http://kodu.ut.ee/~kiho/ads/Praktikum/

5. KOP Kahend(otsimis)puud

Mõisteid

Kahendpuu* (kujutamine joonisel, graafina; programseid töötlusvahendeid)

Kahendotsimispuu, selle käitlemine

AVL-puu, selle käitlemine

* http://kodu.ut.ee/~kiho/ads/Praktikum/Arvutipraktikum/--GP2P/Sissejuhatus/Peafail.pdf

Harjutusülesanded

Kahendpuud*

KP_1. Täieliku kahendpuu valmistamine

KP_2. Kahendpuu kõrgus

KP_3. Tasakaalu kontrollimine

KP_4. Kahendpuu ja puu läbimine

KP_5. Aritmeetilise avaldise puu

KP_6. Tipu järglaste koguarv

KP_7. Tippudele kõrguse- ja tasemeväljad

KP_8. Tasemete generaator
KP_9. Kompaktse kahendpuu valmistamine
KP_10. Kahendpuu peegeldus

Kahendotsimispuud**

KOP_1. Kahendotsimispuu valmistamine

KOP_2. Kahendotsimispuu kontroll

KOP_3. AVL-puu kontroll

KOP_4. Otsimine kahendotsimispuus

KOP_5. Lisamine kahendotsimispuusse

KOP_6. Eemaldamine kahendotsimispuust

KOP_7. AVL-puu tasakaalustamine

* Vt ka http://kodu.ut.ee/~kiho/ads/Praktikum/Arvutipraktikum/5-KOP/N%C3%A4ited/Java/KahendpuuMeetodid.java

** Abiks: http://kodu.ut.ee/~kiho/ads/Praktikum/Arvutipraktikum/5-KOP/%C3%9Clesanded/Java/LisamineAVLMall.java

Abivahendid kahendpuu loomiseks, kuvamiseks ja töötlemiseks

Pakett *ee.ut.kiho.aa.graaf* (kasutamiseks: `import ee.ut.kiho.aa.graaf.*;` **)**
http://kodu.ut.ee/~kiho/ads/Praktikum/Arvutipraktikum/--GP2P/Tugi-tarkvara/doc_graaf/index.html

sisaldab klasside
- *Graaf*
- *Kaar*
- *Kahendpuu*
- *Puu*
- *Tipp*

kirjeldusi.

Kahendpuud käsitletakse kui graafi/puu erijuhtu. Seetõttu on rakendatavad graafi- ja puutöötlemise meetodid klassidest *Graaf/Puu*. Kahendpuu/puu juureks on graafi esimene tipp (tipp indeksiga 0).

Vasak-parem alluvussuhe on määratud tippude *x*-koordinaatidega:
kui tipp *t* on tipu *u* alluv (st on olemas kaar *u*-->*t*), siis
- *t* on tipu u vasak alluv, kui *t.x* < *u.x*,
- *t* on tipu u parem alluv, kui *t.x* > *u.x.*

n-tipulise juhu-kahendpuu *kp* valmistamine:
```
Kahendpuu kp = new Kahendpuu(Puu.juhupuu(n, 2));
```
Tipumärgendid: A, B, …

n-tasemelise juhu-kahendpuu *kp* valmistamine:
```
Kahendpuu kp = Kahendpuu.juhu2puu(n);
```
Tipumärgendid: tühjad

KP_1. Täieliku kahendpuu valmistamine [Õpik, lk 28]

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.
Antud: arv *m* – loodava kahendpuu tasemete arv.
Tulemus: täielik *m*-tasemeline kahendpuu.

Java

Ühetipulise (ainult juur) kahendpuu *kp* tegemine:

```
Kahendpuu kp = new Kahendpuu(new Tipp());
```

Kahendpuus *kp* alluvateta tipule *t* mõlema alluva lisamine:

```
Tipp uus = new Tipp();                  -- tühi tipp
kp.lisadaAlluv(t, uus, true);   -- true: lisamine vasakuks alluvaks
uus = new Tipp();
kp.lisadaAlluv(t, uus, false)   -- false: lisamine paremaks alluvaks
```

Ehk lühemalt:

```
kp.lisadaAlluv(t, new Tipp(), true);
kp.lisadaAlluv(t, new Tipp(), false);
```

Kahendpuu (ka graafi ja puu) kuvamine:

```
kp.faili("fail.txt");              -- kahendpuu kp faili (tabelkujul)
Graaf.kuvada("fail.txt", true);-- tippudes märgendid
Graaf.kuvada("fail.txt");          -- tippudes numbrid
```

KP_2. Kahendpuu kõrgus

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.

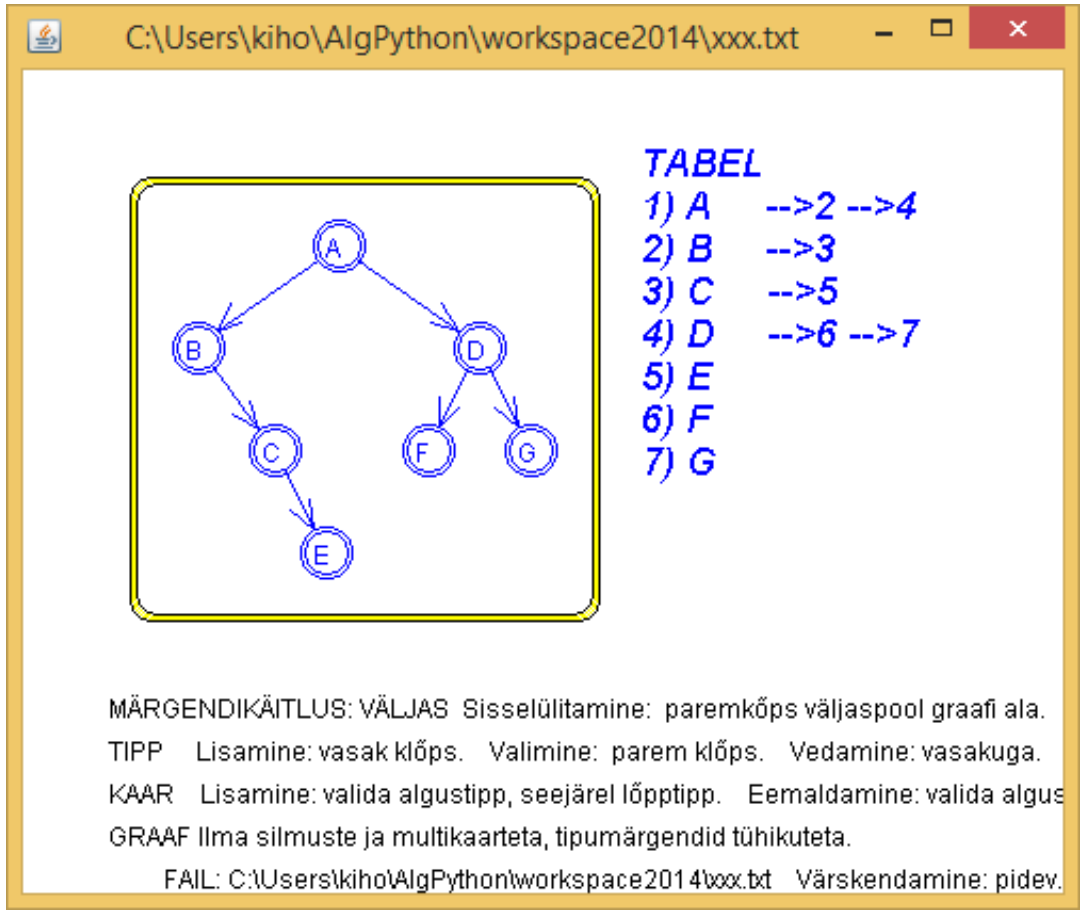
Antud: kahendpuu ja selle tipp.

Tulemus: antud tipust algava (alam)kahendpuu kõrgus.

Kahendpuu (*kp*) isendimeetodid tipu (*t*) alluva leidmiseks:

```
kp.vasakAlluv(t);
kp.paremAlluv(t);
```

Testi tulemuse näide



Alampuu juur	kõrgus
A	4
B	3
C	2
D	2
E	1
F	1
G	1

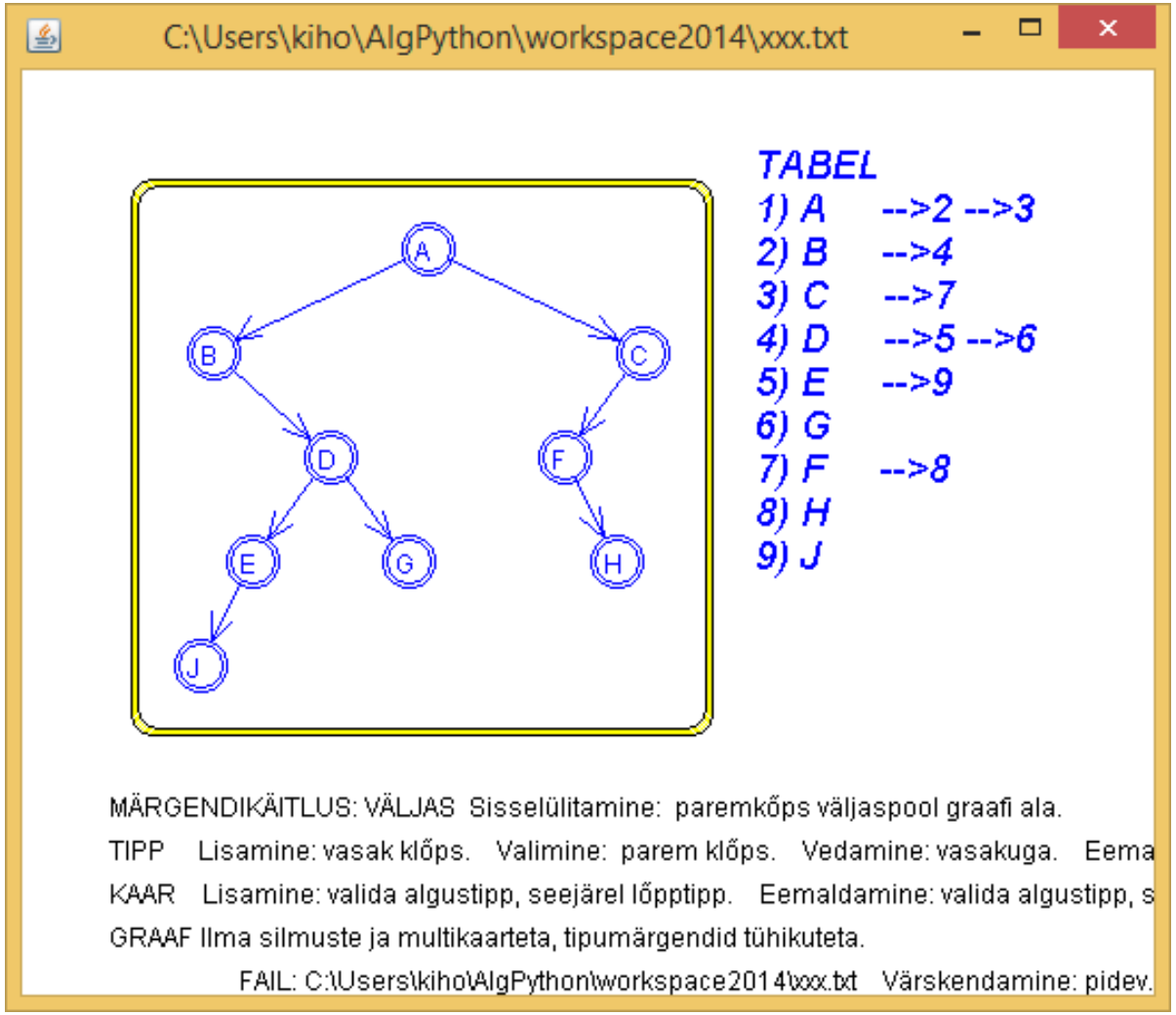
KP_3. Tasakaalu kontrollimine

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud: kahendpuu.

