LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

KVANTU VAICĀJOŠIE ALGORITMI

MAĢISTRA DARBS

Autore: **Jeļena Šišova**
Stud. apl. js05090
Darba vadītājs: Dr.dat. Alina Vasiļjeva

RĪGA 2013

ANOTĀCIJA

Darba mērķis ir uzkonstruēt pēc iespējas efektīvākus kvantu vaicājošos algoritmus multifunkcijām. Pašlaik visefektīvākā zināmā kvantu precīza vaicājošā algoritma sarežģītība ir $O(N^{0.8675\ldots})$; klasiskajam algoritmam, kurš risina tādu pašu problēmu, ir nepieciešami vismaz N vaicājumi. Atrast kvantu algoritmu, kurš būtu labāks par klasisko algoritmu, ir diezgan grūts uzdevums. Tiek pieņemts, ka lielu starpību starp kvantu un klasisko vaicājošo algoritmu sarežģītību var sasniegt, ja konstruēt algoritmus multifunkcijām, t.i. kad rezultāts var būt viena vai vairākas vērtības no noteiktas kopas.

Darbā tiek aprakstīta programma kvantu algoritmu ģenerēšanai, kura rēķina arī klasiskas sarežģītības apakšējo robežu. Ir aprakstītas arī vairākas multifunkcijas, kurām kvantu vaicājošie algoritmi ir efektīvāki par klasiskajiem algoritmiem.

Atslēgvārdi: kvantu vaicājošie algoritmi, algoritmu sarežģītība, algoritmu konstruēšana, multifunkcija

ABSTRACT

The aim of the thesis „Quantum query algorithms" is to design effective quantum query algorithms for computing multifunctions. Currently the complexity of the best known exact quantum query algorithm is $O(N^{0.8675...})$ while the classical analogue needs at least N queries. It is a difficult task to design quantum algorithms that are more effective than the classical ones; we assume that a large gap between classical and quantum query complexity can be achieved if algorithms are designed for multifunctions instead of usual functions, i.e. the result may contain one or more values from the result set.

We present a program that generates quantum query algorithms and calculates lower bounds of the classical complexity and provide examples of multifunctions that can be calculated by quantum query algorithms faster than by classical algorithms.

Keywords: quantum query algorithms, algorithm complexity, algorithm design, multifunctions

AUTOREFERĀTS

Nozīmīgākais autores paveiktais šajā darbā ir sekojošs:

- Ir uzrakstīta programma, kura, saņemot transformāciju secības aprakstu, automātiski taisa aprēķinus. Tā ģenerē bināras virknes tādā garumā, kāds ir mainīgo skaits, sareizina transformācijām atbilstošās unitārās matricas un izvada multifunkcijas vērtības.

  Darba gaitā no šīs programmas izveidotas trīs atsevišķas programmas. Katra no tām ir lietota savas konkrētas multifunkcijas rēķināšanai un izmanto vairākas atšķirīgas metodes šīs multifunkcijas definīcijas pārbaudei.

  Līdzīgā veidā var izveidot programmas arī citu multifunkciju ģenerēšanai.
- Ir izstrādāta programma, kura, saņemot vārdnīcu ar multifunkcijas ievaddatu virknēm un tām asociētām vērtībām, aprēķina sarežģītības apakšējo robežu klasiskajam vaicājošam algoritmam, kas rēķina šādu multifunkciju. Tas palīdz ātri noteikt, vai kvantu algoritms ir labāks par klasisko algoritmu.
- Programmu rakstīšanai autore apguva programmēšanas valodu Python, uz kuras ir balstīta matemātiskā programmatūra Sage.
- Ar augstāk minēto programmu palīdzību tika nodefinētas trīs multifunkcijas, kuras kvantu vaicājošie algoritmi rēķina daudz ātrāk, nekā klasiskie vaicājošie algoritmi.

  Darbā no sākuma tiek aprakstīti šo multifunkciju speciālgadījumi ar mazu mainīgo skaitu (līdz astoņiem mainīgajiem), un pēc tam seko to vispārinājumu apraksti.

SATURA RĀDĪTĀJS

APZĪMĒJUMI

Apzīmējums	Skaidrojums
H_i	Adamāra matrica ar kārtu i
AND(x_1, x_2, ..., x_N)	Loģiska operācija AND (UN) N mainīgajiem
OR(x_1, x_2, ..., x_N)	Loģiska operācija OR (VAI) N mainīgajiem
XOR(x_1, x_2, ..., x_N)	Loģiska operācija XOR (izslēdzošais VAI) N mainīgajiem
sqrt(x)	Skaitļa x kvadrātsakne

IEVADS

Ir ļoti svarīgi samazināt laiku, kuru dators izmanto dažādiem aprēķiniem. Cilvēki izmanto datorus aizvien lielāku un grūtāku uzdevumu risināšanai, un pēc kāda laika klasiskie datori var nebūt pietiekami jaudīgi daudzu uzdevumu risināšanai pieņemamā laikā. Šajā ziņā varētu palīdzēt kvantu datori. Pašlaik tie vēl nav pieejami industriālā līmenī, tomēr eksistē šādu datoru primitīvi prototipi (piemēram, [1,2]). Jau šobrīd ir zināmi vairāki uzdevumi, kuru risināšanā kvantu datori ir pārāki par klasiskajiem datoriem. Piemēram, meklēšanas problēmas (Grovera meklēšanas algoritms [3]) un skaitļu sadalīšana reizinātājos (Šora algoritms [4]).

Šajā darbā tiek pētīta kvantu vaicājošo algoritmu sarežģītība. Ir dota funkcija, kurai nav zināmas ievaddatu mainīgo vērtības. Lai aprēķinātu funkcijas rezultātu, algoritms var vienu pēc otras prasīt atklāt kāda mainīgā vērtību. Ir nepieciešams uzkonstruēt šāda veida algoritmu, kuram būtu nepieciešams pēc iespējas mazāks jautājumu skaits. Kvantu algoritmi šim nolūkam parasti ir efektīvāki, nekā to klasiskie analogi. Līdz nesenam laikam vislielākais zināmais attālums starp precīza klasiskā determinētā vaicājošā algoritma sarežģītību un precīza kvantu vaicājošā algoritma sarežģītību bija divkāršs. To sasniedz XOR (binārās saskaitīšanas) funkcija, kuras aprēķināšanai klasiskajam algoritmam ir nepieciešams zināt visu N mainīgo vērtības, bet kvantu algoritmam pietiek ar (1/2)N vērtībām [5]. Nesen A.Ambainis uzkonstruēja funkciju, kurai attālums starp determinēto un kvantu vaicājumu sarežģītību ir N pret $O(N^{0.8675\ldots})$ [6]. Tomēr joprojām nav zināms, kāds ir vislielākais iespējamais attālums starp determinēto un kvantu vaicājumu sarežģītību.

Šī darba galvenais mērķis ir atrast funkcijas ar lielu starpību starp klasiskajiem un kvantu vaicājošiem algoritmiem. Bet, atšķirībā no daudziem citiem pētījumiem, šī darba autore ir ieinteresēta multifunkcijās parasto funkciju vietā. Multifunkcijas rezultāts var būt viena vai vairākas vērtības no vērtību apgabala.

Var izteikt minējumu, ka kvantu sistēmas varētu būt īpaši piemērotas multifunkciju rēķināšanai, jo skaitļošanas beigās kvantu sistēma var atrasties bāzes stāvokļu superpozīcijā, un pēc mērīšanas tiek iegūta viena vērtība no kopas, nevis kāda noteikta vērtība. Darbinot kvantu algoritmu vairākas reizes ar vienu un to pašu ievadi, var iegūt dažādas vērtības.

Tādu kvantu algoritmu konstruēšana, kas ir efektīvāki par klasiskajiem, ir ļoti netriviāls uzdevums. Tāpēc darbā tiek meklēti nevis kvantu algoritmi kādas funkcijas rēķināšanai, bet gan

multifunkcijas noteiktai kvantu algoritma konstrukcijai. Lai meklētu multifunkcijas, kas atbilst vēlamām īpašībām, var tikt uzrakstītas un izmantotas datora programmas. Autore pieļauj, ka labākos rezultātus ir iespējams sasniegt, uzģenerējot daudz dažādu algoritmu, pēc tam nosakot, kādas multifunkcijas tie rēķina, un tad salīdzināt šos algoritmus ar to klasiskajiem analogiem.

Kas attiecas uz klasiskajiem analogiem, tika meklēts kāds paņēmiens, lai uzreiz saprastu, vai kvantu algoritms ir labāks par attiecīgo klasisko algoritmu. Šim nolūkam ir uzrakstīta programma klasisko algoritmu sarežģītības apakšējās robežas aprēķinam.

Zemāk ir aprakstīta maģistra darba struktūra.

Pirmajā nodaļā lasītājs tiek iepazīstināts ar kvantu skaitļošanas pamatiem, kā arī ar šī darba kontekstā interesantām matricām.

Otrajā nodaļā ir dota multifunkcijas jēdziena definīcija un vairāki citi jēdzieni, kas ir izmantoti šajā darbā. Papildus, ir aprakstīta notācija, kas ir izmantota rezultātu aprakstā.

Trešajā nodaļā tiek apspriesti vaicājošie algoritmi. 3.1. apakšnodaļā ir aprakstīti klasisko algoritmu veidi, bet 3.2. apakšnodaļā - kvantu algoritmi un to izmantošana multifunkciju aprēķināšanai.

Ceturtajā nodaļā ir izskaidrota izmantotā metodoloģija, kam seko izveidoto programmu apraksts.

Piektajā nodaļā autore iepazīstina lasītājus ar darba gaitā iegūtiem algoritmiem.

1. KVANTU SKAITĻOŠANA

Pirms tiks sniegts priekšstats par kvantu vaicājošiem algoritmiem, no sākuma īsi tiek izklāsti kvantu skaitļošanas pamatjēdzieni. Sīkākajai informācijai var apskatīties [5,7,8,9].

Kvantu analogu klasiskajam bitam sauc par kubitu. Tapāt, ka klasiskais bits, kubits var pieņemt vērtības 0 un 1. Bet, atšķirībā no parasta bita, kubits var būt 0 un 1 superpozīcijā – stāvoklī $\alpha|0\rangle + \beta|1\rangle$, kur $\alpha \in \mathbb{C}, \beta \in \mathbb{C}$, un kuriem izpildās (1.1). Skaitļus α un β sauc par amplitūdām.

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.1}$$

Stāvokli $\alpha|0\rangle + \beta|1\rangle$ var izteikt arī vektora veidā (1.2):

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{1.2}$$

N-kubitu sistēmas jebkuru stāvokli ψ var izteikt kā (1.3), kā arī aprakstīt kā n-dimensiju vektoru ar amplitūdu sadalījumu (1.4).

$$|\psi\rangle = \sum_{i=0}^{n-1} a_i|i\rangle \tag{1.3}$$

$$\alpha_i \in \mathbb{C}$$

$$|\psi\rangle = \begin{pmatrix}\alpha_1\\ \dots\\ \alpha_n\end{pmatrix} \tag{1.4}$$

Joprojām saglabājas īpašība (1.5):

$$\sum_{i=0}^{n-1} |a_i|^2 = 1 \tag{1.5}$$

Stāvokļi $|0\rangle$, $|1\rangle$, ..., $|n\text{-}1\rangle$ tiek saukti par bāzes stāvokļiem. Piemēram, kvantu sistēmai ar trīs kubitiem ir astoņi bāzes stāvokļi - $|000\rangle$,$|001\rangle$,$|010\rangle$,$|011\rangle$,$|100\rangle$,$|101\rangle$,$|110\rangle$,$|111\rangle$.

Kvantu sistēmas stāvokli var nomainīt, izmantojot unitārās transformācijas. Unitāra transformācija U darbā tiek aprakstīta unitāras matricas veidā.

Definīcija 1. *Kvadrātisku matricu U sauc par **unitāru**, ja $U^{\dagger} = U^{-1}$, kur $U^{\dagger}$ ir kompleksi saistīta transponēta matrica un U^{-1} ir apgrieztā matrica* [10].

Lai dabūtu jaunu stāvokli $|\varphi\rangle = \sum_{i=0}^{n-1} \beta_i|i\rangle$, vajag sākotnējo stāvokļa $|\psi\rangle = \sum_{i=0}^{n-1} a_i|i\rangle$ vektoru sareizināt ar unitāro matricu U (1.6).

$$|\varphi\rangle = U|\psi\rangle \tag{1.6}$$

Unitārām matricām piemīt tāda īpašība: tās nemaina kompleksa vektora garumu, tātad, saglabājas kvantu sistēmas stāvokļa īpašība (1.7).

$$\sum_{i=0}^{n-1} |\beta_i|^2 = 1 \tag{1.7}$$

Viena no interesantākām matricām ir Adamāra matrica, kura tiek bieži izmantota šajā darbā.

Definīcija 2. ***Adamāra matrica*** *ir kvadrātiska matrica, kuras elementi ir „-1" vai „1" un kurai izpildās tāda īpašība: blakusesošajās rindās vai kolonnās pusei no blakusesošajiem elementiem ir vienāda zīme, un otrajai pusei – pretēja* [11].

Definīcija 3. ***Vienības matrica*** *ir kvadrātiska matrica, kurai uz diagonāles ir vienīnieki, bet visās citās pozīcijās – nulles.*

Darbā n-kārtas Adamāra matrica, kura ir reizināta ar $\frac{1}{\sqrt{n}}$ (lai tā būtu unitāra), tiek apzīmēta ar H_n, piemēram (1.8), (1.9).

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.8}$$

$$H_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{1.9}$$

Otrā operācija, kuru var izdarīt ar kvantu stāvokli, ir mērījums. Darbā tiek izmantots t.s. pilns mērījums skaitļošanas bāzē. Taisot mērījumu uz stāvokļa $|\psi\rangle=a_0|0\rangle+\ldots+a_{n-1}|n\text{-}1\rangle$, var iegūt i ar varbūtību $|a_i|^2$. Pēc mērījuma oriģinālais kvantu sistēmas stāvoklis mainās uz $|i\rangle$, un diemžēl nav iespējams noteikt, kāds bija amplitūdu sadalījums pirms mērījuma.

2. MULTIFUNKCIJAS

Šajā nodaļā tiek aprakstīts galvenais pētījuma objekts – multifunkcijas.

Definīcija 4. *Par **multifunkciju** $M(X): M_D \rightarrow M_R$ sauc tādu funkciju, kur vismaz vienai vērtībai no definīcijas apgabala M_D atbilst divas vai vairākas vērtības no vērtību apgabala M_R* [12]*, t.i. katrai vērtībai no definīcijas apgabala var atbilst viena vai vairākas vērtības no vērtību apgabala.*

Kā multifunkcijas piemērus var minēt inverso moduļa funkciju f'(X)(2.2), kad, piemēram, vērtībai „2" no definīcijas apgabala atbilst vērtības „-2" un „2" no vērtību apgabala, vai inverso funkciju f'' no veselo skaitļu kvadrāta (2.3), kad, piemēram, vērtībai „1" no definīcijas apgabala atbilst vērtības „-1" un „1" no vērtību apgabala (2.4).

$$|2| = |-2| = 2 \tag{2.1}$$

$$f'(2) = \{-2; 2\} \tag{2.2}$$

$$f''(x) = \sqrt{x} \tag{2.3}$$

$$f''(1) = \mp 1 \tag{2.4}$$

Darbā tiek apskatīti multifunkcijas veidā (2.5).

$$M(X): \{0,1\}^N \rightarrow \mathbb{N} \tag{2.5}$$

$$X = (x_1, x_2, ..., x_N),$$

$$x_i \in \{0,1\}.$$

X tiek saukts par ievades bitu virkni vai vienkārši par ievades virkni vai ievades vektoru, kā arī par ievadi, x_i par mainīgajiem, M(X) par multifunkcijas vērtību kopu vai vienkārši par vērtību kopu vai vērtību apgabalu. Autore ievieš arī tādu terminu, kā izvada bitu virkne (vai izvada virkne), kas ir nuļļu un/vai vieninieku virkne garumā $|M_R|$, kur $x_i = 1$, ja $i \in M(X)$, un $x_i = 0$, ja $i \notin M(X)$.

Vēl darbā tiek izmantoti šādi apzīmējumi:

- Ar $|M_R|$ tiek apzīmēta vērtību kopas kardinalitāte, piemēram, ja ir apskatīta multifunkcija $M'^{(X)}: \{0,1\}^N \rightarrow \{1 \dots 8\}$, tad $|M_R| = 8$;
- Ar X^I tiek apzīmēta pirmā puse no ievades virknes; piemēram, ja $X = 10001010$, tad $X^I = 1000$;
- Ar X^{II} tiek apzīmēta otrā puse no ievades virknes; piemēram, ja $X = 10001010$, tad $X^I = 1010$;

- X_{odd} apzīmē mainīgos no nepāra ievades virknes X pozīcijām; piemēram, ja X = 01001000, tad X_{odd} = 0010
- X_{even} apzīmē mainīgos no pāra ievades virknes X pozīcijām; piemēram, ja X = 01001000, tad X_{even} = 1000
- *Operācija ar kopu* ir definēta sekojoši (2.6):

$$<\text{Set}><\text{Operation}><\text{Number}> = \{s_i<\text{Operation}><\text{Number}> \in \mathbb{N} | s_i \in <\text{Set}>\} \quad (2.6)$$

<Set> - patvaļīga kopa: $<\text{Set}> = \{i | i \in \mathbb{N}\}$,

<Operation> - aritmētiska operācija: $<\text{Operation}> = \{+, -, \times, /\}$,

<Number> - naturālais skaitlis: $<\text{Number}> \in \mathbb{N}$.

Piemēram, $\{0, 3, 5, 8\} \times 2 = \{0, 6, 10, 16\}$.

- *Set shift operation* ir īpaša operācija ar kopu, kura ir definēta tā (2.7):

$$shift(<Set>, <Number>) = (<Set> + <Number>) \bmod^{\Psi} 4 \quad (2.7)$$

$\bmod^{\Psi}4$ - modificēta moduļa operācija, kur "0" rezultātā ir aizvietota ar 4.

Piemēram, (2.8), (2.9), (2.10).

$$shift(\{1, 3\},2) = \{3, 5\} \bmod^{\Psi} 4 = \{1, 3\} \quad (2.8)$$

$$shift(\{1, 2\},2) = \{3, 4\} \bmod^{\Psi} 4 = \{3, 4\} \quad (2.9)$$

$$shift(\{4, 6\},4) = \{8, 10\} \bmod^{\Psi} 4 = \{2, 4\} \quad (2.10)$$

Ir nepieciešams ieviest arī tādus jēdzienus:

Definīcija 5. *Bināru virkni* $X = \{x_1, x_2, x_3, ..., x_N\}$ *sauc par binārās virknes* $Y = \{y_1, y_2, y_3, ..., y_N\}$ ***pretējo virkni***, *ja* $\forall i \in \{1...n\} : x_i \neq y_i$, *piemēram, 0100 ir 1011 pretējā virkne*.

Definīcija 6. *Bināru virkni* $X = \{x_1, x_2, x_3, ..., x_N\}$ *sauc par* ***simetrisku***, *ja pirmā X puse ir vienāda ar apgriezto pretējā secībā otro X pusi, piemēram, 0110, 00100100.*

Definīcija 7. *Bināru virkni* $X = \{x_1, x_2, x_3, ..., x_N\}$ *sauc par* ***pusvienādu***, *ja pirmā X puse ir vienāda ar otro X pusi, piemēram, 0101, 00100010.*

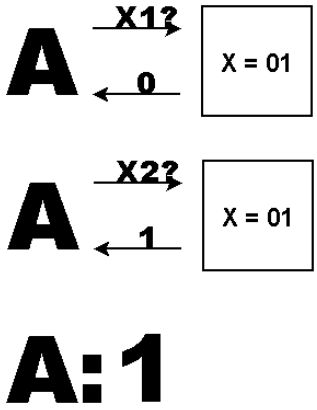
Definīcija 8. *Bināru virkni* $X = \{x_1, x_2, x_3, ..., x_N\}$ *sauc par* ***puspretēju***, *ja pirmā X puse ir otrās X puses pretējā virkne, piemēram, 01001011, 11100001*.

3. VAICĀJOŠIE ALGORITMI

Šajā nodaļā tiek stāstīts par skaitļošanas modeli, kura tiek izmantota funkciju rēķināšanai – vaicājošiem algoritmiem. Funkcijas $f(x_1,x_2,...,x_n)$ definīcija ir zināma, bet visi mainīgie ir paslēpti „melnajā kastē". Lai aprēķinātu funkciju, algoritms var pa vienam jautāt „melnajai kastei" mainīgo vērtības. Katrs jautājums tiek saukts par vaicājumu.

Definīcija 9. *Par vaicājoša algoritma* ***sarežģītību*** *tiek saukts jautājumu skaits, kas ir nepieciešams, lai izrēķināt funkcijas vērtību vissliktākajā gadījumā.*

Piemēram, jāaprēķina funkciju $OR(x_1,x_2)$. Algoritms A jautā x_1 vērtību un saņem atbildi „$x_1 = 1$". Tad algoritms A uzreiz var pateikt, ka $OR(x_1,x_2) = 1$. Bet ja algoritms A saņem atbildi „$x_1 = 0$", tad obligāti vēl jāpajautā x_2 vērtību (sk. 3.1.att). Tātad, algoritma A sarežģītība ir vienāda ar „2".



3.1.att. **Vaicājoša algoritma piemērs**

3.1.Klasiskie vaicājošie algoritmi

Vaicājošos algoritmus var viegli aprakstīt lēmumu koku veidā. Eksistē dažādi koku veidi; šajā pētījumā autore ir interesēta determinētos un varbūtiskos vaicājošos algoritmos.

3.1.1. Determinēti lēmumu koki

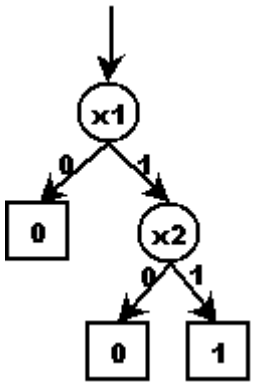
Definīcija 10. ***Determinēts lēmumu koks*** *ir sakārtots binārais koks, kur katra iekšējā virsotne ir marķēta ar mainīgo x_i, un katra lapa – ar vērtību no funkcijas vērtību apgabala* [5].

Šajā darbā pētītā gadījumā (t.i. $M(X):\{0,1\}^N \rightarrow \mathbb{N}$), lēmumu koks ir binārs, un katra bultiņa, kas iziet no iekšējās virsotnes, ir marķēta ar 0 vai 1.

Ja determinēts algoritms saņem vienu un to pašu ieejas virkni, tas vienmēr iziet vienu un to pašu koka ceļu un izvada vienu un to pašu rezultātu ar varbūtību p=1. Algoritms sāk ar koka sakni, jautā to vērtību, kas ir piesaistīta saknes virsotnei; binārā koka gadījumā, ja uzzināta vērtība ir vienāda ar „0", tad algoritms iet pa kreisi, ja tā ir „1" – tad pa labi; pēc tam algoritms jautā x_i, kas atrodas tikko sasniegtajā virsotnē, utt., kamēr algoritms neatnāk līdz lapai ar funkcijas vērtību. Saka, ka determinēts vaicājošais algoritms aprēķina funkciju *f* tad un tikai tad, ja šis algoritms dod pareizo rezultātu visiem ievades virknēm $x \in \{0,1\}^n$.

Determinēta algoritma sarežģītību apzīmē ar D(f), un tā atbilst attiecīgā optimālā (t.i. ar minimālo dziļumu) lēmumu koka dziļumam.

Piemēram, lēmumu koka, kas ir attēlots 3.2.att., dziļums ir vienāds ar „2". Šis lēmumu koks rēķina funkciju $AND(x_1,x_2)$. Tātad, šai funkcijai D(f) = 2.



