Heli Hyttinen

Tietomassan analysointi ja visualisointi Python

2021

Kaakkois-Suomen ammattikorkeakoulu

SISÄLLYS

1 FREKVENSSITAULU JA TUNNUSLUKUJA 3

1.1 Frekvenssitaulu ja tunnuslukuja....4

2 FREKVENSSITAULU JA MOODI....5

2.1 Pylväs- ja piirakkadiagrammit....6

3 TEKSTIN ARVIOINTI PYTHONILLA....8

4 HISTOGRAMMI JA MOODILUOKKA EXCEL-TIEDOSTOSTA....10

5 TIEDOSTOKOON LUOKAN SELVITYS JA YHTEYSAJAT....12

6 GRAAFIN G SOLMUJEN… 15

1 FREKVENSSITAULU JA TUNNUSLUKUJA

Muuttujan sijoitus arvoiksi on annettu seuraavat:
a, a, a, a, a, b, b, b, c, c, a, a, d, d, a, c, c, e, e, e, f, a, a, c, c, b, b. Jossa a tarkoittaa sijoitusta 1, b sijoitusta 2, c sijoitusta 3, jne.

Muodostan frekvenssitaulun (taulukko 1) sijoittamalla taulun ensimmäiseen sarakkeeseen kaikki muuttujan sijoitus arvot pienimmästä suurimpaan eli a – f, jos arvo a vastaa lukua 1 ja arvo f vastaa lukua 6. Toiseen sarakkeeseen lasketaan vastaavan arvon frekvenssi, eli kuinka monta kertaa kyseinen arvo esiintyy muuttujan arvona. Esimerkiksi arvo f esiintyy vain kerran, joten sen frekvenssi on 1. Kolmanteen sarakkeeseen lasken summafrekvenssin, joka kuvaa arvojen kertymää lukumäärillä. Viimeisen arvon f kohdalla näkyy arvojen yhteismäärä, eli 27. Suhteellista frekvenssiä en tähän laske, sillä arvoja on niin vähän.

Sijoitusarvo	**Frekvenssi**	**Summa-frekvenssi**
a	10	10
b	5	15
c	6	21
d	2	23
e	3	26
f	1	27

Taulukko 1. Frekvenssitaulu.

Moodi on muuttujan yleisin arvo, eli se jonka frekvenssi on suurin. Tässä tapauksessa moodi on arvo a, sen frekvenssin ollessa 10, joka on suurin verrattuna muiden arvojen frekvensseihin.

Mediaani on suuruusjärjestetyn listan keskimmäinen arvo. Arvoja on tässä tapauksessa parillinen määrä, joten mediaaneja ovat kaksi keskimmäistä arvoa eli c ja d.
a, b, c, d, e, f.

Mediaanin voisi myös laskea muuttamalla arvot a – f luvuiksi 1–6 ja laskea keskimmäisten arvojen 3 ja 4 keskiarvo eli 3,5. (3+4) /2 = 3,5. Mutta kyseisessä tapauksessa arvot ovat sijoituksia, eikä sijoitus voi olla 3,5, vaan sijoitus on joko 3 tai 4. Sijoitus 3 esiintyy arvoissa useimmin kuin sijoitus 4, joten mediaani voisi olla 3. Lisäksi, jos yhtä sijoitusarvoa f ei olisi lainkaan, mediaani olisi selkeästi 3.

Kvartiilivälin selvittämiseksi tulee ensin selvittää ylä- ja alakvartiilit. Alakvartiili on sillä paikalla oleva arvo, jonka alapuolella on 25 % arvoista ja yläkvartiili sillä paikalla oleva arvo, jonka alapuolella on 75 % arvoista.

Jakamalla arvot 1, 2, 3, 4, 5, 6 kahteen osaan, toiseen osaan jää arvot 1, 2, 3 ja toiseen 4, 5, 6. Alakvartiili on 2, sen ollessa ensimmäisen osan mediaani. Yläkvartiili on 5, sen ollessa toisen osan mediaani. Arvojen lukumäärä on 6 ja 25 % kuudesta on 1,5 sekä 75 % kuudesta on 4,5.

Kvartiiliväli on ala- ja yläkvartiilin muodostama väli eli [2, 5]. Kvartiilivälin pituus on ylä- ja alakvartiilin erotus eli 3.

1.1 Frekvenssitaulu ja tunnuslukuja

On annettu seuraava reaalilukujen joukko: 4, 1, 2, 5, 6, 10, 2, 15, 16, 20. Lukuja on yhteensä 10.

Lasken arvojen aritmeettisen keskiarvon $\bar{x}$ laskemalla arvot yhteen ja sen jälkeen jakamalla saadun summan arvojen lukumäärällä 10.

$\bar{x} = \frac{4+1+2+5+6+10+2+15+16+20}{10} = \frac{81}{10}$ = 8.1. Eli $\bar{x}$ = 8,1.

Seuraavaksi lasken keskihajonnan saadun keskiarvon avulla. Aloitan laskemisen laskemalla jokaisen arvon poikkeaman keskiarvosta, eli $x_i - \bar{x}$.

$4 - 8{,}1 = -4{,}1$

$1 - 8{,}1 = -7{,}1$

$2 - 8{,}1 = -6{,}1$

$5 - 8{,}1 = -3{,}1$

$6 - 8{,}1 = -2{,}1$

$10 - 8{,}1 = 1{,}9$

$2 - 8{,}1 = -6{,}1$

$15 - 8{,}1 = 6{,}9$

$16 - 8{,}1 = 7{,}9$

$20 - 8{,}1 = 11{,}9$

Seuraavaksi lasken poikkeamien neliöt ja niiden keskiarvon eli varianssin.

$-(4{,}1)^2 = 16{,}81$

$-(7{,}1)^2 = 50{,}41$

$-(6{,}1)^2 = 37{,}21$

$-(3{,}1)^2 = 9{,}61$

$-(2{,}1)^2 = 4{,}41$

$1{,}9^2 = 3{,}61$

$-(6{,}1)^2 = 37{,}21$

$6{,}9^2 = 47{,}61$

$7{,}9^2 = 62{,}41$

$11{,}9^2 = 141{,}61$

Varianssi saadaan laskemalla saadut tulokset yhteen ja jakamalla luvulla 10.

$$\frac{16{,}81 + 50{,}41 + 37{,}21 + 9{,}61 + 4{,}41 + 3{,}61 + 37{,}21 + 47{,}61 + 62{,}41 + 141{,}61}{10}$$

= $\frac{410{,}90}{10}$ = 41,09. Varianssista saadaan laskettua keskihajonta ottamalla luvusta 41,09 neliöjuuri. $\sigma_x = \sqrt{41{,}09} \approx 6$. Eli keskihajonta σ_x on 6.

2 FREKVENSSITAULU JA MOODI

Kuvassa 1, sivu 7, määrään frekvenssitaulun sekä moodin annetusta havaintoaineistosta Python-ohjelmointikielellä.

```
In [1]: import numpy as np

        #koko autoaineisto listana
        hh_data = ["Volkkari", "Volkkari", "Mersu", "Viiatti", "Tojota",
                   "Sitikka", "Sitikka", "Sitikka", "Volkkari", "Viiatti",
                   "Viiatti", "Mersu", "Poso", "Poso", "Tojota",
                   "Poso", "Mersu", "Poso", "Viiatti", "Tatsun",
                   "Tojota", "Tatsun", "Viiatti", "Volkkari", "Tojota",
                   "Tojota", "Volkkari"]

        #listan eri alkiot joukkona hh_autot, aakkosjärjestyksessä
        hh_autot = sorted(set(hh_data))
        hh_autot

Out[1]: ['Mersu', 'Poso', 'Sitikka', 'Tatsun', 'Tojota', 'Viiatti', 'Volkkari']

In [2]: #frekvenssitaulu
        #listan hh_frekv sisällä for-loop,
        #löytää frekvenssit eli montako jokaista autoa (hh_autot) on aineistossa (hh_data)
        hh_frekv = [hh_data.count(x) for x in hh_autot]
        hh_frekv

Out[2]: [3, 4, 3, 2, 5, 5, 5]

In [3]: #etsin listan hh_frekv suurimmat luvut
        #moodeja on kolme, joten käytän argwhere ja amax, jotta kaikki moodien paikat löytyvät

        hh_ind = np.argwhere(hh_frekv == np.amax(hh_frekv))

        #laitan löydetyt paikka-arvot saman listan sisälle
        hh_ind = hh_ind.flatten().tolist()
        hh_ind

Out[3]: [4, 5, 6]

In [5]: #moodien paikat ovat 4, 5 ja 6 listassa hh_autot
        #teen moodi listan ja käyn läpi listan hh_ind jossa on moodien paikat,
        #ja lisään vastaavat autojen nimet moodilistaan
        hh_moodi = []
        for i in range(len(hh_ind)):
            hh_moodi.append(hh_autot[hh_ind[i]])

        hh_moodi

Out[5]: ['Tojota', 'Viiatti', 'Volkkari']
```

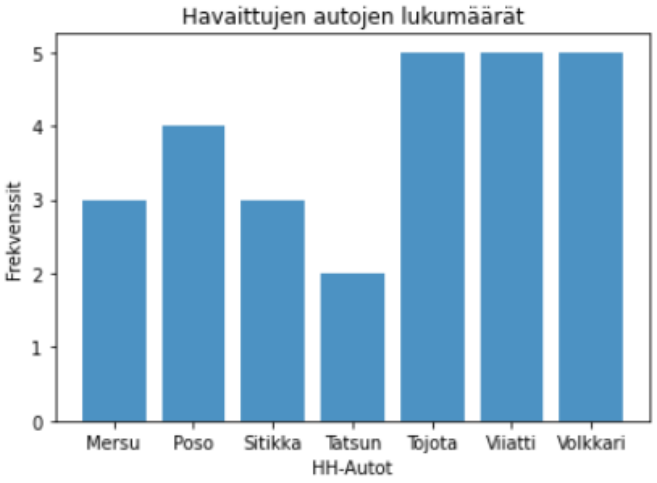
Kuva 1. Frekvenssitaulu ja moodi, kuvakaappaus.

Koodissa on kaksi listaa, toisessa koko aineisto ja toisessa kaikki aineiston eri arvot. Koodi laskee montako jokaista arvoa koko aineistossa on ja näin muodostuu frekvenssitaulu, josta koodi etsii suurimmat frekvenssit ja niiden paikka-arvot. Paikka-arvojen avulla koodi löytää moodit listasta, jossa on alkuperäisen aineiston eri arvot. Moodeja on kolme tässä aineistossa.

2.1 Pylväs- ja piirakkadiagrammit

Kuvassa 2, sivu 8, havainnollistan frekvenssejä pylväsdiagrammin avulla.

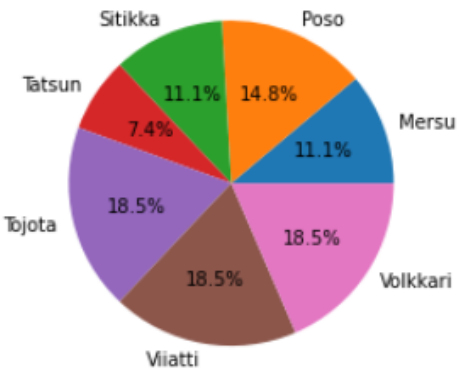
In [6]:
```
import matplotlib.pyplot as plt

#havainnollistan frekvenssit pylväsdiagrammilla
plt.bar(hh_autot,hh_frekv,align = 'center', alpha = 0.8)
plt.xticks(hh_autot)
plt.xlabel("HH-Autot")
plt.ylabel('Frekvenssit')
plt.title("Havaittujen autojen lukumäärät")
plt.savefig("t_3pylvas.jpg")
plt.show()
```



Kuva 2. Pylväsdiagrammi.

