5 Tilastolliset laskelmat

Tässä luvussa annetaan menettelytapoja erilaisten tilastollisten suureiden laskemiseen. Luvussa toteutetaan myös lottorivien generointi. Rivimäärää pyritään pienentämään oletetuilla ehdoilla. Luvun lopussa käsitellään myös bittioperaatiot.

5.1 Keskiarvo ja keskihajonta

Seuraavassa toteutetaan keskiarvon ja keskihajonnan laskeminen. Keskiarvon laskenta on pelkkänä funktiona, kun taas keskihajonnan laskemista varten on mukana kokonainen ohjelma.

5.1.1 Keskiarvo

Keskiarvo on yksinkertainen tilastollinen tunnusluku, joka kaikesta huolimatta on keskeisessä asemassa useimmissa tilastollisissa tarkasteluissa. Useimmiten se muodostaa perustason, johon muut tunnusluvut suhteutetaan.

Keskiarvo:

```
float keskiarvo(float arvo[], int maara)
{
    int lkm;
    float summa = 0.0;
    for (lkm = 0; lkm<=maara; ++lkm)
        sum = sum + arvo[lkm];
    return(summa/maara);
}
```

5.1.2 Keskihajonta (normaalijakautuneen aineiston)

Hyödynnämme edellä annettua keskiarvoalgoritmia keskihajonnan laskemisessa. Keskihajonnassa on lisäksi korotettava luvut toiseen potenssiin, jota varten esittelemme ensin makron nelio: nelio(x) = (x*x)

Keskihajonta:

```
#include <iostream.h>

#define MAARA 10    /*taulukossa olevien arvojen lukumäärä*/

float keskiarvo(float arvo[], int maara);
float anna_luku();
float juuri(float luku);
float nelio(float x);

void main()
{
    int paikka;    /*paikka taulukossa*/
    float arvo[MAARA];
    float k_arvo;
    float summa = 0.0;
    double keskihajonta;

    for (paikka = 0; paikka < MAARA; ++paikka)
        arvo[paikka] = anna_luku();

    /*lasketaan keskiarvo*/
    k_arvo = keskiarvo(arvo, MAARA);

    /*lasketaan summa neliöiden erotusten suhteen*/
    for (paikka = 0; paikka < MAARA; paikka++)
        summa = summa + nelio(arvo[paikka]-k_arvo);

    /*lasketaan keskihajonta*/
    float temp = juuri(summa);
    keskihajonta = temp / (MAARA-1);
    cout << "Arvojen keskihajonta on \n" <<  keskihajonta;
}

float keskiarvo(float arvo[], int maara)
{
    int lkm;
    float sum = 0.0;
/* ensin tallennetaan arvot taulukkoon*/
    for (lkm = 0; lkm<=maara; ++lkm)
        sum = sum + arvo[lkm];
/* palauta keskiarvo */
    return(sum/maara);
}

float anna_luku()
{
float a;
cout << "Anna luku ";
```

```
cin >> a;
return a;
}

float juuri(float luku)
{
    float x = luku/10;   /*ensimmäinen approksimaatio juuresta*/
    float dx;
    float tarkkuus = 0.001;
    float muutos;  /*approksimaation ja oikean tuloksen erotus*/
    if (luku < 0) luku = luku * (-1);
    if (luku == 0) return (0);   /*juurta ei oteta negatiivisesta
luvusta*/
    do
        {
        dx = (luku - x * x) / (2.0 * x);
        x = x + dx;
        cout << x << "\n";
        muutos = luku - x* x;
        }
    while ( (muutos < 0 ? (-1)*muutos : muutos) > tarkkuus);
/*tarpeeksi lähellä?*/
    return (x);
}

float nelio(float x)
{
return x * x;
}
```

5.2 Regressio ja korrelaatio

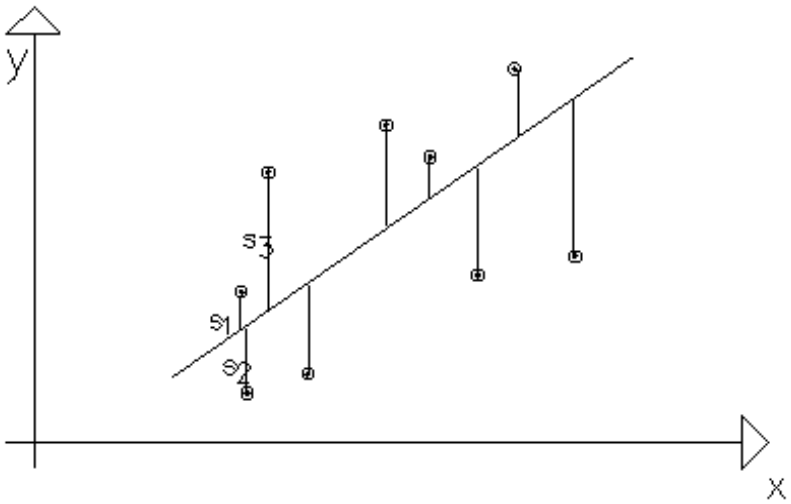
Kahden muuttuvan suureen välistä vuorovaikutusta voidaan kuvata regressiosuoran ja korrelaation avulla. Erityisesti regressiokäyrän sovittaminen pistejoukkoon saattaa olla manuaalisesti tehtynä työlästä.