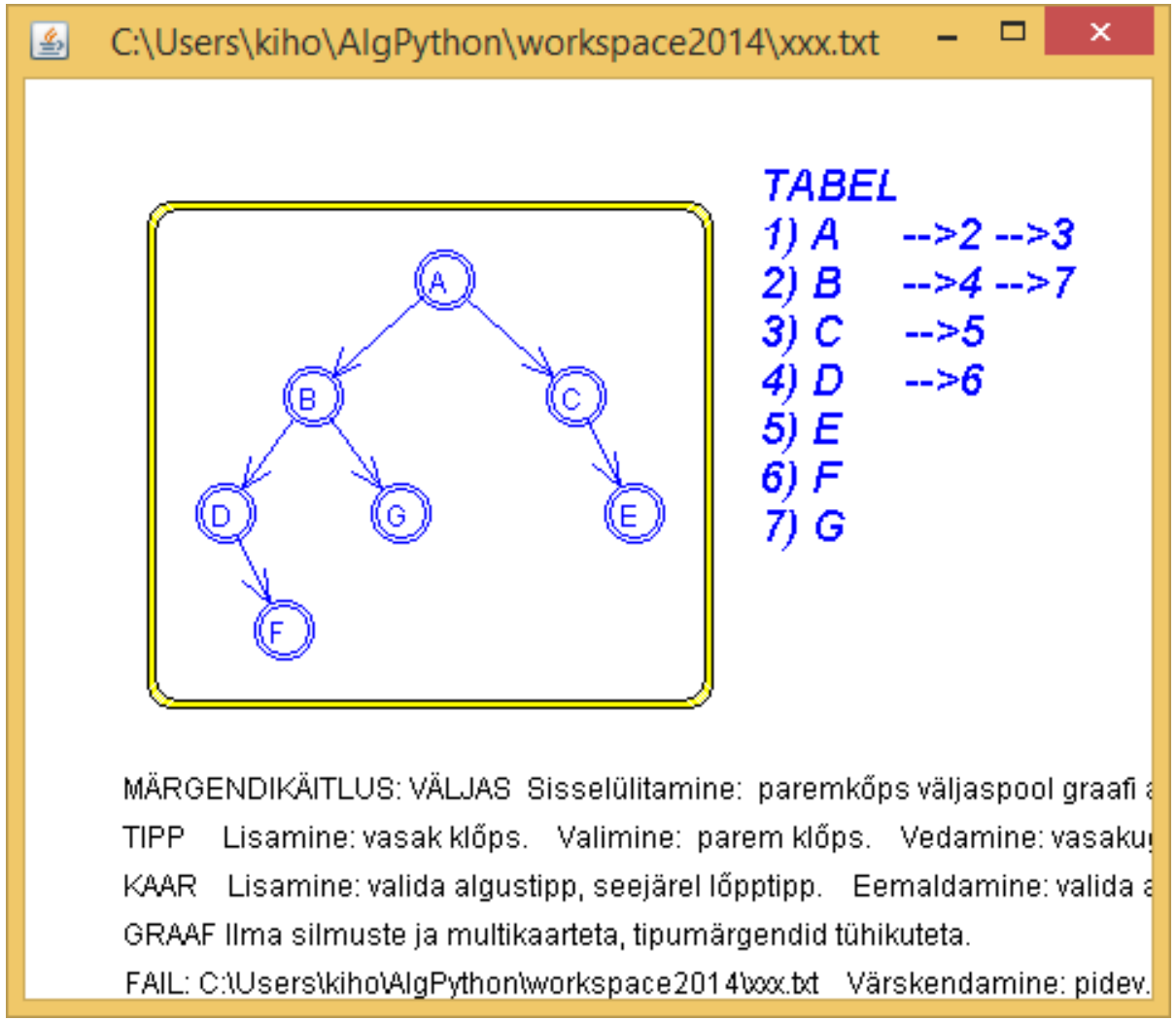
Tulemus: antud kahendpuu nende tippude järjend (listina), kus tasakaal on rikutud, st tipu alluvate kõrgus erineb rohkem kui 1 võrra.

Testi tulemuse näide




```
Tasakaal rikutud
tipus B
tipus C
```

Testi tulemuse näide



On tasakaalus.

KP_4. Kahendpuu ja puu läbimine

KP_4-1. Kahendpuu läbimine [Õpik, lk 29]
Kirjutada ja testida rekursiivsed meetodid järgmiste ülesannete lahendamiseks

KP_4-1E:
Antud: kahendpuu ja selle tipp *t.*
Tulemus: tipust *t* algava (alam)kahendpuu tipud eesjärjestuses (listina).

KP_4-1K:
Antud: kahendpuu ja selle tipp *t.*
Tulemus: tipust *t* algava(alam) kahendpuu tipud keskjärjestuses (listina).

KP_4-1L:
Antud: kahendpuu ja selle tipp *t.*
Tulemus: tipust *t* algava (alam)kahendpuu tipud lõppjärjestuses (listina).

KP_4-1E lahendus

```
static ArrayList<Tipp> eesjrst(Kahendpuu kp, Tipp t){
//Antud: kahendpuu kp ja selle tipp t (või null, kui tippu ei ole)
//Tulemus:tagastatakse listina kp tipust t algava (alam)kahendpuu tipud eesjärjestuses
    ArrayList<Tipp> tulem = new ArrayList<Tipp>();
    if (t == null) return tulem; // baasjuht
    tulem.add(t);
    tulem.addAll(eesjrst(kp, kp.vasakAlluv(t)));
    tulem.addAll(eesjrst(kp, kp.paremAlluv(t)));
    return tulem;
 }
```

KP_4-2. Puu läbimine

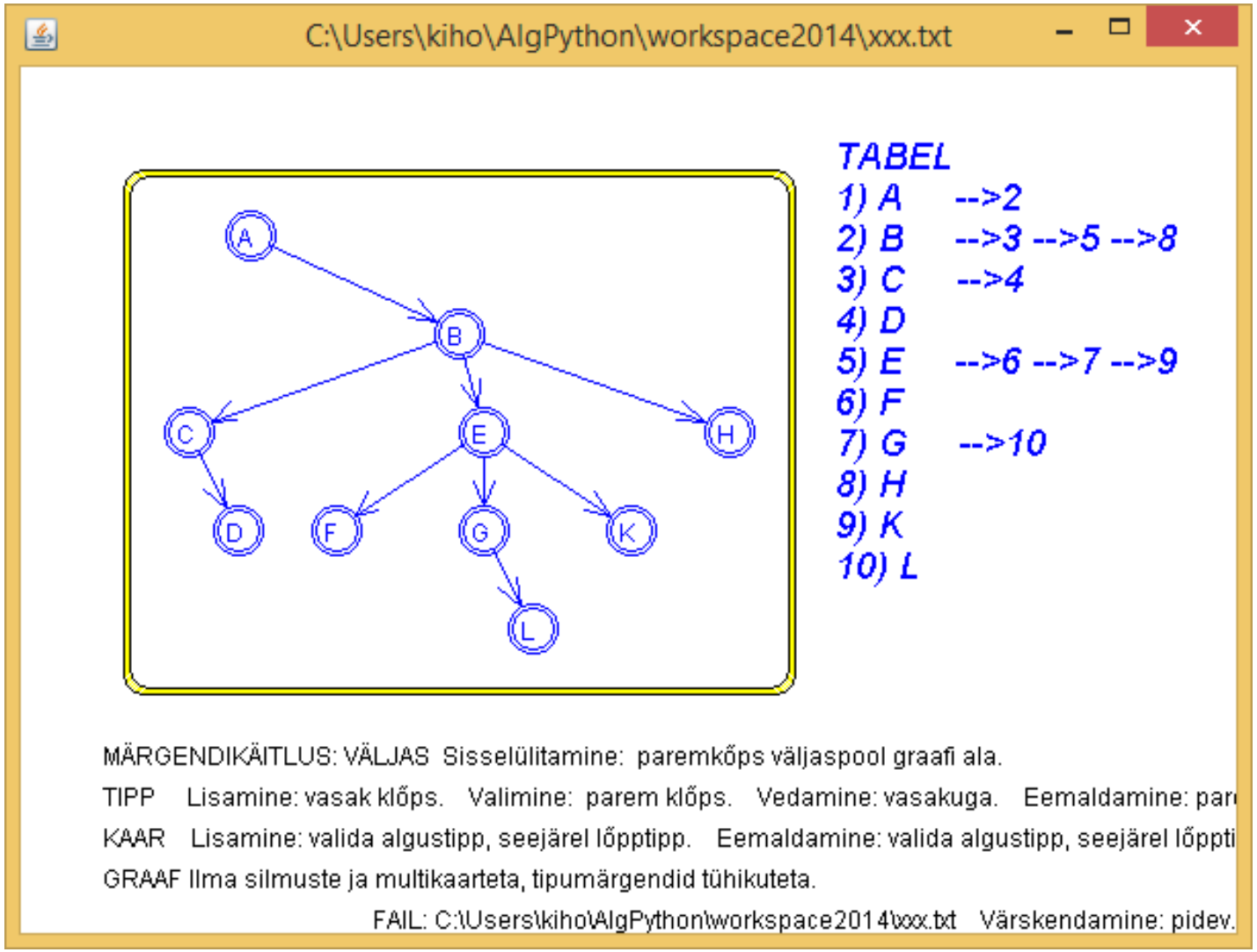
Kirjutada ja testida rekursiivsed funktsioonid järgmiste ülesannete lahendamiseks.

Antud: puu ja selle tipp *t.*

Tulemus: tipust *t* algava (alam)puu tipud eesjärjestuses (listina).

Antud: puu ja selle tipp *t.*

Tulemus: tipust *t* algava (alam)puu tipud lõppjärjestuses (listina).



Testi tulemuse näide

```
Puu eesjrst:
ABCDEFGLKH

Puu lõppjrst:
DCFLGKEHBA
```

KP_5. Aritmeetilise avaldise puu

Koostada programm, mis valmistab (juhusliku) aritmeetilise avaldise kahendpuu ja arvutab vastava avaldise väärtuse. (Vt ka järgmine slaid.)

Eraldi meetoditena kirjeldada järgmised alamülesanded:

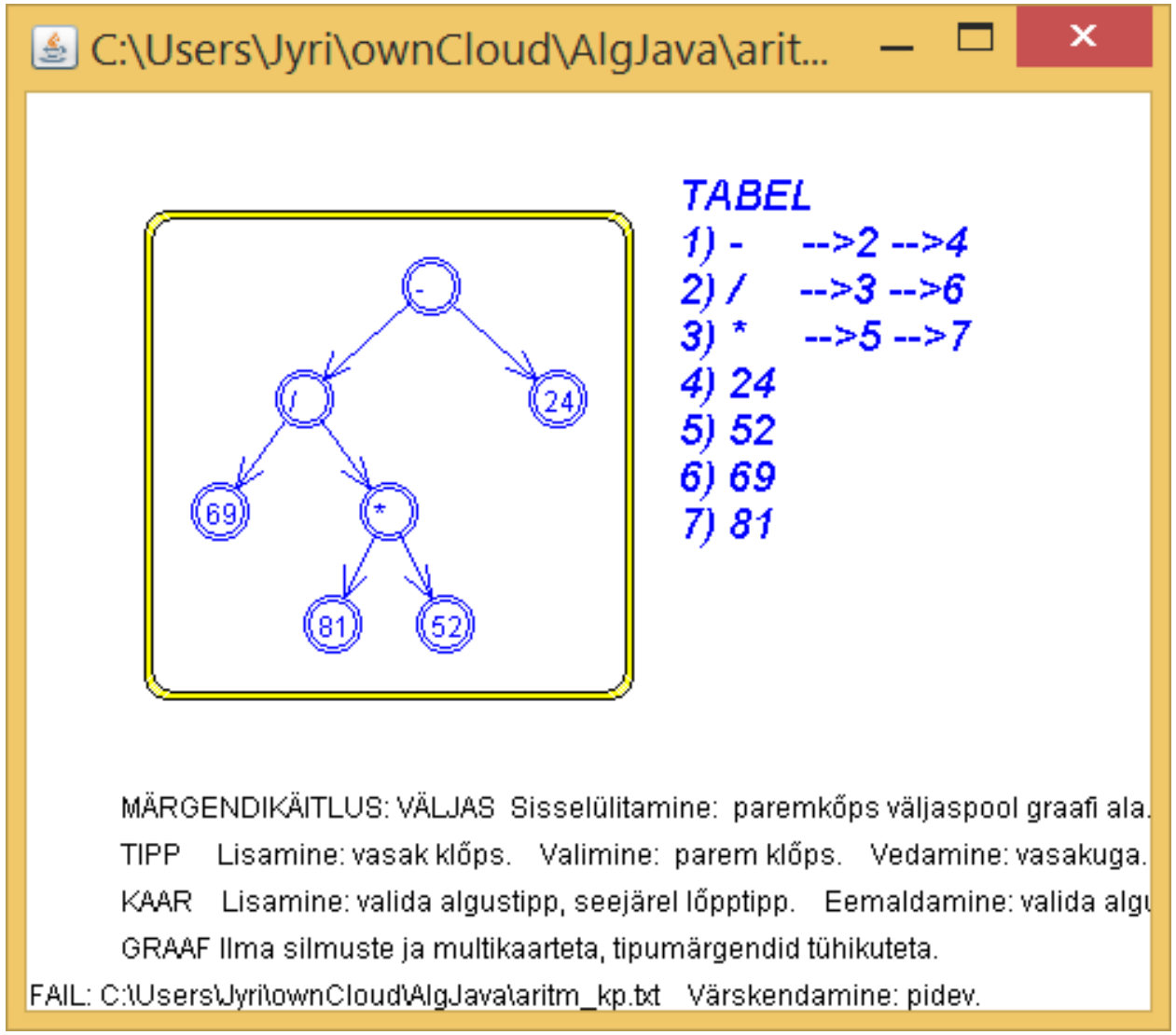
KP_5-1. Aritmeetilise avaldise puu valmistamine

Kirjutada ja testida funktsioon järgmise ülesande lahendamiseks.
Antud: kahendpuu.
Tulemus: antud kahendpuu muudetud 0-2 kahendpuuks (igale ühe alluvaga tipule lisatud teine, puudunud alluv), seejärel
rippuvate e. lehttippude märgenditeks seatud juhuarvud,
ülejäänutele – binaarsete tehete märgid, juhuvalik (+ - * /) seast.