3.2.att. **Determinēta lēmumu koka piemērs**

3.1.2. Varbūtiski lēmumu koki

Definīcija 11. ***Varbūtisks lēmumu koks*** *ir koks, kurš var saturēt iekšējās virsotnes ar sazarojumu p ∈[0; 1] [5], t.i. no vienas virsotnes iziet vairākas bultiņas, un katrai no šīm bultiņām ir piesaistīta varbūtība p ∈[0; 1], ar kuru algoritms ies pa ,šo bultiņu. Protams, ka varbūtību, kuras iziet no vienas virsotnes, summa nav lielāka par 1.*

Tātad, ja varbūtisks algoritms saņem vienu un to pašu ieejas virkni, tas var iziet dažādus koka ceļus un sasniegt dažādas lapas. Varbūtība, ar kuru algoritms dod rezultātu r kādai ieejas virknei X, ir vienāda ar varbūtību summu uz katras lapas, kura ir marķēta ar r, un kuru var sasniegt algoritms, saņemot ieejā X.

Daži algoritmi var izdod nepareizu rezultātu, izmantojot varbūtiskos lēmumu kokus – saka, ka tādi algoritmi rēķina ar ierobežotu kļūdu. Kopējā varbūtība dabūt pareizo rezultātu ir varbūtība sasniegt lapas ar pareizo vērtību vissliktākajā gadījumā. Rēķinot Būla funkcijas, ir vajadzīgs, lai šī varbūtība p būtu lielāka par ½, jo, acīmredzami, ar varbūtību p ≤ ½ funkcijas vērtību var vienkārši uzminēt, netaisot nevienu vaicājumu. Šajā pētījumā autore izskatīs tikai precīzos algoritmus, t.i. tādus, kuri vienmēr izdod pareizo rezultātu.

Tapāt, ka iepriekšējā gadījumā, algoritma sarežģītība ir vienāda ar attiecīgā optimālā (t.i. ar minimālo dziļumu) lēmumu koka dziļumu; varbūtiska algoritma sarežģītību apzīmēsim ar R(f).

3.2. Kvantu vaicājošie algoritmi

Šajā nodaļā ir aprakstīti kvantu vaicājošie algoritmi.

3.2.1. Kvantu vaicājošo algoritmu modelis

Darbā ir izmantots kvantu vaicājošo algoritmu modelis, kas tika aprakstīts [13].

Funkcijas rēķināšana notiek sekojošos soļos:

1. Sākuma kvantu sistēmas stāvoklis ir $|\vec{0}\rangle$, t.i. $|00..0\rangle$.
2. Seko vairāku unitāro transformāciju ķēde(3.1):

$$U_0, Q_0, U_1, Q_1, ..., U_{T-1}, Q_{T-1}, U_T \tag{3.1}$$

T ir vaicājumu skaits,

Matricas U_i apzīmē kādas fiksētas unitāras transformācijas,

Matricas Q_i apzīmē vaicājumu transformācijas.

Matricas U_i nav atkarīgas no mainīgo vērtībām. Tās var būt jebkuras patvaļīgas unitāras matricas, piemēram, Adamāra matrica.

Matricas Q_i izskatās sekojoši(3.2):

$$Q_i = \begin{pmatrix} (-1)^{\varphi_1} & 0 & \cdots & 0 \\ 0 & (-1)^{\varphi_2} & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{\varphi_m} \end{pmatrix} \tag{3.2}$$

$$\varphi_i \in \{x_1, \dots, x_N, 0, 1\}$$

Atkarībā no φ_i vērtības, bāzes stāvokļa, kas atbilst i-tai rindai, amplitūdas zīme mainās uz pretējo (ja φ_i ir vienāds ar 1) vai paliek nemainīga (ja φ_i ir vienāds ar 0).

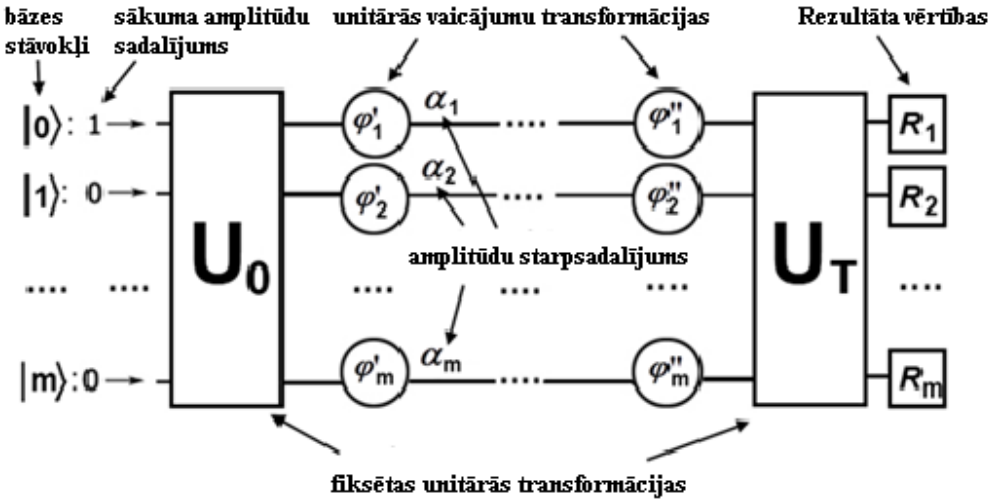
Pēc transformācijām kvantu sistēmas stāvoklis F izskatās tā (3.3).

$$F = U_T\, Q_{T-1} \ldots U_{T1}\, Q_1 U_0\, |\vec{0}\rangle. \tag{3.3}$$

3. Pēdējais solis ir mērījums. Stāvoklis $|i\rangle$ tiek iegūts ar varbūtību $|a_i|^2$. Katrs bāzes stāvoklis tiek piesaistīts kādai vērtībai no funkcijas vērtību kopas. Piemēram, Būla funkcijas gadījumā bāzes stāvokļiem tiek piekārtotās vērtības „0" un „1"; bet multifunkcijas gadījumā – bāzes stāvokļiem vat atbilst vērtības {1...N}. Vienai vērtībai no vērtību kopas var atbilst vairāki bāzes stāvokļi. Lai aprēķināt varbūtību iegūt pēc mērījuma kādu funkcijas vērtību γ, vajag sasummēt visus to amplitūdu moduļu kvadrātus, kuri atbilst bāzes stāvokļiem ar piesaistīto vērtību γ.

3.2.2. Kvantu vaicājošo algoritmu modeļa grafiska attēlošana

Šajā darbā kvantu vaicājošie algoritmi tiek aprakstīti, izmantojot sekojoša veida diagrammas (3.3.att.). Diagrammā tiek attēlots vispārējs gadījums kvantu vaicājošam algoritmam ar T vaicājumiem.



3.3.att. **Diagramma kvantu vaicājoša algoritma attēlošanai [13]**

Diagrammā ir tik horizontālo līniju, cik ir kvantu sistēmas bāzes stāvokļu; katra līnija atbilst attiecīga bāzes stāvokļa amplitūdai. Sākuma stāvoklis ir *|00..0⟩*, tāpēc sākuma amplitūdu sadalījums ir (3.4).

$$|\vec{0}\rangle = \begin{pmatrix} 1 \\ 0 \\ \ldots \\ 0 \\ 0 \end{pmatrix} \tag{3.4}$$

Lielie četrstūri U_0, ..., U_T ir fiksētas unitāras matricas. Vienības matricas var būt izlaistas no zīmējuma. Riņķi, kas atrodas uz vienas vertikālas līnijas, apzīmē vienu vaicājuma transformāciju. Mazie taisnstūri horizontālās līnijas beigās ar R_1, … ,R_m apzīmē funkcijas vērtību, kas ir piesaistīta attiecīgajam kvantu sistēmas bāzes stāvoklim.

3.2.3. Kvantu vaicājošo algoritmu veidi

Iepriekšējā apakšnodaļā, kur bija stāstīts par klasiskajiem vaicājošiem algoritmiem, tika minēts, ka ir algoritmi, kuri rēķina funkciju ar ierobežotu kļūdu, un ir precīzi algoritmi, kuri vienmēr izdod pareizo vērtību. Kvantu vaicājošos algoritmus arī var sadalīt tādos divos veidos.

Definīcija 12. *Kvantu vaicājošs algoritms, kas rēķina funkciju f, ir **precīzs**, ja tas izvada f(X) ar varbūtību p = 1 visiem* $X \in \{0,1\}^N$. *Vaicājoša algoritma sarežģītība ir vienāda ar vaicājumu transformāciju skaitu un tiek apzīmēta ar* $Q_E(f)$ [5].

Definīcija 13. *Kvantu vaicājošs algoritms, kas rēķina funkciju f, rēķina to **ar ierobežoto kļūdu**, ja tas izvada f(X) ar varbūtību* $p \geq ⅔$ *visiem* $X \in \{0,1\}^N$. *Vaicājoša algoritma sarežģītība ir vienāda ar vaicājumu transformāciju skaitu un tiek apzīmēta ar* $Q_P(f)$ [5].

Kā jau bija pateikts iepriekš, šajā darbā tiek pētīti tikai precīzi algoritmi.

Kvantu vaicājošos algoritmus multifunkciju rēķināšanai var sašķirt sekojošās trīs kategorijās.

Definīcija 14. *Saka, ka vaicājošs algoritms rēķina multifunkciju M(X) **noteiktā veidā**, ja katram X tas izvada kādu vienu vērtību no rezultātu kopas ar varbūtību p = 1. Klasiska vaicājoša algoritma sarežģītība tiek apzīmēta ar* $C_D(M)$. *Kvantu vaicājoša algoritma sarežģītība tiek apzīmēta ar* $Q_D(M)$ [13].

Definīcija 15. *Saka, ka vaicājošs algoritms rēķina multifunkciju M(X) **nejauši sadalītā veidā**, ja katram X tas izvada kādu vienu vērtību no rezultātu kopas ar kādu patvaļīgu varbūtību (katrai vērtībai no rezultātu kopas šī varbūtība ir pozitīva) un nekad neizvada nepareizo vērtību. Klasiska vaicājoša algoritma sarežģītība tiek apzīmēta ar* $C_{RD}(M)$. *Kvantu vaicājoša algoritma sarežģītība tiek apzīmēta ar* $Q_{RD}(M)$ [13].

Definīcija 16. *Saka, ka vaicājošs algoritms rēķina multifunkciju M(X) **vienmērīgi sadalītā veidā**, ja katram X katra vērtība no rezultātu kopas var tikt iegūta ar vienādu varbūtību p >0, un vienmēr tiek izvesta pareiza vērtība. Klasiska vaicājoša algoritma sarežģītība tiek apzīmēta ar* $C_{UD}(M)$. *Kvantu vaicājoša algoritma sarežģītība tiek apzīmēta ar* $Q_{UD}(M)$ [13].

Viegli pamanīt, ka multifunkcijas skaitļošana noteiktā veidā ir ļoti līdzīga parastu („vienvērtību”) funkciju skaitļošanai; saņemot vienu un to pašu ieejas bitu virkni, algoritms vienmēr izvada kādu noteiktu vienu un to pašu vērtību. Šīs līdzības dēļ tādi algoritmi šajā darbā netiek pētīti. Darbā tiek uzskatīti algoritmi, kuri rēķinā gan vienmērīgi sadalītā veidā, gan nejauši sadalītā veidā.

Uzkonstruēt algoritmus, kas rēķina multifunkcijas vienmērīgi sadalītā veidā, ir daudz grūtāk, nekā algoritmus, kas rēķina multifunkcijas nejauši sadalītā veidā; bet skaitļošana vienmērīgi sadalītā veidā ir vairāk noderīga vairāku uzdevumu risināšanai, piemēram, tādus algoritmus var lietot, lai sadalīt darba uzdevumus starp darbiniekiem.

Algoritmiem, kas rēķina multifunkcijas nejauši sadalītā veidā, arī var būt praktisks pielietojums. Piemēram, ir daudz dažādu datoru, kuri ir apvienoti vienā klasterī - sākot no veciem, piemēram, Pentium II, un līdz mūsdienu datoriem- i5, i7, utt. HTTP pieprasījumi, kuri tiek nosūtīti klasterim, no sākuma atnāk vienam datoram, un tas pārsūta pieprasījumus tālāk citiem datoriem. Autoraprāt, ir diezgan loģiski pārsūtīt vairāk pieprasījumu moderniem, vairāk jaudīgiem datoriem un mazāk pieprasījumu veciem datoriem.

4. METODOLOĢIJA

Parastas pieejas vietā, t.i. kad no sākuma tiek definēta funkcija un pēc tam tiek meklēts algoritms funkcijas rēķināšanai, autore nolēma"iet pretējā virzienā" – no algoritma līdz (multi-) funkcijai. Autore uzskata, ka tāda pieeja ir diezgan efektīva jaunu netriviālu algoritmu meklēšanai; uzkonstruēt kvantu algoritmu kādas noteiktas funkcijas rēķināšanai ir diezgan grūts uzdevums, bet autores pieejā kvantu algoritmu ģenerēšanai nav nekādu ierobežojumu, tiek noģenerēts pēc iespējas vairāk algoritmu un no tiem tiek paņemti visinteresantākie, t.i. tādi, kas ir efektīvāki par to klasisko analogu.

Automātiskiem aprēķiniem autore izveidoja programmu, izmantojot atklātā pirmkoda matemātisko programmatūru Sage [14]. Programma ir uzrakstīta Python valodā. Rezultāti, kas tiek iegūti ar programmatūras palīdzību, pēc tam tiek manuāli analizēti un klasificēti. Efektīvi kvantu algoritmi tiek meklēti šādos soļos:

1. Sakombinēt dažādas unitāras matricas un vaicājumus. Padot programmai šo unitāro transformāciju ķēdi, kā arī N (mainīgo skaitu) un dažus citus parametrus, un aprēķināt rezultātu. Sīkāk šis solis ir aprakstīts 4.1.apakšnodaļā.
2. Analizēt rezultātus. Salīdzinot ievadi (bināras virknes) un izvadu (attiecīgo kopu, kas tika iegūta ar programmas palīdzību), nodefinēt funkciju.
3. Uzrakstīt orākulu programmai. Pārbaudīt, vai multifunkcijas definīcija ir pareiza lielākajam mainīgo skaitam. Jā kaut-kādi nosacījumi nestrādā, atgriezties pie 2.soļa.
4. Pierādīt tai pašai funkcijai labāka iespējama klasiska vaicājoša algoritma sarežģītību. Kā palīgs šajā soli ir uzrakstīta programma, kas rēķina ar cik jautājumiem noteikti nepietiks, lai aprēķinātu multifunkcijas vērtību. Šīs programmas aprakstu var atrast 4.2.apakšnodaļā.

4.1.Programmas multifunkciju aprēķiniem

Katrai multifunkcijai ir uzrakstīta atsevišķa programma. Programmām ir vienāds „ietvars" – ir daudz vienādu metožu, piemēram, metodes decimālskaitļa konvertēšanai uz bināru sarakstu (list) vai vaicājumu matricu ģenerēšanai no tāda saraksta. Bet katrā programmā ir savas unitārās transformācijas noteiktas multifunkcijas rēķināšanai, kā arī orākuls un dažas transformācijām un orākulam nepieciešamas palīgmetodes. Programmu pirmkodu var atrast sekojošos pielikumos - 1. pielikums, 2. pielikums, 4. pielikums.

Programmas ģenerē visas bināras virknes[1] garumā N (mainīgo skaits), izveido attiecīgas vaicājuma matricas, izdara visus aprēķinus (t.i. sareizina matricas) un izvada rezultātu. Pie tam, programma, kura rēķina attiecīga klasiskā vaicājoša algoritma sarežģītības apakšējo robežu, tika pielikta klāt kā metode šīm programmām.

Programmām var norādīt sekojošus parametrus:

1. N – mainīgo skaits.
2. Size – kvantu sistēmas bāzes stāvokļu skaits.

No mainīgo skaita un bāzes stāvokļu skaita kombinācijas programma saprot, kādas vaicājuma matricas jāveido. Piemēram, multifunkcijas M_1 gadījumā, ja mainīgo skaits ir vienāds ar 16, bet bāzes stāvokļu skaits ir vienāds ar 8, tad būs izmantots M_1 otrais vispārinājums – būs izveidotas četras vaicājošas matricas; bet ja bāzes stāvokļu skaits būs 32, tad būs pielietots M_1 pirmais vispārinājums – ar vienu vaicājošo matricu.

Programmā nav uztaisīta kāda speciāla pārbaude šiem parametriem; bet, ja tiks uzdota nederīga kombinācija, piemēram, mainīgo skaits 8 un bāzes stāvokļu skaits 13, tad programma apstāsies un izdos kļūdu.

3. Formātu, kādā jābūt izvadam.

Pēc noklusējuma multifunkcijas rezultāts tiek parādīts kā kopa ar veseliem skaitļiem. Bet var metodē, kas ir nosaukta „calc_result” , nokonfigurēt to sekojoši:

- „return result” - multifunkcijas rezultāts būs parādīts kā vektors ar amplitūdu sadalījumu;
- „return (one_zero (result))” – tad tiek iegūta izvada bitu virkne;
- „return numbering((one_zero (result)))” – noklusēta vērtība, dod kopu ar veseliem skaitļiem.

Multifunkcijai M_1 vēl jālieto metodi reduce – attiecīgi „return (reduce(one_zero (result)))” vai „return (numbering(reduce(one_zero (result))))”, jo M_1 gadījumā vienai vērtībai no multifunkcijas vērtību kopas atbilst divi bāzes stāvokļi.

4. Oracle – orākuls. To jāuzdod multifunkciju M_1 un M_2 gadījumā. Multifunkcijai M_3 ir tikai viens orākuls.

- Multifunkcijas M_1 „oracle1” darbojas ar izvada bitu virknēm, un tas pārbauda tikai tos nosacījumus, kuri ir definēti jebkuram mainīgo skaitam.

[1] Programma uztaisa binārus sarakstus garumā N, kas atbilst multifunkcijas ievades virknēm

Multifunkcijas M_1 „oracle2” strādā ar veselo skaitļu kopām, un tajā ir iekļauti visi nosacījumi speciālgadījumam ar astoņiem mainīgiem (vai, otrā vispārinājuma gadījumā, to var lietot lielākam mainīgo skaitam, ja kvantu sistēmas bāzes stāvokļu skaits nav lielāks par 16).

- Multifunkcijai M_2 arī ir divi orākuli. Orākulu „oracleN4” jālieto, ja kvantu sistēmai ir četri bāzes stāvokļi, bet orākulu „oracleN8” – attiecīgi astoņiem bāzes stāvokļiem.

Šīs programmas var izmantot jauno kvantu algoritmu ģenerēšanai. Šajā gadījumā metodē „calc_probability” jānorāda jaunas transformācijas. Unitārās matricas var aizvietot ar jebkurām citām unitārām matricām. Ir iespējams ģenerēt arī nejaušus unitāras matricas, piemēram, ar sekojošo metodi:

```
def random_unitary(size, d):
    M = random_matrix(RDF, size, density=d)
    UnitM = M.gram_schmidt()[0]
    return UnitM
```

Vaicājumu matricas tiek ģenerētas no bināriem sarakstiem. Tādos sarakstos var izmantot gan mainīgo vērtības jebkurā secībā, gan fiksētas vērtības „0” vai „1”.

Diemžēl Sage programmatūrā ir ierobežojums izvades simbolu skaitam. Vēl viens programmas trūkums ir tas, ka tā izmanto ļoti daudz atmiņas. Tātad, rēķinot multifunkcijas lielam mainīgo skaitam (~N>16, atkarībā no vēlama izvada un datora jaudas) , vajag palaist programmu vairākas reizes, katru reizi rēķinot mazam bināro virkņu skaitam. Vēl var nomainīt Sage izvades ierobežojumu uz lielāku skaitli.

Lai palielināt atļauto izvades simbolu skaitu, vajag atrast failu ar nosaukumu cell.py direktorijā *$SAGE_ROOT/devel/sagenb-main/sagenb/notebook/*, kur $SAGE_ROOT apzīmē direktoriju, kur ir uzinstalēts Sage. Šajā failā atrast rindas:

- MAX_OUTPUT = 32000
- MAX_OUTPUT_LINES = 120

Šajās rindās vajag nomainīt skaitļus uz lielākiem.

4.2.Programma sarežģītības apakšējās robežai aprēķināšanai

Programmai, kura rēķina klasiskā vaicājoša algoritma sarežģītības apakšējo robežu, jānodod sekojošo informāciju:

1. N – mainīgo skaits.

2. M – vārdnīca (dictionary), kur atslēgas ir bināras virknes garumā N un vērtības ir attiecīgas multifunkcijas vērtību kopas.

Programma dara sekojošo:

1) Ģenerē visas iespējamas kombinācijas noteiktā garumā. Piemēram, ja mainīgo skaits ir vienāds ar četri, un tiek ģenerētas kombinācijas garumā divi, tad noģenerētas kombinācijas būs [0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]

2) Pēc tam tiek ņemtas visas noģenerētas kombinācijas pēc kārtas. Piemēram, pirmā kombinācija ir [0, 1]. Programma iet cauri vārdnīcai M, un meklē atslēgās virknes, kuriem pirmie divi cipari ir nulles – piemēram, četru mainīgo gadījumā, 0000, 0001, 0010, 0011. Programma ņem tādu atslēgu vērtības, t.i. multifunkcijas vērtību tādiem mainīgajiem, un pievieno atsevišķam sarakstam. Jaunizveidotajā sarakstā programma pārbauda, vai ir vismaz viena vērtība, kas ir visās multifunkcijas vērtību kopās.

3) Pēc visu kombināciju pārbaudes, ja nav kādas kopīgas vērtības, tad programma palielina kombināciju izmēru (piemēram ,no divi līdz trīs, un ģenerē visas kombinācijas garumā trīs - [0, 1, 2], [0, 1, 3], [0, 2, 3], [1, 2, 3] un atkal iziet ciklu – izvēlas bināras virknes, kur būtu nulles trīs noteiktās pozīcijās, un salīdzina multifunkcijas vērtību kopas.

4) Ja tiek atrasta kāda kopīga vērtība šādos multifunkcijas vērtību kopās, tad kombināciju garumu var uzskatīt par sarežģītības apakšējo robežu.

Programma pārbauda visas iespējamas kombinācijas, sākot ar kombinācijām garumā „1”, un beidzot ar kombinācijām garumā N (vai kamēr netiek atrasta apakšējā robeža). Protams, šīs opcijas var mainīt.

Nevar apgalvot, ka aprēķināts skaitlis ir klasiskā vaicājoša algoritma sarežģītība. Šī programma nemeklē vissliktāko ievades virkni. Piemēram, ja programma izvada skaitli „4”, tas nozīmē tikai to, ka sarežģītība ir noteikti lielāka par „3”; eksistē vismaz viena ievades virkne, kurai noteikti nevar aprēķināt multifunkcijas vērtību, jautājot tikai trīs mainīgo vērtības.

6.pielikumā ir programmas pirmkods ar aizpildītu vārdnīcu multifunkcijai M_1 ar astoņiem mainīgiem.

5. UZKONSTRUĒTIE KVANTU ALGORITMI

Šajā nodaļā tiek aprakstīti galvenie pētījuma rezultāti.

