ŠIAULIŲ UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Edita Žitkevičienė
Informatikos specialybės magistrantūros II kurso neakivaizdinio skyriaus studentė

DAUGIAKARČIO TIKSLUMO ARITMETIKOS BIBLIOTEKA LiDIA KRIPTOGRAFIJOS KURSE

BAIGIAMASIS DARBAS

Darbo vadovė:
dr. R. Steuding

Recenzentas:
doc. V. Sirius

Šiauliai, 2005/2006 m.m.

Turinys

Iliustracijų sąrašas

1 Įvadas

1994 metų pradžioje Vokietijos Saarland'o universiteto skaičiuojamosios skaičių teorijos tyrimų grupė pradėjo kurti algoritmus. Kuriami algoritmai sveikųjų skaičių faktorizacijoms, nustatantys diskrečius logaritmus baigtiniuose kūnuose, skaičiuojantys taškus ant elipsinių kreivių virš baigtinių kūnų ir panašiai. Jų realizacijoms buvo naudojami trys skirtingi daugiakarčio tikslumo aritmetikos (multiprecision integer arihtmetic) paketai. Šiems algoritmams parašyti kodai buvo sunkiai skaitomi ir prastai aprašyti, todėl paprasčiausios procedūros dažnai buvo perrašomos iš naujo ir nelabai efektyviai. Jų realizacijoms buvo nuspręsta visą programinę įrangą sudėti į vieną biblioteką, kuri buvo pavadinta LiDIA.

Dėl šių priežasčių buvo nuspręsta visą programinę įrangą sudėti į vieną biblioteką, kuri buvo pavadinta LiDIA. Šią biblioteką sumodeliavo ir realizavo Tomas Papanikolaou. Jis sukūrė pagrindines bibliotekos gaires ir komponentus, tokius kaip branduolį, interfeisą ir didžiąją dalį aritmetinių komponentų.

Magistrinio darbo tema: daugiakarčio tikslumo aritmetikos biblioteka LiDIA kriptografijos kurse.

Pasirinkau šią temą, nes ji aktuali. Aktualumas: nagrinėjant ir realizuojant kriptografinius algoritmus programavimo C++ kalba dažnai prireikia įvairių matematinių funkcijų, kurios nėra aprašytos šioje programavimo kalboje. Studentai tokias funkcijas galės rasti nemokamai platinamoje C++ daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA, kuri yra aprašyta ir papildyta virtualiame kriptografijos kurse, ir galės jas tiesiogiai naudoti kurdami savo programas C++ kalba. Naudodamiesi šia daugiakarčio tikslumo aritmetikos biblioteka laboratorinių darbų metu studentai galės greičiau ir efektyviau realizuoti kai kurias kriptosistemas ir kriptografinius protokolus praktikoje. Be to, ši priemonė gali būti naudinga ir mokiniams, besidominantiems skaičių teorija ir kriptografija. Virtualaus kriptografijos kurso adresas: http://mokymai.dist.su.lt/

Magistrinio darbo tikslas: sukonstruoti pagrindines daugiakarčio tikslumo aritmetikos bibliotekos LiDIA funkcijas skirtas algoritmų (RSA schemos, pirminio skaičiaus testų, faktorizacijos algoritmų) realizacijai kriptografijos kurse.

Šiam tikslui realizuoti buvo iškelti šie reikia uždaviniai:

1. atlikti mokslinės literatūros analizę, kuri leis atrinkti funkcijas reikalingas realizuojant kriptografijos kursą;
2. atlikti įrankių analizę kriptografijos kurso ir virtualaus kriptografijos kurso realizacijai;
3. sukurti virtualų kriptografijos kursą, kur pagal temas parinkti ir sukurti uždaviniai kriptografijos mokymuisi (RSA schema, pirminio skaičiaus testai, faktorizacijos algoritmai, AKS algoritmas).

Problema: išstudijavusi kriptografijos kursą pastebėjau, kad trūksta medžiagos reikalingos kriptografijos kurse, nagrinėjamų algoritmų realizacijai.

Tyrimo objektas — daugiakarčio tikslumo aritmetikos biblioteka LiDIA, bei jos realizavimas.

Tyrimo metodai: mokslinės literatūros analizė, medžiagos sisteminimas, palyginimas, klausimyno bei gautų rezultatų analizė.

Praktinė darbo reikšmė: sukurtas virtualus kriptografijos kursas, papildo tradicinį mokymosi būdą, suteikdamas daugiau galimybių besimokantiems studentams.

Mokslinis naujumas: Lietuvoje skelbiamuose darbuose iki šiol nėra aprašyta daugiakarčio tikslumo aritmetikos biblioteka LiDIA, bei nėra sukurto virtualaus kriptografijos kurso realizuoto daugiakarčio tikslumo aritmetikos biblioteka LiDIA, bei parinktų uždavinių šios bibliotekos realizavimui.

2 Viešojo rakto kriptosistemos

Šiame skyriuje pateiksime viešojo rakto kriptografijos koncepciją. Toliau panagrinėsime vieną iš svarbiausių ir iki šiol naudojamų praktikoje viešojo rakto kriptografinių schemų — RSA kriptosistemą.1976 metais W. Diffie ir M.E. Hellman viešai paskelbė savo idėjas jų įžymiame straipsnyje „New Directions in Cryptography“ [9]. Jie aprašė viešo rakto metodą raktų apsikeitimui, kuris naudojamas iki šių dienų. Be to jie aprašė kaip veiks skaitmeniniai parašai ir paliko atvirą klausimą kaip surasti tam tinkamas funkcijas. Pirmoji viešo rakto kripto sistema, kuri galėjo veikti kaip funkcijos raktų apsikeitimui ir skaitmeniniam parašui, buvo RSA kripto sistema, aprašyta 1978 m. [13]. RSA pavadintas pagal jų atradėjų pavardes: R. Rivest A. Shamir, L. Adleman. RSA kriptosistema pateikia šifravimo ir skaitmeninio parašo technologijas ir ji yra viena populiariausių ir plačiausiai naudojamų kripto sistemų šiandien. RSA schemą bus aprašyta 2.3 skyriuje 10 puslapyje. Sistema paremta tuo, kad yra gana sudėtinga faktorizuoti didelius skaičius, kas suteikia galimybę sukurti vienos krypties funkcijas. Kita vienos krypties funkcijų bazė — tai diskrečiųjų algoritmų išskleidimo sudėtingumas. Šios dvi skaičių teorijos problemos yra daugelio šiuolaikinių kripto sistemų pagrindas. Pateiksime uždavinių ir jų sprendimų su LiDIA pavyzdžių, iliustruojančių RSA veikimo principus bei atskleidžiančių RSA efektyvumo ir saugumo problematiką.

2.1 Viešojo rakto kriptografijos koncepcija

Pagrindinė viešo rakto kriptografijos idėja — viešieji raktai. Kiekvieno asmens raktas yra suskirstytas į dvi dalis: viešas raktas šifravimui ir prieinamas visiems; ir slaptas raktas dešifravimui, kurį savininkas laiko paslaptyje. Šiame skyriuje apžvelgsime viešo rakto kriptografijos koncepciją.

Kriptografija - pavadinimas kilęs iš graikų kalbos žodžių kryptos „paslėptas“ ir graph „rašau“, kaip mokslas apie slaptąjį raštą, turi ilgą istoriją. Netrukus po to, kai senovės civilizacijos išrado raštą (maždaug keturi tūkstančiai metų prieš Kristų), atsirado ir bandymai šifruoti tekstus. Nemažai šifravimo pavyzdžių buvo rasta senovės Egipto, Indijos, Kinijos, Mesopotamijos istoriniuose dokumentuose. Tų laikų slaptaraščiai nepretendavo būti neįskaitomi -labiau buvo siekiama „mistinių“ tikslų. Tačiau mokėjimas atskleisti archeologinių tekstų prasmę glaudžiai siejasi su menu įminti šifravimo metodus [15].

Plačiai paplitus raštui, kriptografija pradėjo vystytis kaip mokslas. Pirmosios kriptografinės sistemos priklauso senovės graikams ir romėnams.Praėjusio amžiaus pasauliniai karai padarė didelę įtaką kriptografijos raidai. Tais laikais kriptografiniai metodai daugiausiai buvo naudojami karinių bei diplomatinių ryšių saugumui užtikrinti ir žvalgybinių organizacijų tikslams. Tada kriptografija tapo labai plačia ir svarbia moksline sritimi.

Sparčiai vystantis informacinėms sistemoms XX amžiaus antroje pusėje kriptografinės priemonės tapo neatsiejama mūsų kasdienio gyvenimo dalimi. Kompiuterinių technologijų naudojimas įvairioje veikloje reikalauja užtikrinti laikomos ir perduodamos

informacijos saugumą. Su informacijos saugumo aspektais susijusius matematinius metodus nagrinėja kriptografijos mokslas.

Klasikinė simetrinė kriptografija pateikia saugų duomenų apsikeitimo kanalą kiekvienai vartotojų porai. Kad tokį kanalą sukurti, vartotojai turi naudoti sutartą slaptą raktą. Po to kai saugus kanalas sukurtas, duomenų saugumas yra užtikrintas. Simetrinė kriptografija taip pat turi metodus, kurių pagalba aptinka duomenų pakeitimą ir patikrina pranešimo originalumą. Tokiu būdu slapto rakto technika užtikrina duomenų apsikeitimo konfidencialumą ir vientisumą.

Viešo rakto technologija naudojama saugiam slaptų raktų pasiskirstymui, taip pat kai kurios svarbios autentifikacijos formos (tokios kaip skaitmeninis parašas) reikalauja viešo rakto metodų. Skaitmeninis parašas turėtų būti skaitmeninis priedas prie ranka rašyto parašo. Skaitmeninis parašas turi priklausyti nuo pasirašytos žinutės turinio, o slaptas raktas turėtų būti žinomas tik parašo savininkui. Nešališkas trečios šalies atstovas turi galėti patikrinti parašo autentiškumą neturėdamas priėjimo prie pasirašiusiojo slapto rakto.

Viešo rakto šifravimo schemoje tarp dviejų vartotojų nėra bendro slapto rakto. Kiekvienas vartotojas turi raktų porą: *slaptą raktą* sk žinomą tik jam pačiam ir *viešą raktą* pk, kurį žino visi. Viešas raktas parametrizuoja šifravimų funkcijų šeimą $(E_{pk})_{pk \in PK}$, kur PK žymi viešų raktų aibę. Šifravimo funkcijų šeima $(E_{pk})_{pk \in PK}$ taip pat yra viešai žinoma informacija.

Tarkime, kad Bobas naudoja viešo rakto kriptosistemą ir Alisa nori užšifruoti Bobui žinutę m. Bobas turi slaptą raktą sk ir viešą raktą pk. Kaip ir visi kiti, Alisa žino Bobo viešą raktą pk Ji paprasčiausiai ima Bobo viešo rakto šifravimo funkciją E_{pk} ir apskaičiuoja kriptogramą $c = E_{pk}(m)$. Akivaizdu, kad sistema bus saugi tik tokiu atveju, jei bus praktiškai neįmanoma apskaičiuoti m iš $c = E_{pk}(m)$. Bet kaip tuomet Bobas atstatys žinutę m iš c? Šioje vietoje bus panaudotas Bobo slaptas raktas. Šifravimo funkcija $E_{pk}(m)$ turi turėti tokią savybę, kad žinutę m būtu galima apskaičiuoti pasinaudojant Bobo slaptu raktu sk. Kadangi tik Bobas žino savo slaptą raktą, jis yra vienintelis, kuris gali dešifruoti pranešimą. Netgi Alisa, kuri užšifravo žinutę m negalės apskaičiuoti m iš $E_{pk}(m)$ jei praras originalią žinutę m. Žinoma, turi egzistuoti efektingi algoritmai, kurie atliktų šifravimą ir dešifravimą.

Kiekvienas viešo rakto kriptosistemos vartotojas turi turėti savo asmeninį raktą $k = (pk, sk)$, sudaryto iš viešos ir slaptos (dar vadinamos privačia) dalies. Kad užtikrinti kriptosistemos saugumą, turi būti neimanoma apskaičiuoti slaptą raktą sk iš viešo rakto pk, taip pat turi būti įmanoma pasirinkti atsitiktinį raktą k iš milžiniškos parametrų erdvės. Turi būti efektingas algoritmas, kuris atliktų šį atsitiktinį pasirinkimą. Jei Bobas nori dalyvauti šioje kripto sistemoje, jis turi atsitiktinai pasirinkti savo raktą $k = (pk, sk)$, laikyti sk slaptai ir paviešinti pk. Tuomet keikvienas galės naudoti pk, kad užšifruoti pranešimus Bobui.