KP_5-2. Aritmeetilise avaldise puust väärtuse arvutamine

Kirjutada ja testida rekursiivne funktsioon järgmise ülesande lahendamiseks.
Antud: aritmeetiline kahendpuu ja selle tipp *t.*
Tulemus: tipust *t* algava aritmeetilisest (alam)puust arvutatud väärtus.

Aritmeetilise avaldise kahendpuu (KP_5 testi tulemuse) näide.



Avaldise väärtus: -23.98.

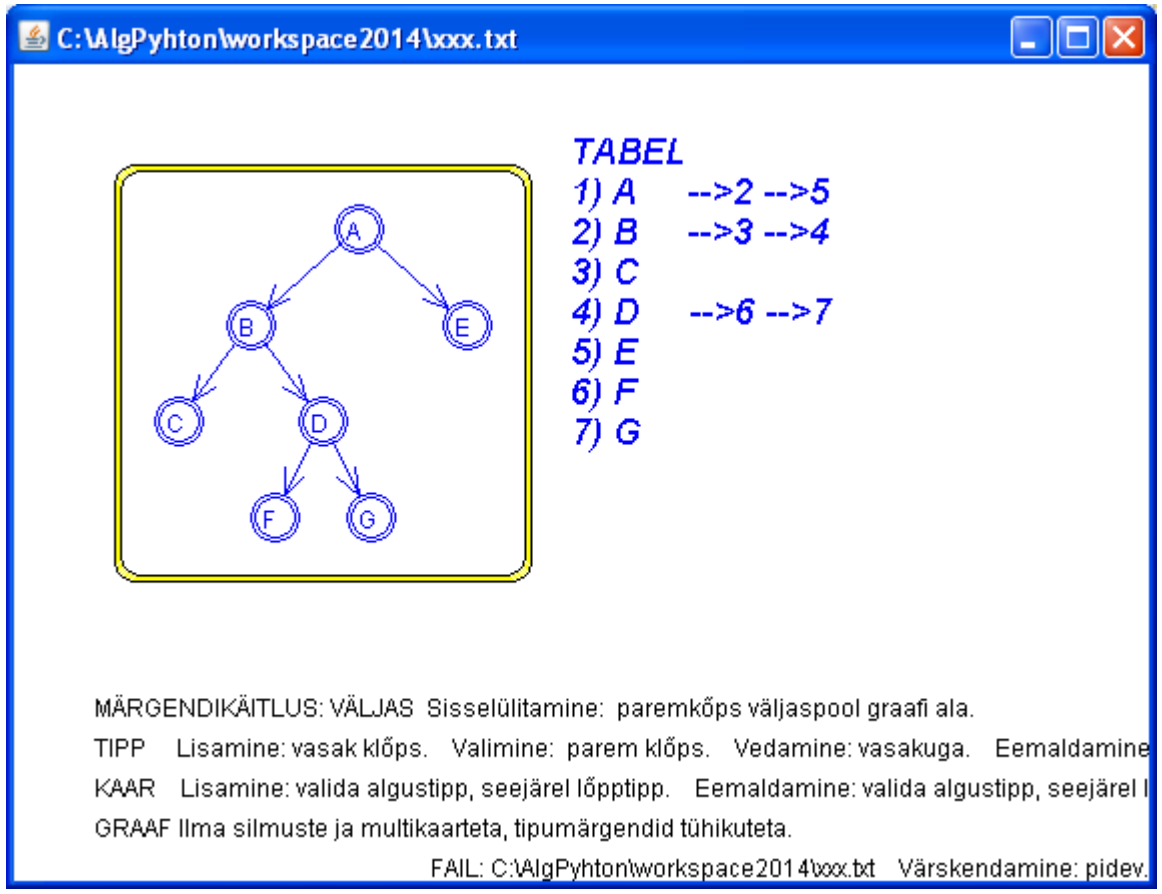
KP_6. Tipu järglaste koguarv

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.

Antud: kahendpuu ja selle tipp.

Tulemus: antud tipu kõigi (vahetute ja kaugemate) järglaste koguarv antud kahendpuus.

Testi tulemuse näide



Tipp	Järglasi
A	6
B	4
C	0
D	2
E	0
F	0
G	0

KP_7. Tippudele kõrguse- ja tasemeväljad

KP_7-1. Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.
Antud: kahendpuu *kp*, selle tipp *t.*
Tulemus: tipust *t* algavas *kp* alampuus on iga tipu *u* väljale "h" omistatud tipust *u* algava alampuu kõrgus; tagastatakse tipust *t* algava alampuu kõrgus.

KP_7-2. Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.
Antud: kahendpuu *kp*, selle tipp *t* ja tipu *t* tasemenumber *m.*
Tulemus: tipust *t* algavas *kp* alampuus on iga tipu väljale "tase" omistatud tasemenumber; antud tipu tasemenumbriks on *m*, selle alluvatel *m*+1 jne.

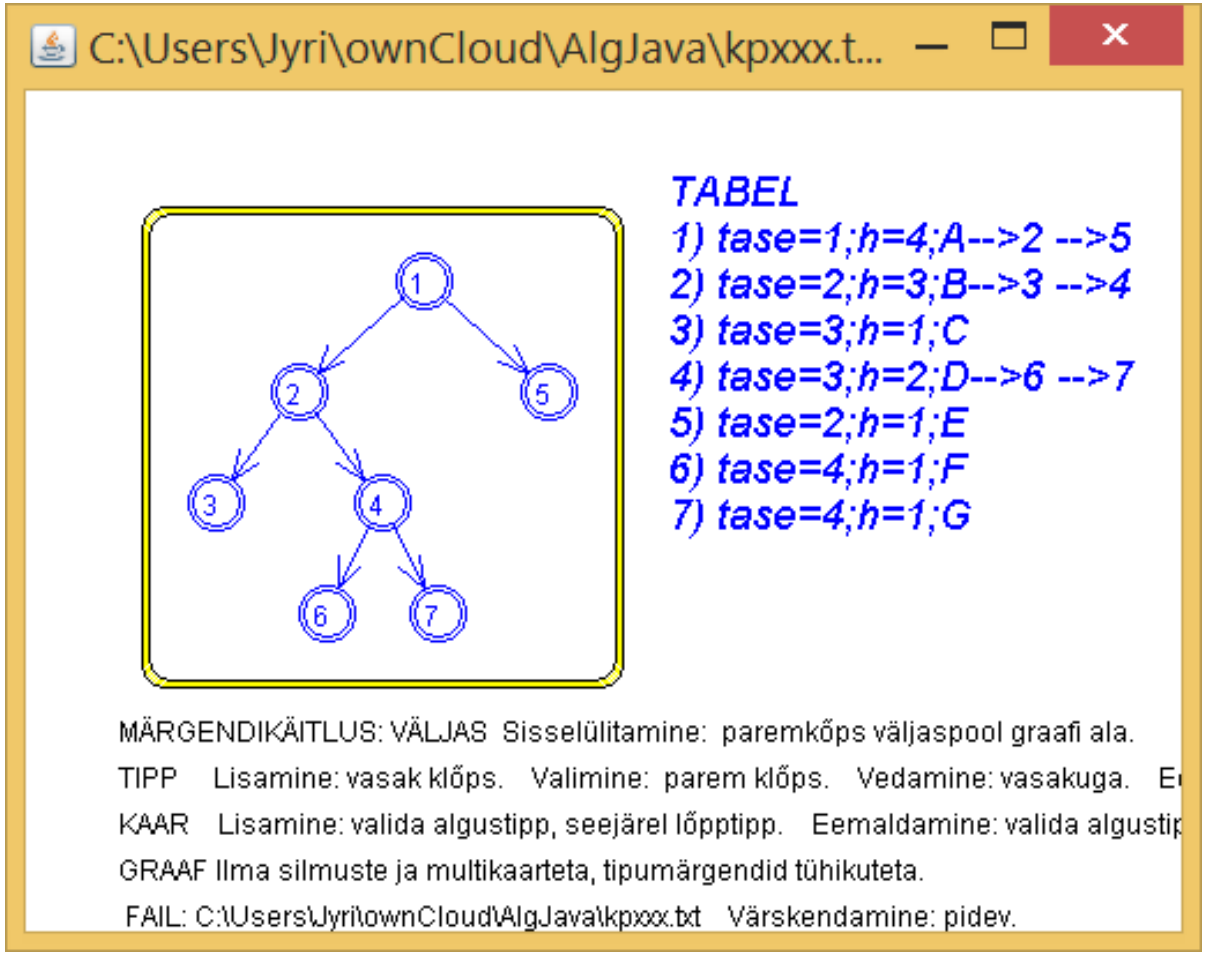
Tipu (*t*) väljale (“*v*“) arvu (*n*) seadmine:

```
t.seadaVäli(“v“, ““ + n);
```

Tipu (*t*) väljalt (“*v*“) arvulise väärtuse lugemine:

```
int n = Integer.parseInt(t.väli(“v“));
```

