Eesti Teadusagentuur
Estonian Research Council
TeaMe
Euroopa Liit
Euroopa Sotsiaalfond
Eesti tuleviku heaks
TTÜ
informaatikainstituut

Tüüpalgoritmid loenditega

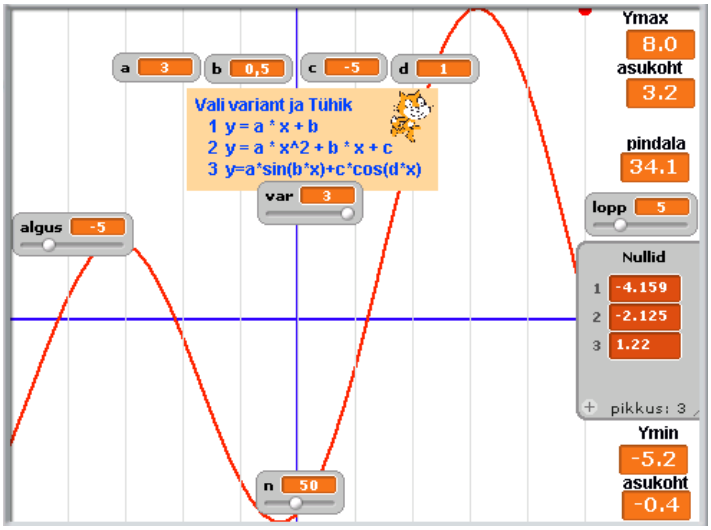
Loendid ehk ühemõõtmelised massiivid leiavad rakendustes laialdast kasutamist.

Antud materjalis vaadeldakse mitmeid tüüptegevusi loendite kasutamisel tuginedes projektidele „Funktsiooni uurimine“ ja „Loto“. Lisaks tutvustatakse mõnda tuntumat sorteerimisalgoritmi.

Sisu

Rakendus „Funktsiooni uurimine“

Ülesande püstitus



Koostada rakendus, mis võimaldab teha erinevate ühemuutuja funktsioonide y = F(x), graafikuid etteantaval lõigul [algus; lõpp] ja leida samal lõigul mõned karakteristikud:

- funktsiooni minimaalse ja maksimaalse väärtuse ning nende asukohad (vastavad x-d),
- pindala joone ja x-telje vahel,
- nullkohad: x väärtused, mille korral **y** = 0.

Peab saama kasutada erinevaid funktsioone, koostades vastavaid eeskirju (skripte) funktsioonide väärtuste arvutamiseks.

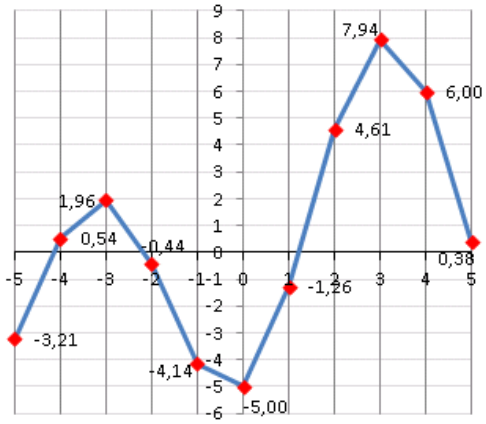
Kasutajaliides laval võiks olla umbes selline, nagu kõrvaloleval pildil. Kasutaja peab saama anda ette lõigu otsapunktide (algus ja lõpp) väärtused. Nende sisestamiseks ja muutmiseks võiks kasutada liugureid muutujate monitoridel laval. Graafikul peaks olema kuvatud koordinaatteljed ning vähemalt X-telje jaotised.

Analüüs

Nii karakteristikute leidmiseks kui ka graafiku joonestamiseks peab arvutama argumendi **x** ja funktsiooni **y** väärtused teatud hulgas punktides. Selleks võib lõigu [**a**; **b**] jagada **n** võrdseks osaks ning arvutada igas punktis vastavad funktsiooni väärtused. Mida suurem on jaotiste arv, seda täpsemalt leitakse karakteristikud (maksimum, pindala, nullkohad jm) ja seda „siledam“ tuleb graafik.

x	-5	-4	-3	-1	0	1	2	3	4	-2	5
y	-7,20	-3,59	-0,32	-0,44	-1,15	0,43	3,95	6,97	7,14	0,59	4,25

Näiteks on mingi funktsiooni jaoks välja arvutatud väärtused lõigul [-5; 5], mis on jagatud 10-ks osaks. Nende väärtuste alusel tehtud graafik kujutab endast murdjoont. Suurendades jaotiste arvu saame sujuvama joone, nagu näha ülemisel joonisel.



Rakenduse loomisel tuleb tavaliselt valida mõistlikud piirid jaotiste arvu jaoks. See sõltub suurel määral rakenduse olemusest ja eesmärkidest. Liiga väike jaotiste arv ei anna õiget pilti uuritavast funktsioonist. Suur jaotiste arv aga põhjustab ülearuseid arvutusi, mis ei pruugi eriti suurendada tulemuste täpsust. Olgu märgitud, et Scratchi praeguses versioonis võtavad mahukamad arvutused üsna palju aega. Kindlasti peab kasutaja saama teatud piirides muuta lõigu pikkust ja jaotiste arvu. Praegu eeldame, et lõigu pikkus (lõpp – algus) on piirides 5 kuni 20 ühikut. Sellisel juhul võiks olla kasutusel 10 kuni 100 jaotist.