Dar viena viešo rakto kriptosistemos panaudojimo sritis, tai sesijos raktų dalijimas. Sesijos raktas, tai slaptas raktas, naudojamas klasikinėje šifravimo schemoje, kad šifruo-

ti duomenis vienai apsikeitimo sesijai. Jei Alisa žino Bobo viešą raktą (ir juo pasitiki), tuomet ji gali sugeneruoti sesijos raktą, jį užšifruoti su Bobo viešu raktu ir jį išsiųsti Bobui. Skaitmeniniai parašai naudojami užtikrinti viešų raktų sertifikavimo agentūrų autentiškumui. Sertifikavimo agentūra pasirašo kiekvieno vartotojo viešą raktą savo slaptu raktu. Parašas gali būti patikrintas pasinaudojant patikimu viešuoju sertifikavimo agentūros raktu. Šiandien šios sistemos yra Interneto komunikacijų ir elektroninės komercijos pagrindas.

2.2 Modulinės aritmetikos elementai

Šiame skyriuje trumpai apžvelgsime modulinę aritmetiką, kuri yra būtina norint suprasti kriptosistemas. Daugiau informacijos šia tema galima rasti vadovėlyje [12] arba [10].

Tarkime, kad m yra teigiamas sveikasis skaičius. Du sveikieji skaičiai a ir b lygsta moduliu m, jei m dalija $a - b$, ir rašysime [15]:

$$a \equiv b \bmod m \tag{1}$$

Pavyzdžiui: $-5 \equiv 3 \bmod 8$. Likinių klasę $a \bmod m$, $m > 1$, sudaro visi sveikieji skaičiai b, kurie lygsta a moduliu m.

Akivaizdu, kad yra m skirtingų likinių klasių moduliu m, o kiekvienai iš jų atstovauja *reprezentuojantis elementas*. Aibė iš elementų $0, 1, m - 1 \bmod m$ yra viena iš reprezentuojančių elementų aibių. Pavyzdžiui, $\{0, 1, 2, 3\}$ yra likinių klasių moduliu 4 aibė. Visų likinių klasių moduliu m aibę žymėsime $\mathbb{Z}/n\mathbb{Z}$. Tada, „perkėlę“ sudėties ir daugybos veiksmus iš sveikųjų skaičių aibės $\mathbb{Z}$ į aibę $\mathbb{Z}/n\mathbb{Z}$ pagal taisykles

$$a \bmod m \overset{+}{\underset{\times}{}} b \bmod m = (a \overset{+}{\underset{\times}{}} b) \bmod m,$$

gauname *likinių klasių moduliu m žiedą* $\mathbb{Z}/n\mathbb{Z}$. Galima įrodyti, kad žiedas yra $\mathbb{Z}/n\mathbb{Z}$ kūnas tada ir tik tada, kai m yra pirminis skaičius; priešingu atveju egzistuoja nulio dalikliai. Tai iliustruojama pavyzdys:

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

1 pav. Daugyba mod 3 ir mod 4

Aibė

$$(\mathbb{Z}/m\mathbb{Z})^* = \{a \bmod m : (a, m) = 1\}$$

yra žiedo $\mathbb{Z}/n\mathbb{Z}$ multiplikatyvioji grupė; šios grupės eilė lygi $\phi(m)$, kur $\phi(m)$ žymi *Oilerio* (*Euler*) *funkciją*, t.y. sveikųjų skaičių $1 \leq a \leq m$, tarpusavyje pirminių su m, skaičių. Nesunku įsitikinti, kad $\phi(p) = p - 1$, jei p yra pirminis skaičius. Elemento $a \in (\mathbb{Z}/m\mathbb{Z})^*$ vadinamas toks mažiausias teigiamas sveikasis skaičius k, kad

$$a^k \equiv 1 \bmod m \tag{2}$$

Mažoji Ferma (*Fermat*) *teorema* teigia, kad jei p yra pirminis skaičius, o sveikasis skaičius a nėra dalus iš p, tai

$$a^{p-1} \equiv 1 \bmod p.$$

Yra žinomas šiek tiek stipresnis mažosios Ferma teoremos formulavimas, priklausantis L. Oileriui.

Teorema Sakykime, $(a, m) = 1$. Tuomet

$$a^{\phi(m)} \equiv 1 \bmod m \tag{3}$$

Nemažai taikymų kriptografijoje turi *kinų teorema liekanoms*.

Teorema (kinų teorema liekanoms). Tarkime, $m_1, \ldots, m_r$ yra poromis tarpusavyje pirminiai skaičiai, o $a_1, \ldots, a_r$ yra bet kurie sveikieji skaičiai. Tuomet lyginių sistema

$$x \equiv a_j \bmod m_j, \quad 1 \leq j \leq r,$$

turi vienintelį sprendinį moduliu $m = m_1 \cdot \ldots \cdot m_r$.

Dalyba su liekana. Jei m ir n yra sveikieji skaičiai, $m \neq 0$, galima dalinti n iš m su liekana. Galima rašyti $n = q \cdot m + r$, vieninteliu būdu, kad $0 \leq r < \text{abs}(m)$. [1] Skaičius q vadinamas *„dalmuo“*, o r vadinamas dalybos *„liekana“*. Jie yra unikalūs. Dažniausiai r žymimas kaip $a \bmod b$.

Sveikasis skaičius m dalija sveikąjį skaičių n, jei n yra m kartotinis ($n = mq$ sveikajam skaičiui q). Tuomet sakome, kad m yra n daliklis arba faktorius. Didžiausio bendrojo daliklio funkcija gcd. Skaičių $m, n \neq 0$ *didžiausias bendras daliklis* $\gcd(m, n)$ yra didžiausias teigiamas skaičius, dalijantis m ir n. Sutarta, kad $\gcd(0, 0)$ yra nulis. Jei $\gcd(m, n) = 1$, tuomet m vadinamas *reliatyviai pirminiu skaičiui* n arba trumpiau *pirminiu skaičiui* n.

Euklido algoritmas apskaičiuoja dviejų skaičių didžiausią bendrą daliklį ir tai yra vienas seniausių algoritmų matematikos istorijoje:

[1] $\text{abs}(m)$ žymi absoliučią m reikšmę.

1. int $gcd(\text{int}\, a, b)$
2. kol $b \neq 0$ kartoti
3. $\quad r \leftarrow a \bmod b$
4. $\quad a \leftarrow b$
5. $\quad b \leftarrow r$
6. grąžinti abs(a)

Algoritmas apskaičiuoja gcd(a, b), kur $a \neq 0$ ir $b \neq 0$. Algoritmas yra baigtinis, kadangi teigiamas skaičius r padidinamas kiekviename žingsnyje. Yra žinoma, kad šis algoritmas veikia lėčiausiai su Fibonačio skaičiais.

Fibonačio skaičiais vadinama tokia seka:

$$F_0 = F_1 = 1; F_n = F_{n-1} + F_{n-2}, \text{ kur } n\text{=2,3,...}$$

Daugiakarčio tikslumo bibliotekos LiDIA lietuvišką aprašymą žiūrėkite 1 priede, o anglišką šios bibliotekos aprašymą galite parsisiųsti iš `ftp://ftp.informatik.tu-darmstadt.de/TI/systems/LiDIA`. Išspręsime uždavinį su daugiakarčio tikslumo aritmetikos biblioteka LiDIA. Tarkime funkcija `gcd`(a, b) grąžina skaičių a ir b didžiausia bendrą daliklį. Rekursinis šios funkcijos apibrėžimas:

1. gcd($a, 0$) = a
2. remainder(liekana, a, b);
3. jei liekana $\neq 0$
4. $\quad a =$ `gcd`($b, liekana$);
5. grąžinti (a)

gali sukelti problemų, jei rekursijoje yra per didelis žingsnių kiekis. Patartina naudoti tradicinį iteracinį metodą:

1. `gcd`(a, b)
2. kol $b \neq 0$ kartoti
3. $\quad r \leftarrow a \bmod b$
4. $\quad a \leftarrow b$
5. $\quad b \leftarrow r$
6. grąžinti abs(a)

Palyginsime rekursinio ir iteracinio algoritmų veikimo trukmę testuodami su didelių Fibonačio skaičių poromis. Uždavinys generuos x atsitiktinių skaičių nuo 0 iki „100 000“. Sugeneravus vieną skaičių z, programa apskaičiuos `bigint` tipo z-tąjį ir $(z + 1)$-tąjį Fibonačio skaičius ir paskaičiuos šių skaičių bendrą didžiausią daliklį rekursiniu ir iteraciniu būdu. `bigint` — tai klasė, kuri pateikia daugiakarčio tikslumo sveikųjų skaičių

aritmetiką. Kintamajam, kurio tipas `bigint`, gali būti priskirtas sveikasis skaičius su kiek norima daug skaitmenų. Kadangi kintamojo tipas `int` taip pat aprašytas klasėje `bigint`, galima sakyti, kad tipas `bigint` yra tipas `int` tik be apribojimų (žiūrėti 1 priede). Pabaigusi darbą, programa palygins kiek laiko truko skaičiavimai skirtingais metodais. Programos kodą žiūrėkite kompaktinio disko kataloge „uzd01“. Pasirinkus generuoti 4 atsitiktinius skaičius gaunami tokie rezultatai:

```
Kiek atsitiktinių skaičių generuoti (x)?: 4
84019-asis  ir 84020-asis Fibonačio skaičiai:
Iteracinis skaičiavimas užtruko: 1420000
Rekursinis skaičiavimas užtruko: 8310000

39439-asis  ir 39440-asis Fibonačio skaičiai:
Iteracinis skaičiavimas užtruko: 370000
Rekursinis skaičiavimas užtruko: 1660000

78310-asis  ir 78311-asis Fibonačio skaičiai:
Iteracinis skaičiavimas užtruko: 1290000
Rekursinis skaičiavimas užtruko: 6850000

79845-asis  ir 798 Fibonačio skaičiai:
Iteracinis skaičiavimas užtruko: 1310000
Rekursinis skaičiavimas užtruko: 7240000

Programa užtruko: 29990000
Iteraciniu būdu užtruko: 4390000
Rekursiniu būdu užtruko: 24060000
```

Iš gaunamų rezultatų matyti, kad rekursiniu būdu skaičiavimai atliekami daug ilgiau, nei iteraciniu. Jei kompiuteris neturi pakankamai resursų,tai rekursiniu metodu uždavinį išspręsti nepavyks. Raskime tokią skaičių porą, kuriai rekursinis metodas neveikia.

Išspręskime lyginių sistemą:

$$\begin{cases} x \equiv 1 \bmod 2 \\ x \equiv 1 \bmod 3 \\ x \equiv 1 \bmod 5 \\ x \equiv 1 \bmod 7 \end{cases}$$

Pasinaudoję daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA aprašyta *kinų teoremą liekanoms*, sudaromelyginių liginių sistemai funkciją:

```
bigint liginiu_sistema(const bigint & y1, const bigint & y2, const bigint & y3,
 const bigint & y4, const bigint & m1, const bigint & m2,
 const bigint & m3, const bigint & m4)
{
  bigint x, x1, x2;

  x  = chinese_remainder(y1, m1, y2, m2);
  x1 = chinese_remainder(x, m1*m2, y3, m3);
  x2 = chinese_remainder(x1, m1*m2*m3, y3, m3);
  return x2;
}
```

ir ją išsprendžiame. Programos kodą žiūrėkite kompaktinio disko kataloge „liginiu_sistema“.

Uždaviniai savarankiškam darbui.

1. Parašykite rekursinę funkciją apskaičiuojančia Euklido algoritmo sekos ilgį. Pavyzdžiui: Euklido algoritmo sekos ilgis `EALength`(a, b). Jei Euklido algoritmo sekos ilgis `EALength`(97, 18), tai turėtų grąžinti 5, nes:

$$\begin{aligned} 97 &= 18 \cdot 5 + 7; \\ 18 &= 7 \cdot 2 + 4; \\ 7 &= 4 \cdot 1 + 3; \\ 4 &= 3 \cdot 1 + 1; \\ 3 &= 1 \cdot 3 + 0. \end{aligned}$$

   Transformuokite gautą rekursinę funkciją į iteracinę, išbandykite ne mažiau kaip 100 pavyzdžių ir suraskite gautų sekų ilgių vidurkį.

2. Daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA funkcija `extended gcd`(a, b) duotiems skaičiams apskaičiuoja jų didžiausią bendrą daliklį d ir suranda sveikuosius skaičius s ir t, tokius, kad $d = s \cdot a + t \cdot b$. Parašykite programą, kuri įvedus a ir b sugeneruoja pilnas s ir t sekas atitinkančias Euklido algoritmą skaičiams a ir b.

3. Ištirkite problemą: kiek procentų atsitiktinių sveikųjų skaičių porų (x, y) yra tarpusavyje pirminiai skaičiai, tai yra `gcd`(x, y)=1. Išnagrinėkite keletą tūkstančių atsitiktinių porų tarp 1 ir 10^6, ir apskaičiuokite kokia dalis iš jų yra tarpusavyje pirminiai skaičiai. Pasinaudoję eksperimento rezultatais pabandykite rasti formulę įvertinančią tikimybę, kad atsitiktinai paimta skaičių pora bus tarpusavyje pirminiai skaičiai (į dydį nusakantį minėtą tikimybę įeina π^2).

2.3 RSA schema

1978 metais R.Rivestas (R.Rivest), A.Šamiras (A.Shamir) ir L.Eidlmanas (L.Adleman) [...] išrado pirmąją viešojo rakto kriptosistemą, kuri buvo pavadinta RSA sistema [13]. RSA algoritmas ir dabar vaidina svarbų vaidmenį kriptografiniuose taikymuose. Mes aptarsime tik pagrindines RSA schemos idėjas.

2.3.1 RSA schemos raktų generavimas

RSA viešojo rakto generavimui reikalingi du atsitiktiniai dideli pirminiai skaičiai. Sakykime, turime du atsitiktinius didelius pirminius skaičius p ir q. Jų sandaugą pažymime N. N vadinamas RSA moduliu.

Kuriant programas reikia apsirašyti skaičiaus tipą. Daugiakarčio tikslumo bibliotekoje LiDIA `bigint` tipas atitinka `int`. `bigint` — tai klasė, kuri pateikia daugiakarčio tikslumo sveikųjų skaičių aritmetiką. Kintamajam, kurio tipas `bigint`, gali būti priskirtas sveikasis skaičius su kiek norima daug skaitmenų. Kadangi kintamojo tipas `int` taip pat aprašytas klasėje `bigint`, galima sakyti, kad tipas `bigint` yra tipas `int` tik be apribojimų.

Norint rasti pirminį skaičių daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA, patogu naudoti funkcija `next_prime(const bigint & a)`, kuri sugeneruoja sekantį po sveikojo skaičiaus a pirminį skaičių. Reiktų rašyti:

`bigint next_prime (const bigint & a)`

tada grąžina mažiausia sveikąjį skaičių, didesnį už a, tenkinantį funkcijos `is_prime(a)` pirminio skaičiaus testą. Bet kuriam $a \leq 1$ ši funkcija grąžina 2.Plačiau skaityti daugiakarčio tikslumo aritmetikos bibliotekos žinyne [7].

Po to pagal formulę:

$$\phi(N) = (p-1)(q-1)$$

apskaičiuojame Oilerio funkciją. Norint apskaičiuoti privatų raktą reikia žinoti šifravimo eksponentę e. Pasirenkame sveikąjį skaičių e, tenkinantį sąlygas $1 < e < \phi(N)$ ir $(e, \phi(N)) = 1$. Norėdama rasti šifravimo eksponentę e pasinaudojau objekto priskyrimo metodu `assign`, kadangi šifravimo eksponentė e turi apribojimus, aprašiau ciklą:

```
e.assign(3);
tinka = false;
while (!tinka){
  if (gcd(e, phi)==1){
  tinka = true;
}
```

```
else{
  e = e + 2;
}
```

Privatus raktas žymimas d ir randamas pagal formulę:

$$d = e^{-1} \bmod (p-1)(q-1).$$

Daugiakarčio tikslumo aritmetikos bibliotekos LiDIA dėka nesunku buvo patikrinti ar du atsitiktinai pasirinkti dideli skaičiai yra pirminiai ar ne, bei sugeneruoti viešą raktą, apskaičiuoti šifravimo eksponentę ir privatų raktą.

2.3.2 Šifravimas ir dešifravimas pagal RSA schemą

RSA kriptosistemoje užšifruoti (šifruoti) ir iššifruoti (dešifruoti) duomenys siunčiami, bei gaunami pranešimai pagal formules. Pasižymime žinutę m, kur žinutė turi tenkinanti sąlygas $m < N$ (žinutė m turi būti mažesnė už RSA modulį N). Sakykime užšifruotas tekstas žymimas c ir galime apskaičiuoti:

$$c = m^e \bmod N.$$

Daugiakarčio tikslumo bibliotekoje LiDIA žinutė užšifruota su funkcija:

`void power (bigmod &` c`, const bigmod &` a`, long` b`)`

$c \leftarrow ab$ (atliekama iš dešinės į kairę dvejetaine eksponentacija)

Pavyzdys:

```
bigmod c, m;
bigmod::set_modulus(N);
cout << "Iveskite zinute skaiciais: tarp 0 ir " << N << "; ";
cin >> m;
power(c, m, e);
cout << "Uzsifruota zinute: " << c << "\n";
```

Daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA iškvietus funkciją `set_modulus(`m`)` po kintamųjų, kurių tipas `bigmod` aprašymo, visi šie kintamieji gali būti neteisingi. Jie gali turėti mantisas kurios yra netinkamame rėžyje. Norėdami naudoti `bigmod` tipo kintamuosius po to kai globalus modulus pasikeičia, juos reikės normalizuoti.

Gautą pranešimą dešifruoti galime pagal formulę:

$$m = c^d \bmod N.$$

Dešifruoti galime analogiškai kaip ir užšifruojant su funkcija `power`.(Programos kodą žiūrėti kompaktinio disko kataloge „rsa_ataka“).

2.3.3 RSA schemos atakos

Siunčiant užšifruotą pranešimą, visada yra asmenų norinčių „nulaužti“ siunčiamą pranešimą. Panagrinėsime keletą RSA schemos atakų:

- žemos eksponentės ataka;
- bendrojo modulio ataka;
- RSA modulio faktorizacija.

Žemos eksponentės ataka. Tarkime žinutė m užšifruota pagal RSA schemą, naudojant viešuosius raktus tokias skaičių poras $(3; 391)$, $(3; 55)$, $(3; 87)$. Atitinkamai užšifruoti tekstai yra 208; 38; 32. Pasinaudodami žemos eksponentės ataka raskime užšifruotą žinutę m. Šiuo atveju užšifravimo eksponentė $e = 3$, RSA moduliai N taip pat žinomi, $N1 = 391$; $N2 = 55$; $N3 = 87$. Visi N tarpusavyje pirminiai dėl to atakuojantysis pagal *kinų teoremą liekanoms* gali surasti žinutę m.

`bigint chinese_remainder (const bigint &` a`, const bigint &` m`,`
`const bigint &` b`, const bigint &` n`)`

grąžina mažiausią neneigiamą sveikąjį skaičių, kuris atitinka a mod m ir b mod n. Jei sprendinys, tenkinantis abi sąlygas vienu metu, neegzistuoja (m ir n nebūtinai tarpusavyje pirminiai skaičiai), tai iškviečiamas `lidia_error_handler`.

Su programos kodu galima susipažinti kompaktinio disko kataloge „rsa_ataka“.

Mažą šifravimo eksponentę siūlyčiau nenaudoti, kai norite išsiųsti tą pačią žinutę daugeliui gavėjų. Apsisaugoti nuo žemos eksponentės atakos galima prie atviro teksto blokų prijungiant keletą atsitiktinai parinktų bitų, tada žemos eksponentės ataka yra neveiksminga. Kita alternatyva būtų didesnė šifravimo eksponentės parinkimas; pavyzdžiui, $e = 2^{16} + 1$ yra gana populiarus pasirinkimas.

Bendrojo modulio ataka. Sakykime žinutė m yra užšifruota pagal RSA schemą naudojant viešuosius raktus $(3, 493)$, $(5, 493)$. Slaptaraščiai yra 293 ir 421. Naudodami bendrojo modulio ataką rasime užšifruotą žinutę m. Taigi žinomos šifravimo eksponentės $e1 = 3$, $e2 = 5$, kurios tarpusavyje yra pirminės. RSA modulis $N = 493$. Kaip ir praeitame uždavinyje analogiškai vyksta pranešimų užšifravimas:

```
bigmod::set_modulus(N);
bigmod m, c1, c2;
bigint c1_, c2_;
m = 142;
c1 = RSAsifravimas(m, e1);
c1_ = c1.mantissa();
cout << "Pirma uzsifruota zinute: " << c1_ << "\n";
```

```
c2 = RSAsifravimas(m, e2);
c2_ = c2.mantissa();
cout << "Antra uzsifruota zinute: " << c2_ << "\n";
```

Bendrojo modulio ataka naudoja Euklido algoritmą, nustatant didžiausio bendro daliklio tiesinės kombinacijos koeficientus:

$$ue1 + ve2 = 1.$$

Daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA iškvietus aukšto lygio funkciją `xgcd` surandame didžiausius bendro daliklio tiesinės kombinacijos koeficientus. Plačiau paskaityti galima daugiakarčio tikslumo aritmetikos bibliotekos LiDIA žinyne [7].

`bigint xgcd (bigint &` u`, bigint &` v`, const bigint &` a`, const bigint &` b`)`
grąžina a ir b gcd; rezultatas yra teigiamas, `bigint` tipo. Apskaičiuojami koeficientai u ir v, kur $|u| \leq \frac{|b|}{2\gcd(a,b)}$ ir $|v| \leq \frac{|a|}{2\gcd(a,b)}$, tokie, kad $\gcd(a,b) = u \cdot a + v \cdot b$, pvz. naudojant išplėsta gcd algoritmą.

Pavyzdys:

```
bigint u, v;
xgcd(u, v, e1, e2);
cout << "u: " << u << "; v: " << v <<"\n";
bigmod m1, m2, m_;
power(m1, c1, u);
power(m2, c2, v);
m_ = m1 * m2;
cout << "Nulauzta zinute: " << m_ << "\n";
```

Visą programos kodą žiūrėti kompaktinio disko kataloge „rsa_ataka2“.

Taigi egzistuoja nemažai būdų kaip galima „nulaužti“ siunčiamus pranešimus, tai jau minėtos atakos (žemos eksponentės ataka, bendrojo modulio ataka), dar yra ciklinė ataka. Atkurti slaptaraštį galima ir pagal Kiniečių teoremą liekanoms ar Vinerio (Wiener) algoritmą. Kad to išvengti reiktų pasirinkti didelę šifravimo eksponentę, jei viešas raktas (n, e) tai dešifravimo eksponentę d naudoti tokio pat dydžio, kaip n.

Uždaviniai savarankiškam darbui .

1. Tarkime, kad ši 40 raidžių abėcėlė naudojama visiems tekstams ir šiframs: A-Z su skaitiniais analogais 0-25, tarpas=26, .=27, ?=28, $=29, skaičiai su skaitiniais analogais 30-39. Tarkime, kad tekstiniai pranešimai yra digrafai ir šifruotės yra trigrafai (pavyzdžiui, $k = 2,\ l = 3, 40^2 < n_A < 40^3$ visiems n_A).

a) Išsiųskite pranešimą „SEND \$7500“ vartotojui, kurio šifravimo raktas yra $(n_A, e_A) = (2047, 179)$.

b) „Nulaužkite“ kodą faktorizuojant n_A, po to apskaičiuodami dešifravimo raktą (n_A, d_A).

c) Paaiškinkite kodėl įsilaužėlis gali palyginti greitai rastą dešifravimo raktą netgi nefaktorizuodamas n_A. Kitaip sakant, kodėl (neatsižvelgiant į jo mažumą) 2047 yra netinkamas kandidatas į n_A?