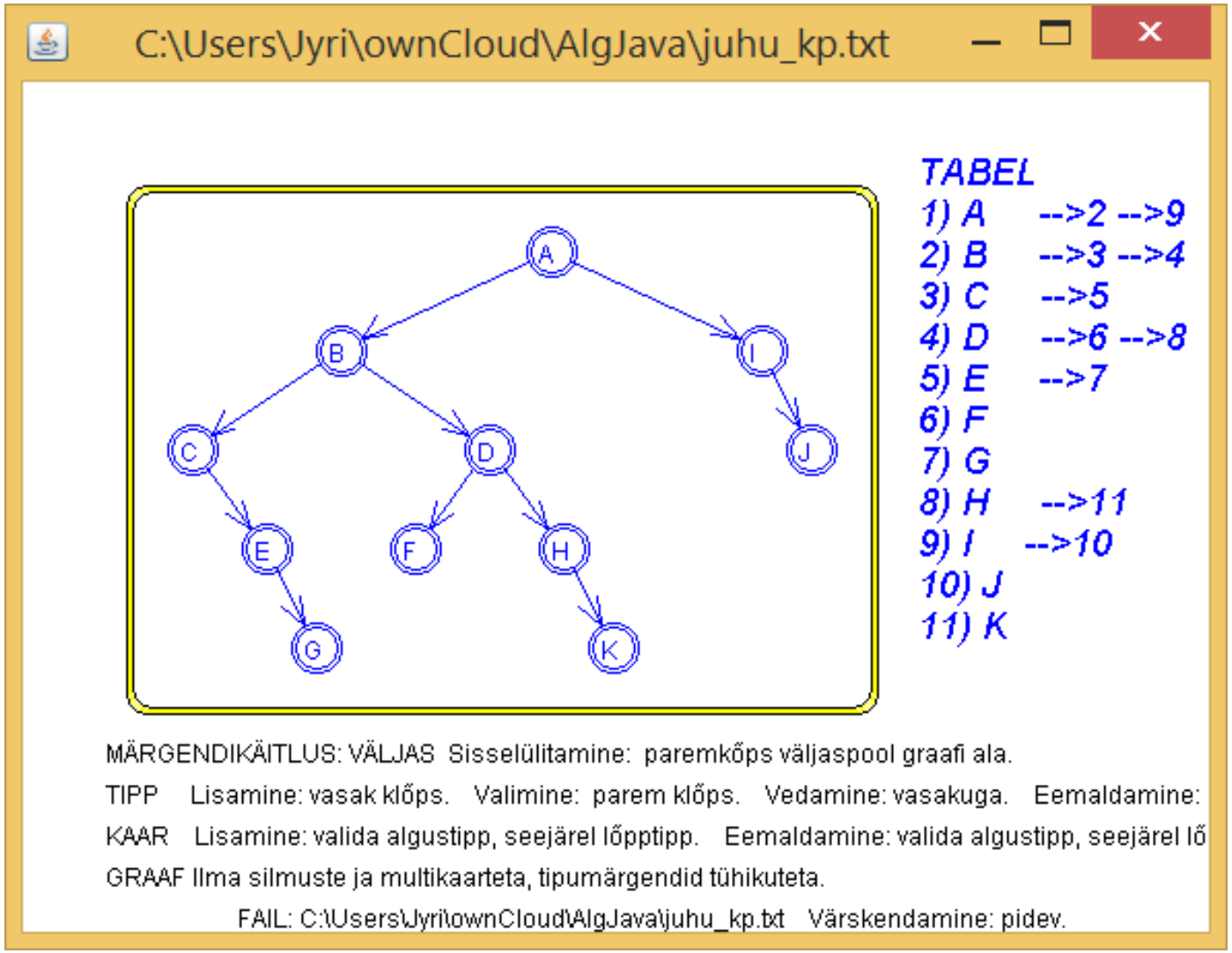
KP_7 testi tulemuse näide



KP_8. Tasemete generaator

Kirjutada ja testida tasemete generaator genereerimaks järjest antud kahendpuu tasemeid (tippude loeteludena).

Testi tulemuse näide




```
Tipumärgendid
tasemeti:
A
B I
C D J
E F H
G K
```

KP_9. Kompaktse kahendpuu valmistamine [Õpik, lk 28]
Kirjutada ja testida meetod järgmise ülesande lahendamiseks.
Antud: arv *n* – loodava kahendpuu tippude arv.
Tulemus: kompaktne *n*-tipuline kahendpuu.

Meetodi üldskeem:

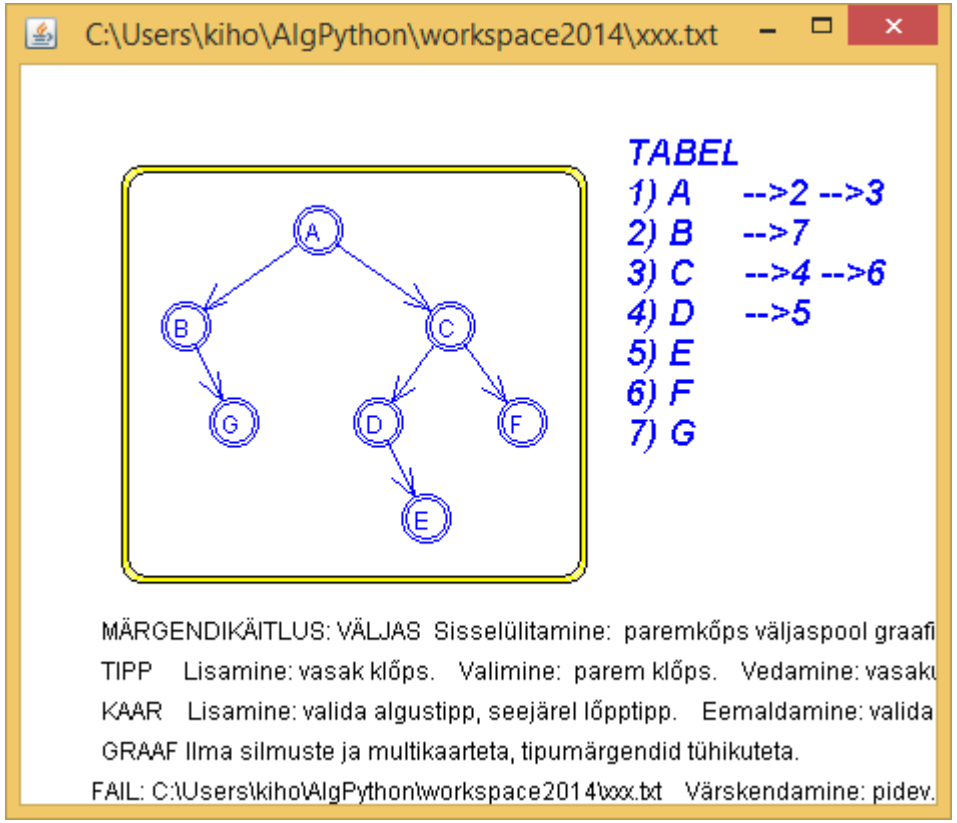
Valmistada (KP_1) täielik kahendpuu, milles on $\log_2(n+1)$ taset.
Leida (KP_8) viimase taseme tipud ja lisada neile alluvateks veel vajalik arv uusi tippe.

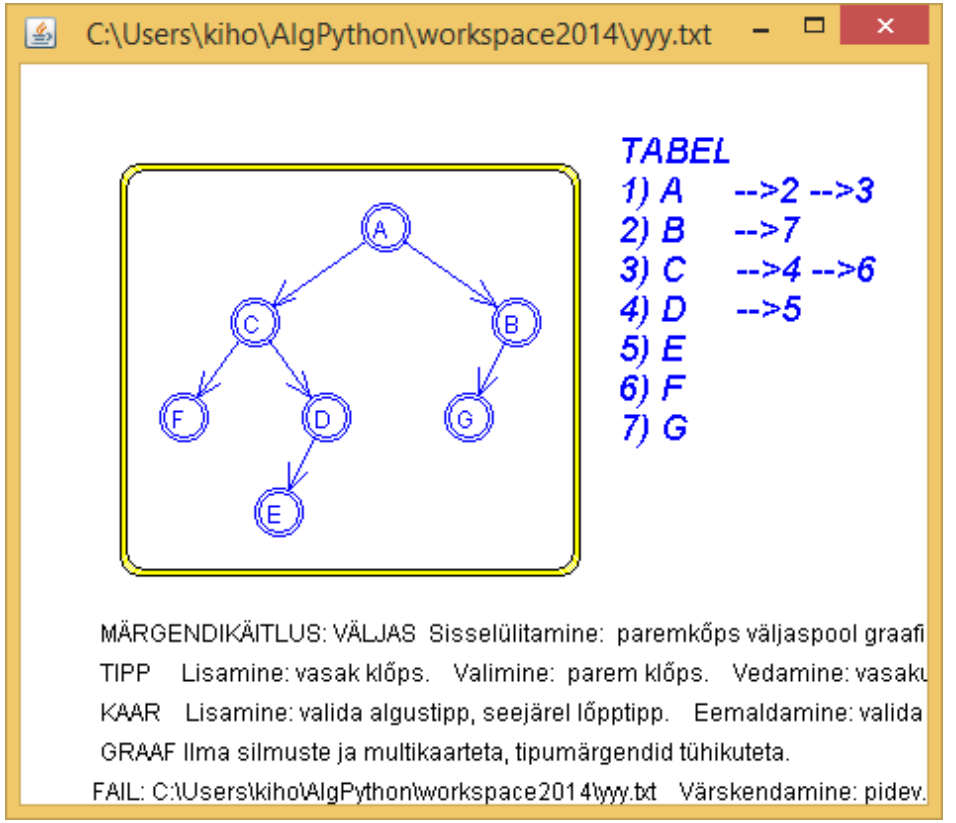
KP_10. Kahendpuu peegeldus

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.
Antud: kahendpuu.
Tulemus: antud kahendpuu peegeldus üle vertikaaltelje.

Idee: tippude *x*-koordinaadid asendada vastandarvudega.

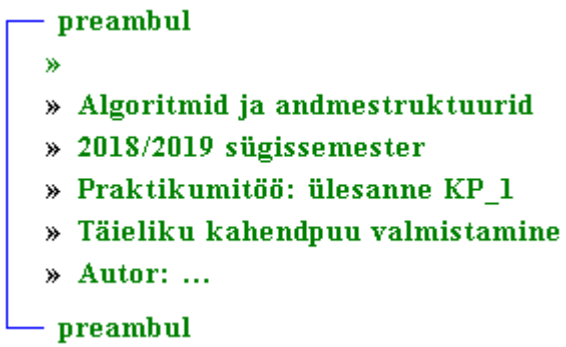
Testi tulemuse näide





Kahendpuu töötlusprogrammide skeeme

Töötlusprogrammi alguses paiknevat preambulit, näiteks KP_1 korral



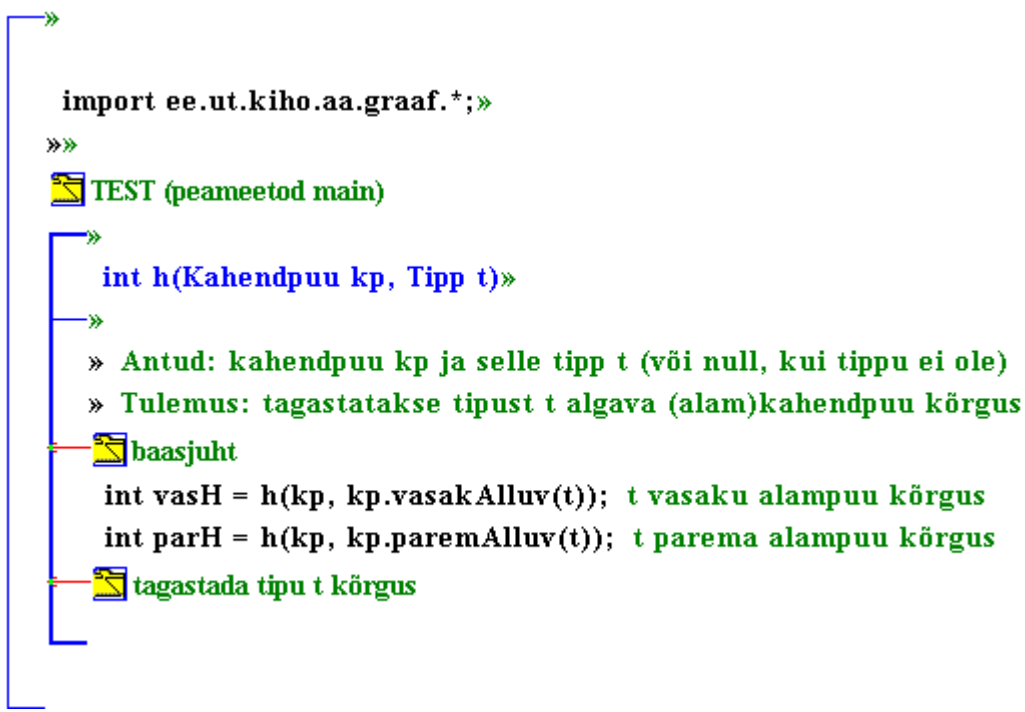
ei ole skeemis näidatud.

