,X,Z,Y)$ **par kuru parveršās sadalījums** $B(y_i)=(X,I,X,Z,Y)$ **ja tām pielieto aprakstīto bijekciju.**

$B(y_j)=(Y,Z,X,I,X)$	
$A_{M[n,p,k]}(B(y_j))$ kopas elementi	$A_{M[n,p,k]}(\sim B(y_j))$ kopas elementi
XIIII	IIIII
ZIIII	YIIII
IXIII	IZIII
IYIII	IIXII
IIYII	IIIXI
IIZII	IIIYI
IIIIY	IIIZI
IIIIZ	IIIIX

2.2c. att. **Sadalījums** $B(y_j)=(Y,Z,X,I,X)$ **par kuru parveršās sadalījums** $B(y_i)=(X,I,X,Z,Y)$ **ja tām pielieto aprakstīto** ***savstarpēji simetriskas pārbaudes*** **aizvietojumu.**

4. Ja tiks atrasts *kļūdu koriģējošais* ***[n,p,k]-kods***, piemēram:

$[7,1,1] = x_1 y_1 y_2 y_3 y_4 y_5 y_6$, kur: $\begin{cases} y_1 \text{ ir parbaudes } B(y_1) = XXXXIII \text{ atgriezta vertiba} \\ y_2 \text{ ir parbaudes } B(y_2) = XXIIXXI \text{ atgriezta vertiba.} \\ y_3 \text{ ir parbaudes } B(y_3) = XIXIXIX \text{ atgriezta vertiba.} \\ y_4 \text{ ir parbaudes } B(y_4) = YYYYIII \text{ atgriezta vertiba.} \\ y_5 \text{ ir parbaudes } B(y_5) = YYIIYYI \text{ atgriezta vertiba.} \\ y_6 \text{ ir parbaudes } B(y_6) = YIYIYIY \text{ atgriezta vertiba.} \end{cases}$,

tad nav jēgas papildus meklēt $[14,1,1]$, $[21,1,1]$, $[28,1,1]$ un vispār visus $[n,1,1]$ kodus, kur n dalās ar 7. To var pamatot ar to, ka jebkuru tādu ziņojumu mēs varam sadalīt blokos pa n, un kodēt tos atsevišķi. Acimredzami: Ja katrs bloks spēs noķert vienas kļūdas tipu un pozīciju, tad bloku apvienojumā viena kļūda arī neizbēgami būs saķerta tajā no blokiem, kurā tā parādīsies.

2.3 Secinājumi un izpētes rezultāti

Apkopojot visas īpašības vienā maisā, un ievērojot tās, varam pēc iespējas maksimāli atvieglot datorpārlases uzdevumus. Principā ir pavisam nedaudz gadijumus kurus mēs vienlaikus varam, un kurus ir vērts pārlasīt, dabūšot kaut kādus jaunus papildus rezultātus. Tik un tā, to ir prātīgi izdarīt. Līdz ar to varam vērsties pie datorpārlases programmas izstrādājuma.

3. DATORPĀRLASE

3.1 Programmatūras prasību specifikācijas un programmatūras projektējuma apraksta izvēlētās nodaļas

3.1.1 Programmatūras prasību specifikācija.

Dokumenta nolūks

Programmatūras prasību specifikācija (***PPS***) paredzēta izstrādājamās stabilizatoru kodu datoru pārlase programmatūras „Boriss_Benzerruki_Bakalaura_Darba_Pielikums" prasību aprakstīšanai.

Programmatūras prasību specifikācija paredzēta programmatūras projektējuma apraksta (***PPA***) izstrādāšanai, programmēšanai un testēšanai.

Darbības sfēra

Programmatūras produkts ir programma, kas paredzēta lietošanai uz datoriem ar operētājsistēmu Windows. Programma ir pielikta *Borisa Benzerruki (apliecības № bb06008)* 2010. gadā aizstāvētam Bakalaura darbam.

Produkta perspektīva

Produkts kalpos darba autoram un visiem ieinteresētiem meklēt stabilizatoru kodus gadījumus priekš nelieliem ***n*** un ***k***. Ka arī teorētisko īpašību, izmeklēšanai vispārīgam koda gadījumam. Visi [n, p, k] – kodi, kurus programma spēs pārlasīt, strādājot mazāk par vienu dienu, būs iekļauti Bakalaura darbā, kurai ir pielikta šī programma.

Ārējā saskarne

Lietotāja saskarne

Programma ir realizēta C# valodā, ar mērķi, lai lietotāja saskarne būtu pēc iespējas vienkāršāka un ērtāka lietošanai. Saskarne ar programmu notiek Windows vidē ar Windows

formas palīdzību. No galvenās saskarnes jābūt pieejamām visām, atļautām uz attiecīgo brīdi, programmas funkcijām.

Aparatūras saskarne

Lietotājs visu programmas kontroli veic tikai ar peles palīdzību.

Programmatūras saskarne

Programmatūras lietošanai operētājsistēmā Windows 98/NT/2000/XP ir nepieciešams *Microsoft .NET Framework Version 2.0*, ko var iegūt pastāvīgā mājas lapā:

http://www.microsoft.com/downloads/.

Veiktspējas prasības

Programmatūras produktam nav jāatbalsta vairāklietotāju režīms. Vienlaikus ar programmatūru jāvar netraucēti strādāt operētājsistēmas procesiem.

3.1.2 Programmatūras projektējuma apraksts.

Dokumenta nolūks

Programmatūras prasību specifikācija (***PPS***) paredzēta izstrādājamās stabilizatoru kodu datoru pārlase programmatūras „Boriss_Benzerruki_Bakalaura_Darba_Pielikums" prasību aprakstīšanai.

Programmatūras prasību specifikācija paredzēta programmatūras projektējuma apraksta (***PPA***) izstrādāšanai, programmēšanai un testēšanai.

Darbības sfēra

Programmatūras produkts ir programma, kas paredzēta lietošanai uz datoriem ar operētājsistēmu Windows. Programma ir pielikta *Borisa Benzerruki (apliecības № bb06008)* 2010. gadā aizstāvētam Bakalaura darbam.