Kuvassa 3 havainnollistan frekvenssejä piirakkadiagrammin avulla.

In [7]:
```
#havainnollistan frekvenssit piirakkadiagrammilla
plt.pie(hh_frekv,labels = hh_autot,autopct = '%1.1f%%')
plt.savefig("t_3piirakka.jpg")
plt.show()
```



Kuva 3. Piirakkadiagrammi.

Kokeilin skriptiä muuttamalla autoaineiston arvoja, toimii.

3 TEKSTIN ARVIOINTI PYTHONILLA

Kuvassa 4 ja 5 luen annetun tekstitiedoston, parsin tekstiä, typistän sanoja ja lasken negatiivisten sekä positiivisten sanojen määrät annetusta tekstistä Python-ohjelmointikielellä.

In [1]:
```
import pandas as pd

#luetaan tekstitiedosto Arvio.txt
hh_arvio_txt = pd.read_csv('Arvio.txt',delimiter = '\n',header = None, keep_default_na = False, encoding = 'U

#alkuperäisen tiedoston yksi rivi muodostaa listan
#"Series" parsii listan ja jakaa listan merkkijonon sanat erilleen omiksi merkkijonoikseen splitin avulla
#muodostuu kaksiuloitteinen lista, merkkijonot listojen sisällä ja listat listan sisällä
hh_arvio_list = [pd.Series(hh_arvio_txt.values[i]).str.split() for i in range(len(hh_arvio_txt.values))]

#otsikko on listan ensimmäinen
hh_otsikko_list = hh_arvio_list[0][0]
#otsikko-lista yhdeksi merkkijonoksi "join" avulla
hh_otsikko = ' '.join([str(i) for i in hh_otsikko_list])
#poistetaan vielä ":" otsikon nimestä "strip" avulla
hh_otsikko = hh_otsikko.strip(':')
print(hh_otsikko)

#sanojen analysoimiseksi,
#kaikki sanat samaan listaan käymällä kaikki rivit läpi ja lisäämällä merkkijonot listaan pienin kirjaimin
hh_apuarray = []
for i in range(len(hh_arvio_list)):
    hh_apuarray = hh_apuarray + hh_arvio_list[i][0]
hh_sanat = list(pd.Series(hh_apuarray).str.lower())

#poistetaan merkit sanojen lopusta, eli . ja ,
hh_csanat = [hh_sanat[i].strip('.') for i in range(len(hh_sanat))]
hh_csanat = [hh_csanat[i].strip(',') for i in range(len(hh_sanat))]
```

Kuva 4. Tekstin arviointi, osa 1/3.

Annettu tekstitiedosto luetaan pandas-kirjaston avulla, tekstitiedostossa yksi rivi muodostaa listan. Koodi parsii listan merkkijonon sanat erilleen omiksi merkkijonoikseen ja muodostaa yhden kaksiulotteisen listan, jossa merkkijonot ovat listojen sisällä ja listat listan sisällä. Listan ensimmäinen lista on tekstitiedoston otsikko. Koodi muodostaa otsikon listan yhdeksi merkkijonoksi ja poistaa siitä kaksoispisteen ”:”, sekä tulostaa otsikon.

Koodi käy läpi kaikki listat, eli tekstirivit tekstitiedostosta, muuttaa jokaisen merkkijonon kirjaimet pieniksi kirjaimiksi ja lisää merkkijonot uuteen listaan. Näin saadaan yksi lista, jossa kaikki alkuperäisen tekstin sanat ovat omina merkkijonoinaan pienin kirjaimin. Tämän jälkeen koodi poistaa mahdolliset pisteet ja pilkut sanojen lopusta.

```
#sanojen typistämiseksi, tuodaan snowballin FinnishStemmer
from nltk.stem.snowball import FinnishStemmer
st = FinnishStemmer()

#typistetään sanat uuden listan sisälle, eli muutetaan mahdollisimman perusmuotoon
hh_st_csanat = [st.stem(i) for i in hh_csanat]

#etsittävät sanat typistetään, jotta ne ovat saman muotoisia kuin tekstissä,
#sanojen taivutukset vaikuttavat erityisesti typistämiseen
#esim. taitava on monikossa taitavat, muoto ei juuri muutu
#esim. kaunis on monikossa kauniit, muoto muuttuu merkittävästi stemmauksen kannalta

#etsittävät positiiviset ja negatiiviset sanat, taivutettuna eri tavoin
hh_pos = ["mielenkiintoinen", "mielenkiintoiset", "kaunis", "kauniit", "kauniisti", "erinomainen",
          "erinomaiset", "loistava", "herkka", "taitava"]
hh_neg = ["tylsa", "ruma", "huono", "kylma", "tasapaksu", "taitamaton", "taitamattomat"]

#etsittävien sanojen typistys
hh_st_pos = [st.stem(i) for i in hh_pos]
hh_st_neg = [st.stem(i) for i in hh_neg]

#lasketaan positiivien sanojen määrä arvioitavasta tekstistä
#arvioitava teksti sekä etsittävät sanat eri muodoissa ovat typistetty
a = 0
for i in range(len(hh_st_pos)):
    if hh_st_pos[i] in hh_st_csanat:
        a = a+1

#lasketaan negatiivisten sanojen määrä samalla tavalla
b = 0
for i in range(len(hh_st_neg)):
    if hh_st_neg[i] in hh_st_csanat:
        b = b+1
```

Kuva 5. Tekstin arviointi, osa 2/3.

Snowball-stemmerin avulla typistetään tekstin sanoja mahdollisimman perusmuotoisiksi. Lisätään koodiin listat positiivista ja negatiivisista sanoista, joita on tarkoitus etsiä tekstistä. Positiiviset ja negatiiviset sanat ovat taivutettu listaan eri muodoissa, jotta niiden typistämisen jälkeen sanat löytyvät tekstistä todennäköisemmin. Koodi typistää nämä etsittävät sanat ja käy läpi arvioitavan tekstin sekä typistetyt positiiviset sanat ja typistetyt negatiiviset sanat, laskee näiden määrät arvioitavassa tekstissä ja tallentaa määrät muuttujiin a ja b.

Kuvassa 6 koodi arvioi laskettujen positiivisten ja negatiivisten sanojen määrien perusteella onko tekstiarvio positiivinen vai negatiivinen sekä näyttää tulosteet.

```
#jos b jaettuna a:lla on enemmän kuin kaksi, arvio elokuvasta on positiivinen
ab = a / b
if ab > 2:
    print('Tämä arvio em. elokuvasta on positiivinen')
else:
    print('Tämä arvio em. elokuvasta ei ole positiivinen')
```

```
Stan & Ollie
Tämä arvio em. elokuvasta on positiivinen
```

Kuva 6. Tekstin arviointi, osa 3/3.

Etsittävät sanat ovat samat mitä tehtävänannossa on kuvattu. Kokeilin skriptiä vielä muuttamalla arvioitavaa tekstiä, mutta käyttäen samoja positiivisia ja negatiivisia sanoja.

4 HISTOGRAMMI JA MOODILUOKKA EXCEL-TIEDOSTOSTA

Kuvassa 7, sivu 11, luen annetun Excel-tiedoston, luokittelen kyseisen tiedoston A-sarakkeen arvot viiteen luokkaan tasavälisesti, selvitän frekvenssit ja moodiluokan ja havainnollistan A-sarakkeen tiedot histogrammin avulla.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#luetaan excel-tiedosto Loki.
#sarakkeen A muuttujat ovat tiedoston kokoja
hh_loki = pd.read_excel('Loki.xlsx', header = None)
hh_A = hh_loki.values.T[0]

#luokitellaan muuttujan A arvot viiteen luokkaan tasavälisesti
#eli A:n arvoista suurin luku jaettuna viidellä olisi ensimmäisen luokkaraja yläraja
#8390/5=1678, arvot jotka esiintyvät välillä 0-1678 (1678 ei sisälly) kuuluvat ensimmäiseen luokkarajaan
#1678 lisättynä 1678 on seuraava luokkaraja eli 3356. Seuraavaan luokkarajaan kuuluu 1678 ja 3356 väli,
#mutta 3356 ei sisälly rajaan, 1678 sisältyy.

#hh_bins kertoo luokkarajat
#hh_freqs kertoo luokkarajojen frekvenssit, kuinka monta arvoa luokkarajassa esiintyy
hh_freqs, hh_bins = np.histogram(hh_A, 5)

#Luodaan histogrammi
plt.title('Histogrammi ladattavien tiedostojen koosta')
plt.xlabel('Tiedoston koko (mt)')
plt.ylabel('Frekvenssit')
plt.hist(hh_A,hh_bins)
plt.show()

#Selvitetään moodiluokka eli luokka jonka frekvenssi on suurin
hh_moodi = np.argmax(hh_freqs)
#Moodiluokan rajat
print('Moodiluokan alaraja on ', hh_bins[hh_moodi])
print('Moodiluokan yläraja on ', hh_bins[hh_moodi+1])
```


```
Moodiluokan alaraja on  0.0
Moodiluokan yläraja on  1678.0
```

Kuva 7. Histogrammi ja moodiluokka Excel-tiedostosta.

Koodi lukee annetun Excel-tiedoston ja tallentaa tiedoston ensimmäisen sarakkeen muuttujaan A, josta numpy-kirjaston histogrammiominaisuuden avulla koodi jakaa A-muuttujan arvot viiteen luokkaan tasavälisesti ja selvittää luokkien frekvenssit.

Viiteen luokkaan jako tapahtuu käytännössä jakamalla A:n arvoista suurin luku viidellä, jonka tulo olisi ensimmäisen luokan yläraja. Eli tässä tapauksessa A-muuttujan suurin arvo on 8390, joka jaetaan viidellä ja saadaan ensimmäisen luokan ylärajaksi 1678. Arvot, jotka esiintyvät välillä 0-1678, mutta ei 1678, kuuluvat ensimmäiseen luokkaan. Seuraavaan luokkarajaan kuuluu 1678 ja 3356 väli ja 1678, mutta ei 3356. Ja niin edelleen. Viimeiseen luokkarajaan sisältyy molemmat raja-arvot, eli myös 8390.

Koodissa histogrammi luodaan matplotlib.pyplot-kirjaston avulla ja moodiluokka selvitetään numpy-kirjaston avulla etsimällä suurin frekvenssi ja sitä vastaavat luokkarajat. Kokeilin koodin toimivuutta vielä erilaisilla arvoilla.

5 TIEDOSTOKOON LUOKAN SELVITYS JA YHTEYSAJAT

Kuvassa 8 ja 9 luen annetun Excel-tiedoston, joka on sama kuin aiemmassa tehtävässä, selvitän mihin aiemmassa tehtävässä määrättyyn luokkaan tiedosto kooltaan 5,5 GB kuuluu. Lisäksi poimin selvitetyn luokan yhteysajat, joista lasken keskiarvon ja keskihajonnan, joiden perusteella arvioin kannattaako tiedostoa ladata annetussa ajassa 8 minuuttia.

In [1]:

```
import pandas as pd
import numpy as np

hh_loki = pd.read_excel('Loki.xlsx', header = None)
hh_A = hh_loki.values.T[0]
hh_freqs, hh_bins = np.histogram(hh_A, 5)

#aika minuutteina, t
hh_t = 8
#muutetaan aika sekunneiksi, jotta voidaan verrata tiedoston Loki.xlsx aikoihin, -
#sillä ne ovat esitetty sekunteina
hh_t = hh_t * 60

#tiedoston koko gigatavuina
hh_gb = 5.5
#tiedoston koko megatavuina, jotta vastaavat koot löytyvät Loki-tiedostosta
hh_mb = hh_gb * 1024