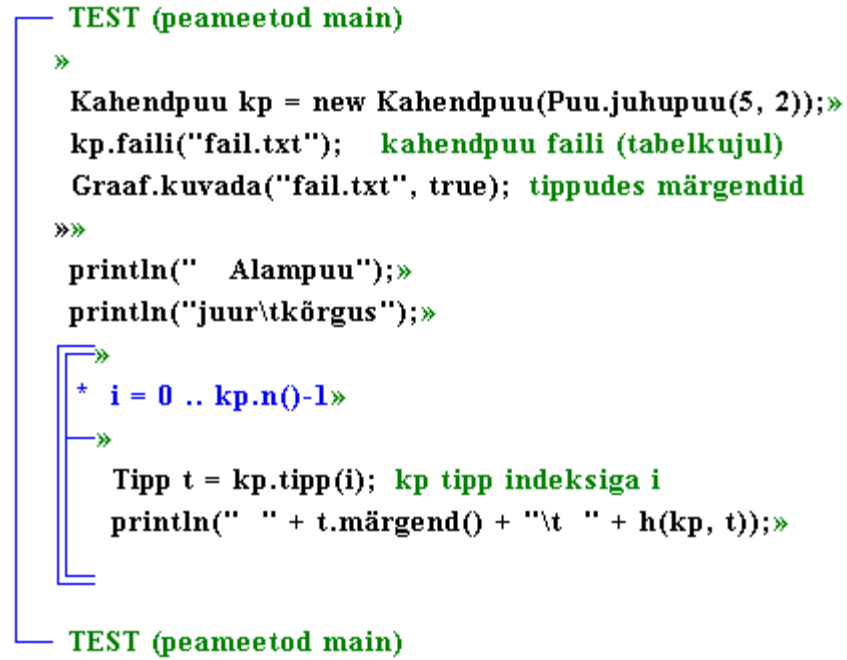
```
//                  --- TÄIELIKU KAHENDPUU VALMISTAMINE ---            Java8 //
/*************************************************************************
 *                  Algoritmid ja andmestruktuurid
 *                  2018/2019 sügissemester
 *                  Praktikumitöö: ülesanne KP_1
 *
 *                    Autor: Ülli Õpilane
 *
 *                  --- Lahendamine käsurealt (cmd) ---
 *      Eeldus: jooksvas kaustas on failid
 *          KP_1.java (see fail) ja ee_ut_kiho_aa.zip
 *      Kompileerimine:
 *                  kui see fail on ANSI kodeeringus, siis
 *         Kaustatee>javac -cp .\;ee_ut_kiho_aa.zip KP_1.java
 *                  kui see fail on utf-8 kodeeringus, siis
 *        Kaustatee>javac -encoding utf8 -cp .\;ee_ut_kiho_aa.zip KP_1.java
 *      Käivitamine:
 *              Kaustatee>java  -cp .\;ee_ut_kiho_aa.zip KP_1
 *
 *************************************************************************/
```

KP_1 skeem

```
import ee.ut.kiho.aa.graaf.*;

int m = 4;  tasemete arv
Kahendpuu kp = tehaTäielik(m);
kp.faili("fail.txt");
Graaf.kuvada("fail.txt");

Kahendpuu tehaTäielik(int m)
  » Antud: arv m – loodava kahendpuu tasemete arv
  » Tulemus: luuakse ja tagastatakse  täielik m-tasemeline kahendpuu
    baasjuht
    if (m == 1)
    return new Kahendpuu(new Tipp());  ühetipuline
    baasjuht
   Kahendpuu kp = tehaTäielik(m-1);
   int k = (int)Math.pow(2, m-2);  tippe kp viimasel tasemel
  » kp viimasel tasemel:
   int esimeseIndeks = kp.n() - k;
   int viimaseIndeks = kp.n() - 1;
  » igale viimase taseme tipule lisada 2 alluvat:
   * i = esimeseIndeks .. viimaseIndeks
      Tipp t = kp.tipp(i);  t on üks tipp viimasel tasemel
      kp.lisadaAlluv(t, new Tipp(), true);
      kp.lisadaAlluv(t, new Tipp(), false);
  return kp;
```

KP_2 skeem


»
import ee.ut.kiho.aa.graaf.*;»
»»
TEST (peameetod main)
»
int h(Kahendpuu kp, Tipp t)»
»
» Antud: kahendpuu kp ja selle tipp t (või null, kui tippu ei ole)
» Tulemus: tagastatakse tipust t algava (alam)kahendpuu kõrgus
baasjuht
int vasH = h(kp, kp.vasakAlluv(t)); t vasaku alampuu kõrgus
int parH = h(kp, kp.paremAlluv(t)); t parema alampuu kõrgus
tagastada tipu t kõrgus

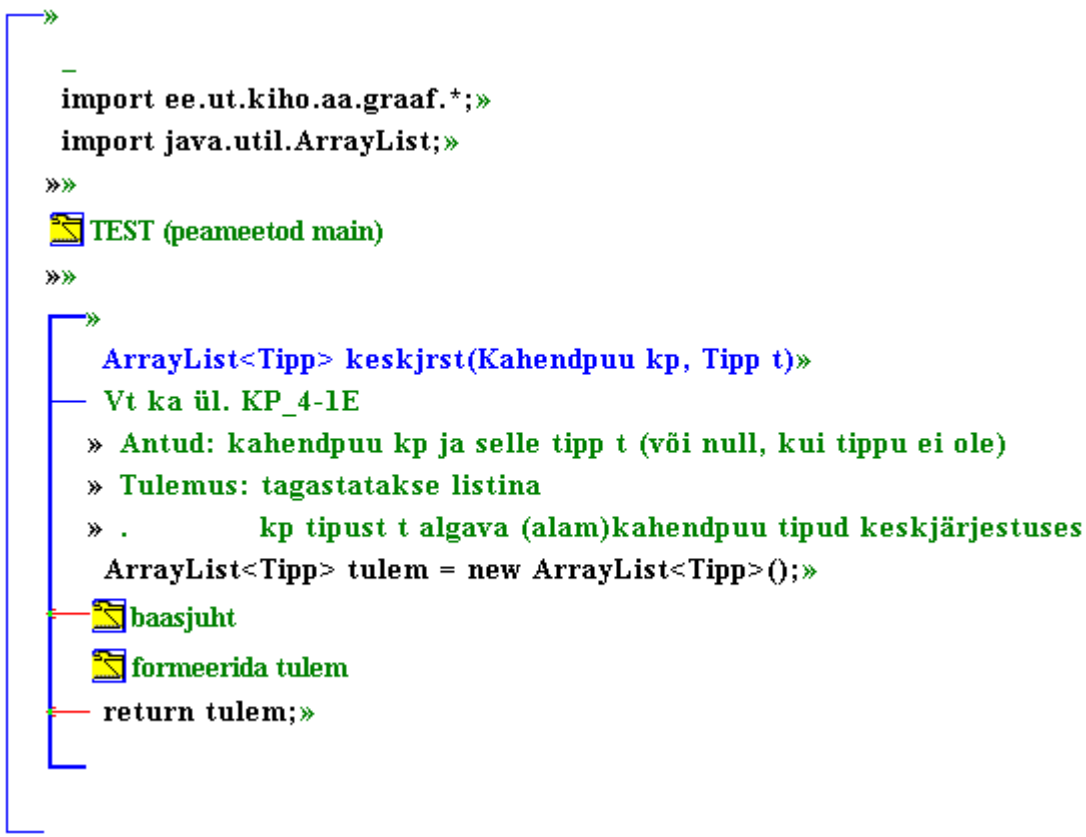

TEST (peameetod main)
»
Kahendpuu kp = new Kahendpuu(Puu.juhupuu(5, 2));»
kp.faili("fail.txt"); kahendpuu faili (tabelkujul)
Graaf.kuvada("fail.txt", true); tippudes märgendid
»»
println(" Alampuu");»
println("juur\tkõrgus");»
»
* i = 0 .. kp.n()-1»
»
Tipp t = kp.tipp(i); kp tipp indeksiga i
println(" " + t.märgend() + "\t " + h(kp, t));»
TEST (peameetod main)

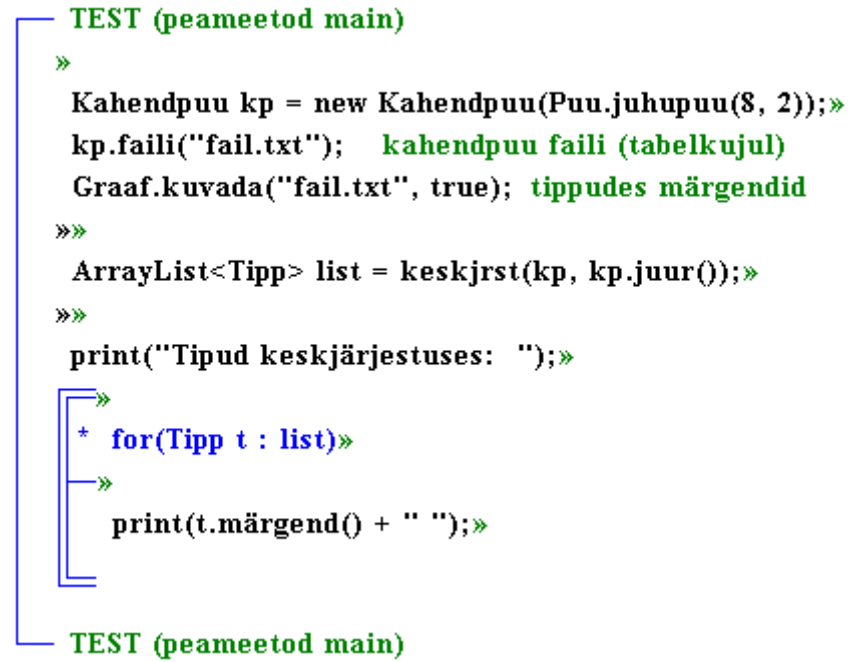
KP_3 skeem

```
»
 import ee.ut.kiho.aa.graaf.*;»
 import java.util.ArrayList;»
»»
 TEST (peameetod main)
»»
   »
    int h(Kahendpuu kp, Tipp t, ArrayList<Tipp> rikutud)»
   »
    » Antud: kahendpuu kp, selle tipp t (või null)  ja formeeritav list 'rikutud'
    » Tulemus: tipp t lisatakse listi 'rikutud', kui  selle alluvate kõrgused
    » .            erinevad rohkem kui 1 võrra;
    » .            tagastatakse tipust t algava (alam)kahendpuu kõrgus
    baasjuht
     int vasH = h(kp, kp.vasakAlluv(t), rikutud);»
     int parH = h(kp, kp.paremAlluv(t), rikutud);»
    tipp t lisada/mitte lisada listi 'rikutud'
   return Math.max(vasH, parH) + 1;  tagastada tipu t kõrgus
```

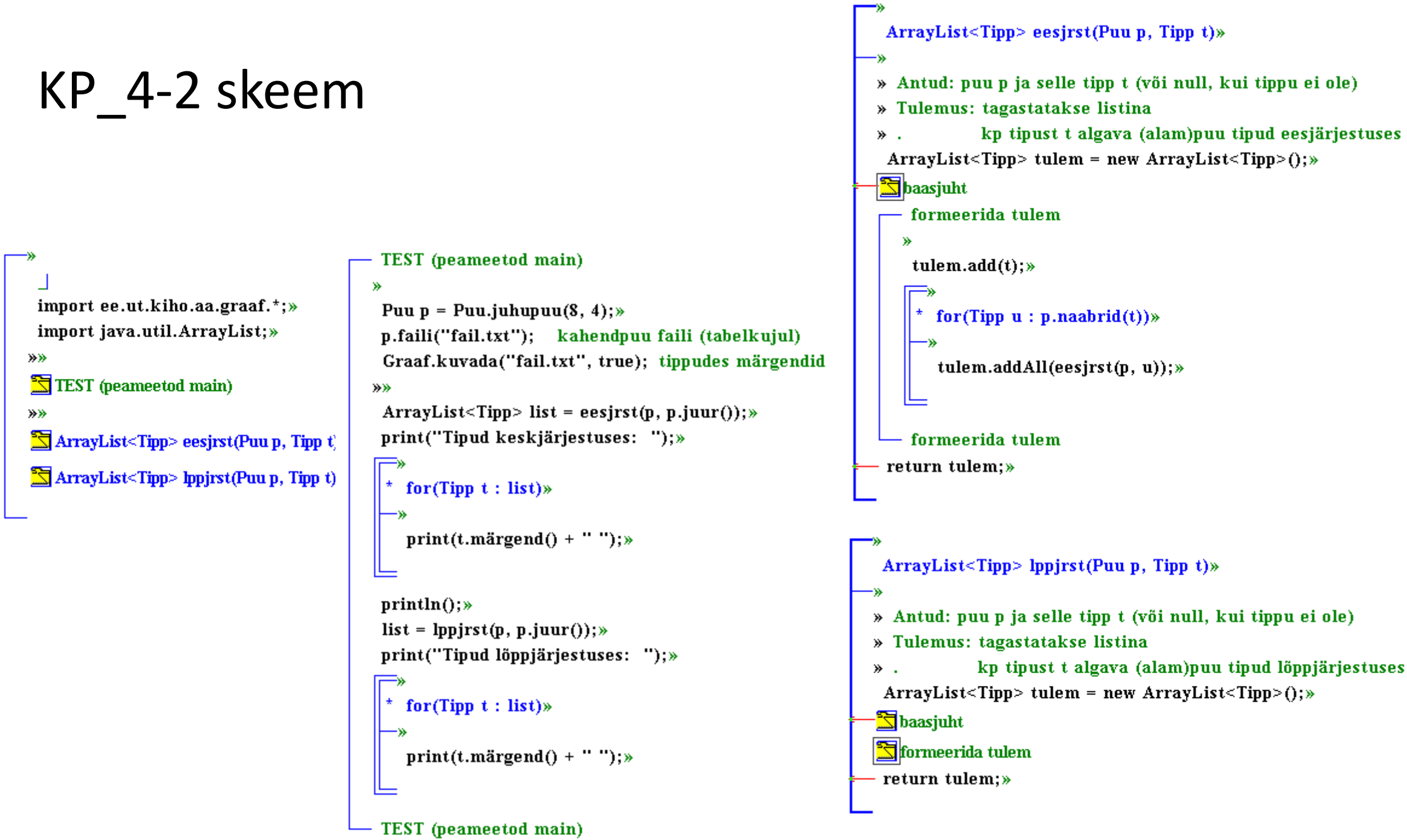
```
TEST (peameetod main)
»
 Kahendpuu kp = Kahendpuu.juhu2puu(5);»
 kp.faili("fail.txt");    kahendpuu faili (tabelkujul)
 Graaf.kuvada("fail.txt", false);  tippudes numbrid
»»
 ArrayList<Tipp> rikutud = new ArrayList<Tipp>();
 h(kp, kp.juur(), rikutud);»
 println("Tasakaal rikutud tippudes" );»
   »
  *  for(Tipp t : rikutud)»
   »
     println("nr " + (kp.indeks(t) + 1) );»

 println();»
TEST (peameetod main)
```