Produkta funkcijas

- Galvenā produkta funkcija ir meklēt stabilizatoru [n, p, k] – kodu caur datoru pārlasi.
- Dot lietotājam iespēju izvēlēties sev vēlamo [n, p, k] – kodu. Savu izvēli lietotājam jānorāda caur diviem NumericUpDown kontroļu izvēlnēm. Ņemot vērā šos divas nofiksētas ***n*** un ***k*** vērtības programmai būs jāizdara [n, p, k] – kodu pārlasi.
- Lietotājam jābūt skaidrs caur kuru NumericUpDown kontroli ir jānorāda ***k*** un caur kuru ir jānorāda ***n*** vērtības.
- Programmai jāsāk darboties tad un tikai tad, ja lietotājs nospiež pogu „Aiziet".
- Lietotājam jābūt iespēja nostartēt datoru pārlasi tikai tad, kad šīs programmas darbības ietvaros nav nostartēta cita datoru pārlase.
- Programmai nav jārūpējas par tiem gadījumiem, kuri jau ir izmeklēti Bakalaura darba ietvaros, lietotājam jābūt iespēja pārlasīt jau zināmus rezultātus pat tad, ja:
    - Pieprasītam [n, p, k] gadījumam ir jau zināms atrisinājums, un ir iekļauts šī Bakalaura darbā vai citos avotos.
    - Bakalaura darbā bija pierādīts, ka kods pieprasītam [n, p, k] gadījumam neeksistē, par to programmai nav jāpaziņo lietotājam iepriekš.
- Pēc veiksmīgas datoru pārlases pabeigšanas ir jānorāda datoru pārlases sasniegtos rezultātus, ka arī svarīgus starprezultātus caur TreeView logu.
- Jādot lietotājam pabeigt datoru pārlases programmas darbību, nospiežot pogu „Iziet".

Vispārējie ierobežojumi

Programmatūras izstrādes valoda ir *C#*, izstrādes vide ir *Microsoft Visual Studio.NET 2005*. Izstrādātā programmatūra ir paredzēta lietošanai *Windows 98/NT/2000/XP* operētājsistēmas vidē, ar instalētu *.NET Framework 2.0*.

3.2 Datorpārlases mērķi

Tā, kā jau tika pieminēts, datoru pārlases laiks pat nelieliem ***n*** joprojām paliek neizbēgami lēns, datorpārlaeses rezultāti nedos īpaši lielus sasniegumus. Datoru pārlases mērķi ir izmantot 2. nodaļas pierādītās īpašības, izdarīt jaunu rezultātu atrašanas mēģinājumu.

3.3 Datorpārlases secinājumi un sasniegto rezultātu apkopums.

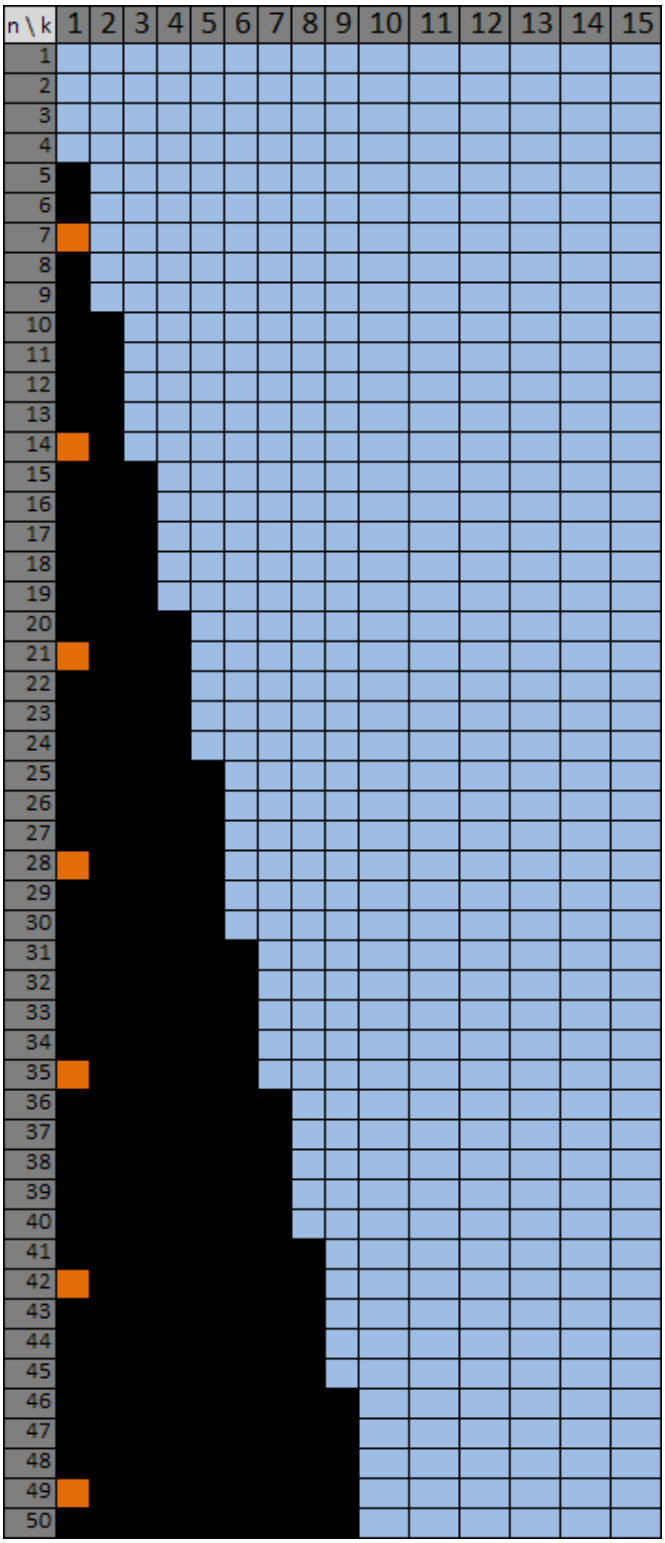
Datorpārlases laikā tika aprēķināti un izdoti starprezultāti, balstoties uz pierādītām īpašībām. Gadījumiem, kad kļūdu skaits ir ***k***=*1*, un ***n*** ir mazāks par 9, programma spēj aprēķināt kopas ar savstarpēji savietojamam grupām, kas ir galvenais starprezultāts, pirms pēdēja pārlases soļa, kurā tiks pārlasīta katra atrastā grupa. Gadījumā, kad n ir 9, programma nepaspēj pabeigt darbu pirms pēdēja soļa, strādājot uz 2.7 GHz procesora datora, vienu dienu laikā. Veiksmīga datoru pārlases mēģinājuma mērķi ir nesasniegti, jo grupu skaits ir krietni liels, jo pat ***n=5*** un ***k=1*** gadījumam atsevišķi uzrakstīta programma nepaspēj pabeigt darbu uz, tai iepriekš noradītajā, atļautajā vienas dienas laikā.