2. Pabandykite „nulaužti“ kodą, kurio šifravimo raktas $(n_A, e_A) = (536813567, 3602561)$. Pasinaudokite kompiuteriu n_A faktorizavimui. Naudokite kvailiausią žinomą algoritmą (atliekant dalybą ir visų nelyginių skaičių 3, 5, 7, ...). Jei neturite kompiuterio, bandykite atspėti pirminį n_A faktorių naudodami specialias pirminių skaičių klases. Faktorizavę n_A raskite dešifravimo raktą. Tuomet dešifruokite pranešimą BNBPPKZAVQZLBJ, su sąlyga, kad tekstas sudarytas iš 6 raidžių blokų įprastinėje 26 raidžių abėcėlėje (pakeiskite į sveikąjį skaičių tarp 0 ir $26^6 - 1$). Šifruotas tekstas sudarytas iš 7 raidžių blokų toje pačioje abėcėlėje. Iš šio uždavinio turėtu paaiškėti, kad net 29 bitų n_A yra stipriai per mažas.

3. Tegul n yra sveikasis skaičius be šaknų. Tegul d ir e yra teigiami sveikieji skaičiai taip, kad $de-1$ yra dalus iš $p-1$ kiekvienam pirminiam dalikliui p iš n (Pavyzdžiui jei $de \equiv 1 \bmod \varphi(n)$.). Įrodykite, kad $a^{de} \equiv a \bmod n$ *bet kokiam* sveikajam skaičiui a (nesvarbu ar jis yra pirminis n faktorius).

2.4 Pirminio skaičiaus testai

Jau prieš kelis šimtmečius matematikai sprendė dvi problemas:

- pirminiai skaičiai;
- RSA modulio faktorizacija.

Kas yra pirminis skaičius sužinome mokykloje. Sveikas skaičius $n > 1$ yra vadinamas *pirminiu*, jei jis turi tik du teigiamus daliklius: 1 ir n. Su nedideliais skaičiais viskas paprasta, bet kaip nuspręsti ar duotas teigiamas sveikasis skaičius n pirminis ar ne, su dideliais skaičiais? Kita problema - tai faktorizacija. Sakykime duotas teigiamas sveikasis skaičius n, reiktų rasti n pirminę faktorizaciją. Šiuo metu tai vienas iš aktualiausių klausimų.

Ypač šios problemos paaštrėjo vystantis kriptografijai. Prisiminkime, kad sudauginti du didelius pirminius skaičius yra nesunku, tačiau daug sunkiau yra faktorizuoti duotą didelį sveikąjį skaičių, bent jau patenkinamo greičio faktorizacijos algoritmai kol kas nėra žinomi. Viešojo rakto kriptosistemose naudojamas raktas yra daugiau kaip dviejų šimtų

skaitmenų sveikasis skaičius n, kuris yra viešai žinomas, bet šio skaičiaus pirminė faktorizacija yra rakto savininko paslaptis. Kriptosistema bus nesaugi jeigu n suskaidomas mažais pirminiais daugikliais. Bet jei n yra dviejų (tinkamai parinktų) pirminių skaičių, turinčių daugiau kaip šimtą skaitmenų, sandauga, tai skaičiaus n faktorizacija, galima sakyti, yra neišsprendžiamas uždavinys šiuolaikiniams kompiuteriams. Tokių raktų generavimui reikalingi dideli pirminiai skaičiai. Kitaip tariant, reikia turėti greitą pirminio skaičiaus testą.

Viena iš pirmųjų idėjų, kylančių norint faktorizuoti duotą skaičių n, galėtų būti *bandomoji dalyba*, t. y. bandyti dalinti n paeiliui iš visų teigiamų sveikųjų skaičių $\sqrt{n}$. Akivaizdu, jei tarp jų nėra skaičiaus n daliklių, tai n yra pirminis skaičius. Ši strategija nėra efektyvi dideliems n. Paprasta bandomosios dalybos idėja yra Eratosteno rėčio pagrindas. Šį metodą pasiūlė graikų matematikas Eratostenas, kuris pirmasis apytiksliai „išmatavo“ Žemės perimetrą 250 metais prieš Kristų. Jei iš sveikųjų skaičių $1 < nx$ sąrašo pašalintume visus pirminių skaičių $p\sqrt{x}$ kartotinius n, tuomet liktų vien tik pirminiai skaičiai tarp $\sqrt{x}$ ir x. Tokiu būdu galime rasti visus pirminius skaičius tam tikrame intervale ir tai, ko gero, geriausias algoritmas iki šiol šiam tikslui pasiekti (neskaitant jo nežymiai patobulintų versijų). Dar daugiau, mes gauname visų sąrašo sveikųjų skaičių faktorizaciją. Ir faktorizacijos algoritmui, ir pirminio skaičiaus testui visos šios informacijos yra per daug, todėl galima pagalvoti apie greitesnius metodus.

Dauguma parašytų pirminio skaičiaus testų yra tikimybiniai. Prisiminkime, kad pirminio skaičiaus testas laikomas „greitu“, jei jis yra polinominio laiko įvedamų duomenų atžvilgiu. Jei mus tenkina pirminio skaičiaus testas, pateikiantis korektišką atsakymą su „didele tikimybe“, tuomet galime pasinaudoti greitesniu algoritmu, nei bandomoji dalyba. Mažoji Ferma teorema teigia, kad jei p yra pirminis ir a nėra p kartotinis, tuomet

$$a^{p-1} \equiv 1 \bmod p \tag{4}$$

Mažajai Ferma teoremai atvirkštinė teorema neteisinga, tai patvirtina toks pavyzdys:

$$2^{340} \equiv 1 \bmod 341 \tag{5}$$

ir $341 = 11 \cdot 31$ Sudėtinis skaičius n, kuriam galioja sąryšis

$$a^{n-1} \equiv 1 \bmod n \tag{6}$$

yra vadinamas *pseudopirminiu skaičiumi* pagrindu a.

AKS pirmas determinantinis polinominio laiko testas.

Sakykime Alisa užšifruoja žinutę m naudodama Bobo viešą RSA raktą (11, 899). Užšifruotas tekstas yra 468. Reikia atstatyti užšifruotą žinutę m. Iš sąlygos žinome c=468, e=11, N=899. Pagal formulę:

$$m = c^d \bmod N.$$

galėtumėm dešifruoti žinutę, tik reikia surasti privatų raktą d. Privatus raktas randamas pagal formulę:

$$d = e^{-1} \bmod \phi(N).$$

Žinome, kad $N = p * q$, jei žinosime N faktorizaciją, tai galėsime apskaičiuoti Oilerio funkciją ir privatų raktą. Sprendžiant šį uždavinį daugiakarčio tikslumo aritmetikos LiDIA bibliotekoje teko prisijungti ne tik prie `bigint` ir `bigmod` klasių, bet ir prie `rational_factorization` klasės. Funkcijos `factor` pagalba du didelius pirminius skaičius p ir q. Viešo rakto skaidymas daugynamaisiais LiDIA bibliotekoje:

```
rational_factorization f;
f = factor (N);
p = f.base(0); q = f.base(1);
cout << "Faktorizacija: " << N << "=" << p << "*" << q << "\n";
```

Programos kodą žiūrėti kompaktinio disko kataloge „rsa_ataka2“. Taigi rašant programas su daugiakarčio tikslumo aritmetikos biblioteka LiDIA ne visos klasės turi konkrečiai programai reikalingas vienas ar kitokias funkcijas, todėl reikia atidžiai rinktis tipus ir juos apsirašinėti.

2.4.1 AKS algoritmas

AKS [1] algoritmo pagrindinė idėja yra pirminio skaičiaus testo Fermat mažosios teoremos apibendrinimas polinomams: teigiamas sveikasis skaičius $n > 1$ yra pirminis tada ir tik tada, kai

$$(x + 1)^n = x^n + 1 \tag{7}$$

polinomų su koeficientais iš Z/nZ žiede. Pavyzdžiui, Carmichael'o skaičiui $n = 561$ gauname tokį polinomą

$$(x + 1)^{561} = x^{561} + ... + 51x^{11} + ... + 1 \bmod 561.$$

Sąryšio (1) įrodymas yra nesudėtingas ir remiasi Fermat mažąją teorema bei binominių koeficientų dalumo savybėmis. Deja, šis kriterijus neduoda polinominio laiko pirminio skaičiaus testo, nes tikrinant ar skaičius n yra pirminis reikėtų apskaičiuoti maždaug n polinomo, esančio kairėje sąryšio (1) pusėje, koeficientų. Agrawal'is ir jo studentai pasiūlė sąlygą (1) pakeisti silpnesniais reikalavimais

$$(x-a)^n = x^n - a \pmod{n, x^r - 1}, \tag{8}$$

kur a turi būti mažomis likinių klasėmis moduliu n, o r yra mažas sveikasis skaičius. Tačiau, užtikrinimui, kad perėjimas nuo polinominio lyginio (1) prie lyginių sekos (2) vis dar duoda pirminio skaičiaus testą a ir r turi „perbėgti“ gana nemažai reikšmių. Iš kitos pusės, šie lyginiai gali būti patikrinti daug greičiau, nei (1), nes pakanka atlikti skaičiavimus su polinomais, kurių laipsnis $\leq 2r$. Tinkamas balansas duoda polinominio laiko determinantinį pirminio skaičiaus testą.

Teorema (Agrawal, Kayal, Saxena) Tegul s, n yra teigiami sveikieji skaičiai. Tarkime, kad q ir r yra tokie pirminiai skaičiai, kad q dalija $r-1$, $n^{(r-1)/q} \not\equiv 0, 1 \bmod r$, ir

$$\binom{q+s-1}{s} \geq n^{2[\sqrt{r}]}.$$

Jei visiems $1 \leq a < s$, a ir n tarpusavyje pirminiai skaičiai, yra patenkinama sąlyga (2), tada n yra pirminio skaičiaus laipsnis.

Šios teoremos įrodymą galima rasti [1].

Iš šios (Agrawal, Kayal, Saxena) teoremos seka AKS algoritmas:

Įvestis: sveikas skaičius $n > 1$

1. jei (n yra pavidalo a^b, $b > 1$) išvestis SUDĖTINIS;
2. $r = 2$;
3. kol ($r < n$)
    4. jei (DBD$(n, r) \neq 1$) išvestis SUDĖTINIS;
    5. jei (r pirminis)
        6. tegul q yra didžiausias skaičiaus $r-1$ pirminis daliklis;
        7. jei ($q \geq 4\sqrt{r}\log_2 n$) ir ($n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$) — nutraukti;
    8. $r \leftarrow r + 1$;
9. a kinta nuo 1 iki $2\sqrt{r}\log_2 n$
    10. jei ($(x-a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$) išvestis SUDĖTINIS;
11. išvestis PIRMINIS;

Vienas iš uždavinių, pateikiamų praktikumo metu, galėtų būti AKS algoritmo pradinės versijos realizacija naudojant biblioteką LiDIA. Algoritmo pseudokodas palengvina šią užduotį. Belieka jį transformuoti į C++ kalbą. Kai kurios matematinės funkcijos jau yra realizuotos bibliotekoje LiDIA nebereikia programuoti iš naujonaujo. Pavyzdžiui, didžiausio bendro daliklio radimas, nustatymas, ar duotas skaičius yra pirminio skaičiaus laipsnis. Smulkiau aptarsime tik sudėtingesnius žingsnius. Vienas iš tokių žingsnių — duoto skaičiaus didžiausio pirminio daliklio radimas. Tam tikslui mes sukūrėme atskirą funkciją:

```
bigint DidziausiasPirminisDaliklis(const bigint & n)
{
  bigint r, p, i;
  if (n<2){
    return 1;
  }
  else{
    r = n;
    if (r % 2 == 0){
      p = 2;
      do{
        r /= 2;
      }while (r%2==0);
    }
    for (i=3; i<=r; i += 2){
      if (r%i==0){
        p=i;
        do{r/=i;}while (r%i==0);
      }
    }
    return p;
  }
```

Daugiausiai sunkumų sukelia 10 žingsnis. Prieš lygindami polinomus turime juos sukonstruoti. Pasinaudosime LiDIA paketo *polynomial* funkcijomis. Jų aprašymą galite rasti priede. Pradedame nuo kairės sąryšio pusės:

```
kaire1.assign_x();
      kaire2.assign(a);
      subtract(kaire3, kaire1, kaire2);
```