#tarkistetaan funktiolla mihin edellisen tehtävän luokkaan tiedostokoko kuuluu
#funktio palauttaa luokan rajat
hh_apuarray1 = []
def luokanTarkistus(hh_mb):
    for i in range(len(hh_bins)):
        #jos sama tai suurempi kuin ensimmäisen luokan alaraja ja -
        #pienempi kuin ensimmäisen luokan yläraja, palautetaan ensimmäisen luokan luokkarajat
        if hh_mb >= hh_bins[0] and hh_mb < hh_bins[1]:
            hh_apuarray1 = [hh_bins[0], hh_bins[1]]
            return hh_apuarray1
        #jos sama tai suurempi kuin toisen luokan alaraja ja -
        #pienempi kuin toisen luokan yläraja, palautetaan toisen luokan luokkarajat. jne.
        #viimeiseen luokkaan kuuluu myös sen luokan ylärajan arvo
        if hh_mb >= hh_bins[1] and hh_mb < hh_bins[2]:
            hh_apuarray1 = [hh_bins[1], hh_bins[2]]
            return hh_apuarray1
        if hh_mb >= hh_bins[2] and hh_mb < hh_bins[3]:
            hh_apuarray1 = [hh_bins[2], hh_bins[3]]
            return hh_apuarray1
        if hh_mb >= hh_bins[3] and hh_mb < hh_bins[4]:
            hh_apuarray1 = [hh_bins[3], hh_bins[4]]
            return hh_apuarray1
        if hh_mb >= hh_bins[4] and hh_mb <= hh_bins[5]:
            hh_apuarray1 = [hh_bins[4], hh_bins[5]]
            return hh_apuarray1

#funktion kutsu
hh_luokka = luokanTarkistus(hh_mb)

#kun luokka on saatu selville, poimitaan luokkaa vastaavat yhteysajat-
#käymällä läpi koko Loki-tiedosto
hh_apuarray2 = []
for i in range(len(hh_loki.values)):
    #käydään läpi lokitiedoston a sarakkeen transpoosi, eli tiedostojen koot
    #poimitaan luokkaan kuuluvien tiedostokokojen lataamiseen käytetyt yhteysajat -
    #eli jos transpoosin arvo on isompi tai sama kuin luokan alaraja -
    #ja pienempi kuin luokan yläraja -
    #samalla paikalla oleva yhteysaika sarakkeen b transpoosista poimitaan listaan
    if hh_loki.values.T[0][i] >= hh_luokka[0] and hh_loki.values.T[0][i] < hh_luokka[1]:
        #sarakkeen b transpoosi, eli lataamisen käytetyt yhteysajat
        hh_apuarray2 = hh_apuarray2 + [hh_loki.values.T[1][i]]
    hh_yhteysajat = hh_apuarray2
```

Kuva 8. Tiedostokoon luokan selvitys ja yhteysajat, osa 1/2.

Koodi käyttää aiemman tehtävän luokkia. Koodi saa ajan minuutteina ja tiedoston koon gigatavuina. Minuutit muutetaan sekunneiksi ja gigatavut muutetaan megatavuiksi, jotta vastaava aika ja tiedoston koko löytyy analysoitavasta Excel-tiedostosta.

Koodi tarkastaa funktiolla mihin luokkaan annettu tiedoston koko kuuluu, funktio palauttaa luokan rajat muuttujaan luokka. Funktio saa tiedoston koon ja if-lausekkeella selvittää vastaavan luokan. Jos tiedoston koko on sama tai suurempi kuin ensimmäisen luokan alaraja ja pienempi kuin ensimmäisen luokan yläraja, funktio palauttaa ensimmäisen luokan luokkarajat. Jos tiedoston koko ei sovi tähän ehtoon, funktio selvittää onko tiedoston koko sama tai suurempi kuin toisen luokan alaraja ja pienempi kuin toisen luokan yläraja ja niin edelleen.

Kun tiedoston koon luokka on saatu selville, poimitaan luokkaa vastaavat yhteysajat listaan käymällä läpi koko Excel-tiedosto. Koodi käy läpi Excel-tiedoston ensimmäisen sarakkeen transpoosin, eli tiedostojen koot ja poimii aiemmin selvittyä luokkaa vastaavat arvot Excel-tiedoston toisen sarakkeen transpoosista, eli tiedostokokojen lataamiseen käytetyt yhteysajat, ja tallentaa yhteysajat listaan.

```
#lasketaan yhteysaikojen keskiarvo käymällä poimitut yhteysajat läpi ja -
#laskemalla ajat yhteen sekä laskemalla niiden yhteismäärän ja -
#jakamalla saatu summa poimittujen yhteysaikojen määrällä
hh_summa = 0
hh_maara = 0
for i in range(len(hh_yhteysajat)):
    hh_summa += hh_yhteysajat[i]
    hh_maara += 1
hh_keskiarvo = hh_summa / hh_maara

#lasketaan keskihajonta numpyn avulla
#keskiarvon olisi tietysti myös voinut laskea np.mean(hh_yhteysajat),
#mutta tein hienon loopin niin antaa mennä sillä.
hh_keskihajonta = np.std(hh_yhteysajat)

#tarkistetaan onko annettu aika (sekunteina) suurempi kuin keskiarvon ja keskihajonnan summa
if hh_t > hh_keskiarvo + hh_keskihajonta:
    print("Ehkäpä kerkiät spruuttaamaan tiedoston")
else:
    print("Ei kannata näin kiireessä")
```

```
Ehkäpä kerkiät spruuttaamaan tiedoston
```

Kuva 9. Tiedostokoon luokan selvitys ja yhteysajat, osa 2/2.

Koodi laskee poimittujen yhteysaikojen keskiarvon käymällä yhteysajat läpi, laskemalla arvojen yhteismäärän ja laskemalla arvot yhteen, jonka jälkeen saatu summa jaetaan poimittujen yhteysaikojen yhteismäärällä. Keskihajonta lasketaan keskiarvosta numpy-kirjaston avulla. Koodi tarkistaa onko annettu aika sekunteina suurempi kuin keskiarvon ja keskihajonnan summa ja tulostaa vastauksen.

Kokeilin koodia antamalla alussa eri minuuttimäärän ja eri tiedostokoon. Toimii kokoon 8.19 GB saakka, eli 8390 mb jaettuna 1024 mb, mutta ei kai tarvinnutkaan toimia Loki-tiedostossa annettuja tiedostokokoja suuremmille tiedostoille.

6 GRAAFIN G SOLMUJEN ASTELUVUT, K-KLIKIT, KESKEISIN SOLMU JA GRAAFI ETÄISYYSMATRIISISTA

Kuvassa 10 luen annetun Excel-tiedoston, joka sisältää viereisyysmatriisin, muodostan matriisista graafin G, jonka solmujen asteluvut selvitän. Etsin graafin k-klikit, kun $k \geq 4$ ja lasken graafin G etäisyysmatriisin NetworkX-kirjaston avulla, josta selvitän solmujen suhteelliset läheisyysluvut kuvassa 11 ja erityisesti selvitän minkä solmun suhteellisin läheisyysluku on suurin. Tulosteet näkyvät kuvassa 11.

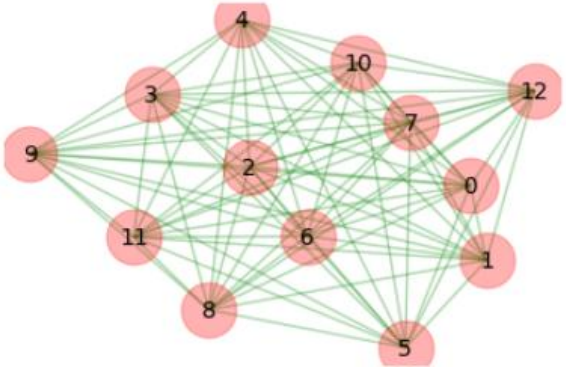
In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd

#luetaan excel-tiedosto Yhteys
hh_yhteys = pd.read_excel('Yhteys.xlsx', header = None)
#viereisyysmatriisi, sama transpoosinsa kanssa
hh_yhteys = hh_yhteys.values

#graafi G
hh_G = nx.from_numpy_matrix(hh_yhteys)

#tulostetaan solmujen asteluvut
hh_asteluvut = hh_G.degree
for i in range(len(hh_asteluvut)):
    print("Solmun ", i, " asteluku on ", hh_asteluvut[i])

#etsitään kaikki klikit networkx avulla
hh_klikit = list(nx.enumerate_all_cliques(hh_G))
#etsitään klikit, jossa on neljä tai enemmän solmua
hh_4klikit = [k for k in hh_klikit if len(k) >= 4]
print("\nk-klikit graafissa G, kun k >= 4: ", hh_4klikit,"\n")

#etäisyysmatriisi networkx avulla
hh_dgmatrix = nx.algorithms.shortest_paths.dense.floyd_warshall_numpy(hh_G, nodelist=None, weight='weight')

#suhteellinen läheisyysluku solmulle selvitetään jakamalla solmujen yhteismäärä-1 (eli 12) -
#solmun lyhyimpien polkujen pituuksien summalla
#graafin keskeisin solmu on se, jolla on suurin suhteellinen läheisyysluku
```

Kuva 10. Graafin G solmujen asteluvut, k-klikit ja etäisyysmatriisi.

Koodi muodostaa networkx-kirjaston avulla graafin ja tulostaa graafin asteluvut. Koodi etsii networkx-kirjaston avulla kaikki graafin klikit, jonka jälkeen etsii ja tulostaa klikit, joissa on neljä tai enemmän solmua. Tulosteet näkyvät kuvassa 11. Tämän jälkeen koodi muodostaa networkx-kirjaston avulla etäisyysmatriisin graafista G. Seuraavaksi selvitetään graafin G keskeisin solmu seuraavassa kuvassa (kuva 11).

```
#lasketaan etäisyysmatriisin avulla jokaisen solmun lyhyimpien polkujen pituudet yhteen
#käydään siis läpi etäisyysmatriisi ja lisätään summat listaan hh_lyhyetpolut_yht
hh_lyhyetpolut_yht = []
for i in range(len(hh_dgmatrix)):
    hh_lyhyetpolut_yht += [sum(hh_dgmatrix[i])]
#jaetaan G:n solmujen yhteismäärä-1 (13-1) aiemmin saaduilla summilla ja lisätään -
#tulos listaan hh_closeness, eli käydään hh_lyhyetpolut_yht-lista läpi for-loopilla
hh_G_solmut = len(hh_G)
hh_closeness = []
for i in range(len(hh_lyhyetpolut_yht)):
    hh_closeness += [(hh_G_solmut-1)/hh_lyhyetpolut_yht[i]]
#hh_closeness sisältää nyt G:n solmujen suhteelliset läheisyysluvut,
#niistä suurin on keskeisin solmu graafissa

hh_keskeisin = max(hh_closeness)
#käydään läpi for-loopilla hh_closeness ja lasketaan monesko arvoista on isoin
def MoneskoSolmu(hh_keskeisin):
    for i in range(len(hh_closeness)):
        if(hh_closeness[i]==hh_keskeisin):
            return i
hh_keskeisin_solmu = MoneskoSolmu(hh_keskeisin)
print("Graafissa G keskeisin solmu on ", hh_keskeisin_solmu)

#graafi etäisyysmatriisista
hh_dg = nx.from_numpy_matrix(hh_dgmatrix)
#positiot linkeille ja solmuille piirtoa varten
hh_pos = nx.spring_layout(hh_dg)
#piirretään solmut
nx.draw_networkx_nodes(hh_dg,hh_pos,node_color='red',node_size=1100,alpha=0.3)
#piirretään linkit
nx.draw_networkx_edges(hh_dg,hh_pos,width=1.5,edge_color='green',alpha=0.3)
#nimet solmuille
hh_labels = {0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10,11:11,12:12}
nx.draw_networkx_labels(hh_dg,hh_pos,hh_labels,font_size=14)
#kehys pois ja piirretään graafi etäisyysmatriisista
plt.axis('off')
plt.show()
```

```
Solmun  0  asteluku on  3
Solmun  1  asteluku on  2
Solmun  2  asteluku on  1
Solmun  3  asteluku on  1
Solmun  4  asteluku on  3
Solmun  5  asteluku on  5
Solmun  6  asteluku on  2
Solmun  7  asteluku on  4
Solmun  8  asteluku on  4
Solmun  9  asteluku on  4
Solmun  10  asteluku on  3
Solmun  11  asteluku on  2
Solmun  12  asteluku on  2

k-klikit graafissa G, kun k >= 4:  [[0, 7, 8, 10]]

Graafissa G keskeisin solmu on  5
```



Kuva 11. Graafin G solmujen asteluvut, k-klikit, keskeisin solmu ja graafi etäisyysmatriisista.