KP_4-1K skeem

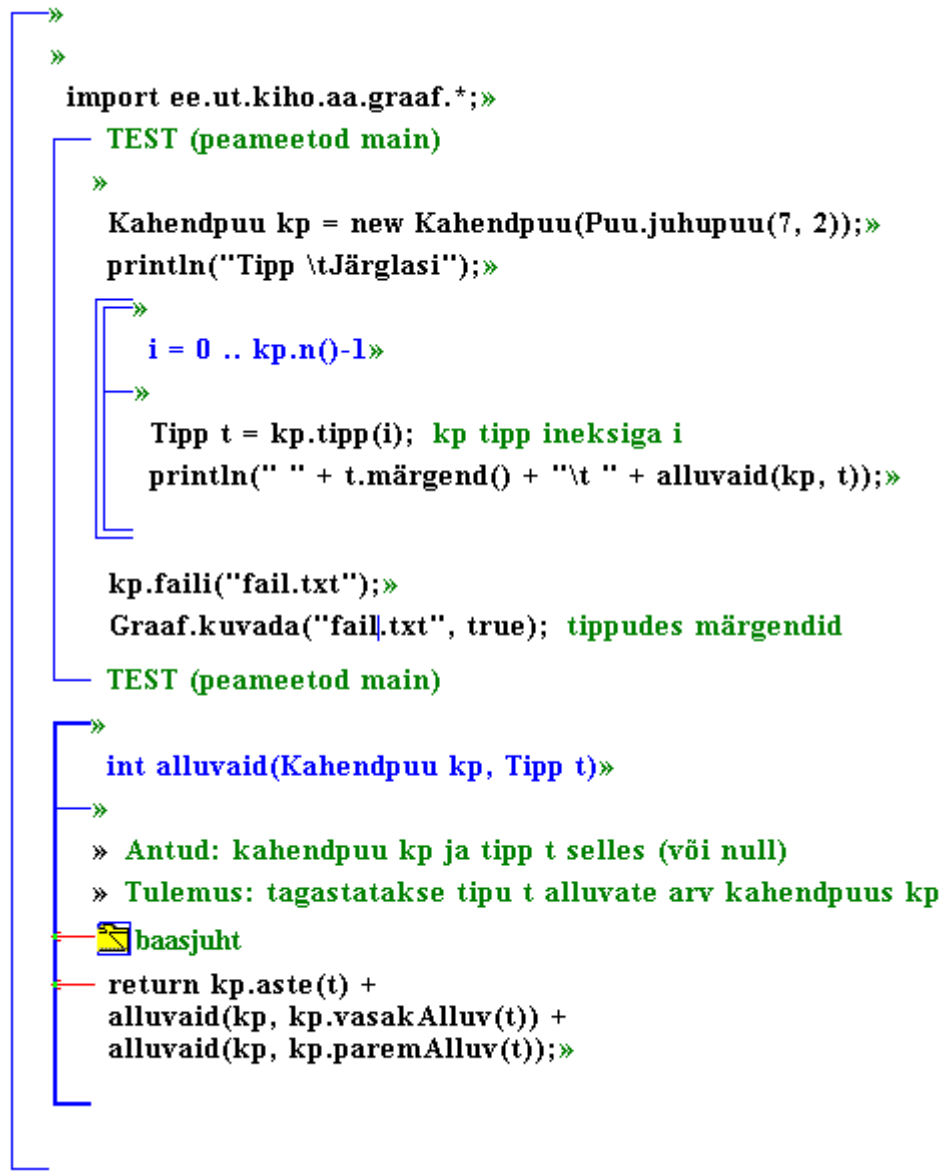

import ee.ut.kiho.aa.graaf.*;
import java.util.ArrayList;
TEST (peameetod main)
ArrayList<Tipp> keskjrst(Kahendpuu kp, Tipp t)
Vt ka ül. KP_4-1E
Antud: kahendpuu kp ja selle tipp t (või null, kui tippu ei ole)
Tulemus: tagastatakse listina
. kp tipust t algava (alam)kahendpuu tipud keskjärjestuses
ArrayList<Tipp> tulem = new ArrayList<Tipp>();
baasjuht
formeerida tulem
return tulem;


TEST (peameetod main)
Kahendpuu kp = new Kahendpuu(Puu.juhupuu(8, 2));
kp.faili("fail.txt"); kahendpuu faili (tabelkujul)
Graaf.kuvada("fail.txt", true); tippudes märgendid
ArrayList<Tipp> list = keskjrst(kp, kp.juur());
print("Tipud keskjärjestuses: ");
for(Tipp t : list)
print(t.märgend() + " ");
TEST (peameetod main)

KP_4-2 skeem


import ee.ut.kiho.aa.graaf.*;
import java.util.ArrayList;
TEST (peameetod main)
ArrayList<Tipp> eesjrst(Puu p, Tipp t)
ArrayList<Tipp> lppjrst(Puu p, Tipp t)
TEST (peameetod main)
Puu p = Puu.juhupuu(8, 4);
p.faili("fail.txt"); kahendpuu faili (tabelkujul)
Graaf.kuvada("fail.txt", true); tippudes märgendid
ArrayList<Tipp> list = eesjrst(p, p.juur());
print("Tipud keskjärjestuses: ");
for(Tipp t : list)
print(t.märgend() + " ");
println();
list = lppjrst(p, p.juur());
print("Tipud lõppjärjestuses: ");
for(Tipp t : list)
print(t.märgend() + " ");
TEST (peameetod main)
ArrayList<Tipp> eesjrst(Puu p, Tipp t)
Antud: puu p ja selle tipp t (või null, kui tippu ei ole)
Tulemus: tagastatakse listina
. kp tipust t algava (alam)puu tipud eesjärjestuses
ArrayList<Tipp> tulem = new ArrayList<Tipp>();
baasjuht
formeerida tulem
tulem.add(t);
for(Tipp u : p.naabrid(t))
tulem.addAll(eesjrst(p, u));
formeerida tulem
return tulem;
ArrayList<Tipp> lppjrst(Puu p, Tipp t)
Antud: puu p ja selle tipp t (või null, kui tippu ei ole)
Tulemus: tagastatakse listina
. kp tipust t algava (alam)puu tipud lõppjärjestuses
ArrayList<Tipp> tulem = new ArrayList<Tipp>();
baasjuht
formeerida tulem
return tulem;

KP_6 skeem


import ee.ut.kiho.aa.graaf.*;
TEST (peameetod main)
Kahendpuu kp = new Kahendpuu(Puu.juhupuu(7, 2));
println("Tipp \tJärglasi");
i = 0 .. kp.n()-1
Tipp t = kp.tipp(i); kp tipp ineksiga i
println(" " + t.märgend() + "\t " + alluvaid(kp, t));
kp.faili("fail.txt");
Graaf.kuvada("fail.txt", true); tippudes märgendid
TEST (peameetod main)
int alluvaid(Kahendpuu kp, Tipp t)
» Antud: kahendpuu kp ja tipp t selles (või null)
» Tulemus: tagastatakse tipu t alluvate arv kahendpuus kp
baasjuht
return kp.aste(t) +
alluvaid(kp, kp.vasakAlluv(t)) +
alluvaid(kp, kp.paremAlluv(t));

KP_7 skeem

```
»
 import ee.ut.kiho.aa.graaf.*;
»»
 TEST (peameetod main)
 int seada_kõrgusväljad(Kahendpuu kp, Tipp t)
 void seada_tasemeNr(Kahendpuu kp, Tipp t, int m)
```

```
TEST (peameetod main)
»
 Kahendpuu kp = new Kahendpuu(Puu.juhupuu(5, 2));»
 kp.faili("fail.txt");   kahendpuu faili (tabelkujul)
 Graaf.kuvada("fail.txt", true);  tippudes märgendid
»»
 seada_kõrgusväljad(kp, kp.juur());»
 kp.faili("failH.txt");   kahendpuu faili (tabelkujul)
 Graaf.kuvada("failH.txt", false);  tippudes numbrid
»»
 seada_tasemeNr(kp, kp.juur(), 1);»
 kp.faili("failHT.txt");   kahendpuu faili (tabelkujul)
 Graaf.kuvada("failHT.txt", false);  tippudes numbrid
TEST (peameetod main)
```

```
»
 int seada_kõrgusväljad(Kahendpuu kp, Tipp t)»
»
» Antud: kahendpuu kp ja selle tipp t (või null, kui tippu ei ole)
» Tulemus: tipust t algavas kp alampuus on iga tipu u väljale "h"
» .          omistatud tipust u algava alampuu kõrgus;
» .          tagastatakse tipust t algava (alam)kahendpuu kõrgus
  baasjuht
  if (t == null)»
»
  return 0;»
  baasjuht
 int hv = seada_kõrgusväljad(kp, kp.vasakAlluv(t));»
 int hp = seada_kõrgusväljad(kp, kp.paremAlluv(t));»
 int h = 1+Math.max(hv, hp);»
 t.seadaVäli("h", ""+h);»
 return h;»
```

```
»
 void seada_tasemeNr(Kahendpuu kp, Tipp t, int m)»
»
» Antud: kahendpuu kp, tipp t selles (või null, kui tippu ei ole)
» .        ja tipu t tasemenumber m
» Tulemus: tipust t algavas kp alampuus on
» .          iga tipu väljale "tase" omistatud tasemenumber,
» .          antud tipu t “tase” on m, selle alluvatel m+1, ...
  baasjuht
  if (t == null) »
»
  return; »
  baasjuht
 t.seadaVäli("tase", ""+m);»
 seada_tasemeNr(kp, kp.vasakAlluv(t), m+1);»
 seada_tasemeNr(kp, kp.paremAlluv(t), m+1);»
```

KOP_1. Kahendotsimispuu valmistamine

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud: arv n, $n > 0$.

Tulemus: n-tipuline juhu-kahendotsimispuu.

Meetodi võimalik skeem:

- Luua n-tipuline juhu-kahendpuu *kp*
- Luua n-elemendiline juhuarvude sorteeritud järjend *a* (näiteks juhuarvudest lõigult [1; 99])
- Omistada arvud järjendist *a* kahendpuu *kp* tippude märgenditeks tippude keskjärjekorras (vt ül KP_4-1K)

KOP_2. Kahendotsimispuu kontroll

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud: kahendpuu, milles tipumärgenditeks arvud.

Tulemus: kontrollitud, kas antud kahendpuu on kahendotsimispuu.

Vt [Õpik, teoreem lk 31]

KOP_3. AVL-puu kontroll

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud: kahendpuu.

Tulemus: kontrollitud, kas antud kahendpuu on AVL-puu.

KOP_3-1. Rikkekohad

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.

