Lukuteoria ja ohjelmointi lukion uudessa opetussuunnitelmassa

Sampo Suomalainen

25. toukokuuta 2021

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

Tiedekunta – Fakultet – Faculty	Koulutusohjelma – Utbildningsprogram – Degree programme
Matemaattis-luonnontieteellinen tiedekunta	Matematiikan, fysiikan ja kemian opettajan maisteriohjelma

Tekijä – Författare – Author

Suomalainen, Sampo

Työn nimi – Arbetets titel – Title

Lukuteoria ja ohjelmointi lukion uudessa opetussuunnitelmassa

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro Gradu -tutkielma	24.05.2021	89

Tiivistelmä – Referat – Abstract

Tutkielman tavoitteina on tarkastella lukuteoriaa ja sen soveltuvuutta lukio-opetukseen sekä kirjallisuuteen perustuen selvittää, mitä hyötyä lukuteorian ja ohjelmoinnin yhdistämisessä opetuksessa voisi olla. Motivaationa taustalla toimii lukion uusi opetussuunnitelma 2019 ja erityisesti pitkän matematiikan valtakunnallinen syventävä kurssi MAA11 – Algoritmit ja lukuteoria, jonka keskeisiin sisältöihin sekä lukuteoria että ohjelmointi kuuluvat. Pääasiallisena osana tutkielmaa esitellään konkreettisia ohjelmointiharjoituksia ja -kokonaisuuksia, joiden avulla lukuteorian eri aihealueita voitaisiin lukio-opetuksessa käsitellä ohjelmoinnin kautta.

Matematiikan ja ohjelmoinnin yhdistämistä opetuksessa on tutkittu jo entuudestaankin paljon. Tähän liittyen usein puhutaan laskennallisen ajattelun käsitteestä. Laskennalliseen ajatteluun sisältyy valikoima erilaisia ajatuksellisia työkaluja, joiden avulla ongelmia voidaan ratkaista ja jäsentää. Laskennallisen ajattelun taidoista on todettu olevan hyötyä monella osa-alueella, esimerkiksi matematiikassa. Yksi luontainen tapa laskennallisen ajattelun kehittämiseen on ohjelmointi. Toisaalta puolestaan tietojenkäsittelytieteen juuret ovat matematiikassa, joten näillä kahdella tieteenalalla on paljon yhteistä. Myös kontekstilähtöisen opettamisen on huomattu parantavan opiskelijoiden motivaatiota, oppimistuloksia sekä ymmärrystä tieteen yhteydestä arkeen ja ympäröivään maailmaan. Yksi lukuteorian tärkeitä sovelluskohteita on erilaiset kryptografian salausmenetelmät, joten ohjelmointi tarjoaa myös mahdollisuuksia tuoda kontekstuaalisuutta ja relevanssia osaksi lukuteorian opetusta.

Sekä laskennallisen ajattelun että kontekstilähtöisen opettamisen haasteiksi on koettu konkreettisten välineiden ja menetelmien puute. Tämän tutkielman tarkoitus on vastata näihin haasteisiin esittelemällä joitakin mahdollisia tapoja lukuteorian ja ohjelmoinnin yhdistämiseen ikään kuin pedagogisena tuotteena. Laaditut ohjelmalliset tehtävät tarjoavat toisaalta matalan kynnyksen lähteä tutkimaan lukuteorian aiheita, mutta myös haastavat kartuttamaan syvempää ymmärrystä pohdinnan ja lisätehtävien kautta. Tutkielmassa esitellään myös lukuteorian keskeistä matemaattista perustaa niin lukion opetussuunnitelmaan sisältyviltä osin, kuin sen ulkopuoleltakin.

Pelkästään lukion opetussuunnitelman lukuteoriaan liittyvien sisältöjen puitteissa mahdollisia ohjelmallisia tehtäviä tai käsiteltäviä aihealueita on paljon, ja tämä tutkielma laajuudessaan pystyy vasta raapaisemaan pintaa kaikkien mahdollisuuksien suhteen. Ohjelmallisten harjoitteiden ja ohjelmointia ja lukuteoriaa yhdistelevien tehtävien osalta tutkielma antaa kuitenkin jo ideoita ja luo pohjaa näitä menetelmiä arvioivalle tai kehittävälle jatkotutkimukselle, sillä tämän tutkielman osalta niitä käsiteltiin vasta teoreettisella tasolla.

Avainsanat – Nyckelord – Keywords

Lukuteoria, ohjelmointi, lukio

Säilytyspaikka – Förvaringställe – Where deposited

E-Thesis: https://ethesis.helsinki.fi/

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto **3**

2 Matematiikka ja ohjelmointi **4**
2.1 Lukion opetussuunnitelma 2019 4
2.2 Lukuteorian opetus 5
2.3 Ohjelmointi osana matematiikan opetusta 6
2.4 Ohjelmointikielenä Python 8

3 Lukuteoria **9**
3.1 Jaollisuus ja jakoyhtälö 10
3.2 Suurin yhteinen tekijä ja pienin yhteinen jaettava 13
3.3 Kongruenssi 17
3.4 Aritmetiikan peruslause 19
3.5 Eukleideen algoritmi 19
3.6 Lineaariset Diofantoksen yhtälöt 20
3.7 Laajennettu Eukleideen algoritmi 22
3.8 Kiinalainen jäännöslause 22
3.9 Eulerin φ-funktio 25
3.10 RSA-kryptografia 28
3.11 Primitiiviset juuret 29

4 Ohjelmointi: Jaollisuus **33**
4.1 Aritmetiikan peruslause 34
4.2 Eratostheneen seula 37
4.3 Eukleideen algoritmi 41
4.4 Pienin yhteinen jaettava 44
4.5 Eulerin φ-funktio 45

5 Ohjelmointi: Diofantoksen yhtälöt **47**
5.1 Kahden muuttujan lineaariset Diofantoksen yhtälöt 47

6 Ohjelmointi: Kongruenssi **55**
6.1 Lukujen ja yhtälöiden tarkastelu kongruenssin avulla 55
6.2 Caesarin salaus 58
6.3 Vigenèren salaus 63
6.4 RSA-algoritmi 67
6.5 Kiinalainen jäännöslause 75
6.6 Diffien–Hellmanin-avaintenvaihtoprotokolla 79

7 Pohdinta **85**

Kirjallisuutta **87**

Luku 1

Johdanto

Tässä pro gradu -tutkielmassa tarkastellaan lukuteorian ja ohjelmoinnin yhdistämistä lukio-opetuksessa. Tutkielman aihetta motivoi uuden lukion opetussuunnitelman perusteiden valtakunnallinen syventävä pitkän matematiikan kurssi MAA11, jonka keskeisissä sisällöissä sekä ohjelmointi että lukuteoria esiintyvät. Tutkielman tavoitteena on kirjallisuuteen pohjautuen kartoittaa, mitä hyötyjä lukuteorian ja ohjelmoinnin yhdistämisestä opetuksessa voisi olla, ja tarjota konkreettista pedagogista materiaalia kurssin toteuttamisen tueksi. Tavoitteena on välttää MAA11-kurssin opetuksessa lukuteorian ja ohjelmoinnin sisältojen jääminen liian toisistaan irrallisiksi kokonaisuuksiksi, jolloin koko kurssin toteutus saattaisi jäädä hajanaiseksi.

Tutkielman luku 2, matematiikka ja ohjelmointi, käsittelee lukion uutta opetussuunnitelmaa sekä taustateoriaa liittyen matematiikan ja ohjelmoinnin opiskeluun ja niiden yhdistämiseen opetuksessa. Luvussa 3 esitellään lukuteoriaan liittyvää matemaattista teoriaa, ja käydään läpi keskeisiä määritelmiä sekä lauseita. Näiden pohjalta luvuissa 4–6 tarkastellaan mahdollisia lukuteoriaa ja ohjelmointia yhdistäviä tehtäviä sekä ohjelmointikokonaisuuksia, joiden avulla aiheita voitaisiin käsitellä käsi kädessä lukio-opetuksessa. Luvussa 7 pohditaan vielä lyhyesti ohjelmointikokonaisuuksien käytännön soveltuvuutta opetukseen luvussa 2 käydyn teorian näkökulmasta, sekä perustellaan tarvetta jatkotutkimukselle.

Luku 2

Matematiikka ja ohjelmointi

2.1 Lukion opetussuunnitelma 2019

Vuonna 2019 julkaistiin uudet lukion opetussuunnitelman perusteet [22]. Näiden perusteiden mukaiset opetussuunnitelmat otetaan käyttöön syksyllä 2021. Aiemmat lukion opetussuunnitelman perusteet otettiin käyttöön syksyllä 2016 [23]. Uusimmassa opetussuunnitelmassa suuri rakennemuutos verrattuna edellisiin opetussuunnitelmiin on laajuudeltaan samankokoisten kurssien muuttuminen keskenään eri laajuisiksi moduuleiksi [22]. Tämän tutkielman kannalta olennaisin uuden opetussuunnitelman sisältö kuitenkin on pitkän matematiikan uusi valtakunnallinen syventävä kurssi MAA11 (Algoritmit ja lukuteoria), jonka keskeisissä sisällöissä esiintyy nimensä mukaisesti sekä lukuteoriaa että ohjelmointia.

Teknologian hyödyntäminen matematiikan opetuksessa on ollut osa myös jo aiempia opetussuunnitelmia. Esimerkiksi lukion opetussuunnitelman perusteissa 2015 jokaisen lyhyen ja pitkän matematiikan kurssin tavoitteisiin on listattu, että opiskelijan tulisi osata hyödyntää teknisiä apuvälineitä kyseessä olevan kurssin sisältöalueiden käsittelyssä ja mallintamisessa. Pääasiassa teknisinä apuvälineinä on kuitenkin tähän asti hyödynnetty symbolisia laskentaohjelmistoja ja graafisia geometriaohjelmistoja, eikä ohjelmointi nimellisesti ole ollut osa lukion matematiikan kurssien sisältöjä. Sen sijaan peruskoulun opetussuunnitelman perusteissa 2014 vuosiluokkien 7–9 matematiikan opetuksen tavoitteissa on listattu algoritmillisen ajattelun kehittäminen sekä ohjelmoinnin soveltaminen osana ongelmanratkaisua [23, 24].

MAA11-moduuli yhdistelee vanhan lukion opetussuunnitelman perusteiden 2015 kurssien MAA11 (Lukuteoria ja todistaminen) sekä MAA12 (Algoritmit matematiikassa) sisältöalueita [22, 23]. Moduulin keskeisiksi sisällöiksi listataan seuraavat:

- algoritmisen ajattelun peruskäsitteet: peräkkäisyys, valinta ja toisto
- vuokaavio
- yksinkertaisten algoritmien, lajittelualgoritmien ja yhtälön numeeriseen ratkaisuun liittyvien algoritmien ohjelmointi
- konnektiivit ja totuusarvot
- kokonaislukujen jaollisuus, jakoyhtälö ja kongruenssi
- Eukleideen algoritmi
- aritmetiikan peruslause

Lukion opetussuunnitelman perusteiden 2015 kurssien MAA11 ja MAA12 joitakin sisältöalueita on muutoksessa karsittu. MAA11-kurssin tavoitteista matemaattisen todistamisen eri muodot, alkuluvut ja Eratostheneen seula eivät siirtyneet uuteen opetussuunnitelmaan [22, 23]. Toisaalta MAA12-kurssin tavoitteista oikeastaan mikään lukuunottamatta algoritmillista ajattelua ja yhtälöiden numeerista ratkaisemista ei enää näy lukion opetussuunnitelman perusteiden 2019 pitkän matematiikan moduulien sisältötavoitteissa.

2.2 Lukuteorian opetus

Monen valtion valtiollisessa opetussuunnitelmassa lukuteorian nähdään olevan pienessä roolissa suhteessa esimerkiksi algebraan, differentiaalilaskentaan tai geometriaan [11]. Näin voidaan nähdä asian laidan olevan suomalaisessakin opetussuunnitelmassa, sillä esimerkiksi peruskoulun opetussuunnitelmassa lukuteoria näkyy lähinnä sisältöalueessa S2: Luvut ja laskutoimitukset, ja sielläkin vain lyhyesti muun laskennan ohessa [24]. Myös lukion opetussuunnitelmissa 2015 ja 2019 lukuteoriaa on käsitelty osana yhtä ainoaa valtakunnallisesti syventävää kurssia [22, 23]. Yleisesti matematiikan opetuksessa ehkä onkin vallinnut ajatus, että tarvittava lukuteoria voidaan oppia muun matematiikan ohessa, ja että se ei tarvitsisi samalla tavoin tilaa kuin esimerkiksi yllä mainitut muut matematiikan osa-alueet, tai sitten mahdollisesti lukuteorian käytännön merkitystä ei ymmärretä kunnolla, ja sitä pidetään turhan teoreettisena ja abstraktina aihealueena [11].

Tutkimuksessa on kuitenkin huomattu lukuteorian positiivinen vaikutus matemaattisen ajattelun tukena [11]. Lukuteoria vaikuttaisi tarjoavan hyvän oppimisalustan ongelmanratkaisulle, päättelemiselle, yleistämiselle ja abstrahoinnille. Lisäksi esimerkiksi

todistamisen ja erilaisten todistusten ymmärtämistä saattaisi auttaa niiden käsittely lukuteorian kautta, sillä lukuteorian perusteisiin liittyvät todistukset koetaan yleisesti selkeämmiksi kuin monet muut matemaattiset todistukset [11].

Ongelmanratkaisu on yleisesti nähty keskeisenä tekijänä matemaattisen ajattelun kehittymisessä [31]. Hyvin suunnitellut ongelmat voivat innostaa ja tehdä oppimisesta palkitsevaa [26]. Myös tässä tutkielmassa lähdetään tarkastelemaan lukuteorian opettamista tältä pohjalta. Luvuissa 4–6 esitellään mahdollisia ongelmia, joiden kautta lukuteoriaa voitaisiin lukio-opetuksessa käsitellä ohjelmoinnin kautta. Usein aiheen kokonaisvaltaisen ymmärryksen tavoittelussa hyödyllistä on tarkastella aluksi konkreettisia esimerkkejä ja tilanteita, ja ohjelmoinnin avulla tällaista tutkintaa voidaan suorittaa nopeasti isollekin joukolle erilaisia erilaisia lukuja tai tapauksia. Toisaalta yleisen ohjelman laadinta, joka toimii kaikissa tapauksissa ja millä tahansa syötteillä, vaatii käsiteltävän asian syvällistä ymmärtämistä [32].

On myös mainitsemisen arvoista, että kontekstilähtöisen opettamisen on huomattu tukevan eri luonnontieteiden oppimista [9]. Kontekstilähtöisellä opettamisella tarkoitetaan opetusta, missä kyseessä olevan oppiaineen sisältöjä opetetaan jonkin oikeaan maailmaan ja käytäntöön liittyvän kontekstin kautta. Hyvin toteutetun kontekstilähtöisen opetuksen on huomattu kasvattavan opiskelijoiden motivaatiota, pystyvyyden tunnetta sekä auttanut opiskelijoita näkemään yhteyden luonnontieteiden ja ympäröivän maailman välillä [5]. Tässä mielessä lukuteorian ja ohjelmoinnin yhdistäminen tarjoaa myös mahdollisuuksia opettaa lukuteoriaa ympäröivän maailman kontekstissa. Esimerkiksi suurilta osin nykyaikaisen kryptografian perusteet ovat algebrassa ja lukuteoriassa [15]. Tämän motivoimana myös tutkielman ohjelmallisiin sovelluksiin luvussa 6 on sisällytetty muutamia tunnettuja kryptografisia menetelmiä.

2.3 Ohjelmointi osana matematiikan opetusta

Matematiikan ja ohjelmoinnin sisältöjen yhdistämistä osana opetusta on tutkittu maailmalla paljon [14]. Vuonna 2006 Jeanette Wing toi tunnetuksi termin *laskennallinen ajattelu* (*computational thinking*), jolla tarkoitetaan valikoimaa kognitiivisia työkaluja ongelmien jäsentämiseen ja ratkaisuun [38]. Tällaisia työkaluja ovat esimerkiksi ongelman haastavuuden ennakointi ja ongelmanratkaisuun valmistava suunnitelmallisuus, ongelman pilkkominen pienempiin osiin, tehokkaiden ratkaisumetodien valitseminen sekä saadun ratkaisun arviointi ja analysointi. Wing argumentoi, että tällaiset tavallisesti tietojenkäsittelytieteen näkökulmasta hyödylliset ajattelulliset työkalut olisivat tärkeitä jokaiselle, ja siitä syystä laskennallisen ajattelun kuuluisi sisältyä yleisiin opetussuunnitelmiin. Muutkin ovat painottaneet laskennallisen ajattelun

merkitystä nyky-yhteiskunnassa [10]. Flórez et al. näkisivätkin tarpeellisena lisätä laskennallisen ajattelun osaksi peruskoulujen opetussuunnitelmaa sekä opettamiseen valmistavia korkeakouluopintoja. He katsovat laskennallisen ajattelun olevan tulevaisuuden taito, josta on hyötyä alalla kuin alalla.

Suomessa laskennallinen ajattelu ei näy vielä kovin vahvasti opetussuunnitelmissa. Ainoastaan perusopetuksen opetussuunnitelmassa 2014 matematiikan opetuksen tavoitteisiin on listattu algoritmillisen ajattelun kehittäminen, jonka voidaan ajatella tarkoittavan samankaltaisia taitoja laskennallisen ajattelun kanssa [24]. Tieto- ja viestintäteknologia nähdään muutenkin apuvälineenä, mutta tässä yhteydessä laskennallisen ajattelun kehittäminen ei painotu [22, 23, 24]. Kuitenkin, esimerkiksi opetusministeriön työryhmä on todennut, että laskennallisen ajattelun kehittäminen on strategisesti tärkeää Suomen kilpailukyvylle [25]. Myös Kankaanranta et al. ovat raportoineet laskennallisen ajattelun opetuksesta Yhdysvalloissa, Iso-Britanniassa ja Tanskassa, ja ovat raportissaan suositelleet hanketta, jolla arvioitaisiin tieto- ja viestintäteknologian, erityisesti laskennallisen ajattelun, tavoitteiden saavuttamista ja vaikuttavuutta perusopetuksessa ja lukiossa, jotta perusopetuksessa ja lukiossa annettava opetus voitaisiin nostaa EU:ssa asetettujen ICT-tavoitteiden tasolle [16].

Vaikka laskennallisen ajattelun tärkeys osana tulevaisuuden keskeisiä taitoja on jo huomattu ympäri maailmaa, ei sen toteuttaminen mielekkääksi osaksi opetusta ole ollut ongelmatonta [17]. Kuitenkin tässä vaiheessa empiiristä tutkimusta on jo ehditty laajalti tekemään, ja esimerkiksi kirjassa [17] listataan useita empiirisesti tutkittuja konkreettisia tapoja lähestyä laskennallisen ajattelun opettamista eri kouluasteilla ja eri oppiaineissa. Matematiikan opetuksen suhteen tarkasteltuja tapoja ovat esimerkiksi digitaalinen pelillistäminen ja lukujen jaollisuuteen perustuvan ohjelman kirjoittaminen.

Matematiikkaan laskennallisen ajattelun opettaminen sopii erityisen hyvin. Esimerkiksi suuri osa lukiomatematiikan sisällöistä on pohjimmiltaan hyvin algoritmista, kuten yhtälönratkaisu, derivointi, integrointi, sekä ääriarvokohtien ja ääriarvojen ratkaiseminen. Myös luovissa ongelmanratkaisutehtävissä laskennallisesta ajattelusta on paljon hyötyä. Matematiikka ja tietojenkäsittelytiede nähdään pääasiassa erillisinä oppiaineina, mutta niillä on paljon yhteistä. Onhan suuri osa tietojenkäsittelytieteestä pohjimmiltaan matematiikkaa [19]. Ohjelmoinnin sisällyttäminen paitsi vahvistaisi näiden oppiaineiden yhteyttä ja opettaisi laskennallista ajattelua, se voisi tarjota mielekkään tavan käsitellä joitakin matematiikan sisältöjä ja tarjota opiskelijoille uusia työkaluja hahmottamiseen ja ongelmanratkaisuun [36].

2.4 Ohjelmointikielenä Python

Lukuteorian ohjelmointikokonaisuuksien esimerkit on tässä tutkielmassa laadittu käyttäen Python-ohjelmointikieltä. Kielen valinnan pohjalla on tutkielman kirjoittajan oma käyttökokemus kyseisestä kielestä, mutta lisäksi myös sen soveltuvuus ohjelmoinnin perusteiden harjoitteluun. Python on suunniteltu lähtökohtaisesti opetukselliseen käyttöön, ja sen etuihin kuuluu mm. yksinkertainen syntaksi [28]. Mannila ja de Raadt ovat vertailleet 11 eri ohjelmointikielen soveltuvuutta opetuskäyttöön, ja heidän vertailussaan parhaiten ohjelmoinnin opettamiseen sopiviksi nousivat Python sekä Eiffel [20]. Python on myös yksi suosituimpia ohjelmointikieliä, joita suomessa yläasteilla opetetaan [13]. Monet oppimateriaalivalmistajat, kuten esimerkiksi Sanoma Pro ja Edita, ovat myös valinneet Pythonin oppimateriaaleihinsa ohjelmointikieleksi [12, 30]. Tästäkin syystä Python tuntuu luontevalta valinnalta lukion ohjelmoinnin opetukseen, sillä monella opiskelijalla voi olla sen käytöstä jo kokemusta aiempien opintojen perusteella. Lisäksi lukion yksi keskeinen tehtävä [22] on valmistaa opiskelijaa jatko-opintoja varten, ja Python on tällä hetkellä hyvin suosittu kieli myös korkeakoulujen keskuudessa ja käytössä tutkielman kirjoitushetkellä mm. Helsingin, Turun, Itä-Suomen, Tampereen, Oulun yliopistoissa sekä LUT-yliopistossa.

Pythonin asentaminen ja ajaminen komentoriviltä on yksinkertaista, mutta sille löytyy myös useita ilmaisia selainpohjaisia editoreita (kuten esimerkiksi [35]), joten se on helposti kaikkien saatavilla. Se myös on tällä hetkellä yksi suosituimmista ohjelmoin- tikielistä [34]. Kielen käyttöä osana opetusta puoltavat myös tieteelliset artikkelit ja tutkimukset [19, 21]. Mainittakoon, että joistakin muista kielivalinnoista, kuten C++:sta tai Javasta, voisi olla hyötyä osana myöhempää ohjelmoinnin harjoittelua, sillä niissä ohjelmoinnin taustalla olevat rakenteet ovat näkyvämpiä [21]. Ohjelmoinnin perusteiden harjoitteluun Python vaikuttaisi kuitenkin olevan oikein perusteltu valinta.

Luku 3

Lukuteoria

Kappaleissa 3.1–3.5 esitellään lukion opetussuunnitelman lukuteorian sisältöalueiden taustalla oleva matemaattinen perusta. Kappaleet 3.6–3.11 esittelevät lukion opetussuunnitelman ulkopuolisia lukuteorian sisältöjä, joiden tutkielman kirjoittaja on nähnyt soveltuvan mahdollisesti lukion opetussuunnitelmaa laajentavaksi materiaaliksi. Laajentava materiaali voisi soveltua esimerkiksi harrastuneille ja lukuteoriasta kiinnostuneille opiskelijoille, tai sitten sitä voitaisiin käydä läpi mahdollisilla koulukohtaisilla syventävillä kursseilla.

Lukijan on hyvä huomata, että lukua 3 ei sellaisenaan ole tarkoitettu lukion oppimateriaaliksi, ja useassa kohtaa käytetty notaatio voi poiketa siitä, mihin lukiomatematiikassa yleisesti on totuttu. Luvun 3 sisältöjen mahdollista toteutusta tai ehdotuksia niiden läpikäymiseksi kommentoidaan osana joitakin lukujen 4–6 ohjelmointikokonaisuuksia ja -tehtäviä, sekä pohditaan tarkemmin luvussa 7. Tässä vaiheessa pysyttäydytään lukuteorian matemaattisen perustan ja määritelmien esittelemisessä sekä olennaisien lauseiden todistamisessa.

Notaatio

Lukujoukkojen suhteen käytetään seuraavia yleisiä merkintöjä:

$$\begin{aligned}
\text{Luonnolliset luvut: } & \mathbb{N} = \{0, 1, 2, 3, \ldots\} \\
\text{Kokonaisluvut: } & \mathbb{Z} = \{\ldots -3, -2, -1, 0, 1, 2, 3, \ldots\} \\
\text{Positiiviset kokonaisluvut: } & \mathbb{Z}_+ = \{1, 2, 3, \ldots\} \\
\text{Negatiiviset kokonaisluvut: } & \mathbb{Z}_- = \{-1, -2, -3, \ldots\} \\
\text{Alkuluvut (ks. 3.4): } & \mathbb{P} = \{2, 3, 5, 7, 11, \ldots\}
\end{aligned}$$

Jos joukko U on tyhjä, merkitään $U = \varnothing$.

Lukuteorian lähteinä kappaleissa 3.1–3.11 on pääasiallisesti käytetty kirjaa [6] sekä luentomateriaalia [29]. Näissä luvuissa esitelty teoria seuraa jossain määrin lähdemateriaalien esitystapaa sekä järjestystä.

3.1 Jaollisuus ja jakoyhtälö

Määritelmä 3.1. (Jaollisuus) Kokonaisluku b on *jaollinen* kokonaisluvulla a, eli luku a on luvun b *tekijä*, jos on olemassa sellainen kokonaisluku k, että

$$b = k \cdot a.$$

Tällöin merkitään $a \mid b$. Jos luku b ei ole jaollinen luvulla a, merkitään $a \nmid b$.

Lause 3.2. Olkoot $a, b, c, b_1, b_2, \ldots b_n \in \mathbb{Z}, n \in \mathbb{Z}_+$. Tällöin

(i) jos $a \mid b$, niin $b = 0$ tai $|b| \geq a$,

(ii) jos $a \mid b$ ja $b \mid c$, niin $a \mid c$,

(iii) jos $a \mid b$, niin $a \mid bc$,

(iv) jos $a \mid b_1, a \mid b_2, \ldots, a \mid b_n$, niin $a \mid (b_1 + b_2 + \ldots + b_n)$.

Todistus. (i) Jos $a \mid b$, niin $b = ka$ jollakin $k \in \mathbb{Z}$. Tehdään vastaoletus, että $|b| = |ka| < a$ ja $b \neq 0$. Toisaalta, koska $b \neq 0$, niin $k \neq 0$. Siis $|k| \geq 1$. Tällöin $|ka| = |k||a| \geq |a| \geq a$, mikä on ristiriita.

(ii) Jos $a \mid b$, niin $b = ka$ jollakin $k \in \mathbb{Z}$. Jos $b \mid c$, niin $c = lb$ jollakin $l \in \mathbb{Z}$. Tällöin $c = lb = lka$, joten jaollisuuden määritelmän nojalla $a \mid c$.