Suhteellinen läheisyysluku solmulle selvitetään jakamalla solmujen yhteismäärä vähennettynä yhdellä, eli tässä tapauksessa 12, solmun lyhyimpien polkujen pituuksien summalla. Ensin lasketaan etäisyysmatriisin avulla jokaisen solmun lyhyimpien polkujen pituudet yhteen. Koodi käy etäisyysmatriisin läpi ja lisää uuteen listaan jokaisen solmun lyhyimpien polkujen pituuksien summan. Seuraavaksi koodi luo taas uuden tyhjän listan ja lisää sinne jokaisen solmun suhteellisen läheisyysluvun käymällä läpi äsken luodun listan lyhyimpien polkujen pituuksien summista ja jakamalla luvun 12 solmua vastaavalla summalla. Kun koodi on selvittänyt kaikkien solmujen suhteellisen läheisyysluvun, ne löytyvät yhdestä listasta. Tämän listan suurin luku on graafin G keskeisimmän solmun ja sen arvo tallennetaan muuttujaan.

Koodi selvittää funktiolla suurinta arvoa vastaavan solmun käymällä läpi suhteellisten läheisyyslukujen listan ja vertaamalla listan arvoja äsken luotuun muuttujaan, eli muuttuja, jolle on tallennettu suurin suhteellinen läheisyysluku. Funktio palauttaa sen arvon paikka-arvon, joka vastaa muuttujaa suurimmalla suhteellisella läheisyysluvulla. Funktion palauttama paikka-arvo on sama kuin graafin G solmu, sillä solmujen nimet ovat koodissa numeroita 0-12. Koodi tulostaa graafin G:n keskeisimmän solmun (kuva 11). Lopuksi koodi muodostaa etäisyysmatriisista graafin (kuva 11). Kokeilin koodia myös muuttamalla viereisyysmatriisia.

Heli Hyttinen

Diskreetti matematiikka 2
Graafiteoria

2020

Kaakkois-Suomen
ammattikorkeakoulu

SISÄLLYS

1 $G = (V,E)$ ASTELUKUJEN SUMMA 3

1.1 Eulerin reitti 3

2 $G = (V,E)$ KAKSIJAKOISUUS 4

2.1 G :n virittävä puu 4

3 MINIMAALINEN VIRITTÄVÄ PUU 5

4 ABSTRAKTI SYNTAKSIPUU 7

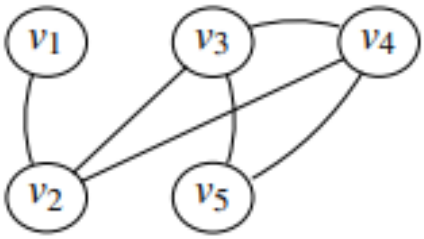
4.1 Puolalainen esijärjestys 9

5 3 NUOLEN MITTAISET SUUNNATUT KÄVELYT 9

5.1 3 nuolen mittaiset suunnatut kävelyt Pythonilla 11

6 ÄÄRELLINEN AUTOMAATTI A 12

1 $G = (V,E)$ ASTELUKUJEN SUMMA



Kuva 1. Graafi $G = (V,E)$.

Linkkejä on yhteensä 6, joten solmujen astelukujen summa on 2 kertaa 6 eli 12.

Lasken jokaisen solmun asteluvun;

$\deg(v1) = 1$

$\deg(v2) = 3$

$\deg(v3) = 3$

$\deg(v4) = 3$

$\deg(v5) = 2$

ja näiden summa on 12.

1 + 3 + 3 + 3 + 2 = 12

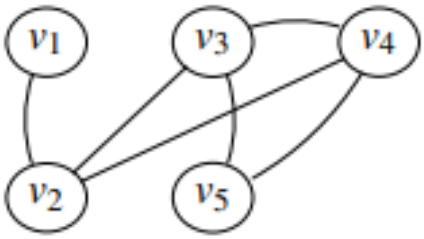
G:n solmujen astelukujen summa on 12.

1.1 Eulerin reitti

G:ssä ei ole Eulerin reittiä. G:n solmuista neljän solmun asteluku on pariton eli G:ssä on 4 kpl paritonta solmua. Jotta G:ssä olisi Eulerin reitti, siinä tulisi olla täsmälleen 2 kpl parittomia solmuja.

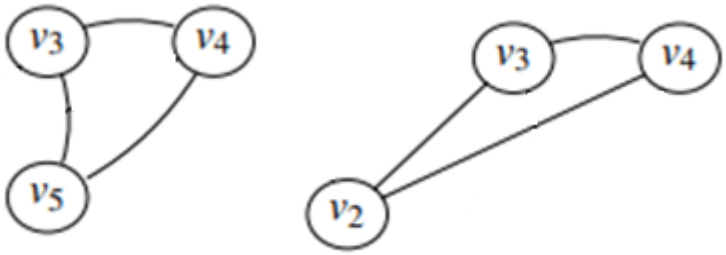
G.ssä ei myöskään ole reittiä, jossa jokainen linkki $e \in E$ esiintyisi.

2 $G = (V,E)$ KAKSIJAKOISUUS



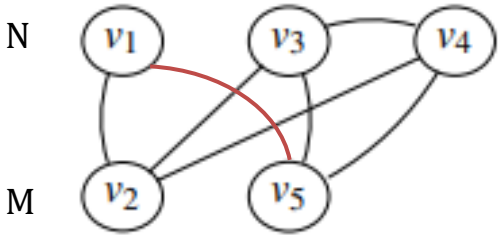
Kuva 2. Graafi $G = (V,E)$.

G ei ole kaksijakoinen. G:ssä on kaksi 3-sykliä. Jotta G olisi kaksijakoinen, sen ei tulisi sisältää yhtään paritonta sykliä ja 3-sykli on pariton sykli.



Kuva 3. G:n syklit.

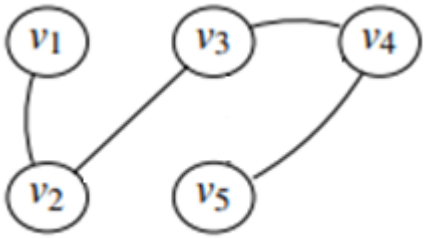
Lisäksi jos solmujoukko V voitaisi jakaa osajoukkoihin M ja N niin, että linkki yhdistäisi jokaisen M:n solmun solmuihin N, voisi G olla kaksijakoinen. $V1$ ja $v5$ välillä ei ole linkkiä, jotta tämä onnistuisi (kuva 4).



Kuva 4. Linkki $v1$ ja $v5$ välillä.

2.1 G :n virittävä puu

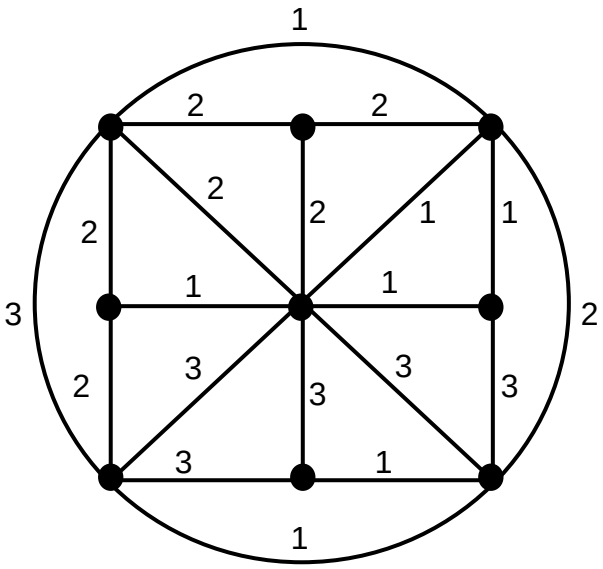
Kuvassa 5 on G:n virittävä puu. Jokainen G:n solmu esiintyy siinä.



Kuva 5. G:n virittävä puu.

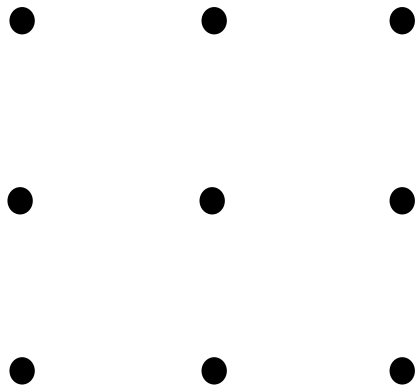
3 MINIMAALINEN VIRITTÄVÄ PUU

Kuvassa 6 esiintyy graafi G painoarvoineen, jolle määrään minimaalisen virittävän puun.

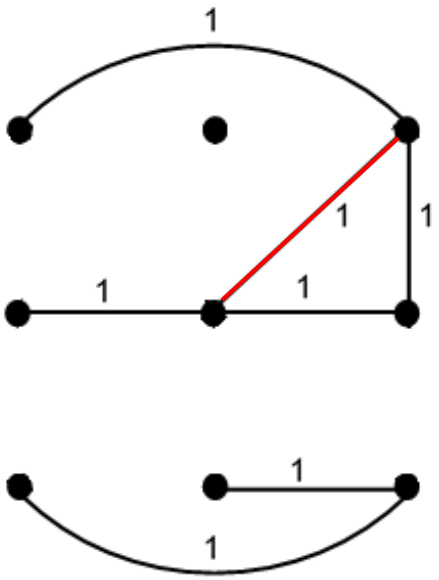


Kuva 6. Graafi G painoarvoineen.

Lähden liikkeelle yhdeksästä solmusta (kuva 7) ja lisään pienimmän, eli 1, painoarvon omaavat linkit solmujen väliin.

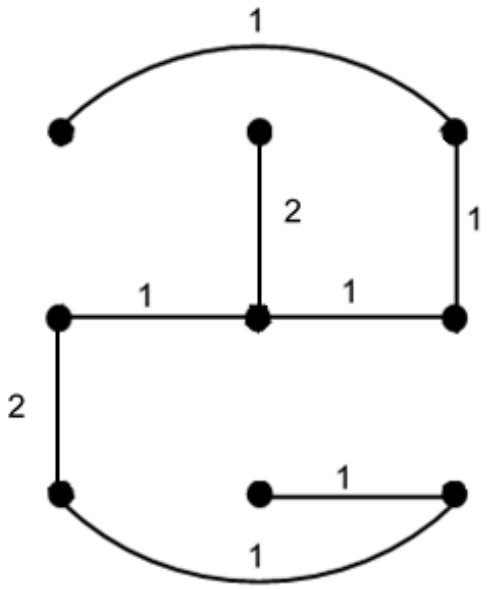


Kuva 7. G:n solmut.



Kuva 8. Linkit joilla on pienimmät painoarvot.

Vielä poistetaan linkki, jonka merkitsin punaisella kuvaan 8, jotta saadaan graafista syklitön. Lisäksi yksi solmu on kokonaan yhdistämättä ja graafi ei ole vielä yhtenäinen. Lisätään linkit, jotta graafista tulee yhdistetty, sen mukaan millä linkeillä on seuraavaksi pienimmät painoarvot, eli arvo 2. Kuvassa 9 on tehty nämä muutokset tuloksena G:n minimaalinen virittävä puu.