Uzkonstruēto kvantu algoritmu speciālgadījumiem tiek dots arī programmas darbības laiks. Programmas ir darbinātas ar datoru, kura specifikācija ir sekojoša:

- Procesors - Intel Core(TM) i5-2430M ar 2,40GHz frekvenci
- RAM - 8,00GB

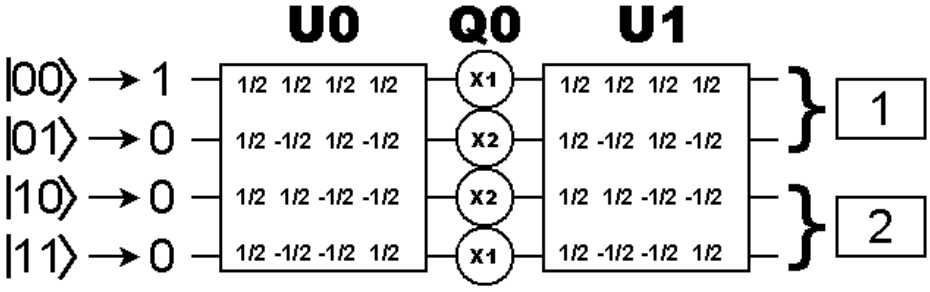
5.1.Pirmais piemērs - Algoritms multifunkcijas M_1 rēķināšanai

Šī nodaļa sākas ar aprakstu, kā multifunkcija $M_1(X): \{0,1\}^N \rightarrow \{1 \dots N\}, N = 2^i, i \in \mathbb{N}^+$ izskatās mazam mainīgo skaitam, bet tālāk tiks apskatīti M_1 vispārinājuma veidi.

Multifunkcijas M_1 definīcija ir diezgan sarežģīta, tāpēc tās aprakstam seko piemēri, kuri demonstrē rēķināšanas soļus.

5.1.1. Multifunkcija M_1 ar diviem mainīgajiem

Tā kā multifunkcija M_1 tiek definēta rekursīvi, multifunkcijas apraksts sākas ar aprēķinu divu mainīgo gadījumā (sk.5.1.att.).



5.1.att. **Kvantu algoritms, kas rēķina multifunkciju M_1 ar diviem mainīgajiem**

5.1. tabulā tiek parādīts skaitļošanas process – kā kvantu sistēma izskatās pēc katras transformācijas, kā arī multifunkcijas vērtība katram ievades vektoram.

Kā redzams, šajā gadījumā multifunkcija $M_1(X): \{0,1\}^2 \rightarrow \{1,2\}$ tiek rēķināta vienmērīgi sadalītā veidā. Viegli pamanīt arī to, ka M_1 ar diviem mainīgajiem ir līdzīga XOR funkcijai.

5.1.tabula

Kvantu algoritma skaitļošanas process multifunkcijai M_1 ar diviem mainīgajiem

X	Stāvoklis pēc U0	Stāvoklis pēc Q0	Stāvoklis pēc U1	$M_1(X)$
00	(1/2, 1/2, 1/2, 1/2)	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	{ 1 }
01	(1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	{ 2 }
10	(1/2, 1/2, 1/2, 1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	{ 2 }
11	(1/2, 1/2, 1/2, 1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	{ 1 }

Teorēma 1. $Q_{UD}(M_1) = 1$ gadījumā, ja mainīgo skaits ir vienāds ar 2.

Pierādījums. 5.1.att. ir redzams, ka pietiek ar vienu vaicājumu.

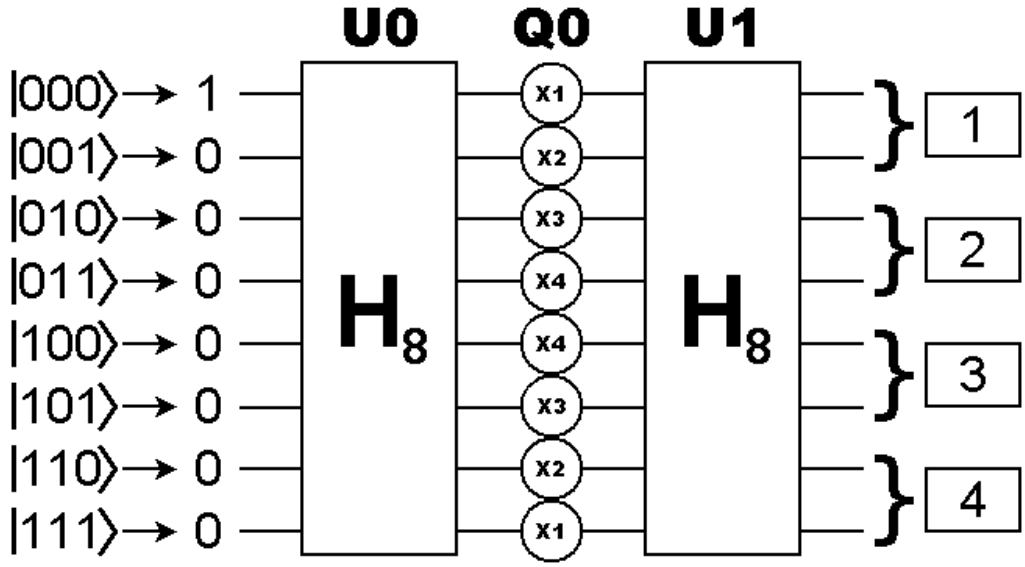
Teorēma 2. $C_{UD}(M_1) = 2$ gadījumā, ja mainīgo skaits ir vienāds ar 2.

Pierādījums. Acīmredzami, ka ar vienu vaicājumu nepietiks. Piemēram, ja tiek jautāta x_1 vērtība, nav zināms, vai x_2 ir vai nav vienāds ar x_1. Ja x_1 un x_2 ir vienādi, izvadā jābūt „1", ja tie ir dažādi – jābūt „2".

Programmas darbības laiks: <1 sekundes.

5.1.2. Multifunkcija M_1 ar četriem mainīgajiem

Ja multifunkcija M_1 tiek izrēķināta lielākam mainīgo skaitam, ir redzams, ka tā tomēr atšķiras no XOR funkcijas.



5.2.att. **Kvantu algoritms, kas rēķina multifunkciju M_1 ar četriem mainīgajiem**

5.2.tabulā var redzēt, kā mainās kvantu sistēmas stāvoklis skaitļošanas laikā.

5.2.tabula

Kvantu algoritma skaitļošanas process multifunkcijai M_1 ar četriem mainīgajiem

X	Stāvoklis pēc U0	Stāvoklis pēc Q0	Stāvoklis pēc U1	$M_1(X)$
0000	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	{ 1 }
0001	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ -1/4*sqrt(2) \\ -1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 0 \\ 0 \\ -1/2 \\ 0 \\ 1/2 \\ 1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
0010	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ -1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ -1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ -1/2 \\ 1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
0011	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4*sqrt(2) \\ 1/4*sqrt(2) \\ -1/4*sqrt(2) \\ -1/4*sqrt(2) \\ -1/4*sqrt(2) \\ -1/4*sqrt(2) \\ 1/4*sqrt(2) \\ 1/4*sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	{ 4 }

0100	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ 1/2 \\ -1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
0101	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	{ 3 }
0110	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	{ 2 }
0111	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ 1/2 \\ 1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
1000	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 1/2 \\ 0 \\ 0 \\ -1/2 \\ 0 \\ -1/2 \\ -1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }

1001	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	{ 2 }
1010	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$	{ 3 }
1011	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/2 \\ 0 \\ 0 \\ -1/2 \\ 0 \\ -1/2 \\ 1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
1100	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{pmatrix}$	{ 4 }
1101	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/2 \\ 0 \\ 0 \\ -1/2 \\ 0 \\ 1/2 \\ -1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }

1110	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/2 \\ 0 \\ 0 \\ 1/2 \\ 0 \\ -1/2 \\ -1/2 \\ 0 \end{pmatrix}$	{ 1, 2, 3, 4 }
1111	$\begin{pmatrix} 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \\ 1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \\ -1/4 * sqrt(2) \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	{ 1 }

$M_1(X)$: $\{0,1\}^4 \rightarrow \{1,2,3,4\}$ var definēt, izmantojot sekojošus nosacījumus:

a) Ja X satur nepāra nuļļu skaitu (un, līdz ar to, arī nepāra vienīnieku skaitu), tad

$$M_1(X) = \{1...N\} \tag{5.1}$$

b) Ja X ir simetrisks, tad

$$M_1(X) = M_1(X^I) = M_1(X^{II}) \tag{5.2}$$

c) Ja X ir pusvienāds, bet nav simetrisks, tad

$$M_1(X) = res_1 \bigcup res_2, \text{ kur} \tag{5.3}$$
$$res_1 = \{i | i \in M_1(X^I) \,\&\, i \leq N/4\}$$
$$res_2 = \{i + N/4 | i \in M_1(X^I) \,\&\, i > N/4\}$$

d) Ja X ir puspretējs, tad

$$M_1(X) = res_1 \bigcup res_2, \text{ kur} \tag{5.4}$$
$$res_1 = \{i | i \in M_1(X^I) \,\&\, i > N/4\}$$
$$res_2 = \{i + 3N/4 | i \in M_1(X^I) \,\&\, i \leq N/4\}$$

Tā kā M_1 definīcija var izskatīties sarežģīta, 5.3.tabulā tiek doti piemēri multifunkcijas aprēķinam. Kopā ar ievades bitu virkni un multifunkcijas rezultātu, tiek norādīts, kāds nosacījums tiek lietots un tiek parādīti aprēķina soļi.

5.3.tabula

Apraksts multifunkcijai M_1 ar četriem mainīgajiem

X	$M_1(X)$	Nosacījums	Aprēķina soļi
0000	{1}	b)	$M_1(0000) = M_1(00) = \{1\}$
0001	{1, 2, 3, 4}	a)	$M_1(0001) = \{1, 2, 3, 4\}$
0010	{1, 2, 3, 4}	a)	$M_1(0010) = \{1, 2, 3, 4\}$
0011	{4}	d)	$M_1(00) = M_1(11) = \{1\}$ Tā kā $1 \leq 4/4$, res1 = {} res2 = {1+4×3/4} = {1 + 3} = {4} Tad $M_1(0011) = \{4\}$
0100	{1, 2, 3, 4}	a)	$M_1(0100) = \{1, 2, 3, 4\}$
0101	{3}	c)	$M_1(01) = \{2\}$ Tā ka $2 > 4/4$, res1 = {} res2 = {2 + 4/4} = {2+1} = {3} Tad $M_1(0101) = \{3\}$
0110	{2}	b)	$M_1(0110) = M_1(01) = M_1(10) = \{2\}$
0111	{1, 2, 3, 4}	a)	$M_1(0111) = \{1, 2, 3, 4\}$
1000	{1, 2, 3, 4}	a)	$M_1(1000) = \{1, 2, 3, 4\}$
1001	{2}	b)	$M_1(1001) = M_1(10) = M_1(01) = \{2\}$
1010	{3}	c)	$M_1(10) = \{2\}$ Tā kā $2 > 1$, res1 = {} res2 = {2 + 4/4} = {2+1} = {3} Tad $M_1(0101) = \{3\}$
1011	{1, 2, 3, 4}	a)	$M_1(1011) = \{1, 2, 3, 4\}$
1100	{4}	d)	$M_1(11) = M(00) = \{1\}$ Tā kā $1 \leq 4/4$, res1 = {}, res2 = {1+4×3/4} = {1 + 3} = {4} Tad $M_1(1100) = \{4\}$
1101	{1, 2, 3, 4}	a)	$M_1(1101) = \{1, 2, 3, 4\}$
1110	{1, 2, 3, 4}	a)	$M_1(1110) = \{1, 2, 3, 4\}$
1111	{1}	b)	$M_1(1111) = M_1(11) = \{1\}$

Arī šajā gadījumā multifunkcija M_1 tiek rēķināta vienmērīgi sadalītā veidā.

Teorēma 3. $Q_{UD}(M_1) = 1$ gadījumā, ja mainīgo skaits ir vienāds ar 4.

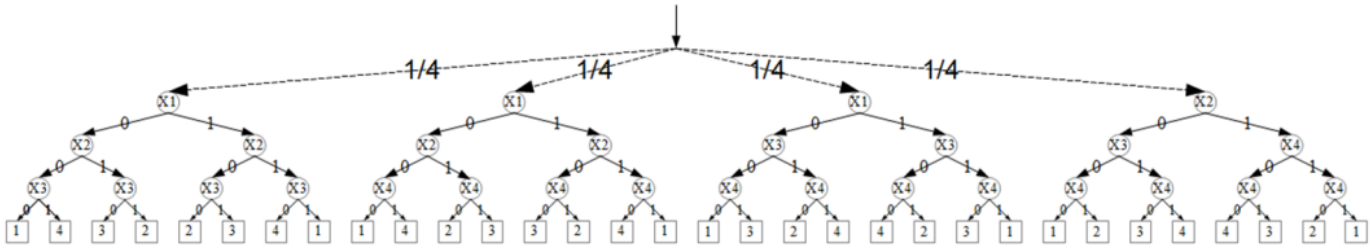
Pierādījums. Sk. 5.2.att.

Teorēma 4. $C_{UD}(M_1) = 3$ gadījumā, ja mainīgo skaits ir vienāds ar 4.

Pierādījums. Kā ir redzams no programmas izvades, apakšējā sarežģītības novērtējuma robeža ir vienāda ar trīs.

Tātad, tas nozīmē, ka, vaicājot divas reizes un dabūjot divas nulles, nevar viennozīmīgi noteikt multifunkcijas vērtību. Ja ievadē ir visas nulles, tad multifunkcijas vērtība ir {1}. Bet ja ievadē ir divas nulles un divi vieninieki, multifunkcijas vērtība atšķiras no {1}. Piemēram, ja ir zināms, ka x_1 un x_2 abi ir vienādi ar "0", algoritms nevar būt drošs, vai ievadē ir 0000, 0011, 0001 vai 0001. Ja izvadē būs {1}, bet ievadē bija 0011, ir kļūda. Bet ja izvadē būs {4}, bet ievadē bija 0000, atkal tiek iegūts kļūdains rezultāts. Vienalga, kuru divu mainīgo vērtības algoritms uzzina - x_2 un x_4, x_1 un x_4, utt. Acīmredzami, ka tam vajag uzzināt kādu trešo mainīgu, lai pareizi aprēķinātu funkciju.

Ir iespējams pierādīt arī to, ka ar trīm vērtībām pietiek, lai izdotu pareizo rezultātu. 5.3.att. tiek attēlots varbūtisks lēmumu koks, kas rēķina multifunkciju M_1 vienmērīgi sadalītā veidā.



5.3.att. **Lēmumu koks, kas rēķina multifunkciju M_1 ar četriem mainīgiem**

Programmas darbības laiks: apmēram 6 sekundes.

5.1.3. Multifunkcija M_1 ar astoņiem mainīgajiem

Ja paplašināt algoritmu līdz astoņiem mainīgajiem, papildus nosacījumiem a)-d) no 5.1.2.punkta, var pievienot sekojošos nosacījumus, lai nodefinētu pārējos multifunkcijas $M_1(X)$: $\{0,1\}^8 \rightarrow \{1, \dots, 8\}$ ieejas datus:

e) Ja X^I ir vienāds ar apgrieztu pretējā secībā X^{II} pretējo virkni, un iepriekš minētie nosacījumi nevar tikt piemēroti X:

$$M_1(X) = \{N + 1 - i | i \in M_1(X^I)\} \tag{5.5}$$

Šo nosacījumu var nodefinēt arī tā:

$$M_1(X) = \{i + 4 | i \in M_1(X^I)\} \quad (5.6)$$

f) Ja X^I satur pāra vieninieku skaitu (un, līdz ar to, arī pāra nuļļu skaitu) un X^{II} arī satur pāra vieninieku skaitu (un, līdz ar to, arī pāra nuļļu skaitu), un iepriekš minētie nosacījumi nevar tikt piemēroti X:

$$M_1(X) = res_1 \bigcup res_2, \text{ kur} \quad (5.7)$$

$$res_1 = M_1(X^I) \bigcup M_1(X^{II}) \quad (5.8)$$

$$res_2 = shift(M_1(X^I) \bigcup M_1(X^{II}), 2) + N/2 \quad (5.9)$$

g) Ja X^I satur vienu vieninieku (vai vienu nulli) un X^{II} arī satur tādu pašu vieninieku un nuļļu skaitu , un iepriekš minētie nosacījumi nevar tikt piemēroti X:

$$M_1(X) = res_1 \bigcup res_2, \text{ kur} \quad (5.10)$$

$$res_1 = 1 \bigcup (2 + S), \quad (5.11)$$

S – starpība starp vieninieku (nuļļu) pozīcijām X^I un X^{II} [2]

$$res_2 = N - res_1 + 1, \text{ ja mēs varam satikt pēc kārtas} \quad (5.12)$$

četras nulles (vai četrus vieniniekus) , uztaisot ciklu caur X, un ne vairāk par četrām nullēm (četriem vieniniekiem)[3]

$$res_2 = res_1 + N/2 \text{ citos gadījumos} \quad (5.13)$$

h) Ja X^I un X^{II} satur nepāra (vieninieku un nepāra nuļļu) skaitu, bet vieninieku (un nuļļu) skaits X^I un X^{II} atšķiras, un iepriekš minētie nosacījumi nevar tikt piemēroti X, tad, lai izrēķinātu $M_1(X)$, no sākuma vajag dabūt tādu kopu - X^{II} vajag aizvietot ar pretējo no X^{II}, tad vajag aprēķināt rezultātu, izmantojot nosacījumu g). Šādu kopu var apzīmēt ar $Inv(M_1(\overline{x}))$. Tad

$$M_1(X) = \{1...N\} \backslash Inv(M_1(\overline{x})) \quad (5.14)$$

Tagad autore sniedz dažus piemērus, lai lasītājam būtu vieglāk saprast šos nosacījumus.

$M_1(11101010) = \{1, 2, 3, 4, 5, 6, 7, 8\}$ (a)

$M_1(00000000) = M_1(0000) = \{1\}$ (b)

$M_1(00011000) = M_1(0001) = M_1(1000) = \{1, 2, 3, 4\}$ (b)

$M_1(00010001) = \{1, 2\} \cup \{3 + 8/4, 4 + 8/4\} = \{1, 2, 5, 6\}$, jo $M_1(0001) = \{1, 2, 3, 4\}$ (c)

$M_1(00011110) = \{3, 4\} \cup \{1 + 8 \times 3/4, 2 + 8 \times 3/4\} = \{3, 4, 7, 8\}$, jo $M_1(0001) = M_1(1110) = \{1, 2, 3, 4\}$ (d)

[2] Piemēram, ja X = 00101000, tad vieninieks X^I atrodas 3.pozīcijā, un X^{II} – 1.pozīcija; S = 3-1 = 2.

[3] Piemēram, X = 00100001 vai X = 01001000

$M_1(11101000) = \{8\text{-}1\text{+}1,\ 8\ \text{-}\ 2\text{+}1,\ 8\ \text{-}\ 3\text{+}1,\ 8\ \text{-}\ 4\text{+}1\} = \{5, 6, 7, 8\}$ (e)

$M_1(00111001) = \{4\} \cup \{2\} \cup \{(4 + 2)\text{mod } 4 + 8/2\} \cup \{(2 + 2)\text{mod } 4 + 4 + 8/2\}$[4] = $\{2, 4, 6, 8\}$, jo $M_1(0011) = \{4\}$ un $M_1(1001) = \{2\}$ (f)

$M_1(11001010) = \{4\} \cup \{3\} \cup \{(4 + 2)\text{mod } 4 + 8/2\} \cup \{(3 + 2)\text{mod } 4 + 8/2\} = \{3, 4, 5, 6\}$, jo $M_1(1100) = \{4\}$ un $M_1(1010) = \{3\}$ (f)

Nosacījumu g) izmanto sekojoši:

Ja X = 00101000, tad
$X^I = 0010$ un 1 ir 3.pozīcijā.
$X^{II} = 1000$ un 1 ir 1.pozīcijā.
Tātad, $S = 3 - 1 = 2$:
$res_1 = \{1, 2 + 2\} = \{1, 4\}$
Tā kā ieejas datos X ir piecas nulles pēc kārtas (pozīcijās 6,7,8,1,2), tad:
$res_2 = \{1 + 4, 4 + 4\} = \{5, 8\}$ (5.13)
$M_1(00101000) = \{1, 4, 5, 8\}$ (g)

Ja X = 01001000, tad
$X^I = 0100$ un 1 ir 2.pozīcijā.
$X^{II} = 1000$ un 1 ir 1.pozīcijā.
Līdz ar to, $S = 2 - 1 = 1$:
$res_1 = \{1, 3\}$
Ieejas datos X ir četras nulles pēc kārtas (pozīcijās 6,7,8,1), tātad:
$res_2 = \{8 - 1 + 1, 8 - 3 + 1\} = \{8, 6\}$ (5.12)
$M_1(01001000) = \{1, 3, 6, 8\}$ (g)

Tagad tiek paskaidrots nosacījums h).

Ja X = 00101110, tad no sākuma tiek rēķināts $\mathrm{Inv}(M_1(\bar{x}))$.
$\mathrm{Inv}(M_1(\bar{x})) = M_1(00100001) = \{1, 3, 6, 8\}$ (izmantojot nosacījumu g)
$M_1(00101110) = \{1, 2, 3, 4, 5, 6, 7, 8\} \setminus \{1, 3, 6, 8\} = \{2, 4, 5, 7\}$ (h)

Ja X = 00100111, tad
$\mathrm{Inv}(M_1(\bar{x})) = M_1(00101000) = \{1, 4, 5, 8\}$ (izmantojot nosacījumu g)
$M_1(00100111) = \{1, 2, 3, 4, 5, 6, 7, 8\} \setminus \{1, 4, 5, 8\} = \{2, 3, 6, 7\}$ (h)

5.4.tabulā var apskatīties, kādi nosacījumi un kādas multifunkcijas vērtību kopas atbilst ievades virknēm ar pāra vienīnieku skaitu.

Ja ievadē X ir nepāra vienīnieku skaits, tad (5.15).

$$M_1(X) = \{1, 2, 3, 4, 5, 6, 7, 8\} \tag{5.15}$$

Ar lielāku, nekā četri, mainīgo skaitu multifunkcija M_1 tiek rēķināta nejauši sadalītā veidā.