Rakenduses on näiteks toodud kolm funktsiooni:

1. sirgjoon **y = a * x + b**
2. ruutparabool **y = a * x2 + b + c**
3. trigonomeetriline funktsioon **y = a * sin(b * x) + c * cos(d * x)**

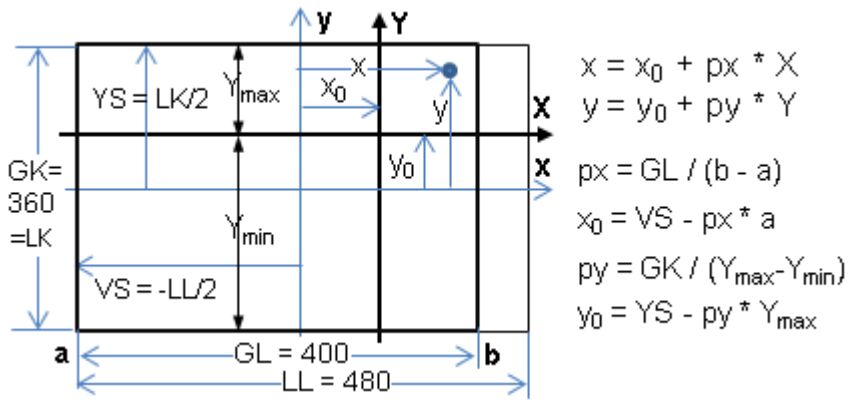
Kasutaja saab sisestada kordajate **a**, **b**, **c** ja **d** väärtusi, arvestades funktsiooni liiki (võrrandit) ja uurida erinevaid funktsioone. Kasutaja võib lisada ja/või asendada funktsioone.

Kuna argumendi ja funktsiooni väärtusi kasutatakse karakteristikute leidmisel ja graafiku joonestamisel korduvalt, siis on otstarbekas need väärtused arvutada üks kord ja salvestada loenditesse (massiividesse). Loendisse on otstarbekas salvestada ka nullkohtade väärtused.

Üheks oluliseks küsimuseks graafiku joonestamisel on sobivate mastaapide valimine horisontaal- ja vertikaalsuunas, et graafik mahuks ära lavale ning täidaks selle ratsionaalselt. Tegemist on ühe klassikalise arvutigraafika probleemiga: erinevate koordinaatsüsteemide kasutamisega ja üleminekuga ühest süsteemist teise. Selle probleemi lahendamine toimub erinevates programmeerimissüsteemides analoogselt.

Määravaks on ekraani **füüsiline koordinaatsüsteem**, mis teatud määral sõltub konkreetsest programmeerimissüsteemist (keelest). Süsteemis on tavaliselt fikseeritud mõõtühikud, telgede suunad ja piirangud kasutatava ala (piirkonna) suurusele.

Scratchis on füüsilised koordinaadid seotud lavaga, mis on 480 ühikut lai (LL) ja 360 ühikut kõrge (LK). Tegemist on tingühikutega (pikselitega), millele ei vasta mingit mõõtühikut. Koordinaatsüsteemi nullpunkt asub lava keskel, x-telg on suunatud paremale, y-telg üles. Graafiku ala saab olla lava mõõtmetega võrdne või väiksem. Praegu on Scratchi lava arvestades mõõtmed valitud järgmiselt: graafiku laius GL=400 pikselit, graafiku kõrgus GK=360.



x, **y** – punkti koordinaadid lava koordinaatsüsteemis

X, **Y** – punkti koordinaadid kasutaja koordinaatsüsteemis

x0, **y0** – kasutaja koordinaatsüsteemi telgede **X** ja **Y** asukoht füüsilises koordinaatsüsteemis

px, **py** – pikseleid x ja y suunas

GL, **GK** – graafiku piirkonna laius ja kõrgus (pikselites)

a, **b** – lõigu algus ja lõpp

VS, **YS** – lava vasakpoolne ja ülemine serv

Ymin, **Ymax** – funkts. min, max

Kasutaja koordinaatsüsteemi (öeldakse ka loogiline või maailma koordinaatsüsteem) valimiseks on mitmeid võimalusi, taoliste ülesannete puhul on üsna tüüpiline järgmine lahendus. Graafikul kuvatakse horisontaalsuunas vahemik **a**-st (lõigu algus) kuni **b**-ni (lõigu lõpp) ning vertikaalsuunas vahemik $\mathbf{Y_{min}}$-st (funktsiooni minimaalne väärtus) kuni $\mathbf{Y_{max}}$-ni (maksimaalne väärtus). Siit tuleneb füüsiliste ühikute (pikselite) arv kasutaja koordinaatsüsteemi ühele pikkusühikule mõlema telje suunas:

$$\mathbf{px = GL/(b - a)}\ , \quad \mathbf{py = GK/(Y_{max} - Y_{min})}$$

Viimaste alusel saab leida ka kasutaja koordinaatsüsteemi telgede X ja Y asukoha füüsilise koordinaatsüsteemi telgede suhtes: $\mathbf{x_0 = VS - px*a}$, $\mathbf{y_0 = YS - py*Y_{max}}$. Kasutaja koordinaatsüsteemi teljed

võivad olla ka väljaspool graafiku piirkonda (kui **a>0**, $\mathbf{Y_{min}>0}$). Suuruste px ja x_0 ning py ja y_0 väärtusi kasutatakse üleminekuks kasutaja koordinaatidelt (X ja Y) füüsilistele (x ja y) koordinaatidele: $\mathbf{x} = x_0 + px * \mathbf{X}$, $\mathbf{y} = y_0 + py * \mathbf{Y}$. Rõhutame, et argumendi ja funktsiooni väärtused leitakse kasutaja koordinaatsüsteemis, joonestamine aga toimub alati füüsilises koordinaatsüsteemis.

Mingeid nö eriomadusega spraite (objekte) ei ole antud rakenduse loomiseks vaja. Põhimõtteliselt võiks kõik vajaliku realiseerida ühe spraidi skriptide abil. Kuid ülevaatlikkuse huvides on otstarbekas jaotada tegevused funktsionaalsuse alusel mitme spraidi vahel.