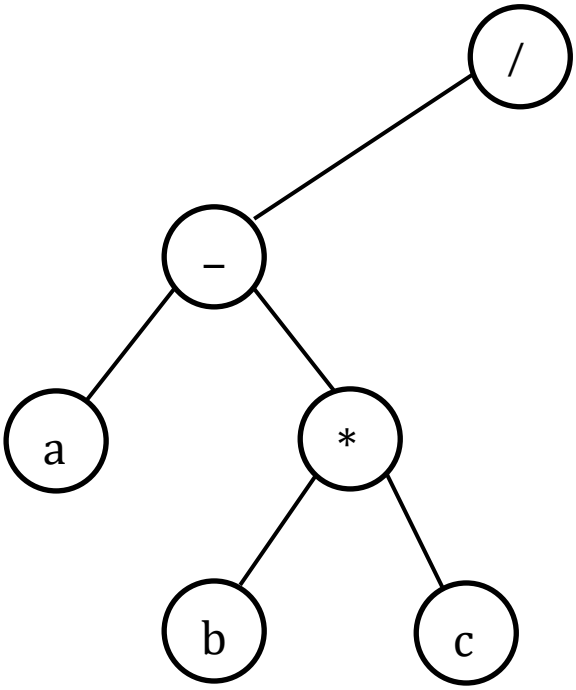
Kuva 9. G:n minimaalinen virittävä puu.

Painoarvojen summa on 10.

4 ABSTRAKTI SYNTAKSIPUU

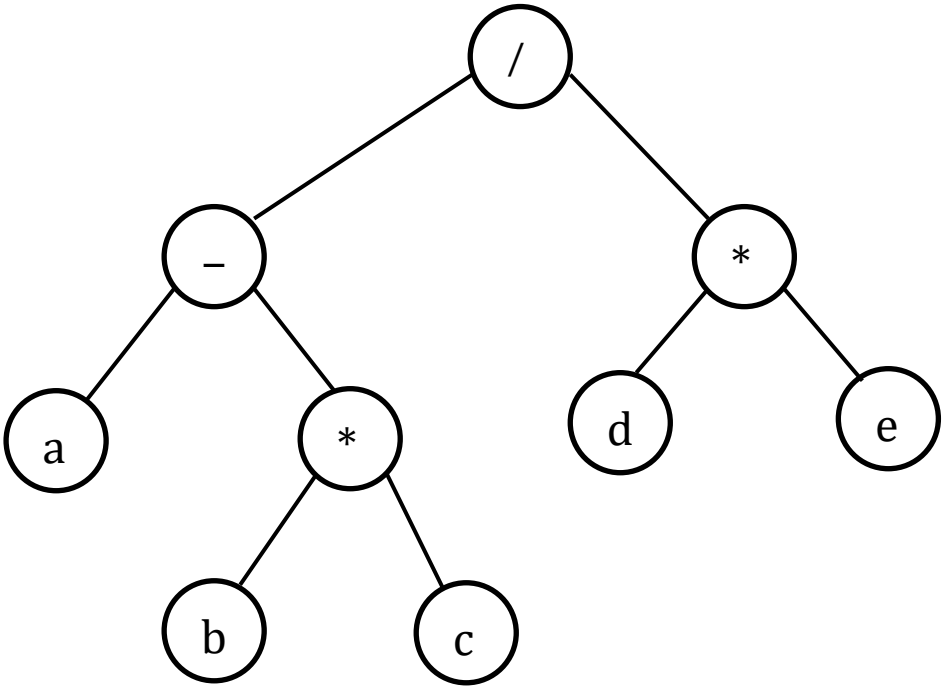
Muodostan laskutoimituksesta $(a-(b*c))/(d*e)$ abstraktin syntaksipuun. Operaattorit tarkoittavat kaksipaikkaisia laskutoimituksia ja operaattori / jakaa koko laskutoimituksen aluksi kahteen osaan ollessa kaikkien ”isäsolmu”.

Jatkan puun muodostamista lukien laskutoimitusta operaattorin / vasemmalta puolelta, jossa seuraava operaattori on −, joka on ”isäsolmu” operandeille a ja $(b*c)$. Tässä $(b*c)$ operaattori * on b:n ja c:n ”isäsolmu” ja operaattori * merkitään puuhun solmun ”−” toiselle paikalle, kun toisella paikalla on a. Lapsisolmut b ja c merkitään solmun ”*” paikoille. Kuvassa 10 näkyy tähän asti muodostettu puu.



Kuva 10. Keskeneräinen abstrakti syntaksipuu.

Jatkan puun muodostamista solmun ”/” oikealle puolelle/paikalle. Jäljellä on operaattori ∗, joka on ”isäsolmu” d:lle ja e:lle. Kuvassa 11 puu on valmis.



Kuva 11. Abstrakti syntaksipuu.

4.1 Puolalainen esijärjestys

Kirjoitan laskutoimituksen $(a-(b*c))/(d*e)$ puolalaisessa esijärjestyksessä äsken muodostetun puun avulla. Ensimmäiseksi merkitään isäsolmu ”/” ja tästä jatketaan vasemmalle niin, että ”vanhempisolmu” merkitään ensin ja ns. tasa-arvoisista solmuista vasen merkitään ensin. Kun solmun ”/” vasen puoli/paikka on kirjoitettu, kirjoitetaan perään oikean puolimmaiset solmut samalla tavalla. Saan seuraavan näköisen järjestyksen aikaiseksi.

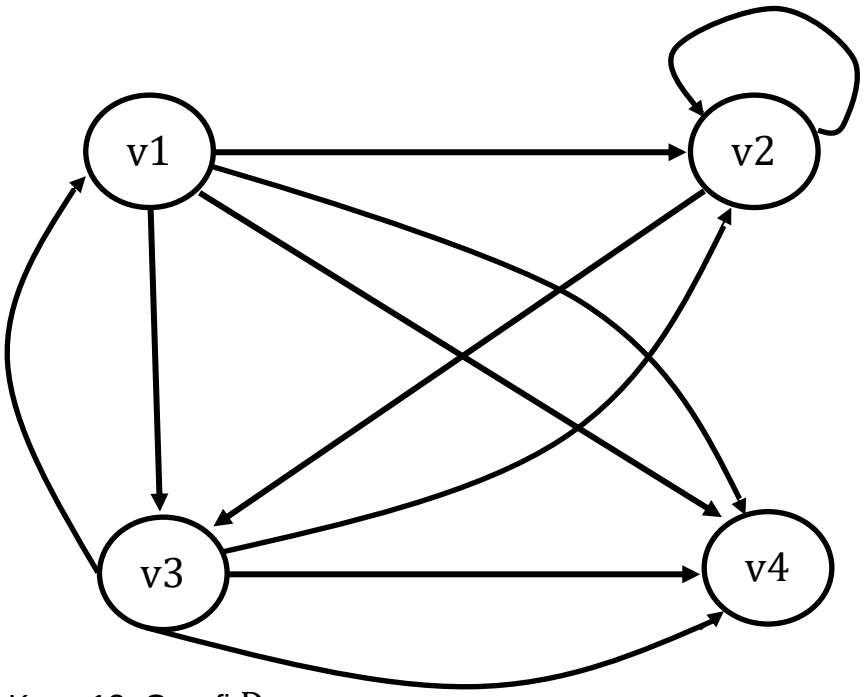
$/-\ a * b\ c * d\ e$

5 3 NUOLEN MITTAISET SUUNNATUT KÄVELYT

Suunnattu graafi $D = (V, A)$, missä $V = (v1, v2, v3, v4)$ on matriisina seuraava

$$M_D = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Lasken kuinka monta kolmen nuolen mittaista suunnattua kävelyä on olemassa solmusta $v1$ solmuun $v2$. Matkan solmusta $v1$ solmuun $v2$ ilmaisee matriisin alkio $m_{1,2}$. Eli solmusta $v1$ solmuun $v2$ on matkaa yhden nuolen verran yllä olevan matriisin mukaan. Havainnollistan suunnatun graafin kuvassa 12.



Kuva 12. Graafi D.

Kerron graafista annetun matriisin M_D kolmesti itsellään, jotta saadaan selville kolmen nuolen mittaisten suunnattujen kävelyiden määrät.

$$M_D^2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{array}{llll} 0*0+1*0+1*1+1*1 & 0*1+1*1+1*1+1*0 & 0*1+1*1+1*0+1*0 & 0*1+1*0+1*2+1*0 \\ 0*0+1*0+1*1+0*1 & 0*1+1*1+1*1+0*0 & 0*1+1*1+1*0+0*0 & 0*1+1*0+1*2+0*0 \\ 1*0+1*0+0*1+2*1 & 1*1+1*1+0*1+2*0 & 1*1+1*1+0*0+2*0 & 1*1+1*0+0*2+2*0 \\ 1*0+0*0+0*1+0*1 & 1*1+0*1+0*1+0*0 & 1*1+0*1+0*0+0*0 & 1*1+0*0+0*2+0*0 \end{array}$$

$$= \begin{bmatrix} 2 & 2 & 1 & 2 \\ 0 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$$M_D^2 * M_D = \begin{bmatrix} 2 & 2 & 1 & 2 \\ 0 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{array}{llll} 2*0+2*0+1*1+2*1 & 2*1+2*1+1*1+2*0 & 2*1+2*1+1*0+2*0 & 2*1+2*0+1*2+2*0 \\ 1*1+2*1+1*1+2*0 & 1*1+2*1+1*1+2*0 & 1*1+2*1+1*0+2*0 & 1*1+2*0+1*2+2*0 \\ 2*0+2*0+2*1+1*1 & 2*1+2*1+2*1+1*0 & 2*1+2*1+2*0+1*0 & 2*1+2*0+2*2+1*0 \\ 0*0+1*0+1*1+1*1 & 0*1+1*1+1*1+1*0 & 0*1+1*1+1*0+1*0 & 0*1+1*0+1*2+1*0 \end{array}$$

$$= \begin{bmatrix} 3 & 5 & 4 & 4 \\ 3 & 4 & 3 & 3 \\ 3 & 6 & 4 & 6 \\ 2 & 2 & 1 & 2 \end{bmatrix} = \mathrm{M}_{\mathrm{D}}^{3}$$

Kolmen nuolen mittaisia matkoja solmusta $\mathrm{v1}$ solmuun $\mathrm{v2}$ ilmaisee matriisin $\mathrm{M}_{\mathrm{D}}^{3}$ alkio $m_{1,2}$, eli 5. Kolmen nuolen mittaisia suunnattuja kävelyitä on siis 5.