5.2.1 Regressio

Kahden muuttujan välistä vuorovaikutussuhdetta kuvataan *regression* ja *korrelaation* avulla. Helpoin ja karkein tapa kuvata muuttujien riippuvuutta toisistaan on piirtää muuttujien arvoja kuvaavat pisteet xy-koordinaatistoon ja tarkastella syntynyttä hajontakuviota. Mikäli pisteiden avulla voidaan piirtää (edes karkea) suora, voidaan sanoa, että riippuvuus on lineaarista.

Edellistä tarkempi menettelytapa on *pienimmän neliösumman menetelmä*, jossa koordinaatistoon sijoitetuille pisteille sovitetaan käyrä. Hajontakuvioon piirrettävä suora määritetään siten, että havaintopisteiden ja suoran välisten poikkeamien neliöiden summa on mahdollisimman pieni. Käyrä voi olla muukin kuin suora, esimerkiksi paraabeli tms. Tarkastelemme tässä vain suorayhtälöä, $y = ax + b$. Tietokonetta kannattaa tietenkin hyödyntää myös muiden sovituskäyrien (kuten $y = ax^2 + bx + x$ tai $y = ae^{bx}$) kertoimien hakemiseen.

Kuvamme havainnollistaa hajontakuviota ja pienimmän neliösumman menetelmää sovitussuoran piirtämisessä.



Suoran kertoimet saadaan seuraavasti:

$$b = (n\sum x_i y_i - \sum x_i \sum y_i)/(n(\sum x_i^2 - (\sum x_i)^2)$$, jossa n = havaintoparien määrä

ja

$$a = y\_ - bx\_$$, jossa $y\_$ ja $x\_$ ovat muuttujien x ja y keskiarvot.

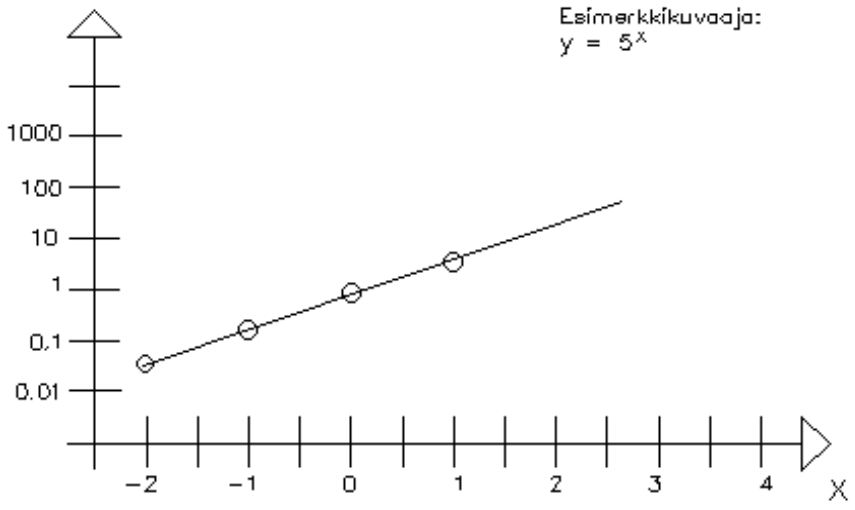
Algoritmi on siis seuraavanlainen:

Regressiosuoran kertoimet:

```
Sijoitetaan havaintoparit [n,2]-taulukkoon, jossa sarakkeeseen (i,1)
tulee x:ää vastaava arvo ja (i,2) y:tä vastaava arvo, i = 1...n (ota
huomioon taulukon alkaminen 0:sta).
Lasketaan silmukassa summa_x, summa_y, summa_xy ja summa_x2.
Lasketaan muut arvot ja keskiarvot summa_y/n ja summa_x/n
Lasketaan b ja a.
```

Koska meillä on jo x-arvoja taulukossa, voinemme hyödyntää niitä tulostamalla, aikaansaatua suoran yhtälöä käyttäen, taulukossa olevia x:n arvoja vastaavat y:n arvot, jolloin saadaan pisteitä suoran piirtämiseksi.

Aiemmin tässä luvussa kerrottiin, että suoran sijasta pistejoukkoon joudutaan sovittamaan muun tyyppistä käyrää, kuten eksponentiaalista käyrää $y = ae^{kx}$. Tällaisen käyrän laskeminen ja muodostaminen voi tulla liian hankalaksi ilman matemaattisia erikoissovellusohjelmia. On kuitenkin olemassa keino, jolla saadaan aikaan eksponenttikäyrä edellä kuvatulla suoran algoritmilla. Tällöin siirrytäänkin logaritmiasteikolle, jolloin y-akselille tulevat arvot lny ja x-akseli pysyy tavallisena tasavälisenä asteikkona. Tällaisella asteikolla eksponenttifunktion kuvaaja on suora. Jos haluamme siis sovittaa pistejoukkoon (x_i, y_i) käyrää $y = ae^{kx}$, sovitammekin sen sijaan käyrää $\ln y = \ln(ae^{kx})$, joka taas sievenee muotoon $\ln y = \ln a + kx$. Nyt voimme merkitä tekijää $\ln y = v$ ja $\ln a = b$, jolloin saamme suoran yhtälön $v = b + kx$.
Nyt voimme sovittaa kyseistä suoraa pistejoukkoon $(x_i, \ln y_i)$. Halutessamme voimme suoran kertoimien löytymisen ja suoran piirtämisen jälkeen muuntaa suoran eksponenttimuotoon. Alla oleva kuva havainnollistaa eksponenttikäyrän piirtämistä suoramuodossa lin-log-asteikolle.