[4] $\{(2 + 2)\text{mod } 4 + 4\} = \{4\ \text{mod}^{\Psi} 4\}$

5.4.tabula

Apraksts multifunkcijai M_1 ar astoņiem mainīgajiem

X	$M_1(X)$	Nosacījums	X	$M_1(X)$	Nosacījums
00000000	{ 1 }	b)	10000001	{ 1, 2, 3, 4 }	b)
00000011	{ 1, 4, 6, 7 }	f)	10000010	{ 1, 4, 5, 8 }	g)
00000101	{ 1, 3, 5, 7 }	f)	10000100	{ 1, 3, 6, 8 }	g)
00000110	{ 1, 2, 7, 8 }	f)	10000111	{ 3, 4, 7, 8 }	d)
00001001	{ 1, 2, 7, 8 }	f)	10001000	{ 1, 2, 5, 6 }	c)
00001010	{ 1, 3, 5, 7 }	f)	10001011	{ 2, 4, 5, 7 }	h)
00001100	{ 1, 4, 6, 7 }	f)	10001101	{ 2, 3, 6, 7 }	h)
00001111	{ 7 }	d)	10001110	{ 5, 6, 7, 8 }	e)
00010001	{ 1, 2, 5, 6 }	c)	10010000	{ 1, 2, 7, 8 }	f)
00010010	{ 1, 3, 6, 8 }	g)	10010011	{ 2, 4, 6, 8 }	f)
00010100	{ 1, 4, 5, 8 }	g)	10010101	{ 2, 3, 5, 8 }	f)
00010111	{ 5, 6, 7, 8 }	e)	10010110	{ 8 }	d)
00011000	{ 1, 2, 3, 4 }	b)	10011001	{ 2 }	b)
00011011	{ 2, 3, 6, 7 }	h)	10011010	{ 2, 3, 5, 8 }	f)
00011101	{ 2, 4, 5, 7 }	h)	10011100	{ 2, 4, 6, 8 }	f)
00011110	{ 3, 4, 7, 8 }	d)	10011111	{ 1, 2, 7, 8 }	f)
00100001	{ 1, 3, 6, 8 }	g)	10100000	{ 1, 3, 5, 7 }	f)
00100010	{ 1, 2, 5, 6 }	c)	10100011	{ 3, 4, 5, 6 }	f)
00100100	{ 1, 2, 3, 4 }	b)	10100101	{ 3 }	b)
00100111	{ 2, 3, 6, 7 }	h)	10100110	{ 2, 3, 5, 8 }	f)
00101000	{ 1, 4, 5, 8 }	g)	10101001	{ 2, 3, 5, 8 }	f)
00101011	{ 5, 6, 7, 8 }	e)	10101010	{ 5 }	c)
00101101	{ 3, 4, 7, 8 }	d)	10101100	{ 3, 4, 5, 6 }	f)
00101110	{ 2, 4, 5, 7 }	h)	10101111	{ 1, 3, 5, 7 }	f)
00110000	{ 1, 4, 6, 7 }	f)	10110001	{ 2, 3, 6, 7 }	h)
00110011	{ 6 }	c)	10110010	{ 5, 6, 7, 8 }	e)
00110101	{ 3, 4, 5, 6 }	f)	10110100	{ 3, 4, 7, 8 }	d)

00110110	{ 2, 4, 6, 8 }	f)	10110111	{ 1, 3, 6, 8 }	g)
00111001	{ 2, 4, 6, 8 }	f)	10111000	{ 2, 4, 5, 7 }	h)
00111010	{ 3, 4, 5, 6 }	f)	10111011	{ 1, 2, 5, 6 }	c)
00111100	{ 4 }	b)	10111101	{ 1, 2, 3, 4 }	b)
00111111	{ 1, 4, 6, 7 }	f)	10111110	{ 1, 4, 5, 8 }	g)
01000001	{ 1, 4, 5, 8 }	g)	11000000	{ 1, 4, 6, 7 }	f)
01000010	{ 1, 2, 3, 4 }	b)	11000011	{ 4 }	b)
01000100	{ 1, 2, 5, 6 }	c)	11000101	{ 3, 4, 5, 6 }	f)
01000111	{ 2, 4, 5, 7 }	h)	11000110	{ 2, 4, 6, 8 }	f)
01001000	{ 1, 3, 6, 8 }	g)	11001001	{ 2, 4, 6, 8 }	f)
01001011	{ 3, 4, 7, 8 }	d)	11001010	{ 3, 4, 5, 6 }	f)
01001101	{ 5, 6, 7, 8 }	e)	11001100	{ 6 }	c)
01001110	{ 2, 3, 6, 7 }	h)	11001111	{ 1, 4, 6, 7 }	f)
01010000	{ 1, 3, 5, 7 }	f)	11010001	{ 2, 4, 5, 7 }	h)
01010011	{ 3, 4, 5, 6 }	f)	11010010	{ 3, 4, 7, 8 }	d)
01010101	{ 5 }	c)	11010100	{ 5, 6, 7, 8 }	e)
01010110	{ 2, 3, 5, 8 }	f)	11010111	{ 1, 4, 5, 8 }	g)
01011001	{ 2, 3, 5, 8 }	f)	11011000	{ 2, 3, 6, 7 }	h)
01011010	{ 3 }	b)	11011011	{ 1, 2, 3, 4 }	b)
01011100	{ 3, 4, 5, 6 }	f)	11011101	{ 1, 2, 5, 6 }	c)
01011111	{ 1, 3, 5, 7 }	f)	11011110	{ 1, 3, 6, 8 }	g)
01100000	{ 1, 2, 7, 8 }	f)	11100001	{ 3, 4, 7, 8 }	d)
01100011	{ 2, 4, 6, 8 }	f)	11100010	{ 2, 4, 5, 7 }	h)
01100101	{ 2, 3, 5, 8 }	f)	11100100	{ 2, 3, 6, 7 }	h)
01100110	{ 2 }	b)	11100111	{ 1, 2, 3, 4 }	b)
01101001	{ 8 }	d)	11101000	{ 5, 6, 7, 8 }	e)
01101010	{ 2, 3, 5, 8 }	f)	11101011	{ 1, 4, 5, 8 }	g)
01101100	{ 2, 4, 6, 8 }	f)	11101101	{ 1, 3, 6, 8 }	g)
01101111	{ 1, 2, 7, 8 }	f)	11101110	{ 1, 2, 5, 6 }	c)
01110001	{ 5, 6, 7, 8 }	e)	11110000	{ 7 }	d)
01110010	{ 2, 3, 6, 7 }	h)	11110011	{ 1, 4, 6, 7 }	f)

01110100	{ 2, 4, 5, 7 }	h)	11110101	{ 1, 3, 5, 7 }	f)
01110111	{ 1, 2, 5, 6 }	c)	11110110	{ 1, 2, 7, 8 }	f)
01111000	{ 3, 4, 7, 8 }	d)	11111001	{ 1, 2, 7, 8 }	f)
01111011	{ 1, 3, 6, 8 }	g)	11111010	{ 1, 3, 5, 7 }	f)
01111101	{ 1, 4, 5, 8 }	g)	11111100	{ 1, 4, 6, 7 }	f)
01111110	{ 1, 2, 3, 4 }	b)	11111111	{ 1 }	b)

Teorēma 5. $Q_{RD}(M_1) = 1$ gadījumā, ja mainīgo skaits ir vienāds ar 8.

Pierādījums. Salīdzinot ar iepriekšējiem gadījumiem, tiek palielināts kvantu sistēmas bāzes stāvokļu skaits, bet joprojām tiek darīts tikai viens vaicājums – visi mainīgie tiek piesaistīti pēc kārtas un pēc tam pretējā secībā.

Teorēma 6. $C_{RD}(M_1) \geq 5$ gadījumā, ja mainīgo skaits ir vienāds ar 8.

Pierādījums. Saskaņā ar programmas aprēķinu, multifunkcijas klasiskā sarežģītība nav mazāka par 5. Ja četri zināmie mainīgie ir vienādi, vajag vismaz vēl vienu piekto vaicājumu, lai viennozīmīgi noteikt vismaz vienu skaitli no multifunkcijas vērtību kopas.

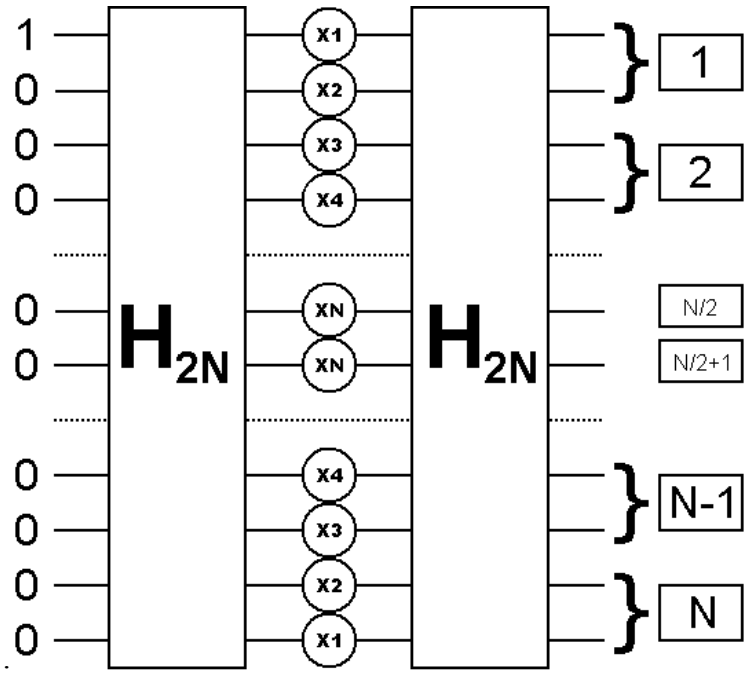
Programmas darbības laiks: apmēram 1 minūte.

5.1.4. Multifunkcijas M_1 pirmais vispārinājums

Pirmais multifunkcijas M_1 vispārinājuma veids, kurš joprojām ir apzīmēts ar M_1, $M_1(X): \{0,1\}^N \rightarrow \{1 \dots N\}, N = 2^i, i \in \mathbb{N}$, ir parādīts 5.4.att.

Šis algoritms var likties līdzīgs algoritmam,kas rēķina multifunkciju M_2 no [15], bet tomēr šī darba autore uzskata šo algoritmu par atsevišķu pilnvērtīgu algoritmu, nevis algoritma no [15] modifikāciju.

Kvantu sistēma sastāv no 2N bāzes stāvokļiem. Pirmā transformācija ir $2N \times 2N$ Adamāra matrica. Pēc tam viena un vienīga vaicājuma transformācija. Vaicājumam mainīgie tiek piesaistīti secīgi un pēc tam pretējā secībā. Pēdējā transformācija atkal ir $2N \times 2N$ Adamāra matrica. Multifunkcijas M_1 vērtība r_i tiek piesaistīta bāzes stāvokļiem $(r_i\text{-}1)*2+1$ un $(r_i\text{-}1)*2+2$; tātad, divi bāzes stāvokļi atbilst vienai un tai pašai multifunkcijas vērtībai.



5.4.att. **Kvantu algoritms, kas rēķina multifunkciju M_1**

Šajā gadījumā multifunkcija tiek definēta tikai daļēji, izmantojot augstāk minētos nosacījumus a)-d). Piemēram,

$M_1(11100011111000110000000010000000) = \{1 \dots 32\}$ (a)

$M_1(1110001111100011) = \{1, 2, 3, 4\} \cup \{5 + 16/4, 6 + 16/4, 7 + 16/4, 8 + 16/4\} =$
$\{1, 2, 3, 4, 9, 10, 11, 12\}$, jo $M(11100011) = \{1, 2, 3, 4, 5, 6, 7, 8\}$ (c)

$M_1(1100001111000011) = \{4\}$, jo $M(11000011) = \{4\}$ (b)

$M_1(1110001100011100) = \{5, 6, 7, 8\} \cup \{1 + 12, 2 + 12, 3 + 12, 4 + 12\} =$
$\{5, 6, 7, 8, 13, 14, 15, 16\}$, jo $M_1(11100011) = M_1(00011100) = \{1,2,3,4,5,6,7,8\}$ (d)

$M_1(1100001100111100) = \{4 + 12\} = \{16\}$, jo $M_1(11000011) = M_1(00111100) = \{4\}$ (d)

Teorēma 7. $Q_{RD}(M_1) = 1$.

Pierādījums. Sk. 5.4.att.

Teorēma 8. $C_{RD}(M_1) \geq N/2 + 1$.

Pierādījums. Ir iespējams izmantot pierādījumu, kas tika izmantots [15] multifunkcijas M_2 otrā vispārinājuma gadījumā.

Ja ievadē visi mainīgie ir vienādi ar 0, tad vaicājuma transformācija ir vienkārši vienības matrica. Ir skaidri redzams, ka šajā gadījumā multifunkcijas vērtībai jābūt vienādai ar$\{1\}$.

Vispārīgā gadījumā pirms mērīšanas stāvokļa $|00\dots0\rangle$ amplitūda izskatās kā (5.16):

$$\alpha_1 = \frac{(-1)^{x_1} + (-1)^{x_2} + \dots + (-1)^{x_N}}{N} \qquad (5.16)$$

Ja puse no ievades mainīgiem ir vienādi ar 0, un otrā puse – ar 1, tad izvadē jābūt kādam skaitlim, kas ir atšķirīgs no „1”, jo stāvokļa $|00...0\rangle$ amplitūda šajā gadījumā ir vienāda ar 0. Tātad, klasiskajam algoritmam arī nedrīkst izvadīt „1” gadījumā, ja tieši puse no mainīgajiem ir vienāda ar 0, un puse – ar 1; bet ja visi mainīgie ir vienādi, rezultātā jābūt tikai un vienīgi „1”. No tā seko, ka, zinot tikai N/2 mainīgo vērtības, nevar precīzi uzzināt multifunkcijas vērtību.

5.1.5. Multifunkcijas M_1 otrais vispārinājums

Otrā vispārinājuma metode ir līdzīga tai, kas tika izmantota [16]. Definīcijās, kas tika izmantotas iepriekš, katru mainīgo var aizvietot ar XOR no vairākiem mainīgajiem.

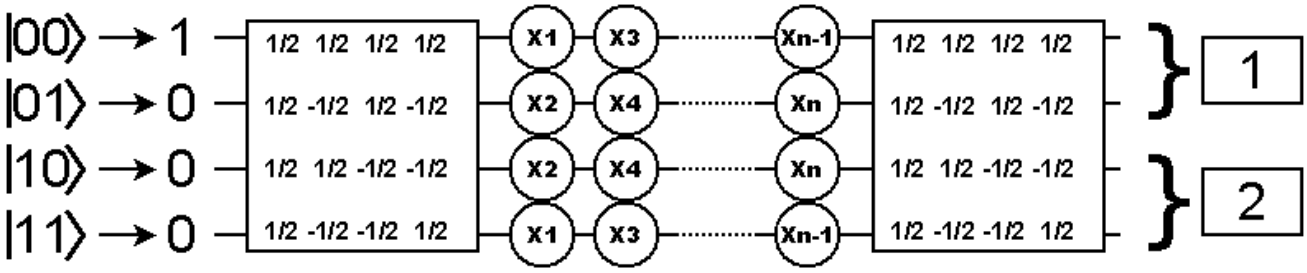
Piemēram, ja katrā vaicājumā jautāt pa diviem mainīgajiem, tā, kā parādīts 5.5. att., var dabūt sekojošo multifunkciju $M_1{}^{Gen2/2}(X)\colon\{0,1\}^N \to 2, N = 2^i, i \in \mathbb{N}^+$:

I. Ja XOR(x_1, x_3, x_5, ...,x_{N-1}) ir vienāds ar XOR(x_2, x_4, x_6, ...,x_N), tad

$$M_1{}^{Gen2/2}(X)\colon\{0,1\}^N \to 1 \tag{5.17}$$

II. Ja XOR(x_1, x_3, x_5, ...,x_{N-1}) nav vienāds ar XOR(x_2, x_4, x_6, ...,x_N), tad

$$M_1{}^{Gen2/2}(X)\colon\{0,1\}^N \to 2 \tag{5.18}$$



5.5. att. **Kvantu algoritms, kas rēķina multifunkciju $M_1^{Gen2/2}$ ar N/2 vaicājumiem**

Autore apzīmē ar X’ tādu vektoru (5.19):

$$X' = (a, b, c, d) \tag{5.19}$$

$$\mathrm{a} = \mathrm{XOR}(\mathrm{x}_1, \mathrm{x}_5, \mathrm{X}_9, ..., \mathrm{x}_{\mathrm{N}-3})$$

$$\mathrm{b} = \mathrm{XOR}(\mathrm{x}_2, \mathrm{x}_6, \mathrm{X}_{10}, ..., \mathrm{x}_{\mathrm{N}-2})$$

$$\mathrm{c} = \mathrm{XOR}(\mathrm{x}_3, \mathrm{x}_7, \mathrm{X}_{11}, ..., \mathrm{x}_{\mathrm{N}-1})$$

$$\mathrm{d} = \mathrm{XOR}(\mathrm{x}_4, \mathrm{x}_8, \mathrm{X}_{12}, ..., \mathrm{x}_{\mathrm{N}})$$

Ja katrā vaicājumā būs četri mainīgie, tā, kā parādīts 5.6.att., var dabūt sekojošo multifunkciju $M_1{}^{Gen2/4}(X)\colon\{0,1\}^N \to 4, N = 2^i, i \in \mathbb{N}^+$:

I. Ja X (kā arī X’) satur nepāra nuļļu skaitu (un, līdz ar to, arī nepāra vieninieku skaitu), tad

$$\mathrm{M}_1^{\mathrm{Gen2/4}}(\mathrm{X}) = \mathrm{M}_1^{\mathrm{Gen2/4}}(\mathrm{X'}) = \{1, 2, 3, 4\} \tag{5.20}$$

II. Ja X' ir simetrisks, tad

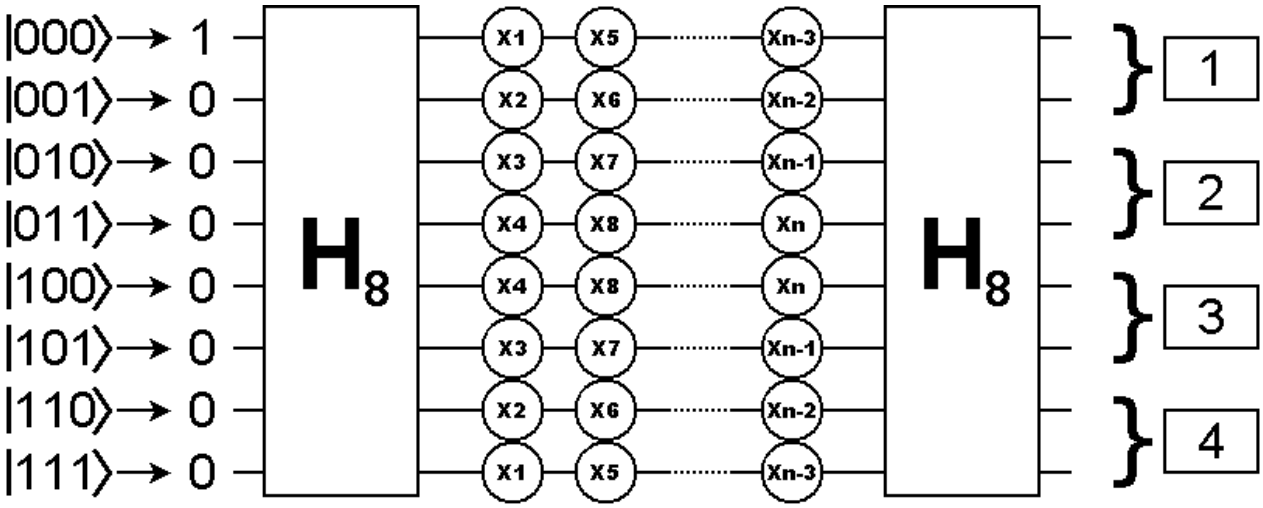
$$M_1^{Gen2/4}(X) = M_1(X') = M_1(X'^{I}) = M_1(X'^{II}) \tag{5.21}$$

III. Ja X' ir pusvienāds, bet nav simetrisks, tad

$$M_1^{Gen2/4}(X) = M_1(X') = res_1 \bigcup res_2, \text{ kur} \tag{5.22}$$
$$res_1 = \{i | i \in M_1(X'^{I}) \,\&\, i = 1\}$$
$$res_2 = \{i + N/4 | i \in M_1(X'^{I}) \,\&\, i > 1\}$$

IV. Ja X' ir puspretējs, tad

$$M_1^{Gen2/4}(X) = M_1(X') = res_1 \bigcup res_2, \text{ kur} \tag{5.23}$$
$$res_1 = \{i | i \in M_1(X'^{I}) \,\&\, i > 1\}$$
$$res_2 = \{i + 3N/4 | i \in M_1(X'^{I}) \,\&\, i = 1\}$$



5.6.att. **Kvantu algoritms, kas rēķina multifunkciju $M_1^{Gen2/4}$ ar N/4 vaicājumiem**

Teorēma 9. $Q_{RD}(M_1^{Gen2/N}) = N/|M_R|$.

Pierādījums. Katrā vaicājumā tiek jautāti $|M_R|$ jautājumi, kopā jābūt $N/|M_R|$vaicājumu.

Teorēma 10. $C_{RD}(M_1^{Gen2/N}) \geq (|M_R|/2 + 1) * N/|M_R|$.

Pierādījums. Pierādījums ir līdzīgs tam, kas tika izmantots multifunkcijas M_2 no [15] pirmā vispārinājuma gadījumā. Jāzina visas vērtības, kuras atrodas uz vismaz $|M_R|/2 + 1$ līnijām, t.i. $|M_R|/2 + 1$ reizināts ar vaicājumu skaitu. Pieņemsim, ka ir zināmas $(|M_R|/2 + 1) * N/|M_R| - 1$ vērtības, un tās visas ir vienādas ar 0. Ja visas pārējās nezināmas vērtības ir arī vienādas ar 0, rezultātā jābūt {1}. Bet, ja $|M_R|/2$ no nezināmiem mainīgiem ir vienādi ar 1, un katrs no šiem vieniniekiem atrodas uz savas līnijas, tad rezultātā jābūt kādai vērtībai, kas ir atšķirīga no „1". Tātad, no tā seko, ka jāzina vismaz „$(|M_R|/2 + 1) * N/|M_R|$" vērtības; t.i. piemēram,

$C_{RD}(M_1^{Gen2/2}) \geq N$ (sk. 5.5. att.) (5.24), $C_{RD}(M_1^{Gen2/4}) \geq 3N/4$ (sk.5.6.att.) (5.25), $C_{RD}(M_1^{Gen2/8}) \geq 5N/8$ (5.26).

$$C_{RD}(M_1^{Gen2/2}) \geq (2/2+1) * N/2 = N \quad (5.24)$$

$$C_{RD}(M_1^{Gen2/4}) \geq (4/2+1) * N/4 = 3N/4 \quad (5.25)$$

$$C_{RD}(M_1^{Gen2/8}) \geq (8/2+1) * N/8 = 5N/8 \quad (5.26)$$

5.2.Otrais piemērs - Algoritms multifunkcijas M_2 rēķināšanai

Tapāt, kā iepriekšējā piemērā, no sākuma ir aprakstīti multifunkcijas $M_2(X): \{0,1\}^N \rightarrow N, N = 2^i, i \in \mathbb{N}^+$ speciālgadījumi ar mazu mainīgo skaitu, un pēc tam tiek aprakstīts M_2 vispārinājums.

5.2.1. Multifunkcija M_2 ar četriem mainīgajiem

Algoritms, kas ir attēlots 5.7.att., rēķina sekojošo multifunkciju $M_2(X): \{0,1\}^4 \rightarrow \{1 \dots 4\}$ (5.27):

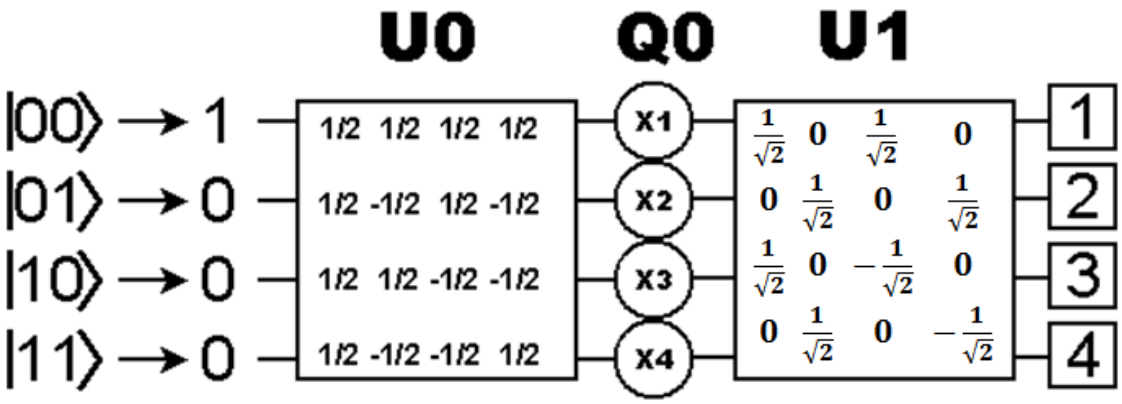
$$M_2(X) = \bigcup_{i=1}^{i=4} res_i, \text{ kur} \quad (5.27)$$

$$res_1 = \{1\}, \text{ja } x_1 = x_3$$

$$res_2 = \{3\}, \text{ja } x_1 \neq x_3$$

$$res_3 = \{2\}, \text{ja } x_2 = x_4$$

$$res_4 = \{4\}, \text{ja } x_4 \neq x_4$$



5.7.att. **Kvantu algoritms, kas rēķina multifunkciju M_2 ar četriem mainīgajiem**

Sīkāku multifunkcijas aprakstu, t.i. skaitļošanas soļus, var apskatīties 5.5.tabulā.