5.1 3 nuolen mittaiset suunnatut kävelyt Pythonilla

```
In [1]: import numpy as np
        #matriisi
        Md = np.array([[0,1,1,1], [0,1,1,0], [1,1,0,2], [1,0,0,0]])
        print(Md)

[[0 1 1 1]
 [0 1 1 0]
 [1 1 0 2]
 [1 0 0 0]]

In [2]: Md3 = np.linalg.matrix_power(Md,3)
        #matriisin md tulo itsensä kanssa md * md * md
        print(Md3)

[[3 5 4 4]
 [3 4 3 3]
 [3 6 4 6]
 [2 2 1 2]]
```

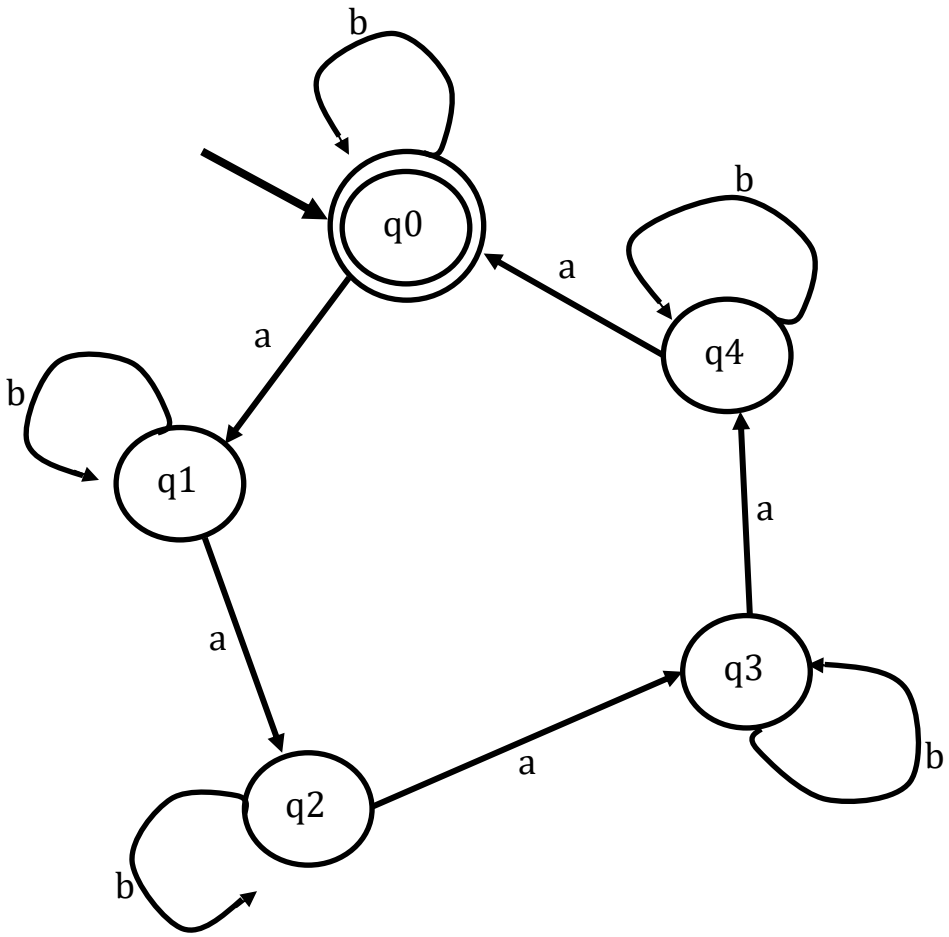
Kuva 13. Matriisit Pythonilla.

```
[[3 5 4 4]
 [3 4 3 3]
 [3 6 4 6]
 [2 2 1 2]]
```

Kuva 14. Kolmen nuolen mittaiset kävelyt $m_{1,2}$.

6 ÄÄRELLINEN AUTOMAATTI A

Äärellinen automaatti A hyväksyy syötejonot missä on 5:llä jaollinen määrä symboleita a. A saa syötteenä symbolit a ja b. Kuvassa 15 on äärellisen automaatin A diagrammi suunnatusta ja merkitystä graafista.



Kuva 15. Suunnatun merkityn graafin diagrammi äärellisestä automaatista A.

Nuoli osoittaa alkutilan eli solmun q0. Kaksinkertainen ympyrä samassa solmussa ilmaisee automaatin hyväksyvän tilan. Kun automaatille syötetään sana, automaatti lukee sen ja mikäli automaatti on viimeisen syötesymbolin lukemisen jälkeen hyväksyvässä tilassa, niin automaatti hyväksyy syötesanan. Kuten diagrammista näkyy, syötesanassa voi olla syötesymbolina a vähintään 5 kertaa tai ei ollenkaan ja syötesymboli b voi esiintyä missä välissä tahansa tai pelkästään.

Heli Hyttinen

Diskreetti matematiikka
Joukko-oppi

2019

Kaakkois-Suomen ammattikorkeakoulu

SISÄLLYS

1 ARGUMENTTI PROPOSITIOLOGIIKAN SYMBOLEIN 3

1.1 Totuustaulu 3

2 JOUKOT A, B 4

2.1 $A \cap B$ ja $(A \cup B)^c$ 4

3 SUUNNATTU GRAAFI JA PARTITIO 4

4 RELAATIOTAULUT 6

4.1 Q ei ole funktio 7

4.2 Alkioiden määrä $\mathcal{P}(U \times V)$ 8

5 TÄYDELLINEN TULOJEN-SUMMAMUOTO 8

1 ARGUMENTTI PROPOSITIOLOGIIKAN SYMBOLEIN

Jos tekstinkäsittely onnistuu, niin tietokone toimii. Jos tekstinkäsittely ei onnistu, niin mainoskirjettä ei voi korjata. Mainoskirjeen voi korjata. Siis, tietokone toimii.

Tekstinkäsittely onnistuu	p
Tietokone toimii	q
Mainoskirjeen voi korjata	r

Jos p niin q, jos ei p niin ei r, kuitenkin r, siis q.

$$p \longrightarrow q, \neg p \longrightarrow \neg r, r \vdash Q$$

1.1 Totuustaulu

p	q	r	$p \longrightarrow q$	$\neg p$	$\neg r$	$\neg p \longrightarrow \neg r$
T	T	T	T	F	F	T
T	T	F	T	F	T	T
T	F	T	F	F	F	T
T	F	F	F	F	T	T
F	T	T	T	T	F	F
F	T	F	T	T	T	T
F	F	T	T	T	F	F
F	F	F	T	T	T	T

taulukko 1

Taulukosta 1 reunoilla korostettujen sarakkeiden r, $p \longrightarrow q$ ja $\neg p \longrightarrow \neg r$ tulosten (T tai F) tulee taulukon ainakin yhdellä rivillä olla kaikkien totta (true/T). Tässä totuustaulussa ensimmäisellä rivillä r, $p \longrightarrow q$ ja $\neg p \longrightarrow \neg r$ ovat kaikki totta (T), joten argumentti kohdassa 1.2 on validi.

2 JOUKOT A, B

Perusjoukko $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$A = \{x \in U \mid x \text{ on parillinen}\}$, otetaan perusjoukosta U kaikki parilliset luvut.
Joten $A = \{2, 4, 6, 8\}$

$B = \{x \in U \mid x \leq 7\}$, otetaan perusjoukosta U kaikki lukua 7 pienemmät luvut.
Joten $B = \{1, 2, 3, 4, 5, 6, 7\}$.

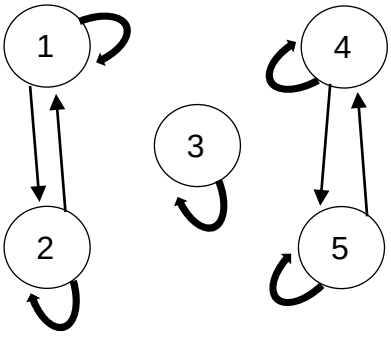
2.1 $A \cap B$ ja $(A \cup B)^c$

Luvut 2, 4 ja 6 esiintyvät molemmissa joukoissa A ja B.
Joten $A \cap B = \{2, 4, 6\}$.

Määrätäkseni $(A \cup B)^c$, selvitän aluksi molempien joukkojen alkiot,
$(A \cup B) = \{1, 2, 3, 4, 5, 6, 7, 8\}$.
Seuraavaksi tarkistan mitä jäisi perusjoukkoon U jäljelle, jos yllä olevat alkiot otettaisi sieltä pois. Jäljelle jäisi 9, joten $(A \cup B)^c = \{9\}$.

3 SUUNNATTU GRAAFI JA PARTITIO

$S = \{1, 2, 3, 4, 5\}$
Ekvivalenssirelaatio S:ssa on $R = \{(1,1), (1,2), (2,1), (2,2), (3,3), (4,4), (4,5), (5,4), (5,5)\}$



Kuva 1: suunnattu graafi relaatiosta R

Relaation R indusoima joukon S partitio määrätään kuvan 1 perusteella aloittamalla solmusta 1. Merkitään graafin solmu $\mathrm{R}[\mathrm{x}]$ ja nuolen osoittaman solmun luku otetaan ylös. Solmut ovat nimetty alkioiden mukaan.

$$
\begin{aligned}
\mathrm{R}[1] \quad &= \{1\} \\
&= \{2\} \\
\mathrm{R}[2] \quad &= \{1\} \\
&= \{2\} \\
\mathrm{R}[3] \quad &= \{3\} \\
\mathrm{R}[4] \quad &= \{4\} \\
&= \{5\} \\
\mathrm{R}[5] \quad &= \{4\} \\
&= \{5\}
\end{aligned}
$$

$\mathrm{R}[1]$ on sama kuin $\mathrm{R}[2]$, joten jätetään $\mathrm{R}[2]$ pois. Nyt partitiossa on $\{1,2\}$ $[1]$ mukaisesti.

Alkio 3 ei esiinny muissa kuin $\mathrm{R}[3]$, joten $\{3\}$ myös osaksi partitiota.

$\mathrm{R}[4]$ on sama kuin $\mathrm{R}[5]$, joten jätetään $[5]$ pois, ja saadaan $\{4,5\}$ osaksi partitiota.

S partitio R indusoimana on $[\{1,2\}, \{3\}, \{4,5\}]$.

4 RELAATIOTAULUT

$U = \{1,2,3,4\}$

$V = \{50,60,70\}$

Tulojoukko $U \times V$ relaatio $R = \{(1,50), (2,60),(3,50),(4,60)\}$

Tulojoukko $V \times U$ relaatio $Q = \{(70,4),(50,2),(50,3)\}$

Relaatiotaulu R

U	V
1	50
2	60
3	50
4	60

Taulun vasempaan sarakkeeseen U on sijoitettu relaation R vasemman puoleiset koordinaatit ja sarakkeeseen V on vastaavasti sijoitettu relaation R oikean puoleiset koordinaatit.

Relaatiotaulu Q

V	U
70	4
50	2
50	3

Taulun vasempaan sarakkeeseen V on sijoitettu relaation Q vasemman puoleiset koordinaatit ja sarakkeeseen U on vastaavasti sijoitettu relaation Q oikean puoleiset koordinaatit. Järjestys on tauluissa ylhäältä alas.

Relaatiotaulu $Q \circ R$

U	V
2	60
2	50
3	50
4	60
4	70

Kompositio $Q \circ R$ tulee siitä, kun relaatiot $R \subseteq U \times V$ ja $Q \subseteq V \times U$ yhdistetään. Ajattelin relaatiotaulujen joukon U henkilöinä ja joukon V eri automerkkeinä.

Relaatiotaulussa R henkilöllä 1 on automerkki 50, mutta relaatiotaulussa Q ei ole henkilöä 1. Jätän sen pois komposition relaatiotaulusta.
Relaatiotaulussa R henkilöllä 2 on automerkki 60 ja relaatiotaulussa Q tämän automerkki on 50, komposition relaatiotauluun laitan molemmat autot henkilölle 2.
Relaatiotaulussa R henkilöllä 3 on automerkki 50 samoin kuin relaatiotaulussa Q. Laitan tämän relaatiotauluun, sillä se esiintyy molemmissa.
Relaatiotaulussa R henkilöllä 4 on automerkki 60 ja relaatiotaulussa Q tämän automerkki on 70, komposition relaatiotauluun laitan molemmat autot henkilölle 4.

4.1 Q ei ole funktio

Q ei ole funktio, koska joukosta U alkio 60 ei määrää mitään ja alkio 50 määrää kahta. Relaatiossa Q pitäisi jokaista $u \in U$ kohti olla olemassa $v \in V$, jotta Q voisi olla funktio.

4.2 Alkioiden määrä $\mathcal{P}(\mathrm{U} \times \mathrm{V})$

Lasken joukon $\mathrm{U} \times \mathrm{V}$ potenssijoukon alkioiden määrän kertomalla 2 itsellään alkioiden määrän verran. Joukolla U on neljä alkiota ja joukolla V on kolme alkiota, yhteensä 7.