Datorpārlases programmas uzrakstīšanas laikā tika izdomāta īpašība ***n=5*** un ***k=1*** gadījumam, kura izlaboja programmas darbību konstantes laikā, bet pat tad nepalīdzēja atrast vēlamo atrisinājumu. Taisīt jaunu rezultātu atrašanas mēģinājumu uz šo brīdi citiem gadījumiem pagaidām nav vērts.

REZULTĀTI

Datorpārlase mums neatklāja nekādus jaunus nezināmus rezultātus. Līdz ar to, izmantot 2. teorētiskās nodaļas pierādītās īpašības, varam ar to izdarīt pilno pārskatu ar darbā sasniegtiem rezultātiem. Tabulā 3.2 ir apkopoti līdz šim brīdim (1. un 2. nodaļā) pierādītie rezultāti pirmajiem ***n*** un ***k***.

3.2. tabula



Zināmie pamatuzdevuma atrisinājumi.

Kur tabulā:

- *tie n un k gadījumi, kuriem [n ,p, k]-kods ir atrasts.*

- *tie n un k gadījumi, kuriem ir pierādīta [n, p, k]-koda nepastāvēšana.*

- *tie n un k gadījumi, kuriem uz šo brīdi atrisinājums mums nav zināms.*

SECINĀJUMI

- ✓ Stabilizatoru ***[n, p, k] – kodi*** var eksistēt tikai tiem ***n*** un ***k***, kuriem izpildās nevienādība: $2^{n-1} - 1 - \sum_{i=1}^{k} \frac{n! \cdot 3^i}{(n-i)!\, i!} \geq 0.$
- ✓ Ja ir atrasts *kļūdu koriģējošais* ***[n, p, k] – kods***, tad nav jēgas papildus meklēt tos ***[n_1, p, k] – kodus***, kuriem ***n_1*** dalās ar ***n*** bez atlikuma. Visi tādi ***[n_1, p, k] – kodi*** arī eksistē. Un pie tam, tos var uzbūvēt pa ***n_1 div n*** blokiem un katrā tā ***n****-elementu* blokā tie būvējas kā ***[n, p, k] – kods.***
- ✓ Sekojošo $M_{[7,1,1]}$ ziņojumu, kuru izdevās uzbūvēt, izmantojot klasisko *Haminga* ***[7,4,1]-koda*** principu, varam uzskatīt par stabilizatoru *[7,1,1]* – ***kodu***, jo tajā ir 7 nokodētā ziņojuma biti, 1 brīvais bits un tas spēj noķert katru viena kļūdas rašanas gadījumu, noķerot gan tās kļūdas vietu, gan tās tipu:

$$M_{[7,1,1]} = x_1 y_1 y_2 y_3 y_4 y_5 y_6, \text{ kurā: } \begin{cases} y_1 \text{ ir parbaudes } B(y_1) = XXXXIII \text{ atgriezta vertiba} \\ y_2 \text{ ir parbaudes } B(y_2) = XXIIXXI \text{ atgriezta vertiba.} \\ y_3 \text{ ir parbaudes } B(y_3) = XIXIXIX \text{ atgriezta vertiba.} \\ y_4 \text{ ir parbaudes } B(y_4) = YYYYIII \text{ atgriezta vertiba.} \\ y_5 \text{ ir parbaudes } B(y_5) = YYIIYYI \text{ atgriezta vertiba.} \\ y_6 \text{ ir parbaudes } B(y_6) = YIYIYIY \text{ atgriezta vertiba.} \end{cases}$$

Apkopojot visu paveikto, varam izsecināt, ka teorētisko uzdevumu pētījums, ka arī parasti, ir daudz nopietnāka ierīce problēmas atrisināšanai, nekā pilno variantu pārlase. Līdz ar to nav jēgas pievērst tam tik lielu uzmanību, kā tika izdarīts šajā darbā. Daudz produktīvāk ir apskatīt vairāk teorētisko pētījumu, kas jau tika darīts šajā jomā, lai kooperētos ar citiem prāta biedriem, kuri laužot galvu, risinot šo problēmu vispārīgā formā. Atrast tos cilvēkus un dalīties ar saviem maziem sasniegumiem, lai kopsummā būt spējīgiem virzīt zinātni uz priekšu.

Katra īpašība, kas tika pieradīta šajā darbā, ir patiešām ļoti svarīga, bet sasniegtais nav pietiekams, lai tikt galā ar šo problēmu. Ne katra tēma, kas skan viegli ir patiesībā tik viegla, ka mēs to sakumā iedomājamies.

Runājot par tālākiem mērķiem un virzieniem, gribētos izpētīt tos gadījumus, piemēram, kurās kļūdas, ja tie parādās, parādās nokodētā ziņojuma tieši pēc kārtas. Tiem gadījumiem arī būs atrasti dzīves pielietojumi, jo kļūdas parasti arī parādās, ka viens kļūdas nepārtraukts bloks. Vēl var ciešāk pievērst uzmanību *2.1 tipa uzdevumam* (*skaties Nodaļu **2.1**),* kad kļūdas ir jādetektē tikai pēc vietām kuros tie parādās, nedetektējot kļūdas tipu. Ja ziņojuma atkārtota pārraide nav apgrūtināta, tad to arī būs vērts pielietot dzīvē.

Arī varētu būt vērts turpināt ar šo vispārīgo pamat gadījumu, pētot to tālāk, dabūšot kaut kādus jaunus papildus rezultātus.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. **Žurnāli**

1.1 **Борн Д.,** *Квантовый компьютер справился с химическими вычислениями* [tiešsaiste]. 3D News: 3DNews Daily Digital Digest, 2010 – [atsauce 13.01.2010.]. Pieejams internetā: http://www.3dnews.ru/news/kvantovii_komputer_spravilsya_s_himicheskimi_vichisleniyami/.

2. **Raksti grāmatās**

2.1 **P. W. Shor,** Algorithms for quantum computation: discrete log and factoring. Proceedings of the 35th Annual Symposium on the Foundations of Computer Science. *IEEE Computer Society Press*, 1994, Los Alamitos, CA.

2.2 **Grover, L. K.** *A fast quantum mechanical algorithm for database search.Analysis.* Proceedings of the 28th Annual Symposium on the Theory of Computing: ACM Press. New York, 1996.

2.3 **J. Preskill.** *Fault-tolerant quantum computation.* Nodaļa grāmatā *"Introduction to Quantum Computation"* (redaktori: ***H.-K. Lo***, ***S. Popescu*** un ***T. P. Spiller***) Pieejams internetā: http://arxiv.org/abs/quant-ph/9712048

2.4 **P. W. Shor,** *Scheme for reducing decoherence in quantum computer memory*, Phys. Rev. A 52, lpp. 2493-2496, 1995g.