5.2.2 Korrelaatio

Kahden muuttujan välistä riippuvuutta voidaan kuvata myös *korrelaatiolla*. Korrelaation suuruutta kuvataan *korrelaatiokertoimella*. Korrelaatiokertoimella on etumerkki ja suuruus (-1...+1). Korrelaatiokerroin on +1 silloin, kun muuttujien arvot riippuvat täysin toinen toisistaan. Kun muuttujilla ei ole mitään riippuvuutta, kertoimen arvo on nolla ja kun muuttujilla on käänteinen riippuvuus, kertoimen arvo on -1. Viimeksi mainitussa tapauksessa toisen muuttujan arvot pienenevät silloin, kun toisen arvot kasvavat. Pearsonin korrelaatiokerroin on hyvin yleisesti käytetty. Se saadaan kaavalla

$$r_{xy} = bs_x/s_y,$$

jossa b on regressiokerroin ja s_x on x-arvojen keskihajonta ja s_y taas y-arvojen keskihajonta. Olemme aiemmin tässä teoksessa käsitelleet sekä b-arvon laskemisen että keskihajonnan, joten tätä kaavaa lienee lukijan vaivatonta käyttää.

tai kaavalla

$$r_{xy} = (n\sum x_i y_i - \sum x_i \sum y_i)/ [(n(\sum x_i^2 - (\sum x_i)^2) * (n \sum y_i^2 - (\sum y_i)^2)]^{1/2}$$

Tämä kaava muistuttaa runsaasti aiemmin käsiteltyä sovitussuoran kerroinkaavaa, joten myös algoritmi voidaan johtaa aiemmasta algoritmista.

5.3 Kombinaatioiden lukumäärä

Kun perusjoukossa on n lukua ja niistä valitaan k alkiota siten, että samaa alkiota ei voida valita uudelleen, saadaan esille kombinaatiot. Kombinaatiot ovat k-alkioisia osajoukkoja joukosta, jossa on n alkiota. Esimerkki kombinaatiosta on lottorivi: perusjoukossa on 39 numeroa (n), joista valitaan 7-alkioisia (k) kombinaatioita.

kombinaatioiden lukumäärä saadaan kaavalla:

$$n!/k!(n-k)!$$

Käytämme algoritmissa hyödyksi aiemmin määrittelemäämme kertoma-algoritmia. Tässä yhteydessä voi mainita, että kertoma ilmoittaa permutaatioiden lukumäärän, eli sen, kuinka monella tavalla n-alkioinen joukko voidaan asettaa jonoon eli permutoida.

Kombinaatiot:

```
#include <iostream.h>
int kombi(int n, int k);
int kertoma(int kert);

void main()
{
int n, k;

cout << "Anna perusjoukon koko \n";
cin >>  n;

cout << "Anna osajoukon koko \n";
cin >>  k;

cout << "Kombinaatioita on " << kombi(n,k);

}

int kertoma(int kert)
{
int i, j;
j = 1;
for (i=1; i<= kert; i++)
j = j * i;
return j;
}


int kombi(int n, int k)
{
    return (kertoma(n) / (kertoma(k)*kertoma(n-k)));
}
```

Edellä oli puhetta lottopelistä, joten tarkastelemme hieman satunnaislukuja. Satunnaislukugeneraattorin kehittämiseen on kehitelty useita erilaisia algoritmeja, joissa käytetään hyväksi esimerkiksi aikaa ja bittien (bittijonojen) siirtämistä.

Satunnaislukuja voidaan kehittää esimerkiksi rand()-funktiolla, joka on ANSI-yhteensopiva:

```
#include <stdlib.h>
int rand(void);
```

Luvut ovat väliltä 0 - 2^{32} (32-bittinen työkalu)
Maksimisatunnaisluku on vakiossa RAND_MAX.

Esimerkiksi satunnaisluku väliltä 0-9 saadaan lauseella:

```
rand() % 10;
```

Jos esimerkiksi simuloidaan lottopeliä, on perusjoukon koko 39. Luku nolla voidaan pyyhkiä pois joko ehtolauseella, joka testaa satunnaisluvun tai käyttämällä lausetta rand() % 39 + 1.

Lottorivien generointi:

```
#include <iostream.h>
#include <stdlib.h>

void main()
{
int i;
for (i = 0; i < 7; i++)
cout << rand() % 39 + 1 << "\t";
}
```

Lottorivejä voidaan muodostaa useallakin eri tavalla. Seuraavassa on eräs tapa.