5.5.tabula

Apraksts multifunkcijai M_2 ar četriem mainīgajiem

X	Stāvoklis pēc U0	Stāvoklis pēc Q0	Stāvoklis pēc U1	$M_2(X)$
0000	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$(1,0,0,0)$	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)$	{1,2}
0001	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$\left(\frac{1}{\sqrt{2}},0,0,\frac{1}{\sqrt{2}}\right)$	{1,4}
0010	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$\left(0,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0\right)$	{2,3}
0011	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$(0,0,1,0)$	$\left(0,0,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$	{3,4}
0100	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$	$\left(\frac{1}{\sqrt{2}},0,0,-\frac{1}{\sqrt{2}}\right)$	{1,4}
0101	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$(0,1,0,0)$	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)$	{1,2}
0110	$\left(\frac{1}{2},-\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$	$(0,0,0,1)$	$\left(0,0,\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$	{3,4}
0111	$\left(\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$\left(-\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$\left(0,-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0\right)$	{2,3}
1000	$\left(-\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$\left(\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$\left(0,\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0\right)$	{2,3}
1001	$\left(-\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$(0,0,0,-1)$	$\left(0,0,-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$	{3,4}
1010	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$	$(0,-1,0,0)$	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)$	{3,4}
1011	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$\left(-\frac{1}{\sqrt{2}},0,0,\frac{1}{\sqrt{2}}\right)$	{1,4}
1100	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$	$(0,0,-1,0)$	$\left(0,0,-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$	{3,4}
1101	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)$	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$\left(0,-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0\right)$	{2,3}
1110	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$	$\left(-\frac{1}{\sqrt{2}},0,0,-\frac{1}{\sqrt{2}}\right)$	{1,4}
1111	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)$	$(-1,0,0,0)$	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)$	{1,2}

Kā redzams no 5.5. tabulas 4.kolonnas, multifunkcija M_2 tiek rēķināta vienmērīgi sadalītā veidā.

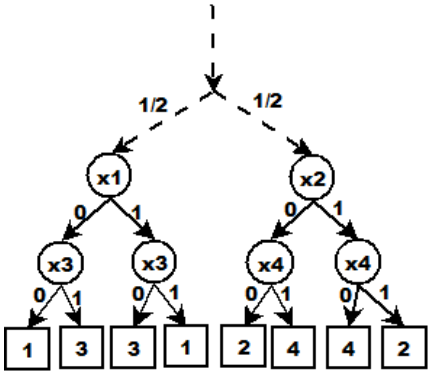
Teorēma 11. $Q_{UD}(M_2) = 1$ gadījumā, ja mainīgo skaits ir vienāds ar 4.

Pierādījums. Uzkonstruēto kvantu algoritmu ar vienu vaicājumu var redzēt 5.7.att.

Teorēma 12. $C_{UD}(M_1) = 2$ gadījumā, ja mainīgo skaits ir vienāds ar 4.

Pierādījums. Ir iespējams aplūkot lēmumu koku, kas ir uzzīmēts 5.8.att. Ir redzams, ka, zinot divu mainīgu vērtības, var precīzi pateikt vienu vērtību no rezultāta kopas.

Vienlaikus, no programmas izvades datiem ir redzams, ka sarežģītības apakšējā robeža ir 2, tātad, ar vienu vaicājumu nepietiek, lai droši pateiktu multifunkcijas vērtību.



5.8.att. **Lēmumu koks, kas rēķina multifunkciju M_2 ar četriem mainīgiem**

Programmas darbības laiks: apmēram 2 sekundes.

5.2.2. Multifunkcija M_2 ar astoņiem mainīgajiem

5.9.att. ir redzams algoritms, kas rēķina sekojošo multifunkciju $M_2(X)$: $\{0,1\}^8 \rightarrow \{1 \dots 8\}$ (5.28):

$$M_2(X) = \bigcup_{i=1}^{i=10} res_i, \text{ kur} \qquad (5.28)$$

$res_1 = \{1, 3, 5, 7\}$, t.i. kopa no visiem nepāra skaitļiem no 1 līdz N, ja X_{odd} satur nepāra vieninieku skaitu

$res_2 = \{2, 4, 6, 8\}$, t.i. kopa no visiem pāra skaitļiem no 1 līdz N, ja X_{even} satur nepāra vieninieku skaitu

$res_3 = \{1\}$, ja visi X_{odd} mainīgie ir vienādi (t.i. 0000 vai 1111)

$res_4 = \{2\}$, ja visi X_{even} mainīgie ir vienādi (t.i. 0000 vai 1111)

$res_5 = \{3\}$, ja X_{odd} ir pusvienāds un $res_1 = res_3 = \{\}$

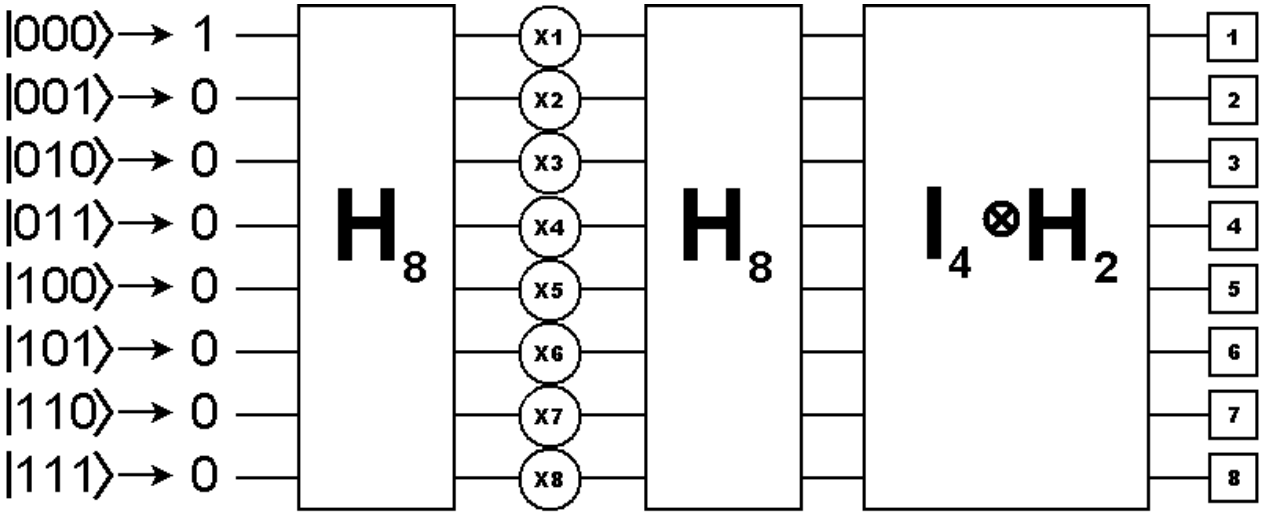
$res_6 = \{4\}$, ja X_{even} ir pusvienāds un $res_2 = res_4 = \{\}$

$res_7 = \{7\}$, ja X_{odd} ir simetrisks un $res_1 = res_3 = res_5 = \{\}$

$res_8 = \{8\}$, ja X_{even} ir simetrisks un $res_2 = res_4 = res_6 = \{\}$

$\mathrm{res}_9 = \{5\}$, ja X_{odd} ir puspretējs un $res_1 = res_3 = res_5 = res_7 = \{\}$

$\mathrm{res}_{10} = \{6\}$, ja X_{even} ir puspretējs un $res_2 = res_4 = res_6 = res_8 = \{\}$



5.9.att. **Kvantu algoritms, kas rēķina multifunkciju M_2 ar astoņiem mainīgajiem**

3.pielikumā atrodas sīkāks apraksts multifunkcijai M_2 ar astoņiem mainīgajiem.

Teorēma 13. $Q_{RD}(M_2) = 1$ gadījumā, ja mainīgo skaits ir vienāds ar 8.

Pierādījums. 5.9.att. var redzēt sekojošas kvantu sistēmas transformācijas. Algoritms sāka ar Adamāra matricu ar kārtu 8. Pēc tam seko viena vaicājuma transformācija. Un pēdējā transformācija ir Adamāra matrica ar kārtu N, kas ir reizināta ar vienības matricas ar kārtu 4 un Adamāra matricas ar kārtu 2 tenzorreizinājumu.

Teorēma 14. $C_{RD}(M_2) \geq 3$ gadījumā, ja mainīgo skaits ir vienāds ar 8.

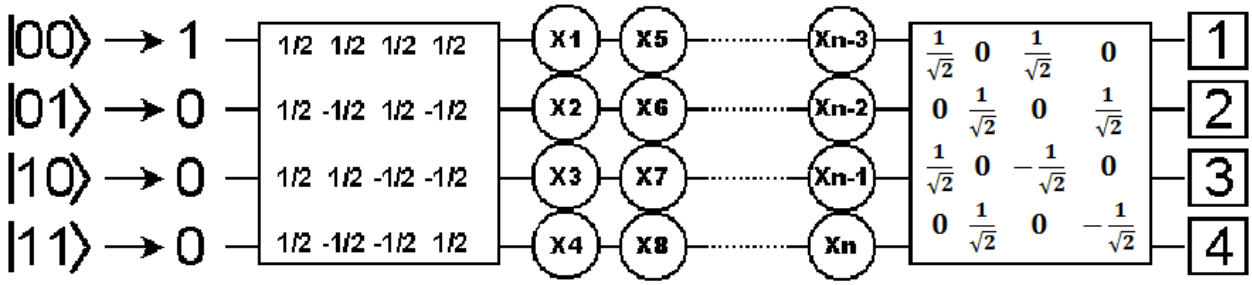
Pierādījums. Acīmredzami, ka var jautāt vai nu tikai pāra, vai tikai nepāra mainīgos, lai precīzi noteiktu vismaz vienu vērtību no rezultāta kopas ar minimālo vaicājumu skaitu. Pēc tam pierādījums ir līdzīgs pierādījumam Teorēmai 4. No četriem mainīgajiem, kuri sastāda $X_{odd}(X_{even})$ vajag uzzināt vismaz trīs mainīgo vērtības. Programma arī aprēķināja, ka klasiskās sarežģītības apakšējā robeža ir vienāda ar 3.

Programmas darbības laiks: apmēram 36 sekundes.

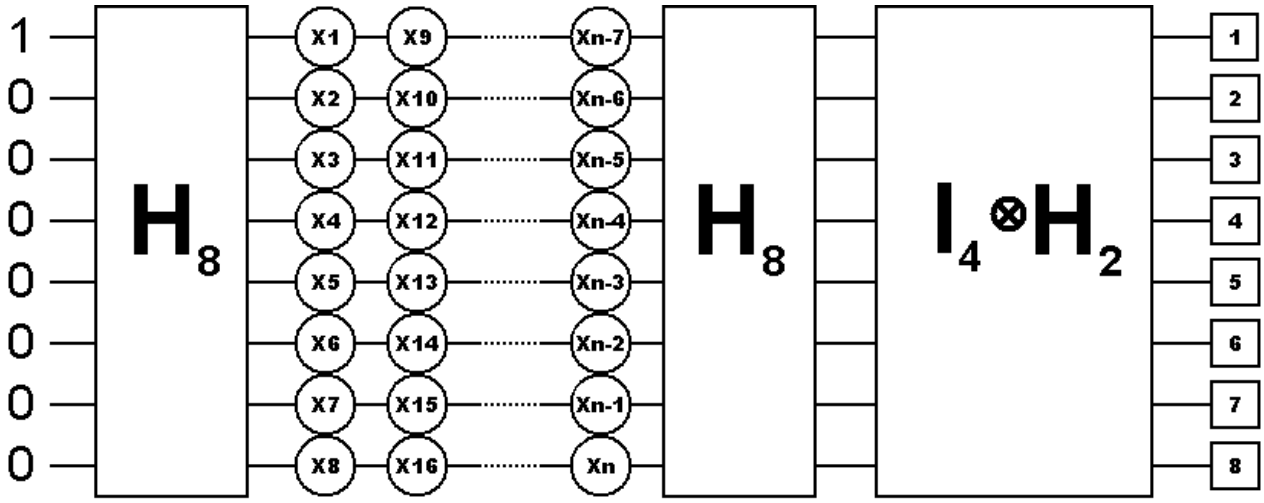
5.2.3. Multifunkcijas M_2 vispārinājums

Šeit ir izmantota tā pati metode, kura tika izmantota 5.1.5. punktā. 5.10. att.un 5.11.att. tiek parādīti divi algoritmi. Multifunkcijas, kuras rēķina šie algoritmi, ir apzīmēti attiecīgi $M_2^{Gen/4}$ un $M_2^{Gen/8}$.

Algoritms, kas ir parādīts 5.10. att., ir 5.2.1.punktā aprakstīta algoritma paplašinājums; tātad, lai dabūtu multifunkcijas $M_2{}^{Gen/4}(X)\colon\{0,1\}^N \rightarrow \{1 \dots 4\}$ definīciju, vajag 5.2.1.punktā aprakstītas multifunkcijas definīcijā aizvietot mainīgo x_i ar XOR(x_i, x_{i+4},x_{i+8},...). Ja to pašu izdarīt multifunkcijas definīcijā no 5.2.2.punkta, tad iegūsim $M_2{}^{Gen/8}(X)\colon\{0,1\}^N \rightarrow \{1 \dots 8\}$ definīciju.



5.10. att. **Kvantu algoritms, kas rēķina multifunkciju $M_2{}^{Gen/4}$ ar N/4 vaicājumiem**



5.11.att. **Kvantu algoritms, kas rēķina multifunkciju $M_2{}^{Gen/8}$ ar N/8 vaicājumiem**

Teorēma 15. $Q_{RD}(M_2{}^{Gen/N}) = N/|M_R|$.

Pierādījums. Attiecīgi $M_2{}^{Gen/4}$ kvantu sarežģītība ir vienāda ar N/4, bet $M_2{}^{Gen/8}$ - ar N/8, jo katrā vaicājumā tiek jautātas tik daudz vērtības, cik ir kvantu sistēmas bāzes stāvokļu. Katram bāzes stāvoklim atbilst viena vērtība no rezultātu kopas.

Teorēma 16. $C_{RD}(M_2{}^{Gen/N}) \geq (|M_R|/4 + 1) * N/|M_R|$.

Pierādījums. Pierādījums ir uztaisīts atsevišķi $M_2{}^{Gen/4}$ un $M_2{}^{Gen2/8}$ gadījumiem.

Acīmredzami, ka $M_2{}^{Gen2/4}$ gadījumā jāzina vismaz visas vērtības vai nu uz pāra līnijām, vai nu uz nepāra līnijām. Pieņemsim, ka ir zināmas N/2-1 vērtības, un tās visas ir vienādas ar „1”. Tā kā uz vienas līnijas atrodas N/4 vērtības, tad tikai uz vienas līnijas var zināt pilnīgi visas vērtības; uz trīm pārējām līnijām vismaz viena vērtība nav zināma. Ja nav zināmas visas vērtības no līnijas,

tad nevar izrēķināt XOR no šīs līnijas vērtībām. Bet ja mēs nezinām, vai XOR no vienās līnijas vērtībām ir vai nav vienāda ar XOR no citas līnijas vērtībām, mēs nevaram droši pateikt funkcijas vērtību. Tātad, $C_{RD}(M_2^{Gen/4}) \geq N/2$ (5.29).

Teorēma 14 saka, ka $C_{RD}(M_2) \geq 3$. $M_2^{Gen/8}$ ir M_2 paplašinājums. Tātad, jāzina pilnīgi visas vērtības vismaz uz trīm līnijām , lai aprēķinātu $M_2^{Gen/8}$ vērtību. Noteikti ar 3/8N-1 vaicājumiem nepietiks, lai uzjautātu pilnīgi visas vērtības uz trīm līnijām, jo katrā līnijā atrodas N/8 mainīgie. No tā seko, ka $C_{RD}(M_2^{Gen/8}) \geq 3N/8$ (5.30).

$$C_{RD}(M_2^{Gen/4}) \geq (4/4 + 1) * N/4 = N/2 \quad (5.29)$$

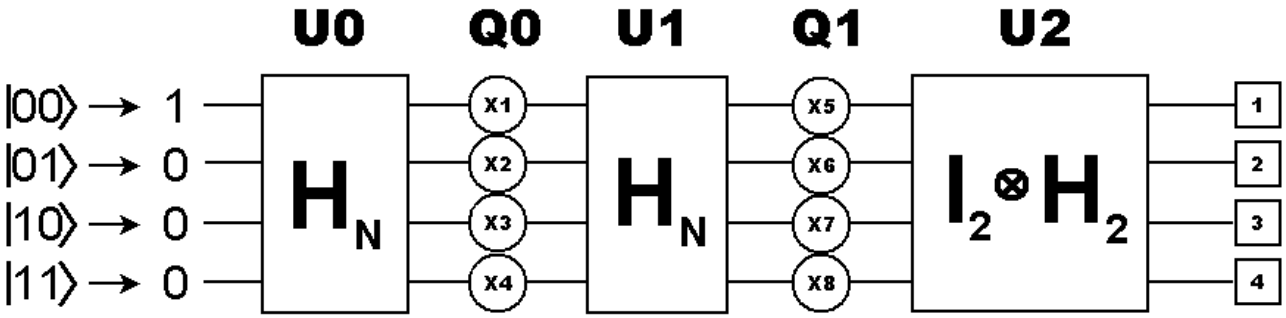
$$C_{RD}(M_2^{Gen/8}) \geq (8/4 + 1) * N/8 = 3N/8 \quad (5.30)$$

5.3. Trešais piemērs - Algoritms multifunkcijas M_3 rēķināšanai

Šajā nodaļā tiek aplūkots tikai viens speciālgadījums, un pēc tam seko vispārinājuma apraksts.

5.3.1. Multifunkcija M_3 ar astoņiem mainīgajiem

Trešais multifunkcijas piemērs tiek parādīts 5.12.att.



5.12.att. **Kvantu algoritms, kas rēķina multifunkciju M_3 ar astoņiem mainīgajiem**

Multifunkciju $M_3: \{0,1\}^8 \rightarrow \{1,2,3,4\}$ var aprakstīt sekojoši:

a) Ja X^I satur pāra vieninieku skaitu (un pāra nuļļu skaitu), tad (5.31):

$$M_3(X) = \bigcup_{i=1}^{i=4} res_i, \text{ kur} \quad (5.31)$$

$$res_1 = \{1\}, \text{ja } x_1 = x_3$$

$$res_2 = \{3\}, \text{ja } x_1 \neq x_3$$

$$res_3 = \{2\}, \text{ja } x_2 = x_4$$

$$res_4 = \{4\}, \text{ja } x_4 \neq x_4$$

b) Ja X^I satur nepāra vieninieku skaitu (un nepāra nuļļu skaitu), tad (5.32):

$$M_3(X) = \bigcup_{i=1}^{i=4} res_i, \text{ kur} \tag{5.32}$$

$$res_1 = \{1\}, \text{ja } XOR(x_1, x_3) = XOR(x_5, x_6)$$

$$res_2 = \{2\}, \text{ja } XOR(x_1, x_3) \neq XOR(x_5, x_6)$$

$$res_3 = \{3\}, \text{ja } XOR(x_2, x_4) = XOR(x_7, x_8)$$

$$res_4 = \{2\}, \text{ja } XOR(x_2, x_4) \neq XOR(x_7, x_8)$$

Sīkāku multifunkcijas M_3 aprakstu var apskatīties 5.pielikumā.

Teorēma 17. $Q_{UD}(M_3) = 2$.

Pierādījums. 5.12.att. ir redzams, ka starp trīm fiksētajām transformācijām ir divas vaicājuma transformācijas.

Teorēma 18. $C_{UD}(M_3) \geq 6$.

Pierādījums. Pieņemsim, ka algoritmam ir zināmas x_1 un x_3 vērtības un abas ir 0. Gadījumā, ja X^I satur pāra vieninieku skaitu, tad rezultātā jābūt {1}; bet ja X^I satur nepāra vieninieku skaitu, rezultāts ir atkarīgs no x_5 un x_6 vērtībām un var būt {1} vai {2}.

Ja algoritms, zinot tikai x_1 un x_3 vērtības, izvada rezultātu {1}, tad gadījumā, ja ievade ir, piemēram, 01001000, ir kļūda.

Ja algoritms jautā x_2 un x_4 vērtības un ievades X^I satur nepāra vieninieku skaitu, tad nav iespējams noteikt, vai rezultātā jābūt {1} vai {2}, neuzprasot x_5 un x_6 vērtības.

Pieņemsim, ka algoritms, zinot tikai x_1 un x_3 vērtības, nolemj vēl pajautāt tikai x_5 un x_6 vērtības, un x_5 ir vienāds ar 0 un x_6 ir vienāds ar 1. Ja algoritms izvada {2}, bet ievade ir, piemēram, 01011000, ir kļūda.

Acīmredzami, ka algoritmam jāzina vismaz x_1, x_2, x_3, x_4, un vai nu x_5 un x_6 vērtības, vai nu x_7 un x_8 vērtības.

Programma aprēķināja, ka apakšējā sarežģītības robeža ir 4. Bet tas liecina tikai par to, ka dažos gadījumos var izrēķināt multifunkciju, zinot tikai četros mainīgos (piemēram, ja x_1, x_2, x_3, x_4 ir vienādi ar 0). Bet nebija testēta multifunkcijas sarežģītība sliktākajā gadījumā.

Programmas darbības laiks: apmēram 1 minūte.

5.3.2. Multifunkcijas M_3 vispārinājums

Sk. 5.13.att. Šeit ir izmantots tāds pats princips, kāds tika lietots 5.1.5. un 5.2.3.punktos.

Multifunkciju ${M_3}^{Gen}: \{0,1\}^n \rightarrow \{1,2,3,4\}$ var aprakstīt sekojoši:

i. Ja X^I satur pāra vieninieku skaitu (un pāra nuļļu skaitu), tad (5.33):

$$M_3^{Gen}(X) = \bigcup_{i=1}^{i=4} res_i, \text{ kur} \tag{5.33}$$

$$res_1 = \{1\}, \text{ja } XOR(x_1, x_5, ..., x_{N/2-3}) = XOR(x_3, x_7, ..., x_{N/2-1})$$

$$res_2 = \{3\}, \text{ja } XOR(x_1, x_5, ..., x_{N/2-3}) \neq XOR(x_3, x_7, ..., x_{N/2-1})$$

$$res_3 = \{2\}, \text{ja } XOR(x_2, x_6, ..., x_{N/2-2}) = XOR(x_4, x_8, ..., x_{N/2})$$

$$res_4 = \{4\}, \text{ja } XOR(x_2, x_6, ..., x_{N/2-2}) \neq XOR(x_4, x_8, ..., x_{N/2})$$

ii. Ja X^I satur nepāra vienīnieku skaitu (un nepāra nuļļu skaitu), tad (5.34):

$$M_3^{Gen}(X) = \bigcup_{i=1}^{i=4} res_i, \text{ kur} \tag{5.34}$$

$$res_1 = \{1\}, \text{ja}$$

$$XOR(x_1, x_3, x_5, ..., x_{N/2-3}, x_{N/2-1}) = XOR(x_{N/2+1}, x_{N/2+2}, x_{N/2+5}, ..., x_{N-3}, x_{N-2})$$

$$res_2 = \{2\}, \text{ja}$$

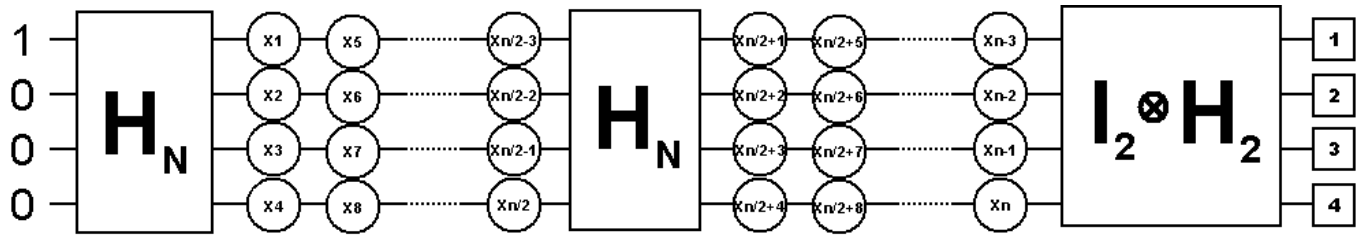
$$XOR(x_1, x_3, x_5, ..., x_{N/2-3}, x_{N/2-1}) \neq XOR(x_{N/2+1}, x_{N/2+2}, x_{N/2+5}, ..., x_{N-3}, x_{N-2})$$

$$res_3 = \{3\}, \text{ja}$$

$$XOR(x_2, x_4, x_6, ..., x_{N/2-2}, x_{N/2}) = XOR(x_{N/2+3}, x_{N/2+4}, x_{N/2+7}, ..., x_{N-1}, x_N)$$

$$res_4 = \{4\}, \text{ja}$$

$$XOR(x_2, x_4, x_6, ..., x_{N/2-2}, x_{N/2}) \neq XOR(x_{N/2+3}, x_{N/2+4}, x_{N/2+7}, ..., x_{N-1}, x_N)$$



5.13.att. **Kvantu algoritms, kas rēķina multifunkciju M_3^{Gen}**

Teorēma 19. $Q_{UD}(M_3^{Gen}) = N/4$.