Disain ja realisatsioon

Andmed

Loendid

- XM(*n* + 1), YM(*n* + 1) – argumendi ja funktsiooni väärtused, *n* – jaotiste arv
- NV(*) – nullkohtade väärtused; loend võib olla ka tühi

Põhimuutujad

- ***algus, lõpp*** – lõigu algus ja lõpp
- ***n*** – jaotiste arv
- ***h*** – argumendi muutuse samm
- ***min***, ***minX*** – funktsiooni minimaalne väärtus ja selle asukoht (vastav x)
- ***max***, ***maxX*** – funktsiooni maksimaalne väärtus ja selle asukoht (vastav x)
- ***X***, ***Y*** – argumendi ja funktsiooni jooksvad väärtused kasutaja koordinaatsüsteemis
- ***x***, ***y*** – argumendi ja funktsiooni jooksvad väärtused füüsilises koordinaatsüsteemis
- ***a, b, c, d*** – kordajad erinevates funktsioonides
- ***x0, y0*** – kasutaja koordinaatsüsteemi telgede asukohad füüsilises koordinaatsüsteemis
- ***px, py*** – füüsilise koordinaatsüsteemi ühikute arv kasutaja koordinaatsüsteemi ühiku kohta

Põhitegevused ja programmi struktuur

- algandmete lugemine – **Loe alg**
- argumendi ja funktsiooni väärtuste arvutamine ja salvestamine loenditesse – **Tee XY**
  funktsiooni väärtuse arvutamine antud punktis – **Leia Y**
- pindala arvutamine – **Leia pind**
- nullkohtade leidmine – **Leia nullid**
- ekstreemumite (min, max) ja nende asukoha (minX, maxX) leidmine – **Leia eks**
- graafiku joonestamine – **Tee graaf**
- graafiku joone tegemine – **Joon**
- telgede joonestamine – **Teljed**
- horisontaalsete jaotusjoonte tegemine – **Jaotised**

Peaskript

Peaskript, millest algab rakenduse täitmine, koosneb teavituskäskudest, mis käivitavad järjest alamskripte. Erandina on siin ka käsk kustuta, mis eemaldab eelmise graafiku.

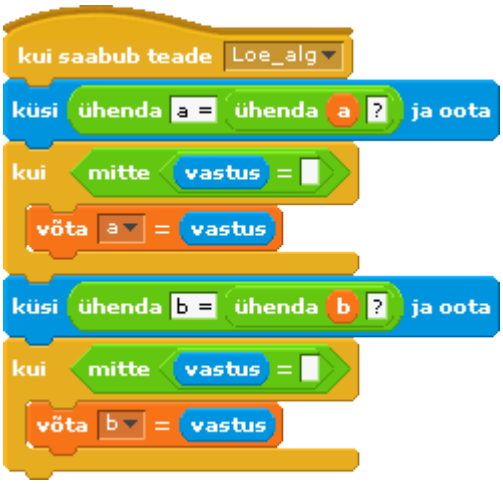
Skript **Tee XY** arvutab ja salvestab loenditesse **XM** ja **YM** argumendi ja funktsiooni väärtused ning kõik järgnevad skriptid kasutavad neid andmeid. Skripte **Leia pind**, **Leia nullid** ja **Leia eks** täidetakse paralleelselt. Kuna esimeses kahes neist puudub ootamise nõue, käivituvad skriptid praktiliselt üheaegselt peale skripti **Tee XY** töö lõppu. Skript **Tee graaf** kasutab funktsiooni minimaalset ja maksimaalselt väärtust ning seepärast ootab see eelmise skripti töö lõppu.

Algandmete lugemine – skript Loe alg

Lõigu otspunktide väärtuste **algus** ja **lõpp** ning jaotiste arvu ***n*** sisestamiseks ja muutmiseks on ette nähtud liugurid. Soovi korral võib lisada ka võimaluse nende sisestamiseks klaviatuurilt.

Käsku **küsi** kasutatakse ka funktsioonide arvutamise eeskirjades (avaldistes) esinevate kordajate (***a***, ***b***, ***c*** ja ***d***) lugemiseks. Sisestamine on korraldatud nii, et kui kasutaja ei soovi muuta mingit kordajat, vajutab ta vastuseks küsimusele klahvi **Enter**. Sellisel juhul võetakse sisemuutuja **vastus** väärtuseks tühi väärtus. Kui kasutaja ei sisestanud uut väärtust, jääb kehtima eelmine.

On näidatud ainult ***a*** ja ***b*** väärtuste lugemine.



Argumendi ja funktsiooni väärtuste arvutamine – skriptid **Tee XY** ja **Leia Y**

<u>Andmed</u>:

Loendid:

XM(*n*+1), YM(*n*+1) – argumendi ja funktsiooni väärtused

Muutujad:

algus, *lõpp* – lõigu algus ja lõpp

n – jaotiste arv

X – argumendi jooksev väärtus

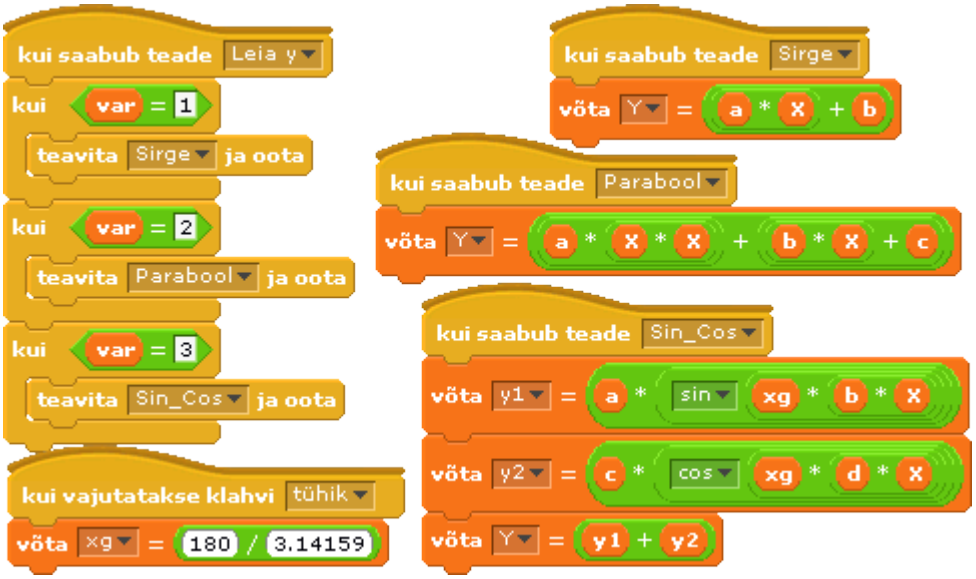
Y – funktsiooni jooksev väärtus

Algoritm

h = (lõpp – algus) / n
eemalda XM, YM
X = algus
kordus n + 1 korda
    teavita Leia_Y
    lisa XM-le X
    lisa YM-le Y
 X = X + h
lõpp kordus

Skript **Tee XY** kasutab skripti **Leia Y**, millel on omakorda kolm alamskripti: **Sirge**, **Parabool** ja **Sin_Cos**.