Lottorivien generointi:

```
#include <iostream.h>
#include <stdlib.h>

void main()
{

int i, j, ei=1;
int rivi[] = {0,0,0,0,0,0,0};

i = 0;
randomize();
while (i < 7)
{
    ei = 1;
    int nro = rand() % 39 + 1;
//  cout << i;
    for (j = 0; j <= i; j++)
    if (nro == rivi[j])
    {
        ei = 0;
        break;
```

```
        }

        if (ei == 1)
        {
        rivi[i] = nro;
        i++;
        }

    }

    for (i = 0; i < 7; i++)
    cout << rivi[i] << "\n";

    }
```

Joissakin ohjelmointikielissä on mukana joukko-opillisia funktioita, joiden avulla on helppo testata vaikkapa se 'kuuluuko nyt generoitu lottonumero aiemmin saatujen numeroiden joukkoon". Joukko-opilliset funktiot voi C++-kielessä kehittää vaikkapa luokan metodeiksi, jolloin operaattorit voidaan ylimääritellä vastaamaan joukko-opillisia operaattoreita (unioni, leikkaus jne.). C++-kielen STL-kirjaston avulla voidaan myöskin kehittää joukko-operaatiot.

Seuraavassa on esimerkki joukkojen muodostamisesta ja joukko-operaatioista STL-kirjaston avulla.

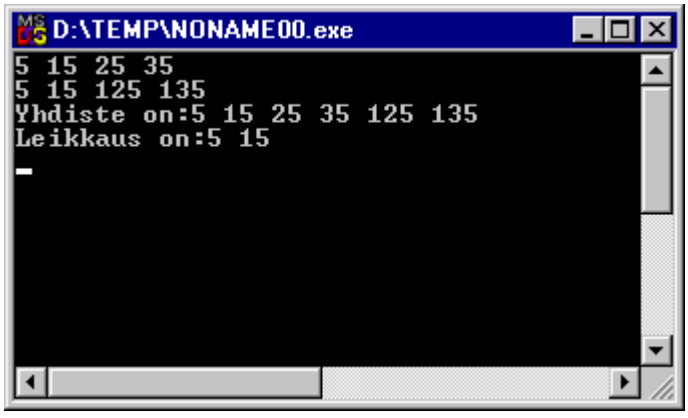
Joukot STL-kirjaston kautta:

Joukko – set

```
#include <algorithm>
 #include <vector>
 #include <iostream.h>
 #include <conio.h>
#include <set>


 using namespace std;
 typedef set<double,less<double> > set_type;
     ostream& operator<< (ostream& out, const set_type& s)
 {
   copy(s.begin(), s.end(),
        ostream_iterator<set_type::value_type>(cout," "));
   return out;

 }
 int main(void)
```

```
{


    set_type  joukko1, joukko2, joukko3, joukko4;
    joukko1.insert(5);    joukko1.insert(15);
        joukko1.insert(25);     joukko1.insert(35);
         cout << joukko1 << endl;
    joukko2.insert(5);    joukko2.insert(15);
        joukko2.insert(125);     joukko2.insert(135);
     cout << joukko2 << endl;

set_union(joukko1.begin(),joukko1.end(),joukko2.begin(),joukko2.end(
),
          inserter(joukko3,joukko3.begin()));

   cout << "Yhdiste on:" << joukko3 << endl;


   set_intersection(joukko1.begin(),joukko1.end(),

joukko2.begin(),joukko2.end(),inserter(joukko4,joukko4.begin()));

  cout << "Leikkaus on:" << joukko4 << endl;


      getch();

}
```

Tulos:



Mainittakoon tässä yhteydessä, että satunnaislukugeneraattorit eivät yleensä anna täydellisiä satunnaislukuja, vaan jokin luku voi esiintyä muita useammin. Erityisesti tämä tulee esille silloin, kun satunnaisluvut otetaan pienestä perusjoukosta (esimerkiksi väliltä 1-5). Tämä johtuu siitä, että satunnaislukuja generoidaan

algoritmilla, jota generaattori käyttää. Algoritmi ei välttämättä ole täydellinen, siis täysin satunnaisia lukuja antava. Kannattaakin yrittää kasvattaa perusjoukkoa silloin, kun se on mahdollista.

Lotossa on mahdollisia rivikombinaatioita yli 15 milj. kappaletta:

Kaava oli siis n!/k!(n-k)!

Kun n = 39 ja k = 7, saadaan lottorivien määräksi 39!/(7!(32!).

Oikean rivin sattumisen todennäköisyys on näin suunnilleen 1/15 milj. Jotta oltaisiin täysin varmoja täysosumasta, täytyisi kirjoittaa kaikkia rivi, mikä tulisi maksamaan kymmeniä miljoonia markoissa! Tämä on tietenkin järjetöntä ja viime aikoina onkin pyritty löytämään keinoja, joilla parannettaisiin huomattavasti täysosuman (tai edes pienemmän voiton) todennäköisyyttä. Ajatus ei ehkä olekaan aivan mahdoton. Tarkastelemalla eri numeroiden esiintymistaajuuksia, havaitaan, että jotkut numerot ovat esiintyneet jopa kaksi kertaa useammin kuin toiset. Ehkä lottopallot eivät esiinnykään täysin satunnaisesti, vaan arvonnassa vaikuttavat hieman myös jotkut palloihin liittyvät parametrit. Tällaisia tekijöitä voisivat olla vaihtelut pallojen painoissa, johtuen siitä, että pallot eivät ole aivan samankokoisia, eikä niissä käytetty materiaali ole homogeenista. Myös pallojen pyöreys vaihtelee, samoin kuin lottonumeroiden työstämiseen liittyvät tekijät kuten numeron pinta-ala tai tilavuus. Myös pallojen pudotusjärjestyksellä tai erilaisiin numeromuotoihin eri tavoin vaikuttavalla ilmanvastuksella voi olla vaikutuksensa!?
Tällaisia tekijöitä on mahdotonta saada esille ilman konkreettisia tutkimuksia. Lisäksi myös lottopallosarjoja vaihdetaan satunnaisin aikavälein. Tekijöitä voi kuitenkin yrittää haarukoida esille tutkimalla numeroiden esiintymistaajuuksia ja rivien sisältöä aiemmilta kierroksilta. Tällaisen tutkimuksen tuloksena voivat edellä mainitut (satunnaisuutta muuttavat) parametrit tulla jossain määrin näkyville. Tilastollisesti voidaan tutkia esimerkiksi, mihin luokkiin numerot sijoittuvat eri kierroksilla, mikä on rivien hajonta ja keskiarvo, mikä on vaihteluväli jne.