Analogiškai sukonstruojame dešinėje pusėje esantį polinomą:

```
desine1.assign_x();
desine2.assign(a);
power(desine1, desine1, n);
subtract(desine3, desine1, desine2);
```

bei polinomą, kuriuo redukuojame:

```
daliklis1.assign_x();
power(daliklis1, daliklis1, r);
daliklis2.assign(1);
subtract(daliklis, daliklis1, daliklis2);
```

Redukciją galime atlikti pasinaudoję funkcija `power_mod` bei `reminder` atitinkamai:

```
power_mod(kaire, kaire3, n, daliklis);
cout <<"kaires liekana: "<< kaire<<"\n";
n_poli.assign(n);
remainder(desine, desine3, daliklis);
```

Norėdami atlikti koeficientų redukciją moduliu n sukūrėme atskirą ciklą, nes tinkamų funkcijų bibliotekoje LiDIA neradome. Koeficientai ir jų redukcijos rezultatai saugomi masyvuose:

```
bigint* koeficientaik;
bigint koefskaiciusk;
koefskaiciusk = kaire.degree();
koeficientaik = kaire.get_data();
vector<bigint> liekanosk;
for (int i=0; i<=koefskaiciusk; i++){
   bigint l;
   remainder(l, koeficientaik[i], n);
   liekanosk.push_back(l);
   cout << "koeficientas " << i << ": " << koeficientaik[i]
     << ", liekana: " << liekanosk[i] << endl;
}
```

Analogišką procedūrą atliekama ir su dešinėje tikrinamo sąryšio pusėje esančiu polinomu

AKS algoritmo daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA, programos kodą žiūrėti kompaktinio disko kataloge „aks“

2.4.2 Miller'io testas

1976 metais Miller'is paskelbė pirmąjį modernųjį pirminio skaičiaus testą. Jis teigė, kad, esant teisingai išplėstinei Rymano hipotezei, pirminio skaičiaus testas gali būti atliktas polinominiame laike. Miller'io algoritmas paremtas mažąja Fermat teorema, kuri teigia, kad jei teigiamam sveikąjam skaičiui n galime rasti sveikąjį skaičių a tokį, kad

$$1 < a < n \tag{9}$$

ir

$$a^{n-1} \not\equiv 1 \pmod{n} \tag{10}$$

tada mes įrodysime, kad n yra sudėtinis. Šis metodas turi du pagrindinius trūkumus:

- testuojant visus galimus a, skaičiavimai „išeis“ už polinominio laiko ribų. Nors mums ir nebūtina testuoti visų a mažesnių už n (kadangi pusė iš jų yra priešingi kitos pusės elementams moduliu n), bet testavimas iki $n/2$ lyginių taip pat neduoda reikiamų rezultatų.

- Carmichaelo skaičiais vadiname tokius sudėtinius skaičius n, kuriems galioja sąryšis

$$a^{n-1} \equiv 1 \pmod{n} \tag{11}$$

su visais a, $DBD(a, n) = 1$. Kitaip tariant, netgi jei galėtume patikrinti visus a, mes nebūsime tikri, kad n yra pirminis.DBD — tai didžiausias bendras daliklis.

Papildomų sąlygų įvedimas mažojoje Fermat teoremoje padeda išvengti minėtų trūkumų. Miller'is pasiūlė naudoti tokį algoritmą, kuris nustato, ar duotas skaičius n yra pirminis (tariant, kad išplėstinė Riemann'o hipotezė teisinga).

Įvestis: n — nelyginis teigiamas sveikas skaičius, $f(n)$ yra suskaičiuojama funkcija aibėje $\mathbb{N}$.

1. jei $n = a^b$, kai $a, b \in \mathbb{Z}$ ir b>1, tai išvestis SUDĖTINIS;
2. a kinta nuo 2 iki $f(n)$
   3. jei a/n, tai išvestis SUDĖTINIS;
   4. jei $a^{n-1} \not\equiv 1 \pmod{n}$, tai išvestis SUDĖTINIS;
   5. jei DBD$((a^{n-1/2k} - 1 \pmod{n}), n) \neq 1$ arba n kuriam nors k, kuris kinta tarp 1 ir didžiausio dvejeto laipsnio, dalijančio $n - 1$, tai išvestis SUDĖTINIS;
6. išvestis PIRMINIS;

Daugiau informacijos apie pirminio skaičiaus testus galima rasti [6].

Uždaviniai savarankiškam darbui.

1. Parašykite programą, randančią visus pirminius skaičius pagrindu a.

2. Remiantis aukščiau pateiktu pseudokodu realizuokite Miller'io pirminio skaičiaus testą. Palyginkite su AKS algoritmų.

3. AKS algoritmo patobulinimai ir susijusios realizacijos yra aptariami, pavyzdžiui, internete adresu `http://fatphil.org/maths/AKS/`. Išnagrinėkite siūlomus patobulinimus ir naudojantis biblioteka LiDIA realizuokite naujausią AKS variantą.

2.5 Faktorizacijos algoritmai

RSA kriptosistemos saugumas glaudžiai susijęs su sunkumais iškylančiais faktorizuojant sveikuosius teigiamus skaičius į pirminius skaičius. Tačiau nėra žinoma ar tikrai taip sudėtinga išspręsti šias faktorizavimo problemas. Per paskutinius keletą metų atrasti daug efektyvesni sveikųjų skaičių faktorizavimo algoritmai. Saugaus RSA modulio dydis išaugo nuo 512 iki 1024 bitų.

Šiame skyriuje aprašysime keletą svarbių faktorizavimo algoritmų. Platesnė apžvalga pateikiama [6]. Tegul n yra teigiamas sveikasis skaičius, ir yra žinoma, kad jis sudėtinis. Tai gali būti nustatyta Fermat testo pagalba, arba Millerio - Rabino testo pagalba. Tačiau šie testai nenustato n daliklio. Čia aprašyti algoritmai realizuoti LiDIA bibliotekoje.

2.5.1 Pollard'o $(p-1)$ metodas

Egzistuoja faktorizavimo algoritmai, kurie veikia gerai sudėtiniams sveikiesiems skaičiams su tam tikromis savybėmis. Reikia vengti tokių skaičių RSA ar Rabino modulio algoritmuose. Kaip tokio faktorizavimo algoritmo pavyzdį, aprašysime John Pollard metodą $p-1$.

$(p-1)$ metodas geriausiai veikia su sudėtiniais sveikaisiais skaičiais su pirminiu faktoriumi p tokiu, kad $p-1$ turi tik mažus pirminius daliklius. Tuomet yra įmanoma nustatyti $p-1$ daliklį k nežinant $p-1$, kaip mažų sveikųjų skaičių šaknų produktą. Detalės aprašytos toliau. Ferma mažoji teorema numato

$$a^k \equiv 1 \bmod p$$

visiems sveikiesiems a kurie nėra dalūs iš p. Tai reiškia, kad p dalo $a^k - 1$. Jei $a^k - 1$ nėra dalus iš n, tuomet $\gcd(a^k - 1, n)$ yra tinkamas n daliklis, taigi n faktorius rastas.

Kaip k kandidatus, $p-1$ metodas naudoja visų pirminių skaičių, neviršijančių žemiau duotosios B ribsandaugąaugą; būtent

$$k = \prod_{q \in \mathbb{P},\ q^e \leq B} q^e.$$

Jei visos pirminės šaknys, kurios dalija $p-1$ yra mažesnės už B, tuomet k yra $p-1$ daliklis. Algoritmas apskaičiuoja $g = \gcd(a^k - 1, n)$ bazei a. Jei n daliklis nerastas, tuomet naudojama nauja riba B.

2.5.2 Pell'io konikių faktorizavimo algoritmas

Vienas iš svarbiausių faktorizavimo metodų yra Lenstros elipsinių kreivių faktorizacijos metodas [14]. Elipsinių kreivių teorija yra gana sudėtinga, todėl apsiribosime šio

algoritmo versija paprastesnėms algebrinėms kreivėms, turinčiomis analogiškų savybių. Nagrinėsime kreives pavidalo $C : X^2 - Dy^2 = 1$, apibrėžtas virš baigtinio kūno Z_p. Šios kreivės taškų aibėje galima apibrėžti sudėties operacija tokiu būdu:

$$(x_1, y_1) + (x_2, y_2) = (x_1x_2 + Dy_1y_2, x_1y_2 + x_2y_1).$$

Kreivės C taškai formuoja grupę, aukščiau apibrėžtos sudėties operacijos atžvilgiu. Neutraliojo elemento vaidmenį atlieka taškas $(1, 0)$. Tarkime norime faktorizuoti skaičių N. Tuomet atliekame tokius žingsnius (reikia transformuoti į pseudokodą:

1. Parinkti bekvadratį sveikąjį skaičių D ir tašką $P \neq (1, 0)$ atitinkamoje kreivėje, apibrėžtoje formule: $C : X^2 - DY^2 = 1 \bmod N$.
2. Parinkite teigiamą sveikąjį skaičių B ir apskaičiuokite $(B!)P = (x, y)$ kreivėje C. Čia $kP = P + P + ... + P$ k kartų.
3. Jei y nedalus iš N grąžinti $DBD(x - 1, N)$ arba $DBD(y, N)$. Kitu atveju padidinkite B arba sugeneruokite naują kreivę ir atitinkamą tašką P.

Praktinių užsiėmimų metu studentams būtų galima pasiūlyti atlikti šio algoritmo realizaciją bibliotekos LiDIA pagalba ir ištirti jo efektyvumą. Mes siūlome vieną realizacijos variantą. Pradžioje generuosime pasirinktos kreivės tašką:

```
bigint sumax (const bigint & x1, const bigint & y1,
            const bigint & x2, const bigint & y2){
  return (x1 * x2 + D * y1 * y2) % n;
}
bigint sumay (const bigint & x1, const bigint & y1,
            const bigint & x2, const bigint & y2){
  return (x1 * y2 + y1 * x2) % n;
}
```

Po to apibrėžiame taškų sudėtį ir daugybą iš skaliaro atitinkamai:

```
void dalikliai (bigint& daliklis1, bigint& daliklis2,
        const bigint& taskas1, const bigint& taskas2){
  bigint t, B, daugiklis;
  B=3;
  daugiklis = faktorialas(B);
  bigint taskasx, taskasy;
  taskasx=taskas1;
```

```
  taskasy=taskas2;
  bigint i, xx1, xx2, yy1, yy2;

  xx1=taskasx; xx2=taskasx; yy1=taskasy; yy2=taskasy;
  for (i=2; i<=daugiklis; i++){
    taskasx = sumax(xx1, yy1, xx2, yy2);
    taskasy = sumay(xx1, yy1, xx2, yy2);
    xx2=taskasx; yy2=taskasy;
  }
  daliklis1=gcd(taskasx-1, n);
  daliklis2=gcd(taskasy,n);
}//dalikliai
```

Realizuodami šį algoritmą mes pasirinkome tokią strategiją - užfiksuojame daugiklį $B = 3$ ir varijuojame kreivės taškus. Pilną algoritmo kodą galima rasti kompaktinio disko kataloge `faktorizavimas.cpp`.

Uždaviniai savarankiškam darbui.

1. Realizuokite Pollard'o faktorizacijos algoritmą, naudojantis biblioteka LiDIA. Įvertinkite jo efektyvumą.

2. Teorinėje dalyje nagrinėtų Pell'io Konikių taškų daugybą iš skaliaro galima apskaičiuoti atliekant nuoseklų dvigubinimą. Pavyzdžiui, norėdami apskaičiuoti $7P$ surandame $2P = P + P$, $4P = 2P + 2P$, tuomet $7P = 4P + 2P + P$, Tai efektyviau, negu šešis kartus atlikti sudėties operaciją. Remdamiesi šią idėją, pagreitinkite Pell'io Konikių faktorizacijos algoritmus.

3. Realizuokite Pell'io Konikių faktorizacijos algoritmo variantus kai didinamas daugiklis B. Palyginkite su pateikta teorinėje dalyje algoritmo versija.

3 Darbo eiga

3.1 Įrankių analizė

Viskas ko reikia kuriant: algoritmus sveikųjų skaičių faktorizacijoms, algoritmus nustatančius diskrečius logaritmus baigtiniuose kūnuose, skaičiuojantys taškus ant elipsinių kreivių virš baigtinių kūnų ir panašiai, sudėti į vieną biblioteką, kuri buvo pavadinta LiDIA.