Viimases kasutatakse funktsioone sin ja cos, mille argument peab olema kraadides. Üleminekuks kasutatakse tegurit 180/3.14159. Selle leidmiseks on ühekäsuline skript **xg**.



Pindala leidmine – skript Leia Pind

Pindala leidmiseks kasutatakse siin trapetsivalemit:

$S = h * (|y_1| / 2 + |y_2| + |y_3| + \ldots + |y_n| + |y_{n+1}|/2)$,

kus $|y_1|, |y_2|, \ldots |y_{n+1}|$ on funktsiooni absoluutväärtused sõlmpunktides. Ilma absoluutväärtusteta annab valem määratud integraali väärtuse

Pindala leidmine. Andmed

Loend: YM(*n*+1) – argumendi väärtused**.**

Muutujad:

n – jaotiste arv

h – argumendi samm

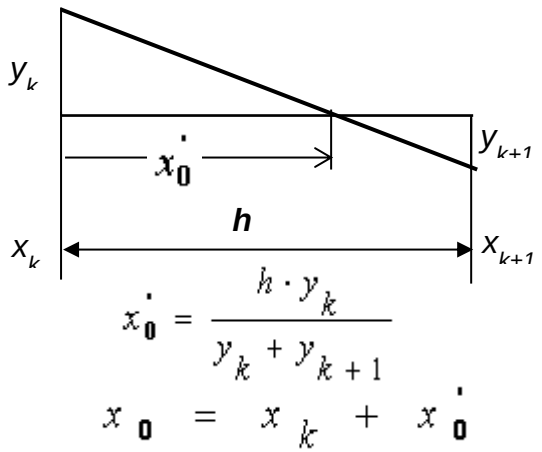
k – punkti järjenumber

S – summa ehk pindala

Pindala leidmine. Algoritm

```
S = |YM(1)| / 2
k = 2
kordus n - 1 korda
    S = S + |YM(k)|
    k = k + 1
lõpp kordus
S = h * (S + |YM(n + 1)| / 2)
```

Nullkohtade leidmine – skript Leia nullid



Nullkohtade all mõistetakse siin selliseid x väärtusi, millele vastavad y väärtused on nullid. Vaadeldaval lõigul võib olla suvaline hulk nullkohti. Nende väärtuste kindlakstegemiseks võib kasutada erinevaid viise. Eeldades, et jaotiste arv on piisavalt suur ehk lõigu alamjaotiste pikkus (***h***) on väike (näiteks 0,1 … 0,01 ühikut), võib toimida järgmiselt. Vaadata järjest läbi funktsioonide väärtused (loend YM) ning võrrelda naaberpunktide väärtusi. Kui väärtused on erineva märgiga ($y_k \cdot y_{k+1}<0$), on kahe punkti vahel nullkoht.

Nullkohtade leidmine. Algoritm

```
k = 1
kordus n korda
 kui YM(k) * YM(k + 1) < 0 siis
 x0 = h * YM(k) / (YM(k + 1) – YM(k))
 x0 = XM(k) + |x0|
 lisa NV-le x0
 lõpp kui
 k = k + 1
lõpp kordus
```

Kui tegemist on suhteliselt väikese pikkuse alamjaotistega ja ei taotleta väga suurt täpsust, võib leida nullpunkti väärtuse lihtsalt naaberpunktide x-de aritmeetilise keskmisena: $x_0=(x_k+y_{k+1})/2$. Täpsema tulemuse saamiseks võib kasutada lineaarset aproksimeerimist. Selleks vajalikud valemid on toodud kõrvaloleval pildil.

Nullkohtade leidmine. Andmed

Loendid **XM**(n+1), **YM**(n+1) – argumendi ja funktsiooni väärtused punktides.

NV(*) – nullkohtade loend. Elementide arv loendis on määramatu. Loend võib olla ka tühi, kui nullkohad lõigul puuduvad.

X0 – nullkoha väärtus (kaugus x-teljest)

h – alamjaotise pikkus: h = (lõpp - algus) / n

Ekstreemumite ja nende asukohtade leidmine – skript Leia Ekstr

Vajadust ekstreemumite (minimaalne ja/või maksimaalne väärtus mingis andmekogumis) leidmiseks tuleb ette paljudes ülesannetes. Funktsiooni minimaalse (**Ymin**) ja maksimaalse (**Ymax**) väärtuse ja nende asukohtade (**minX** ja **maxX**) leidmine on siin pandud ühte protseduuri (skripti). Mõlema suuruse jaoks toimub see praktiliselt ühtemoodi.

Ekstreemumite leidmine. Andmed

Loendid XM(n+1), Y(n+1) – argumendi ja funktsiooni väärtused.

Ymin, ***minX*** – funktsiooni minimaalne väärtus ja selle asukoht.