Kaikki mahdolliset lottorivit voidaan muodostaa esimerkiksi seuraavalla johdattavalla algoritmilla.

Kaikki lottorivit:

```
int a,b,c,d,e,f,g, b1,c1,d1,e1,f1,g1;
a = 0;
Lottorivejä muodostuisi silmukassa
j = 0;
for a = 1 to 33
b1 = a + 1
    for b = b1 to 34
    c1 = b + 1
```

```
            for c = c1 to 35
            d1 = c + 1
                for d = d1 to 36
                e1 = d + 1
                    for e = e1 to 37
                    f1 = e + 1
                        for f = f1 to 38
                        g1 = f + 1
                            for g = g1 to 39
j = j + 1;
```

Tallenna rivi taulukkoon Rivit[j,7], jossa siis alkiot ovat:

Rivit[j,1] = a
Rivit[j,1] = b
Rivit[j,1] = c
Rivit[j,1] = d
Rivit[j,1] = e
Rivit[j,1] = f
Rivit[j,1] = g

Simuloimme nyt algoritmia pienillä joukoilla: Olkoon perusjoukko (1..6), josta halutaan saada esille kaikki 3:n alkion kombinaatiot. Kombinaatioita pitäisi tulla edellä annetun kaavan mukaan: 6!/(3!(6-3)!) = 720/(6*6) = 20 kappaletta. Perusjoukko on siis luvut 1, 2, 3, 4, 5, 6.

Kombinaatioiden tulostus:

```
#include <iostream.h>

void main()
{
int a,b,c;
int kombi[6][3];

// Kombinaatioita
int i = 0;
int j = 0;
for (a = 1; a < 5; a++)
    for (b = a+1; b < 6; b++  )
        for (c = b+1; c < 7; c++)
        {
            cout << " a on " << a << "\t b on " << b << " \tc on "
<< c << "\n";
        }
}
```

Manuaalinen simulointi:

a = 1 ... 4
b = a+1 ... 5
c = b+1 ... 6
j = 0

1. kierroksella: a = 1, b = 2, c = 3
2. kierroksella: a = 1, b = 2, c = 4
3. kierroksella: a = 1, b = 2, c = 5
4. kierroksella: a = 1, b = 2, c = 6
5. kierroksella: a = 1, b = 3, c = 4
6. kierroksella: a = 1, b = 3, c = 5
7. kierroksella: a = 1, b = 3, c = 6
8. kierroksella: a = 1, b = 4, c = 5
9. kierroksella: a = 1, b = 4, c = 6
10. kierroksella: a = 1, b = 5, c = 6
11. kierroksella: a = 2, b = 3, c = 4
12. kierroksella: a = 2, b = 3, c = 5
13. kierroksella: a = 2, b = 3, c = 6
14. kierroksella: a = 2, b = 4, c = 5
15. kierroksella: a = 2, b = 4, c = 6
16. kierroksella: a = 2, b = 5, c = 6
17. kierroksella: a = 3, b = 4, c = 5
18. kierroksella: a = 3, b = 4, c = 6
19. kierroksella: a = 3, b = 5, c = 6
20. kierroksella: a = 4, b = 5, c = 6

Palatkaamme vielä lottoriveihin. Koska kaikista lottoriveistä koostuisi suunnaton (muistia vievä) taulukko, jota olisi hidasta käsitellä, kannattaa generoidut lottorivit analysoida tilastollisesti jo ennen niiden tallentamista taulukkoon. Kuten aiemmin mainittiin, rivien suodattamiseen voi käyttää erilaisia tilastollisia tunnuslukuja, joihin generoituja lottorivejä verrataan. Tällöin on laadittava funktiot, joilla tutkitaan esimerkiksi, mihin luokkiin numerot sijoittuvat, mikä on rivien hajonta ja keskiarvo, mikä on vaihteluväli jne. Kyseisten vertailussa tarvittavien tunnuslukujen arvoja ei voi saada muualta kuin tutkimalla aiempia lottorivejä ja eri numeroiden esiintymistaajuuksia. Lisäksi kannattaa lisätä algoritmiin muutamia laskureita, joilla nähdään poistettujen rivien määrät.
Tutkimalla aiempia lottorivejä voidaan esimerkiksi nähdä, ettei riveissä ole esiintynyt 5 peräkkäistä numeroa, tai rivien suurimman ja pienimmän numeron erotus on aina ollut yli tietyn arvon, tai että jotkut numerot ovat esiintyneet kaksi kertaa useammin kuin jotkut muut jne.