2.5 **A. Calderbank, E. Rains, P. Shor,** and **N. Sloane**, *„Quantum error correction via codes over GF(4),"* IEEE Trans. Inf. Theory, vol. 44, pp. 1369–1387, 1998g.

3. **Elektroniskie informācijas avoti**

3.1 From Wikipedia, the free encyclopedia [tiešsaiste]. Wikimedia Foundation, Inc., [atsauce 24.5.2010]. Pieejams: http://en.wikipedia.org/wiki/Gordon_Moore.

3.2 From Wikipedia, the free encyclopedia [tiešsaiste]. Wikimedia Foundation, Inc., [atsauce 8.11.208]. Pieejams: http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2008.svg.

3.3 From Wikipedia, the free encyclopedia [tiešsaiste]. Wikimedia Foundation, Inc., [atsauce 26.5.2010]. Pieejams: http://en.wikipedia.org/wiki/Richard_Feynman.

3.4 Материал из Википедии — свободной энциклопедии [tiešsaiste]. Wikimedia Foundation, Inc., [atsauce 17.3.2010]. Pieejams:http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%BE%D1%80%D0%B0.

4. **Zinātniskās prezentācijas.**

4.1 Matemātika programmētāju dzīvē. prezentācija: **Ambainis, A**. „Kvantu skaitļošana". LU, Rīgā : Datorikas fakultāte, *Ādas Lavleisas prēmijas atklātais seminārs*, 2008.)

PIELIKUMI

1. pielikums. Programmatūras koda izvēlētie resursu faili.

Kopumā Form1.cs, „Boriss_Benzerruki_Bakalaura_Darba_Pielikums v.0.5” izstrādātas programmatūras pirmkoda fails, ir aptuveni 600 rindu garš. Un „5_1_case v0.1” - ir aptuveni 400 rindu garš.

Šajā sadaļā iekļauti viss programmatūras kods, izņemot automātiski saģenerēto Visual Studio.Net 2005 kodu, kas apraksta vienkāršos kontroļu noklusēto formas inicializāciju.

Kods ar komentāriem

Form1.cs

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

namespace Boriss_Benzerruki_Bakalaura_Darba_Pielikums
{
    public partial class Form1 : Form
    {
        // This block is used to load controls on the form.
        public Form1()
        {
            InitializeComponent();
        }

        //Some global lists is neaded for list collection preset in some functions.
        List<int> tempList = new List<int>();
        List<int> tempList2 = new List<int>();
        List<int> tempList3 = new List<int>();
        List<int> smartFunctionTempList = new List<int>();

        // Checkes with theese orders was already included in smart function list.
        List<long> alreadyIncludedIndexes = new List<long>();

        //How much stabilizing codes was found.
        int discovered = 0;

        // This block is used to set next vector value. For example:
        // IIX-> IIY, XYZ->XZI, IZZ->XII
        List<int> nextListAfter(List<int> givenList)
        {
            List<int> anotherList = new List<int>();
            if (givenList.Count != 0)
            {
                if (givenList[givenList.Count - 1] != 3)
                {
                    for (int i = 0; i < givenList.Count - 1; i++)
                    {
                        anotherList.Add(givenList[i]);
                    }

                    anotherList.Add(givenList[givenList.Count - 1] + 1);
                    return anotherList;
                }
                else
                {

                    for (int i = 0; i < givenList.Count - 1; i++)
                    {
                        anotherList.Add(givenList[i]);
```

```
                }
                anotherList = nextListAfter(anotherList);
                anotherList.Add(0);
                return anotherList;
            }
        }
        return anotherList;
    }

    // This block is used to convert Generic List<int> data elemets to string.
    string convertAListToString(List<int> givenList)
    {
        string result = "(";
        for (int i = 0; i < givenList.Count; i++)
        {
            if (givenList[i] == 0) result += "I";
            if (givenList[i] == 1) result += "X";
            if (givenList[i] == 2) result += "Y";
            if (givenList[i] == 3) result += "Z";
            if (i < givenList.Count-1) result += ",";
        }
        result += ")";
        return result;
    }

    // This block notify if list corresponding elements are equal.
    bool sameLists(List<int> list1, List<int> list2)
    {
        if (list1.Count != list2.Count) return false;
        for (int i = 0; i < list1.Count; i++)
        {
            if (list1[i] != list2[i]) return false;
        }
        return true;
    }

    //This block returns the index of correction candidat. For example:
    //IIX->[0], IIY->[1], IIZ->[2], XIXZI->[(1*4^4)+(1*4^2)+(3*4^1)+1=285]
    long indexOf(List<int> currentList)
    {
        long result = 0;
        long FourFlour = 1;
        for (int i = currentList.Count - 1; i >= 0; i--)
        {
            result += currentList[i] * FourFlour;
            FourFlour *= 4;
        }
        return result-1;
    }

    //This function returns 2^parameter value.
    int twoExp(int parameter)
    {
        int result = 1;
        for (int i = 0; i < parameter; i++)
        {
            result *= 2;
        }
        return result;
    }

    // This bijection type transformes given list elements as: G(I)=I; G(X)=Y; G(Y)=Z;
G(Z)=X;
    List<int> doBijection1(List<int> GivenList)
    {
        for (int i = 0; i < GivenList.Count; i++)
        {
            switch (GivenList[i])
            {
                case 0:
                    tempList[i] = 0;
                    break;
                case 1:
                    tempList[i] = 2;
                    break;
                case 2:
                    tempList[i] = 3;
                    break;
                case 3:
```

```
                tempList[i] = 1;
                break;
            }
        }
        return tempList;
    }

    // This bijection type transformes given list elements: G(I)=I; G(X)=Z; G(Y)=X; G(Z)=Y;
    List<int> doBijection2(List<int> GivenList)
    {
         for (int i = 0; i < GivenList.Count; i++)
         {
             switch (GivenList[i])
             {
                 case 0:
                     tempList[i] = 0;
                     break;
                 case 1:
                     tempList[i] = 3;
                     break;
                 case 2:
                     tempList[i] = 1;
                     break;
                 case 3:
                     tempList[i] = 2;
                     break;
             }
         }
         return tempList;
    }