Ymax, ***maxX*** - funktsiooni maksimaalne väärtus ja asukoht.

n – jaotiste arv.

k – punkti järjenumber

Ekstreemumite leidmine. Algoritm

```
Ymin = YM(1); minX = XM(1)
Ymax = YM(1); maxX = XM(1)
k = 2
kordus n korda
    kui YM(k) < Ymin siis
      Ymin = YM(k)
      minX = XM(k)
    muidu
      kui YM(k) < Ymax siis
        Ymax = YM(k)
        maxX = XM(k)
      lõpp kui
    lõpp kui
    k = k + 1
lõpp kordus
```

Graafiku joonestamine – skriptid Tee graaf, Joon, Teljed, Jaotised

Graafik. Andmed

GL – graafiku laius
GK – graafiku kõrgus
VS – lava vasak serv
AS – lava alumine serv
YS – lava ülemine serv
algus – lõigu algus
lõpp – lõigu lõpp
min – funktsiooni miinimum
max – funktsiooni maksimum
px, **py** – pikseleid ühiku kohta x ja y telje suunas
x0, y0 – kasutaja telgede asukohad lava koordinaatides
XM(n+1), **YM**(n+1) – argumendi ja funktsiooni väärtused kasutaja koordinaatsüsteemis
x, **y** – punkti koordinaadid lava koordinaatsüsteemis.

Graafik. Peaskript

Graafik. Joone tegemine

Joonestamiseks kasutatakse spraiti, mis kujutab endast joonistamisredaktoriga tehtud ringikest. Kõigepealt algväärtustatakse koordinaatsüsteemide kasutamisega seotud suurused: graafiku laius, kõrgus jmt ning leitakse ***px***, ***py***, ***x0***, ***y0*** väärtused. Viimaseid kasutatakse üleminekuks kasutaja koordinaatidelt lava koordinaatidele funktsiooni joone ja telgede joonestamiseks. Joone tegemisel viiakse sprait kõigepealt graafiku esimesse punkti (lava koordinaatsüsteemis). Edasi läbitakse järjest kõik punktid (k=2, 3, …), teisendades kasutaja koordinaadid **XM(k)**, **YM(k)** lava koordinaatidesse **x** ja **y** ning kasutades liikumiseks käsku **mine x…y…**.

Graafik. Telgede joonestamine

Graafik. Jaotusjoonte tegemine

Rakendus „Loto“

Ülesande püstitus

Koostada rakendus, mis võimaldab imiteerida lotomängu sarnaselt näiteks Keno loto või Viking lotoga. Mängija sisestab või laseb arvutil genereerida teatud hulga juhuslikke üksteisest erinevaid numbreid (näiteks 3 kuni 10). Loosimisel luuakse samuti teatud hulk juhuslikke arve (näiteks 10 kuni 20). Numbrid peavad olema mingis kindlas vahemikus (näiteks 1 … 64, 1 … 36 vm). Programm peab tegema kindlaks kokkulangevate numbrite arvu piletil ja loosimisel.

Kasutajaliides võiks sisaldada järgmisi elemente: loendite monitorid, kus kuvatakse pileti (Pilet), loosi (Loos) ja kokkulangevad numbrid (Pihtas). Muutujate liuguritega monitorid, mille abil saab mängija valida numbrite arvu piletil (**np**), numbrite arvu loosimisel (**nl**) ning maksimaalse võimaliku numbri piletil ja loosimisel (**max**). Samuti võiks olla muutuja monitor, mis näitab kokkulangevate numbrite arvu (**tabas**). Vastava tegevuse käivitamiseks võib teha käsunupud või kasutada selleks mingeid muid spraite.

Analüüs

Rakendus peab võimaldama luua kaks komplekti juhuslikke arve etteantavas vahemikus. Arvud salvestatakse vastavates loendites: üks sisaldab pileti numbreid, teine loosimise numbreid. Loosimise numbreid peab olema rohkem kui pileti numbreid. Kõik pileti numbrid peavad olema erinevad. Samuti ei tohi olla ühesuguseid numbreid loosimise loendis.

Selleks, et kasutajal oleks lihtsam võrrelda ja kontrollida pileti ja loosimise numbreid, peab programm kuvama need sorteerituna kasvavas järjestuses.

Programm peab leidma kokkulangevate numbrite arvu ja kuvama need numbrid eraldi loendis (kui on). Võib kehtestada ka mingid reeglid võidusumma määramiseks.

Seega peab rakenduse loomisel realiseerima järgmised ülesanded:

- etteantud hulga üksteisest erinevate kindlas vahemikus olevate juhuarvude genereerimine ja salvestamine loendis,
- loendite elementide järjestamine (sorteerimine),
- kahes loendis olevate kokkulangevate väärtuste kindlakstegemine ja eraldamine omaette loendisse.

Numbrite genereerimine ja sorteerimine toimub pileti ja loosimise jaoks ühtemoodi, erinevus on ainult väärtuste hulgas. Hiljem näeme, et erinevate arvude loomise ja kokkulangevate numbrite kindlakstegemisega, kaasneb veel üks sageli programmides esinev tegevus – otsimine. Arvestades, et nimetatud tegevused esinevad algoritmides ja programmides tihti (eriti sorteerimine ja otsimine), on otstarbekas nende tegevuste täitmiseks luua suhteliselt universaalsed protseduurid. Kahjuks puuduvad Scratchi praeguses versioonis vahendid parameetrite ja argumentide kasutamiseks andmevahetusel skriptide vahel. Seepärast pole võimalik luua täiesti universaalseid protseduure (skripte), mida saab näiteks kopeerimise või importimise teel viia ühest programmist teise ilma spetsiaalsete meetmete rakendamiseta protseduuride

sidumiseks. Kuid kasutades võimalust spraitide ekspordiks ja importimiseks (koos skriptidega), saab tekitada siiski võimaluse üldise iseloomuga skriptide lülitamiseks erinevatesse projektidesse. Olgu märgitud, et Scratchi järgmises versioonis peaks olema võimalik parameetrite ja argumentide kasutamine, mis lihtsustab universaalsete protseduuride loomist.

Et praegu saaks kasutada erinevate arvude loomisel ja sorteerimisel pileti ja loosimise numbrite jaoks samu skripte (protseduure), võib võtta kasutusele abiloendi, milles tehakse nimetatud tegevused, ja tulemused, nö lõplikul kujul, kopeerida abiloendist pileti ja loosimise loenditesse.