Matematinių algoritmų realizavimui tinka objektinio programavimo kalba. LiDIA realizacijai buvo nuspręsta naudoti C++. Tam, kad užtikrinti daugiakarčio tikslumo aritmetikos bibliotekos LiDIA portabilumą, naudojamas labai mažas, nuo platformos priklausomas, branduolys. Šis branduolys sudarytas iš daugiakarčio tikslumo aritmetikos modulių ir atminties tvarkyklės. Visi kiti daugiakarčio tikslumo aritmetikos bibliotekos LiDIA objektai realizuoti naudojant C++ ir sukompiliuoti skirtingais kompiliatoriais, naudojamais skirtingose platformose.

Daugiakarčio tikslumo aritmetikos bibliotekai LiDIA sukompiliuoti reikalinga UNIX tipo operacinė sistema. Darbą atlikti pasirinkau Linux OS su programavimo įrankiu KDevelop ir GNU C++ kompiliatoriumi. KDevelop yra programavimo įrankis skirtas daugeliui programavimo kalbų, tokių kaip C, C++, Fortran, Java, Perl, PHP ir kitos. Rašant programas Borland C++ kompiliatoriumi, tai tinkamiausias pasirinkimas, mano nuomone, programavimo įrankis Borland C++ Builder x. Mano nuomone, naudojant GNU C++ kompiliatorių vienas iš geriausių pasirinkimų Linux OS, programavimo įrankis KDevelop. Linux OS aplinkoje galima naudoti programavimo įrankius Emacs arba Vim, tačiau jie tinkamesni profesionaliems (patyrusiems) programuotojams. Kadangi biblioteka LiDIA kompiliuojasi su kompiliatoriumi GNU C++, tai pasirinkau programavimo įrankį KDevelop. Norėdama susikompiliuoti ir įsidiegti daugiakarčio tikslumo aritmetikos biblioteką LiDIA pirmiausia atsisiunčiau naujausius išeities tekstus iš `ftp://ftp.informatik.tu-darmstadt.de/TI/systems/LiDIA`

Daugiakarčio tikslumo aritmetikos biblioteka LiDIA aprašyta virtualiame kriptografijos kurse adresu http://mokymai.dist.su.lt/. Virtualus kriptografijos kursas sukurtas Moodle aplinkoje. Kiekvienam kurso moduliui yra parengta teorinė medžiaga, savarankiškam darbui skirti uždaviniai, uždavinių sprendimo pavyzdžiai, pagrindinių sąvokų žodynėlis. Studentai gali bandyti paskaitas virtualioje Moodle aplinkoje neribotą kartą skaičių bei jiems patogiu laiku. Daugiakarčio tikslumo aritmetikos biblioteka LiDIA veikia visose POSIX standartu suderinamose sistemose.

3.2 Daugiakarčio tikslumo bibliotekų palyginimas

Šiame skyriuje palyginsime įvairias daugiakarčio tikslumo bibliotekas. Daugelis šių bibliotekų yra laisvai prieinamos ne komerciniam naudojimui. Skaičių teorijai yra sukurta labai daug bibliotekų, tačiau daugelis jų yra ne intuityvios ir nedraugiškos vartotojui.

Aprašysime bibliotekas optimizuotas darbui virš ribotų kūnų, daugiausiai $\mathbb{Z}/n\mathbb{Z}$, GF(2^n) arba GF(p^n). ZEN vienintelė biblioteka, kuri veikia efektyviai bet kokiame polinominiame baigtinių kūnų plėtinyje.

- **NTL** — tai galinga, portabili C++ biblioteka, pateikianti duomenų struktūras ir algoritmus sveikųjų skaičių, vektorių, matricų ir polinomų manipuliavimams virš baigtinių kūnų. Pagrindinis **NTL** bibliotekos privalumas, tai greitis. Ji gali būti naudojama kartu su **GMP** (GNU daugiakarčio tikslumo biblioteka), UNIX platformoje. **WinNTL** distributyvas gali būti naudojamas su bet kuria Windows versija (aukštesne už Windows 3.11). **NTL** yra laisva programinė įranga platinama pagal GNU bendrąją viešąją licenziją. **NTL** polinominė aritmetika, viena greičiausių ir buvo panaudota pasiekiant keletą „pasaulio rekordų“ polinominėje faktorizacijoje ir ieškant elipsinių kreivių tvarkos. Taip pat ši biblioteka buvo panaudota „nulaužiant“ keletą kriptosistemų.

- **LiDIA** — tai C++ biblioteka naudojama skaičiuojamojoje skaičių teorijoje. Ši biblioteka pateikia optimizuotas įvairaus duomenų tipo daugiakarčio tikslumo laikui imlių algoritmų realizacijas.

  – sveikųjų, realiųjų, racionaliųjų ir kompleksinių skaičių aritmetika;
  – aritmetika virš $\mathbb{Z}/n\mathbb{Z}$, GF(2^n) ir GF(p^n);
  – sveikoji faktorizacija;
  – polinomų faktorizacija virš baigtinių kūnų.;
  – ir kt.

  Daugiakarčio tikslumo aritmetikos biblioteką LiDIA kuria LiDIA grupė Darmstadt technologijų universitete. Ši biblioteka taip pat yra laisvos prieigos programinė įranga. LiDIA veikia visose su POSIX standartu suderinamose sistemose. Windows palaikymas planuojamas ateityje.

- **MIRACL** – tai didelių skaičių biblioteka, kuri realizuoja visus algoritmus, būtinus didelių skaičių kriptografijoje. **MIRACL** yra kompaktiškas greitas ir efektyvus. Tai C biblioteka, realizuojanti pilną daugiakarčio tikslumo aritmetiką. Biblioteka naudojama viešo rakto kriptografijoje ir skaitmeninio parašo realizavimui. **MIRACL** yra nemokamas nekomerciniams tikslams. Ji gali būti naudojama Windows NT/95/98/2000/XP ir Linux platformose.

- **GP/PARI** — tai skaičių teorijos kalkuliatorius. Naudojami tipai, tai sveikieji skaičiai (iki 300 000 skaitmenų), realieji skaičiai (atitinkamai realiajam tikslumui), $\mathbb{Z}/n\mathbb{Z}$ elementai, racionalieji skaičiai, kompleksiniai skaičiai, polinomai ir kt. **GP/PARI** veikia MS DOS, OS/2, UNIX (tame tarpe ir Linux), Mac sistemose. Išeities tekstai laisvai prieinami. Šia biblioteka naudotis sunkiau, nei aukščiau išvardintomis. Ji yra simbolinė, objektai laikomi ne kaip abstraktūs duomenų tipai.

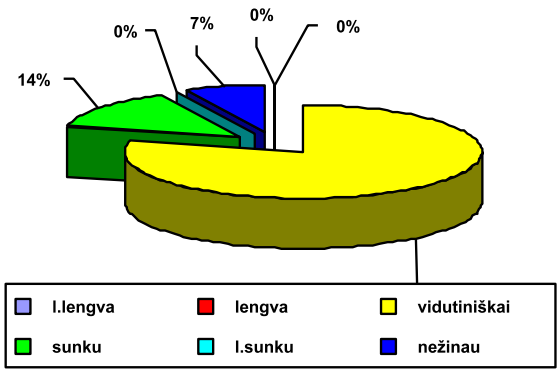
Pasirinkta daugiakarčio tikslumo aritmetikos biblioteka LiDIA, nes išstudijavusi kriptografijos kursą pastebėjau, kad trūksta medžiagos reikalingos kriptografijos kurse, nagrinėjamų algoritmų realizacijai. Kadangi LiDIA turi plačiausią realizuotų tipų ir algoritmų spektrą, ji buvo pasirinkta kaip šio magistrinio darbo įrankis. Kitas bibliotekas drauge su daugiakarčio tikslumo aritmetikos biblioteka LiDIA galima naudoti be jokių papildomų veiksmų. Daugiakarčio tikslumo aritmetikos biblioteka LiDIA gali būti naudojama nemokamai nekomerciniais tikslais.

3.3 Anketinės apklausos analizė

Siekiant išanalizuoti ar reikalingas virtualus kriptografijos kursas, buvo panaudotas tyrimo metodas — studentų anketinė apklausa.

Tyrime dalyvavo 20 studentų magistrantų, kurie turėjo atsakyti į 11 anketos klausimų (anketos pavyzdys pateikiamas priede).

Respondentų atsakymai buvo susisteminti, o gauti rezultatai pateikiami diagramose. Į klausimą: Jūsų nuomone, studijuoti kriptografijos kursą labai lengva, lengva, vidutiniškai, sunku, labai sunku, nežinau, gauti tokie rezultatai (žr. 1 pav.).

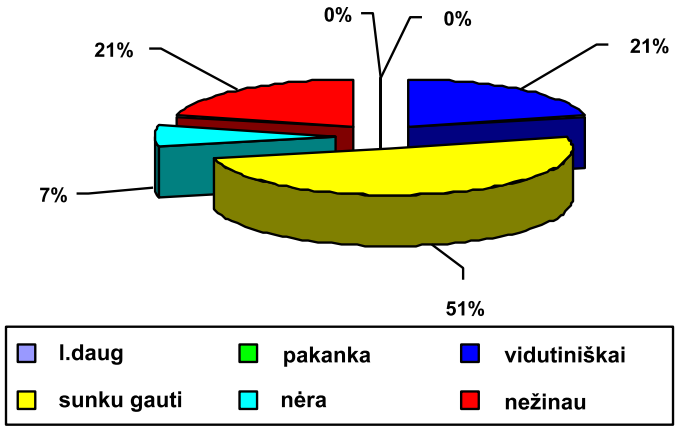


1 pav.: Studijuoti kriptografijos kursą

Iš diagramos matyti, kad nėra apklausoje dalyvavusių tiriamųjų, kurie galvoja, kad studijuoti kriptografijos kursą yra labai lengva, lengva ar labai sunku. Net 79% studentų mano, jog studijuoti kriptografiją yra vidutiniškai, o 14% — sunku. Taigi galima daryti išvadą, kad daugumai studijuoti kriptografijos kursą yra vidutiniškai.

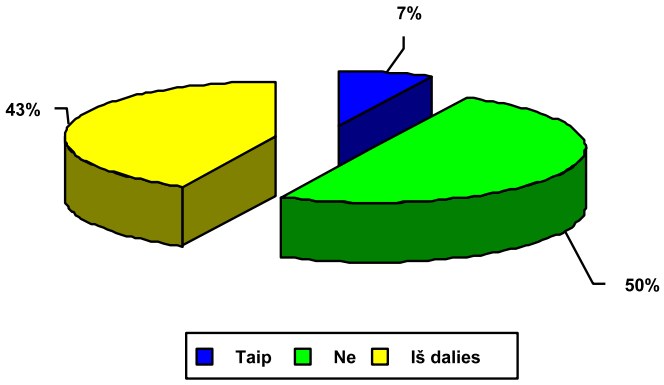
Norėdama išsiaiškinti ar medžiagos lietuvių kalba pakanka savarankiškoms kriptografijos studijoms pateikiau sekantį klausimą. Į antrą klausimą: kaip manote, ar medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms yra labai daug, pakanka, vidutiniškai, sunku gauti, nėra ar nežinote, gauti tokie rezultatai (žr. 2 pav. 27 puslapyje).

Į pateiktą klausimą didžioji dalis respondentų atsakė jog medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms sunku gauti (51%), 21% nežino ar pakanka medžiagos studijoms, tiek pat apklausoje dalyvavusių studentų (21%) pažymėjo, kad yra



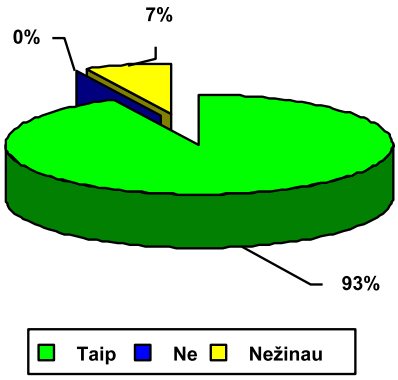
2 pav.: Pakanka medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms

vidutiniškai lietuviškos literatūros. 7%. respondentų mano, jog nėra medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms. Nei vienas tiriamasis neatsakė, kad yra labai daug ar pakanka medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms. Iš šios diagramos galima spręsti, kad trūksta medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms. Į klausimą: kaip manote, ar yra pakankamai pavyzdžių kriptografijos uždaviniams spręsti, gauti tokie anketinės apklausos rezultatai (žr. 3 pav. 27 puslapyje).