Antud:

- kahendpuu *kp,* milles iga tipuga on seotud kõrguseväli *h* (vt ül KP_7)
- *kp* tipp *t*
- tipu *t* ülemus (Λ, kui *t* on *kp* juurtipp) /Java: $\Lambda = \text{null}$ /

Tulemus: tagastatakse

kp tasakaalu rikkekohtade list paaridest <*v*, *y*>, kus

v on AVL-rikkega tipp, st *v* vasaku ja parema alampuu kõrgused erinevad rohkem kui ühe võrra

y on selle tipu (*v*) ülemus (või Λ , kui *v* on *kp* juurtipp)

Vihje: kahendpuu läbimine toimetada eesjärjestuses.

KOP_3 skeem:

- Omistada *kp* tippudele kõrgusevälјad (ül KP_7-1)
- Tulemus: *kp* on AVL-puu, kui *kp* on kahendotsimispuu (ül KOP_2) ja *kp* rikkekohtade list on tühi

KOP_3-2. Kõrgused ja AVL-rikkekohad

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.

Antud:

- kahendpuu *kp*
- *kp* tipp *t*
- tipu *t* ülemus (Λ, kui *t* on *kp* juurtipp) / Java: Λ = null /
- rikkekohtade list (kuhu lisada leitavaid rikkekohti), rikkekohaks on paar <rikkega tipp, selle ülemus(või Λ)>

Tulemus: tipust *t* algavas *kp* alampuus *alamkp*

- seatud kõikide tippude kõrguseväljad
- list täiendatud selles (*alamkp)* leiduvate rikkekohtadega;
- tagastatakse *alamkp* kõrgus

KOP_3-2 skeem

```
KOP_3-2 MALL
 int hJaRikkekohad(Kahendpuu kp, Tipp t, Tipp ülemus, ArrayList<Tipp[]> rikkekohad)»
»
» Antud: kahendotsimispuu kp,
» .        selle tipp t ja tipu t ülemus (null, kui t on kp juur)
» .       ja rikkekohtade list (kuhu lisada leitavaid rikkekohti <rikkega tipp, selle ülemus>)
» Tulemus: tipust t algavas kp alampuus (alamkp) seatud kõikide tippude kõrguseväljad
» .            ja rikkekohtade list täiendatud selles alampuus leiduvate rikkekohtadega;
» .            tagastatakse alamkp kõrgus
  »
   if (t == null) baasjuht
  »
   return ...;»

 Tipp tvas = kp.vasakAlluv(t);»
 Tipp tpar = kp.paremAlluv(t);»
» tipp t on ülemuseks tippudele tvas ja tpar
 int vas_h = hJaRikkekohad(kp, ..., ..., rikkekohad);»
 int par_h = hJaRikkekohad(kp, ..., ..., ...);»
  »
   if (Math.abs(vas_h - par_h) > 1)»
  »
   ...»
   rikkekohad.add(...);»

 int h = ...;  tipu t kõrgus
 t.seadaVäli("h", ""+h);»
 return ...;»
KOP_3-2 MALL
```

```
»
 _
preambul
 import ee.ut.kiho.aa.graaf.*;»
 import ee.ut.kiho.aa.util.*;  Juhu.järjend(a,b, n)
 import java.util.ArrayList;»
 import java.util.Arrays;»
»»
 KOP_3-2 TEST
  »
   Kahendpuu kop = tehaKOPjuhu(9);»
   kop.faili("kopyyy.txt");»
   Graaf.kuvada("kopyyy.txt", true);  tippudes märgendid
  »»
   ArrayList<Tipp[]> rikked = new ArrayList<Tipp[]>();»
   hJaRikkekohad(kop, kop.juur(), null, rikked);»
  »»
   println("Rikkekohad:");»
    »
   * for(Tipp[] rike : rikked)»
    »
     Tipp t = rike[0];»
     Tipp ülemus = rike[1];»
     println(t.nimi() + "    " + ((ülemus == null)? "null" : ülemus.nimi()));»

   kop.faili("kophhh");»
   Graaf.kuvada("kophhh", true);»
 KOP_3-2 TEST
int hJaRikkekohad(Kahendpuu kp, Tipp t, Tipp ülemus, ArrayList<Tipp[]> rikkekohad
Kahendpuu tehaKOPjuhu(int n)
ArrayList<Tipp> keskjrst(Kahendpuu kp, Tipp t)
```

KOP_4. Otsimine kahendotsimispuus

Kirjutada ja testida rekursiivne meetod järgmise ülesande lahendamiseks.

Antud: *kop* - mittetühi kahendotsimise puu, *t* - selle tipp ja arv *k*, mille kohta (ja ülemust) otsida alates tipust *t.*

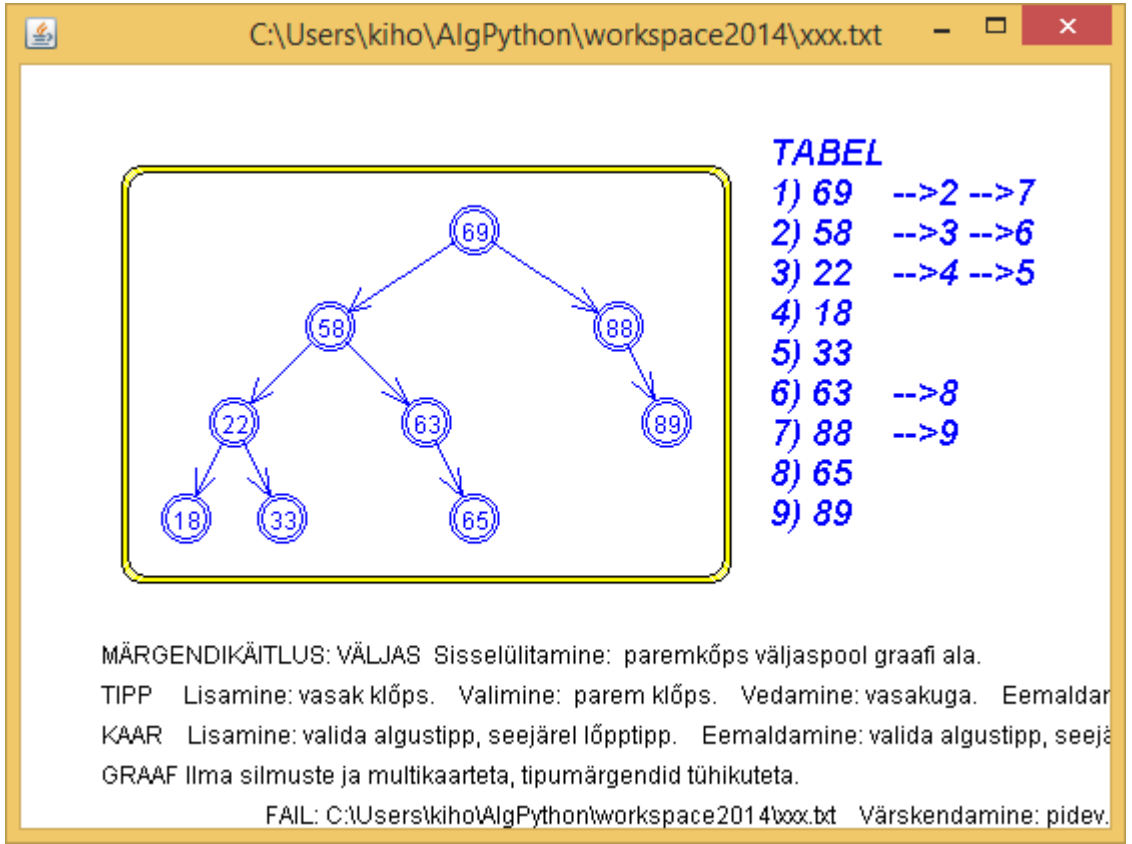
Tulemus: tagastatakse paar <*t*1, *t*2>, kus

*t*1 on tipp, milles arv *k* leiti (või Λ, kui arvu ei leitud)

*t*2 on *t*1 ülemus (või Λ, kui *t*1 on juurtipp);

juhul kui arvu ei leitud (*t*1 = Λ), siis *t*2 on tipp, millele arv *k* sobiks alluvaks.

Testi tulemuse näide




```
Otsitav arv: 63
Leitud arv ja selle ülemus: 63 58

Otsitav arv: 75
Otsitavat arvu ei ole, koha ülemus: 88

Otsitav arv: 69
Otsitav arv leitud juurtipus (ülemus: null)
```

KOP_5. Lisamine kahendotsimispuusse

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud: kahendotsimispuu *kop* (arvud-võtmed unikaalsed) ja lisatav arv *k*.

Tulemus: tipp arvuga *k* lisatud antud kahendotsimispuusse;
kui arv *k* juba leidus, siis ei tehta midagi.