    // This bijection type transformes given list elements: G(I)=I; G(X)=Y; G(Y)=X; G(Z)=Z;
    List<int> doBijection3(List<int> GivenList)
    {
         for (int i = 0; i < GivenList.Count; i++)
         {
             switch (GivenList[i])
             {

                 case 0:
                     tempList[i] = 0;
                     break;
                 case 1:
                     tempList[i] = 2;
                     break;
                 case 2:
                     tempList[i] = 1;
                     break;
                 case 3:
                     tempList[i] = 3;
                     break;
             }
         }
         return tempList;
    }

    // This bijection type transformes given list elements: G(I)=I; G(X)=Z; G(Y)=Y; G(Z)=X;
    List<int> doBijection4(List<int> GivenList)
    {
         for (int i = 0; i < GivenList.Count; i++)
         {
             switch (GivenList[i])
             {

                 case 0:
                     tempList[i] = 0;
                     break;
                 case 1:
                     tempList[i] = 3;
                     break;
                 case 2:
                     tempList[i] = 2;
                     break;
                 case 3:
                     tempList[i] = 1;
                     break;
             }
         }
         return tempList;
```

```
}

//This bijection type transformes given list elements: G(I)=I; G(X)=X; G(Y)=Z; G(Z)=Y;
List<int> doBijection5(List<int> GivenList)
{
    for (int i = 0; i < GivenList.Count; i++)
    {
        switch (GivenList[i])
        {

            case 0:
                tempList[i] = 0;
                break;
            case 1:
                tempList[i] = 1;
                break;
            case 2:
                tempList[i] = 3;
                break;
            case 3:
                tempList[i] = 2;
                break;
        }
    }
    return tempList;
}



// This bijection reverse elements order in given list.
List<int> reverseList(List<int> GivenList)
{
    for (int i = 1; i <= GivenList.Count; i++)
    {
        switch (GivenList[i-1])
        {
            case 0:
                tempList2[tempList2.Count - i] = 0;
                break;
            case 1:
                tempList2[tempList2.Count - i] = 1;
                break;
            case 2:
                tempList2[tempList2.Count - i] = 2;
                break;
            case 3:
                tempList2[tempList2.Count - i] = 3;
                break;
        }
    }
    return tempList2;
}

// This function determines if "firstList" as a check could catch
// "secondList" as a mistake case.
// Olso this block determines if "firstList" and "secondList" as
//two checks are uncompatible.
bool firstCatchSecond(List<int> firstList, List<int> secondList)
{
    bool result = false;
    for (int i = 0; i < firstList.Count; i++)
    {
        if (firstList[i]*secondList[i] != 0)
        {
            if (firstList[i] != secondList[i])
            {
                if (result)
                    result = false;
                else
                    result = true;
            }
        }
    }
    return result;
}

// Smart function. It's used for to finaly use not obvious way for to
// find stabilizer code check groops:
```

```
        void doSmartFunction(List<List<int>> smartFunctionList, TreeNode mainNode, long
fullExpansionLength, bool[] resultlessChecks, int n, List<List<int>> allPossibleMistakes)
        {
            for (long i = 0; i < fullExpansionLength-1; i++)
            {
                smartFunctionTempList = nextListAfter(smartFunctionTempList);
                if (!resultlessChecks[i])
                {
                    bool candidatFailed = false;
                    for (int j = 0; j < smartFunctionList.Count; j++)
                    {
                        if (firstCatchSecond(smartFunctionList[j],smartFunctionTempList))
                        {
                            candidatFailed = true;
                        }
                        if (sameLists(smartFunctionList[j], smartFunctionTempList))
                        {
                            candidatFailed = true;
                        }
                    }
                    if (!candidatFailed)
                    {
                        smartFunctionList.Add(smartFunctionTempList);
                        alreadyIncludedIndexes.Add(indexOf(smartFunctionTempList));
                    }
                }
            }
            discovered++;
            mainNode.Nodes[3].Nodes.Add("Stablizer code groop:");
            for (int ij = 0; ij < smartFunctionList.Count; ij++)
            {
                mainNode.Nodes[3].Nodes[discovered -
1].Nodes.Add(convertAListToString(smartFunctionList[ij]));
            }
        }

        //This block do the main work after button was clicked.
        private void buttonStart_Click(object sender, EventArgs e)
        {
            // This line makes impossible to click the button as soon
            // as main work is in progress.
            buttonStart.Enabled = false;

            // This block receive the values of n and k from controls.
            int n = 0;
            int k = 0;
            n = int.Parse(numericUpDownM.Value.ToString());
            k = int.Parse(numericUpDownK.Value.ToString());

            //Reset discovered value;
            discovered = 0;

            // This block creates mane node for TreeView control.
            TreeNode mainNode = new TreeNode();
            mainNode.Name = "theMainNode";
            mainNode.Text = "Results:";
            mainNode.Nodes.Add("All possible mistakes (r<=k) found:");
            mainNode.Nodes.Add("Simetric groops was successfully combined into:");
            mainNode.Nodes.Add("First lists of searching stuff:");
            mainNode.Nodes.Add("Stabilizer code possible only inside theese groops:");


            //This block is used instead of (fourExpN := 4^n) and (twoExpN := 2^n).
            long fourExpN = 1;
            long twoExpN = 1;
            for (int i = 0; i < n; i++)
            {
                fourExpN *= 4;
                twoExpN *= 2;
            }

            //all possible I,X,Y,Z combinations in n-sised check.
            long fullExpansionLength = fourExpN;

            //Length from X to X, IX to XZ, IIX to XZZ, IIIX to XZZZ and so on.
             //Used for groop case status check.
            long firstExpansionLength = fourExpN / 2 - 1;
```

```
// All elements to start from
bool[] elementsToStart = new bool[fullExpansionLength-1];
// Checkes with theese orders will not be in a main search.
bool[] resultlessChecks = new bool[fullExpansionLength-1];
//Array to catch Bijections and Reversions in for future vector skip cases.
bool[] alternativeWasCathed = new bool[fullExpansionLength-1];
// All array elements values are false as a default.