(iii) Jos $a \mid b$, niin $b = ka$ jollakin $k \in \mathbb{Z}$. Tällöin $bc = kca$, joten jaollisuuden määritelmän nojalla $a \mid bc$.

(iv) Jos $a \mid b_1, a \mid b_2, \ldots, a \mid b_n$, niin

$$b_1 = k_1 a, b_2 = k_2 a, \ldots, b_n = k_n a$$

joillakin $k_1, k_2, \ldots, k_n \in \mathbb{Z}, n \in \mathbb{Z}_+$. Tällöin

$$b_1 + b_2 + \ldots + b_n = k_1 a + k_2 a + \ldots + k_n a = (k_1 + k_2 + \ldots + k_n)\, a,$$

joten jaollisuuden määritelmän nojalla $a \mid (b_1 + b_2 + \ldots + b_n)$. □

Lause 3.3. (Jakoyhtälö) Olkoon $b \in \mathbb{Z}_+$. Tällöin jokaiselle kokonaisluvulle a on olemassa yksikäsitteiset $r \in \mathbb{N}, r < b$ ja $k \in \mathbb{Z}$, joilla

$$a = kb + r.$$

Todistus. Todistetaan aluksi lukujen k ja r olemassaolo. Olkoon

$$U = \{a - ub \mid u \in \mathbb{Z}, a - ub \geq 0\}.$$

Näytetään, että $U \neq \varnothing$. Riittää näyttää, että on olemassa sellainen $u \in \mathbb{Z}$, jolla $a - ub \geq 0$. Valitaan $u = -|a|$. Tällöin

$$a - ub = a - (-|a|)\, b = a + |a|b.$$

Koska $b \geq 1$, niin $|a|b \geq |a|$, joten $a + |a|b \geq a + |a| \geq 0$. Näin ollen $a - (-|a|)\, b \in U$ eli $U \neq \varnothing$.

Olkoon nyt $u_1 \in \mathbb{Z}$ ja $r = a - u_1 b$ joukon U pienin alkio. Jos $r \geq b$, niin

$$r > r - b = a - u_1 b - b = a - (u_1 + 1)\, b \geq 0,$$

mikä on ristiriidassa luvun r määritelmän kanssa. Siis $0 \leq r < b$, ja voidaan valita $k = u_1$.

Todistetaan seuraavaksi lukujen k ja r yksikäsitteisyys. Olkoot $k_1, k_2, r_1, r_2 \in \mathbb{Z}$, $0 \leq r_1 < b$, $0 \leq r_2 < b$ sellaiset, että

$$r_1 = a - k_1 b \quad \text{ja} \quad r_2 = a - k_2 b.$$

Tällöin $r_1 - r_2 = a - k_1 b - (a - k_2 b) = (k_2 - k_1)\, b$, joten $b \mid (r_1 - r_2)$. Lauseen 3.2 kohdan *(i)* nojalla nyt $r_1 - r_2 = 0$ tai $|r_1 - r_2| \geq b$. Koska $|r_1 - r_2| < b$, niin $r_1 - r_2 = 0$, eli $r_1 = r_2$. Luvun r yksikäsitteisyydestä seuraa myös luvun k yksikäsitteisyys. □

Määritelmä 3.4. (Alkuluvut) Olkoon $p \in \mathbb{Z}, p \geq 2$. Jos kaikilla $k \in \mathbb{Z}_+$ pätee, että $k \mid p$ jos ja vain jos $k = 1$ tai $k = p$, niin p on *alkuluku*. Alkulukujen joukkoa merkitään kirjaimella $\mathbb{P}$.

Todetaan vielä, että $\mathbb{P} \neq \varnothing$. Olkoon $k \in \mathbb{Z}_+$. Tällöin $k \mid 2$ jos ja vain jos $k = 1$ tai $k = 2$, sillä jos $k > 2$, niin lauseen 3.2 kohdan *(i)* nojalla $k \nmid 2$. Siis $2 \in \mathbb{P}$. Vastaavasti $k \mid 3$ jos ja vain jos $k = 1$ tai $k = 3$, sillä $2 \nmid 3$. Siis $3 \in \mathbb{P}$. Toisaalta $4 \notin \mathbb{P}$, sillä $2 \mid 4$. Jatkamalla edelleen saadaan, että $\mathbb{P} = \{2, 3, 5, 7, 11, \ldots\}$.

Määritelmä 3.5. (Yhdistetyt luvut) Olkoon $n \in \mathbb{Z}, n \geq 2$. Jos $n \notin \mathbb{P}$, niin n on *yhdistetty luku*.

Lause 3.6. Olkoon $n \in \mathbb{Z}, n \geq 2$. Tällöin $n = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, joillain $p_1, p_2, \ldots, p_k \in \mathbb{P}$, $k \in \mathbb{Z}_+$.

Todistus. Tehdään vastaoletus, että on olemassa ainakin yksi $n \in \mathbb{Z}$, $n \geq 2$, jota ei voi muodostaa alkulukujen tulona $p_1 p_2 \ldots p_k$ joillakin $p_1, p_2, \ldots, p_k \in \mathbb{P}, k \in \mathbb{Z}_+$. Olkoon n_0 pienin tällainen luku. Tällöin $n_0 \notin \mathbb{P}$, eli $n_0 = n_1 n_2$ joillakin $n_1, n_2 \in \mathbb{Z}_+, n_1 < n_0$ ja $n_2 < n_0$. Tällöin oletuksen nojalla $n_1 = q_1 \cdot q_2 \cdot \ldots \cdot q_l$ ja $n_2 = q_1' \cdot q_2' \cdot \ldots \cdot q_m'$ joillakin $q_1, q_2, \ldots, q_l, q_1', q_2', \ldots, q_m' \in \mathbb{P}, l, m \in \mathbb{Z}_+$. Valitsemalla nyt

$$p_1 = q_1, p_2 = q_2, \ldots, p_l = q_l, p_{l+1} = q_1', p_{l+2} = q_2', \ldots, p_{l+m} = q_m'$$

nähdään, että $n = p_1 p_2 \ldots p_k$, missä $k = l + m$, mikä on ristiriita. □

Lause 3.7. Olkoon $n \in \mathbb{Z}, n \geq 2$ ja $n = p_1 p_2 \ldots p_k$ joillakin $p_1, p_2, \ldots, p_k \in \mathbb{P}, k \in \mathbb{Z}_+$. Jos n on yhdistetty luku, niin $p_i \leq \sqrt{n}$ jollakin $i \in \mathbb{Z}_+, i \leq k$.

Todistus. Jos n on yhdistetty luku, niin $k \geq 2$. Tehdään vastaoletus, että $p_i > \sqrt{n}$ kaikilla i. Nyt $n > (\sqrt{n})^k \geq (\sqrt{n})^2 = n$, mikä on ristiriita. □

Lause 3.8. Alkulukuja on äärettömän monta.

Todistus. Tehdään vastaoletus, että alkulukujen määrä on äärellinen. Tällöin kaikki alkuluvut voidaan luetella: $p_1, p_2, p_3, \ldots, p_k$, missä $k \in \mathbb{Z}_+$ ja $p_i \in \mathbb{P}$ kaikilla $i \in \mathbb{Z}_+$, $i \leq k$. Olkoon

$$n = 1 + \prod_{i=1}^{k} p_i,$$

Tällöin $p_i \nmid n$ kaikilla p_i, mikä on ristiriidassa lauseen 3.6 kanssa. □

3.2 Suurin yhteinen tekijä ja pienin yhteinen jaettava

Lause 3.9. (Suurin yhteinen tekijä) Olkoot $a, b \in \mathbb{Z}$, $a \neq 0$ tai $b \neq 0$. Tällöin on olemassa yksikäsitteinen $d \in \mathbb{Z}_+$, jolla on seuraavat ominaisuudet:

(i) $d \mid a$ ja $d \mid b$.

(ii) Olkoon $d' \in \mathbb{Z}_+$. Jos $d' \mid a$ ja $d' \mid b$ niin $d' \mid d$.

Lukua d kutsutaan lukujen a ja b *suurimmaksi yhteiseksi tekijäksi,* ja sitä merkitään $\text{syt}(a, b)$.

Todistus. Olkoon

$$d = \min\{ax + by \mid x, y \in \mathbb{Z}, ax + by \geq 1\}. \tag{3.10}$$

Nähdään, että joukko on epätyhjä, sillä joukkoon kuuluu esimerkiksi $ax'+by'$, missä $x' = a$ ja $y' = b$, sillä

$$ax' + by' = a^2 + b^2 \geq 1.$$

Siis d on hyvin määritelty. Joukon määritelmän nojalla $d \in \mathbb{Z}_+$. Lausutaan d muodossa $d = ax_0 + by_0$ joillakin $x_0, y_0 \in \mathbb{Z}$. Nähdään, että näin ollen d täyttää myös ehdon *(ii)*. Ehdon *(i)* toteutumisen varmistamiseksi tehdään vastaoletus, että $d \nmid a$. Tällöin lauseen 3.3 nojalla

$$a = kd + r, \quad \text{joillakin } k, r \in \mathbb{Z}, 0 < r < d.$$

Tästä edelleen saadaan, että

$$r = a - kd = a - k\,(ax_0 + by_0) = a - akx_0 - kby_0 = a\,(1 - kx_0) + b\,(-ky_0)\,.$$

Tämä on kuitenkin ristiriidassa luvun d määritelmän kanssa, sillä $1 \leq r < d$. Siis $d \mid a$. Vastaavasti voidaan päätellä $d \mid b$.

Suurimman yhteisen tekijän d yksikäsittäisyyden takaa se, että mikäli positiiviset kokonaisluvut d_1 ja d_2 täyttävät ehdot *(i)* ja *(ii)*, niin $d_1 \mid d_2$ ja $d_2 \mid d_1$, joten $d_1 = d_2$. □

Määritelmä 3.11. Olkoot $a, b \in \mathbb{Z}, a \neq 0$ ja $b \neq 0$. Jos $\text{syt}(a, b) = 1$, niin lukuja a ja b kutsutaan *keskenään jaottomiksi* tai *yhteistekijättömiksi.*

Lause 3.12. (Pienin yhteinen jaettava) Olkoot $a, b \in \mathbb{Z}$, $a \neq 0, b \neq 0$. Tällöin on olemassa yksikäsitteinen $h \in \mathbb{Z}_+$, jolla on seuraavat ominaisuudet:

(i) $a \mid h$ ja $b \mid h$

(ii) Olkoon $h' \in \mathbb{Z}_+$. Jos $a \mid h'$ ja $b \mid h'$ niin $h \mid h'$

Lukua h kutsutaan lukujen a ja b *pienimmäksi yhteiseksi jaettavaksi,* ja sitä merkitään $\text{pyj}(a, b)$.

Todistus. Olkoon

$$h = \min\{m \geq 1 \mid m \in \mathbb{Z}, a \mid m, b \mid m\}. \tag{3.13}$$

Joukko on epätyhjä, sillä esimerkiksi $m = |ab|$ kuuluu joukkoon, joten h on hyvin määritelty. Joukon määritelmän nojalla $h \in \mathbb{Z}_+$. Nähdään, että h täyttää myös ehdon *(i)*. Olkoon $h' \in \mathbb{Z}_+, a \mid h'$ ja $b \mid h'$. Ehdon *(ii)* varmistamiseksi tehdään vastaoletus, että $h \nmid h'$. Tällöin lauseen 3.3 nojalla

$$h' = kh + r, \quad \text{joillakin } k, r \in \mathbb{Z}, 0 < r < h.$$

Tästä edelleen saadaan, että $r = h' - kh$. Koska h' ja h kuuluvat yhtälössä (3.13) käytettyyn joukkoon, kuuluu myös r. Koska h on joukon pienin alkio, niin tämä on ristiriita. Siis $h \mid h'$.

Pienimmän yhteisen jaettavan yksikäsitteisyyden takaa se, että mikäli positiiviset kokonaisluvut h_1 ja h_2 täyttävät ehdot *(i)* ja *(ii)*, niin $h_1 \mid h_2$ ja $h_2 \mid h_1$. Siis $h_1 = h_2$. □

Lause 3.14. Olkoot $a, b \in \mathbb{Z}, a \neq 0, b \neq 0$. Tällöin $|ab| = \text{syt}(a, b) \cdot \text{pyj}(a, b)$.

Todistus. 1) Huomataan aluksi, että

$$a \mid \frac{|ab|}{\text{syt}(a, b)} \quad \text{ja} \quad b \mid \frac{|ab|}{\text{syt}(a, b)}.$$

Tällöin lauseen 3.12 kohdan *(ii)* nojalla

$$\text{pyj}(a, b) \mid \frac{|ab|}{\text{syt}(a, b)}$$

ja erityisesti

$$\text{pyj}(a, b) \leq \frac{|ab|}{\text{syt}(a, b)},$$

mistä edelleen $|ab| \geq \text{syt}(a, b) \cdot \text{pyj}(a, b)$.

2) Toisaalta lauseen 3.12 nojalla $\text{pyj}(a, b) \mid |ab|$, jolloin nähdään, että

$$\frac{|ab|}{\text{pyj}(a, b)}$$

on lukujen a ja b yhteinen tekijä, sillä

$$\frac{a}{\left(\frac{|ab|}{\operatorname{pyj}(a,b)}\right)} = \pm\frac{\operatorname{pyj}(a,b)}{b},$$

mikä on kokonaisluku. Vastaavasti luvulle b. Tällöin lauseen 3.9 kohdan *(ii)* nojalla

$$\frac{|ab|}{\operatorname{pyj}(a,b)} \mid \operatorname{syt}(a,b)\,,$$

ja erityisesti

$$\operatorname{syt}(a,b) \geq \frac{|ab|}{\operatorname{pyj}(a,b)},$$

mistä edelleen $|ab| \leq \operatorname{syt}(a,b) \cdot \operatorname{pyj}(a,b)$.

Kohtien 1) ja 2) nojalla $|ab| = \operatorname{syt}(a,b) \cdot \operatorname{pyj}(a,b)$. □

Korollaari 3.15. Olkoot $a, b \in \mathbb{Z}, a \neq 0$ tai $b \neq 0$ ja $d = \operatorname{syt}(a,b)$. Tällöin

(i) $d = ax_0 + by_0$ joillakin $x_0, y_0 \in \mathbb{Z}$,

(ii) $\{ax + by \mid x, y \in \mathbb{Z}\} = \{kd \mid k \in \mathbb{Z}\}$.

Todistus. (i) Seuraa suoraan yhtälöstä (3.10) lauseen 3.9 todistuksessa.

(ii) Osoitetaan aluksi, että $\{ax + by \mid x, y \in \mathbb{Z}\} \subset \{kd \mid k \in \mathbb{Z}\}$. Koska $d \mid a$ ja $d \mid b$, niin $d \mid (ax + by)$ kaikilla $x, y \in \mathbb{Z}$. Siis kaikilla $x, y \in \mathbb{Z}$ löytyy jokin $k \in \mathbb{Z}$, jolla $ax + by = kd$.

Osoitetaan seuraavaksi toinen suunta, eli että $\{ax + by \mid x, y \in \mathbb{Z}\} \supset \{kd \mid k \in \mathbb{Z}\}$. Tämä seuraa suoraan kohdasta *(i)*, sillä $kd = k\,(ax_0 + by_0) = akx_0 + bky_0$ joillakin $x_0, y_0 \in \mathbb{Z}$. □

Lause 3.16. Olkoot $a, b, k \in \mathbb{Z}$, $a \neq 0$ tai $b \neq 0$. Olkoon $d \in \mathbb{Z}_+, d \mid a$ ja $d \mid b$.

(i) $d = \operatorname{syt}(a,b)$ jos ja vain jos $\operatorname{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1$,

(ii) $\operatorname{syt}(a,b) = \operatorname{syt}(a + kb, b)$.

Todistus. (i) Osoitetaan aluksi suunta, että jos $d = \operatorname{syt}(a,b)$, niin $\operatorname{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1$. Nyt korollaarin 3.15 kohdan *(i)* nojalla on olemassa sellaiset $x_0, y_0 \in \mathbb{Z}$, joilla $ax_0 + by_0 = d$. Tällöin

$$\frac{a}{d} \cdot x_0 + \frac{b}{d} \cdot y_0 = 1, \tag{3.17}$$

ja $\frac{a}{d}, \frac{b}{d} \in \mathbb{Z}$. Osoitetaan, että $\text{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1$. Näytetään tämä vastaoletuksella, että on olemassa $\text{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = d' > 1$ Mutta tällöin, koska d' jakaa yhtälön (3.17) vasemman puolen, täytyy sen jakaa myös oikea puoli, eli $d' \mid 1$, mikä on ristiriita.

Osoitetaan vastaavasti, että jos $\text{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1$, niin $d = \text{syt}(a, b)$. On jälleen olemassa sellaiset $x_0, y_0 \in \mathbb{Z}$, joilla

$$\frac{a}{d} \cdot x_0 + \frac{b}{d} \cdot y_0 = 1.$$

Tällöin $ax_0 + by_0 = d$. Tästä seuraa, että $\text{syt}(a, b) = d$. Näytetään tämä vastaoletuksella, että on olemassa lukujen a ja b yhteinen tekijä $d' > d$. Mutta jälleen, koska d' jakaa yhtälön vasemman puolen, täytyy sen jakaa myös oikea puoli. Siis $d' \mid d$, mikä on ristiriita.

(ii) Oletetaan, että $\text{syt}(a, b) = d$ ja $\text{syt}((a + kb), b) = d'$. Nyt $d \mid a$ ja $d \mid b$, joten $d \mid (a + kb)$. Siis $d \mid d'$. Toisaalta, $d' \mid b$ ja koska $d' \mid kb$ ja $d' \mid (a + kb)$, niin $d' \mid a$. Siis $d' \mid d$, mistä seuraa, että $d = d'$. □

Lause 3.18. Olkoot $a, b, c \in \mathbb{Z}$. Jos $a \mid c, b \mid c$ ja $\text{syt}(a, b) = 1$, niin $ab \mid c$.

Todistus. Korollaarin 3.15 kohdan *(i)* nojalla $ax_0 + by_0 = 1$ joillakin $x_0, y_0 \in \mathbb{Z}$. Nyt $acx_0 + bcy_0 = c$. Toisaalta koska $a \mid c$, niin $c = ka$ jollakin $k \in \mathbb{Z}$ ja koska $b \mid c$, niin $c = lb$ jollakin $l \in \mathbb{Z}$. Nyt yhtälön vasen puoli voidaan kirjoittaa muodossa $a\,(lb)\,x_0 + b\,(ka)\,y_0$, mistä

$$ab\,(lx_0 + ky_0) = c.$$

Jaollisuuden määritelmän nojalla $ab \mid c$. □

Lause 3.19. Olkoot $a, b, c \in \mathbb{Z}$. Jos $a \mid bc$ ja $\text{syt}\,(a, b) = 1$, niin $a \mid c$.

Todistus. Korollaarin 3.15 kohdan *(i)* nojalla $ax_0 + by_0 = 1$ joillakin $x_0, y_0 \in \mathbb{Z}$. Nyt $acx_0 + bcy_0 = c$. Koska a jakaa yhtälön vasemman puolen, täytyy sen jakaa myös oikea puoli, eli $a \mid c$. □

Korollaari 3.20. Olkoot $a, b \in \mathbb{Z}$. Jos $p \in \mathbb{P}$ ja $p \mid ab$, niin $p \mid a$ tai $p \mid b$.

Todistus. Seuraa suoraan lauseesta 3.19, sillä jos $p \nmid a$, niin $\text{syt}(p, a) = 1$, jolloin $p \mid b$. □

Korollaari 3.21. Olkoon $k \in \mathbb{Z}_+$. Jos $p \mid q_1 q_2 \ldots q_k$, missä $p, q_1, q_2, \ldots, q_k \in \mathbb{P}$, niin $p = q_j$ jollakin $j \in \mathbb{Z}$, $1 \leq j \leq k$.

Todistus. Osoitetaan induktiolla positiivisten kokonaislukujen $k \geq 2$ suhteen, että jos $p \mid q_1 q_2 \ldots q_k$, niin $p \mid q_j$ jollakin $j \in \mathbb{Z}$, $1 \leq j \leq k$. Tästä seuraa, että $p = q_j$, sillä $p, q_j \in \mathbb{P}$.

Olkoon $k = 2$. Oletetaan, että $p \mid q_1 q_2$. Tällöin korollaarin 3.20 nojalla $p \mid q_1$ tai $p \mid q_2$. Siis $p = q_1$ tai $p = q_2$.

Oletetaan, että väite pätee jollakin $k \in \mathbb{Z}_+$. Osoitetaan, että jos $p \mid q_1 q_2 \dots q_{k+1}$, niin $p \mid q_j$ jollakin $j \in \mathbb{Z}$, $1 \leq j \leq k+1$.

Korollaarin 3.20 nojalla jos $p \mid q_1 q_2 \dots q_{k+1}$, niin $p \mid q_1 q_2 \dots q_k$ tai $p \mid q_{k+1}$. Oletuksen nojalla, jos $p \mid q_1 q_2 \dots q_k$, niin $p = q_i$ jollakin $1 \leq i \leq k$. Jos toisaalta $p \mid q_{k+1}$, niin $p = q_{k+1}$. Siis $p = q_j$ jollakin $1 \leq j \leq k+1$, mikä haluttiin näyttää. □

3.3 Kongruenssi

Määritelmä 3.22. (Kongruenssi) Olkoot $a, b \in \mathbb{Z}$ ja $m \in \mathbb{Z}_+$. Jos $m \mid (a-b)$, niin sanotaan, että a on *kongruentti* luvun b kanssa modulo m. Tästä käytetään merkintää

$$a \equiv b \pmod{m}.$$

Lemma 3.23. Olkoot $a, b, c \in \mathbb{Z}$ ja $m \in \mathbb{Z}_+$. Kongruenssirelaatiolla on seuraavat ominaisuudet:

(i) $a \equiv a \pmod{m}$,

(ii) $a \equiv b \pmod{m}$ jos ja vain jos $b \equiv a \pmod{m}$,

(iii) jos $a \equiv b \pmod{m}$ ja $b \equiv c \pmod{m}$, niin $a \equiv c \pmod{m}$.

Todistus. (i) $a - a = 0$, joten selvästi $m \mid (a-a)$. Siis $a \equiv a \pmod{m}$.

(ii) Jos $m \mid (a-b)$ niin $a - b = km$ jollakin $k \in \mathbb{Z}$. Tällöin $b - a = -km$, joten $m \mid (b-a)$. Siis $b \equiv a \pmod{m}$. Vastaavasti toiseen suuntaan.

(iii) Jos $m \mid (a-b)$ ja $m \mid (b-c)$, niin $a - b = km$ jollakin $k \in \mathbb{Z}$ ja $b - c = lm$ jollakin $l \in \mathbb{Z}$. Tällöin $a - c = km + b - (b - lm) = (k+l)\, m$, eli $m \mid (a-c)$. Siis $a \equiv c \pmod{m}$. □

Lause 3.24. Olkoot $a, b, c, d, k \in \mathbb{Z}$ ja $m \in \mathbb{Z}_+$. Tällöin

(i) $a \equiv b \pmod{m}$ jos ja vain jos $a + k \equiv b + k \pmod{m}$,

(ii) jos $a \equiv b \pmod{m}$, niin $ka \equiv kb \pmod{m}$,

(iii) jos $a \equiv b \pmod{m}$ ja $c \equiv d \pmod{m}$, niin myös $a + c \equiv b + d \pmod{m}$ ja $ac \equiv bd \pmod{m}$,

(iv) jos $a \equiv b \pmod{m}$, niin $a^k \equiv b^k \pmod{m}$, kun $k \geq 1$.

Todistus. (i) Oletuksen nojalla $m \mid (a-b)$. Toisaalta $a-b = a-b+k-k = (a+k)-(b+k)$, joten $m \mid ((a+k)-(b+k))$. Siis $a+k \equiv b+k \pmod{m}$. Vastaavasti toiseen suuntaan.

(ii) Koska $m \mid (a-b)$, lauseen 3.2 nojalla $m \mid k(a-b)$ eli $m \mid (ka-kb)$. Tällöin $ka \equiv kb \pmod{m}$.

(iii) Jos $m \mid (a-b)$ ja $m \mid (c-d)$, niin $a-b = km$ jollakin $k \in \mathbb{Z}$ ja $c-d = lm$ jollakin $l \in \mathbb{Z}$. Nyt

$$a-b+c-d = (a+c)-(b+d) = km+lm = (k+l)\,m,$$

eli $m \mid ((a+c)-(b+d))$. Siis $a+c \equiv b+d \pmod{m}$.

Toisaalta

$$ac = (b+km)(d+lm) = bd+blm+kdm+klm^2,$$

mistä

$$ac-bd = m(bl+kd+klm),$$

joten $m \mid (ac-bd)$. Siis $ab \equiv cd \pmod{m}$.

(iv) Osoitetaan väite todeksi induktiolla positiivisten kokonaislukujen k suhteen.

Oletuksen nojalla $a \equiv b \pmod{m}$, joten väite pätee, kun $k = 1$.

Oletetaan, että väite pätee jollakin $k \in \mathbb{Z}$. Näin ollen $a \equiv b \pmod{m}$ ja $a^k \equiv b^k \pmod{m}$. Lauseen kohdan *(ii)* nojalla tällöin $aa^k \equiv bb^k \pmod{m}$, eli $a^{k+1} \equiv b^{k+1} \pmod{m}$. □

Lause 3.25. Olkoot $a, b, k \in \mathbb{Z}$ ja $m \in \mathbb{Z}_+$. Jos $ka \equiv kb \pmod{m}$, niin $a \equiv b \left(\mathrm{mod}\ \frac{m}{d}\right)$, missä $d = \mathrm{syt}\,(k, m)$.

Todistus. Koska $m \mid (ka-kb)$, niin $ka-kb = k(a-b) = nm$ jollakin $n \in \mathbb{Z}$. Koska $\mathrm{syt}\,(k,m) = d$, niin on olemassa sellaiset $r, s \in \mathbb{Z}$, joilla $k = dr, m = ds$ ja $\mathrm{syt}\,(r,s) = 1$. Yhtälöstä $k(a-b) = nm$ saadaan, että $dr(a-b) = nds$ ja edelleen $r(a-b) = ns$. Koska $\mathrm{syt}\,(r,s) = 1$, lauseen 3.19 nojalla $s \mid (a-b)$. Koska $s = \frac{m}{d}$, niin $a \equiv b \left(\mathrm{mod}\ \frac{m}{d}\right)$. □

Korollaari 3.26. Olkoot $a, b, k \in \mathbb{Z}$ ja $m \in \mathbb{Z}_+$. Jos $ka \equiv kb \pmod{m}$ ja $\mathrm{syt}\,(k,m) = 1$, niin $a \equiv b \pmod{m}$.