Alternatiivse variandina võiks pileti ja loosimise numbrite loomiseks ja sorteerimiseks teha kaks sõltumatut skriptide gruppi: üks pileti jaoks, teine loosimise jaoks. Skriptid erinevate arvude genereerimiseks ja sorteerimiseks peab sellisel juhul dubleerima ja seadistama vastavalt kasutatavatele andmetele (loenditele).

Disain ja realisatsioon

Andmed

Põhimuutujad

- **max** – maksimaalne number piletil ja loosimisel, minimaalne väärtus on alati 1,
- **np** – numbrite arv piletil,
- **nl** – numbrite arv loosimisel,
- **tabas** – kokkulangevate numbrite arv.

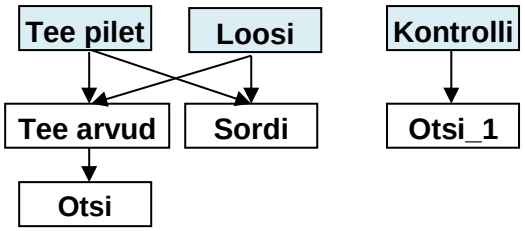
Lisaks neile on kasutusel mitmeid täiendavaid, peamiselt lokaalseid, muutujaid erinevates skriptides.

Loendid

- Pilet (*np*) – pileti numbrite loend (massiiv),
- Loos (*nl*) – loosi numbrite loend,
- Pihtas (*tabas*) – kokkulangevate numbrite loend (võib olla ka tühi),
- V(*) – abiloend pileti ja loosi numbrite loomisel ja sorteerimisel (vt allpool).

Programmi struktuur ja skriptid

On ette nähtud kolm suhteliselt sõltumatut skriptide gruppi: **Tee pilet**, **Loosi** ja **Kontrolli**. Iga grupp koosneb peaskriptist ja alamskriptidest. Täitmine algab peaskriptist, mis teadete abil paneb tööle alamskriptid.



Kuna pileti ja loosi numbrite loomine ja sorteerimine toimub analoogselt, on otstarbekas salvestada tegemise ajal pileti ja loosimise numbrid kordamööda abiloendisse ja kopeerida tulemused sealt loenditesse **Pilet** ja **Loos**.

Pileti ja loosimise numbrite tegemine – skriptid Tee pilet ja Loosi

Pileti ja loosimise numbrite loomine ja sorteerimine toimub täiesti sarnaselt. Erinevus on ainult elementide arvus **np** või **nl** ning loendis, kuhu salvestatakse tulemused. Mõlemal juhul kasutatakse abiloendit **V**, kust loodud elemendid kopeeritakse loendisse **Pilet** või **Loos**. Siin on näidatud ainult pileti numbrite töötlemine.

Pileti numbrite loomine ja sorteerimine. Andmed

np – numbrite arv piletil

Pilet – loend pileti numbritega (np väärtust)

V(*np*) – abiloend, kus toimub pileti numbrite loomine ja sorteerimine (np väärtust). Andmed kopeeritakse siit loendisse **Pilet**. Loendit **V** kasutatakse ka loosimise numbrite loomisel ja sorteerimisel.

Pileti numbrite loomine

eemalda Pileti elemendid
n = *np*
Tee_arvud
Sordi
k = 1
kordus *n* **korda**
lisa Pilet-le V(*k*)
k = *k* + 1
lõpp kordus

Erinevate arvude loomine – skriptid Tee arvud ja Otsi

Skript **Tee arvud** loob üksteisest erinevad juhuarvud ja salvestab need loendis **V**.

Tee arvud. Andmed

Loend V(*) - kasutatakse numbrite loomiseks pileti ja loosimise jaoks

Muutujad

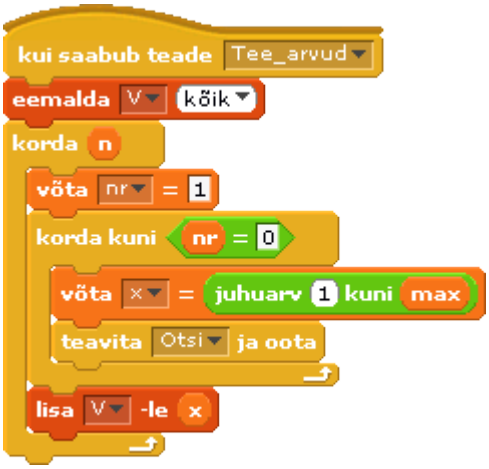
n - elementide arv loendis, võrdub ***np*** või ***nl***,

x – jooksev juhuarv,

nr – x-ga võrdse väärtuse järjenumber loendis **V** või 0, kui sellist väärtust ei ole.

Tee arvud. Algoritm

eemalda V elemendid
kordus *n* **korda**
nr = 1
kordus kuni *nr* = 0
x = juhuarv(1...*max*)
teavita Otsi
lõpp kordus
lisa V-le *x*
lõpp kordus



Otsi. Andmed

V**(*)** – loend, kus toimub otsimine

k – elemendi number loendis

nr – otsitava (*x*) järjenumber loendis või **0**, kui vastavat väärtust loendis ei ole

k – elemendi järjenumber

Otsi. Algoritm

k = 1
kordus *V.pikkus* korda
kui *x* = V(*k*) siis
nr = *k*
tagasi
lõpp kui
k = *k* + 1
lõpp kordus
nr = 0

Kõigepealt eemaldatakse loendist **V** kõik elemendid ning ükshaaval luuakse ja lisatakse loendisse uued elemendid. Iga uue elemendi korral võetakse kõigepealt muutuja **nr** väärtuseks 1 (*nr*<>0), luuakse juhuarv ja omistatakse see muutujale **x**. Skript **Otsi** kontrollib, kas **x**-ga võrdne väärtus on loendis **V** juba olemas või mitte. Seda näitab muutuja **nr** väärtus peale skripti töö lõppu. Loendi elemente võrreldakse järjest **x**-ga. Kui tingimus *x*=V(*k*) osutub tõeseks, omistatakse muutujale **nr** väärtus ***k*** ja skripti töö katkestatakse. Kui **x**-ga võrdset väärtust ei ole, jõutakse korduse lõpuni ja muutujale **nr** omistaks väärtus null. Esimesel juhul (*nr*<>0) genereeritakse uus juhuarv ja kontrollitakse uuesti. Seda tehakse seni, kuni muutuja **nr** väärtus saab nulliks (sellist väärtust ei ole). Seejärel lisatakse **x** loendi lõppu ja sama toimub järgmise elemendiga.