Mainittakoon tässä yhteydessä, että teoksen kirjoittaja ei edes harrasta lottoamista, ja suhtautuu jopa epäillen siihen, että lottoriveistä olisi löydettävissä tekijöitä, jotka sotkevat satunnaisuuden. Joka tapauksessa viime aikoina on tällaisesta ollut keskustelua eri lehtien palstoilla. 'Puhtaassa' tilastotieteessä ei tietenkään hyväksytä lottonumeroilla tapahtuvia operaatioita, vaan lottonumero esittää yksinkertaisesti jotain luokkaa, eli itse numeroarvoa ei voida tilastollisesti hyödyntää.

Seuraavana on ohjelma, joka generoi (ja tarvittaessa tulostaakin) kaikki mahdolliset lottorivit. Ohjelman käyttäjällä voi olla 'mystisiä' olettamuksia siitä, mitkä rivit eivät varmaankaan tule esiintymään oikeina lottoriveinä. Näin käyttäjä voi määrittää reunaehtoja, jolloin 15 miljoonan lottorivijoukko pienenee (ja sen tulee todellakin pienetä).

LOTTO-ohjelma:

```
#include <iostream.h>

/* Generoidaan kaikki lottorivit ja poistetaan rivejä seuraavasti:
perusjoukko eli numerojoukko, joita pidetään 'onnennumeroina'
hyväksytään veikattaville riveille. Kyseinen joukko saadaan
esimerkiksi ottamalla 20 aiemmilla kierroksilla eniten
esiintynyttä lottonumeroa. Tällöinkin veikattavia rivejä tulee
yli 70000, joten niistä on edelleen poistettava
joillakin tunnusluvuilla rivejä. Tässä poistetaan esimerkiksi
rivit, joissa on 4 tai enemmän peräkkäisiä numeroita sekä
rivit, joissa kaikki numerot ovat joko parillisia tai parittomia.
Lisäksi poistetaan rivit, joiden vaihteluväli < 13. Haluttaessa
voidaan vielä poistaa keskiarvojen ja keskiarvon hajonnan mukaan.
Kun poistoehtojen päällekkäisyydestä ollaan epävarmoja, on
käytettävä OR-funktiota.   */

/* Perusjoukko koostuu noin 450 kierroksen eniten esiintyneistä
lottonumeroista (20 eniten esiintynyttä).

Numerot ovat:

16, 25, 10, 9, 7, 14, 11, 36, 1, 3, 20, 17, 19, 15, 8, 32, 4, 5, 12,
35

Nämä numerot ovat siis esiintyneet useimmin kierroksilla
2/86 - 24/95.
Näistä otetaan nyt perusjoukkoon 18 numeroa.

Kun lottorivejä generoidaan, tarkistetaan heti, onko generoitu
numero perusjoukossa. Jos ei ole, aletaan generoida uutta riviä.

NÄILLÄ EHDOILLA SYNTYY muutama tuhat LOTTORIVIÄ:
```

```
1. PERUSJOUKKO ON 18 ENITEN ESIINTYNYTTÄ NUMEROA.
2. POIS RIVIT, JOIDEN KESKIARVO < 19 TAI > 23.
3. POIS RIVIT, JOIDEN VAIHTELUVÄLI < 20.
4. POIS RIVIT, JOISSA KAIKKI NUMEROT OVAT JOKO PARILLISIA TAI
   PARITTOMIA.
5. POIS RIVIT, JOISSA ON 3 PERÄKKÄISTÄ NUMEROA.
OHJELMA TALLENTAA RIVIT TAULUKKOON.
ILMAN POISTOJA RIVEJÄ TULISI YLI 20 000 KPL !!! KUITENKIN
PERUSJOUKKO ON VIELÄ VARSIN SUURI.   */

short t[7];
short o[4000][7];

int a,b,c,d,e,z,f,g;
int j;
long m;
int i,s,p,r,q,k, summa;
double hajo, nsumma, apu, k_arvo;
int ok;

// PERUSJOUKKO: Eniten toistaiseksi esiintyneet; lisää tarkistus
ohjelmaan!
// short perus[] =
// {16,25,10,9,7,14,11,36,1,3,20,17,19,15,8,32,4,5};
// Tämä joukko tulee päivittää

void vaihteluvali(short t[])
{
/* tarkistetaan rivin pituus eli maksimi-minimiarvo; samalla poistuu
rivi, jossa 7 peräkkäistä numeroa */

    short Max, Min;
        p = 0;
        Max = t[0];
        Min = t[0];
/* hae maksimi ja minimi*/
    for (i = 0; i < 7; i++)
    {
        if (t[i] > Max) Max = t[i];
        if (t[i] <= Min) Min = t[i];
    }

    /* aiemmilta kierroksilta laskettu minimivaihteluväli on
käytettävissä */
    if ( ( (Max - Min) < 19) || ((Max-Min) > 35)) p = 1;
}


void keskiarvo(short t[])
{
// Tarvitaan myös keskihajonnan laskemiseen
```

```
// Lisää halutessasi keskihajonnan laskenta
// Ehkä tällöin voit poistaa vaihteluvälin laskennan

      q = 0;
      summa = 0;
        for (i = 0; i < 7; i++)
    {
      summa = summa + t[i];
    }
      k_arvo = summa/7;
      if ( (k_arvo < 17.5) || (k_arvo > 25.5)) q = 1;
}



void parillinen(short t[])
{
/* Tarkistetaan, ovatko kaikki numerot joko parillisia tai
parittomia; haluttaessa voidaan poistaa rivit, joissa kaikki