$2^7 =$ 128 alkiota.

$\mathcal{P}(\mathrm{U} \times \mathrm{V}) = 128$

Selvitin vielä erikseen joukkojen V ja U potenssijoukkojen alkioiden määrän.

$2^4 = 16$ alkiota. $\mathcal{P}(\mathrm{U}) = 16$

$2^3 = 8$ alkiota. $\mathcal{P}(\mathrm{V}) = 8$

$\mathrm{U} = \{\mathrm{U}, \{1\},\{2\},\{3\},\{4\}, \{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},$

$\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\},\{\emptyset\}\}$

$\mathrm{V} = \{\mathrm{V}, \{50\},\{60\},\{70\},\{50,60\},\{50,70\},\{60,70\},\{\emptyset\}\}$

Ja joukkojen potenssijoukkojen alkioiden määrät kerrottuna

$8 * 16 = 128$

5 TÄYDELLINEN TULOJEN-SUMMAMUOTO

$\mathrm{E}(\mathrm{x},\mathrm{y},\mathrm{z}) = (\mathrm{x}+\mathrm{y}'\mathrm{z})(\mathrm{y}+\mathrm{z}')$

$(\mathrm{x}+\mathrm{y}'\mathrm{z})(\mathrm{y}+\mathrm{z}')$

$=(\mathrm{xy}'+\mathrm{z})+(\mathrm{yz}')$, DeMorganin sääntö vaihtaa operaattorit päinvastoin

$= \mathrm{xy}(\mathrm{z}+\mathrm{z}') + \mathrm{x}(\mathrm{y}+\mathrm{y}')\mathrm{z}'$, perusteena Distributiivisuus eli osittelulaki, jossa yhteiset yhteenlaskettavat määritellään.

$= \mathrm{xy} + \mathrm{xz}'$, soveltaen kompelementtilakia a + a' = 0 ja absorptiota

Täydellisen tulojen-summamuodon päättelin ihan vain päättelemällä muuttujan x lisäksi nuo kaikki mahdolliset muuttujat.

$= xyz + xyz' + xyz + xy'z'$

$= xyz + xyz' + xy'z'$, sillä xyz esiintyy jo kerran.

Heli Hyttinen

Diskreetti matematiikka
Lineaarialgebra

2020

Kaakkois-Suomen
ammattikorkeakoulu

SISÄLLYS

1 LINEAARISESTI RIIPPUMATON JOUKKO S 3

2 S:N VEKTOREIDEN LINEAARIKOMBINAATIOT 3

3 AB 4

3.1 Ab Pythonilla 5

3.2 bb^T 5

3.3 bb^T Pythonilla 6

4 KOLMION SKAALAUS 7

4.1 Kolmion skaalaus Pythonilla 8

5 GAUSSIN ELIMOINTIALGORITMI 9

5.1 Gaussin elimointialgoritmi Pythonilla 11

6 ORTOGONAALIPROJEKTIO PIENIMMÄN NELIÖSUMMAN MENETELMÄLLÄ 13

1 LINEAARISESTI RIIPPUMATON JOUKKO S

Joukko S = {a, b}, missä a = (4, 1, 0) ja b = (0, 6, 1).
s(4, 1, 0) + k(0, 6, 1) = 0 pätee vain jos skalaarit s ja k ovat nollia (0), joten joukko S on lineaarisesti riippumaton. Jos toinen kertoimista olisi muu kuin nolla, jotta s(4, 1, 0) + k(0, 6, 1) = 0 pätisi, joukko olisi lineaarisesti riippuva.

s(4, 1, 0) + k(0, 6, 1) = (x, y, z), vaikka kertoimet olisivat mitä hyvänsä niin z = k ja jos k $\neq$ 0, niin z $\neq$ 0. Sekä s = $\frac{x}{4}$, joten jos s $\neq$ 0, niin x $\neq$ 0.

Esimerkiksi jos kertoimet olisivat s = 1 ja k = 0, niin (x, y, z) = (4, 1, 0), ei (0, 0, 0).
1(4, 1, 0) + 0(0, 6, 1)
= (4 + 0, 1 + 0, 0 + 0)
= (4, 1, 0)
Joten joukko S ei ole lineaarisesti riippuva.

2 S:N VEKTOREIDEN LINEAARIKOMBINAATIOT

Vektori c = (8, 20, 3) on joukon S lineaarikombinaatio. Skalaarit voitaisi selvittää seuraavasti.

s(4, 1, 0) + k(0, 6, 1) = (8, 20, 3)
s = $\frac{8}{4}$, eli 8 jaetaan 4, todeltaan s = 2. Ja k = $\frac{20 - s}{6}$, eli 20 vähennetään 2 ja erotus 18 jaetaan kuudella, todetaan k = 3.

Lasketaan:
2(4, 1, 0) + 3(0, 6, 1)
= (8 + 0, 2 + 18, 0 + 3)
= (8, 20, 3)

Lisäksi vektori d = (20, 17, 2) on joukon S lineaarikombinaatio. Skalaarit voitaisi selvittää seuraavasti.

s(4, 1, 0) + k(0, 6, 1) = (20, 17, 2)

s = $\frac{20}{4}$, eli 20 jaetaan 4, todetaan s = 5. Ja k = $\frac{17 - s}{6}$, eli 17 vähennetään 5 ja erotus 12 jaetaan kuudella, todetaan k = 2.

Lasketaan:

5(4, 1, 0) + 2(0, 6, 1)

= (20 + 0, 5 + 12, 0 + 2)

= (20, 17, 2)

3 AB

$\boldsymbol{A} = \begin{bmatrix} 4 & 0 & -3 \\ -1 & -2 & 3 \end{bmatrix}$ ja $\boldsymbol{b} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$

Matriisi $\mathbf{A}$ on kooltaan 2x3 ja matriisi $\mathbf{b}$ on 3x1, kertomalla $\mathbf{A}$:n rivit eli (4, 0, 3) ja (-1, -2, 3) $\mathbf{b}$:n sarakkeella eli (2, -1, 3) ja laskemalla tulot yhteen riveillä, saadaan 2x1 matriisi eli Ab.

$$\mathbf{Ab} = \begin{bmatrix} 4 \cdot 2 + 0 \cdot (-1) + (-3) \cdot 3 \\ -1 \cdot 2 + (-2) \cdot (-1) + 3 \cdot 3 \end{bmatrix}$$

$$= \begin{bmatrix} 8 + 0 - 9 \\ -2 + 2 + 9 \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ 9 \end{bmatrix}$$

Eli $\mathbf{Ab} = \begin{bmatrix} -1 \\ 9 \end{bmatrix}$

3.1 Ab Pythonilla

```
In [1]: import numpy as np
A = (4,0,-3), (-1,-2,3)
b = [2], [-1], [3]
A = np.array(A)
b = np.array(b)

#tulostetaan matriisit A ja b
print("A:\n",A)
print ("b:\n",b)
```

```
A:
 [[ 4  0 -3]
 [-1 -2  3]]
b:
 [[ 2]
 [-1]
 [ 3]]
```

```
In [2]: np.dot(A,b)
#kertoo A:n rivit b:n sarakkeella ja laskee tulot yhteen
```

```
Out[2]: array([[-1],
               [ 9]])
```

Kuva 1. Kuvakaappaus.

3.2 $\mathbf{bb^T}$

Kun $\boldsymbol{b} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$, saadaan $\mathbf{b}$:n transpoosi $\mathbf{b^T}$ muuttamalla $\mathbf{b}$:n rivit sarakkeiksi. Eli $\mathbf{b}^{\mathrm{T}} = [2 \quad -1 \quad 3]$. Nyt $\mathbf{b^T}$ on kooltaan 1x3 matriisi ja $\mathbf{b}$ on 3x1 matriisi, joten $\mathbf{bb^T}$ pitäisi saada neliömatriisi 3x3. Kerrotaan $\mathbf{b}$:n rivit $\mathbf{b^T}$:n sarakkeilla seuraavasti.

$$\mathbf{bb^T} = \begin{bmatrix} 2 \cdot 2 & 2 \cdot (-1) & 2 \cdot 3 \\ -1 \cdot 2 & -1 \cdot (-1) & -1 \cdot 3 \\ 3 \cdot 2 & 3 \cdot (-1) & 3 \cdot 3 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & -2 & 6 \\ -2 & 1 & -3 \\ 6 & -3 & 9 \end{bmatrix}$$

Eli $\mathbf{bb^T} = \begin{bmatrix} 4 & -2 & 6 \\ -2 & 1 & -3 \\ 6 & -3 & 9 \end{bmatrix}$

3.3 $\mathbf{bb^T}$ Pythonilla

```
In [1]: import numpy as np
        b = [2], [-1], [3]
        print(b)
        #tässä huomasin b:n olevan sama kuin 3-tupla
        #esittämällä matriisin b näin, transpoosi onnistuu alla

([2], [-1], [3])
```

```
In [2]: bt = np.transpose(b)
        print(bt)

[[ 2 -1  3]]
```

```
In [3]: np.dot(b, bt)
        #kertoo b:n rivit bt:n sarakkeilla, saadaan 3x3 matriisi

Out[3]: array([[ 4, -2,  6],
               [-2,  1, -3],
               [ 6, -3,  9]])
```

Kuva 2. Kuvakaappaus.

4 KOLMION SKAALAUS

Kolmion kärkipisteet ovat P_1 = (1, 1, 1), P_2 = (5, 0, 2) ja P_3 = (0, 4, 4).
Skalaari on s = 4,2.
Painopiste saadaan laskemalla kärkipisteiden vektoreiden alkiot yhteen järjestyksessä ja jakamalla summa kärkipisteiden määrällä. Painopisteen **c** koordinaatit lasketaan seuraavasti.

$\mathbf{c} = \frac{1}{3}(1 + 5 + 0), \frac{1}{3}(1 + 0 + 4), \frac{1}{3}(1 + 2 + 4)$
= (2 ; 1,67; 2,33)

Eli **c** = (2 ; 1,67; 2,33)

Seuraavaksi lasketaan kärkipisteen P_1 uusi sijainti skaalaamalla vektori skaalarilla 4,2 ja vähentämällä vektorin P_1 jokaisesta alkiosta puolikas painopisteen (eli vektorin **c**) ensimmäinen alkio.

P_1 = 4,2(1, 1, 1) – 0,5**c**
= (4,2 ; 4,2 ; 4,2) – (1 ; 1 ; 1)
= (3,2 ; 3,2 ; 3,2)

Eli uusi sijainti on P_1 = (3,2 ; 3,2 ; 3,2). Lasketaan seuraavaksi kärkipisteen P_2 uusi sijainti samalla tavalla, mutta vähennetään vektorin **c** toisen alkion puolikas vektorin P_2 alkioista.

P_2 = 4,2(5, 0, 2) – 0,5**c**
= (21 ; 0 ; 8,4) – (0,83 ; 0,83; 0,83)
= (20,17 ; -0,83 ; 7,57)

Eli uusi sijainti on P_2 = (20,17 ; -0,83 ; 7,57). Lasketaan seuraavaksi kärkipisteen P_3 uusi sijainti samalla tavalla, mutta vähennetään vektorin **c** kolmannen alkion puolikas vektorin P_3 alkioista.

P_3 = 4,2(0, 4, 4) – 0,5**c**
= (0 ; 16,8 ; 16,8) – (1,17 ; 1,17 ; 1,17)
= (-1,17 ; 15,63 ; 15,63)

Eli uusi sijainti P_3 = (-1,17 ; 15,63 ; 15,63).

Kolmion uudet kärkipisteet skaalattuna luvulla 4,2 ovat

P_1 = (3,2 ; 3,2 ; 3,2),

P_2 = (20,17 ; -0,83 ; 7,57),

P_3 = (-1,17 ; 15,63 ; 15,63).