Paneme tähele, et esimese elemendi korral kordust ei toimu, sest loend on veel tühi (pikkus on null) ning muutujale **nr** omistatakse kohe väärtus 0.

Kui ***x***-ga võrdne väärtus on loendis olemas, näitab muutuja **nr** väärtus skripti **Otsi** töö lõppemisel selle järjenumbrit loendis **V**. Antud juhul ei ole oluline järjenumber vaid ainult see, kas **nr** väärtus on 0 (võrdset väärtust ei ole) või mitte (väärtus on). Kuid otsitava järjenumbrit loendis saab kasutada otsimistel paralleelsetes loendites või ka tabeli veergudes. Näiteks tõlkimisel, leides eestikeelse sõna järjenumbri loendis **eesti** (näiteks hiir nr 3), saab võtta sama numbriga sõna loendist **inglise** (*mouse*). Taolist lihtsat põhimõtet kasutatakse otsimiseks loendites ja tabelites sageli.

Sorteerimine

Sorteerimine seisneb väärtuse järjestamises kasvavalt või kahanevalt. Enamasti sorteeritakse kasvavas järjestuses, kuid põhimõttelist vahet siin ei ole. Reeglina toimub sorteerimine samas loendis (massiivis). Sorteerimine leiab väga sageli kasutamist andmetöötluses. Selleks on loodud palju erinevaid meetodeid ja algoritme, mille täitmise ajas võivad olla väga suured erinevused. Siin kasutatakse ühte kõige lihtsamat algoritmi nn jadasorteerimist, mis on suurte loendite korral väga aeglane. Praegu ei ole andmehulgad eriti suured.

Jadasortimine. Andmed

V – loend, mille elemente sorteeritakse (n väärtust).

n – elementide arv loendis

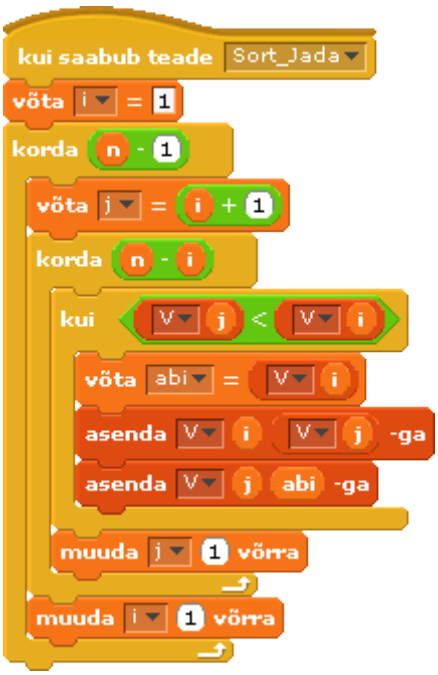
i, ***j*** – indeksid

abi – abimuutuja, kasutatakse vahetamisel

Iga elementi, alates esimesest kuni eelviimaseni, võrreldakse kõikide järgnevatega. Kui mõni neist on jooksvast elemendist väiksem, vahetatakse selle väärtus jooksva elemendiga.

Jadasortimine. Algoritm

i = 1
kordus *n*-1 **korda**
  j = *i* + 1
  kordus *n* - *i* **korda**
    kui V(*j*) < V(*i*) **siis**
      abi = V(*i*)
      V(*i*) = V(*j*)
      V(*j*) = *abi*
    lõpp kui
    j = *j* + 1
  lõpp kordus
  i = *i* + 1
lõpp kordus



Algoritmi täitmisel toimub ca n^2 võrdlust, vahetuste arv sõltub andmete sorteerituse astmest. Taoliste algoritmide kohta öeldakse, et nende **keerukus** on n^2. Elementide arvu kasvamisel n korda, suureneb aeg n^2 korda. Näiteks, kui elementide arv suureneb 10 korda, suureneb täitmise aeg 100 korda!

Kokkulangevate numbrite kontrollimine – skriptid Kontroll ja Otsi_1

Võrreldakse väärtusi kahes loendis **Pilet** ja **Loos**. Leitakse kokkulangevate väärtuste arv ja salvestatakse see muutujas **tabas**. Kokkulangevad numbrid (kui on) salvestatakse loendis **Pihtas**.

Kontroll ja Otsi_1. Andmed

Pilet– pileti numbrid (np)

Loos– loosi numbrid (nl)

Pihtas – kokkulangevad numbrid (*)

np – numbreid piletil

nl – numbreid loosimisel

tabas – kokkulangevuste arv

x – abimuutuja, jooksev väärtus loendist **Pilet**.

Kontroll. Algoritm

eemalda Pihtas elemendid
tabas **= 0**
i **= 1**
kordus Pilet.*pikkus* **korda**
  x = Pilet(*i*)
  teavita Otsi_1
  kui nr <> 0 **siis**
    tabas = *tabas* + 1
    lisa Pihtas Pilet(*i*)
  lõpp kui
  i = *i* + 1
lõpp kordus

Skript **Otsi_1** on analoogne skriptiga **Otsi**, ainult otsimine toimub siin loendis **Loos**. Iga numbrit loendis **Pilet** võrreldakse elementidega loendis **Loos** ja kui on tegemist võrdsete väärtustega, omistatakse muutujale ***nr*** järjenumber ***k***.