numerot ovat joko parittomia tai parillisia. */

      s = 0;

      if ( (t[0] % 2 != 0) && (t[1] % 2 != 0) && (t[2] % 2 != 0)
  && (t[3] % 2 != 0) && (t[4] % 2 != 0) && (t[5] % 2 != 0) &&
 (t[6] % 2 != 0) ) s = 1;

      if ( (t[0] % 2 == 0) && (t[1] % 2 == 0) && (t[2]% 2 == 0) &&
 (t[3]% 2 == 0) && (t[4]% 2 == 0) && (t[5]% 2 == 0) && (t[6]%
  2 == 0) ) s = 1 ;
}


void lajittele(short t[])
{
/* laitetaan generoidut rivit suuruusjärjestykseen, jolloin
voidaan poistaa esimerkiksi rivit, joissa on 3 tai enemmän
peräkkäisiä numeroita */

int temp;
      r = 0;
      for (k = 0; k < 6; k++)
            for (i = k+1; i < 7; i++)
            {
                 if (t[k] < t[i])
                 {
                     temp = t[k];
                     t[k] = t[i];
                     t[i] = temp;
                 }
             }
```

```
      /* Poistetaan rivit, joissa on 3 tai enemmän peräkkäisiä
numeroita. */
      if ( (t[0] - t[2] == 2) || (t[1] - t[3] == 2) || (t[2] -
t[4] == 2) || (t[3] - t[5] == 2) || (t[4] - t[6] == 2) )
 r = 1;
}


void vertaa(short t[])
{
/*  if ((t[1] in perus) and (t[2] in perus) and (t[3] in perus)
      and (t[4] in perus) and (t[5] in perus) and (t[6] in perus)
      and (t[7] in perus)) then don't accept...
          Toteuta halutessasi!             */

          m = m+1; s = 0; p = 0; q = 0; r = 0;
          ok = 1;
          keskiarvo(t);
          parillinen(t);
          lajittele(t);
          vaihteluvali(t);
          if ((s == 1) || (p==1) || (q == 1) || (r==1))
          {
          m = m-1;
          ok = 0;
          }

          if (ok)
          {
      cout << "rivi nro: "<< m << "  " << t[0]<< ", " << t[1];
        cout << ", " <<t[2]<< ", " <<t[3]<< ", " << t[4];
        cout << ", " <<t[5]<< ", " <<t[6];
        cout << "\n";

/* Voit sijoittaa hyväksytyt rivit taulukkoon halutessasi
          o[m][0] = t[0];
          o[m][1] = t[1];
          o[m][2] = t[2];
          o[m][3] = t[3];
          o[m][4] = t[4];
          o[m][5] = t[5];
          o[m][6] = t[6];  */
    //    cout << " m: " << m << "\n";
          }
}


void generoi_rivit()
{

/* Generoidaan kaikki lottorivit ja poistetaan rivejä sen jälkeen;
```

```
ilmoitetaan paljonko rivejä poistuu (j kpl) ja paljonko rivejä
olisi sekä paljonko on vielä veikattava rivejä poistamisen
jälkeen. Käytetään OR-funktiota, kun ollaan
epävarmoja poistoehtojen päällekkäisyydestä. */


      for ( a = 1; a <= 33; a++)
      for ( b = a+1; b <= 34; b++)
      for ( c = b+1; c <= 35; c++)
      for ( d = c+1; d <= 36; d++)
      for ( e = d+1; e <= 37; e++)
      for ( f = e+1; f <= 38; f++)
      for ( g = f+1; g <= 39; g++)
      {
         t[0] = a;
         t[1] = b;
         t[2] = c;
         t[3] = d;
         t[4] = e;
         t[5] = f;
         t[6] = g;
         vertaa(t);

// if ( m % 5000 == 0)
// cout << "Riveja on yhteensa: " << m << "\n";
    }
}

//  15380937  riviä pitäisi tulla ilman suodatuksia!

void tulosta()
{
        for (z = 0; z <m; z++)
        {
        cout << "rivi nro: "<< z << "  " << o[z][0]<< "\t" <<
o[z][1];
        cout << "\t" <<o[z][2]<< "\t" <<o[z][3]<< "\t" << o[z][4];
        cout << "\t" <<o[z][5]<< "\t" <<o[z][6];
        cout << "\n";
        }
}


void main()
{
      m = 0;
      generoi_rivit();

// Tarvittaessa voit tulostaa taulukon
//      tulosta();
}
```

Vakioveikkauksessa kaikkien mahdollisten rivien lukumäärä lasketaan tuloperiaatteen mukaan eli kolmesta vaihtoehdosta (1, x, 2) voidaan tehdä valinta kussakin 13:ssa kohteessa.

Rivien määrä on siis 3^{13} = 1594323 riviä.

5.4 Permutaatioiden lukumäärä

Permutaatio kertoo, kuinka moneen eri järjestykseen voidaan jonon alkiot asettaa. Esimerkiksi kolme alkiota a, b, c voidaan asettaa seuraaviin järjestyksiin:

1	a	b	c
2	a	c	b
3	b	c	a
4	c	b	a
5	b	a	c
6	c	a	b

Järjestettyjen jonojen eli permutaatioiden lukumäärä on siis kolmella alkiolla kuusi.