Pierādījums. 5.13.att. ir parādīts, ka vienā vaicājumā ir četri secīgi mainīgie.

Teorēma 20. $C_{UD}(M_3^{Gen}) \geq 3N/4$.

Pierādījums. Pirmkārt, algoritmam jāzina visas X^I mainīgo vērtības, lai zinātu, vai vajag pielietot nosacījumu i vai ii. Acīmredzot, ka nevar noskaidrot, vai X^I satur pāra vai nepāra vienīnieku skaitu, ja nav zināmas visas X^I vērtības. Un ja tiek lietots Nosacījums ii, vajag noskaidrot pusi no atlikušajiem mainīgiem, t.i. kādi mainīgie atrodas vismaz uz divām līnijām, jo citādi nevar aprēķināt XOR no šīm mainīgiem.

NOBEIGUMS UN SECINĀJUMI

Galvenais darba mērķis ir sasniegts, tika uzkonstruēti vairāki kvantu vaicājošie algoritmi multifunkciju rēķināšanai bez kļūdas varbūtības, kuri ir pārāki par to klasiskajiem analogiem. Ir iegūti sekojoši rezultāti:

- Lai aprēķinātu multifunkciju M_1, kvantu algoritmam pietiek ar 1 vaicājumu, bet klasiskajam algoritmam ir nepieciešams vismaz N/2 + 1 vaicājumu.
  Gadījumā, ja mainīgo skaits nepārsniedz 4, tad multifunkcija tiek rēķināta vienmērīgi sadalītā veidā, un ja mainīgo skaits ir lielāks – tad nejauši sadalītā veidā.
  Ja mainīgo skaits pārsniedz 8, tad multifunkcija ir definēta tikai daļēji.
- $M_1^{Gen2/N}$ kvantu algoritma sarežģītība ir $N/|M_R|$, bet klasiskā algoritma sarežģītība ir lielāka vai vienāda ar $(|M_R|/2 + 1) * N/|M_R|$. Ja $|M_R|$ nepārsniedz 4, tad multifunkcija tiek rēķināta vienmērīgi sadalītā veidā, citos gadījumos tā tiek rēķināta nejauši sadalītā veidā.
- Lai aprēķinātu $M_2^{Gen/N}$, kvantu algoritms izdarīs $N/|M_R|$ vaicājumu, bet klasiskais algoritms – vismaz $(|M_R| /4 + 1) * N/|M_R|$ vaicājumu. $M_2^{Gen/4}$ tiek rēķināta vienmērīgi sadalītā veidā, $M_2^{Gen/8}$ - nejauši sadalītā veidā.
- Pēdējā piemērā kvantu algoritms izrēķinās M_3^{Gen} vienmērīgi sadalītā veidā ar N/4 vaicājumiem, bet klasiskajam algoritmam ir nepieciešams vismaz trīs reizēs vairāk jautājumu: $C_{UD}(M_3^{Gen}) \geq 3N/4$.

Par visnozīmīgāko no iegūtajiem algoritmiem autore uzskata pēdējo, kurš rēķina M_3^{Gen} multifunkciju, jo šajā gadījumā tiek sasniegts vislielākais no darbā iegūtajiem attālumiem starp klasisko un kvantu vaicājumu sarežģītību un multifunkcija tiek rēķināta vienmērīgi sadalītā veidā. Tādu pašu attālumu ieguva A.Vasilieva savos pētījumos par multifunkcijām [15]. To var uzskatīt par vislielāko attālumu starp kvantu vaicājošo un klasisko vaicājošo algoritmu sarežģītību multifunkciju skaitļošanai, kas ir sasniegta uz doto brīdi.

Par savu galveno sasniegumu autore uzskata programmu jaunu kvantu algoritmu ģenerēšanai. Arī programma klasisko vaicājošo algoritmu sarežģītības aprēķināšanai ir īpašas intereses cienīga.

Darba gaitā autore ir apguvusi Python programmēšanas valodu un uz tās balstīto Sage programmatūru. Pēc autores viedokļa Sage ir ļoti spēcīgs rīks matemātiskiem aprēķiniem, un tai pašā laikā šī programmatūra ir ļoti viegli apgūstama un lietojama.

Turpmākais darbs ir uzlabot programmu klasisko vaicājošo algoritmu sarežģītības novērtēšanai, piemēram, uzrakstīt metodes sliktākā gadījuma meklēšanai. Šādu uzlabojumu gala mērķis būtu programma, kas precīzi izskaitļotu klasisko vaicājošo algoritmu sarežģītību.

Kā viens no veidiem pilnveidot jaunu kvantu algoritmu ģenerēšanas programmu ir pievienot funkcionalitāti, kas palīdzētu atrast sakarības starp binārām virknēm un multifunkcijas vērtību kopām, t.i. palīdzētu noteikt multifunkcijas definīciju.

Protams, joprojām paliek aktuāls uzdevums meklēt jaunus efektīvus kvantu algoritmus. Šim nolūkam var turpināt izmantot tai skaitā arī šajā darbā uzrakstītās programmas pašreizējo versiju.

IZMANTOTĀ LITERATŪRA

[1] L. DiCarlo, J.M. Chow, J.M. Gambetta, L.S. Bishop, B.R. Johnson, D.I.Schuster, J. Majer, A. Blais, L. Frunzio, S.M. Girvin, R.J. Schoelkopf, "Demonstration of Two-Qubit Algorithms with a Superconducting Quantum Processor," *Nature*, vol. 460, pp. 240-244, 2009.

[2] D. Golze, M. Icker, S. Berger, "Implementation of two-qubit and three-qubit quantum computers using liquid-state nuclear magnetic resonance," *Concepts in magnetic resonance. Part A, Bridging education and research*, vol. 40A, no. 1, pp. 25-37, 2012.

[3] L.K. Grover, "A fast quantum mechanical algorithm for database search, in *28th Annual ACM Symposium on the Theory of Computing*, 1996, lpp. 212-219. Pieejams:. http://arxiv.org/abs/quant-ph/9605043

[4] P.W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *35th Annual Symposium on the Foundations of Computer Science*, 1994, pp. 124-134.

[5] H. Buhrman and R. de Wolf, "Complexity Measures and Decision Tree Complexity: a survey," in *Theoretical Computer Science 288*, 2002, pp. 21-43.

[6] A. Ambainis, "Superlinear advantage for exact quantum algorithms," in *STOC'2013*, 2012. Pieejams:. http://arxiv.org/abs/1211.0721v3

[7] J. Gruska, "Quantum Computing," *Wiley Encyclopedia of Computer Science and Engineering*, 2007.

[8] P. Kaye, R. Laflamme, and M. Mosca, *An introduction to quantum computing*, Oxford University Press, Ed., 2007.

[9] M.A. Nielsen and I.L. Chuang, *Quantum computation and quantum information*, Cambridge University Press, Ed., 2010.

[10] T. Rowland. Unitary Matrix. [tiešsaiste]. – [atsauce 27.05.2013.]. Pieejams: http://mathworld.wolfram.com/UnitaryMatrix.html

[11] E.W. Weisstein. Hadamard Matrix. [tiešsaiste]. – [atsauce 27.05.2013.]. Pieejams: http://mathworld.wolfram.com/HadamardMatrix.html

[12] Weisstein, E.W. Multivalued Function. [tiešsaiste]. – [atsauce 27.05.2013.]. Pieejams:

http://mathworld.wolfram.com/MultivaluedFunction.html

[13] A. Vasilieva, "Quantum Query Algorithms for Relations.," in *MFCS & CSL 2010 Satellite Workshop Randomized and quantum computation*, 2010, pp. 78-89.

[14] The Sage Development Team. (2005) Sage Computer Algebra System. [tiešsaiste]. – [atsauce 27.05.2013.]. Pieejams: http://www.sagemath.org/

[15] A. Vasiljeva, Complexity of quantum algorithms and communication protocols, Doctoral Thesis, University of Latvia, 2012.

[16] A. Vasilieva, "Uniformly Distributed Quantum Query Algorithms for Multifunctions.," in *Proc. of the Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, 2011, pp. 86-93.

PIELIKUMI

1. pielikums

Programmas pirmkods multifunkcijas M_1 aprēķinam

```
#converts decimal number to binary number of length N
def convert_to_binary(number, N):
   binary = []
   for i in range(N):
      if number >= (2^((N-1)-i)):
         binary.append(1)
         number = number - (2^((N-1)-i))
      else:
         binary.append(0)
   return binary

#returns matrix of a quantum query v
def gen_query(v):
   Q0= Matrix(0,0)
   for i in range(len(v)):
      AA= Matrix([(-1)^v[i]])
      Q0 = block_diagonal_matrix(Q0, AA, subdivide=False)
   return Q0

#create and multiply few queries
def make_queries(variables, size):
   Qres = identity_matrix(size)
   for kk in range((len(variables)*2)/(size)):
          var_temp = []
          for i in range(size/2):
              var_temp.append(variables[kk*(size/2) +i])
          qTemp = Sequence(var_temp)
          qTemp.reverse()
          q0=var_temp+ qTemp
          Qtmp = gen_query(q0)
          Qres= Qres *  Qtmp
   return Qres

#replace X with XOR from row variables
def XOR_var(variables, size):
   xor_var = []
   for i in range (size/2):
      var_temp = 0
      for kk in range ((len(variables)*2)/size):
         var_temp = var_temp + variables[kk*size/2+i]
      xor_var.append(var_temp%2)
   return xor_var

#returns "inversed" string (e.g. 0010 -> 1101)
def inverse_var(variables):
```

```
            inv = []
            for i in range (len(variables)):
                if (variables[i]==1):
                    inv.append(0)
                else:
                    if (variables[i]==0):
                        inv.append(1)
            return inv

        #returns first half of a list
        def first(variables):
            return variables[0:(ceil((len(variables))/2))]

        #returns second half of a list
        def second(variables):
            return variables[(ceil((len(variables))/2)):len(variables)]

        #Bitwise OR
        def bitwiseOR_for_lists(list1, list2):
            resM = []
            for i in range(len(list1)):
                if (list1[i]==1 or list2[i]==1):
                    resM.append(1)
                else:
                    resM.append(0)
            return resM

        #makes oracle (first variant - choose if |Mr|=2; operates on binary strings; checks rules which work for any size input)
        def oracle1(variables, size, variables1, variables2):
            res1 = one_zero (calc_probability(variables1,int(size/2)))
            res2 = one_zero (calc_probability(variables2,int(size/2)))
            resM = bitwiseOR_for_lists(res1, res2)
            shift = len(resM)/2
            resM2 = []

            for i in range(len(resM)):
                resM2.append(resM[int( (i+shift) % ( len(resM) ) )])

           #01000010
            if (Word((variables)).is_palindrome()):
                zeroV = zero_vector(size/2)
                oracle = resM + zeroV.list()
            else:
                #01001011
                if (variables1==(inverse_var(variables2))):
                    oracle = list(zero_vector(len(resM)/2))+ second(resM) + list(zero_vector(len(resM)/2)) + first(resM)
                else:
                    #01000100
                    if (variables1==variables2):
                        oracle = first(resM) + list(zero_vector(len(resM)/2)) + second(resM) + list(zero_vector(len(resM)/2))
```

```
            else:
                #01001101
                if(variables1==(inverse_var(variables2)).reverse()):
                    zeroV = zero_vector(size/2)
                    oracle = list(zeroV) + resM2
                else:
                    if ((Word(variables).count(1)%2==1)):
                        oracle = resM + resM2
                    else:
                        oracle = [-1]
                        return oracle
        return numbering(reduce(oracle))

    #check if sublist is in the list
    def sublistExists(list, sublist):
        for i in range(len(list)-len(sublist)+1):
            if sublist == list[i:i+len(sublist)]:
                return True
        return False

    #rules for the second oracle
    #rule1 = If X is symmetrical, M(X) = M(XI ) = M(XII )
    def rule1(resM, size):
        return resM

    #rule2 = If X if half-equal but not symmetrical,M(X) = res1&res2, where
    #res1 = M(XI ) = M(XII ) for ri ≤ N/4
    #res2 = M(XI ) + N/4 = M(XII ) + N/4 for ri > N/4
    def rule2(resM, size):
        resM2 = []
        for i in range (len (resM)):
            if (resM[i]<=(size/8)):
                resM2.append(resM[i])
            else:
                resM2.append(resM[i]+(size/8))
        return resM2

    #rule3 = If X is half-inverted, M(X) = res1&res2, where
    #res1 = M(XI ) = M(XII ) for ri > N/4
    #res2 = M(XI )+N ∗ 3/4 = M(XII )+N ∗ 3/4 for ri ≤ N/4
    def rule3(resM, size):
        resM2 = []
        for i in range (len (resM)):
            if (resM[i]>(size/8)):
                resM2.append(resM[i])
            else:
                resM2.append(resM[i]+(size*3/8))
        return resM2

    #rule4 = If XI contains even number of 1's (and even number of 0's) and XII contains even number
of 1's (and even number of 0's) and the rules above cannot be applied to X, M(X) = (M(XI )&M(XII ))&
(((M(XI )&M(XII ))shift(2) + N/2)
    def rule4(res1, res2, size):
```

```
        part1 = res1 + res2
        part2 = []
        for i in range (len (part1)):
            if(part1[i]!=2):
                part2.append( ((part1[i]+2)%4)+(size/4) )
            else:
                part2.append(8)
            oracle = part1 + part2
        return oracle
```

rule5 = If XI contains one 1 (or one 0) and XII contains one 1 (or one 0) and the number of 1's (and 0's) is equal in XI and XII and the rules above cannot be applied to X, M(X) = res1&res2, where
#res1 = 1&(2 + S), where S is the difference in positions of 1's (or 0's) in XI and XII
res2 = N − res1 + 1 if we can meet four 0's(or 1's) in a row while cycling through X and not more than four 0's (or 1;s)(e.g. X = 00100001 or X = 01001000) or
#res2 = res1 + N/2 otherwise.

```
def rule5(variables, variables1, variables2, size, num):
    part1 = [1]
    s = abs(variables1.index((num+1)%2)-variables2.index((num+1)%2))
    part1.append(Integer(2+s))
    part2 = []
    cyclestr = list(variables1 + variables2 + variables1 + variables2)

    list1 = []    #[0,0,0,0] or [1,1,1,1]
    list2 = []    #[0,0,0,0,0] or [1,1,1,1,1]
    for i in range(4):
        list1.append(num)
        list2.append(num)
    list2.append(num)

    if sublistExists(cyclestr, list1) and (not(sublistExists(cyclestr, list2))):
          for i in range (len (part1)):
              part2.append((size/2)+1-part1[i])
              oracle = part1 + part2
    else:
     if (not(sublistExists(cyclestr, list1)) or sublistExists(cyclestr, list2)):
                for i in range (len (part1)):
                    part2.append(part1[i]+(size/4))
     else:
          print "error - rule5"
    oracle = part1 + part2
    return oracle
```

#If XI contains one 1's (or one 0's) and XII contains N-1 1's (or N-1 0's)
#rule6 = If XI contains odd number of 1's (and odd number of 0's) and XII contains odd number of 1's (and odd number of 0's) and the number of 1's (and 0's) is not equal in XI and XII and the rules above cannot be applied to X, in order to calculate M(X) at first we must get the following set:
#XII in X should be replaced with the inverse of XII , then the result should be calculated using (Rule 5.) Let's denote this set Inv(M(x)). Then M(X) = {1...N}\Inv(M(x))

```
def rule6(variables, variables1, variables2, size):
    inverted_variables2 = inverse_var(variables2)
    if ((variables1).count(1) >(variables1).count(0)):
```

```
            num=1
        else:
            if ((variables1).count(0) >(variables1).count(1)):
                num =0
        inv_oracle = rule5(variables, variables1, inverted_variables2 , size, Integer(num))
        oracle = list(set(range(1, (size/2)+1)).difference(set(inv_oracle)))
        return oracle

    #rule7 = If XI is equal to reversed inverted XII, M(X) = M(XI)+4 = M(XII)+4 = N - M(XI) + 1
    def rule7(resM, size):
    #    print "Symmetrical inverse oracle: rule 7 -", # oracle
        oracle = []
        for i in range (len (resM)):
          oracle.append((size/2)+1-resM[i])
        return oracle

    #second variant of oracle
    def oracle2(variables, size, variables1, variables2):

    #if number of 1's is odd, M(X) = {1...N}
    #this rule should be moved after all other rules if it is applied for |Mr|=2; I put it here in order to save some time and not to check all 'if'-s for half of inputs

      if ((Word(variables).count(1)%2==1)):
          oracle = range (1,(size/2)+1)
          #print "Odd oracle", oracle
      else:
        res1 = list(calc_result(variables1,int(size/2)))
        res2 = list(calc_result(variables2,int(size/2)))
        resM = res1

        #01000010
        if (Word((variables)).is_palindrome()):
            oracle = rule1(resM,size)
        else:
            #01001011
            if (variables1==(inverse_var(variables2))):
                oracle=rule3(resM,size)
            else:
                #01000100
                if (variables1==variables2):
                    oracle = rule2(resM,size)
                else:
                    #01001101
                    reversed_variables2 = Sequence(variables2)
                    reversed_variables2.reverse();
                    if(variables1==(inverse_var(reversed_variables2))):
                        oracle = rule7(resM,size)
                    else:
                     #00001001
                       if(
                       (Word(variables1).count(1)%2==0) and
                       (Word(variables2).count(1)%2==0)):
```

```
                oracle = rule4(res1, res2, size)
            else:
              if(
              ((Word(variables1).count(1))==1) and ((Word(variables2).count(1))==1)):
                 oracle = rule5(variables, variables1, variables2, size, Integer(0))
              else:
                if(((Word(variables1).count(0))==1) and ((Word(variables2).count(0))==1)):
                  oracle = rule5(variables, variables1, variables2, size, Integer(1))
                else:
                  if(
                  (((Word(variables1).count(1))==1) and ((Word(variables2).count(0))==1)) or
                  (((Word(variables1).count(0))==1) and ((Word(variables2).count(1))==1))
                  ):
                      oracle = rule6(variables, variables1, variables2, size)
                  else:
                          oracle = [-1]
  return oracle

#calculate result
def calc_probability(variables,size):
   #The matrix used for fixed transformations
   U0 = hadamard_matrix(size)* 1/sqrt(size)

   #Initial vector
   w = zero_vector(size)
   w[0]=1

   Q0 = make_queries(variables,size)
   result = (w * U0 * Q0 * U0)
   return result

#replace all non-zero values with 1
def one_zero(str):
   res = Sequence([])
   for i in range (len(str)):
      if (str[i]==0):
        res.append(0)
      else:
        res.append(1)
   return res

#reduce 2N output into N (01001011->1011)
def reduce(str):
   res = Sequence([])
   it = 0
   for i in range (len(str)/2):
      if ((str[it]==0) and (str[it+1]==0)):
        res.append(0)
      else:
        res.append(1)

      it = it + 2
   return res
```

```
#replace 1 with position number(multifunction value)
def numbering(str):
   res = Sequence([])
   for i in range (len(str)):
       if (str[i]==1):
         res.append(int(i+1))
   return res

#calculate multifunction value - choose output format
def calc_result(variables,size):
   result = calc_probability(variables,size)
   #choose in which way output the result
#   return result
#   return reduce(one_zero (result))
   return (numbering(reduce(one_zero (result))))

#calculate lower bound
def lower_bound(N, dictM):
  for l in range(1,N):
    #all possible combinations of length l
    C = Combinations(range(N),l);
    a = C.list()

    for i in range(len(a)):
       # a combination from the combination list
       comb = a[i]
       comb_res = []

       # go through the dictionary with all multifunction inputs and find all inputs with 0's on "comb"
positions
       for k in range (len(dictM.keys())):
          key = dictM.keys()[k]
          f = true
          for j in range(len(comb)):
             if (key[comb[j]]!=0):
                f = false
          #if found a suitable input, append the multifunction result to the list comb_res
          if (f):
            comb_res.append(dictM[key])

       #check if all outputs contain one and the same value
       lower_bound = false
       for i in range (1,N+1):
          i_in_all = true
          for j in range (len(comb_res)):
             if not(i in (comb_res[j])):
                i_in_all = false
          if (i_in_all):
             lower_bound = l
             return l

#for large loops (N>16)
```

```
def irange(start, stop=None, step=1):
    if stop is None:
        stop = long(start)
        num = 1L
    else:
        stop = long(stop)
        num = long(start)
    step = long(step)
    while num < stop:
        yield num
        num += step

#Main function
N = 8    #number of variables
Size=16   #number of basis states - must be <=N*2

count = 0  #counts number of checked inputs
dictM = {} #dictionary for input/result values

for ss in irange(0,2^N):
    variables = convert_to_binary(ss, N)
    xor_var = XOR_var(variables, Size)
    xor_var1 = XOR_var(first(variables), Size)
    xor_var2 = XOR_var(second(variables), Size)

    result = (calc_result(variables,Size))
    #choose which oracle variant to use
    #oracle2 - only for |Mr|>2
    #choose oracle1 if |Mr|= 2
    oracle = oracle2(xor_var, Size,first(xor_var),second(xor_var))

    if (Set(result) ==Set(oracle)):
        rule = true
    else:
        if (oracle==[-1]):
            rule = "Not defined"
        else:
            rule=false
    if (true):
            #uncomment for a certain rule
            #if (rule == false):
            #if (rule != true):
                count = count + 1
                print Word(variables), "\t",  Set(result), "\t",  rule, "\t", "oracle: ", oracle

                #update dictionary
                dictM.update({tuple(variables):result})

print
print "count ", count

#calculate and print lower bound
print "lower bound: ", lower_bound(N, dictM)
```