4.1 Kolmion skaalaus Pythonilla

In [1]:
```
import numpy as np

#kärkipisteet
p1=(1,1,1)
p2=(5,0,2)
p3=(0,4,4)

p1=np.array(p1)
p2=np.array(p2)
p3=np.array(p3)
print("Kärkipisteet:\np1: ",p1,"\np2: ",p2,"\np3: ",p3)

#painopisteen koordinaatit erikseen
c1=p1[:1]+p2[:1]+p3[:1]
c2=p1[1:2]+p2[1:2]+p3[1:2]
c3=p1[2:3]+p2[2:3]+p3[2:3]
c1=c1/3
c2=c2/3
c3=c3/3
print("\nPainopisteen c koordinaatit:\n",c1, c2, c3)
```

```
Kärkipisteet:
p1:  [1 1 1]
p2:  [5 0 2]
p3:  [0 4 4]

Painopisteen c koordinaatit:
 [2.] [1.66666667] [2.33333333]
```

In [2]:
```
#puolet painopisteen koordinaateista
c1=c1/2
c2=c2/2
c3=c3/2

#skaalaus
p1=4.2*p1-c1
p2=4.2*p2-c2
p3=4.2*p3-c3
print(p1, p2, p3)
```

```
[3.2 3.2 3.2] [20.16666667 -0.83333333  7.56666667] [-1.16666667 15.63333333 1
5.63333333]
```

```
In [3]: p1=np.round_(p1, decimals = 2)
        p2=np.round_(p2, decimals = 2)
        p3=np.round_(p3, decimals = 2)
        print("Kärkipisteiden uudet sijainnit:\np1:",p1,"\np2: ",p2,"\np3: ",p3)

Kärkipisteiden uudet sijainnit:
p1: [3.2 3.2 3.2]
p2:  [20.17 -0.83  7.57]
p3:  [-1.17 15.63 15.63]
```

Kuva 4. Kuvakaappaus.

5 GAUSSIN ELIMOINTIALGORITMI

Ratkaisen seuraavan yhtälöryhmän matriisien ja Gaussin elimointialgortimin avulla.

$$\begin{cases} 2x_1 - 5x_2 = 1 \\ 3x_1 + 2x_2 = 11 \end{cases}$$

Ensimmäiseksi sijoitetaan luvut matriisiin ja nimetään ensimmäinen rivi R1 ja toinen rivi R2. Tarkoituksena on eliminoida ensimmäisen rivin toinen luku ja toisen rivin ensimmäinen luku. Sekä saada vastakkaiset luvut luvuksi 1. Rivien viimeiset luvut tulevat lopulta olemaan muuttujat x_1 ja x_2, eli $\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & x_2 \end{bmatrix}$.

$$\begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 2 & -5 & 1 \\ 3 & 2 & 11 \end{bmatrix}$$

Ensimmäisellä kierroksella kerrotaan toisen rivin ensimmäisellä luvulla ensimmäinen rivi ja toisen rivin kertojana käytetään ensimmäisen rivin ensimmäistä lukua negatiivisena.

$$\xrightarrow[\text{-2 R2} \longrightarrow \text{R2}]{\text{3 R1} \longrightarrow \text{R1}} \quad \begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 6 & -15 & 3 \\ -6 & -4 & -22 \end{bmatrix}$$

Koska käytettiin negatiivista kertojaa toisella rivillä, saadaan toisella kierroksella toisen rivin ensimmäinen luku eliminoitua lisäämällä toiseen riviin ensimmäinen rivi.

$$\xrightarrow{\text{R1 + R2} \longrightarrow \text{R2}} \quad \begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 6 & -15 & 3 \\ 0 & -19 & -19 \end{bmatrix}$$

Kolmannella kierroksella jaetaan toinen rivi luvulla -19 ja saadaan toisen rivin toiseksi ja kolmanneksi luvuksi 1.

$\frac{1}{-19}$ R2 ⟶ R2 ⟶ R1 $\begin{bmatrix} 6 & -15 & 3 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Neljännellä kierroksella lisätään ensimmäiseen riviin toinen rivi kerrottuna luvulla 15, jotta saadaan ensimmäisen rivin toinen luku eliminoitua.

R1 + 15R2 ⟶ R1 ⟶ R1 $\begin{bmatrix} 6 & 0 & 18 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Viimeisellä kierroksella jaetaan ensimmäinen rivi luvulla 6, jotta saadaan ensimmäisen rivin ensimmäinen luku luvuksi 1.

$\frac{1}{6}$ R1 ⟶ R1 ⟶ R1 $\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Nyt matriisin molempien rivien viimeiset luvut näyttävät muuttujat x_1 ja x_2.

Eli x_1= 3 ja x_2= 1. Kokeillaan laskea.

2(3) - 5(1)
= 6 – 5
= 1

3(3) + 2(1)
= 9 + 2
= 11

Vaikuttaa toimivan, eli:

$\begin{cases} 2x_1 - 5x_2 = 1 \\ 3x_1 + 2x_2 = 11 \end{cases}$ = $\begin{cases} 2(3) - 5(1) = 1 \\ 3(3) + 2(1) = 11. \end{cases}$

5.1 Gaussin elimointialgoritmi Pythonilla

Kuvissa 5 ja 6 on tehty sama Pythonilla numpy-kirjaston avulla. Pythonilla sijoitin rivit omiin matriiseihinsa ja kuvassa 6 ilmoitetaan muuttujien x_1 ja x_2 arvot.

In [2]:
```
import numpy as np
R1 = np.array([[2,-5,1]])
R2 = np.array([[3,2,11]])
print('R1:\n',R1,'\nR2:\n',R2)

#---ensimmäinen kierros
#3R1 on uusi R1
R1 = 3*R1
#-2R2 on uusi R2
R2 = -2*R2
print('\nensimmäinen kierros\nR1:\n',R1,'\nR2:\n',R2)

#---toinen kierros
#R1 + R2 on uusi R2
R2 = R1[0:] + R2
print('\ntoinen kierros\nR1:\n',R1,'\nR2:\n',R2)

#---kolmas kierros
#-19 / R2 on uusi R2
R2[...,1:] = R2[...,1:] /-19
print('\nkolmas kierros\nR1:\n',R1,'\nR2:\n',R2)

#---neljäs kierros
#R1 + 15R2 on uusi R1
R1 = R1 + 15*R2[0:]
print('\nneljäs kierros\nR1:\n',R1,'\nR2:\n',R2)

#---viimeinen kierros
#6 / R1 on uusi R1
R1 = R1/6
print('\nviimeinen kierros\nR1:\n',R1,'\nR2:\n',R2)
```

```
R1:
 [[ 2 -5  1]]
R2:
 [[ 3  2 11]]

ensimmäinen kierros
R1:
 [[  6 -15   3]]
R2:
 [[ -6  -4 -22]]
```

```
toinen kierros
R1:
 [[  6 -15   3]]
R2:
 [[  0 -19 -19]]

kolmas kierros
R1:
 [[  6 -15   3]]
R2:
 [[0 1 1]]

neljäs kierros
R1:
 [[ 6  0 18]]
R2:
 [[0 1 1]]

viimeinen kierros
R1:
 [[1. 0. 3.]]
R2:
 [[0 1 1]]
```

```
In [3]: x1 = R1[...,2:]
        x2 = R2[...,2:]
        print('muuttuja x1: ',x1,'\nmuuttuja x2: ',x2)
```

```
muuttuja x1:  [[3.]]
muuttuja x2:  [[1]]
```

Kuva 6. Kuvakaappaus.

6 ORTOGONAALIPROJEKTIO PIENIMMÄN NELIÖSUMMAN MENETELMÄLLÄ

$\mathbf{R}^3$:n vektorit $\boldsymbol{a_1}$ = (4,6,0) ja $\boldsymbol{a_2}$ = (0,6,3) määräävät kaksiulotteisen aliavaruuden (tason), johon määrään vektorin **b** = (8,4,3) ortogonaaliprojektion käyttämällä pienimmän neliösumman menetelmää Pythonin numpy-kirjaston avulla.

Aluksi sijoitetaan $\boldsymbol{a_1}$ ja $\boldsymbol{a_2}$ vektorit matriisiin **A**, josta saadaan 3x2 matriisi eli $m \, x \, n$ matriisi, jossa m on suurempi kuin n, eli rivejä on enemmän kuin sarakkeita.

```
In [1]: import numpy as np
        #sijoitetaan vektorit a1 ja a2 matriisiin A
        A = np.array([[4,0],[6,6],[0,3]])
        print('Matriisi A on 3x2 matriisi\n',A)
        #lineaarisesti riippumattomien sarakkeiden maksimimäärä
        print('Rangi on ',np.linalg.matrix_rank(A))

Matriisi A on 3x2 matriisi
 [[4 0]
 [6 6]
 [0 3]]
Rangi on  2
```

Kuva 7. Kuvakaappaus.

Kuvassa 7 selvitettiin myös, että **A**:n sarakkeet (n) ovat lineaarisesti riippumattomia, sillä $rank(\boldsymbol{A}) = n$. Eli rangi on sama kuin sarakkeiden määrä (2). Seuraavaksi sijoitetaan vektori **b** omaan matriisiinsa (kuva 8).

```
In [3]: #vektori b:n transpoosi, jotta saadaan 3x1 matriisi
        b = np.array([[8,4,3]]).T
        print(b)

[[8]
 [4]
 [3]]
```

Kuva 8. Kuvakaappaus.

b' = **A**x ja $\mathbf{A}^T\mathbf{A}x = \mathbf{A}^T\mathbf{b}$, jossa $x = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, koska **A**:n sarakkeet ovat lineaarisesti riippumattomat.

Seuraavaksi (kuva 9) luodaan **A**:n transpoosin ja **A**:n tulosta neliömatriisi $\mathbf{A}^T\mathbf{A}$. josta saadaan käänteismatriisi $(\mathbf{A}^T\mathbf{A})^{-1}$.

```
In [5]: #koska A on m x n matriisi, jossa m on suurempi kuin n ja rank = n, niin
        #matriisilla AtA on käänteismatriisi
        #AtA eli A:n transpoosin ja A:n tulo
        AtA = A.T @ A
        print('AtA:\n',AtA)
```

```
AtA:
 [[52 36]
 [36 45]]
```

```
In [6]: #AtA:n käänteismatriisi iAtA
        iAtA = np.linalg.inv(AtA)
        print('iAtA:\n',iAtA)
```

```
iAtA:
 [[ 0.04310345 -0.03448276]
 [-0.03448276  0.04980843]]
```

Kuva 9. Kuvakaappaus.

Kuvassa 10 lasketaan ensin **A**:n transpoosin ja **b**:n tulo, ja sen jälkeen niiden tulon ja $\mathbf{A}^T\mathbf{A}$:n käänteismatriisin tulo eli $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, joka on x. Jonka jälkeen voidaan laskea pienimmän neliösumman ratkaisu **b'**, eli **A**:n ja x:n tulo.

```
In [7]: x = iAtA @ ((A.T) @ b)
        #A:n ja x:n tulo eli pienimmän neliösumman ratkaisu
        b_ = A @ x
        print('pienimmän neliösumman ratkaisu on\n',b_)
```

```
pienimmän neliösumman ratkaisu on
 [[ 5.10344828]
 [ 5.93103448]
 [-0.86206897]]
```

Kuva 10. Kuvakaappaus

Ortogonaaliprojektio $\mathbf{b}'$ on $\mathbf{R}^3$:n vektori, sillä taso sijaitsee $\mathbf{R}^3$:ssa.

$$\mathbf{b}' = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b} \approx \begin{bmatrix} 5{,}1 \\ 5{,}93 \\ -0{,}86 \end{bmatrix}$$