//Refresh for the next search without exiting.
tempList.Clear();
tempList2.Clear();
tempList3.Clear();

// Min vector constant set as "III...II"
List<int> startingList1 = new List<int>();
for (int i = 1; i <= n; i++)
{
    startingList1.Add(0);
    tempList.Add(0);
    tempList2.Add(0);
    tempList3.Add(0);
}

// Current used list for calculations. Starting from Min.
List<int> currentList = new List<int>();

//All possible mistake list.
List<List<int>> allPossibleMistakes = new List<List<int>>();

//This block will fill the list with all possible k mistakes in n.
for (long i = 0; i < fullExpansionLength; i++)
{
    int kInThis = 0;
    for (int j = 0; j < startingList1.Count; j++)
    {
        if (startingList1[j] != 0)
        {
            kInThis += 1;
        }
    }

    if (kInThis <= k)
    {
        allPossibleMistakes.Add(startingList1);
        mainNode.Nodes[0].Nodes.Add(convertAListToString(startingList1));
    }

    startingList1 = nextListAfter(startingList1);
}

// MessageBox.Show("All possibles mistakes are calculated.
 //Huge part are done. Hold on!");


// Min vector set as "III...II"
currentList.Clear();
for (int i = 1; i <= n; i++)
{
    currentList.Add(0);
}


// Main setup calculations
for (long i = 0; i < firstExpansionLength; i++)
{
    currentList = nextListAfter(currentList);

    if (!alternativeWasCathed[i])
    {
        elementsToStart[i] = true;
        int catchDevisionGroopCount = 0;
        int missDevisionGroopCount = 0;
        int biggestDevisionGroopCountOfBoth = 0;

        for (int j = 0; j < allPossibleMistakes.Count; j++)
        {
            if (firstCatchSecond(currentList, allPossibleMistakes[j]))
            {
```

```
                            catchDevisionGroopCount++;
                            //MessageBox.Show(convertAListToString(allPossibleMistakes[j]) + "
catched " + convertAListToString(currentList));
                        }
                        else
                        {
                            missDevisionGroopCount++;
                            //MessageBox.Show(convertAListToString(allPossibleMistakes[j]) + "
missed " + convertAListToString(currentList));
                        }
                    }
                    //MessageBox.Show("Results are " + catchDevisionGroopCount + "/" +
missDevisionGroopCount);

                    if (catchDevisionGroopCount > missDevisionGroopCount)
                    {
                        biggestDevisionGroopCountOfBoth = catchDevisionGroopCount;
                    }
                    else
                    {
                        biggestDevisionGroopCountOfBoth = missDevisionGroopCount;
                    }

                    long transf0Index = indexOf(currentList);
                    long transf1Index = indexOf(doBijection1(currentList));
                    long transf2Index = indexOf(doBijection2(currentList));
                    long transf3Index = indexOf(doBijection3(currentList));
                    long transf4Index = indexOf(doBijection4(currentList));
                    long transf5Index = indexOf(doBijection5(currentList));
                    long transf6Index = indexOf(reverseList(currentList));
                    long transf7Index = indexOf(reverseList(doBijection1(currentList)));
                    long transf8Index = indexOf(reverseList(doBijection2(currentList)));
                    long transf9Index = indexOf(reverseList(doBijection3(currentList)));
                    long transf10Index = indexOf(reverseList(doBijection4(currentList)));
                    long transf11Index = indexOf(reverseList(doBijection5(currentList)));

                    alternativeWasCathed[transf0Index] = true;
                    alternativeWasCathed[transf1Index] = true;
                    alternativeWasCathed[transf1Index] = true;
                    alternativeWasCathed[transf2Index] = true;
                    alternativeWasCathed[transf3Index] = true;
                    alternativeWasCathed[transf4Index] = true;
                    alternativeWasCathed[transf5Index] = true;
                    alternativeWasCathed[transf6Index] = true;
                    alternativeWasCathed[transf7Index] = true;
                    alternativeWasCathed[transf8Index] = true;
                    alternativeWasCathed[transf9Index] = true;
                    alternativeWasCathed[transf10Index] = true;
                    alternativeWasCathed[transf11Index] = true;

                    bool checkIsSenseless = biggestDevisionGroopCountOfBoth > twoExpN/4;
                    if (checkIsSenseless)
                    {
                        resultlessChecks[transf0Index] = true;
                        resultlessChecks[transf1Index] = true;
                        resultlessChecks[transf2Index] = true;
                        resultlessChecks[transf3Index] = true;
                        resultlessChecks[transf4Index] = true;
                        resultlessChecks[transf5Index] = true;
                        resultlessChecks[transf6Index] = true;
                        resultlessChecks[transf7Index] = true;
                        resultlessChecks[transf8Index] = true;
                        resultlessChecks[transf9Index] = true;
                        resultlessChecks[transf10Index] = true;
                        resultlessChecks[transf11Index] = true;
                    }
                    long twoExpNdiv4 = twoExpN/4;
                    string stringOfCheckDesision = "";
                    if (checkIsSenseless) stringOfCheckDesision = "a bad one. (-)";
                    else stringOfCheckDesision = "a good one! (+)";
                    mainNode.Nodes[1].Nodes.Add(convertAListToString(currentList) +
                        " (" +
                        catchDevisionGroopCount.ToString() +
                        "/" +
                        missDevisionGroopCount.ToString() +
                        ")->" +
                        biggestDevisionGroopCountOfBoth.ToString() +
                        " stored in " +
                        twoExpNdiv4.ToString() +
```

```
                        ": is " +
                        stringOfCheckDesision +
                        ")" );
                }
            }

            // Min vector constant set as "III...II"
            tempList3.Clear();
            for (int j = 1; j <= n; j++)
            {
                tempList3.Add(0);
            }
            for (int j = 0; j < fullExpansionLength-1; j++)
            {
                tempList3 = nextListAfter(tempList3);
                if (elementsToStart[j] == true)
                {
                    mainNode.Nodes[2].Nodes.Add(convertAListToString(tempList3));
                    alreadyIncludedIndexes.Clear();
                    alreadyIncludedIndexes.Add(indexOf(currentList));
                    smartFunctionTempList.Clear();
                    for (int ij = 1; ij <= n; ij++)
                    {
                        smartFunctionTempList.Add(0);
                    }
                    List<List<int>> recurtionList = new List<List<int>>();
                    recurtionList.Add(tempList3);
                    doSmartFunction(recurtionList, mainNode, fullExpansionLength,
resultlessChecks, n, allPossibleMistakes);
                }
            }

            // This block is used to display results in TreeView control.
            treeViewMain.Nodes.Clear();
            treeViewMain.Nodes.Add(mainNode);

            // Next block, taken in comments, was used for to construct 2.2 table.
            /*
            double resultCollector = 0;
            double accumulation1 = 1;
            double accumulation2 = 1;
            double accumulation3 = 1;
            double twoExpN = 1;

            for (int i = 1; i < n; i++)
            {
                twoExpN *= 2;
            }

            resultCollector += twoExpN - 1;
            for (int i = 1; i <= k; i++)
            {
                accumulation1 *= 3;
            }

            for (int i = 1; i <= k; i++)
            {
                double currentSumResult = 0;
                accumulation2 *= (n-i+1);
                accumulation3 *= i;
                currentSumResult = accumulation1*accumulation2/accumulation3;
                resultCollector -= currentSumResult;
            }
            MessageBox.Show(resultCollector.ToString());
            */