Todistus. Seuraa suoraan lauseesta 3.25. □

3.4 Aritmetiikan peruslause

Lause 3.27. Jokainen $n \in \mathbb{Z}, n \geq 2$ voidaan kirjoittaa muodossa

$$n = p_1 p_2 p_3 \dots p_k,$$

missä $p_1, p_2, p_3, \dots, p_k \in \mathbb{P}, k \in \mathbb{Z}_+$. Tämä esitys on yksikäsitteinen jokaiselle n, lukuunottamatta alkulukutekijöiden järjestystä.

Todistus. Lauseen 3.6 nojalla luku n voidaan kirjoittaa muodossa $n = p_1 p_2 \dots p_k$ joillakin $p_1, p_2, \dots, p_k \in \mathbb{P}, k \in \mathbb{Z}_+$. Tehdään vastaoletus, että on olemassa ainakin yksi luku n, jolle tämä esitys ei ole yksikäsitteinen. Olkoon $n_0 \in \mathbb{Z}, n_0 \geq 2$ pienin tällainen luku, eli se voidaan esittää alkulukujen tulona kahdella eri tavalla:

$$n_0 = p_1 p_2 \dots p_k = q_1 q_2 \dots q_m,$$

missä $p_1, p_2, \dots, p_k, q_1, q_2, \dots, q_m \in \mathbb{P}, k, m \in \mathbb{Z}_+$ ja esitysmuodoista ei saada samoja, vaikka ne uudelleenjärjesteltäisiin. Nyt $p_1 \mid q_1 q_2 \dots q_k$, sillä $p_1 \mid n_0$. Korollaarin 3.21 nojalla $p_1 = q_j$ jollakin $j \in \mathbb{Z}, 1 \leq j \leq k$. Jakamalla luku n_0 luvulla p_1 nähdään, että myös luvulla n_0/p_1 on kaksi erilaista esitysmuotoa, mikä on ristiriidassa luvun n_0 määritelmän kanssa. □

3.5 Eukleideen algoritmi

Huomataan aluksi, että kahden luvun suurinta yhteistä tekijää etsittäessä voidaan rajoittua luonnollisten lukujen tarkastelemiseen, sillä $\text{syt}(a, b) = \text{syt}(|a|, |b|)$ kaikilla $a, b \in \mathbb{Z}, b \neq 0$. Kahden luvun suurimman yhteisen tekijän löytämiseksi voidaan menetellä nyt seuraavasti:

Olkoot $a, b \in \mathbb{N}, b \neq 0$. Lukujen a ja b suurimman yhteisen tekijän löytämiseksi käytetään jakoyhtälöä 3.3. Merkitään $a = r_{-1}$ ja $b = r_0$, ja kirjoitetaan r_{-1} jakoyhtälönä $r_{-1} = r_0 q_0 + r_1$ joillakin $q_0, r_1 \in \mathbb{Z}$. Jatkamalla jakoyhtälöiden muodostamista aina edellisen jakoyhtälön jakajan ja jakojäännöksen avulla, saadaan yhtälöt

$$\begin{aligned}
r_{-1} &= r_0 q_0 + r_1, && 0 \leq r_1 < r_0, \\
r_0 &= r_1 q_1 + r_2, && 0 \leq r_2 < r_1, \\
r_1 &= r_2 q_2 + r_3, && 0 \leq r_3 < r_2, \\
&\vdots && \vdots \\
r_{k-2} &= r_{k-1} q_{k-1} + r_k, && 0 \leq r_k < r_{k-1}, \\
r_{k-1} &= r_k q_k, &&
\end{aligned}$$

joillakin $q_0, q_1, \ldots, q_k, r_0, r_1, \ldots, r_k \in \mathbb{Z}$ ja $k \in \mathbb{N}$. Luku k on pienin luonnollinen luku, jolla $r_{k+1} = 0$. Voimme vakuuttua siitä, että tällainen jakojäännös tulee vastaan jossain vaiheessa, sillä jakojäännökset muodostavat jonon $b > r_1 > r_2 > \ldots \geq 0$, missä voi olla korkeintaan b kokonaislukua.

Lause 3.28. Olkoot $a, b, k \in \mathbb{N}, b \neq 0$, ja $r_k \in \mathbb{Z}$ muodostettu kuten yllä. Tällöin

$$r_k = \text{syt}(a, b)\,.$$

Todistus. Merkitään $a = r_{-1}$ ja $b = r_0$. Käymällä yllä muodostetut jakoyhtälöt ensimmäisestä viimeiseen, lauseen 3.16 kohdan *(ii)* nojalla nähdään, että

$$\text{syt}(a, b) = \text{syt}(r_{-1}, r_0) = \text{syt}(r_0, r_1) = \text{syt}(r_1, r_2) = \ldots = \text{syt}(r_{k-1}, r_k) = \text{syt}(r_k, 0) = r_k.$$

□

3.6 Lineaariset Diofantoksen yhtälöt

Lause 3.29. Olkoot $a, b, c \in \mathbb{Z}$ ja $a \neq 0, b \neq 0$. Yhtälöllä $ax + by = c$ on kokonaislukuratkaisu jos ja vain jos $d \mid c$, missä $d = \text{syt}(a, b)$. Yhtälön ratkaisut ovat täsmälleen lukuparit

$$\begin{cases} x = x_0 + \frac{bt}{d}, \\ y = y_0 - \frac{at}{d}, \end{cases}$$

missä $t \in \mathbb{Z}$ ja x_0, y_0 on yksi ratkaisu.

Todistus. Todistetaan aluksi lauseen ensimmäinen väite.

Oletetaan ensin, että Diofantoksen yhtälöllä $ax + by = c$ on jokin kokonaislukuratkaisu $x = x_0$ ja $y = y_0$. Koska $d \mid a$ ja $d \mid b$, on olemassa sellaiset $k, l \in \mathbb{Z}$, joilla $a = kd$ ja $b = ld$. Näin ollen $ax_0 + by_0 = c$ voidaan kirjoittaa muodossa $kdx_0 + ldy_0 = c$, mistä edelleen $d\,(kx_0 + ly_0) = c$. Jaollisuuden määritelmän nojalla $d \mid c$.

Osoitetaan seuraavaksi toinen suunta ja oletetaan, että $d \mid c$. Tällöin on olemassa $m \in \mathbb{Z}$, jolla $c = md$. Korollaarin 3.15 nojalla $d = ax_0 + by_0$ joillakin $x_0, y_0 \in \mathbb{Z}$. Nyt

$$c = md = m\,(ax_0 + by_0) = a\,(mx_0) + b\,(my_0)\,.$$

Näin ollen Diofantoksen yhtälöllä $ax + by = c$ on kokonaislukuratkaisu $x = mx_0, y = my_0$.

Todistetaan nyt lauseen toinen väite. Voimme jakaa yhtälön $ax + by = c$ puolittain luvulla d, jolloin saadaan

$$\frac{a}{d} \cdot x + \frac{b}{d} \cdot y = \frac{c}{d}.$$

Lauseen 3.16 kohdan *(i)* nojalla $\text{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1$. Jos yhtälön yksi ratkaisu on kokonaisluvut x_0 ja y_0, muille ratkaisuille x ja y pätee, että

$$\frac{a}{d} \cdot x_0 + \frac{b}{d} \cdot y_0 = \frac{a}{d} \cdot x + \frac{b}{d} \cdot y, \text{ mistä } \frac{a}{d}(x - x_0) = \frac{b}{d}(y_0 - y).$$

Koska

$$\text{syt}\left(\frac{a}{d}, \frac{b}{d}\right) = 1 \text{ ja } \frac{b}{d} \mid \left(\frac{a}{d}(x - x_0)\right),$$

niin

$$\frac{b}{d} \mid (x - x_0).$$

Tällöin

$$x - x_0 = \frac{b}{d} \cdot t \text{ jollakin } t \in \mathbb{Z}, \text{ eli } x = x_0 + \frac{bt}{d}.$$

Muuttujan y arvo saadaan sijoittamalla

$$x - x_0 = \frac{bt}{d}$$

yhtälöön

$$\frac{a}{d}(x - x_0) = \frac{b}{d}(y_0 - y).$$

Tällöin

$$\begin{aligned} \frac{a}{d} \cdot \frac{bt}{d} &= \frac{b}{d}(y_0 - y) \\ \frac{at}{d} &= y_0 - y \\ y &= y_0 - \frac{at}{d}. \end{aligned}$$

Todetaan, että lukupari

$$\begin{cases} x = x_0 + \frac{bt}{d}, \\ y = y_0 - \frac{at}{d}, \end{cases}$$

toteuttaa Diofantoksen yhtälön $ax + by + c$ kaikilla $t \in \mathbb{Z}$, sillä

$$\begin{aligned} ax + by &= a\left(x_0 + \frac{bt}{d}\right) + b\left(y_0 - \frac{at}{d}\right) \\ &= ax_0 + by_0 + \frac{abt}{d} - \frac{abt}{d} \\ &= c. \end{aligned}$$

□

3.7 Laajennettu Eukleideen algoritmi

Laajennetun Eukleideen algoritmin notaatio ja periaate nojaa lähteeseen [3].

Olkoot $a, b \in \mathbb{N}, b \neq 0$. Eukleideen algoritmia voi hyödyntää muotoa

$$ax + by = \text{syt}\,(a, b)$$

olevan Diofantoksen yhtälön jonkin ratkaisun löytämisessä. Kuten algoritmin kuvailussa 3.5, kirjoitetaan $r_{i-1} = r_i q_i + r_{i+1}$, missä $r_{-1} = a, r_0 = b$ ja $r_i, q_i \in \mathbb{Z}, i \in \mathbb{N}, i \leq k$, ja $k \in \mathbb{N}$ on pienin luonnollinen luku, jolla $r_{k+1} = 0$. Tällöin lauseen 3.28 nojalla $r_k = \text{syt}(a, b)$. Määritellään lisäksi kokonaisluvut u_i ja v_i seuraavasti:

$$\begin{aligned} &u_{-1} = 1, && u_0 = 0, && u_{i+1} = u_{i-1} - u_i q_i, \\ &v_{-1} = 0, && v_0 = 1, && v_{i+1} = v_{i-1} - v_i q_i. \end{aligned} \tag{3.30}$$

Lause 3.31. Olkoot $a, b, k \in \mathbb{N}, b \neq 0$. Olkoot $u_k, v_k \in \mathbb{Z}$ muodostettu kuten yllä. Tällöin $au_k + bv_k = \text{syt}\,(a, b)$.

Todistus. Lauseen 3.28 perusteella riittää näyttää, että $au_{i-1} + bv_{i-1} = r_{i-1}$ kaikilla $i \in \mathbb{N}, i \leq k$. Osoitetaan tämä induktiolla luonnollisten lukujen i suhteen.

Kun $i = 0$, niin $au_{-1} + bv_{-1} = a = r_{-1}$, ja kun $i = 1$, niin $au_0 + bv_0 = b = r_0$.

Oletetaan nyt, että $au_{i-1} + bv_{i-1} = r_{i-1}$ ja $au_i + bv_i = r_i$. Nyt

$$\begin{aligned} au_{i+1} + bv_{i+1} &= a\,(u_{i-1} - u_i q_i) + b\,(v_{i-1} - v_i q_i) \\ &= au_{i-1} + bv_{i-1} - (au_i + bv_i)\,q_i \\ &= r_{i-1} - r_i q_i \\ &= r_{i+1}. \end{aligned}$$

□

3.8 Kiinalainen jäännöslause

Määritelmä 3.32. Olkoot $a, b, x \in \mathbb{Z}, m \in \mathbb{Z}_+$. *Lineaariseksi kongruenssiksi* kutsutaan yhtälöä, joka on muotoa

$$ax \equiv b\,(\text{mod } m)\,.$$

Yhtälön ratkaisu on kokonaisluku x_0, jolla

$$ax_0 \equiv b\,(\text{mod } m)\,.$$

Huomataan, että x_0 on lineaarisen kongruenssin $ax \equiv b \pmod{m}$ ratkaisu jos ja vain jos $m \mid (ax_0 - b)$, eli $ax_0 - b = my_0$ jollakin $y_0 \in \mathbb{Z}$. Näin ollen kaikkien sellaisten kokonaislukujen löytäminen, jotka ratkaisevat jonkin lineaarisen kongruenssin, on samanlainen ongelma kuin Diofantoksen yhtälön $ax_0 - my_0 = b$ ratkaisuiden x_0 ja y_0 löytäminen. Erona on se, että lineaaristen kongruenssin tapauksessa ratkaisuja x_0 ja x_0' pidetään eri ratkaisuina vain, jos ne eivät ole keskenään kongruentteja modulo m.

Lause 3.33. Olkoot $a, b, x \in \mathbb{Z}, m \in \mathbb{Z}_+$. Lineaarisella kongruenssilla $ax \equiv b \pmod{m}$ on ratkaisu jos ja vain jos $d \mid b$, missä $d = \text{syt}(a, m)$. Jos $d \mid b$, niin lineaarisella kongruenssilla on täsmälleen d ratkaisua, jotka eivät ole keskenään kongruentteja modulo m.

Todistus. Kuten edellä todettiin, lineaarisella kongruenssilla $ax \equiv b \pmod{m}$ on samat ratkaisut, kuin Diofantoksen yhtälöllä $ax - my = b$, missä $y \in \mathbb{Z}$. Lauseen 3.29 nojalla yhtälöllä on kokonaislukuratkaisuja jos ja vain jos $d \mid b$, ja yhtälön ratkaisuja ovat lukuparit

$$\begin{cases} x = x_0 + \frac{mt}{d}, \\ y = y_0 + \frac{at}{d}, \end{cases}$$

missä $t \in \mathbb{Z}$ ja $x_0, y_0 \in \mathbb{Z}$ on yksi ratkaisu.

Tarkastellaan yhtälön toteuttavia muuttujan x arvoja, kun $t = 0, 1, 2, \ldots, d-1$:

$$x_0, x_0 + \frac{m}{d}, x_0 + \frac{2m}{d}, \ldots, x_0 + \frac{(d-1)\,m}{d}. \tag{3.34}$$

1) Näytetään aluksi, että näistä kokonaisluvuista mitkään eivät ole keskenään kongruentteja modulo m. Tehdään vastaoletus, että

$$x_0 + \frac{nt_1}{d} \equiv x_0 + \frac{mt_2}{d} \pmod{m},$$

missä $0 \le t_1 < t_2 \le d - 1$. Lauseen 3.24 kohdan *(i)* nojalla nyt

$$\frac{mt_1}{d} \equiv \frac{mt_2}{d} \pmod{m}.$$

Nyt $\text{syt}\left(\frac{m}{d}, m\right) = \frac{m}{d}$, joten lauseen 3.25 nojalla

$$t_1 \equiv t_2 \pmod{d},$$

joten $d \mid (t_2 - t_1)$. Tämä on ristiriita, sillä $0 < t_2 - t_1 < d$.

2) Näytetään seuraavaksi, että kaikki lineaarisen kongruenssin ratkaisut ovat kongruentteja jonkun ratkaisun (3.34) kanssa. Jakoyhtälön 3.3 avulla voimme ilmaista kokonaisluvun t muodossa $t = kd + r$ joillakin $k, r \in \mathbb{Z}, 0 \leq r \leq d-1$. Näin ollen

$$\begin{aligned} x_0 + \frac{mt}{d} &= x_0 + \frac{m\,(kd+r)}{d} \\ &= x_0 + mq + \frac{mr}{d} \\ &\equiv x_0 + \frac{mr}{d} \pmod{m}, \end{aligned}$$

missä $x_0 + \frac{mr}{d}$ on yksi kokonaisluvuista (3.34).

Kohtien 1) ja 2) nojalla lineaarisella kongruenssilla on täsmälleen d ratkaisua, jotka eivät ole keskenään kongruentteja modulo m. □

Korollaari 3.35. (Modulaariaritmetiikan käänteisluku) Olkoot $a, b, x \in \mathbb{Z}, m \in \mathbb{Z}_+$. Jos $\text{syt}(a, m) = 1$, niin lineaarisella kongruenssilla $ax \equiv b \pmod{m}$ on yksikäsitteinen ratkaisu modulo m.

Todistus. Seuraa suoraan lauseesta 3.33. □

Lause 3.36. (Kiinalainen jäännöslause) Olkoon $r \in \mathbb{Z}_+$. Olkoot $m_1, m_2, \ldots m_r$ positiivisia kokonaislukuja niin, että $\text{syt}(m_i, m_j) = 1$ kaikilla $i, j \in \mathbb{Z}, 1 \leq i < j \leq r$. Olkoot $x, a_1, a_2, \ldots a_r \in \mathbb{Z}$. Tällöin lineaaristen kongruenssien systeemillä

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \quad \vdots \\ x \equiv a_r \pmod{m_r} \end{cases}$$

on olemassa ratkaisu $x_0 \in \mathbb{Z}$, joka on yksikäsitteinen modulo $M = m_1 m_2 \ldots m_r$.

Todistus. Olkoon $k \in \mathbb{Z}_+, k \leq r$. Määritellään luku M_k seuraavasti:

$$M_k = \frac{M}{m_k} = m_1 \ldots m_{k-1} m_{k+1} \ldots m_r.$$

Oletuksen nojalla $\text{syt}(M_k, m_k) = 1$. Korollaarin 3.35 nojalla tällöin lineaarisella kongruenssilla $M_k x \equiv 1 \pmod{m_k}$ on yksikäsitteinen ratkaisu x_k modulo m_k.

Osoitetaan nyt, että kokonaisluku

$$x_0 = a_1 M_1 x_1 + a_2 M_2 x_2 + \ldots + a_r M_r x_r \tag{3.37}$$

on annetun lineaaristen kongruenssien systeemin ratkaisu. Huomataan, että $m_k \mid M_i$, kun $k \neq i$, joten $M_i \equiv 0 \pmod{m_k}$. Näin ollen

$$x_0 \equiv a_k M_k x_k \pmod{m_k}.$$

Toisaalta $M_k x_k \equiv 1 \pmod{m_k}$, joten

$$x_0 \equiv a_k \cdot 1 \equiv a_k \pmod{m_k}.$$

On osoitettu, että annetulla kongruenssisysteemillä on olemassa ratkaisu x_0. Osoitetaan vielä, että ratkaisu x_0 on yksikäsitteinen modulo m. Oletetaan, että x_0' on jokin toinen kokonaisluku, joka toteuttaa annetun kongruenssisysteemin. Tällöin

$$x_0 \equiv a_k \equiv x_0' \pmod{m_k},$$

eli $m_k \mid (x_0 - x_0')$ kaikilla $1 \leq k \leq r$. Koska $\text{syt}(m_i, m_j) = 1$ kaikilla $i \neq j$, niin lauseen 3.18 nojalla $m_1 m_2 \ldots m_k \mid (x_0 - x_0')$. Siis $x_0 \equiv x_0' \pmod{m}$. □

3.9 Eulerin φ-funktio

Määritelmä 3.38. (Eulerin φ-funktio) Olkoon $m \in \mathbb{Z}_+$. Funktio $\varphi : \mathbb{Z}_+ \to \mathbb{Z}_+$ määritellään niin, että funktion arvo $\varphi(m)$ on niiden positiivisten kokonaislukujen $k \leq m$ lukumäärä, jotka ovat yhteistekijättömiä luvun m kanssa.

Huomataan, että $\varphi(1) = 1$, ja että $\varphi(m) < m$, kun $m > 1$, sillä $\text{syt}(m, m) = 1$ jos ja vain jos $m = 1$.

Korollaari 3.39. $\varphi(m) = m - 1$ jos ja vain jos $m \in \mathbb{P}$.

Todistus. Oletetaan aluksi, että $\varphi(m) = m - 1$. Tehdään vastaoletus, että $m \notin \mathbb{P}$. Tällöin on olemassa $k \in \mathbb{Z}_+, k < m$, jolla $k \mid m$. Seuraa, että on olemassa ainakin kaksi enintään luvun m suuruista positiivista kokonaislukua, jotka eivät ole yhteistekijättömiä luvun m kanssa, nimittäin k ja luku m itse, joten $\varphi(m) \leq m - 2$, mikä on ristiriita. Siis $m \in \mathbb{P}$.

Oletetaan seuraavaksi, että $m \in \mathbb{P}$. Tällöin kaikki kokonaislukua m pienemmät positiiviset kokonaisluvut ovat sen kanssa yhteistekijättömiä, joten $\varphi(m) = n - 1$. □

Lause 3.40. Olkoot $k, m \in \mathbb{Z}_+, \text{syt}(k, m) = 1$. Tällöin $\varphi(km) = \varphi(k)\,\varphi(m)$.

Todistus. Koska $\varphi(1) = 1$, niin selvästi väite pätee, kun $k = 1$ tai $m = 1$, joten voidaan olettaa, että $k > 1$ ja $m > 1$. Järjestetään luvut $1, 2, \ldots, km$ seuraavaksi taulukoksi, jossa on k saraketta ja jokaisessa sarakkeessa m kokonaislukua.

1	2	$\dots$	r	$\dots$	k
$k+1$	$k+2$	$\dots$	$k+r$	$\dots$	$2k$
$2k+1$	$2k+2$	$\dots$	$2k+r$	$\dots$	$3k$
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$(m-1)\,k+1$	$(m-1)\,k+2$	$\dots$	$(m-1)\,k+r$	$\dots$	km

Nyt $\varphi(km)$ on niiden taulukon alkioiden lukumäärä, jotka ovat luvun km kanssa yhteistekijättömiä. Jos taulukon alkio on yhteistekijätön luvun km kanssa, se on myös yhteistekijätön lukujen k ja m kanssa. Olkoon $r \in \mathbb{Z}_+, r \leq k$. Tarkastellaan r:nnen sarakkeen alkiota

$$n_l = lk + r,$$

missä $l \in \mathbb{N}, l < m$. Lauseen 3.16 kohdan *(ii)* nojalla $\operatorname{syt}(n_l, k) = \operatorname{syt}(k, r)$. Toisin sanoen, jos k on yhteistekijätön luvun r kanssa, se on yhteistekijätön kaikkien r:nnen sarakkeen alkioiden kanssa. Eulerin φ-funktion määritelmän 3.38 nojalla tällaisia sarakkeita on taulukossa $\varphi(k)$.

Näytetään seuraavaksi, että jokaisessa näistä sarakkeista on täsmälleen $\varphi(m)$ alkiota, jotka ovat yhteistekijättömiä luvun m kanssa. Olkoon $\operatorname{syt}(k, r) = 1$. Tarkastellaan r:nnen sarakkeen alkioita

$$n_i = ik + r \text{ ja } n_j = jk + r,$$

missä $i, j \in \mathbb{N}, 0 \leq i < j < m$. Osoitetaan, että alkiot n_i ja n_j eivät ole keskenään kongruentteja modulo m. Näytetään tämä tekemällä vastaoletus, että $n_i \equiv n_j \pmod{m}$, jolloin

$$ik + r \equiv jk + r \pmod{m}.$$

Mutta tällöin $ik \equiv jk \pmod{m}$. Koska $\operatorname{syt}(k, m) = 1$, niin lauseen 3.26 nojalla $i \equiv j \pmod{m}$, mikä on ristiriita. Koska mitkään r:nnen sarakkeen alkiot eivät ole keskenään kongruentteja modulo m, ne ovat jossain järjestyksessä kongruentteja lukujen $1, 2, \dots, m$ kanssa modulo m. Olkoon $t \in \mathbb{Z}_+, t \leq m$ ja $n' \in \mathbb{Z}_+, r \leq n' < mk + r$ se r:nnen sarakkeen alkio, jolla $t \equiv n' \pmod{m}$. Tällöin $\operatorname{syt}(n', m) = 1$ jos ja vain jos $\operatorname{syt}(t, m) = 1$. Siis r:nnessä sarakkeessa luvun m kanssa yhteistekijättömiä alkioita on $\varphi(m)$. Tällöin koko taulukossa lukujen k ja m kanssa yhteistekijättömiä alkioita on täsmälleen $\varphi(k)\,\varphi(m)$, mikä haluttiin näyttää. □

Lemma 3.41. Olkoot $a \in \mathbb{Z}, m \in \mathbb{Z}_+$ ja $\operatorname{syt}(a, m) = 1$. Jos $a_1, a_2, \dots, a_{\varphi(m)}$ ovat lukua m pienemmät ja luvun m kanssa yhteistekijättömät positiiviset kokonaisluvut, niin luvut

$$aa_1, aa_2, \dots, aa_{\varphi(m)}$$

ovat jossain järjestyksessä kongruentteja lukujen $a_1, a_2, \ldots a_{\varphi(m)}$ kanssa.

Todistus. 1) Huomataan, etteivät mitkään luvuista $aa_1, aa_2, \ldots, aa_{\varphi(m)}$ ole keskenään kongruentteja modulo m, sillä jos $aa_i \equiv aa_j \pmod{m}, i, j \in \mathbb{Z}, 1 \leq i < j \leq \varphi(m)$, niin lauseen 3.26 nojalla $a_i \equiv a_j \pmod{m}$ mikä on ristiriita. Tarkemmin, koska $\text{syt}(a, m) = \text{syt}(a_i, m) = 1$ kaikilla i, niin myös $\text{syt}(aa_i, m) = 1$ kaikilla i. Näin ollen kaikki luvut aa_i ovat yhteistekijättömiä luvun m kanssa.

2) Tarkastellaan seuraavaksi jotain lukua aa_i. Nyt on olemassa $b \in \mathbb{N}, b < m$ niin, että $aa_i \equiv b \pmod{m}$. Koska

$$\text{syt}(b, m) = \text{syt}(aa_i, m) = 1,$$

niin luvun b täytyy olla yksi luvuista $a_1, a_2, \ldots, a_{\varphi(m)}$.