Programmas pirmkods multifunkcijas M_2 aprēķinam

```
#converts decimal number to binary number of length N
def convert_to_binary(number, N):
   binary = []
   for i in range(N):
      if number >= (2^((N-1)-i)):
         binary.append(1)
         number = number - (2^((N-1)-i))
      else:
         binary.append(0)
   return binary

#returns matrix of a quantum query v
def gen_query(v):
   Q0= Matrix(0,0)
   for i in range(len(v)):
      AA= Matrix([(-1)^v[i]])
      Q0 = block_diagonal_matrix(Q0, AA, subdivide=False)
   return Q0

#create and multiply few queries
def make_queries(variables, size):
   Qres = identity_matrix(size)
   for kk in range((len(variables))/size):
         var_temp = []
         for i in range(size):
            var_temp.append(variables[kk*size +i])
         Qtmp = gen_query(var_temp)
         Qres= Qres *  Qtmp
   return Qres

#generates matrix with Hadamard matrices of size 2 on the diagonal and 0 on other positions
def gen_diag_H(size):
   Hadamard = hadamard_matrix(2)* 1/sqrt(2)
   U1 = identity_matrix(int(size/2)).tensor_product(Hadamard,subdivide=False)
   return U1

#make oracle for N=4
def oracleN4(variables):
   oracle = []
   #(a)     res1 = {1} if x1 = x3
   if (variables[0]==variables[2]):
      oracle.append(1);

   #(c)     res3 = {2} if x2 = x4
   if (variables[1]==variables[3]):
      oracle.append(2);

   #(b)     res2 = {3} if x1 != x3
   if (variables[0]!=variables[2]):
```

```
            oracle.append(3);

          #(d)     res4 = {4} if x2 != x4
          if (variables[1]!=variables[3]):
             oracle.append(4);
          return oracle

        #make oracle for N=8
        def oracleN8(variables, even, odd, size):
          oracle = []
          if (true):
        #(a) res1 is equal to the set of all odd numbers from 1 to N (i.e. {1,3,5,7} if N = 8) if Xodd contains
odd number of 1's
             if (number_of_one(odd)%2==1):
                i=1
                for n in range (len(variables)/2):
                   oracle.append(i)
                   i=i+2
                print "res1, ",
        #(c) res3 = {1} if variables in Xodd are equal
             else:
                if is_all_equal(odd):
                   oracle.append(1)
                   print "res3, ",
                else:
        #(e) res5 = {3} if variables in Xodd are half-equal
                   if is_half_equal(odd):
                      oracle.append(3)
                      print "res5, ",
                   else:
        # (g) res7 = {7} if Xodd is symmetrical
                      if is_symmetrical(odd):
                         oracle.append(7)
                         print "res7, ",
                      else:
        # (i) res9 = {5} if Xodd is half-inverted
                         if is_half_inverse(odd):
                            oracle.append(5)
                            print "res9, ",
                         else:
                            print "ERROR. Rule for odd not found!",
         #(b) res2 is equal to the set of all even numbers from 1 to N (i.e. {2,4,6,8} if N = 8) if Xeven
contains odd  number of 1's
             if (number_of_one(even)%2==1):
                i=2
                for n in range (len(variables)/2):
                   oracle.append(i)
                   i=i+2
                print "res2",
             else:
         #(d) res4 = {2} if variables in Xeven are equal
                if is_all_equal(even):
                  oracle.append(2)
```

```
            print "res4",
        else:
# (f) res6 = {4} if Xeven is half-equal
            if is_half_equal(even):
                oracle.append(4)
                print "res6",
            else:
# (h) res8 = {8} if Xeven is symmetrical
                if is_symmetrical(even):
                    oracle.append(8)
                    print "res8",
                else:
# (j) res10 = {6} if Xeven is half-inverted
                    if is_half_inverse(even):
                        oracle.append(6)
                        print "res10",
                    else:
                        print "ERROR. Rule for even not found!",
    print "\t",
  oracle = sorted(list(Set(oracle)))
  return sorted(oracle)

#replace X with XOR from row variables
def XOR_var(variables, size):
    xor_var = []
    for i in range (size):
        var_temp = 0
        for kk in range ((len(variables))/size):
            var_temp = var_temp + variables[kk*size+i]
        xor_var.append(var_temp%2)
    return xor_var

#count number of 1's in a list
def number_of_one(variables):
    count =0
    for i in range (len(variables)):
        if (variables[i]==1):
            count = count + 1
    return count

#return "inversed" string
def inverse_var(variables):
    inv = []
    for i in range (len(variables)):
        if (variables[i]==1):
            inv.append(0)
        else:
            if (variables[i]==0):
                inv.append(1)
    return inv

#return "symmetrical" string
def sym_var(variables):
```

```
    sim = []
    for i in range (len(variables)):
            sim.insert(0, variables[i])
    return sim

#return first half of a string
def first(variables):
    return variables[0:(ceil((len(variables))/2))]

#return second half of a string
def second(variables):
    return variables[(ceil((len(variables))/2)):len(variables)]

#return true if a string is "symmetrical"
def is_symmetrical(variables):
    if (first(variables)==sym_var(second(variables))):
        return true
    else:
         return false

#return true if the first half of a string equals the second half
def is_half_equal(variables):
    if (first(variables)==second(variables)):
        return true
    else:
         return false

#return true if the first half of a string equals the "inversed" second half
def is_half_inverse(variables):
    if (first(variables)==inverse_var(second(variables))):
        return true
    else:
         return false

#return true if all variables are equal
def is_all_equal(variables):
    for i in range (len(variables)-1):
        if (variables[0] != variables[i+1]):
            return false
    return true

#change all non-zero values with 1
def one_zero(str):
    res = Sequence([])
    for i in range (len(str)):
         if (str[i]==0):
            res.append(0)
         else:
            res.append(1)
    return res

#change 1 with position number(multifunction value)
def numbering(str):
```

```
    res = Sequence([])
    for i in range (len(str)):
        if (str[i]==1):
            res.append(int(i+1))
    return res

#return numbers from odd positions from a list
def odd_pos(variables):
    res = []
    i=0
    for n in range (len(variables)/2):
            res.append(variables[i])
            i=i+2
    return res

#return numbers from evenpositions from a list
def even_pos(variables):
    res = []
    i=1
    for n in range (len(variables)/2):
            res.append(variables[i])
            i=i+2
    return res

#calculate result
def calc_probability(variables,size):

    #List of matrices
    U0 = hadamard_matrix(size)* 1/sqrt(size)
    U1 = gen_diag_H(size)

    #Initial vector (100...00)
    w= vector(QQ, size, {0:1})

    Q6 = make_queries(variables, size)
    result = (w * U0 * Q6 * U0 * U1)
#    print (w * U0 * Q6), "\t", result, "\t",
    return result

#calculate multifunction value - choose in which format to output
def calc_result(variables,size):
    result = calc_probability(variables,size)
#    return (one_zero (result))
    return numbering((one_zero (result)))

#calculate lower bound
def lower_bound(N, dictM):
  for l in range(1,N):
    #all possible combinations of length l
    C = Combinations(range(N),l);
    a = C.list()

    for i in range(len(a)):
```

```
            # a combination from the combination list
            comb = a[i]
            comb_res = []

            # go through the dictionary with all multifunction inputs and find all inputs with 0's on "comb"
positions
            for k in range (len(dictM.keys())):
               key = dictM.keys()[k]
               f = true
               for j in range(len(comb)):
                  if (key[comb[j]]!=0):
                     f = false
               #if found a suitable input, append the multifunction result to the list comb_res
               if (f):
                 comb_res.append(dictM[key])

            #check if all outputs contain one and the same value
            lower_bound = false
            for i in range (1,N+1):
               i_in_all = true
               for j in range (len(comb_res)):
                  if not(i in (comb_res[j])):
                     i_in_all = false
               if (i_in_all):
                  lower_bound = l
                  return l

         #for large loops (N>16)
         def irange(start, stop=None, step=1):
            if stop is None:
               stop = long(start)
               num = 1L
            else:
               stop = long(stop)
               num = long(start)
            step = long(step)
            while num < stop:
               yield num
               num += step

         #Main function
         N=8       #number of variables
         Size=8    #number of basis states in a quantum system

         count = 0  #counts number of checked inputs
         dictM = {} #dictionary for input/result values

         for ss in irange(0,2^N):
            variables = convert_to_binary(ss, N)
            print Word(variables), "\t",

            var_oracle = XOR_var(variables, Size)
            odd = odd_pos(var_oracle)
```

```
    even = even_pos(var_oracle)

    #choose an appropriate oracle
#    oracle = oracleN4(var_oracle)
    oracle = oracleN8(var_oracle, even, odd, Size)

    result = calc_result(variables,Size)

    if (result == oracle):
        rule = true
    else:
        rule=false

    if (true):
        #uncomment for a certain rule if necesary
        #if (rule == false):
        #if (rule != true):

            count = count + 1

            print " {", ', '.join([str(item) for item in result]),"} ", "\t",
            print "Rule: ", rule

            #update dictionary
            dictM.update({tuple(variables):result})

print "count ", count

#calculate and print lower bound
print "lower bound: ", lower_bound(N, dictM)
```

Multifunkcijas M_2 apraksts ar astoņiem mainīgajiem

X	Nosacījums	$M_2(X)$
00000000	res3, res4	{ 1, 2 }
00000001	res3, res2	{ 1, 2, 4, 6, 8 }
00000010	res1, res4	{ 1, 2, 3, 5, 7 }
00000011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00000100	res3, res2	{ 1, 2, 4, 6, 8 }
00000101	res3, res10	{ 1, 6 }
00000110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00000111	res1, res10	{ 1, 3, 5, 6, 7 }
00001000	res1, res4	{ 1, 2, 3, 5, 7 }
00001001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00001010	res9, res4	{ 2, 5 }
00001011	res9, res2	{ 2, 4, 5, 6, 8 }
00001100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00001101	res1, res10	{ 1, 3, 5, 6, 7 }
00001110	res9, res2	{ 2, 4, 5, 6, 8 }
00001111	res9, res10	{ 5, 6 }
00010000	res3, res2	{ 1, 2, 4, 6, 8 }
00010001	res3, res6	{ 1, 4 }
00010010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00010011	res1, res6	{ 1, 3, 4, 5, 7 }
00010100	res3, res8	{ 1, 8 }
00010101	res3, res2	{ 1, 2, 4, 6, 8 }
00010110	res1, res8	{ 1, 3, 5, 7, 8 }
00010111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00011000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00011001	res1, res6	{ 1, 3, 4, 5, 7 }
00011010	res9, res2	{ 2, 4, 5, 6, 8 }
00011011	res9, res6	{ 4, 5 }
00011100	res1, res8	{ 1, 3, 5, 7, 8 }
00011101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00011110	res9, res8	{ 5, 8 }
00011111	res9, res2	{ 2, 4, 5, 6, 8 }
00100000	res1, res4	{ 1, 2, 3, 5, 7 }
00100001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00100010	res5, res4	{ 2, 3 }
00100011	res5, res2	{ 2, 3, 4, 6, 8 }
00100100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00100101	res1, res10	{ 1, 3, 5, 6, 7 }
00100110	res5, res2	{ 2, 3, 4, 6, 8 }
00100111	res5, res10	{ 3, 6 }
00101000	res7, res4	{ 2, 7 }
00101001	res7, res2	{ 2, 4, 6, 7, 8 }

00101010	res1, res4	{ 1, 2, 3, 5, 7 }
00101011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00101100	res7, res2	{ 2, 4, 6, 7, 8 }
00101101	res7, res10	{ 6, 7 }
00101110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00101111	res1, res10	{ 1, 3, 5, 6, 7 }
00110000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00110001	res1, res6	{ 1, 3, 4, 5, 7 }
00110010	res5, res2	{ 2, 3, 4, 6, 8 }
00110011	res5, res6	{ 3, 4 }
00110100	res1, res8	{ 1, 3, 5, 7, 8 }
00110101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00110110	res5, res8	{ 3, 8 }
00110111	res5, res2	{ 2, 3, 4, 6, 8 }
00111000	res7, res2	{ 2, 4, 6, 7, 8 }
00111001	res7, res6	{ 4, 7 }
00111010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
00111011	res1, res6	{ 1, 3, 4, 5, 7 }
00111100	res7, res8	{ 7, 8 }
00111101	res7, res2	{ 2, 4, 6, 7, 8 }
00111110	res1, res8	{ 1, 3, 5, 7, 8 }
00111111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01000000	res3, res2	{ 1, 2, 4, 6, 8 }
01000001	res3, res8	{ 1, 8 }
01000010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01000011	res1, res8	{ 1, 3, 5, 7, 8 }
01000100	res3, res6	{ 1, 4 }
01000101	res3, res2	{ 1, 2, 4, 6, 8 }
01000110	res1, res6	{ 1, 3, 4, 5, 7 }
01000111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01001000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01001001	res1, res8	{ 1, 3, 5, 7, 8 }
01001010	res9, res2	{ 2, 4, 5, 6, 8 }
01001011	res9, res8	{ 5, 8 }
01001100	res1, res6	{ 1, 3, 4, 5, 7 }
01001101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01001110	res9, res6	{ 4, 5 }
01001111	res9, res2	{ 2, 4, 5, 6, 8 }
01010000	res3, res10	{ 1, 6 }
01010001	res3, res2	{ 1, 2, 4, 6, 8 }
01010010	res1, res10	{ 1, 3, 5, 6, 7 }
01010011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01010100	res3, res2	{ 1, 2, 4, 6, 8 }
01010101	res3, res4	{ 1, 2 }
01010110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01010111	res1, res4	{ 1, 2, 3, 5, 7 }

01011000	res1, res10	{ 1, 3, 5, 6, 7 }
01011001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01011010	res9, res10	{ 5, 6 }
01011011	res9, res2	{ 2, 4, 5, 6, 8 }
01011100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01011101	res1, res4	{ 1, 2, 3, 5, 7 }
01011110	res9, res2	{ 2, 4, 5, 6, 8 }
01011111	res9, res4	{ 2, 5 }
01100000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01100001	res1, res8	{ 1, 3, 5, 7, 8 }
01100010	res5, res2	{ 2, 3, 4, 6, 8 }
01100011	res5, res8	{ 3, 8 }
01100100	res1, res6	{ 1, 3, 4, 5, 7 }
01100101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01100110	res5, res6	{ 3, 4 }
01100111	res5, res2	{ 2, 3, 4, 6, 8 }
01101000	res7, res2	{ 2, 4, 6, 7, 8 }
01101001	res7, res8	{ 7, 8 }
01101010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01101011	res1, res8	{ 1, 3, 5, 7, 8 }
01101100	res7, res6	{ 4, 7 }
01101101	res7, res2	{ 2, 4, 6, 7, 8 }
01101110	res1, res6	{ 1, 3, 4, 5, 7 }
01101111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01110000	res1, res10	{ 1, 3, 5, 6, 7 }
01110001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01110010	res5, res10	{ 3, 6 }
01110011	res5, res2	{ 2, 3, 4, 6, 8 }
01110100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01110101	res1, res4	{ 1, 2, 3, 5, 7 }
01110110	res5, res2	{ 2, 3, 4, 6, 8 }
01110111	res5, res4	{ 2, 3 }
01111000	res7, res10	{ 6, 7 }
01111001	res7, res2	{ 2, 4, 6, 7, 8 }
01111010	res1, res10	{ 1, 3, 5, 6, 7 }
01111011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01111100	res7, res2	{ 2, 4, 6, 7, 8 }
01111101	res7, res4	{ 2, 7 }
01111110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
01111111	res1, res4	{ 1, 2, 3, 5, 7 }
10000000	res1, res4	{ 1, 2, 3, 5, 7 }
10000001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10000010	res7, res4	{ 2, 7 }
10000011	res7, res2	{ 2, 4, 6, 7, 8 }
10000100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10000101	res1, res10	{ 1, 3, 5, 6, 7 }

10000110	res7, res2	{ 2, 4, 6, 7, 8 }
10000111	res7, res10	{ 6, 7 }
10001000	res5, res4	{ 2, 3 }
10001001	res5, res2	{ 2, 3, 4, 6, 8 }
10001010	res1, res4	{ 1, 2, 3, 5, 7 }
10001011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10001100	res5, res2	{ 2, 3, 4, 6, 8 }
10001101	res5, res10	{ 3, 6 }
10001110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10001111	res1, res10	{ 1, 3, 5, 6, 7 }
10010000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10010001	res1, res6	{ 1, 3, 4, 5, 7 }
10010010	res7, res2	{ 2, 4, 6, 7, 8 }
10010011	res7, res6	{ 4, 7 }
10010100	res1, res8	{ 1, 3, 5, 7, 8 }
10010101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10010110	res7, res8	{ 7, 8 }
10010111	res7, res2	{ 2, 4, 6, 7, 8 }
10011000	res5, res2	{ 2, 3, 4, 6, 8 }
10011001	res5, res6	{ 3, 4 }
10011010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10011011	res1, res6	{ 1, 3, 4, 5, 7 }
10011100	res5, res8	{ 3, 8 }
10011101	res5, res2	{ 2, 3, 4, 6, 8 }
10011110	res1, res8	{ 1, 3, 5, 7, 8 }
10011111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10100000	res9, res4	{ 2, 5 }
10100001	res9, res2	{ 2, 4, 5, 6, 8 }
10100010	res1, res4	{ 1, 2, 3, 5, 7 }
10100011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10100100	res9, res2	{ 2, 4, 5, 6, 8 }
10100101	res9, res10	{ 5, 6 }
10100110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10100111	res1, res10	{ 1, 3, 5, 6, 7 }
10101000	res1, res4	{ 1, 2, 3, 5, 7 }
10101001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10101010	res3, res4	{ 1, 2 }
10101011	res3, res2	{ 1, 2, 4, 6, 8 }
10101100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10101101	res1, res10	{ 1, 3, 5, 6, 7 }
10101110	res3, res2	{ 1, 2, 4, 6, 8 }
10101111	res3, res10	{ 1, 6 }
10110000	res9, res2	{ 2, 4, 5, 6, 8 }
10110001	res9, res6	{ 4, 5 }
10110010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10110011	res1, res6	{ 1, 3, 4, 5, 7 }

10110100	res9, res8	{ 5, 8 }
10110101	res9, res2	{ 2, 4, 5, 6, 8 }
10110110	res1, res8	{ 1, 3, 5, 7, 8 }
10110111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10111000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10111001	res1, res6	{ 1, 3, 4, 5, 7 }
10111010	res3, res2	{ 1, 2, 4, 6, 8 }
10111011	res3, res6	{ 1, 4 }
10111100	res1, res8	{ 1, 3, 5, 7, 8 }
10111101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
10111110	res3, res8	{ 1, 8 }
10111111	res3, res2	{ 1, 2, 4, 6, 8 }
11000000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11000001	res1, res8	{ 1, 3, 5, 7, 8 }
11000010	res7, res2	{ 2, 4, 6, 7, 8 }
11000011	res7, res8	{ 7, 8 }
11000100	res1, res6	{ 1, 3, 4, 5, 7 }
11000101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11000110	res7, res6	{ 4, 7 }
11000111	res7, res2	{ 2, 4, 6, 7, 8 }
11001000	res5, res2	{ 2, 3, 4, 6, 8 }
11001001	res5, res8	{ 3, 8 }
11001010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11001011	res1, res8	{ 1, 3, 5, 7, 8 }
11001100	res5, res6	{ 3, 4 }
11001101	res5, res2	{ 2, 3, 4, 6, 8 }
11001110	res1, res6	{ 1, 3, 4, 5, 7 }
11001111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11010000	res1, res10	{ 1, 3, 5, 6, 7 }
11010001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11010010	res7, res10	{ 6, 7 }
11010011	res7, res2	{ 2, 4, 6, 7, 8 }
11010100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11010101	res1, res4	{ 1, 2, 3, 5, 7 }
11010110	res7, res2	{ 2, 4, 6, 7, 8 }
11010111	res7, res4	{ 2, 7 }
11011000	res5, res10	{ 3, 6 }
11011001	res5, res2	{ 2, 3, 4, 6, 8 }
11011010	res1, res10	{ 1, 3, 5, 6, 7 }
11011011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11011100	res5, res2	{ 2, 3, 4, 6, 8 }
11011101	res5, res4	{ 2, 3 }
11011110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11011111	res1, res4	{ 1, 2, 3, 5, 7 }
11100000	res9, res2	{ 2, 4, 5, 6, 8 }
11100001	res9, res8	{ 5, 8 }

11100010	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11100011	res1, res8	{ 1, 3, 5, 7, 8 }
11100100	res9, res6	{ 4, 5 }
11100101	res9, res2	{ 2, 4, 5, 6, 8 }
11100110	res1, res6	{ 1, 3, 4, 5, 7 }
11100111	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11101000	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11101001	res1, res8	{ 1, 3, 5, 7, 8 }
11101010	res3, res2	{ 1, 2, 4, 6, 8 }
11101011	res3, res8	{ 1, 8 }
11101100	res1, res6	{ 1, 3, 4, 5, 7 }
11101101	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11101110	res3, res6	{ 1, 4 }
11101111	res3, res2	{ 1, 2, 4, 6, 8 }
11110000	res9, res10	{ 5, 6 }
11110001	res9, res2	{ 2, 4, 5, 6, 8 }
11110010	res1, res10	{ 1, 3, 5, 6, 7 }
11110011	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11110100	res9, res2	{ 2, 4, 5, 6, 8 }
11110101	res9, res4	{ 2, 5 }
11110110	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11110111	res1, res4	{ 1, 2, 3, 5, 7 }
11111000	res1, res10	{ 1, 3, 5, 6, 7 }
11111001	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11111010	res3, res10	{ 1, 6 }
11111011	res3, res2	{ 1, 2, 4, 6, 8 }
11111100	res1, res2	{ 1, 2, 3, 4, 5, 6, 7, 8 }
11111101	res1, res4	{ 1, 2, 3, 5, 7 }
11111110	res3, res2	{ 1, 2, 4, 6, 8 }
11111111	res3, res4	{ 1, 2 }