            //This line finally release the button for to make possible another attempt.
            buttonStart.Enabled = true;
        }

        //This code processed when m value is chenged, to set new Maximum for K.
        private void numericUpDownM_ValueChanged(object sender, EventArgs e)
        {
            numericUpDownK.Maximum = numericUpDownM.Value;
        }
        //This block provide exit option.
        private void buttonExit_Click(object sender, EventArgs e)
        {
            this.Close();
```

```
            }
        }
    }
```

5_1_case.cs

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

namespace __1_case
{
    public partial class Form1 : Form
    {

        //Some global lists is neaded for list collection preset in some functions.
        List<int> tempList = new List<int>();
        List<int> tempList2 = new List<int>();

        //This block returns the index of correction candidat. For example:
        //XIXZI->[(1*4^4)+(1*4^2)+(3*4^1)+1=285]
        int indexOf(List<int> currentList)
        {
            int result = 0;
            int FourFlour = 1;
            for (int i = 4; i >= 0; i--)
            {
                result += currentList[i] * FourFlour;
                FourFlour *= 4;
            }
            return result - 1;
       }

       // This bijection type transformes given list elements: G(I)=I; G(X)=Y; G(Y)=Z; G(Z)=X;
       List<int> doBijection1(List<int> GivenList)
        {
            for (int i = 0; i < GivenList.Count; i++)
            {
                switch (GivenList[i])
                {
                    case 0:
                        tempList[i] = 0;
                        break;
                    case 1:
                        tempList[i] = 2;
                        break;
                    case 2:
                        tempList[i] = 3;
                        break;
                    case 3:
                        tempList[i] = 1;
                        break;
                }
            }
            return tempList;
       }

       // This bijection type transformes given list elements: G(I)=I; G(X)=Z; G(Y)=X; G(Z)=Y;
       List<int> doBijection2(List<int> GivenList)
        {
            for (int i = 0; i < GivenList.Count; i++)
            {
                switch (GivenList[i])
                {
                    case 0:
                        tempList[i] = 0;
                        break;
                    case 1:
                        tempList[i] = 3;
                        break;
                    case 2:
                        tempList[i] = 1;
                        break;
                    case 3:
                        tempList[i] = 2;
                        break;
                }
            }
```

```
     return tempList;
}

// This bijection type transformes given list elements: G(I)=I; G(X)=Y; G(Y)=X; G(Z)=Z;
List<int> doBijection3(List<int> GivenList)
{
     for (int i = 0; i < GivenList.Count; i++)
     {
         switch (GivenList[i])
         {

             case 0:
                 tempList[i] = 0;
                 break;
             case 1:
                 tempList[i] = 2;
                 break;
             case 2:
                 tempList[i] = 1;
                 break;
             case 3:
                 tempList[i] = 3;
                 break;
         }
     }
     return tempList;
}

// This bijection type transformes given list elements: G(I)=I; G(X)=Z; G(Y)=Y; G(Z)=X;
List<int> doBijection4(List<int> GivenList)
{
     for (int i = 0; i < GivenList.Count; i++)
     {
         switch (GivenList[i])
         {

             case 0:
                 tempList[i] = 0;
                 break;
             case 1:
                 tempList[i] = 3;
                 break;
             case 2:
                 tempList[i] = 2;
                 break;
             case 3:
                 tempList[i] = 1;
                 break;
         }
     }
     return tempList;
}

// This bijection type transformes given list elements: G(I)=I; G(X)=X; G(Y)=Z; G(Z)=Y;
List<int> doBijection5(List<int> GivenList)
{
     for (int i = 0; i < GivenList.Count; i++)
     {
         switch (GivenList[i])
         {

             case 0:
                 tempList[i] = 0;
                 break;
             case 1:
                 tempList[i] = 1;
                 break;
             case 2:
                 tempList[i] = 3;
                 break;
             case 3:
                 tempList[i] = 2;
                 break;
         }
     }
     return tempList;
 }
```

```
// This bijection reverse elements order in given list.
List<int> reverseList(List<int> GivenList)
{
    for (int i = 1; i <= GivenList.Count; i++)
    {
        switch (GivenList[i - 1])
        {
            case 0:
                tempList2[tempList2.Count - i] = 0;
                break;
            case 1:
                tempList2[tempList2.Count - i] = 1;
                break;
            case 2:
                tempList2[tempList2.Count - i] = 2;
                break;
            case 3:
                tempList2[tempList2.Count - i] = 3;
                break;
        }
    }
    return tempList2;
}


// This block is used to set next vector value. For example:
// IIX-> IIY, XYZ->XZI, IZZ->XII
List<int> nextListAfter(List<int> givenList)
{
    List<int> anotherList = new List<int>();
    if (givenList.Count != 0)
    {
        if (givenList[givenList.Count - 1] != 3)
        {
            for (int i = 0; i < givenList.Count - 1; i++)
            {
                anotherList.Add(givenList[i]);
            }

            anotherList.Add(givenList[givenList.Count - 1] + 1);
            return anotherList;
        }
        else
        {

            for (int i = 0; i < givenList.Count - 1; i++)
            {
                anotherList.Add(givenList[i]);
            }
            anotherList = nextListAfter(anotherList);
            anotherList.Add(0);
            return anotherList;
        }
    }
    return anotherList;
}

// This block is used to convert Generic List<int> data elemets to string.
string convertAListToString(List<int> givenList)
{
    string result = "(";
    for (int i = 0; i < givenList.Count; i++)
    {
        if (givenList[i] == 0) result += "I";
        if (givenList[i] == 1) result += "X";
        if (givenList[i] == 2) result += "Y";
        if (givenList[i] == 3) result += "Z";
        if (i < givenList.Count - 1) result += ",";
    }
    result += ")";
    return result;
}

// This block determines if "firstList" and "secondList" as two checks are compatible.
bool isCompatible(List<int> firstList, List<int> secondList)
{
    bool result = true;
    for (int i = 0; i < firstList.Count; i++)
```

```
            {
                if ((firstList[i] != 0) && (secondList[i] != 0))
                {
                    if (firstList[i] != secondList[i])
                    {
                        if (result)
                            result = false;
                        else
                            result = true;
                    }
                }
            }
            return result;
        }
        public Form1()
        {
            InitializeComponent();
        }


        //this block returns list instead of it index value