Kohtien 1) ja 2) perusteella luvut $a_1, a_2, \ldots, a_{\varphi(m)}$ ja $aa_1, aa_2, \ldots, aa_{\varphi(m)}$ ovat jossain järjestyksessä kongruentit modulo m. □

Lause 3.42. (Eulerin lause) Olkoot $a \in \mathbb{Z}, m \in \mathbb{Z}_+$. Jos $\text{syt}(a, m) = 1$, niin

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Todistus. Väite triviaalisti pätee, kun $m = 1$, joten oletetaan, että $m > 1$. Olkoot $a_1, a_2, \ldots, a_{\varphi(m)}$ lukua m pienempiä, luvun m kanssa yhteistekijättömiä positiivisia kokonaislukuja. Koska $\text{syt}(a, m) = 1$, niin lemman 3.41 nojalla luvut $aa_1, aa_2, \ldots, aa_{\varphi(m)}$ ovat jossain järjestyksessä kongruentteja lukujen $a_1, a_2, \ldots, a_{\varphi(m)}$ kanssa modulo m. Näin ollen

$$\begin{aligned} aa_1 &\equiv a_1' \pmod{m} \\ aa_2 &\equiv a_2' \pmod{m} \\ \vdots &\qquad \vdots \\ aa_{\varphi(m)} &\equiv a_{\varphi(m)}' \pmod{m}, \end{aligned}$$

missä $a_1', a_2', \ldots, a_{\varphi(m)}'$ ovat luvut $a_1, a_2, \ldots, a_{\varphi(m)}$ jossain järjestyksessä. Kertomalla nämä $\varphi(m)$ kongruenssia puolittain keskenään, saadaan

$$\begin{aligned} aa_1 \cdot aa_2 \cdot \ldots \cdot aa_{\varphi(m)} &\equiv a_1' a_2' \ldots a_{\varphi(m)}' \\ &\equiv a_1 a_2 \ldots a_{\varphi(m)} \pmod{m} \end{aligned}$$

ja näin ollen

$$a^{\varphi(m)} \left(a_1 a_2 \ldots a_{\varphi(m)}\right) \equiv a_1 a_2 \ldots a_{\varphi(m)} \pmod{m}. \tag{3.43}$$

Koska $\text{syt}(a_i, m) = 1$ kaikilla $i \in \mathbb{Z}, 1 \leq i \leq \varphi(m)$, niin $\text{syt}\big(a_1 a_2 \dots a_{\varphi(m)}, m\big) = 1$. Lauseen 3.26 nojalla voimme jakaa aiemman kongruenssin (3.43) puolittain yhteisellä tekijällä $a_1 a_2 \dots a_{\varphi(m)}$, jolloin saamme

$$a^{\varphi(m)} \equiv 1 \,(\text{mod } m)\,,$$

mikä haluttiin näyttää. □

Lause 3.44. (Fermat'n pieni lause) Olkoon $a \in \mathbb{Z}$. Jos $p \in \mathbb{P}$ ja $p \nmid a$, niin

$$a^{p-1} \equiv 1 \,(\text{mod } p)\,.$$

Todistus. Koska $p \in \mathbb{P}$, niin lauseen 3.39 nojalla $\varphi(p) = p - 1$. Lisäksi koska $p \nmid a$, niin $\text{syt}(p, a) = 1$, joten Eulerin lauseen 3.42 nojalla

$$a^{p-1} = a^{\varphi(p)} \equiv 1 \,(\text{mod } p)\,.$$ □

3.10 RSA-kryptografia

Vuonna 1977 R. Rivest, A. Shamir ja L. Adleman kehittivät yksinkertaiseen lukuteoriaan perustuvan salausmenetelmän, joka nimettiin kehittäjiensä sukunimien ensimmäisten kirjainten mukaan RSA:ksi [6]. RSA on laajalti käytetty algoritmi viestien salaamiseen ja purkamiseen. RSA-algoritmi on niin kutsuttu julkisen avaimen salausalgoritmi, eli se perustuu kahden avaimen käyttöön, joista toinen on julkinen ja toinen yksityinen. Julkista avainta ei nimensä mukaan tarvitse pitää salassa, vaan sen voi jakaa kelle tahansa. Toisaalta yksityinen avain täytyy pitää salassa, tai kuka tahansa voi purkaa algoritmilla salattuja viestejä.

Algoritmin turvallisuus perustuu ajatukseen siitä, että yhdistettyjen lukujen, joilla on tekijöinään suuria alkulukuja, tekijöihin jakaminen on työlästä. Toisaalta ei ole todistettu, etteikö olisi mahdollista keksiä sellainen algoritmi, joka pystyisi tehokkaasti tämän toteuttamaan, mutta sellaista ei ole löydetty [29].

Algoritmin idea on seuraava [6]:

1. Valitse kaksi suurta eri alkulukua p ja q.
2. Laske $n = pq$.
3. Laske $\varphi(n) = (p-1)(q-1)$.
4. Valitse $e \in \mathbb{Z}$ niin, että $1 < e < \varphi(n)$ ja $\text{syt}(e, \varphi(n)) = 1$.

5. Valitse $d \in \mathbb{Z}$ niin, että $1 < d < \varphi(n)$ ja $de \equiv 1 \pmod{\varphi(n)}$.

Nyt algoritmin salainen avain muodostuu lukuparista (n, d) ja julkinen avain puolestaan lukuparista (n, e).

Oletetaan esimerkiksi, että henkilö A olisi tehnyt ylläolevat vaiheet, ja antaisi julkisen avaimen (n, e) henkilölle B, joka haluaa lähettää A:lle viestin M. Aluksi salattu viesti täytyy kuvataan jonkin valitun protokollan[1] mukaisesti positiiviseksi kokonaisluvuksi $m < n$. Nimittäin, jos $m > n$, niin viestiä m on mahdotonta erottaa mistään muusta kokonaisluvusta, joka on sen kanssa kongruentti modulo n. Vaaditaan lisäksi, että $\text{syt}\,(m, n) = 1$, sillä jos näin ei ole, niin salauksen turvallisuus vaarantuu. Tällöin nimittäin lukujen m ja n suurin yhteinen tekijä on p tai q, jolloin luvun n tekijät saadaan tehokkaasti selvitettyä.

Kun viesti on muutettu sopivaksi luvuksi m, B voi muodostaa salatun viestin $c \in \mathbb{Z}_+, c < n$ ehdosta

$$c \equiv m^e \pmod{n}.$$

Seuraavaksi B voi lähettää salatun viestin c henkilölle A. Henkilö A saa laskettua alkuperäisen viestin ehdosta

$$m \equiv c^d \pmod{n}.$$

Näytetään vielä, että algoritmi toimii. Luvun d määrittelyn perusteella $\varphi\,(n) \mid (de - 1)$, mistä $de = k\varphi\,(n) + 1$ jollakin $k \in \mathbb{Z}$. Tällöin Eulerin lauseen 3.42 nojalla

$$c^d \equiv m^{de} = m^{k\varphi(n)+1} = m^{k\varphi(n)} \cdot m \equiv 1 \cdot m \equiv m \pmod{n}.$$

3.11 Primitiiviset juuret

Määritelmä 3.45. Olkoot $a, m \in \mathbb{Z}_+$ ja $\text{syt}(a, m) = 1$. Luvun a *kertaluku modulo* m on pienin positiivinen kokonaisluku k, jolla

$$a^k \equiv 1 \pmod{m}.$$

Tällöin merkitään $\text{ord}_m(a) = k$.

Huomataan, että jokaiselle luvulle $a \in \mathbb{Z}_+$ löytyy kertaluku modulo m kaikilla $m \in \mathbb{Z}_+$, jolla $\text{syt}(a, m) = 1$, sillä Eulerin lauseen 3.42 nojalla luku $k = \varphi(m)$ toteuttaa ehdon $a^k \equiv 1 \pmod{m}$.

[1] Luvussa 5.5 RSA-algoritmin käsittelyn yhteydessä kommentoidaan lyhyesti käytäntöä käsitellä merkkijonoja lukuna, ja tarjotaan mahdollinen tapa tämän demonstrointiin osana algoritmin harjoittelua.

Lause 3.46. Olkoot $a, k, h, m \in \mathbb{Z}_+, \text{syt}(a, m) = 1$ ja $\text{ord}_m(a) = k$. Tällöin $a^h \equiv 1 \pmod{m}$ jos ja vain jos $k \mid h$. Erityisesti, $k \mid \varphi(m)$.

Todistus. Oletetaan aluksi, että $a^h \equiv 1 \pmod{m}$ jollain h. Jakoyhtälön 3.3 nojalla on olemassa sellaiset $q \in \mathbb{Z}_+$ ja $r \in \mathbb{N}, r < k$, joilla $h = qk + r$. Saadaan, että $a^h = a^{qk+r} = \left(a^k\right)^q a^r$. Oletuksen nojalla $a^h \equiv 1 \pmod{m}$ ja $a^k \equiv 1 \pmod{m}$, joten nyt

$$1 \equiv a^h \equiv \left(a^k\right)^q a^r \equiv 1^q a^r \equiv a^r \pmod{m}.$$

Koska $r < k$, niin $r = 0$, sillä muutoin syntyisi ristiriita sen oletuksen kanssa, että k on pienin positiivinen kokonaisluku, jolla $a^k \equiv 1 \pmod{m}$. Näin ollen $h = qk$, eli $k \mid h$.

Oletetaan seuraavaksi, että $k \mid h$. Tällöin $h = jk$ jollakin $j \in \mathbb{Z}_+$. Koska $a^k \equiv 1 \pmod{m}$, niin lauseen 3.24 kohdan *(iv)* nojalla $\left(a^k\right)^j \equiv 1^j \pmod{m}$, mistä $a^h \equiv 1 \pmod{m}$. □

Lause 3.47. Olkoot $a, k, m, i, j \in \mathbb{Z}_+, \text{syt}(a, m) = 1$ ja $\text{ord}_m(a) = k$. Nyt $a^i \equiv a^j \pmod{m}$ jos ja vain jos $i \equiv j \pmod{k}$.

Todistus. Oletetaan aluksi, että $a^i \equiv a^j \pmod{m}$, missä $i \geq j$. Koska $\text{syt}(a, m) = 1$, niin myös $\text{syt}(a^j, m) = 1$, joten korollaarin 3.26 nojalla saadaan, että $a^{i-j} \equiv 1 \pmod{m}$. Lauseen 3.46 nojalla tällöin $k \mid (i - j)$, eli $i \equiv j \pmod{k}$.

Oletetaan seuraavaksi, että $i \equiv j \pmod{k}$. Tällöin $k \mid (i - j)$, mistä saadaan, että $i = j + qk$ jollakin $q \in \mathbb{Z}_+$. Tällöin

$$a^i = a^{j+qk} = a^j \left(a^k\right)^q.$$

Oletuksen nojalla $a^k \equiv 1 \pmod{m}$, joten nyt $a^i = a^j \left(a^k\right)^q \equiv a^j 1^q \equiv a^j \pmod{m}$. □

Korollaari 3.48. Olkoot $a, k, m \in \mathbb{Z}_+$ ja $\text{syt}(a, m) = 1$. Jos $\text{ord}_m(a) = k$, niin mitkään kokonaisluvuista $a, a^2, \ldots, a^k$ eivät ole kongruentteja keskenään modulo m.

Todistus. Jos $a^i \equiv a^j$ joillakin $i, j \in \mathbb{Z}, 1 \leq j \leq i \leq k$, niin lauseen 3.47 nojalla $i \equiv j \pmod{k}$, eli toisin sanoen $k \mid (i - j)$. Koska $0 \leq i - j < k$, niin tällöin $i - j = 0$ eli $i = j$. □

Lause 3.49. Olkoot $a, k, h, m \in \mathbb{Z}_+$. Jos $\text{syt}(a, m) = 1$ ja $\text{ord}_m(a) = k$, niin

$$\text{ord}_m\left(a^h\right) = \frac{k}{\text{syt}(h, k)}.$$

Todistus. Olkoon $d = \text{syt}(h,k)$, $h' = \frac{h}{d}$ ja $k' = \frac{k}{d}$. Tällöin lauseen 3.16 kohdan *(i)* nojalla $\text{syt}(h',k') = 1$. Nyt

$$\left(a^h\right)^{k'} = \left(a^{h'd}\right)^{\frac{k}{d}} = \left(a^k\right)^{h'} \equiv 1 \,(\text{mod } m)\,.$$

Oletetaan, että $\text{ord}_m\left(a^h\right) = r$. Tällöin lauseen 3.46 nojalla $r \mid k'$. Koska toisaalta $\text{ord}_m(a) = k$, niin kongruenssista

$$a^{hr} = \left(a^h\right)^r \equiv 1 \,(\text{mod } m)$$

seuraa, että $k \mid hr$, eli $k'd \mid h'dr$, mistä edelleen $k' \mid h'r$. Koska $\text{syt}(h',k') = 1$, niin $k' \mid r$. Koska myös $r \mid k'$, niin

$$r = k' = \frac{k}{d} = \frac{k}{\text{syt}(h,k)},$$

mikä haluttiin näyttää. □

Korollaari 3.50. Olkoot $a,k,h,m \in \mathbb{Z}_+, \text{syt}(a,m) = 1$ ja $\text{ord}_m(a) = k$. Nyt $\text{ord}_m\left(a^h\right) = k$ jos ja vain jos $\text{syt}(h,k) = 1$.

Todistus. Seuraa suoraan lauseesta 3.49. □

Määritelmä 3.51. (Primitiiviset juuret) Olkoot $a,m \in \mathbb{Z}_+$. Jos $\text{syt}(a,m) = 1$ ja $\text{ord}_m(a) = \varphi(m)$, niin luku a on *primitiivinen juuri* modulo m.

Lause 3.52. Olkoot $a,m \in \mathbb{Z}_+, \text{syt}(a,m) = 1$ ja $a_1, a_2, \ldots, a_{\varphi(m)}$ lukua m pienemmät positiiviset kokonaisluvut, jotka ovat keskenään jaottomia luvun m kanssa. Jos a on primitiivinen juuri modulo m, niin

$$a, a^2, \ldots, a^{\varphi(m)}$$

ovat jossain järjestyksessä kongruentit lukujen $a_1, a_2, \ldots, a_{\varphi(m)}$ kanssa modulo m.

Todistus. Koska $\text{syt}(a,m) = 1$, niin myös $\text{syt}\left(a^k,m\right) = 1$ kaikilla k. Näin ollen jokaiselle a^k pätee, että $a^k \equiv a_i \,(\text{mod } m)$ jollain $i \in \mathbb{Z}, 1 \leq 1 \leq \varphi(m)$. Toisaalta korollaarin 3.48 nojalla mitkään luvuista $a, a^2, \ldots, a^{\varphi(m)}$ eivät ole kongruentteja keskenään modulo m, joten näiden lukujen täytyy olla kongruentteja lukujen $a_1, a_2, \ldots, a_{\varphi(m)}$ kanssa jossain järjestyksessä. □

Korollaari 3.53. Olkoon $m \in \mathbb{Z}_+$. Primitiivisiä juuria modulo m, jotka eivät ole keskenään kongruentteja modulo m, on olemassa täsmälleen 0 tai $\varphi(\varphi\,(m))$.

Todistus. Näytetään, että jos on olemassa primitiivinen juuri modulo m, niin primitiivisiä juuria modulo m, jotka eivät ole keskenään kongruentteja modulo m on $\varphi(\varphi(m))$. Olkoon $a \in \mathbb{Z}_+$. Oletetaan, että a on primitiivinen juuri modulo m. Lauseen 3.52 nojalla mikä tahansa toinen luvun m primitiivinen juuri löytyy lukujen $a, a^2, \ldots, a^{\varphi(m)}$ joukosta. Mutta sellaisia lukuja $a^k, k \in \mathbb{Z}_+, k \leq \varphi(m)$, joilla $\operatorname{ord}_m\left(a^k\right) = \varphi(m)$ on korollaarin 3.50 nojalla saman verran kuin lukuja k, joilla $\operatorname{syt}(k, \varphi(m)) = 1$. Tällaisia lukuja on tasan $\varphi(\varphi(m))$ kappaletta, joten luvulla m on $\varphi(\varphi(m))$ primitiivistä juurta. □

Esitellään vielä yksi olennainen lause, jonka todistus tässä tutkielmassa kuitenkin sivuutetaan liian työläänä.

Lause 3.54. Olkoon $m \in \mathbb{Z}_+$. Primitiivinen juuri modulo m on olemassa jos ja vain jos luku m on

$$2, 4, p^k \text{ tai } 2p^k,$$

missä p on jokin pariton alkuluku ja $k \in \mathbb{Z}_+$.

Todistus. ks. [6] □

Luku 4

Ohjelmointi: Jaollisuus

Tässä luvussa esitellään mahdollisia toteustapoja, kuinka lukuteorian sisältöjä aritmetiikan peruslause, Eratostheneen seula, Eukleideen algoritmi (ja sen kautta suurin ja pienin yhteinen tekijä) voitaisiin käsitellä ohjelmoinnin kautta. Lisäksi ihan lyhyesti esitellään yksi esimerkkiohjelma Eulerin φ-funktioon liittyen. Jokaisesta esitellään myös Python -ohjelmointikielellä kirjoitettuja esimerkkitoteutuksia. Erilaiset ohjelmointikokonaisuuksiin liittyvät tehtävät on jaoteltu kolmeen kategoriaan:

1) **Lämmittelyt**: Tehtäviä, jotka valmistavat aiheeseen liittyviin isompiin tehtäviin.

2) **Tehtävät**: Käsiteltävän lukuteorian aihealueen keskeiseen teoriaan liittyviä ohjelmointitehtäviä.

3) **Lisätehtävät**: Ylimääräisiä tehtäviä, jotka eivät ole niin keskeisiä aiheen kannalta, mutta joita voi olla antoisaa pohtia keskeisten tehtävien lisäksi.

Lukujen 4–6 esimerkkiohjelmien toteutukseen ja testaukseen on käytetty Pythonin versiota 3.9.5. Niiden toimivuutta ei ole testattu muilla versioilla. Lisäksi esimerkkiohjelmissa on lähtökohtaisesti oletettu, että käyttäjä käyttää niitä ”järkevästi” eli esimerkiksi tarkoituksenmukaisilla syötteillä. Kuitenkin tehtävien tarkoituksena on tässä vaiheessa keskittyä lukuteorian käsittelyyn ja niiden yhteydessä laadittujen ohjelmien pääasiallinen käyttäjä on ohjelman rakentanut opiskelija itse.

Luvuissa 4–6 materiaali on tuotettu siitä näkökulmasta, että ohjelmoinnin kautta lukuteoriaa harjoittelevilla on entuudestaan ohjelmoinnin ja Pythonin alkeet pääosin hallussa. Erikseen kommentoidaan, jos jonkin kokonaisuuden käsittelyä varten olisi suotavaa käydä läpi joitakin sellaisia ohjelmointiin liittyviä seikkoja tai esimerkiksi Pythonin funktioita, joita ei välttämättä alkeiden yhteydessä niin paljoa harjoitella.

4.1 Aritmetiikan peruslause

Aritmetiikan peruslauseen nojalla jokainen luku voidaan kirjoittaa yksikäsitteisesti (järjestystä lukuunottamatta) alkulukujen tulona (ks. lause 3.27). Tehtävän tavoitteena on harjoitella jaollisuuteen liittyviä ohjelmointikomentoja ja rakentaa ohjelma, joka jakaa annetun luvun alkulukutekijöihinsä.

Lämmittely 1. Kirjoita ohjelma, jolle kaksi kokonaislukua a ja b, ja ohjelma tarkistaa, onko luku b luvun a tekijä.

Pythonissa on suoraan sisäänrakennettuna funktio `%`, joka laskee jakolaskun jakojäännöksen, jota tulemme muutenkin tarvitsemaan paljon erilaisissa lukuteorian ohjelmallisissa tarkasteluissa. Esimerkiksi olkoot x ja y ovat kokonaislukuja ja $y > 0$. Tällöin funktiokutsu `x % y` palauttaa pienimmän epänegatiivisen kokonaisluvun, joka on kongruentti luvun x kanssa modulo y. Kyseinen luku on jakojäännös r, kun luku x kirjoitetaan jakoyhtälön (ks. lause 3.3) mukaisesti muodossa $x = ky + r$ joillakin $k, r \in \mathbb{Z}, 0 \leq r < y$. Näin ollen tehtävän vaatima ohjelma voidaan toteuttaa tarkistaen, päteekö `a % b == 0`:

```
a = 612
b = 48

if a % b == 0:
  print("Luku " + str(b) + " on luvun " + str(a) + " tekijä.")
else:
  print("Luku " + str(b) + " ei ole luvun " + str(a) + " tekijä.")
```

Esimerkkiohjelman tuloste on:

```
Luku 48 ei ole luvun 612 tekijä.
```

Lämmittely 2. Kirjoita ohjelma, joka käy läpi positiiviset kokonaisluvut johonkin lukuun n asti, ja lisää ne listatyyppiseen muuttujaan.

Toisen lämmittelytehtävän idea on palauttaa mieleen silmukoiden ja listojen käyttämistä.

Esimerkiksi:

```
n = 1000
luku = 1
lista = []
```

```
while luku <= n:
  lista.append(luku)
  luku += 1

print(lista)
```

Lämmittely 3. Kirjoita ohjelma, joka etsii kaikki annetun luvun n tekijät.

Käytännössä ohjelma voi käydä läpi positiiviset kokonaisluvut k lukuun n asti, ja lisätä niistä listaan ne, jotka toteuttavat ehdon `n % k == 0`.

```
def tekijat(n):
  lista = []
  k = 1

  while k <= n:

    if n % k == 0:
      lista.append(k)

    k += 1

  return lista

print(tekijat(59324001))
```

Esimerkkiohjelma etsii luvun 59 324 001 tekijät, ja sen tuloste on:

```
[1, 3, 11, 33, 83, 121, 179, 249, 363, 537, 913, 1331, 1969, 2739, 3993,
5907, 10043, 14857, 21659, 30129, 44571, 64977, 110473, 163427, 238249,
331419, 490281, 714747, 1797697, 5393091, 19774667, 59324001]
```

Laadittu ohjelma on melko hidas, sillä se käy läpi kaikki luvut aina lukuun n asti, vaikka todellisuudesta hakua voitaisiin optimoida karsimalla sellaisia lukuja, jotka eivät voi olla luvun n tekijöitä. Tällaisia ovat esimerkiksi kaikki luvut $\frac{n}{2} < k < n$, mikä on melkein puolet ohjelman läpikäymistä luvuista. Muitakin optimointeja voitaisiin tehdä. Algoritmien optimointia käydään hieman tarkemmin läpi vasta seuraavan tehtävän jälkeen.

Näiden palikoiden avulla olemme valmiita rakentamaan ohjelman, joka jakaa luvun alkulukutekijöihinsä.

Tehtävä: Aritmetiikan peruslause. Kirjoita ohjelma, joka jollain annetulla positiivisella kokonaisluvulla n tulostaa listan alkuluvuista, joiden tulo on n.

Tehtävän voisi toteuttaa samassa hengessä lämmittelytehtävän 3 tavoin, jolloin jokaisen luvun k kohdalla tarkistetaan ehdon `n % k == 0` lisäksi, onko k alkuluku. Tämän voi toteuttaa esimerkiksi tarkistamalla silmukan avulla, ovatko luvun k tekijät pelkästään luvut 1 ja k. Tällainen toteutus luonnollisesti on vielä lämmittelytehtävän esimerkkiratkaisuakin moninkerroin hitaampi, joten voi olla syytä lähteä pohtimaan hieman optimointinäkökulmaa jo näin alkuun. Yksi hyvä tapa toteuttaa alkulukutekijöiden haravointi on jakaa lukua n alkulukutekijöillään sitä mukaa, kun niitä löydetään. Tällä tavalla toimimalla voimme olla myös varmoja siitä, että listaan päätyy pelkkiä alkulukuja, sillä jos tarkistuksessa oleva k ei ole alkuluku, olemme jo aiemmassa vaiheessa ohjelman suoritusta käyneet läpi luvun k alkulukutekijät, ja jakaneet luvun n niillä, joten luku k ei enää voi päätyä listaan. Lukua n jakaessa on myös hyödyllistä käyttää Pythonin `//`-jakolaskua, joka palauttaa pelkästään jakolaskun kokonaislukuosan. Tavallinen `/`-jakolasku muuttaa Pythonissa oletuksena tuloksen liukuluvuksi, ja meille sopii paremmin käsitellä ainoastaan kokonaislukutyyppisiä muuttujia.

Esimerkiksi:

```
def alkulukutekijat(n):
  lista = []
  k = 2

  while k <= n:

    while n % k == 0:
      lista.append(k)
      n = n//k


    k += 1

  return lista

print(alkulukutekijat(59324001))
```

Esimerkkiohjelman tuloste on:

```
[3, 11, 11, 11, 83, 179]
```

Lisätehtävä 1. Pohdi, mitkä luvut k ylipäätään voivat olla luvun n alkulukutekijöitä. Voisiko alkulukutekijöiden etsinnästä karsia jo valmiiksi sellaisia, jotka eivät voi olla luvun n alkulukutekijöitä. Optimoi laatimaasi algoritmia havaintojesi perusteella.

Ainakaan luvulla n ei ole yhtään alkulukutekijää, joka on suurempi kuin $\frac{n}{2}$, ellei n ole itse alkuluku. Toisaalta esimerkiksi ainoa parillinen k joka tarvitsee tarkistaa, on $k = 2$, joten jatkossa nekin voisi suoraan jättää pois hausta. Jos alkulukutekijöihin jakamista joutuu tekemään paljon, yksi mahdollisuus olisi rakentaa erikseen alkulukutaulukko (esimerkiksi seuraavassa kappaleessa esitellyn 4.2 Eratostheneen seulan kaltaisesti), ja käydä läpi ainoastaan siinä esiintyviä lukuja.

4.2 Eratostheneen seula

Eratostheneen seulassa käytetään hyväksi taulukkoa, johon on koottu kaikki kokonaisluvut alkaen luvusta 2 ja päättyen johonkin haluttuun kokonaislukuun n. Idea on, että aloittaen ensimmäisestä alkuluvusta 2, poistetaan taulukosta kaikki tämän luvun monikerrat. Tämän jälkeen seuraava taulukossa jäljellä oleva luku on myös alkuluku, jonka monikerrat voidaan taas poistaa. Jatkamalla edelleen vastaavasti muille taulukon luvuille, lopulta taulukossa on jäljellä pelkästään alkulukuja [6]. Tarkastellaan esimerkiksi alkulukujen seulomista taulukosta, jossa on luvut $2, 3, \ldots, 30$.

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

Aloitetaan ympyröimällä ensimmäinen alkuluku 2, ja poistetaan taulukosta kaikki sen monikerrat:

	(2)	3	~~4~~	5	~~6~~
7	~~8~~	9	~~10~~	11	~~12~~
13	~~14~~	15	~~16~~	17	~~18~~
19	~~20~~	21	~~22~~	23	~~24~~
25	~~26~~	27	~~28~~	29	~~30~~

Taulukon seuraava jäljellä oleva alkio 3 on seuraava alkuluku. Ympyröidään se ja poistetaan sen monikerrat:

	(2)	(3)	~~4~~	5	~~6~~
7	~~8~~	~~9~~	~~10~~	11	~~12~~
13	~~14~~	~~15~~	~~16~~	17	~~18~~
19	~~20~~	~~21~~	~~22~~	23	~~24~~
25	~~26~~	~~27~~	~~28~~	29	~~30~~

Taulukon seuraava jäljellä oleva alkio 5 on seuraava alkuluku. Ympyröidään se ja poistetaan sen ainoa jäljellä oleva monikerta 25:

	(2)	(3)	~~4~~	(5)	~~6~~
7	~~8~~	~~9~~	~~10~~	11	~~12~~
13	~~14~~	~~15~~	~~16~~	17	~~18~~
19	~~20~~	~~21~~	~~22~~	23	~~24~~
~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~

Tässä vaiheessa voimme päätellä, että taulukossa ei enää voi olla jäljellä muita kuin alkulukuja, sillä lauseen 3.7 nojalla kaikilla lukua 30 pienemmillä yhdistetyillä luvuilla on alkulukutekijänä 2, 3 tai 5, joten ne on jo onnistuttu karsimaan pois. Merkitään taulukossa vielä mukana olevat alkiot alkuluvuiksi:

	(2)	(3)	~~4~~	(5)	~~6~~
(7)	~~8~~	~~9~~	~~10~~	(11)	~~12~~
(13)	~~14~~	~~15~~	~~16~~	(17)	~~18~~
(19)	~~20~~	~~21~~	~~22~~	(23)	~~24~~
~~25~~	~~26~~	~~27~~	~~28~~	(29)	~~30~~

Lämmittely 1. Luo ohjelmallisesti lista, jossa on 1000 alkiota ja jokainen alkio on `boolean`-tyyppinen muuttuja, jonka arvoksi alustetaan `True`.