Yleismuodossa permutaatioiden lukumäärä saadaan *kertoma*kaavalla:
Jos alkiota on n kappaletta, permutaatiota on **n!** kappaletta.

Edellisessä esimerkissämme permutaatioiden lukumäärä oli siis 3! = 1*2*3 = 6 kpl.

5.5 Kumulatiivinen summa

Kumulatiivinen summa annetuista arvoista on hyvin yleisesti käytetty ja tärkeä tekijä erilaisissa analyyseissä.

Oletetaan, että lukuarvot on talletettu luetteloon summat[maara].

Kumulatiivinen summa:

```
#include <iostream.h>
```

```
const int maara = 5;

void main()
{
int kohta, valisumma, summat[maara], luku;
valisumma = 0;
for (kohta = 0; kohta < maara; kohta++)
{
 cout << "Anna luku \n";
 cin >> luku;
 valisumma = valisumma + luku;
 summat[kohta] = valisumma;
 }

for (kohta = 0; kohta < maara; kohta++)
cout << "Summa on " << summat[kohta] << "\t";

}
```

5.6 Bittioperaatiot

Bitin tilan katsominen

Bittijonon tietyn bitin tilan katsominen voidaan tehdä maskaamalla muut bittipaikat katsottavan bitin ulkopuolelle.

Bittioperaattorit lyhyesti:

&	JA
|	TAI
<<	vasemmalle siirto
>>	oikealle siirto
~	yhden komplementti
^	ehdoton tai (XOR)

Sijoittakaamme muuttujiin a ja b seuraavat bittijonot ja tarkastelkaamme bittioperaattoreiden vaikutuksia:

a = 10101010 (170)
b = 11111111 (255)

Nyt bittioperaattorit antavat seuraavia tuloksia:

a & b 10101010

11111111
10101010(170)

ja

a | b 10101010
11111111
11111111(255)

ja

a ^ b 10101010
11111111
01010101(85)

ja

a << 1 01010100 (84, unsigned char, int: 340)
b << 1 11111110(254, unsigned char, int: 510)

Jos siis vasemmalle siirrossa bitti 'ei mahdu tilaansa', se heitetään pois; muutoin se siirtyy eteenpäin vasemmalle.

ja

a >> 2 00101010 (42)
b >> 2 00111111 (63)

ja

~a 01010101
~b 00000000

Huomaa erityisesti vasemmalle siirrossa tietotyyppi: mahtuvatko siirtyvät bitit 'tilaansa' vai heitetäänkö ne pois.

Kun jonkun bitin tilaa katsotaan, voimme hyödyntää oikealle siirtoa (>>) ja bittitason ja-operaattoria (&).
Tällöin bittijonoa siirretään oikealle yhtä monta askelta kuin haluttu bittipaikka on.
Jos kohteena on esimerkiksi bittipaikka 3, siirretään bittijonoa oikealle 3 kertaa.

Sen jälkeen tehdään bittikohtainen ja-operaatio siirretyn bittijonon ja arvon 1 välillä.

Esimerkki:

Olkoon bittijono = 10011011 (155)
Haluamme tietää, mikä on 3. bitin tila (LSB:n painoarvo on 0).
Siirrämme bittijonoa 3 pykälää: 10011011 >> 3
Tuloksena on jono: 00010011

Toteutamme sitten uuden jonon ja arvon 1 (0000 0001) ja-operaation:

0001 0011
0000 0001 &
0000 0001 SAADAAN TULOS: Kyseinen bitti on 1.

Bittien käsittely:

```
#include <iostream.h>
#include <conio.h>

class oheislaite
{
private:
unsigned char portti;
public:
void aseta(unsigned char x){portti = x;};
unsigned short tulostaportti(){return portti;};
friend int katsobitti(oheislaite o, int paikka);
}   ;


int katsobitti(oheislaite o, int paikka)
{
int i;
int b = o.portti;
i = (b >> paikka) & 1;
return (i);

}

void main()
{
int a;
oheislaite kk;
kk.aseta(155); // 10011011
kk.tulostaportti();
for (int i = 0; i < 8; i++)
{
```

```
cout << " Kyseisen oheislaitteen " << i << " . bitti on " <<
katsobitti(kk, i) << "\n";
}
getche();

}
```

Vielä bitin kääntämisestä:
Kuinka asettaisimme bitin tilan nollaksi, jos se on päällä?

Ja-operaattorilla voimme nollata bitin. Tällöin muita bittipaikkoja vastaavien maskin bittien tulee olla ykkösiä, jolloin bittijonon ykköset ja nollat eivät muutu nollattavan bitin ulkopuolella. Nollattavan bitin tila on maskissa nolla.

Esimerkiksi
Jos meillä on tilatavu = 10011011 (155)
Jos haluamme nollata 4. bitin, voimme tehdä operaation
1001 1011 & 111**0** 1111 ja tulos olisi 1000 1011.

Entä toisinpäin?
Ehdoton tai-operaattori kääntää bitin tilan päinvastaiseksi. Maskaamme muut bitit
1000 1011 ^ 000**1** 0000 ja tulos olisi 1001 1011.