Programmas pirmkods multifunkcijas M_3 aprēķinam

```
#converts decimal number to binary number of length N
def convert_to_binary(number, N):
    binary = []
    for i in range(N):
        if number >= (2^((N-1)-i)):
            binary.append(1)
            number = number - (2^((N-1)-i))
        else:
            binary.append(0)
    return binary

#returns matrix of a quantum query v
def gen_query(v):
    Q0= Matrix(0,0)
    for i in range(len(v)):
        AA= Matrix([(-1)^v[i]])
        Q0 = block_diagonal_matrix(Q0, AA, subdivide=False)
    return Q0

#create and multiply few queries
def make_queries(variables, size):
    Qres = identity_matrix(size)
    for kk in range((len(variables))/(size)):
            var_temp = []
            for i in range(size):
                var_temp.append(variables[kk*(size) +i])
            Qtmp = gen_query(var_temp)
            Qres= Qres *  Qtmp
    return Qres

#generates matrix with Hadamard matrices of size 2 on the diagonal and 0 on other positions
def gen_diag_H(size):
    Hadamard = hadamard_matrix(2)* 1/sqrt(2)
    U1 = identity_matrix(int(size/2)).tensor_product(Hadamard,subdivide=False)
    return U1

#make oracle
def oracle(variables1, variables2):
    oracle = []
    if (number_of_one(variables1)%2==0):
        if (variables1[0]==variables1[2]):
            oracle.append(1)
        else:
            oracle.append(3)
        if (variables1[1]==variables1[3]):
            oracle.append(2)
        else:
            oracle.append(4)
```

```
    else:
        if ((number_of_one(odd_pos(variables1))%2)==(number_of_one([variables2[0],variables2[1]])%2)):
            oracle.append(1)
        else:
            oracle.append(2)
        if
((number_of_one(even_pos(variables1))%2)==(number_of_one([variables2[2],variables2[3]])%2)):
            oracle.append(3)
        else:
            oracle.append(4)
#    print "oracle: ", oracle
    return oracle

#count number of 1's in a list
def number_of_one(variables):
    count =0
    for i in range (len(variables)):
        if (variables[i]==1):
            count = count + 1
    return count

#return first half of a string
def first(variables):
    return variables[0:(ceil((len(variables))/2))]

#return second half of a string
def second(variables):
    return variables[(ceil((len(variables))/2)):len(variables)]

#change all non-zero values with 1
def one_zero(str):
    res = Sequence([])
    for i in range (len(str)):
        if (str[i]==0):
            res.append(0)
        else:
            res.append(1)
    return res

#change 1 with position number(multifunction value)
def numbering(str):
    res = Sequence([])
    for i in range (len(str)):
        if (str[i]==1):
            res.append(int(i+1))
    return res

#return numbers from odd positions from a list
def odd_pos(variables):
    res = []
    i=0
    for n in range (len(variables)/2):
            res.append(variables[i])
```

```
        i=i+2
    return res

#return numbers from evenpositions from a list
def even_pos(variables):
    res = []
    i=1
    for n in range (len(variables)/2):
        res.append(variables[i])
        i=i+2
    return res

#calculate result
def calc_probability(variables,size):

    #List of matrices
    U0 = hadamard_matrix(size)* 1/sqrt(size)
    U1 = gen_diag_H(size)

    #Initial vector (100...0)
    w = zero_vector(QQ, size)
    w[0] = 1

    #generate query(-ies)
    Q1N = make_queries(first(variables), size)
    Q2N = make_queries(second(variables), size)

    #apply all transformations
    result = (w * U0 * Q1N * U0 * Q2N * U1)

# uncomment next two lines for intermediate amplitudes
#    print (w * U0 * Q1N), "\t", (w * U0 * Q1N * U0), "\t",(w * U0 * Q1N * U0 * Q2N), "\t",
#    print result,

    return result

#replace X with XOR from row variables
def XOR_var(variables, size):
    xor_var = []
    for i in range (size):
        var_temp = 0
        for kk in range ((len(variables))/size):
            var_temp = var_temp + variables[kk*size+i]
        xor_var.append(var_temp%2)
    return xor_var

#calculate multifunction value - choose in which format to output
def calc_result(variables,size):
    result = calc_probability(variables,size)
#    return result
    return numbering(one_zero (result))

#calculate lower bound
```

```
def lower_bound(N, dictM):
  for l in range(1,N):
    #all possible combinations of length l
    C = Combinations(range(N),l);
    a = C.list()

    for i in range(len(a)):
        # a combination from the combination list
        comb = a[i]
        comb_res = []

        # go through the dictionary with all multifunction inputs and find all inputs with 0's on "comb"
positions
        for k in range (len(dictM.keys())):
            key = dictM.keys()[k]
            f = true
            for j in range(len(comb)):
                if (key[comb[j]]!=0):
                    f = false
            #if found a suitable input, append the multifunction result to the list comb_res
            if (f):
              comb_res.append(dictM[key])

        #check if all outputs contain one and the same value
        lower_bound = false
        for i in range (1,N+1):
            i_in_all = true
            for j in range (len(comb_res)):
                if not(i in (comb_res[j])):
                    i_in_all = false
            if (i_in_all):
                lower_bound = l
                return l

#for large loops (N>16)
def irange(start, stop=None, step=1):
    if stop is None:
        stop = long(start)
        num = 1L
    else:
        stop = long(stop)
        num = long(start)
    step = long(step)
    while num < stop:
        yield num
        num += step

#Main function
N=8        #number of variables
Size=4     #number of basis states in a quantum system

count = 0  #counts number of checked inputs
dictM = {} #dictionary for input/result  values
```

```
for ss in irange(0, 2^N):
    variables = convert_to_binary(ss,N)
    xor_var1 = XOR_var(first(variables), Size)
    xor_var2 = XOR_var(second(variables), Size)

    print  Word(variables), "\t",

    result = calc_result(variables,Size)

    #check if multifunction definition is correct
    if (result == oracle(xor_var1, xor_var2)):
        rule = true
    else:
        rule=false
    if (true): #for indents

        #uncomment for a certain rule if necesary
        #if (rule == false):
        #if (rule != true):

            count = count + 1

            #print the result set
            print " {", ', '.join([str(item) for item in result]), "} ",
            print "\t",  "Rule: ", rule

            #update dictionary
            dictM[tuple(variables)] = result

print "count ", count

#calculate and print lower bound
print "lower bound: ", lower_bound(N, dictM)
```

Multifunkcijas M_3 apraksts ar astoņiem mainīgajiem

X	Stāvoklis pēc Q0	Stāvoklis pēc U1	Stāvoklis pēc Q1	Stāvoklis pēc U2	$M_3(X)$
00000000	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000001	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000010	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000011	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000100	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000101	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000110	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00000111	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001000	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001001	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001010	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001011	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001100	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001101	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001110	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00001111	(1/2, 1/2, 1/2, 1/2)	(1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
00010000	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
00010001	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
00010010	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
00010011	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
00010100	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
00010101	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
00010110	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
00010111	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
00011000	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
00011001	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
00011010	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
00011011	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
00011100	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
00011101	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
00011110	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
00011111	(1/2, 1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
00100000	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
00100001	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
00100010	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
00100011	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
00100100	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
00100101	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
00100110	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
00100111	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
00101000	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
00101001	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
00101010	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
00101011	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
00101100	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
00101101	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
00101110	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
00101111	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
00110000	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00110001	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00110010	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }

00110011	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00110100	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00110101	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00110110	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00110111	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00111000	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00111001	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00111010	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00111011	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00111100	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00111101	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
00111110	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
00111111	(1/2, 1/2, -1/2, -1/2)	(0, 0, 1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01000000	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
01000001	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
01000010	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
01000011	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
01000100	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
01000101	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
01000110	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
01000111	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
01001000	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
01001001	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
01001010	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
01001011	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
01001100	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
01001101	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
01001110	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
01001111	(1/2, -1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
01010000	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010001	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010010	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010011	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010100	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010101	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010110	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01010111	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011000	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011001	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011010	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011011	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011100	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011101	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011110	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01011111	(1/2, -1/2, 1/2, -1/2)	(0, 1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
01100000	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01100001	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01100010	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01100011	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01100100	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01100101	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01100110	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01100111	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01101000	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01101001	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }

01101010	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01101011	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01101100	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01101101	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01101110	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
01101111	(1/2, -1/2, -1/2, 1/2)	(0, 0, 0, 1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
01110000	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
01110001	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
01110010	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
01110011	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
01110100	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
01110101	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
01110110	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
01110111	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
01111000	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
01111001	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
01111010	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
01111011	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
01111100	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
01111101	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
01111110	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
01111111	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
10000000	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
10000001	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
10000010	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
10000011	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
10000100	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
10000101	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
10000110	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
10000111	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
10001000	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
10001001	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
10001010	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
10001011	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
10001100	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
10001101	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
10001110	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
10001111	(-1/2, 1/2, 1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
10010000	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10010001	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10010010	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10010011	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10010100	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10010101	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10010110	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10010111	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10011000	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10011001	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10011010	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10011011	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10011100	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10011101	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10011110	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, -1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
10011111	(-1/2, 1/2, 1/2, -1/2)	(0, 0, 0, -1)	(0, 0, 0, 1)	(0, 0, 1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
10100000	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }

10100001	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100010	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100011	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100100	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100101	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100110	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10100111	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101000	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101001	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101010	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101011	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, -1, 0, 0)	(-1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101100	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101101	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101110	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10101111	(-1/2, 1/2, -1/2, 1/2)	(0, -1, 0, 0)	(0, 1, 0, 0)	(1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
10110000	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
10110001	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
10110010	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
10110011	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
10110100	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
10110101	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
10110110	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
10110111	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
10111000	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
10111001	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
10111010	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
10111011	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
10111100	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
10111101	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
10111110	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
10111111	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
11000000	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11000001	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11000010	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11000011	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11000100	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11000101	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11000110	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11000111	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11001000	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11001001	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11001010	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11001011	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11001100	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11001101	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1/2*sqrt(2), -1/2*sqrt(2))	{ 3, 4 }
11001110	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11001111	(-1/2, -1/2, 1/2, 1/2)	(0, 0, -1, 0)	(0, 0, 1, 0)	(0, 0, 1/2*sqrt(2), 1/2*sqrt(2))	{ 3, 4 }
11010000	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
11010001	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
11010010	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
11010011	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
11010100	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
11010101	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
11010110	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
11010111	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }

11011000	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
11011001	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
11011010	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
11011011	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
11011100	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
11011101	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
11011110	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
11011111	(-1/2, -1/2, 1/2, -1/2)	(-1/2, 1/2, -1/2, -1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
11100000	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
11100001	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, -1/2)	(-1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
11100010	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, 1/2, 1/2)	(-1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
11100011	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, 1/2, -1/2)	(-1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
11100100	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, 1/2, -1/2, 1/2)	(0, -1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
11100101	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, 1/2, -1/2, -1/2)	(0, -1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
11100110	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, 1/2, 1/2, 1/2)	(0, -1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
11100111	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(-1/2, 1/2, 1/2, -1/2)	(0, -1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
11101000	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, -1/2, -1/2, 1/2)	(0, 1/2*sqrt(2), 0, -1/2*sqrt(2))	{ 2, 4 }
11101001	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, -1/2, -1/2, -1/2)	(0, 1/2*sqrt(2), -1/2*sqrt(2), 0)	{ 2, 3 }
11101010	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, 1/2)	(0, 1/2*sqrt(2), 1/2*sqrt(2), 0)	{ 2, 3 }
11101011	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, -1/2, 1/2, -1/2)	(0, 1/2*sqrt(2), 0, 1/2*sqrt(2))	{ 2, 4 }
11101100	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, 1/2, -1/2, 1/2)	(1/2*sqrt(2), 0, 0, -1/2*sqrt(2))	{ 1, 4 }
11101101	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, 1/2, -1/2, -1/2)	(1/2*sqrt(2), 0, -1/2*sqrt(2), 0)	{ 1, 3 }
11101110	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, 1/2, 1/2, 1/2)	(1/2*sqrt(2), 0, 1/2*sqrt(2), 0)	{ 1, 3 }
11101111	(-1/2, -1/2, -1/2, 1/2)	(-1/2, -1/2, -1/2, 1/2)	(1/2, 1/2, 1/2, -1/2)	(1/2*sqrt(2), 0, 0, 1/2*sqrt(2))	{ 1, 4 }
11110000	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110001	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110010	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110011	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110100	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110101	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110110	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11110111	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1/2*sqrt(2), -1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111000	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111001	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111010	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111011	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111100	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111101	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111110	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }
11111111	(-1/2, -1/2, -1/2, -1/2)	(-1, 0, 0, 0)	(1, 0, 0, 0)	(1/2*sqrt(2), 1/2*sqrt(2), 0, 0)	{ 1, 2 }

Programmas pirmkods klasiskās sarežģītības apakšējās robežas aprēķināšanai

N = 8

#function dictionary
M = {
(0, 0, 0, 0, 0, 0, 0, 0) : [1] , (0, 0, 0, 0, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 0, 0, 1, 1) : [1, 4, 6, 7] , (0, 0, 0, 0, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 0, 1, 0, 1) : [1, 3, 5, 7] , (0, 0, 0, 0, 0, 1, 1, 0) : [1, 2, 7, 8] , (0, 0, 0, 0, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 1, 0, 0, 1) : [1, 2, 7, 8] , (0, 0, 0, 0, 1, 0, 1, 0) : [1, 3, 5, 7] , (0, 0, 0, 0, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 1, 1, 0, 0) : [1, 4, 6, 7] , (0, 0, 0, 0, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 0, 1, 1, 1, 1) : [7] , (0, 0, 0, 1, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 0, 0, 0, 1) : [1, 2, 5, 6] , (0, 0, 0, 1, 0, 0, 1, 0) : [1, 3, 6, 8] , (0, 0, 0, 1, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 0, 1, 0, 0) : [1, 4, 5, 8] , (0, 0, 0, 1, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 0, 1, 1, 1) : [5, 6, 7, 8] , (0, 0, 0, 1, 1, 0, 0, 0) : [1, 2, 3, 4] , (0, 0, 0, 1, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 1, 0, 1, 1) : [2, 3, 6, 7] , (0, 0, 0, 1, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 0, 1, 1, 1, 0, 1) : [2, 4, 5, 7] , (0, 0, 0, 1, 1, 1, 1, 0) : [3, 4, 7, 8] , (0, 0, 0, 1, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 0, 0, 0, 1) : [1, 3, 6, 8] , (0, 0, 1, 0, 0, 0, 1, 0) : [1, 2, 5, 6] , (0, 0, 1, 0, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 0, 1, 0, 0) : [1, 2, 3, 4] , (0, 0, 1, 0, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 0, 1, 1, 1) : [2, 3, 6, 7] , (0, 0, 1, 0, 1, 0, 0, 0) : [1, 4, 5, 8] , (0, 0, 1, 0, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 1, 0, 1, 1) : [5, 6, 7, 8] , (0, 0, 1, 0, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 0, 1, 1, 0, 1) : [3, 4, 7, 8] , (0, 0, 1, 0, 1, 1, 1, 0) : [2, 4, 5, 7] , (0, 0, 1, 0, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 0, 0, 0, 0) : [1, 4, 6, 7] , (0, 0, 1, 1, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 0, 0, 1, 1) : [6] , (0, 0, 1, 1, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 0, 1, 0, 1) : [3, 4, 5, 6] , (0, 0, 1, 1, 0, 1, 1, 0) : [2, 4, 6, 8] , (0, 0, 1, 1, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 1, 0, 0, 1) : [2, 4, 6, 8] , (0, 0, 1, 1, 1, 0, 1, 0) : [3, 4, 5, 6] , (0, 0, 1, 1, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 1, 1, 0, 0) : [4] , (0, 0, 1, 1, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 0, 1, 1, 1, 1, 1, 1) : [1, 4, 6, 7] , (0, 1, 0, 0, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 0, 0, 0, 1) : [1, 4, 5, 8] , (0, 1, 0, 0, 0, 0, 1, 0) : [1, 2, 3, 4] , (0, 1, 0, 0, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 0, 1, 0, 0) : [1, 2, 5, 6] , (0, 1, 0, 0, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 0, 1, 1, 1) : [2, 4, 5, 7] , (0, 1, 0, 0, 1, 0, 0, 0) : [1, 3, 6, 8] , (0, 1, 0, 0, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 1, 0, 1, 1) : [3, 4, 7, 8] , (0, 1, 0, 0, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 0, 1, 1, 0, 1) : [5, 6, 7, 8] , (0, 1, 0, 0, 1, 1, 1, 0) : [2, 3, 6, 7] , (0, 1, 0, 0, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 0, 0, 0, 0) : [1, 3, 5, 7] , (0, 1, 0, 1, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 0, 0, 1, 1) : [3, 4, 5, 6] , (0, 1, 0, 1, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 0, 1, 0, 1) : [5] , (0, 1, 0, 1, 0, 1, 1, 0) : [2, 3, 5, 8] , (0, 1, 0, 1, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 1, 0, 0, 1) : [2, 3, 5, 8] , (0, 1, 0, 1, 1, 0, 1, 0) : [3] , (0, 1, 0, 1, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 1, 1, 0, 0) : [3, 4, 5, 6] , (0, 1, 0, 1, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 0, 1, 1, 1, 1, 1) : [1, 3, 5, 7] , (0, 1, 1, 0, 0, 0, 0, 0) : [1, 2, 7, 8] , (0, 1, 1, 0, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 0, 0, 1, 1) : [2, 4, 6, 8] , (0, 1, 1, 0, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 0, 1, 0, 1) : [2, 3, 5, 8] , (0, 1, 1, 0, 0, 1, 1, 0) : [2] , (0, 1, 1, 0, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 1, 0, 0, 1) : [8] , (0, 1, 1, 0, 1, 0, 1, 0) : [2, 3, 5, 8] , (0, 1, 1, 0, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 1, 1, 0, 0) : [2, 4, 6, 8] , (0, 1, 1, 0, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 0, 1, 1, 1, 1) : [1, 2, 7, 8] , (0, 1, 1, 1, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 0, 0, 0, 1) : [5, 6, 7, 8] , (0, 1, 1, 1, 0, 0, 1, 0) : [2, 3, 6, 7] , (0, 1, 1, 1, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 0, 1, 0, 0) : [2, 4, 5, 7] , (0, 1, 1, 1, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 0, 1, 1, 1) : [1, 2, 5, 6] , (0, 1, 1, 1, 1, 0, 0, 0) : [3, 4, 7, 8] , (0, 1, 1, 1, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 1, 0, 1,

0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 1, 0, 1, 1) : [1, 3, 6, 8] , (0, 1, 1, 1, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (0, 1, 1, 1, 1, 1, 0, 1) : [1, 4, 5, 8] , (0, 1, 1, 1, 1, 1, 1, 0) : [1, 2, 3, 4] , (0, 1, 1, 1, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 0, 0, 0, 1) : [1, 2, 3, 4] , (1, 0, 0, 0, 0, 0, 1, 0) : [1, 4, 5, 8] , (1, 0, 0, 0, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 0, 1, 0, 0) : [1, 3, 6, 8] , (1, 0, 0, 0, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 0, 1, 1, 1) : [3, 4, 7, 8] , (1, 0, 0, 0, 1, 0, 0, 0) : [1, 2, 5, 6] , (1, 0, 0, 0, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 1, 0, 1, 1) : [2, 4, 5, 7] , (1, 0, 0, 0, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 0, 1, 1, 0, 1) : [2, 3, 6, 7] , (1, 0, 0, 0, 1, 1, 1, 0) : [5, 6, 7, 8] , (1, 0, 0, 0, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 0, 0, 0, 0) : [1, 2, 7, 8] , (1, 0, 0, 1, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 0, 0, 1, 1) : [2, 4, 6, 8] , (1, 0, 0, 1, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 0, 1, 0, 1) : [2, 3, 5, 8] , (1, 0, 0, 1, 0, 1, 1, 0) : [8] , (1, 0, 0, 1, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 1, 0, 0, 1) : [2] , (1, 0, 0, 1, 1, 0, 1, 0) : [2, 3, 5, 8] , (1, 0, 0, 1, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 1, 1, 0, 0) : [2, 4, 6, 8] , (1, 0, 0, 1, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 0, 1, 1, 1, 1, 1) : [1, 2, 7, 8] , (1, 0, 1, 0, 0, 0, 0, 0) : [1, 3, 5, 7] , (1, 0, 1, 0, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 0, 0, 1, 1) : [3, 4, 5, 6] , (1, 0, 1, 0, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 0, 1, 0, 1) : [3] , (1, 0, 1, 0, 0, 1, 1, 0) : [2, 3, 5, 8] , (1, 0, 1, 0, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 1, 0, 0, 1) : [2, 3, 5, 8] , (1, 0, 1, 0, 1, 0, 1, 0) : [5] , (1, 0, 1, 0, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 1, 1, 0, 0) : [3, 4, 5, 6] , (1, 0, 1, 0, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 0, 1, 1, 1, 1) : [1, 3, 5, 7] , (1, 0, 1, 1, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 0, 0, 0, 1) : [2, 3, 6, 7] , (1, 0, 1, 1, 0, 0, 1, 0) : [5, 6, 7, 8] , (1, 0, 1, 1, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 0, 1, 0, 0) : [3, 4, 7, 8] , (1, 0, 1, 1, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 0, 1, 1, 1) : [1, 3, 6, 8] , (1, 0, 1, 1, 1, 0, 0, 0) : [2, 4, 5, 7] , (1, 0, 1, 1, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 1, 0, 1, 1) : [1, 2, 5, 6] , (1, 0, 1, 1, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 0, 1, 1, 1, 1, 0, 1) : [1, 2, 3, 4] , (1, 0, 1, 1, 1, 1, 1, 0) : [1, 4, 5, 8] , (1, 0, 1, 1, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 0, 0, 0, 0) : [1, 4, 6, 7] , (1, 1, 0, 0, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 0, 0, 1, 1) : [4] , (1, 1, 0, 0, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 0, 1, 0, 1) : [3, 4, 5, 6] , (1, 1, 0, 0, 0, 1, 1, 0) : [2, 4, 6, 8] , (1, 1, 0, 0, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 1, 0, 0, 1) : [2, 4, 6, 8] , (1, 1, 0, 0, 1, 0, 1, 0) : [3, 4, 5, 6] , (1, 1, 0, 0, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 1, 1, 0, 0) : [6] , (1, 1, 0, 0, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 0, 1, 1, 1, 1) : [1, 4, 6, 7] , (1, 1, 0, 1, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 0, 0, 0, 1) : [2, 4, 5, 7] , (1, 1, 0, 1, 0, 0, 1, 0) : [3, 4, 7, 8] , (1, 1, 0, 1, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 0, 1, 0, 0) : [5, 6, 7, 8] , (1, 1, 0, 1, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 0, 1, 1, 1) : [1, 4, 5, 8] , (1, 1, 0, 1, 1, 0, 0, 0) : [2, 3, 6, 7] , (1, 1, 0, 1, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 1, 0, 1, 1) : [1, 2, 3, 4] , (1, 1, 0, 1, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 0, 1, 1, 1, 0, 1) : [1, 2, 5, 6] , (1, 1, 0, 1, 1, 1, 1, 0) : [1, 3, 6, 8] , (1, 1, 0, 1, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 0, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 0, 0, 0, 1) : [3, 4, 7, 8] , (1, 1, 1, 0, 0, 0, 1, 0) : [2, 4, 5, 7] , (1, 1, 1, 0, 0, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 0, 1, 0, 0) : [2, 3, 6, 7] , (1, 1, 1, 0, 0, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 0, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 0, 1, 1, 1) : [1, 2, 3, 4] , (1, 1, 1, 0, 1, 0, 0, 0) : [5, 6, 7, 8] , (1, 1, 1, 0, 1, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 1, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 1, 0, 1, 1) : [1, 4, 5, 8] , (1, 1, 1, 0, 1, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 0, 1, 1, 0, 1) : [1, 3, 6, 8] , (1, 1, 1, 0, 1, 1, 1, 0) : [1, 2, 5, 6] , (1, 1, 1, 0, 1, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 0, 0, 0, 0) : [7] , (1, 1, 1, 1, 0, 0, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 0, 0, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 0, 0, 1, 1) : [1, 4, 6, 7] , (1, 1, 1, 1, 0, 1, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 0, 1, 0, 1) : [1, 3, 5, 7] , (1, 1, 1, 1, 0, 1, 1, 0) : [1, 2, 7, 8] , (1, 1, 1, 1, 0, 1, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 1, 0, 0, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 1, 0, 0, 1) : [1, 2, 7, 8] , (1, 1, 1, 1, 1, 0, 1, 0) : [1, 3, 5, 7] , (1, 1, 1, 1, 1, 0, 1, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 1, 1, 0, 0) : [1, 4, 6, 7] , (1, 1, 1, 1, 1, 1, 0, 1) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 1, 1, 1, 0) : [1, 2, 3, 4, 5, 6, 7, 8] , (1, 1, 1, 1, 1, 1, 1, 1) : [1]}

```
for l in range(1,N):
   #all possible combinations of length l
```

```
    C = Combinations(range(N),l);
    a = C.list()
#    print "Combinations: ", a

    for i in range(len(a)):
        comb = a[i]
        comb_res = []

        for k in range (len(M.keys())):
            key = M.keys()[k]
            f = true
            for j in range(len(comb)):
                if (key[comb[j]]!=0):
                    f = false
            if (f):
              comb_res.append(M[key])

        #check if all results contain one and the same value
        lower_bound = false
        for i in range (1,N+1):
            i_in_all = true
            for j in range (len(comb_res)):
                if not(i in (comb_res[j])):
                    i_in_all = false
            if (i_in_all):
                lower_bound = l
                print "lower_bound: ", l
                sys.exit(0) #to stop executing the program after finding the lower bound
```

Maģistra darbs: **Kvantu vaicājošie algoritmi** izstrādāts LU Datorikas fakultātē.

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

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs(-ja): _____
(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.
(Iesniegšanas datums)
Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.
Studiju metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____
(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)