Lämmittelytehtävien kautta voidaan muistella listojen alustamista ja käsittelemistä. Myöhemmän tehtävän kannalta on hyödyllistä muistella myös Pythonin `range()`-funktiota. Kun `range()`-funktiolle annetaan parametriksi jokin luku k, se muodostaa lukujonon $0, 1, \ldots, k-1$. Esimerkiksi `range(6)` vastaa lukuja $0, 1, 2, 3, 4, 5$. Funktio `range()` on hyödyllinen, kun halutaan käydä tietyt luvut läpi esimerkiksi `for`-silmukan avulla.

Lämmittelytehtävän voi toteuttaa esimerkiksi seuraavasti:

```
n = 1000

lista = [True for i in range(n)]

print(lista)
```

Lämmittely 2. Vaihda laatimasi listan joka toisen alkion arvoksi `False`.

Funktiolle `range()` voidaan antaa 1–3 parametria. Jos sille annetaan 2 parametria, ensimmäinen merkitsee, mikä on sen muodostaman lukujonon ensimmäinen luku ja jälkimmäinen parametri ilmaisee, mitä lukua ennen jonon muodostaminen lopetetaan. Esimerkiksi `range(3,6)` vastaa lukuja $3, 4, 5$. Jos funktiolle annetaan kolme parametria, ensimmäinen parametri kertoo, mistä luvusta jono alkaa, toinen parametri kertoo, mitä lukua ennen jonon muodostaminen lopetetaan, ja kolmas parametri ilmaisee, mikä on kahden peräkkäisen luvun väli. Esimerkiksi `range(1, 20, 2)` vastaa lukuja 1, 3, 5, 7, 9, 11, 13, 15, 17, 19.

Ohjelman voi toteuttaa esimerkiksi seuraavasti:

```
n = 1000

lista = [True for i in range(n)]

for i in range(1, n, 2):
  lista[i] = False

print(lista)
```

Tehtävä: Eratostheneen seula. Kirjoita ohjelma, joka etsii kaikki annettua lukua n pienemmät alkuluvut Eratostheneen seulan tavoin.

Tämä voidaan toteuttaa lämmittelytehtävien mukaisesti luomalla taulukko, joka sisältää `boolean`-tyyppisiä muuttujia. Käsiteltäviä lukuja, joista alkulukuja etsitään, vastaavat nyt taulukon indeksit, ja taulukossa sijaitseva `boolean`-arvo merkitsee, onko kyseinen luku alkuluku vai ei. Aluksi voidaan asettaa kaikille listan alkioille arvo `True`, ja sitten seulomisen myötä vaihtaa sellaisten indeksien arvoksi `False`, jotka tulevat karsituiksi.

Esimerkkiohjelmassa käsiteltävää listan alkiota on merkitty kirjaimella `p`. Kun ohjelma löytää listalta alkion, jolla on arvo `True`, se antaa arvon `False` kaikille listan alkioille, joiden indeksi on löydetyn alkion monikerta.

```
def EratostheneenSeula(n):
  lista = [True for i in range(n)]

  p = 2

  while p < n:

    if lista[p]:
      for i in range(2*p, n, p):
        lista[i] = False

    p += 1

  for i in range(2, n):
    if lista[i]:
      print(i, end=" ")

n = 1000
print("Alkuluvut, jotka ovat pienempiä kuin " + str(n)
 + ":")
EratostheneenSeula(n)
```

Esimerkkiohjelman tuloste on seuraava:

```
Alkuluvut, jotka ovat pienempiä kuin 1000:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191
193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283
293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401
409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509
521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631
641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751
757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877
881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

Lisätehtävä 1. Jos algoritmille annettu luku n on kovin suuri, esimerkiksi $n > 1\,000\,000\,000$, niin algoritmilla alkaa kestää melko kauan[1] aikaa löytää lukua n pienemmät alkuluvut. Pohdi, kuinka algoritmia voisi optimoida.

[1]Riippuen toki ohjelmointiympäristöstä ja suorittimista. Kokeillessani algoritmin rajoja itse luvun n arvolla $1\,000\,000\,000$, selainpohjainen Python-editori (ks. [35]) ei jaksanut pyörittää algoritmia loppuun, ja komentoriviltäkin ajettuna algoritmin suoritukseen kului reilu 5 minuuttia.

Ensinnäkin, kuten Eratostheneen seulan esittelyssä todettiin, ei ole tarpeen käydä kaikkia listan indeksejä läpi. Kaikki yhdistetyt luvut ovat nimittäin seuloutuneet listalta pois jo siinä vaiheessa, kun on käyty läpi indeksit $p \leq \sqrt{n}$ (ks. lause 3.7). Tämän lisäksi algoritmissa käytettyä `for`-silmukkaa voisi tehostaa. Voidaan suoraan lähteä liikkeelle alkuarvosta p^2, sillä kaikki alkuluvun p monikerrat kp on jo edeltävästi karsittu listalta kaikilla $k < p$. Lisäksi vastaavasti kuin lukuja alkulukutekijöihin jakavan algoritmin kanssa, ei ole syytä käydä kaikkia listan alkioita läpi. Esimerkiksi ainoa parillinen listan indeksi, joka tarvitsee tarkistaa, on $p = 2$. Tämän jälkeen listalla voitaisiin käydä vain joka toinen indeksi läpi.

4.3 Eukleideen algoritmi

Tehtävän tavoitteena on laatia ohjelma, joka Eukleideen algoritmia hyödyntämällä löytää kahden positiivisen kokonaisluvun suurimman yhteisen tekijän.

Lämmittely 1. Pohdi, kuinka voit etsiä kahden positiivisen kokonaisluvun suurimman yhteisen tekijän.

Ensimmäisenä mieleen saattaa tulla, että yksinkertaisesti yritetään jakaa lukuja kaikilla enintään annetuista luvuista pienemmän suuruisilla luvuilla, ja luvuista kerätään talteen suurin luku, jolla molemmat olivat jaollisia. Toinen hieman vastaava idea, joka mahdollisesti voi tulla mieleen, on lukujen jakaminen ensiksi alkulukutekijöihin, ja sitten poimimalla lukujen yhteiset alkulukutekijät, jolloin poimittujen alkulukutekijöiden tulo muodostaa suurimman yhteisen tekijän (tai jos yhtäkään yhteistä alkulukutekijää ei ole, suurin yhteinen tekijä on 1).

Vaikka näillä tavoin rakennetut ohjelmat toimivat, ovat ne kuitenkin hieman hitaita, jos annetut luvut ovat suuria. Johdannoksi tehokkaampaan ideaan eli Eukleideen algoritmiin voi tarkastella annetun kahden kokonaisluvun a ja b erotusta. Olkoot $a, b \in \mathbb{N}, b \neq 0$ ja $d = \text{syt}(a, b)$. Tällöin luvut a ja b voidaan kirjoittaa muodoissa $a = dm$ ja $b = dn$ joillakin $m, n \in \mathbb{Z}_+$. Huomataan, että erotus $a - b = dm - dn = d(m - n)$ on edelleen jaollinen lukujen suurimmalla yhteisellä tekijällä d. Siis d on myös lukujen a ja $a - b$ yhteinen tekijä, ja tarkemmin lauseen 3.16 kohdan *(ii)* nojalla myös niiden suurin yhteinen tekijä. Jos $a - b > b$, niin edelleen lukujen b ja $a - b - b = a - 2b$ suurin yhteinen tekijä on sama. Voidaan siis edetä vähentämällä luvusta a luku b niin monta kertaa kuin pystytään (siis $k = \max\{k \in \mathbb{Z}_+ \mid a - kb \geq 0\}$), jolloin päädytään lukuihin b ja $r = a - kb$. Näillä kahdella luvulla on edelleen on sama suurin yhteinen tekijä, kuin alkuperäisillä luvuilla a ja b. Nyt $r < b$, sillä muutoin k ei olisi maksimaalinen. Jatketaan menettelyä luvuilla r ja b, niin että vähennetään luvusta b niin monta kertaa luku r kuin mahdollista, jolloin

taas saadaan jokin jakojäännös, jolla tätä käsittelyä voidaan jatkaa. Jossain vaiheessa algoritmia jakojäännökseksi tulee 0, jolloin tätä edeltävän vaiheen jakojäännös on etsitty suurin yhteinen tekijä (Tarkempi matemaattinen tarkastelu kohdissa 3.5 ja 3.28).

Esimerkiksi lukujen 315 ja 231 suurin yhteinen tekijä voidaan selvittää Eukleideen algoritmilla seuraavasti:

$$\begin{aligned} 315 &= 231 \cdot 1 + 84 \\ 231 &= 84 \cdot 2 + 63 \\ 84 &= 63 \cdot 1 + 21 \\ 63 &= 21 \cdot 3. \end{aligned}$$

Suurin yhteinen tekijä on viimeinen nollasta poikkeava jakojäännös eli 21.

Lämmittely 2. Laadi ohjelma, joka muodostaa kahden annetun luvun $a, b \in \mathbb{N}, b \neq 0$, pohjalta lauseen 3.3 mukaisen jakoyhtälön $a = kb + r$ joillakin $k, r \in \mathbb{Z}$. Jälleen on hyödyllistä käyttää Pythonin `//`-jakolaskua, joka palauttaa jakolaskun kokonaislukuosan.

Esimerkiksi:

```
def jakoyhtalo(a, b):
  k = a//b
  r = a - k*b
  print(str(a) + " = " + str(k) + "*" + str(b) + " + " + str(r))
```

Tehtävä: Eukleideen algoritmi. Kirjoita ohjelma, joka etsii kahden annetun positiivisen kokonaisluvun suurimman yhteisen tekijän Eukleideen algoritmin avulla.

Tämän voi toteuttaa esimerkiksi muokkaamalla lämmittelytehtävässä laadittua ohjelmaa niin, että se toistaa `jakoyhtalo`-funktiota annetuilla kahdella luvulla, kunnes jakojäännös r on nolla. Tämän voi toteuttaa silmukoiden tai rekursion avulla. Alla olevassa esimerkissä on käytetty rekursiota `jakoyhtalo`-funktion sisällä.

Esimerkkiohjelmassa algoritmiin on demonstraation vuoksi sisällytetty Eukleideen algoritmin välivaiheiden tulostaminen.

```
def syt(a, b):

  k = a//b
  r = a - k*b
```

```
  print(str(a) + " = " + str(k) + "*" + str(b) + " + " + str(r))

  if r == 0:
    return b
  else:
    return syt(b, r)

a = 13842327123312
b = 24134454252
print("\nsyt(" + str(a) + "," + str(b) + ") = " + str(syt(a, b)))
```

Yllä oleva esimerkki toteuttaa Eukleideen algoritmin luvuille 13 842 327 123 312 ja 24 134 454 252. Ohjelman tuloste on seuraava:

```
13842327123312 = 573*24134454252 + 13284836916
24134454252 = 1*13284836916 + 10849617336
13284836916 = 1*10849617336 + 2435219580
10849617336 = 4*2435219580 + 1108739016
2435219580 = 2*1108739016 + 217741548
1108739016 = 5*217741548 + 20031276
217741548 = 10*20031276 + 17428788
20031276 = 1*17428788 + 2602488
17428788 = 6*2602488 + 1813860
2602488 = 1*1813860 + 788628
1813860 = 2*788628 + 236604
788628 = 3*236604 + 78816
236604 = 3*78816 + 156
78816 = 505*156 + 36
156 = 4*36 + 12
36 = 3*12 + 0

syt(13842327123312,24134454252) = 12
```

Lisätehtävä 1. Pohdi, miksi Eukleideen algoritmin suoritus aina päättyy, eli päädytään lopulta vaiheeseen, missä lukujen jakojäännökseksi tulee 0.

Jokaisella algoritmin iteraatiolla saatu jakojäännös pienenee edellisestä ja jokaisen jakojäännöksen arvo on vähintään 0. Näin ollen jossain vaiheessa algoritmin suoritusta väistämättä saadaan jakojäännökseksi 0 (ks. luku 3.5).

4.4 Pienin yhteinen jaettava

Tehtävän tavoitteena on laatia ohjelma, joka löytää kahden annetun positiivisen kokonaisluvun pienemmän yhteisen jaettavan.

Lämmittely 1. Pohdi, kuinka voisit löytää kahden positiivisen kokonaisluvun pienimmän yhteisen jaettavan.

Vastaavasti kuin suurimman yhteisen tekijän tapauksessa, tässäkin luultavasti voi kehitellä sellaisia toteutuksia, missä lähdetään yksinkertaisesti testaamaan eri lukuja, ja etsimään sellaisia, jotka ovat molemmilla annetuilla luvuilla jaollisia. Vaihtoehtoisesti tässäkin voidaan hyödyntää luvut alkulukutekijöihin jakavaa ohjelmaa, joka sitten kerää lukujen alkulukutekijöiden joukosta jokaisen luvuissa esiintyvän alkulukutekijän, ja ottaa niitä mukaan luvuissa esiintyvän alkulukutekijän korkeimman eksponentin ilmaiseman määrän. Näiden kerättyjen alkulukutekijöiden tulo on haluttu pienin yhteinen jaettava.

Vaikka näillä tavoin rakennetut ohjelmat jälleen toimivat, ne ovat melko hitaita, jos annetut luvut ovat kovin suuria. Tehokkaampi tapa löytyy hyödyntämällä tietoa, että $|ab| = \text{syt}(a,b)\,\text{pyj}(a,b)$ (ks. lause 2.2). Tästä ratkaisemalla pienimmän yhteisen jaettavan saadaan, että

$$\text{pyj}(a,b) = \frac{ab}{\text{syt}\,(a,b)}.$$

Tehtävä: Pienin yhteinen jaettava. Laadi ohjelma, joka etsii kahden annetun luvun pienimmän yhteisen tekijän. Voit käyttää hyödyksesi jo laadittua `EukleideenAlgoritmi`-funktiota.

Esimerkiksi:

```
def syt(a, b):

  k = a//b
  r = a - k*b

  if r == 0:
    return b
  else:
    return syt(b, r)
```

```
a = 54325643
b = 1273775
pyj = (a*b)//syt(a, b)

print("pyj(" + str(a) + "," + str(b) + ") = " + str(pyj))
```

Esimerkkiohjelman tuloste on seuraava:

```
pyj(54325643,1273775) = 69198645912325
```

4.5 Eulerin φ-funktio

Tämä tehtävä on lyhyt, sillä monessa muussa ohjelmointiosiossa meille riittää tieto, että $\varphi(n) = n - 1$, jos $n \in \mathbb{P}$. Joka tapauksessa tämä tehtävä voi olla hyvä tapa tutustua Eulerin φ-funktion ideaan.

Tehtävä 1. Laadi ohjelma, joka laskee Eulerin φ-funktion arvon $\varphi(n)$ annetulle luvulle n.

Voidaan laatia ohjelma, joka yksinkertaisesti testaa kaikki lukua n pienemmät positiiviset kokonaisluvut k, ja laskee kappalemäärän niistä, joilla pätee $\text{syt}(n, k) = 1$. Käytetään hyväksi jo valmista toteutusta Eukleideen algoritmille.

```
def syt(a, b):

  k = a//b
  r = a - k*b

  if r == 0:
    return b
  else:
    return syt(b, r)

def euler(n):
  phi = 1
  for i in range(2, n):
    if syt(i, n) == 1:
      phi += 1

  return phi
```

```
print("phi(14) = " + str(euler(14)))
print("phi(1254) = " + str(euler(1254)))
print("phi(459469) = " + str(euler(459469)))
```

Esimerkkiohjelman tuloste testiarvoille $n = 14, 1254$ ja 459469 on:

```
phi(14) = 6
phi(1254) = 360
phi(459469) = 459468
```

Lisätehtävä 1. Pohdi, kuinka ohjelmaa voitaisiin optimoida.

Yksi hyvä mahdollisuus olisi aluksi tarkistaa, onko annettu luku alkuluku vai ei. Jos se on, voidaan φ-funktion arvo laskea sille huomattavasti suoraviivaisemmin.

Luku 5

Ohjelmointi: Diofantoksen yhtälöt

Tässä luvussa esitellään tapoja tarkastella ja ratkaista ohjelmallisesti lineaarisia kahden muuttujan Diofantoksen yhtälöitä.

5.1 Kahden muuttujan lineaariset Diofantoksen yhtälöt

Lineaariset kahden muuttujan Diofantoksen yhtälöt ovat muotoa $ax + by = c$ olevia yhtälöitä, missä $a, b, c \in \mathbb{Z}$ sekä x ja y ovat kokonaislukuarvoja saavia muuttujia. Kokonaislukupari x_0, y_0 on yhtälön ratkaisu, mikäli $ax_0 + by_0 = c$ (ks. 3.29).

Tehtävän tavoitteena on tutkimalla ja ohjelmallisesti ratkaisemalla lineaarisia kahden muuttujan Diofantoksen yhtälöitä löytää, milloin lineaarisella kahden muuttujan Diofantoksen yhtälöllä on ratkaisuja ja milloin ei, sekä oppia etsimään yksittäisiä ratkaisuja lineaarisille kahden muuttujan Diofantoksen yhtälöille. Ohjelma voidaan lopulta laajentaa etsimään annetun Diofantoksen yhtälön kaikki ratkaisut.

Lämmittely 1. Tutki ohjelmallisesti, mitkä muuttujien x ja y kokonaislukuarvot ratkaisevat Diofantoksen yhtälön $208x + 136y = 120$.

Voidaan laatia ohjelma, joka käy lukupareja läpi, ja tulostaa sellaiset, jotka toteuttavat annetun yhtälön. Esimerkkiohjelmassa on käyty läpi muuttujien x ja y arvot väliltä $[-1000, 1000]$.

```
def testaaRatkaisu(luku1, luku2):
  if 208*luku1 + 136*luku2 == 120:
    print("Ratkaisu: x=" + str(luku1) + ", y=" + str(luku2))
```

```
for x in range(-1000, 1000):
  for y in range(-1000, 1000):
    testaaRatkaisu(x, y)
```

Lämmittely 2. Mikä yhteys äskeisen tehtävän ratkaisuiden välillä on? Arvaa niiden perusteella sääntö, minkä muotoisia annetun Diofantoksen kaikki ratkaisut ovat.

Alla on äskeisen ohjelman tuloste.

```
Ratkaisu: x=-650, y=995
Ratkaisu: x=-633, y=969
Ratkaisu: x=-616, y=943
Ratkaisu: x=-599, y=917
Ratkaisu: x=-582, y=891
Ratkaisu: x=-565, y=865
Ratkaisu: x=-548, y=839
Ratkaisu: x=-531, y=813
Ratkaisu: x=-514, y=787
Ratkaisu: x=-497, y=761
    ⋮             ⋮
```

Tulosteen pohjalta voidaan arvata, että annetuksen Diofantoksen yhtälön yksi ratkaisu on $x = -650, y = 995$, ja muita ratkaisuja saadaan lisäämällä muuttujan x arvoon luvun 17 monikerta ja vähentämällä muuttujan y arvosta vastaava luvun 26 monikerta, eli annetun Diofantoksen yhtälön ratkaisevat lukuparit ovat

$$\begin{cases} x = -650 + 17t, \\ y = 995 - 26t, \end{cases}$$

missä $t \in \mathbb{Z}$. Jos lukuparin muotoa halutaan sieventää, voidaan esimerkiksi valita $t = n + 38$, jolloin ratkaisut saadaan muotoon

$$\begin{cases} x = -4 + 17n, \\ y = 7 - 26n, \end{cases}$$

missä $n \in \mathbb{Z}$. Vaihtoehtoisesti oltaisiin voitu suoraan etsiä ohjelman löytämistä ratkaisuista ratkaisu $x = -4, y = 7$, jolloin oltaisiin suoraan päästy tähän yhtälön ratkaisevien lukuparien muotoiluun.

Lämmittely 3. Todista, että äskeisessä tehtävässä arvatut ratkaisut todellakin toteuttavat Diofantoksen yhtälön $208x + 136y = 120$ kaikilla $n \in \mathbb{Z}$.

Tämä voidaan todeta yksinkertaisesti sijoittamalla lukupari

$$\begin{cases} x = -4 + 17n, \\ y = 7 - 26n, \end{cases}$$

lausekkeeseen $208x + 136y$:

$$\begin{aligned} 208x + 136y &= 208\,(-4 + 17n) + 136\,(7 - 26n) \\ &= -832 + 3536n + 952 - 3536n \\ &= 120. \end{aligned}$$

Lämmittely 4. Testaa ohjelmallisesti, onko Diofantoksen yhtälöllä $3x + 6y = 20$ kokonaislukuratkaisuja. Entä yhtälöllä $3x + 6y = 30$? Pohdi, mistä tämä johtuu. Yleistä hypoteesisi yhtälölle $ax + by = c$ ja kuvaile, miten vakioiden a, b ja c arvoista voidaan arvata, onko yhtälöllä ratkaisuja.

Testaamalla esimerkiksi lämmittelytehtävän 1 ohjelmalla näyttäisi siltä, että yhtälöllä $3x + 6y = 20$ ei ole kokonaislukuratkaisuja, mutta yhtälöllä $3x + 6y = 30$ on.

Summasta $ax + by$ voidaan muodostaa vain lukujen a ja b suurimman yhteisen tekijän monikertoja, joten koska $\operatorname{syt}(3, 6) = 3$, ja $3 \nmid 20$, niin Diofantoksen yhtälöllä $3x + 6y = 20$ ei ole ratkaisuja. Toisaalta, koska $3 \mid 30$, niin Diofantoksen yhtälö $3x + 6y = 30$ on ratkaistavissa.

Yleisesti ehto voidaan muotoilla niin, että Diofantoksen yhtälöllä $ax + by = c$ on ratkaisuja täsmälleen silloin, kun $\operatorname{syt}(a, b) \mid c$ (ks. lause 3.29).

Tehtävä 1. Lineaarisen Diofantoksen yhtälön ratkaisun etsiminen. Kirjoita ohjelma, joka etsii annetulle lineaariselle kahden muuttujan Diofantoksen yhtälölle yhden ratkaisun x, y. Jos annettu yhtälö ei ole ratkaistavissa, ohjelman tulee tulostaa, että ”yhtälöllä ei ole ratkaisuja”.

Tehtävän ohjelman voisi periaatteessa kirjoittaa etsimään yhtä ratkaisua samoin, kuten ensimmäisessä lämmittelytehtävässä. Tämä kuitenkin on epäkäytännöllistä, sillä

Diofantoksen yhtälön ratkaisut voivat joskus olla harvassa, jolloin algoritmilla kestäisi pitkään. Jos algoritmi ei löytäisi ratkaisuja, emme myöskään saisi varmuutta yhtälön ratkaisemattomuudesta, sillä emme pysty käymään kaikkia kokonaislukuja läpi.

Tässä vaiheessa onkin siis hyvä harjoitella, kuinka Eukleideen algoritmia voidaan hyödyntää Diofantoksen yhtälöiden ratkaisussa. Johdatellaan tähän seuraavan esimerkin avulla.

Etsitään Diofantoksen yhtälölle $56x + 191y = 1$ yksi ratkaisu Eukleideen algoritmin avulla. Ensinnäkin huomataan, että yhtälö on ratkaistavissa, sillä $\operatorname{syt}(56, 191) = 1$. Tämä nähdään Eukleideen algoritmilla:

$$\begin{aligned} 191 &= 56 \cdot 3 + 23 \\ 56 &= 23 \cdot 2 + 10 \\ 23 &= 10 \cdot 2 + 3 \\ 10 &= 3 \cdot 3 + 1 \\ (3 &= 1 \cdot 3). \end{aligned}$$

Ratkaistaan algoritmin jokaiselta riviltä viimeistä lukuunottamatta jakojäännökset.

$$\begin{aligned} 23 &= 191 - 56 \cdot 3 \\ 10 &= 56 - 23 \cdot 2 \\ 3 &= 23 - 10 \cdot 2 \\ 1 &= 10 - 3 \cdot 3 \end{aligned}$$

Aloittaen viimeisestä rivistä, sijoittaen aina algoritmissa aiemman rivin jakojäännöksen paikalle sen ratkaistun lausekkeen, saamme

$$\begin{aligned} 1 &= 10 - 3 \cdot 3 \\ &= 10 + 3 \cdot (-3) \\ &= 10 + (23 - 10 \cdot 2) \cdot (-3) \\ &= 10 + 23 \cdot (-3) + 10 \cdot 6 \\ &= 23 \cdot (-3) + 10 \cdot 7 \\ &= 23 \cdot (-3) + (56 - 23 \cdot 2) \cdot 7 \\ &= 23 \cdot (-3) + 56 \cdot 7 + 23 \cdot (-14) \\ &= 56 \cdot 7 + 23 \cdot (-17) \\ &= 56 \cdot 7 + (191 - 56 \cdot 3) \cdot (-17) \\ &= 56 \cdot 7 + 191 \cdot (-17) + 56 \cdot 51 \\ &= 56 \cdot 58 + 191 \cdot (-17)\,. \end{aligned}$$

Viimeiseltä riviltä näemme, että yhtälön $56x+191y = 1$ eräs ratkaisu on $x = 58, y = -17$.

Ohjelmoinnin kannalta kuitenkin yksinkertaisempaa on ajaa algoritmi vain kertaalleen yhteen suuntaan. Samalla Diofantoksen yhtälö $ax + by = \text{syt}\,(a, b)$ voidaan ratkaista esittämällä jokainen jakojäännös $r_i =, i \in \mathbb{N}, i \leq k$, muodossa $u_i a + v_i b$, missä

$$\begin{aligned} u_{-1} &= 1, & u_0 &= 0, & u_{i+1} &= u_{i-1} - u_i q_i, \\ v_{-1} &= 0, & v_0 &= 1, & v_{i+1} &= v_{i-1} - v_i q_i \end{aligned}$$

(tarkemmille kirjainten määrittelyille ja matemaattiselle perustelulle ks. laajennetun Eukleideen algoritmin esittely s. 19). Viimeinen nollasta poikkava jakojäännös r_k tuottaa näin etsityt ratkaisut $x = u_k, y = v_k$, sillä $d = \text{syt}\,(a, b) = r_k = u_k a + v_k b$.

Jos annettu yhtälö on muotoa $ax + by = c$, missä $c \neq \text{syt}(a, b)$, mutta se on ratkaistavissa, voidaan silti löytää sen ratkaisu. Voidaan ratkaista yllä selitetyllä tavalla muotoa $ax + by = \text{syt}(a, b)$ oleva yhtälö, sillä tällöin $c = m\text{syt}(a, b)$ jollakin $m \in \mathbb{Z}$ ja haluttu ratkaisu on $x = mu_k, y = mv_k$.

Esimerkkiohjelmassa ohjelma kysyy käyttäjältä syötteenä vakioiden a, b ja c arvot, ja etsii laajennetun Eukleideen algoritmin avulla yhtälölle yhden ratkaisun, mikäli sellainen on olemassa.