3 pav.: Pakanka pavyzdžių kriptografijos uždaviniams spręsti

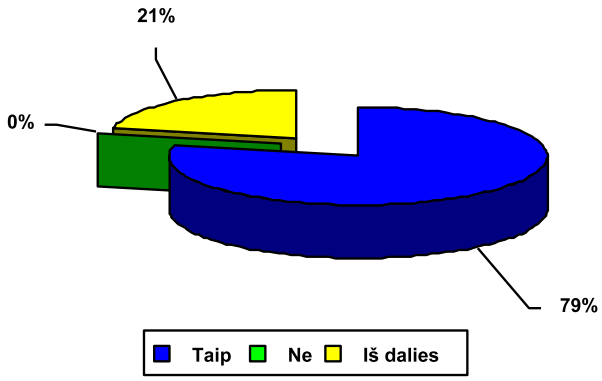
Iš diagramos matyti, jog pusė apklaustųjų (50%) mano, kad nepakanka pavyzdžių kriptografijos uždaviniams spręsti, o 43% — iš dalies ir tik 7% atsakė, kad taip pakanka pavyzdžių kriptografijos uždaviniams spręsti. Iš šių duomenų galima daryti išvadą, jog daugumai nepakanka arba iš dalies pakanka pavyzdžių kriptografijos uždaviniams spręsti. Šiuo metu studijuoti galima įvairiai. Vienas iš studijavimo būdų yra virtualus kursas. Į klausimą: ar studijuojant kriptografijos kursą, studijas palengvintų virtualus kriptografijos kursas; gauti tokie respondentų atsakymai (žr. 4 pav. 28 puslapyje).



4 pav.: Palengvintų studijas virtualus kriptografijos kursas

Iš diagramos matyti, kad net 93% apklaustųjų mano, kad virtualus kriptografijos kursas palengvintų studijas, o 7% atsakė nežinantys. Iš šių rezultatų matyti, kad daugumai respondentų virtualus kriptografijos kursas palengvintų studijas.

Į sekantį klausimą: ar studijuodami kriptografijos kursą naudotumėtės virtualiu kriptografijos kursu, jei toks būtų; tiriamieji atsakė taip (žr. 5 pav.).

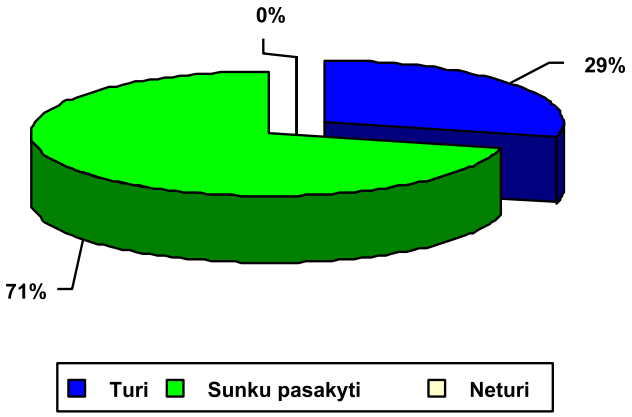


5 pav.: Naudotųsi virtualiu kriptografijos kursu

Iš šios diagramos matoma, kad 79% respondentų teigia, jog naudotųsi virtualiu kriptografijos kursu, o 21% — iš dalies naudotųsi, o nebuvo tokių studentų, kurie nesinaudotų virtualiu kriptografijos kursu. Taigi galima teigti, kad didžioji dalis apklausoje dalyvavusių žmonių naudotųsi virtualiu kriptografijos kursu.

Pateiktas klausimas: kaip manote, ar studijuojant kriptografiją reikia mokėti programuoti? Gauti tokie respondentų atsakymai (žr. 6 pav. 29 puslapyje).

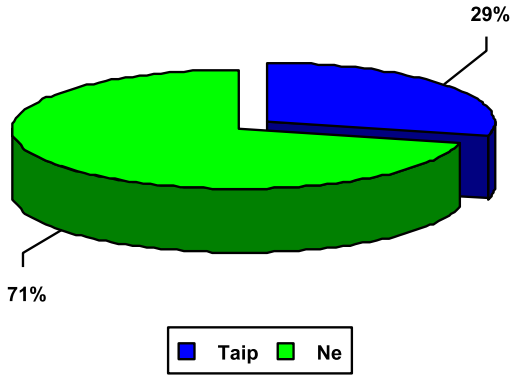
Iš diagramos duomenų matyti, kad didžiajai daliai respondentų (71%) sunku pasakyti ar studijuojant kriptografijos kursą reikia mokėti programuoti ir tik 29% — atsakė teigiamai. Nebuvo tokių kurie pažymėtų, kad studijuojant kriptografijos kursą nereikia



6 pav.: Ar reikia mokėti programuoti studijuojant kriptografiją?

mokėti programuoti. Taigi galima daryti išvadą, kad dauguma studentų nežino ar reikia mokėti programuoti studijuojant kriptografijos kursą.

Į septintą klausimą: ar mokate programuoti C, C++ ar kitomis kalbomis, gauti tokie apklausoje dalyvavusių asmenų atsakymai (žr. 7 pav.)

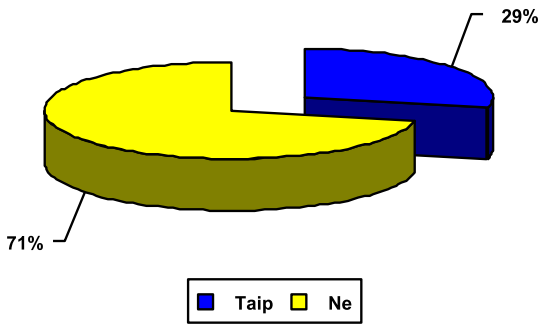


7 pav.: Moka programuoti

71% respondentų nemoka programuoti C, C++ ar kitomis programavimo kalbomis, o 29% — atsakė teigiamai. Tai reiškia, kad dauguma apklausoje dalyvavusių tiriamųjų nemoka programuoti C, C++ ar kitomis programavimo kalbomis.

Į aštuntą klausimą: ar teko rašyti programas su labai dideliais skaičiais (pvz.: 10^6), gauti tokie rezultatai (žr. 8 pav. 30 puslapyje).

Į šį klausimą 71% apklaustųjų atsakė neigiamai, 29% studentų pažymėjo, kad yra tekę rašyti programas su labai dideliais skaičiais (pvz.: 10^6). Taigi galima spręsti, kad didžioji dalis tiriamųjų nėra rašę programų su labai dideliais skaičiais. Į sekantį klausimą: ar teko rašyti programas su labai dideliais skaičiais, ar „užlūždavo“ programa? Turėjo atsakyti tie respondentai, kurie į aštuntą klausimą atsakė teigiamai. 40% atsakiusiųjų



8 pav.: Rašė programas su labai dideliais skaičiais

pažymėjo, kad rašant programas su dideliais skaičiais jiems programa neužlūždavo, o 60% — pritarė. Išvada: daugiau nei puse tiriamųjų pažymėjo, kad rašant programas su labai dideliais skaičiais jiems programa „užlūždavo“.

Į atvirą dešimtą klausimą: parašykite siūlymus, kaip reiktų išspręsti problemas su kuriomis teko susidurti studijuojant kriptografiją, gauti tokie apklausoje dalyvavusių studentų pasiūlymai:

- sukurti virtualų kriptografijos kursą, kuriame būtų patalpinta daugiau teorinės medžiagos lietuvių kalba ir uždavinių pavyzdžių;
- susisteminti teoriją, teoriją išdėstyti nuosekliai, visiems terminams parašyti apibrėžimą;
- daugiau lietuviškos literatūros.

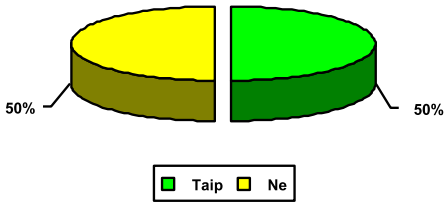
Galime daryti išvadą, kad trūksta medžiagos lietuvių kalba savarankiškoms kriptografijos studijoms ir reikalingas virtualus kriptografijos kursas, kuriame būtų patalpinta daugiau teorinės medžiagos lietuvių kalba ir uždavinių pavyzdžių bei terminams parašyti apibrėžimai.

Pateikiau sekantį klausimą: ar teko rašyti programas naudojantis kokia nors matematine sistema (Matchad, Matematika ar kt.), apklausoje dalyvavę respondentai atsakė (žr. 9 pav. 31 puslapyje).

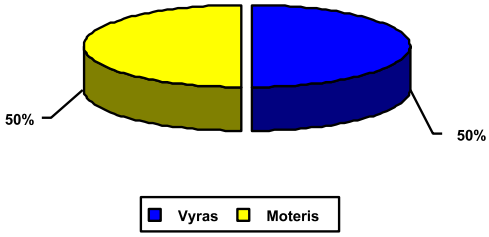
Taigi matoma, kad 50% apklausoje dalyvavusių studentų pažymėjo, jog teko rašyti programas naudojantis kokia nors matematine sistema (Matchad, Matematika ar kt.) ir tiek pat (50%) — neteko.

Norėdama sužinoti šiek tiek informacijos apie tiriamuosius pateikiau klausimus apie lytį, amžių. Į klausimą: kokia Jūsų lytis, gauti tokie respondentų atsakymai (žr. 10 pav. 31 puslapyje).

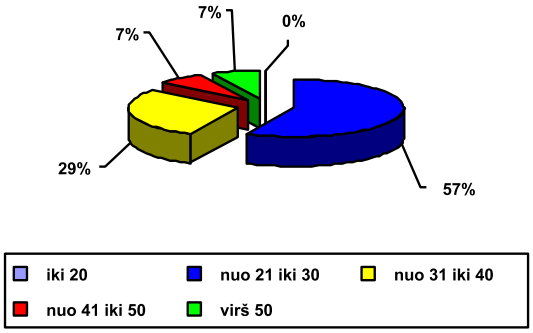
Iš diagramos matyti, kad tyrime dalyvavo 50% vyrų ir 50% moterų.



9 pav.: Rašė programas naudojantis kokia nors matematine sistema



10 pav.: Anketinėje apklausoje dalyvavusių tiriamųjų lytis



11 pav.: Respondentų amžius

Į pateiktą klausimą apie savo amžių tiriamieji taip atsakė (žr. 11 pav. 31 puslapyje).

Dauguma respondentų (57%) yra nuo 21 iki 30 metų, 29% nuo 31 iki 40 metų amžiaus ir 7% nuo 41 iki 50 metų ir virš 50 metų. Iki 20 metų. Virš 50 metų apklausoje dalyvavo taip pat 7 procentai. Iki 20 metų amžiaus apklausoje nedalyvavo. Taigi galime daryti išvadą, jog apklausoje dalyvavo įvairaus amžiaus tiriamieji, daugumos amžius nuo 21 iki 30 metų.

Dėl anketinių duomenų patikimumo neabejojama, nes anketa buvo anoniminė.

3.4 Iškilusios problemos darbe

Daugiakarčio tikslumo bibliotekoje LiDIA dažnai atliekami veiksmai su skirtingų tipų skaičiais. Viena iš esminių problemų buvo tipų keitimas. Toje pačioje programoje kartais neužtekdavo vieno tipo `bigint` jį reikėdavo keisti kitu pvz. `bigmod`. Norint atlikti veiksmus su skirtingais tipais, tenka kintamuosius konvertuoti į vienodo tipo kintamuosius. Norint pakeisti tipus reiktų prieš priskiriant `bigint` tipo kintamojo reikšmę kompiuterio tipo kintamajam (pvz.: `int`), atlikti testą, kuris patikrins ar toks priskyrimas gali būti atliktas be perpildymo. Platesnis aprašymas vartotojo vadove.