        List<int> listByIndex (int anIndex)
        {
            List<int> result  = new List<int>();
            result.Add( anIndex       / 256);
            result.Add((anIndex % 256) / 64);
            result.Add((anIndex % 64) / 16);
            result.Add((anIndex % 16) / 4);
            result.Add( anIndex % 4     );
            return result;
        }

        private void buttonStart_Click(object sender, EventArgs e)
        {
            List<int> currentCases = new List<int>();
            List<List<int>> goodCaseList = new List<List<int>>();

            currentCases.Add(0);
            currentCases.Add(0);
            currentCases.Add(0);
            currentCases.Add(0);
            currentCases.Add(0);

            bool[] goodCaseIndexes = new bool[1023];

            for (int i = 0; i < 1024; i++)
            {
                if ((currentCases[0] == 0) && (currentCases[1] * currentCases[2] *
currentCases[3] * currentCases[4] != 0) ||
                    (currentCases[1] == 0) && (currentCases[0] * currentCases[2] *
currentCases[3] * currentCases[4] != 0) ||
                    (currentCases[2] == 0) && (currentCases[0] * currentCases[1] *
currentCases[3] * currentCases[4] != 0) ||
                    (currentCases[3] == 0) && (currentCases[0] * currentCases[1] *
currentCases[2] * currentCases[4] != 0) ||
                    (currentCases[4] == 0) && (currentCases[0] * currentCases[1] *
currentCases[2] * currentCases[3] != 0))
                {
                    goodCaseIndexes[i] = true;
                }
                currentCases = nextListAfter(currentCases);
            }

            List<int> currentList = new List<int>();
            currentList.Add(0);
            currentList.Add(0);
            currentList.Add(0);
            currentList.Add(0);
            currentList.Add(0);

            tempList.Clear();
            tempList.Add(0);
            tempList.Add(0);
            tempList.Add(0);
            tempList.Add(0);
            tempList.Add(0);

            tempList2.Clear();
            tempList2.Add(0);
```

```
            tempList2.Add(0);
            tempList2.Add(0);
            tempList2.Add(0);
            tempList2.Add(0);


            bool[] alternativeWasCathed = new bool[1023];
            bool[] shouldBeStartedWith = new bool[1023];

            for (int i = 0; i < 1023; i++)
            {
                currentList = nextListAfter(currentList);

                if (!alternativeWasCathed[i])
                {
                    shouldBeStartedWith[i] = true;
                    int transf0Index = indexOf(currentList);
                    int transf1Index = indexOf(doBijection1(currentList));
                    int transf2Index = indexOf(doBijection2(currentList));
                    int transf3Index = indexOf(doBijection3(currentList));
                    int transf4Index = indexOf(doBijection4(currentList));
                    int transf5Index = indexOf(doBijection5(currentList));
                    int transf6Index = indexOf(reverseList(currentList));
                    int transf7Index = indexOf(reverseList(doBijection1(currentList)));
                    int transf8Index = indexOf(reverseList(doBijection2(currentList)));
                    int transf9Index = indexOf(reverseList(doBijection3(currentList)));
                    int transf10Index = indexOf(reverseList(doBijection4(currentList)));
                    int transf11Index = indexOf(reverseList(doBijection5(currentList)));

                    alternativeWasCathed[transf0Index] = true;
                    alternativeWasCathed[transf1Index] = true;
                    alternativeWasCathed[transf1Index] = true;
                    alternativeWasCathed[transf2Index] = true;
                    alternativeWasCathed[transf3Index] = true;
                    alternativeWasCathed[transf4Index] = true;
                    alternativeWasCathed[transf5Index] = true;
                    alternativeWasCathed[transf6Index] = true;
                    alternativeWasCathed[transf7Index] = true;
                    alternativeWasCathed[transf8Index] = true;
                    alternativeWasCathed[transf9Index] = true;
                    alternativeWasCathed[transf10Index] = true;
                    alternativeWasCathed[transf11Index] = true;
                }
            }


            TreeNode mainNode = new TreeNode();
            mainNode.Name = "theMainNode";
            mainNode.Text = "Results:";
            mainNode.Nodes.Add("Stabilizing codes:");
            int codeCase = 1;
            for (int i = 0; i < 1023; i++)
            {
                if ((shouldBeStartedWith[i] == true) && (goodCaseIndexes[i] == true))
                {
                    for (int j = 0; j < 1023; j++)
                    {
                        if (goodCaseIndexes[i] == true)
                        {
                            if (isCompatible(listByIndex(i), listByIndex(j)) && (i != j))
                            {
                                for (int k = 0; k < 1023; k++)
                                {
                                    if (goodCaseIndexes[k] == true)
                                    {
                                        if ((isCompatible(listByIndex(k), listByIndex(j)) &&
(k != j)) && (isCompatible(listByIndex(k), listByIndex(i)) && (k != i)))
                                        {
                                            mainNode.Nodes.Add(codeCase.ToString());
                                            mainNode.Nodes[codeCase -
1].Nodes.Add(convertAListToString(listByIndex(i)));
                                            mainNode.Nodes[codeCase -
1].Nodes.Add(convertAListToString(listByIndex(j)));
                                            codeCase++;
                                        }
                                    }
                                }
                            }
                        }
```

```
                    }
                }
                if (i%30 == 0) MessageBox.Show("Is done with " + i.ToString() + "of 1023");
            }


            // This block is used to display results in TreeView control.
            treeViewMain.Nodes.Clear();
            treeViewMain.Nodes.Add(mainNode);
        }
    }
}
```

2. pielikums. Programmatūras CD disks ar pēdējo programmatūras versiju instalācijas failiem, koda resursu failiem un programmatūras dokumentāciju.

Šajā lapā ir jābūt pieliktam programmatūras diskam.

Piezīmēm.

Piezīmēm.

DOKUMENTĀRĀ LAPA

Bakalaura darbs „Stabilizatoru kodi kvantu kļūdu korekcijai” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, ka izmantoti tikai tajā norādītie informācijas avoti, un ka iesniegtā darba elektroniskā kopija atbilst izdrukātam darbam.

Darba autors:

Boriss Benzerruki _____ datums _____.

Rekomendēju darbu aizstāvēšanai.

Vadītājs:

profesors Dr. dat.

Andris Ambainis _____ datums _____

Recenzents:

(vārds, uzvārds) _____

(amats, zinātniskais grāds) _____

Paraksts: _____

Darbs iesniegts Datorikas fakultātē.

Metodiķe:

Ārija Sproģe _____ datums _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē.

prot. Nr. _____, vērtējums ____ (_____)

datums _____

Komisijas sekretāre:

_____

Paraksts: _____