```
def laajennettuEukleides(a, b):
  return lsyt(a, b, 1, 0, 0, 1)

def lsyt(a, b, u1, u2, v1, v2):

  k = a//b
  r = a - k*b
  u3 = u1 - k*u2
  v3 = v1 - k*v2

  if r == 0:
    return b, u2, v2
  else:
    return lsyt(b, r, u2, u3, v2, v3)

print("Syötä Diofantoksen yhtälön ax + by = c kertoimet:")
a = int(input("a ="))
```

```
b = int(input("b ="))
c = int(input("c ="))
print("")

syt, u, v = laajennettuEukleides(a, b)

if (c%syt != 0):
  print("Yhtälöllä ei ole ratkaisuja.")
else:
  k = c//syt
  print("Yhtälön " + str(a) + "x + " + str(b) + "y = " + str(c) +
  " yksi ratkaisu on x=" + str(u*k) + ", y=" + str(v*k) + ".")
```

Esimerkkiohjelman tuloste, kun sillä ratkaistaan Diofantoksen yhtälö $13500x + 7212y = 51180$:

```
Syötä Diofantoksen yhtälön ax + by = c kertoimet:
a = 13500
b = 7212
c = 51180

Yhtälön 13500x + 7212y = 51180 yksi ratkaisu on x=-1198465, y=2243390.
```

Toisaalta, jos yhtälöllä ei ole ratkaisuja, kuten Diofantoksen yhtälöllä $123557x + 12341y = 10$ (sillä syt $(123557, 12341) = 7$ ja $7 \nmid 10$), esimerkkiohjelma tulostaa, että ”yhtälöllä ei ole ratkaisuja”:

```
Syötä Diofantoksen yhtälön ax + by = c kertoimet:
a = 123557
b = 12341
c = 10

Yhtälöllä ei ole ratkaisuja.
```

Tehtävä 2. Laajenna äskeisen tehtävän ohjelmaa niin, että jos annetulla Diofantoksen yhtälöllä on ratkaisuja, niin ohjelma tulostaa yhtälön kaikki ratkaisut ilmaisevan lukuparin.

Tehtävä on luultavasti helpointa toteuttaa käymällä ensin läpi johdon lineaarisen kahden muuttujan Diofantoksen yhtälöiden ratkaisuihin (ks. 3.29). Jos Diofantoksen yhtälö $ax +$

$by = c$ on ratkaistavissa, sen ratkaisuja ovat täsmälleen lukuparit

$$\begin{cases} x = x_0 + \frac{bt}{d}, \\ y = y_0 - \frac{at}{d}, \end{cases}$$

missä x_0, y_0 on yksi ratkaisu ja $d = \text{syt}\,(a, b)$. Muutetaan aiemman ohjelman viimeiset rivit esimerkiksi seuraavaan muotoon:

```
if (c%syt != 0):
  print("Yhtälöllä ei ole ratkaisuja.")
else:
  k = c//syt

  print("Yhtälön " + str(a) + "x + " + str(b) + "y = " + str(c) +
  " ratkaisut ovat lukuparit\nx = " + str(u*k) + " + " + str(b//syt) +
  "t,\ny = " + str(v*k) + " - " + str(a//syt) + "t.")
```

Esimerkkiohjelman tuloste nyt, kun sillä ratkaistaan Diofantoksen yhtälö $12340x + 8664y = 252$:

```
Syötä Diofantoksen yhtälön ax + by = c kertoimet:
a = 12340
b = 8664
c = 252

Yhtälön 12340x + 8664y = 252 ratkaisut ovat lukuparit
x = 46179 + 2166t,
y = -65772 - 3085t.
```

Lisätehtävä 1. Ratkaise Diofantoksen yhtälö $788x + 7532y = 4$.

Tehtävän 2 ohjelman avulla saadaan:

```
Yhtälön 788x + 7532y = 4 ratkaisut ovat lukuparit
x = 736 + 1883t,
y = -77 - 197t.
```

Lisätehtävä 2. Ratkaise Diofantoksen yhtälö $-321x - 24y = 14058$.

Tehtävän 2 ohjelman avulla saadaan:

```
Yhtälön -321x + -24y = 14058 ratkaisut ovat lukuparit
x = -14058 + 8t,
y = 187440 - 107t.
```

Lisätehtävä 3. Kirjoita ohjelma, joka ratkaisee lineaarisen kolmen muuttujan Diofantoksen yhtälön eli yhtälön muotoa $ax + by + cz = d$.

Luku 6

Ohjelmointi: Kongruenssi

Tässä luvussa keskitytään ohjelmointiin, jossa käytetään hyödyksi modulaariaritmetiikkaa. Aluksi tarkastellaan hieman yleisesti kongruenssien laskemista ohjelmallisesti, minkä jälkeen käsitellään tunnetut salausmenetelmät Caesarin salausmenetelmä, Vigèneren salausmenetelmä sekä RSA-salausmenetelmä. Kappaleen lopussa käydään lisäksi läpi ohjelmallisesti kiinalainen jäännöslause sekä Diffien–Hellmanin avaintenvaihtoprotokolla.

6.1 Lukujen ja yhtälöiden tarkastelu kongruenssin avulla

Tehtävän tavoitteena on tutustua erilaisten kongruenssien tarkasteluun ohjelmointiympäristössä, ja samalla huomata joitakin kongruenssien ominaisuuksia, joista voi olla hyötyä esimerkiksi Diofantoksen yhtälöiden ratkaisuissa. Tässä vaiheessa oletetaan, että kongruenssin määritelmä, ominaisuudet ja laskusäännöt (ks. määritelmä 3.22, lemma 3.23 ja lause 3.24) on jo pääosin käyty läpi.

Lämmittely 1. Tutki ohjelmallisesti kokonaislukujen neliöiden jakojäännöksiä jaettaessa luvulla 3. Mitä huomaat?

Esimerkiksi:

```
for i in range(-1000, 1000):
  print(str((i**2)%3))
```

Ohjelman perusteella näyttäisi, että $x^2 \equiv 0 \pmod 3$ tai $x^2 \equiv 1 \pmod 3$ kaikilla $x \in \mathbb{Z}$, eli kokonaisluvun neliön jakojäännös jaettaessa luvulla 3 on 0 tai 1.

Lämmittely 2. Tutki ohjelmallisesti kokonaislukujen neliöiden jakojäännöksiä jaettaessa luvulla 9. Mitä huomaat?

Esimerkiksi:

```
for i in range(-1000, 1000):
  print(str((i**2)%9))
```

Ohjelman perusteella näyttäisi, että $x^2 \equiv 0, 1, 4$ tai $7 \pmod 9$ kaikilla $x \in \mathbb{Z}$, eli kokonaisluvun neliön jakojäännös jaettaessa luvulla 9 on 0, 1, 4 tai 7.

Tehtävä 1. Todista lämmittelytehtävien 1 ja 2 havainnot

$$x^2 \equiv 0 \text{ tai } 1 \pmod 3,$$
$$x^2 \equiv 0, 1, 4 \text{ tai } 7 \pmod 9,$$

kaikilla $x \in \mathbb{Z}$.

Tehtävä 2. Osoita, että yhtälöllä $x^2 + y^2 = 2001$ ei ole kokonaislukuratkaisuja.

Huomataan, että $2001 \equiv 3 \pmod 9$. Tehtävän 1 nojalla $x^2 \equiv 0, 1, 4$ tai $7 \pmod 9$ kaikilla $x \in \mathbb{Z}$. Vastaavasti $y^2 \equiv 0, 1, 4$ tai $7 \pmod 9$ kaikilla $y \in \mathbb{Z}$. Tutkitaan tämän perusteella, minkä kokonaislukujen kanssa summa $x^2 + y^2$ on kongruentti:

$$0 + 0 \equiv 0 \pmod 9$$
$$0 + 1 \equiv 1 \pmod 9$$
$$0 + 4 \equiv 4 \pmod 9$$
$$0 + 7 \equiv 7 \pmod 9$$
$$1 + 1 \equiv 2 \pmod 9$$
$$1 + 4 \equiv 5 \pmod 9$$
$$1 + 7 \equiv 8 \pmod 9$$
$$4 + 4 \equiv 8 \pmod 9$$
$$4 + 7 \equiv 2 \pmod 9$$
$$7 + 7 \equiv 5 \pmod 9$$

Huomataan, että $x^2 + y^2 \not\equiv 3 \pmod 9$ kaikilla $x, y \in \mathbb{Z}$. Näin ollen $x^2 + y^2 \neq 2001$ kaikilla $x, y \in \mathbb{Z}$.

Lämmittely 3. Tutki ohjelmallisesti lukujen x^4 ja x^6 jakojäännöksiä, kun jaetaan luvulla 13 ja $x \in \mathbb{Z}$. Mitä huomaat?

Esimerkiksi:

```
for i in range(-1000, 1000):
  print(str((i**4)%13))

for i in range(-1000, 1000):
  print(str((i**6)%13))
```

Ohjelman perusteella näyttäisi, että $x^4 \equiv 0, 1, 3$ tai $9 \pmod{13}$ ja $x^6 \equiv 0, 1$ tai $12 \pmod{13}$ kaikilla $x \in \mathbb{Z}$.

Tehtävä 3. Todista lämmittelytehtävän 3 havainnot

$$x^4 \equiv 0, 1, 3 \text{ tai } 9 \pmod{13},$$
$$x^6 \equiv 0, 1 \text{ tai } 12 \pmod{13},$$

kaikilla $x \in \mathbb{Z}$.

Tehtävä 4. Osoita, että yhtälöllä $x^4 + y^6 = 5$ ei ole kokonaislukuratkaisuja.

Lämmittelytehtävän nojalla $x^4 \equiv 0, 1, 3$ tai $9 \pmod{13}$ ja $y^6 \equiv 0, 1$ tai $12 \pmod{13}$ kaikilla $x, y \in \mathbb{Z}$. Tutkitaan tämän perusteella, minkä kokonaislukujen kanssa summa $x^4 + y^6$ on kongruentti:

$$\begin{aligned}
0 + 0 &\equiv 0 \pmod{13} \\
0 + 1 &\equiv 1 \pmod{13} \\
0 + 12 &\equiv 12 \pmod{13} \\
1 + 0 &\equiv 1 \pmod{13} \\
1 + 1 &\equiv 2 \pmod{13} \\
1 + 12 &\equiv 0 \pmod{13} \\
3 + 0 &\equiv 3 \pmod{13} \\
3 + 1 &\equiv 4 \pmod{13} \\
3 + 12 &\equiv 2 \pmod{13} \\
9 + 0 &\equiv 9 \pmod{13} \\
9 + 1 &\equiv 10 \pmod{13} \\
9 + 12 &\equiv 8 \pmod{13}
\end{aligned}$$

Huomataan, että $x^4 + y^6 \not\equiv 5 \pmod{13}$ kaikilla $x, y \in \mathbb{Z}$. Näin ollen $x^4 + y^6 \neq 5$ kaikilla $x, y \in \mathbb{Z}$.

Lisätehtävä 1. Etsi kaikki yhtälön $x^2 + y! = 2001$ positiiviset kokonaislukuratkaisut. Kappaleen 1.4 tehtävä 2 kirjasta [1].

Lisätehtävä 2. Osoita, että yhtälöllä $x^5 - y^2 = 4$ ei ole kokonaislukuratkaisuja. Kappaleen 1.4 esimerkkitehtävä 3 kirjasta [1].

6.2 Caesarin salaus

Caesarin salaus on laajalti tunnettu yksinkertainen salausmenetelmä. Caesarin salauksen idea on, että salattavan tekstin jokainen kirjain korvataan toisella aakkosten kirjaimella, joka on aakkostossa alkuperäisen kirjaimen jälkeen sovitun kirjainmäärän päässä.

Esimerkiksi, jos sovittu kirjainmäärä on 2, vaihdetaan jokainen salattavan tekstin kirjain kirjaimeen, joka on aakkostossa toinen kirjain alkuperäisen kirjaimen jälkeen, jolloin $a \to c, b \to d, c \to e$ ja niin edelleen. Jos aakkostossa ei ole enää riittävästi kirjaimia salattavan kirjaimen jälkeen, palataan aakkoston alkuun, joten englanninkielisessä aakkostossa äskeisen esimerkin mukaan $y \to a$ ja $z \to b$. Yleisesti, jos aakkoset on numeroitu niin, että $a \Rightarrow 0, b \Rightarrow 1, c \Rightarrow 2, ..., z \Rightarrow 25$ ja sovittu kirjainmäärä on $k \in \mathbb{Z}$, viestin $M = M_1 M_2 \dots M_n$, missä $M_i, i \in \mathbb{Z}_+, i \leq n$, on viestin kohdassa i olevaa kirjainta vastaava luku ja $n \in \mathbb{Z}_+$ on viestin pituus, salaus voidaan matemaattisesti ilmaista muodossa

$$C_i \equiv M_i + k \,(\mathrm{mod}\ 26)\,,$$

missä $C = C_1 C_2 \dots C_n$ on salattu viesti ja C_i on salatun viestin kohdassa i olevaa kirjainta vastaava luku. Salauksen purku tällöin olisi

$$M_i \equiv C_i - k \,(\mathrm{mod}\ 26)\,.$$

Lämmittely 1. Tutustutaan Pythonin `ord`- ja `chr`-funktioihin. Funktio `ord` muuttaa annetun merkin sitä vastaavaksi luvuksi, ja funktio `chr` puolestaan muuttaa annetun luvun sitä vastaavaksi merkiksi. Kokeile, mitä numeroita pienet kirjaimet $a, b, \dots, z$ vastaavat.

Testaamalla esimerkiksi koodia `print(ord("a"))` eri kirjaimilla voi huomata, että `ord("a") = 97`, `ord("b") = 98`, ..., `ord("z") = 122`. Nämä luvut tulevat ASCII-merkistöstä.

Lämmittely 2. Kirjoita ohjelma, joka muuttaa sille annetun pienen kirjaimen aakkostossa sitä kolme kirjainta myöhemmin esiintyväksi kirjaimeksi $a \to d$, $b \to e, \dots, z \to c$.

Esimerkiksi:

```
kirjain = "k"

kirjaimenPaikka = ord(kirjain) - ord("a")
uudenKirjaimenPaikka = (kirjaimenPaikka + 3) % 26
uusiKirjain = chr(uudenKirjaimenPaikka + ord("a"))

print(uusiKirjain)
```

Esimerkkiohjelmalle on annettu kirjain `k`, jonka se muuttaa kirjaimeksi `n`.

Lämmittely 3. Kirjoita ohjelma, joka muuttaa sille annetun pienellä kirjoitetun sanan kaikki kirjaimet niitä aakkostossa kolme kirjainta myöhemmin esiintyviksi kirjaimiksi $a \to d, b \to e, \ldots, z \to c$.

Esimerkiksi:

```
sana = "matematiikka"

muokattuSana = ""
for i in range(len(sana)):
  kirjain = sana[i]

  kirjaimenLuku = ord(kirjain) - ord("a")
  uudenKirjaimenLuku = (kirjaimenLuku + 3) % 26
  muokattuSana += chr(uudenKirjaimenLuku + ord("a"))

print(muokattuSana)
```

Esimerkkiohjelmalle on annettu sana `matematiikka`, jonka se muuttaa muotoon `pdwhpdwllnnd`.

Tehtävä 1: Caesarin salaus. Laadi ohjelma, joka salaa annetun viestin Caesarin salauksella annetulla kirjainmäärällä. Voit olettaa, että viesti on kirjoitettu pienillä kirjaimilla. Laadi ohjelma niin, että se ei muuta viestissä esiintyviä välilyöntejä.

Esimerkiksi:

```
def salaus(viesti, maara):
  salattuViesti = ""
  for i in range(len(viesti)):
```

```
    kirjain = viesti[i]
    if kirjain == " ":
      salattuViesti += " "
    else:
      kirjaimenLuku = ord(kirjain) - ord("a")
      salatunKirjaimenLuku = (kirjaimenLuku + maara) % 26
      salattuViesti += chr(salatunKirjaimenLuku + ord("a"))

  return salattuViesti

print(salaus("lukuteoriaa ohjelmoimalla", 2))
```

Yllä oleva esimerkki salaa viestin ”lukuteoriaa ohjelmoimalla” siirtämällä jokaista kirjainta kaksi paikkaa aakkostossa. Ohjelman tuloste on seuraava:

```
nwmwvgqtkcc qjlgnoqkocnnc
```

Tehtävä 2: Caesarin salauksen purku A. Pyydä toiselta opiskelijalta salattu viesti, ja käytä ohjelmaasi viestin purkamiseen, kun salauksessa käytetty kirjainmäärä tiedetään.

Esimerkiksi:

```
def salaus(viesti, maara):
  salattuViesti = ""
  for i in range(len(viesti)):
    kirjain = viesti[i]
    if kirjain == " ":
      salattuViesti += " "
    else:
      kirjaimenLuku = ord(kirjain) - ord("a")
      salatunKirjaimenLuku = (kirjaimenLuku + maara) % 26
      salattuViesti += chr(salatunKirjaimenLuku + ord("a"))

  return salattuViesti

viesti = "lukuteoriaa ohjelmoimalla"
salattuViesti = salaus(viesti, 2)
print("Salattu viesti: " + salattuViesti)
print("Purettu viesti: " + salaus(salattuViesti, -2))
```

Yllä oleva esimerkki salaa viestin ”lukuteoriaa ohjelmoimalla”, jonka jälkeen se purkaa salauksen. Ohjelman tuloste on seuraava:

```
Salattu viesti: nwmwvgqtkcc qjlgnoqkocnnc
Purettu viesti: lukuteoriaa ohjelmoimalla
```

Tehtävä 3: Caesarin salauksen purku B. Laajenna äskeisen kohdan ohjelmaa niin, että sillä voi purkaa Caesarin salauksella salatun viestin, vaikkei kirjainmäärä ole tiedossa. Tätä voi myös testata toisen opiskelijan kanssa niin, että salaa laaditulla ohjelmalla viestejä ja lähettää ne toiselle, muttei kerro, millä kirjainmäärällä on salauksen toteuttanut.

Ohjelma voi yksinkertaisesti käydä kaikki erilaiset vaihtoehdot salauksen purkamiselle läpi ja tulostaa niiden avulla puretut viestit. Purettujen viestien joukosta luultavasti yksi erottuu oikeasi viestiksi, jos viestin salaaja on käyttäny jotain tavallista kieltä viestinsä kirjoittamiseen. Toiminnallisuus voidaan toteuttaa esimerkiksi silmukan avulla. Esimerkkiohjelmma äskeisen ohjelman loppu on muutettu muotoon:

```
def purkuIlmanKirjainmaaraa(salattuViesti):
  for i in range(26):
    print("Kirjainmäärä " + str(i) + ": " + salauksenPurku(salattuViesti,
     i))

print("Salattu lause: qdqdqi akkbkk fyjiqqd\n")
purkuIlmanKirjainmaaraa("qdqdqi akkbkk fyjiqqd")
```

Esimerkkiohjelman tulostus on seuraava:

```
Salattu lause: qdqdqi akkbkk fyjiqqd

Kirjainmäärä 0: qdqdqi akkbkk fyjiqqd
Kirjainmäärä 1: pcpcph zjjajj exihppc
Kirjainmäärä 2: obobog yiizii dwhgoob
Kirjainmäärä 3: nananf xhhyhh cvgfnna
Kirjainmäärä 4: mzmzme wggxgg bufemmz
Kirjainmäärä 5: lylyld vffwff atedlly
Kirjainmäärä 6: kxkxkc ueevee zsdckkx
Kirjainmäärä 7: jwjwjb tddudd yrcbjjw
Kirjainmäärä 8: ivivia scctcc xqbaiiv
Kirjainmäärä 9: huhuhz rbbsbb wpazhhu
Kirjainmäärä 10: gtgtgy qaaraa vozyggt
Kirjainmäärä 11: fsfsfx pzzqzz unyxffs
Kirjainmäärä 12: ererew oyypyy tmxweer
```

```
Kirjainmäärä 13: dqdqdv nxxoxx slwvddq
Kirjainmäärä 14: cpcpcu mwwnww rkvuccp
Kirjainmäärä 15: bobobt lvvmvv qjutbbo
Kirjainmäärä 16: ananas kuuluu pitsaan
Kirjainmäärä 17: zmzmzr jttktt ohsrzzm
Kirjainmäärä 18: ylylyq issjss ngrqyyl
Kirjainmäärä 19: xkxkxp hrrirr mfqpxxk
Kirjainmäärä 20: wjwjwo gqqhqq lepowwj
Kirjainmäärä 21: vivivn fppgpp kdonvvi
Kirjainmäärä 22: uhuhum eoofoo jcnmuuh
Kirjainmäärä 23: tgtgtl dnnenn ibmlttg
Kirjainmäärä 24: sfsfsk cmmdmm halkssf
Kirjainmäärä 25: rererj bllcll gzkjrre
```

Kirjaimia oli siis siirretty 16 askelta, ja alkuperäinen viesti oli ”ananas kuuluu pitsaan”.

Lisätehtävä 1. Pohdi tehtävän 3 perusteella, kuinka turvallinen salaus Caesarin salaus on.

Ei kovinkaan turvallinen, sillä kaikki mahdolliset salausvaihtoehdot on nopeaa käydä läpi. Erilaisia salausvaihtoehtoja on aina aakkosten lukumäärä, riippumatta esimerkiksi viestin pituudesta.

Lisätehtävä 2. Laajenna ohjelmaasi niin, että viesteissä voi olla myös isoja kirjaimia. Vinkki: Tutki ensin, mitä lukuja isot kirjaimet vastaavat `ord`-funktion avulla.

Tutkimalla huomataan, että `ord("A") = 65`, `ord("B") = 66`, ..., `ord("Z") = 90`. Laajennuksen voi totettaa esimerkiksi Pythonin `isupper`- tai `islower`-funktion avulla.

Esimerkiksi aiemmin laadittua `salaus`-funktiota voitaisiin muokata seuraavanlaiseksi, jotta se ottaisi isot kirjaimet huomioon:

```
def salaus(viesti, kirjainMaara):
  salattuViesti = ""
  for i in range(len(viesti)):
    kirjain = viesti[i]
    if kirjain == " ":
      salattuViesti += " "
    else:
      if kirjain.isupper():
        kirjaimenPaikka = ord(kirjain) - ord("A")
```

```
            salatunKirjaimenPaikka = (kirjaimenPaikka + kirjainMaara) % 26
            salattuViesti += chr(salatunKirjaimenPaikka + ord("A"))
        else:
            kirjaimenPaikka = ord(kirjain) - ord("a")
            salatunKirjaimenPaikka = (kirjaimenPaikka + kirjainMaara) % 26
            salattuViesti += chr(salatunKirjaimenPaikka + ord("a"))

    return salattuViesti
```

Esimerkkiohjelma on toteutettu niin, että isot kirjaimet pysyvät salatussakin muodossa isoina, ja pienet kirjaimet puolestaan pieninä. Ohjelman toki voisi toteuttaa myös niin, että se esimerkiksi muuttaa heti aluksi kaikki kirjaimet pieniksi. Tällaisella toteutuksella se ei välttämättä säilytä informaatiota siitä, mitkä kirjaimet alunperin olivat isoja, mutta muuten salaa ja purkaa salauksen kyllä oikein. Toisaalta, jos lähetettävä tieto on esimerkiksi verkkosivu tai salasana, voi olla joskus tärkeääkin säilyttää tieto isoista kirjaimista.

Lisätehtävä 3. Pohdi, kuinka voisit muokata ohjelmaasi niin, että salatusta viestistä ei näe, missä kohdissa alkuperäisessä viestissä on välilyöntejä.

Tähän on muutama tapa. Välilyönnit voi yksinkertaisesti poistaa, viesti luultavasti on ymmärrettävä siitä huolimatta. Edelleen viestin purkajaa hämätäkseen salattuun viestiin voisi lisätä satunnaisia välilyöntejä, sillä nekään tuskin häiritsisivät alkuperäisen viestin ymmärtämistä, kun se on purettu.

Jos haluaa, että viestin purkamisen jälkeen välilyönnit ovat ennallaan, yksi mahdollisuus on laajentaa käytettävää aakkostoa niin, että käytössä on 27 merkkiä, eli aiemmat kirjaimet $a, b, \ldots, z$ ja sitten lisäksi välilyönti. Koodista tulee hieman monimutkaisempi, sillä välilyöntiä vastaava luku `ord`-funktiolla on 32.

6.3 Vigenèren salaus

Vigenèren salaus on käytännössä useampi perättäin toteutettu Caesarin salaus. Vigenèren salauksessa jokaiselle kirjaimelle edelleen toteutetaan Caesarin salaus, mutta aakkostossa siirrytty kirjainmäärä muuttuu kirjaimesta toiseen erillisen salausavaimen määräämällä tavalla. Oletetaan taas, että käytössä on englanninkielinen aakkosto, ja kirjaimet on koodattu numeroiksi kuten aiemmin, eli $a \Rightarrow 0, b \Rightarrow 1, c \Rightarrow 2, ..., z \Rightarrow 25$. Olkoon nyt salattava viesti esimerkiksi ”koiranpentu” ja salausavain ”omena”. Avaimen ensimmäinen kirjain määrää siirrettävän kirjainmäärän ensimmäiselle viestin kirjaimelle,

eli koska $o \Rightarrow 14$, siirretään ensimmäistä kirjainta 14 askelta, eli siis $k \rightarrow y$. Vastaavasti avaimen toinen kirjain on m eli viestin toista kirjainta siirretään $m \Rightarrow 12$ askelta, eli $o \rightarrow a$. Kun päästään viestin kuudenteen kirjainteen, avainta aletaan käydä uudestaan alkaen sen ensimmäisestä kirjaimesta, sillä avain oli vain 5 kirjaimen mittainen. Kun salaus toistetaan jokaiselle viestin kirjaimelle, loppujen lopuksi viesti ”koiranpentu” saadaan muotoon ”yameabbiati”.

Olkoon alkuperäinen viesti $M_1M_2 \dots M_n$, missä $M_i, i \in \mathbb{Z}_+, i \leq n$, on viestin kohdassa i olevaa kirjainta vastaava luku ja $n \in \mathbb{Z}_+$ on viestin pituus. Olkoon salausavain $K = K_1K_2 \dots K_m$, missä $K_j, j \in \mathbb{Z}_+, j \leq m$, on avaimen kohdassa j olevaa kirjainta vastaava luku ja $m \in \mathbb{Z}_+$ on avaimen pituus. Tällöin salattu viesti voidaan ilmaista matemaattisesti muodossa

$$C_i \equiv M_i + K_j \,(\mathrm{mod}\ 26)\,,$$

missä $j \equiv i \,(\mathrm{mod}\ m)\,, C = C_1C_2 \dots C_n$ on salattu viesti ja C_i on salatun viestin kohdassa i olevaa kirjainta vastaava luku. Vastaavasti salatun viestin purku olisi

$$M_i \equiv C_i - K_j \,(\mathrm{mod}\ 26)\,,$$

missä $j \equiv i \,(\mathrm{mod}\ m)$.

Kuten Caesarin salauksen kohdalla huomattiin, se oli melko helppo purkaa, vaikkei tiennyt salaukseen käytettyä kirjainmäärää, sillä jokaisella viestillä on ainoastaan 26 mahdollista erilaista salausta. Näin ei kuitenkaan ole Vigenèren salauksen kanssa, sillä sen tapauksessa jokaiselle *kirjaimelle* on 26 mahdollista erilaista salausta. Tämä tarkoittaa, että viestille $M = M_1M_2 \dots M_n$ on olemassa 26^n erilaista salausta, joten jos n on suuri, on varsin työlästä käydä kaikki vaihtoehdot läpi. Usein Vigenèren salauksen purkuun käytetäänkin tietoa, että salausavain on yleensä lyhyempi kuin salattu viesti, jolloin salauksesta löytyy tiettyä toisteisuutta. Tätä tietoa voidaan käyttää hyödyksi, kun viesti yritetään purkaa [6]. Tässä tutkielmassa kuitenkin keskitymme pääasiassa viestin salaamiseen ja purkamiseen vain, kun salausavain on tiedossa.

Tehtävä 1: Vigenèren salaus. Laadi ohjelma, joka salaa annetun sanan Vigenèren salauksella annetun salausavaimen avulla. Voit olettaa, että viesti on kirjoitettu pienillä kirjaimilla. Laadi ohjelma niin, että se ei muuta viestissä esiintyviä välilyöntejä.

Sinänsä salaaminen toimii hyvin samaan tapaan kuin Caesarin salauksessa. Ainoa iso ero on, että siirrettävä kirjainmäärä täytyy koodissa selvittää annetun avaimen avulla. Kuten yllä on esitetty, tämä voidaan toteuttaa kongruenssin avulla.

Esimerkiksi:

```
def salaus(viesti, avain):
  salattuViesti = ""

  for i in range(len(viesti)):
    kirjain = viesti[i]
    kirjainMaara = ord(avain[i%len(avain)]) - ord("a")
    if kirjain == " ":
      salattuViesti += " "
    else:
      kirjaimenPaikka = ord(kirjain) - ord("a")
      salatunKirjaimenPaikka = (kirjaimenPaikka + kirjainMaara) % 26
      salattuViesti += chr(salatunKirjaimenPaikka + ord("a"))

  return salattuViesti

print(salaus("lukuteoriaa ohjelmoimalla", "euler"))
```

Yllä oleva ohjelma salaa viestin ”lukuteoriaa ohjelmoimalla” käyttäen salausavainta ”euler”. Ohjelman tuloste on seuraava:

```
povykiicmre zlaifxszquwpr
```

Tehtävä 2: Vigenèren salauksen purku. Laajenna ohjelmaa niin, että sillä voi purkaa Vigenèren salauksella salattuja viestejä, kun salausavain tiedetään. Ohjelmaa voi testata lähettämällä toiselle opiskelijalle salausavaimen ja salaisen viestin, jonka opiskelija purkaa omalla ohjelmallaan.

Yksi luonnollinen tapa lähestyä tehtävää on yksinkertaisesti luoda toinen funktio, joka hoitaa viestien purkamisen, kuten tässä esimerkkiohjelmassa. Lisätehtävässä 1 pohditaan toista tapaa toteuttaa ohjelma.