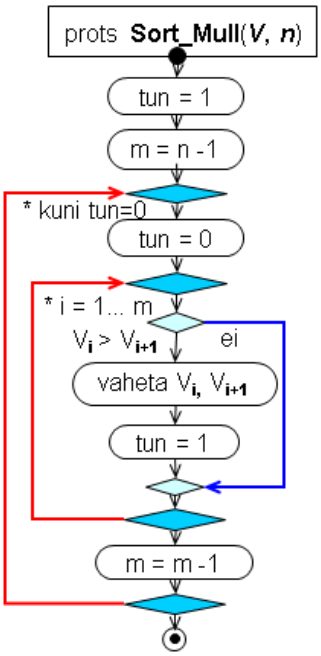
Kui ***nr*** <> 0, suurendatakse skriptis **Kontroll** muutuja ***tabas*** väärtust ühe võrra ja ***x*** (vastav element loendist **Pilet**) lisatakse loendisse **Pihtas**.

Mõned täiendavad sorteerimisalgoritmid

Mullsortimine

Mullsortimine on üks vanemaid sorteerimise algoritme. Massiiv vaadatakse korduvalt läbi. Iga kord võrreldakse naabreid ning kui V(i) > V(i+1), vahetakse need. Väiksemad väärtused „tõusevad“ ülespoole, suuremad laskuvad. Läbivaatamine kestab seni, kuni ei toimu ühtegi vahetust.

Keerukus n^2. Meetod on 2-3 korda aeglasem kui valiksortimine. Üldjuhul veidi aeglasem kui jadasortimine. Kiire, kui väärtused on peaaegu järjestatud



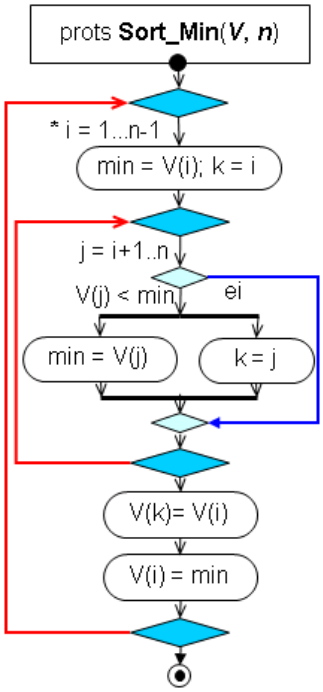

```
protseduur Sort_Mull(V, n)
 tun = 1
 m = n - 1
 kordus kuni tun = 0
   tun = 0
   kordus i = 1... m
     kui V(i) > V(i + 1) siis
     abi = V(i)
     V(i) = V(i + 1)
     V(i + 1) = abi
     tun = 1
   lõpp kui
   m = m - 1
 lõpp kordus
```

```
# Python
 def sort_mull(V):
  n = len(V)
  m = n - 1
  tun = 1
  while (tun != 0):
      tun = 0
      for i in range (m):
          j = i + 1
          if (V[i] > V[j]):
              V[i], V[j] = V[j], V[i]
              tun = 1
              m = m - 1
```

Valiksortimine minimaalse elemendi valimisega

Muudetakse indeksi i väärtust vahemikus 1 kuni n - 1, iga i korral leitakse minimaalne element vektori osas i kuni n ja vahetatakse see elemendiga i.

Keerukus **n^2**, 2-3 korda kiirem kui mullsortimine.



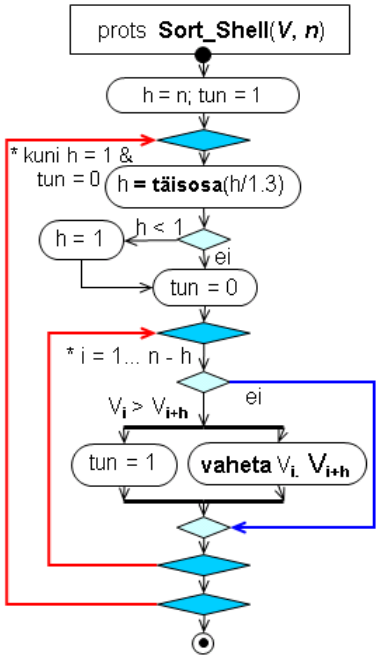

```
protseduur Sort_Min(V, n)
 kordus i = 1 kuni n - 1
   min = V(i)
   k = i
   kordus j = i + 1 kuni n
     kui V(j) < min siis
       min = V(j)
       k = j
     lõpp kui
   lõpp kordus
   V(k) = V(i)
   V(i) = min
 lõpp kordus
```

```
void sordi_min(int V[], int n) // C
{
 int min, k;
 for (int i = 1; i <= n-1; i++)
 {
 min = V[i]; k = i;
 for (int j = i+1; j <= n; j++)
 if (V[j] < min)
 {min = V[j]
 k = j; }
 V[k]= V[i];
 V[i] = min;
 }
}
```

Shelli meetod

Sarnane mullsortimisele, kuid kasutatakse **muutuvat sammu**: alguses suur, edaspidi järjest väheneb. Keerukus sõltub oluliselt sammu valikust. Üks paremaid sammu muutmise reegleid - $n_i = n_{i-1} / 1{,}3$. Keerukus ca $n*\log_2 n$. Oluliselt kiirem eelmistest, eriti kui n on suur:

n = 1000 - ca 10 korda, n = 100 000 - ca 500 korda, n = 1 000 000 - ca 5000 korda!




```
prots Sort_Shell(V, n)
 h = n
 tun = 1
 kordus kuni h=1 & tun=0
   h = täisosa(h/1,3)
   kui h < 1 siis h = 1
   tun = 0
   kordus i = 1... n - h
     kui V(i) > V(i + h) siis
       abi = V(i)
       V(i) = V(i + h)
       V(i + h) = abi
       tun = 1
     lõpp kui
   lõpp kordus
 lõpp kordus
```

```
Sub Sort_Shell(V, n) ' VBA
 Dim h, i, j, tun, abi
 h = n: tun = 1
 Do Until h = 1 And tun = 0
   h = Int(h / 1.3)
   If h < 1 Then h = 1
   tun = 0
   For i = 1 To n - h
     j = i + h
     If V(i) > V(j) Then
       tun = 1: abi = V(i)
       V(i) = V(j): V(j) = abi
     End If
   Next i
 Loop
End Sub
```