Realizuojant AKS algoritmą reikėjo vykdyti skaičiavimus su polinomais su koeficientais iš likinių klasių moduliu N žiedo Z_N, kur N – sudėtinis skaičius. Bibliotekoje LiDIA yra aprašyta tik klasė `Fp_polynomial`, skirta darbui su polinomais virš baigtinių kūnų Z_p, kur p – pirminis skaičius. Šiuo atveju polinomo koeficientai redukuojami pirminiu moduliu p. Esant tokiam apribojimui prireikė aprašyti atskirą procedūrą, kurios metu nagrinėjamų polinomų koeficientai buvo redukuojami reikiamu moduliu N.

Atliekant savo darbą pirmiausiai reikėjo išmokti naudotis sukompiliuota daugiakarčio tikslumo aritmetikos biblioteka LiDIA. Šios bibliotekos funkcijų aprašymų lietuvių kalboje nėra, teko ilgokai aiškintis teorinę dalį apie daugiakarčio tikslumo aritmetikos biblioteką LiDIA.

4 Išvados

1. Surinkau ir susisteminau medžiagą, kurią galima būtų dėstyti kriptografijos kurse bei sukūriau virtualų kriptografijos kursą.

2. Parinkti uždaviniai (RSA schemos, pirminio skaičiaus testų, faktorizacijos algoritmų) kriptografijos praktikumui.

3. Realizuotos pagrindinės daugiakarčio tikslumo aritmetikos bibliotekos LiDIA funkcijos skirtos algoritmų (RSA schemos, pirminio skaičiaus testų, faktorizacijos algoritmų) realizacijai.

4. Sprendžiant uždavinius su daugiakarčio tikslumo aritmetikos biblioteka LiDIA problema yra tipų keitimas. Norint atlikti veiksmus su skirtingais tipais, tenka kintamuosius konvertuoti į vienodo tipo kintamuosius. Kita problema - veiksmai su polinomais. Realizuojant AKS algoritmą reikėjo vykdyti skaičiavimus su polinomais su koeficientais iš likinių klasių moduliu N žiedo Z_N, kur N – sudėtinis skaičius. Bibliotekoje LiDIA yra aprašyta tik klasė `Fp_polynomial`, skirta darbui su polinomais virš baigtinių kūnų Z_p, kur p – pirminis skaičius. Šiuo atveju polinomo koeficientai redukuojami pirminiu moduliu p. Esant tokiam apribojimui prireikė aprašyti atskirą procedūrą, kurios metu nagrinėjamų polinomų koeficientai buvo redukuojami reikiamu moduliu N.

Literatūra

[1] M. Agrawal, N. Kayal, N. Saxena Primes is in P, http://www.cse.iitk.ac.in/news/primality.html

[2] T. Aoyama. Polynomial Time Primality Testing Algorithm. http://www.cs.rit.edu/axt9690/ms_project/ms_project.html

[3] K. Biliūnas. Apie keletą Debian/Linux paketų. http://kebil.ghost.lt

[4] J. A. Buchmann. (2002). Introduction to Cryptography, Springer-Verlag.

[5] K. Bulota, P. Survila. (1990). Algebra ir skaičių teorija 2. Mokslas.

[6] R. Crandall, C. Pomerance "Prime Numbers. A Computational perspective", Springer-Verlag 2002.

[7] S. Hamdy (2004 may). LiDIA a library for computational number theory. ftp://ftp.informatik.tu-darmstadt.de/pub/TI/systems/LiDIA/current/LiDIA.pdf

[8] Hanz Delfs, Helmut Knebl. (2002). Introduction to Cryptography. Principeles and Applications Springer.

[9] W. Diffie, M.E. Hellman. (1976). New directions in cryptography. IEEE Transactions on Information Theory.

[10] N. Koblitz. (1994). A course in Number Theory and Cryptography". Springer-Verlag.

[11] Lietuviškas Tex/Lithuanian Tex (2003 01 22) [2004 02 12 aprašymo]. http://www.vtex.lt/tex/littex/

[12] A. Matuliauskas. (1985). Algebra : vadovėis aukštųjų mokyklų matematikos ir taikomosios matematikos specialybių studentams. Mokslas.

[13] R. Rivest, A. Shamir, and L.M. Adleman. (1978). A method of obtaining digital signatures and public key cryptosystems. Communications of the ACM.

[14] R. Šleževičienė (2004). Factoring with Pell conics, Liet. Matem. Rinkinys 44 (spec. nr) p. 120-124.

[15] R. Šleževičienė (2005). Kriptografijos įvadas, ŠU leidykla.

[16] R. Valatkaitė Z. Kudirka. (1999). Informatika. Keturkalbis terminų žodynas. Vilnius.

[17] A. Vidžiūnas.(2002). C++ ir Builder C++ pradmenys.Kaunas : Technologija.

5 Anotacija

Edita Žitkevičienė. Magistro darbo tezės tema: daugiakarčio tikslumo aritmetikos biblioteka LiDIA kriptografijos kurse. Darbo vadovė dr. R. Steuding. Šiaulių universitetas, Šiauliai, 2006. 38 puslapiai.

Nagrinėjant ir realizuojant kriptografinius algoritmus programavimo C++ kalba dažnai prireikia įvairių matematinių funkcijų, kurias galima rasti nemokamai platinamoje C++ daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA. Ši daugiakarčio tikslumo aritmetikos biblioteka LiDIA aprašyta virtualiame kriptografijos kurse adresu `http://mokymai.dist.su.lt/`

Nagrinėjama: viešojo rakto kriptografijos koncepcija, kai kurie modulinės aritmetikos elementai, RSA schema (raktų generavimas, šifravimas, dešifravimas, atakos), pirminio skaičiaus testai, faktorizacijos algoritmai.

Daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA gali būti naudojama nemokamai nekomerciniais tikslais. LiDIA veikia visose POSIX standartu suderinamose sistemose.

Summary

Edita Žitkevičienė. Informatics Master's Final Thesis. LiDIA is a Libary For Computational Number Theory in Cryptography Course. Work leader dr. R. Steuding. Šiauliai University. Siauliai, 2006. 38 pages.

LiDIA is a library for computational number theory. Since we find that object oriented programming is appropriate for implementing mathematical algorithms and since C++ belongs to the most accepted programming languages in scientific computing, we decided to use C++ as the implementation language for LiDIA. To guarantee easy portability of LiDIA we decided to have a very small machine dependent kernel in LiDIA. That kernel currently contains the multiprecision integer arithmetic and a memory manager. All LiDIA objects are implemented in C++ and compiled with many different compilers on various architectures.

Information about LiDIA can be found in website `http://mokymai.dist.su.lt/`. RSA scheme, algorithms of factorization, tests of prime numbers, some elements of modular arithmetic are analysed there. Although not in the public domain, LiDIA can be used freely for non commercial purposes.

6 Priedai

6.1 Vartotojo vadovas

Vartotojo vadovą galima rasti kompaktinio disko šakninio katalogo faile VartotojoVadovas.pdf. Jame aprašyta kaip reiktų įsidiegti ir susikompiliuoti biblioteką LiDIA, koks būtų scenarijus `configure` pagalba, bei aprašyti klasių šablonų įdiegimas.

6.2 Techninė užduotis

1. RSA schema (raktų generavimas, šifravimas, dešifravimas, atakos) daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA;
2. Modulinė aritmetika plius algoritmų sudėtingumas;
3. Pirminio skaičiaus testai. (Miller'io testo realizacija daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA;
4. Aprašyti ir išnagrinėti AKS algoritmą;
5. Padaryti pirminių skaičių generatorių pasinaudojant AKS algoritmu;
6. Įgyvendinti AKS algoritmą programiškai;
7. Faktorizacijos algoritmai. (Pollard'o $(p-1)$ metodas) daugiakarčio tikslumo aritmetikos bibliotekoje LiDIA;
8. Atlikti testavimą ir įsitikinti, kad programos veikia korektiškai;
9. Sukurti virtualų kriptografijos kursą.

6.3 Kompaktinio disko aprašymas

Kompaktinis diskas susideda iš katalogų `dokumentai`, `programos`, `projektai`.

Kataloge `dokumentai` yra darbo dokumentacija, tai `LiDIAmag.pdf` — magistranto darbas, `VartotojoVadovas.pdf` — instrukcijos kaip įdiegti ir naudoti C++ biblioteką LiDIA, `LiDIA.pdf` — bibliotekos LiDIA žinynas.

Kataloge `programos` yra sukompiliuotos visos darbe parašytos programos. Programa `aks` — tai AKS algoritmo realizacija, `faktorizavimas` — tai naujo faktorizavimo algoritmo išbandymas, `rsa` — tai supaprastinas RSA šifravimo ir dešifravimo sistema, `rsa_ataka`, `rsa_ataka2` ir `rsa_ataka3` — tai kelių RSA atakų realizavimas, `uzd01` — tai dviejų didelių skaičių bendro didžiausio daliklio paieška rekursiniu ir iteraciniu būdu ir abiejų metodų efektyvumo palyginimas, `liginiu_sistema` — tai keturių lyginių sistemos sprendimo funkcija. Programos veikia Linux operacinėje sistemoje, kurioje suinstaliuota

daugiakarčio tikslumo aritmetikos biblioteka LiDIA. Norint paleisti, pavyzdžiui programą aks, reikia atlikti šiuos veiksmus (jei jūsų kompaktinių diskų skaitymo įrenginys montuojamas ne prie `/cdrom` katalogo, atitinkamai pakeiskite komandas):

```
mount /cdrom
cd /cdrom/programos/aks
./aks
```

Įvykdžius šias komandas bus pradėta vykdyti aks programa. Atkreipkite dėmesį, kad Linux operacinėje sistemoje nenaudojami programų plėtiniai `.exe`, kadangi UNIX tipo operacinėse sistemose vykdomosios bylos atskiramos pagal bylų sistemos leidimus. Baigus naudotis kompaktine plokštele, ją reikia išmontuoti komandų pagalba:

```
cd /
umount /cdrom
eject
```

gali būti, kad jūsų sistemoje nėra programos `eject`, tuomet kompaktinę plokštelę išimsite paspaudę plokštelių skaitymo įrenginio atitinkamą mygtuką.

Kataloge `projektai` yra sukurtų programų tekstai. Sistema, kurioje norima sukompiliuoti sukurtas programas, turi atitikti tam tikrus reikalavimus:

- sistema turi atitikti POSIX standartą (žr. LiDIA reikalavimus sistemai priede `VartotojoVadovas.pdf`);
- sistemoje turi būti įdiegtas LiDIA branduolys (žr. LiDIA reikalavimus sistemai priede `VartotojoVadovas.pdf`), šiame darbe buvo naudota biblioteka „GNU Multiple Precision Arithmetic Library“ (`libgmp3` paketas Debian GNU/Linux sistemoje), tai atviro kodo biblioteka, kurią laisvai galima atsisiųsti iš `http://www.swox.com/gmp`;
- reikalingas C++ kompiliatorius, suderinamas su ISO C++, darbe naudotas „GNU Compiler Collection“, kuris taip pat yra atviro kodo ir laisvai prieinamas adresu `http://gcc.gnu.org/`;
- programos naudoja LiDIA biblioteką, taigi, sistemoje turi būti sukompiliuota ir įdiegta LiDIA biblioteka (žr. `VartotojoVadovas.pdf`).

Programos sukurtos naudojant programavimo aplinką KDevelop, todėl paprasčiausia programas kompiliuoti šio įrankio pagalba. Užtenka su šia programa atidaryti projekto bylą (projekto bylos pavadinimas visada baigiasi `.kdevelop`). Atidarius projektą, jį reikia sukompiliuoti mygtuko F8 pagalba (arba meniu: Build-Build project) ir įvykdyti sukompiliuotą programą klavišų kombinacijos pagalba Shift+F9 (arba meniu: Build-Execute program).

Kataloge `projektai` taip pat yra pakatalogis `tex`, jame visų šiame darbe esančių dokumetų išeities tekstai LaTeX formatu. Kataloge `lidia` yra bibliotekos LiDIA žinynas, kataloge `magistro` — šio dokumento tekstai. Norint iš šių šaltinių sukompiliuoti lietuviškas PDF, HTML ar RTF bylas, reikalingas LaTeX paketas `http://www.latex-project.org/` ir jo sulietuvinimas `http://www.vtex.lt/tex/littex/`. Debian GNU/Linux operacinėje sistemoje patogiau naudoti Kęstučio Biliūno paruoštus LaTeX lietuvinimo paketus `http://kebil.ghost.lt/#littex`.