```
def salaus(viesti, avain):
  salattuViesti = ""

  for i in range(len(viesti)):
    kirjain = viesti[i]
    kirjainMaara = ord(avain[i%len(avain)]) - ord("a")
    if kirjain == " ":
      salattuViesti += " "
    else:
      kirjaimenPaikka = ord(kirjain) - ord("a")
```

```
      salatunKirjaimenPaikka = (kirjaimenPaikka + kirjainMaara) % 26
      salattuViesti += chr(salatunKirjaimenPaikka + ord("a"))

  return salattuViesti

def salauksenPurku(salattuViesti, avain):
  viesti = ""

  for i in range(len(salattuViesti)):
    salattuKirjain = salattuViesti[i]
    kirjainMaara = ord(avain[i%len(avain)]) - ord("a")
    if salattuKirjain == " ":
      viesti += " "
    else:
      salatunKirjaimenPaikka = ord(salattuKirjain) - ord("a")
      kirjaimenPaikka = (salatunKirjaimenPaikka - kirjainMaara) % 26
      viesti += chr(kirjaimenPaikka + ord("a"))

  return viesti

viesti = "lukuteoriaa ohjelmoimalla"
avain = "euler"
salattuViesti = salaus(viesti, avain)

print("Salattu viesti: " + salattuViesti)
print("Purettu viesti: " + salauksenPurku(salattuViesti, avain))
```

Yllä oleva ohjelma salaa viestin ”lukuteoriaa ohjelmoimalla”, jonka jälkeen se purkaa salauksen. Ohjelman tuloste on seuraava:

```
Salattu viesti: povykiicmre zlaifxszquwpr
Purettu viesti: lukuteoriaa ohjelmoimalla
```

Lisätehtävä 1. Pohdi, miten voisit alkuperäistä `salaus`-funktiota käyttämällä purkaa Vigenèren salauksella salattuja viestejä, kun salausavain tiedetään. Vinkki: Yritä muodostaa sopiva purkuavain, jolla saat suoritettua salaamisen toiseen suuntaan.

Lisätehtävä 2. Pohdi, mitä haittaa viestin salaamisen näkökulmasta on, että laaditussa ohjelmassa jätetään viestiä salattaessa välilyönnit ennalleen. Pohdi tapoja, millä voisit poistaa ohjelmasta tämän heikkouden.

Jos salatusta viestistä näkee, missä kohdissa välilyönnit ovat, viestiä on helpompi analysoida ja mahdollisesti purkaa salaus. On useampi tapa, kuinka tästä heikkoudesta päästään eroon (ks. luku 5.4, lisätehtävä 2).

Lisätehtävä 3. Selvitä, kuinka Vigenèren salaus voidaan yrittää purkaa, mikäli salausavainta ei tiedetä. Tehtävää on hyvä lähteä tarkastelemaan esimerkiksi siltä kannalta, että opiskelija lähettää salatun viestin ilman salausavainta jollekin toiselle opiskelijalle, ja toinen opiskelija yrittää purkaa sen.

6.4 RSA-algoritmi

RSA-salausmenetelmän toiminnan periaate ja matemaattinen perustelu on käsitelty tarkemmin luvussa 3.10.

Tehtävän tarkoituksena on laatia ohjelma, joka salaa ja purkaa viestejä RSA -salausmenetelmän avulla. Tehtävä voi olla yksinkertaisempi toteuttaa, mikäli käsittelee viesteinä pelkkiä lukuja merkkijonojen sijaan. Ensimmäisessä ja toisessa lämmittelytehtävässä harjoitellaan kuitenkin skeema, jonka avulla voi testaamisen vuoksi kääntää lauseita luvuiksi ja päinvastoin. Tehtävän yhteydessä on hyvä puhua siitä, että yleisestikin tietokoneet käsittelevät erilaisia merkkejä lukuina (ks. Lisätehtävä 1), ja siinä mielessä tällainen kirjainten muuttaminen luvuiksi on hieman ylimääräistä ja turhaa työtä. Toisaalta on myös hyvä mainita, että todellisesti RSA-salausmenetelmän yhteydessä käytetyt protokollat viestien kuvaamiseksi lukuina ovat hieman monimutkaisempia, ja usein esimerkiksi salattava viesti ”pehmustetaan” turvallisuuden lisäämiseksi käyttämällä jotakin täytejärjestelmää, kuten esimerkiksi OAEP [2]. Tällöin käytännössä viestiä täytetään lisäämällä siihen tietoa, joka vaikeuttaa viestin purkamista. Tässä mielessä tällainen lämmittelytehtävien mukainen epärealistinen järjestelmä voi olla helpompi ymmärtää, jolloin huomio säilyy RSA-algoritmin toiminnassa. On hyvä myös huomata, että todellisuudessa salataan paljon muutakin kuin pelkkiä ”viestejä”. Salattava, lukuna tai lukuina esitetty asia, voi olla esimerkiksi vaikka kuvatiedosto tai tietokonepelin tallennustiedosto.

Lämmittely 1. Laadi ohjelma, joka kääntää annetun merkkijonon luvuksi niin, että merkkijono voidaan haluttaessa kääntää takaisin kirjaimiksi. Voit käyttää hyödyksesi Pythonin `ord`-funktiota.

Käytetään jälleen avuksi Pythonin `ord`-funktiota kääntämisessä, jolloin kirjaimia vastaavat luvut ovat $a \Rightarrow 97, b \Rightarrow 98, \ldots, z \Rightarrow 122$. Voimme nyt muodostaa käännettävää viestiä vastaavan luvun summaamalla sopivasti kirjaimia vastaavia lukuja järjestyksessä

yhteen. Koska suurimmassa jotain kirjainta vastaavassa luvussa on 3 numeroa, voimme aina kertoa seuraavaksi lukuun summattavan kirjaimen luvun luvulla 1000^{n-i}, missä $i \in \mathbb{Z}_+, i \leq n$, on kirjaimen paikka salattavassa viestissä ja $n \in \mathbb{Z}_+$ on viestin pituus. Havainnollistetaan tätä tarkastelemalla esimerkiksi sanan ”arpa” kääntämistä.

Alustetaan muuttuja `luku`, johon sanaa vastaava luku rakennetaan. Alussa `luku = 0`. Sanan ensimmäinen kirjain on `a`, jota vastaa luku 97. Kerrotaan se luvulla 1000^3 ja summataan muuttujaan `luku`, jolloin `luku = 97 000 000 000`.

Sanan toinen kirjain on `r`, jota vastaa luku 114. Kerrotaan se luvulla 1000^2 ja summataan muuttujaan `luku`, jolloin `luku = 97 114 000 000`.

Sanan kolmas kirjain on `p`, jota vastaa luku 112. Kerrotaan se luvulla 1000^1 ja summataan muuttujaan `luku`, jolloin `luku = 97 114 112 000`.

Sanan viimeinen kirjain on `a`, jota vastaa luku 97. Kerrotaan se luvulla 1000^0 ja summataan muuttujaan `luku`, jolloin `luku = 97 114 112 097`.

Näin olemme saaneet rakennettua luvun 97 114 112 097, jossa esiintyy järjestyksessä sanan ”arpa” kirjaimia vastaavat luvut. Ohjelman voi siis kirjoittaa seuraavasti:

```
def viestiLukuna(viesti):
  luku = 0
  for i in range(1, len(viesti) + 1):
    kirjain = viesti[i - 1]
    kirjaimenNumero = ord(kirjain)

    luku += kirjaimenNumero*(1000**(len(viesti) - i))

  return luku

print(str(viestiLukuna("lukuteoria")))
```

Esimerkkiohjelma kääntää sanan ”lukuteoria” luvuksi 108 117 107 117 116 101 111 114 105 097.

Lämmittely 2. Laadi ohjelma, joka kääntää annetun luvun merkkijonoksi äskeisen lämmittelytehtävän mukaisesti.

Luvun kolme viimeistä numeroa saadaan laskemalla sen jakojäännös jaettaessa tuhannella, joten viesti voidaan koota ikään kuin käänteisesti aloittamalla viestin viimeisestä kirjaimesta, kuten esimerkkiohjelmassa on tehty.

```
def lukuViestina(luku):
  viesti = ""
  while luku > 0:
    kirjaimenNumero = luku%1000

    viesti = chr(kirjaimenNumero) + viesti

    luku = luku//1000

  return viesti

print(lukuViestina(108117107117116101111114105097))
```

Esimerkkiohjelma kääntää luvun 108 117 107 117 116 101 111 114 105 097 sanaksi ”lukuteoria”, kuten pitikin.

Lisätehtävä 1. Selvitä, miten merkkijonoja oikeasti käsitellään lukuina Python3-ohjelmointiympäristössä.

Python3 käyttää merkkijonojen esitykseen Unicode-merkistöä [33]. Unicode-merkistö on standardi, jonka mukaan erilaiset merkit kuvataan biteiksi. Siis merkkijonoja oikeasti käsitellään jo lähtökohtaisestikin lukuina. Näin ollen paljon lämmittelytehtävien ratkaisuja tehokkaampaa olisi käyttää merkkijonojen bittiesityksiä suoraan, jolloin ylimääräisiä muutoksia ei tarvittaisi.

Tehtävä 1: Julkisen ja salaisen avaimen muodostus. Laadi ohjelma, joka muodostaa julkisen ja salaisen avaimen valitsemiesi alkulukujen p ja q avulla. Alkulukuja voi arpoa esimerkiksi sivustojen [7, 27] avulla. Tehtävässä ei ole välttämättä syytä käyttää kuitenkaan aivan massiivisia alkulukuja, sillä tavoitteena on vain demonstroida algoritmin toimintaa. Valittujen alkulukujen täytyy kuitenkaan olla vähintään niin suuret, että luvuksi muutettu salattu viesti on pienempi kuin $n = pq$ (ks. luku 3.10).

Esimerkiksi:

```
p = 5829019031616413
q = 384073254745027153

n = p*q
phi = (p - 1)*(q - 1)
```

Esimerkkiohjelman alkuluvut 5 829 019 031 616 413 ja 384 073 254 745 027 153 on valittu sivuston [7] alkulukuja arpovan algoritmin avulla. Sopivien alkulukujen etsiminen ja arpominen itse voisi myös olla mielenkiintoinen aihe tutustua, mutta se sivuutetaan tämän tutkielman yhteydessä. Tässä vaiheessa on hyvä varmistaa, että valittu n on kyllin suuri. Nämä luvut riittävät hyvin esimerkkiviestin ”lukuteoria” salaamiseen ja purkamiseen, sillä siinä on 30 numeroa, ja näiden alkulukujen tulossa on 34 numeroa. Tarkistetaan vielä, että valitsemamme luku n on yhteistekijätön luvuksi muutetun viestin ”lukuteoria” kanssa. Tähän voidaan käyttää jo aiemmin kirjoittamaamme ohjelmaa, joka etsii lukujen suurimman yhteisen tekijän, tai voimme tässä vaiheessa ottaa avuksi Pythonin oman `math`-kirjaston, josta sama toiminnallisuus löytyy:

```
import math

print(math.gcd(5829019031616413*384073254745027153,
108117107117116101111114105097))
```

Viesti on yhteistekijätön luvun n kanssa. Harvinaisemmassa tapauksessa, missä näin ei olisi, voisimme vaihtaa alkuluvut p ja q sellaisiksi, joilla ehto pätee.

Julkinen eksponentti e voidaan valita luvuista, jotka täyttävät ehdot $1 < e < \varphi(n)$ ja $\text{syt}(e, \varphi(n)) = 1$. Perinteinen valinta luvuksi e on luku $2^{16} + 1 = 65537$ [4]. Täytyy kuitenkin vielä varmistaa, että $\text{syt}(65537, \varphi(n)) = 1$. Käytetään taas Pythonin `math.gcd`-funktiota.

```
import math

e = 65537
p = 5829019031616413
q = 384073254745027153

n = p*q
phi = (p - 1)*(q - 1)
print("syt(e, phi) = " + str(math.gcd(e, phi)))
```

Jälleen tuloksemme on se, mitä toivottiinkin:

```
syt(e, phi) = 1
```

Harvinaisemmassa tapauksessa, missä näin ei olisi, voisimme taas esimerkiksi vaihtaa alkuluvun p tai q toiseksi alkuluvuksi. Todennäköisesti hyvin nopeasti valitsemme alkuluvut, joille ehto pätee.

Sopivan muuttujan d valinta on hieman monimutkaisempaa, sillä sen täytyy olla sellainen, joka täyttää ehdon $de \equiv 1 \pmod{\varphi(n)}$. Hyödynnetään tehtävässä aiemmin laadittua laajennettua Eukleideen algoritmia, sillä sopiva $d = x$ on myös yhtälön $ex + \varphi(n)\, y = 1 = \text{syt}(e, \varphi(n))$ yksi ratkaisu (vrt. Diofantoksen yhtälö $ax + by = \text{syt}(a, b)$, ks. kappale 5.1: Tehtävä 1).

```
def laajennettuEukleides(a, b):
  return lsyt(a, b, 1, 0, 0, 1)

def lsyt(a, b, u1, u2, v1, v2):

  k = a//b
  r = a - k*b
  u3 = u1 - k*u2
  v3 = v1 - k*v2

  if r == 0:
    return b, u2, v2
  else:
    return lsyt(b, r, u2, u3, v2, v3)

p = 5829019031616413
q = 384073254745027153

n = p*q
phi = (p - 1)*(q - 1)

e = 65537
syt, d, y = laajennettuEukleides(e, phi)

print("n = " + str(n))
print("phi = " + str(phi))
print("e = " + str(e))
print("d = " + str(d))
```

Esimerkiksi valituilla alkuluvuilla ohjelman tuloste on seuraava:

```
n = 2238770311443622074625895165462189
phi = 2238770311443621684723621388818624
e = 65537
```

```
d = 805809739180059995436860169688001
```

Ei kuitenkaan ole takuita, että tällä tavalla löydetty d olisi positiivinen tai pienempi kuin $\varphi(n)$. Ohjelman toimivuuden ja tehokkuuden kannalta on varmempaa valita sellainen d, jolle pätee, että $1 < d < \varphi(n)$ Tässä esimerkkiohjelmassa on myöhemmin viestien salaamisen ja purkamisen toteutuksessa käytetty Pythonissa valmiiksi olevaa `pow(x, y, z)`-funktiota, joka ei edes salli toisen parametrin olevan negatiivinen, joten esimerkkiohjelma ei toimi ollenkaan, kun $d < 0$). Luvun d muokkaus sopivaksi voidaan toteuttaa helposti esimerkiksi laskemalla d modulo $\varphi(n)$ luvun d etsimisen jälkeen:

```
d = d%phi
```

Tarkemmin julkisen eksponentin e ja salaisen eksponentin d valintaa tarkastellaan lisätehtävässä 3.

Tehtävä 2: RSA-salaus ja salauksen purku. Laadi ohjelma, jolla voit salata viestejä ja purkaa salattuja viestejä RSA-salausmenetelmällä. Voit käyttää lämmittelytehtävien ohjelmia, joilla viestejä voi muuttaa luvuiksi ja päinvastoin. Ohjelmaa voi testata lähettämällä toiselle opiskelijalle salatun viestin ja julkisen avaimen (n, d), joiden avulla toinen opiskelija voi purkaa viestin salauksen omalla ohjelmallaan.

Pythonissa on valmiiksi sisään rakennettuna funktio `pow(x, y, z)`, joka suorittaa laskutoimituksen $x^y \pmod z$. Tämän avulla voimme suoraan muodostaa salauksia ja purkaa niitä, kun tiedämme avaimet. Voimme nyt koota kaikista tämän luvun yhteydessä kirjoittamistamme ohjelmista esimerkiksi seuraavan tapaisen ohjelman (Ohjelmakoodin tiivistämiseksi ei toisteta tässä niiden funktioiden koodia, jotka on tässä luvussa jo aiemmin esitelty):

```
def viestiLukuna(viesti):

def lukuViestina(luku):

def laajennettuEukleides(a, b):

def lsyt(a, b, u1, u2, v1, v2):

def luoAvaimet(p, q):
  n = p*q
  phi = (p - 1)*(q - 1)
```

```
  e = 65537
  syt, d, y = laajennettuEukleides(e, phi)
  d = d%phi

  return n, phi, e, d

p = 5829019031616413
q = 384073254745027153
n, phi, e, d = luoAvaimet(p,q)

viesti = "lukuteoria"
print("Alkuperäinen viesti on " + viesti)

viesti = viestiLukuna(viesti)
c = pow(viesti, e, n)
print("Salattu viesti on " + str(c))

purettuViesti = pow(c, d, n)
viesti = lukuViestina(purettuViesti)
print("Purettu viesti on " + viesti)
```

Esimerkkiohjelman tuloste on seuraava:

```
Alkuperäinen viesti on lukuteoria
Salattu viesti on 151445836741508142823635050933765 7
Purettu viesti on lukuteoria
```

Esimerkkiohjelmassa viestin salausta ei ole muutettu merkkijonoksi, vaan säilytetty lukuna. Nimittäin jos muuttaisimme salatun viestin merkkijonoksi, saattaisi ilmetä ongelmia sen säilömisessä ja siirtelyssä, riippuen mitä reittejä viestiä liikutellaan (eli mikä merkistö/merkkikoodaus on käytössä). On hyvin mahdollista, että osa erikoisemmista merkeistä menetetään matkalla tai ne saattavat muuttua toisiksi, jolloin alkuperäinenkin viesti hukkuu.

Lisätehtävä 2. Selvitä, miksi lukujen p ja q tulee olla alkulukuja.

Mitä vaikeampaa luku n on jakaa tekijöihin, sitä hankalampi on purkaa RSA-salaus. Mitä pienemmät tekijät luvulla n on, sitä helpompi ne on löytää [18], joten on parasta valita luvun n ainoiksi tekijöiksi kaksi isoa alkulukua.

Toisekseen se, että luvut p ja q ovat alkulukuja, helpottaa lukujen d ja e määrittämistä. Esimerkiksi, jotta $\varphi(n)$ olisi tehokasta laskea (ks. Lisätehtävä 3), pitää meidän ensin tietää luvun n alkulukutekijät. Jos p ja q ovat massiivisia yhdistettyjä lukuja, voi niiden tekijöiden selvittäminen olla todella työläs tehtävä (tähänhän koko RSA:n turvallisuuskin perustuu), joten tämä ei ole kovinkaan käytännöllistä.

Lisätehtävä 3. RSA-salauksen purkamiseen tarvitsee oikeastaan selvittää vain $\varphi(n)$. Pohdi, miksi tämä ei ole juuri sen helpompaa, kuin luvun n jakaminen tekijöihin.

Tällä hetkellä ei tunneta algoritmia, joka tehokkaasti voisi jakaa luvun n tekijöihin, jos luvut p ja q on valittu hyvin. Jos toisaalta pystyttäisiin tehokkaasti ratkaisemaan $\varphi(n)$, pystyisimme sen avulla myös ratkaisemaan hyvin suoraviivaisesti luvun n tekijät tiedosta, että

$$\varphi(n) = (p-1)(q-1) = pq - p - q + 1.$$

Koska $pq = n$, niin saamme

$$q = n - \varphi(n) - p + 1.$$

Sijoittamalla tämän yhtälöön $n = pq$, saadaan

$$n = p(n - \varphi(n) - p + 1),$$

mistä edelleen

$$n = -p^2 - \varphi(n)\,p + pn + p,$$

joka voidaan järjestellä muotoon

$$p^2 + (\varphi(n) - n - 1)\,p - n = 0.$$

Koska $\varphi(n)$ ja n tiedetään, luvun n tekijät p ja q voidaan selvittää ratkaisemalla tämä toisen asteen yhtälö. Toisin sanoen, jos tiedossamme olisi keino selvittää tehokkaasti $\varphi(n)$, osaisimme myös tehokkaasti jakaa luvun n tekijöihin. Koska tiedossa ei ole tehokasta tapaa jakaa lukua n tekijöihin, voimme päätellä, ettei tiedossa myöskään ole tehokasta tapaa selvittää lukua $\varphi(n)$.

Lisätehtävä 4. Selvitä, miksi julkiseksi eksponentiksi e tai salaiseksi eksponentiksi d ei kannata valita ihan pienimpiä mahdollisia lukuja.

Usein saattaisi laskennan tehostamisen nimissä ollakin houkuttelevaa valita pienempiä arvoja käytetyille eksponenteille. Usein esimerkiksi jo valinta $e = 3$ tai $e = 5$ kävisi salauksen toiminnan näkökulmasta. Kuitenkin pieni julkinen eksponentti altistaa salauksen joillekin purkamishyökkäyksille [4], joten arvon $e = 65537$ käyttö on suositeltavaa.

Laskennan tehokkuuden nimissä voisi harkita myös käyttävänsä pientä salaisen eksponentin d arvoa. Tämä kuitenkin myös vaarantaa pahasti salauksen turvallisuuden esimerkiksi tietynlaisille ketjumurtolukuja hyödyntäville purkuyrityksille [4, 37].

6.5 Kiinalainen jäännöslause

Kiinalainen jäännöslause (ks. 3.36) sanoo, että lineaaristen kongruenssien systeemillä

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \quad \vdots \\ x \equiv a_r \pmod{m_r}, \end{cases}$$

missä $r \in \mathbb{Z}_+, x, a_1, a_2, \ldots, a_r \in \mathbb{Z}, m_1, m_1, \ldots, m_r \in \mathbb{Z}_+$ ja $\text{syt}(m_i, m_j) = 1$ kaikilla $i, j \in \mathbb{Z}_+, 1 \leq i < j \leq r$, on ratkaisu $x_0 \in \mathbb{Z}$, joka on yksikäsitteinen modulo M, missä $M = m_1 m_2 \ldots m_r$.

Lämmittely 1. Etsi ohjelmallisesti kokeilemalla ratkaisuita lineaaristen kongruenssien systeemille

$$\begin{cases} x \equiv 3 \pmod{5} \\ x \equiv 2 \pmod{7} \\ x \equiv 1 \pmod{11}, \end{cases}$$

Esimerkiksi:

```
for i in range(-1000,1000):
  if i%5==3 and i%7==2 and i%11==1:
    print("Ratkaisu: x=" + str(i))
```

Esimerkkiohjelma käy läpi muuttujan x arvot väliltä $[-1000, 1000]$, ja sen tuloste on seuraava:

```
Ratkaisu: x=-747
Ratkaisu: x=-362
Ratkaisu: x=23
Ratkaisu: x=408
Ratkaisu: x=793
```

Voidaan vielä todeta, että saadut ratkaisut ovat yksikäsitteisiä modulo $M = 5 \cdot 7 \cdot 11 = 385$, sillä

$$-747 \equiv -362 \equiv 23 \equiv 408 \equiv 793 \,(\mathrm{mod}\ 385)\,.$$

Tehtävä 1. Kirjoita ohjelma, joka ratkaisee lauseen 3.36 mukaisen lineaaristen kongruenssien systeemin.

Tehtävä voidaan ratkaista hyvin lauseen 3.36 todistuksen kaltaisella menettelyllä. Esimerkkiohjelmassa ohjelmalle annetaan listat `a` ja `m`, joista ensimmäinen sisältää järjestyksessä luvut $a_1, a_2, \ldots, a_r$ ja toinen järjestyksessä luvut $m_1, m_2, \ldots, m_r$. Listat `a` ja `m` annetaan parametreita ohjelman funktiolle `kiinalainenJaannoslause`. Aluksi funktio laskee luvun M, ja tämän jälkeen konstruoi yhtälön (3.37) mukaisen ratkaisun x_0 yksi summan termi kerrallaan. Sopiva $x_k \in \mathbb{Z}$, missä $k \in \mathbb{Z}_+, k \leq r$, joka ratkaisee lineaarisen kongruenssin $M_k x_k \equiv 1 \,(\mathrm{mod}\ m_k)$ löydetään laajennetun Eukleideen algoritmin avulla (ks. Luku 5.1, tehtävä 1).

```
def laajennettuEukleides(a, b):
  return lsyt(a, b, 1, 0, 0, 1)

def lsyt(a, b, u1, u2, v1, v2):

  k = a//b
  r = a - k*b
  u3 = u1 - k*u2
  v3 = v1 - k*v2

  if r == 0:
    return b, u2, v2
  else:
    return lsyt(b, r, u2, u3, v2, v3)

def kiinalainenJaannoslause(m, a):
  M = 1

  for i in range(len(m)):
    M *= m[i]

  ratkaisu = 0
```

```
  for i in range(len(m)):
    a_k = a[i]
    m_k = m[i]
    M_k = M//m_k

    syt, u, x_k = laajennettuEukleides(m_k, M_k)
    ratkaisu += a_k*M_k*x_k

  return ratkaisu%M, M

a = [1, 2, 3, 4]
m = [5, 7, 9, 11]
x, M = kiinalainenJaannoslause(m, a)

print("x=" + str(x) + " (mod " + str(M) + ")")
```

Tässä esitelty esimerkkiohjelma ratkaisee esimerkiksi lineaaristen kongruenssien systeemin

$$\begin{cases} x \equiv 1 \pmod 5 \\ x \equiv 2 \pmod 7 \\ x \equiv 3 \pmod 9 \\ x \equiv 4 \pmod{11}, \end{cases}$$

ja tulostaa yksikäsitteiseksi ratkaisuksi `x=1731 (mod 3465)`.

Lisätehtävä 1. Ratkaise lineaaristen kongruenssien systeemi

$$\begin{cases} x \equiv 3 \pmod 3 \\ x \equiv 6 \pmod 5 \\ x \equiv 7 \pmod{11} \\ x \equiv 10 \pmod{13}. \end{cases}$$

Tehtävässä 1 kirjoitettu ohjelma tulostaa annetuilla parametreilla `x=1206 (mod 2145)`.

Lisätehtävä 2. Ratkaise lineaaristen kongruenssien systeemi

$$\begin{cases} 2x \equiv 3 \pmod 5 \\ 18x \equiv 5 \pmod{11} \\ 5x \equiv 1 \pmod{31}. \end{cases}$$

Kirjoittamaamme ohjelmaa ei voi suoraan käyttää annettun systeemin ratkaisemiseksi. Jokainen yksittäinen lineaarinen kongruenssi voidaan kuitenkin ratkaista aluksi erikseen käyttäen hyväksi laajennettua Eukleideen algoritmia.

Ratkaistaan aluksi laajennetulla Eukleideen algoritmilla lineaariset kongruenssit $2x \equiv 1 \pmod 5$, $18x \equiv 1 \pmod{11}$ ja $5x \equiv 1 \pmod{31}$ esimerkiksi seuraavasti:

```
def laajennettuEukleides(a, b):
  return lsyt(a, b, 1, 0, 0, 1)

def lsyt(a, b, u1, u2, v1, v2):

  k = a//b
  r = a - k*b
  u3 = u1 - k*u2
  v3 = v1 - k*v2

  if r == 0:
    return b, u2, v2
  else:
    return lsyt(b, r, u2, u3, v2, v3)

syt, u, x_1 = laajennettuEukleides(5, 2)
syt, u, x_2 = laajennettuEukleides(11, 18)
syt, u, x_3 = laajennettuEukleides(31, 5)

print("x_1 = " + str(x_1) + ", x_2 = " + str(x_2) + ", x_3 = " + str(x_3))
```

Ohjelman tuloste on:

```
x_1 = -2, x_2 = -3, x_3 = -6
```

Ensimmäisestä kongruenssista saamme, että $2x \equiv 1 \pmod 5$, kun $x \equiv -2 \pmod 5$. Tällöin $2x \equiv 3 \pmod 5$, kun $x \equiv -6 \equiv 4 \pmod 5$. Toisesta kongruenssista saamme, että $18x \equiv 1 \pmod{11}$, kun $x \equiv -3 \pmod{11}$. Tällöin $18x \equiv 5 \pmod{11}$, kun $x \equiv -15 \equiv 7 \pmod{11}$. Kolmannesta kongruenssista saamme suoraan, että $5x \equiv 1 \pmod{31}$, kun $x \equiv -6 \equiv 25 \pmod{31}$. Ratkaistaan saatu lineaaristen kongruenssien systeemi

$$\begin{cases} x \equiv 4 \pmod 5 \\ x \equiv 7 \pmod{11} \\ x \equiv 25 \pmod{31}. \end{cases}$$

tehtävän 1. ohjelmalla, jolloin ohjelman tuloste on:

```
x=304 (mod 1705)
```

6.6 Diffien–Hellmanin-avaintenvaihtoprotokolla

Diffien–Hellmanin avaintenvaihtoprotokolla tarjoaa ratkaisun siihen, miten kaksi henkilöä voivat sopia yhteisestä salaisesta salausavaimesta niin, että avain pysyy salaisena vaikka sopiminen toteutettaisiin julkisia tai turvattomia tietoliikennekanavia pitkin. Protokollan julkaisivat Whitfield Diffie ja Martin Hellman vuonna 1976, ja protokolla on nimetty heidän mukaansa [8]. Protokollan toteutuksessa hyödynnetään primitiivisiä juuria (ks. 3.51). Tehtävissä oletetaan, että primitiivisten juurten määritelmä on tässä vaiheessa jo käyty läpi.

Alla on esitelty protokollan idea [15]. Oletetaan, että henkilö A ja henkilö B haluavat sopia yhteisestä salaisesta avaimesta.

1. Aluksi A ja B sopivat julkisesti alkuluvusta p sekä primitiivisestä juuresta a modulo p.
2. Seuraavaksi A valitsee jonkin positiivisen kokonaisluvun x ja B valitsee jonkin positiivisen kokonaisluvun y pitäen ne salassa.
3. A laskee luvun $c \equiv a^x \pmod{p}$ ja B laskee luvun $d \equiv a^y \pmod{p}$.
4. A ja B lähettävät toisilleen luvut c ja d, pitäen edelleen luvut x ja y salassa.
5. Tämän jälkeen A laskee luvun $d' \equiv d^x \pmod{p}$ ja B laskee luvun $c' \equiv c^y \pmod{p}$. Heidän saamansa tulokset d' ja c' ovat yhtäsuuret modulo p:

$$d' \equiv d^x \equiv (a^y)^x \equiv a^{xy} \equiv (a^x)^y \equiv c^y \equiv c' \pmod{p}\,.$$

Nyt henkilöillä A ja B on molemmilla hallussa $a^{xy} \pmod{p}$, jota he voivat käyttää yhteisenä salaisena avaimenaan.

Tehtävä 1. Oletetaan, että pahantahtoinen henkilö C on kuunnellut koko ajan henkilöiden A ja B viestintää, eli hän myös tietää luvut p, a, c ja d. Selvitä, miksi voidaan luottaa, ettei henkilö C kuitenkaan onnistu selvittämään jaettua salaista avainta näiden tietojen pohjalta?

Henkilön C tavoite on ratkaista joko eksponentti x tai eksponentti y, sillä kumpi vain riittää jaetun salaisen avaimen selvittämiseen. Mutta vaikka henkilö C tietääkin luvut

p, a ja $a^x \pmod{p}$, ei ole tiedossa mitään tehokasta tapaa ratkaista eksponentin x arvoa näiden tietojen pohjalta [15]. Ongelma on niin kutsuttu ”diskreetin logaritmin ongelma”.

Tehtävä 2. Pohdi, miksi valitun luvun a halutaan olevan primitiivinen juuri modulo p.

Primitiivisen juuren määritelmän (ks. Määritelmä 3.51) nojalla tiedetään, että jos a on primitiivinen juuri, niin $k = \varphi(p) = p - 1$ on pienin kokonaislukueksponentti, jolla $a^k \equiv 1 \pmod{p}$. Erityesti korollaarin 3.48 nojalla tällöin tiedetään, että mitkään luvuista $a, a^2, a^3, \ldots, a^{p-1}$ eivät ole kongruentteja keskenään modulo p. Lauseen 3.52 nojalla on salaisella avaimella on lukujen $1, 2, \ldots, p-1$ joukossa tasan yksi luku, jonka kanssa se on kongruentti modulo p. Jos luku a ei ole primitiivinen juuri modulo p, vaan sen kertaluku on pienempi kuin $\varphi(n)$ modulo p, salaisen avaimen selvittämiseen tarvittavien läpikäytävien lukujen määrä olisi pienempi. Todellisuudessa kuitenkaan luvun a ei ole pakko olla primitiivinen juuri modulo p, mutta tällöin on syytä varmistaa, että sillä on riittävän suuri kertaluku modulo p, tai salauksen turvallisuus vaarantuu.

Tehtävä 3. Kirjoita ohjelma, joka etsii primitiivisen juuren modulo p jollain annetulla alkuluvulla p. Lauseen 3.54 nojalla primitiivinen juuri modulo p on olemassa kaikilla p.

Voidaan rakentaa suhteellisen raskas algoritmi, joka yksinkertaisesti käy läpi lukuja $a \in \mathbb{Z}, 2 \leq a \leq p$, ja jokaisen kohdalla käy läpi positiiviset kokonaisluvut k järjestyksessä luvusta 1 lukuun $p-1$ asti testaten, onko $a^k \equiv 1 \pmod{p}$. Jos on, niin ohjelma tarkistaa, oliko $k = p - 1$. Jos oli, a on primitiivinen juuri, ja jos ei, kasvatetaan luvun a arvoa yhdellä ja kokeillaan, onko se nyt primitiivinen juuri.

```
def etsiPrimitiivinenJuuri(p):
  a = 2

  while True:
    for k in range(1, p):
      if pow(a, k, p) == 1:
        if k == (p - 1):
          return a
        else:
          break
    a += 1

print(str(etsiPrimitiivinenJuuri(71)))
```

Ohjelman tulosteen perusteella 7 on primitiivinen juuri modulo 71.

Toki on selvää, että tällainen ohjelma on todella hidas, jos p on yhtään isompi ja primitiivinen juuri ei tule vastaan nopeasti.

Lisätehtävä 1. Pohdi, kuinka voisit etsiä primitiivisiä juuria äskeistä ohjelmaa tehokkaammin.

Lause 3.46 tarjoaa yhden ratkaisun tähän pulmaan. Lauseen nojalla, jos $\operatorname{ord}_p(a) = k$, niin $k \mid \varphi(p)$. Jos $k < \varphi(p)$ ja luvun $\varphi(p) = p - 1$ alkulukutekijäesitys on $p_1 p_2 \dots p_n, n \in \mathbb{Z}_+$, niin

$$a^{\varphi(p)/p_i} \equiv 1 \pmod{p},$$

jollakin $1 \leq i \leq n$. Voimme siis rakentaa ohjelman niin, että se aluksi selvittää luvun $\varphi(p)$ alkulukutekijät, esimerkiksi käyttäen ohjelmaa, jonka rakensimme aritmetiikan peruslause -osiossa. Tämän jälkeen ohjelma testaa kandiaatilla a, onko $a^{\varphi(p)/p_i} \equiv 1 \pmod{p}$ käymällä läpi jokaisen luvun $\varphi(p)$ alkulukutekijän p_i. Tämä ei toteudu millään p_i jos ja vain jos luku a on primitiivinen juuri modulo p.

```
def alkulukutekijat(n):
  lista = []
  k = 2

  while k <= n:

    while n % k == 0:
      lista.append(k)
      n = n//k

    k += 1

  return lista

def etsiPrimitiivinenJuuri(p):
  a = 2
  lista = alkulukutekijat(p - 1)

  while True:
    for k in lista:
      if pow(a, (p-1)//k, p) == 1:
        break
      elif k == lista[len(lista) - 1]:
```

```
        return a

    a += 1

print(str(etsiPrimitiivinenJuuri(4367924681)))
```

Ohjelman tulosteen perusteella 3 on primitiivinen juuri modulo 4 367 924 681.

Lisätehtävä 2. Pohdi, kuinka lisätehtävän 1. ohjelmaa voisi vielä optimoida.

Jos muistellaan, kuinka alkulukutekijöitä etsivä ohjelmamme toimi, se etsi sellaiset alkuluvut, joiden tulo on sille annettu luku. Tässä tapauksessa kuitenkin meidän ei olisi tarpeen poimia kuin kertaalleen jokainen luvun alkulukutekijä, joten `alkulukutekijat`-funktiota voisi muokata ohjelman kannalta tehokkaammaksi.

Lisätehtävä 3. Laajenna lisätehtävän 1 ohjelmaa niin, että se etsii kaikki primitiiviset juuret modulo p.

Käytetään hyödyksi korollaarin 3.50 tietoa, että jos luku a on primitiivinen juuri modulo p, niin luku a^k on primitiivinen juuri modulo p jos ja vain jos $\text{syt}\,(k, \varphi\,(p)) = 1$. Lauseen 3.52 nojalla kaikki muut primitiiviset juuret löytyvät lukujen $a, a^2, \ldots, a^{\varphi(p)}$ joukosta. Käydään ohjelmassa nämä vaihtoehdot läpi.

```
def syt(a, b):

  k = a//b
  r = a - k*b

  if r == 0:
    return b
  else:
    return syt(b, r)

def alkulukutekijat(n):
  lista = []
  k = 2

  while k <= n:

    while n % k == 0:
```

```
      lista.append(k)
      n = n//k

    k += 1

  return lista

def etsiPrimitiivinenJuuri(p):
  a = 2
  lista = alkulukutekijat(p - 1)

  while True:
    for k in lista:
      if pow(a, (p-1)//k, p) == 1:
        break
      elif k == lista[len(lista) - 1]:
        return a

    a += 1

def etsiMuutPrimitiivisetJuuret(a, p):
  lista = [a]

  for k in range(2, p):
    if syt(k, (p - 1)) == 1:
      b = pow(a, k, p)
      if b not in lista:
        lista.append(b)

  return lista

p = 71
prim = etsiPrimitiivinenJuuri(p)
print("Kaikki luvun " + str(p) + " primitiiviset juuret ovat:")
primJuuret =  etsiMuutPrimitiivisetJuuret(prim, p)
primJuuret.sort()
for i in primJuuret:
  if i != primJuuret[len(primJuuret) - 1]:
    print(str(i) + ",", end=" ")
```

```
  else:
    print(str(i) + ".")
```

Esimerkkiohjelman tuloste on:

```
Kaikki luvun 71 primitiiviset juuret ovat:
7, 11, 13, 21, 22, 28, 31, 33, 35, 42, 44, 47, 52, 53, 55, 56, 59,
61, 62, 63, 65, 67, 68, 69.
```

Tehtävä 4. Testaa Diffien-Hellmanin avainten vaihtoa toisen opiskelijan kanssa tekemällä protokollan vaiheet 1–5.

Luku 7

Pohdinta

Tutkielmassa on nyt esitelty lukuteorian osa-alueita niin lukion opetussuunnitelman sisältöalueiden pohjalta kuin myös niiden ulkopuolelta. Lukion uusi opetussuunnitelman tulee yhdistämään ohjelmoinnin ja lukuteorian saman kurssin alle, ja luvussa 2 esitelty teoreettinen viitekehys loi paljon positiivisia odotuksia näiden oppiaineiden yhdistämisen suhteen. Tässä luvussa pyritään hieman arvioimaan, kuinka hyvin noita odotuksia mahdollisesti pystytään lunastamaan tutkielmassa esiteltyjen ohjelmallisten harjoitteiden kautta, ja toisaalta, mitä haasteita näihin liittyy.

Lukion kurssit on yleensä suunniteltu melko tarkasti niin, ettei opetussuunnitelman kurssin keskeisten sisältöjen läpikäymisen ohella jää juuri oppitunteja syventävän materiaalin läpikäymiseen. Ainakin pitäisin hyvin selvänä, ettei kaikkea tässä tutkielmassa esiteltyä lukuteoriaa tai ohjelmointia pysty mitenkään toteuttamaan uuden MAA11-kurssin puitteissa. Tutkielman tavoitteena onkin tarjota vain alustavia ehdotuksia, jotka kaipaavat vielä tutkimista ja testaamista. Jos aloitetaan tarkastelemalla tutkielmassa esiteltyä lukion opetussuunnitelman tavoitteita laajempaa lukuteoriaa, siinä pysytellään loppujen lopuksi kuitekin melko lähellä lukion opetussuunnitelman sisältöjä. Lineaarisiin Diofantoksen yhtälöihin tarvitsee pääasiassa ymmärryksen Eukleideen algoritmista, suurimmasta yhteisestä tekijästä ja jaollisuudesta, jotka on jo katettu kurssin keskeisissä sisällöissä. Yhdellä kertaa saataisiin käytyä läpi tällöin myös laajennettu Eukleideen algoritmi. Kiinalaiseen jäännöslauseeseen pääsee käsiksi modulaariaritmetiikan ja lineaaristen Diofantoksen yhtälöiden avulla, ja se tarjoaa monenlaisia ongelmia pyöriteltäväksi kongruenssien kautta. Eulerin φ-funktiota pääsee hyvin tutkimaan jo sen jälkeen, kun on käyty läpi suurin yhteinen tekijä ja kongruenssi, ja edelleen kongruenssien, alkulukujen ja Eulerin φ-funktion jälkeen voi tutustua jo RSA-salausmenetelmään, puhumattakaan Caesarin ja Vigèneren salausmenetelmistä, joiden käsittely ei vaadi juurikaan pohjatietoa. Vaikka Eratos-

theneen seulaa tai alkulukujakaan ei uuden opetussuunnitelman MAA11-kurssin sisällöissä mainita, jos lukujen jaollisuutta kuitenkin käsitellään, nämäkin sisällöt astuvat kuvaan luontevasti. Salausmenetelmät tarjoavat yhden lukuteorian konkreettisen sovelluskohteen, ja sitä kautta voivat motivoida ja auttaa lukuteorian sisäistämisessä. Primitiiviset juuretkin voi näiden tietojen pohjalta nähdä ainakin jollain tasolla mahdollisena sisäistää. Totta kuitenkin on, että varsinkin viimeisimpänä mainittujen aiheiden suhteen esimerkiksi lauseiden todistaminen lukioympäristössä voi olla työlästä, ja kaikkeen ei kuitenkaan aika riitä.

Ohjelmointitehtävien tavoite oli tarjota mahdollisuuksia päästä matalalla kynnyksellä ohjelmallisesti testailemaan erilaisia lukuteorian sisältöjä, minkä kautta saisi hyvän pohjan ja motivaatiota lähteä rakentamaan syvempää ymmärrystä käsiteltävästä aiheesta. Lopullinen isomman ohjelman rakentaminen tulisi vaatimaan aiheen syvempää omaksumista. On kuitenkin mahdollista nähdä joitakin kompastuskiviä ohjelmointitehtävien onnistumisten suhteen. Lukion opetussuunnitelma ei anna kovin tarkkaa ohjeistusta siihen, kuinka paljon ohjelmointia tullaan kurssin aikana opettelemaan. Jos ohjelmoinnin perusteita ei käydä kunnolla läpi, voi lukuteoriankin läpikäynti ohjelmoinnin kautta olla haasteellista, sillä tällöin keskittyminen menee ohjelmoinnin ymmärtämiseen, kun luultavasti taustalla oleva lukuteoria tarvitsisi myös paljon ajatuksellisia resursseja. Paljon varmasti ohjelmoinnin opetukseen ja käsittelyyn vaikuttaa myös opettajan omat taidot sekä opiskelijoiden aiempi ohjelmointikokemus, jonka mukaan lukuteorian käsittelytapoja tulevalla MAA11-kurssilla kannattaneekin tiettyyn pisteeseen asti räätälöidä.

Todellisuudessa opettajien toimintaa ohjaavat myös jossain määrin oppikirjat, joten nähtäväksi jää, minkälaisesta näkökulmasta niissä lähdetään käsittelemään ja yhdistelemään kurssin aiheita. Jos kurssin sisällöt linkitetään heikosti, voi esimerkiksi lukuteoria ja ohjelmointi jäädä kokonaan erillisiksi sisällöiksi. Näkisin, että tällöin kurssin eheys kärsii, ja menetetään joitakin etuja, joita monialainen oppiminen voisi tarjota. Tässä tutkielmassa esiteltyjen konkreettisten esimerkkien ja tehtävien tavoitteena on osaltaan tarjota tukea tämän ongelman ratkaisuun, ja herätellä pohtimaan, kuinka kurssin rakenne olisi mielekästä toteuttaa. Kuitenkin esiteltyjen menetelmien opetuksellisen toimivuuden toteamiseksi tarvitaan lisää tutkimusta.

Kirjallisuutta

[1] Andreescu, T. & Andrica, D. (2002). *An Introduction to Diophantine Equations.* GIL Publishing House.

[2] Bellare, M. & Rogaway, P. (1995). *Optimal Asymmetric Encryption - How to Encrypt with RSA*. Springer-Verlag.

[3] Berhuy, G. (2020). *Algèbre : le grand combat.* Mathématiques en devenir, 121, Calvage & Mounet.

[4] Boneh, D. (1999). *Twenty years of attacks on the RSA cryptosystem.* Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203-213.

[5] Brok, P., Pilot, A., & Taconis, R. (2016). *Teachers creating context-based learning environments in science.* SensePublishers.

[6] Burton, D. (2007). *Elementary Number Theory.* 6th ed. The McGraw-Hill Companies, Inc.

[7] BigPrimes. (n.d.). https://bigprimes.org/. Luettu: 23.5.2021.

[8] Diffie, W. & Hellman, M. (1976). *New directions in cryptography.* IEEE Transactions on Information Theory, Vol. 22, No. 6, 644-654.

[9] Fensham, P. (2009). *Real world contexts in PISA science: Implications for context-based science education.* Journal of Research in Science Teaching, Vol. 46, No. 8, 884-896. DOI: 10.1002/tea.20334

[10] Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). *Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming.* Review of Educational Research, Vol. 87, No. 4, 834-860. DOI: 10.3102/0034654317710096

[11] Campbell, S. & Zazkis, R. (2002). *Learning and Teaching Number Theory: Research in Cognition and Instruction*. Ablex Pub.

[12] Edita. (2021). www.editapublishing.fi/oppimateriaalit/perusopetus/kirjasarjat/sade. Luettu: 28.4.2021.

[13] Haapakangas, E. (2020). *Yläkoulun ohjelmoinnin opetuksen kehittäminen oppimateriaalin keinoin*. Department of Mathematics and Statistics, University of Helsinki. Maisterintutkielma.

[14] Hickmott, D., Prieto-Rodriguez, E. & Holmes, K. (2017). *A Scoping Review of Studies on Computational Thinking in K-12 Mathematics Classrooms*. Digital Experiences in Mathematics Education 4, 48-69. DOI: 10.1007/s40751-017-0038-8

[15] Hoffstein, J., Pipher, J., & Silverman, J. (2014). *An introduction to mathematical cryptography*. 2nd ed. Springer. DOI: 10.1007/978-1-4939-1711-2

[16] Kankaanranta, M., Lehto, M. & Neittaanmäki, P. (2014). *Kohti laskennallisen ajattelun osaamista*. Informaatioteknologian tiedekunnan julkaisuja No. 14/2014.

[17] Kong, S., & Abelson, H. (2019). *Computational Thinking Education*. 1st ed. Springer.

[18] Lenstra, A. (2000). *Integer Factoring*. Designs, Codes and Cryptography 19, 101-128. DOI: 10.1023/A:1008397921377

[19] Mannila, L. (2009). *Teaching Mathematics and Programming: New Approaches with Empirical Evaluation*. TUCS Dissertations 124. Turku Centre for Computer Science.

[20] Mannila, L. & Raadt, M. (2006). *An objective comparison of languages for teaching introductory programming*. Baltic Sea '06.

[21] Mannila, L., Peltomäki, M., Back, R. & Salakoski, T. (2006). *Why Complicate Things? Introducing Programming in High School Using Python*. Conferences in Research and Practice in Information Technology Series. 52.

[22] Opetushallitus. (2019). *Lukion opetussuunnitelman perusteet*. Määräykset ja ohjeet, 2019:2a.

[23] Opetushallitus. (2015). *Lukion opetussuunnitelman perusteet*. Määräykset ja ohjeet, 2015:48.

[24] Opetushallitus. (2014). *Perusopetuksen opetussuunnitelman perusteet*. Määräykset ja ohjeet, 2014:96.

[25] Opetusministeriö. (2007). *Laskennallisen tieteen kehittäminen Suomessa*. Opetusministeriön työryhmämuistioita ja selvityksiä, 2007:23.

[26] Pólya, G. (1945). *How to solve it: A new aspect of mathematical method*. Princeton, NJ: Princeton University Press.

[27] PrimePages. (n.d.). *The Nth Prime Page*. PrimePages. https://primes.utm.edu/nthprime/index.php. Luettu: 18.5.2021.

[28] Rossum, G. (1999). *Computer Programming for Everybody*. CNRI: Corporation for National Research Initiatives.

[29] Saksman, E. (2019). *Introduction to Number Theory*. Department of Mathematics and Statistics, University of Helsinki.

[30] Sanoma Pro. (2021). www.sanomapro.fi/sarjat/kuutio/. Luettu: 28.4.2021.

[31] Schoenfield, A. (1992). *Learning to think mathematically: Problem-solving, metacognition, and sense-making in mathematics*. In D. Grouws (Ed.), Handbook for research on mathematics teaching and learning , 334-370. New York, NY: MacMillan.

[32] Stepanov, A. & Rose, D. (2015). *From Mathematics to Generic Programming*. Third printing. Pearson Education, Inc.

[33] Summerfield, M. (2009). *Programming in Python 3: A Complete Introduction to the Python Language*. 2nd ed. Addison-Wesley Professional.

[34] Tiobe. (2021). Tiobe index for April 2021. https://www.tiobe.com/tiobe-index/. Luettu: 30.4.2021.

[35] Trinket. (n.d.). *Trinket.io*. https://trinket.io/. Luettu: 20.5.2021.

[36] Weintrop, D., Beheshti E., Horn, M., Orton, K., Jona, K., Trouille, L. & Wilensky, U. (2016). *Defining Computational Thinking for Mathematics and Science Classrooms*. Journal Of Science Education And Technology 25, 127-147. DOI: 10.1007/s10956-015-9581-5

[37] Wiener, M. (1990). *Cryptanalysis of short RSA secret exponents*. IEEE Transactions on Information Theory, 36, 553-558.

[38] Wing, J. (2006). *Computational Thinking*. Communications of the ACM, 49, 33-35. DOI: 10.1145/1118178.1118215.