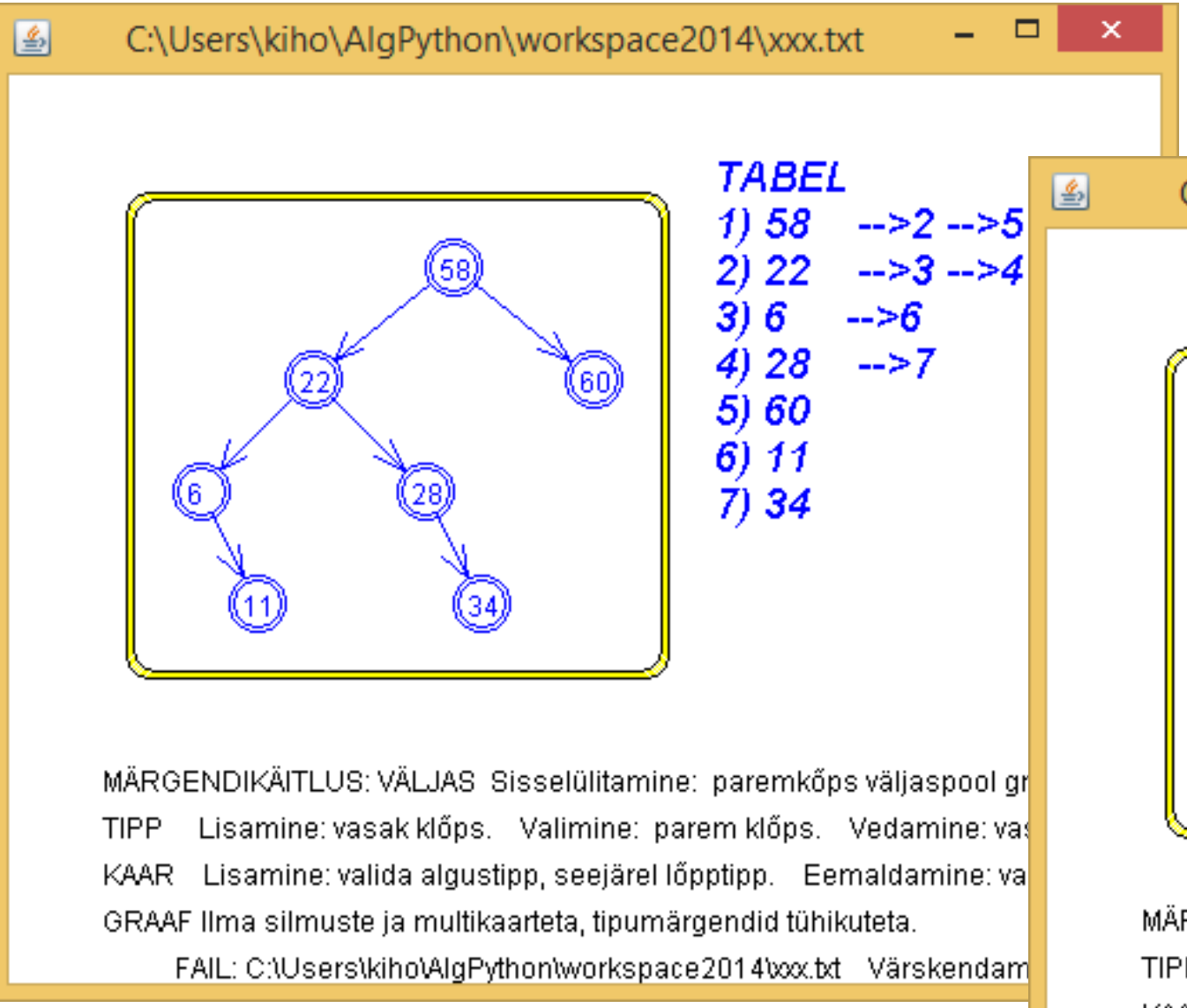
Java

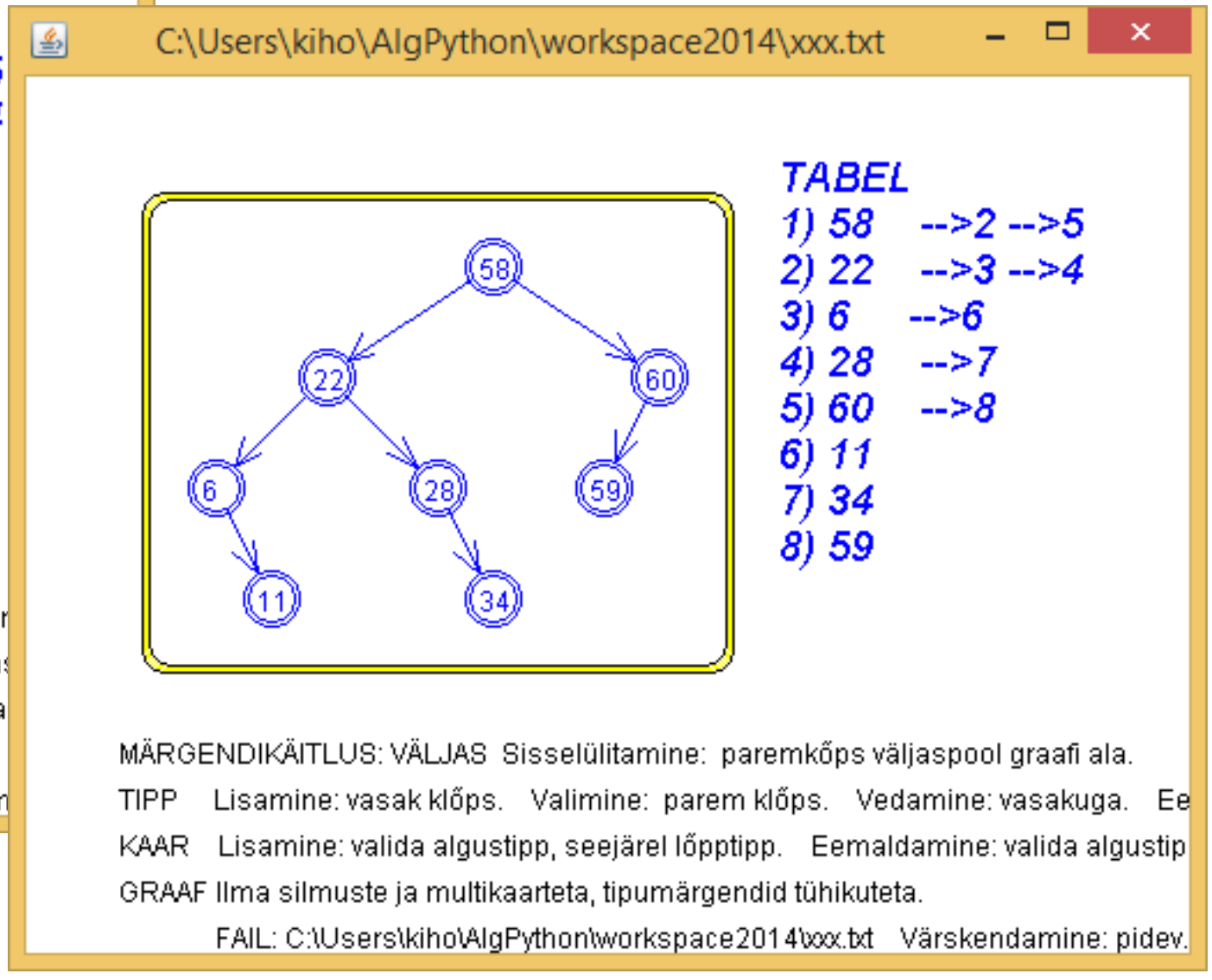
```
Tipp t = new Tipp(k + "[0;0]");
```

-- tehtud uus tipp *t* märgendiga *k* ja (suva)koordinaatidega $x = 0$; $y = 0$

Testi tulemuse näide

```
Lisatav arv: 59
On KOP = true
```





KOP_6. Eemaldamine kahendotsimispuust

KOP_6-1. Tipu eemaldamine [Õpik, lk 31]

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.
Antud: *kop* - kahendotsimispuu, *t* - selle tipp, *y* - tipu t ülemus (Λ, kui *t* on *kop* juurtipp).
Tulemus: tipp *t* eemaldatud kahendotsimispuust *kop.*

KOP_6-2. Arvu eemaldamine

Kirjutada ja testida funktsioon järgmise ülesande lahendamiseks.
Antud: *kop* - kahendotsimispuu, *k* – otsitav arv.
Tulemus: tipp arvuga *k* (kui leidus) eemaldatud kahendotsimispuust *kop.*

KOP_7. AVL-puu tasakaalustamise võte

Kirjutada ja testida meetod järgmise ülesande lahendamiseks.

Antud**:**

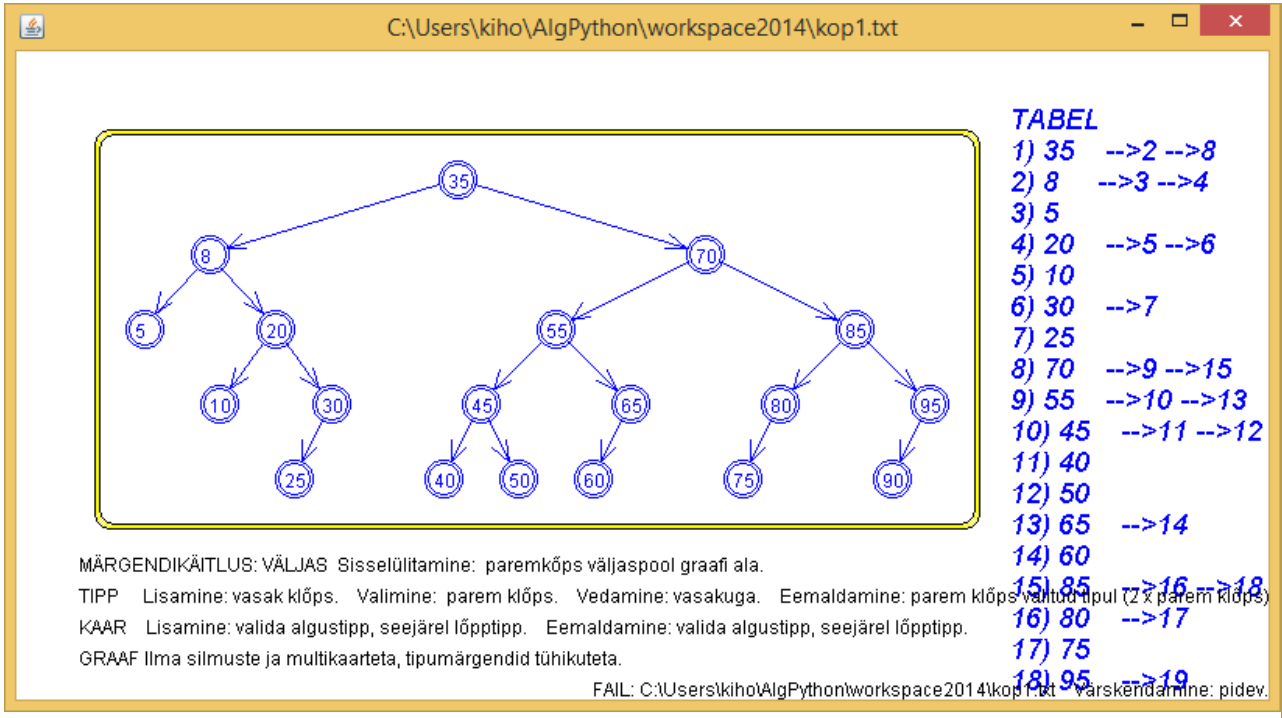
- kahendotsimispuu *kop* AVL-rikkega ühes tipus, nagu [Kiho 2003, Joonis 2.4 või selle peegeldus]
- kolm tippu – rikkega tipp *a* (allaviidav), selle alluv *b* (ülesviidav), tipu a ülemus *y* (Λ, kui *a* on *kop* juurtipp).

Tulemus: antud kahendotsimispuu *kop* tipus *a* sooritatud tasakaalustamise võte.

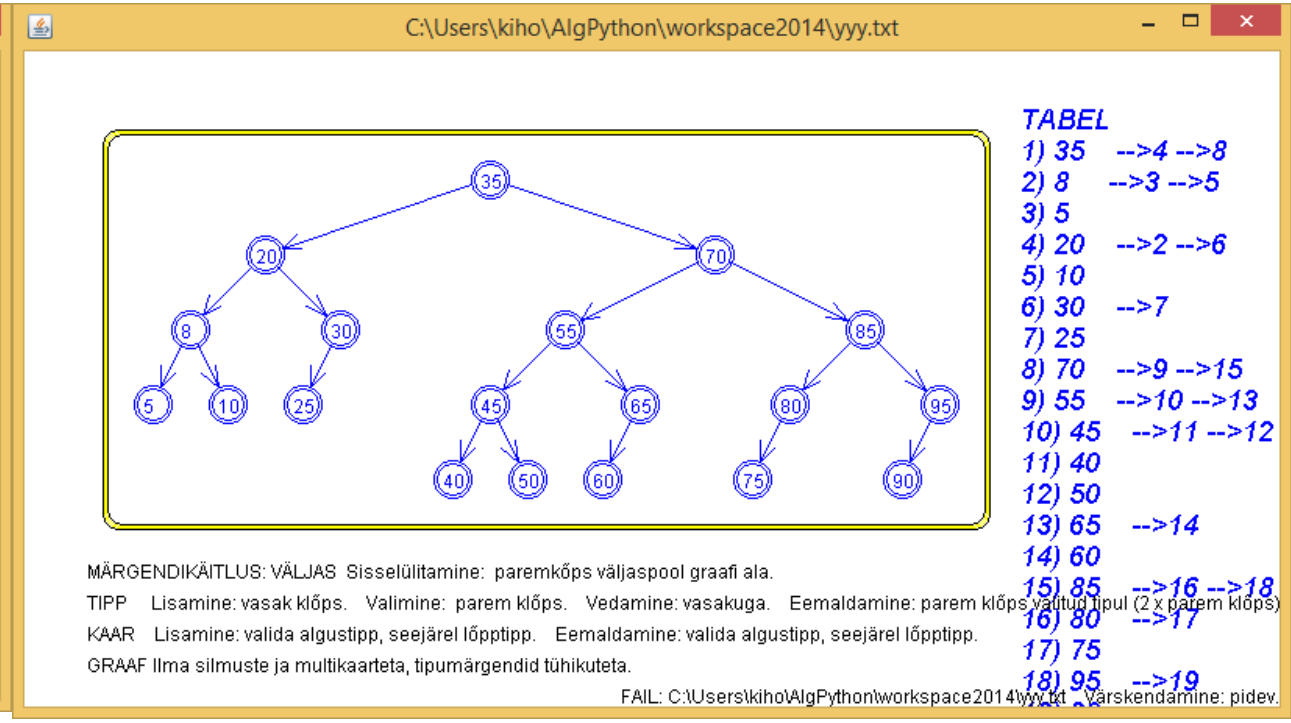
Testi tulemuse näide

```
AVL-rikkega tipu (a) nr: 2
selle alluva tipu (b) nr: 4
tipu a ülemuse nr: 1
```

Antud



Tasakaalustatud



Fail Toimeti Lisada Minna Teisendus Raam Erisoovid Abi

projekt: AlgJavaWorks2015oCDB

```
»
Kahendpuu alla_üles(Kahendpuu kop, Tipp v, Tipp w, Tipp vy) sooritada tasakaalustamise võte
»
spetsifikatsioon
boolean juurtipus = (vy==null);»
Tipp keskmine;»
»
if (kop.paremAlluv(v) == w)  Joonis 2.4
»
keskmine = kop.vasakAlluv(w);  II
»
else  Joonis 2.4 peegel
keskmine = kop.paremAlluv(w);»

» korraldada ümber möned kaared:
»
»
Kaar k;»
if (! juurtipus)
k = kop.kaar(v, w);»
k.seadaLõpptipp(v);»
k.seadaAlgustipp(w);»
»
if (keskmine != null)»
»
k = kop.kaar(w, keskmine);»
k.seadaAlgustipp(v);»
» seada "keskmine" v ja w vahele (topeltvõtte puhul vajalik):
nihutadaH(kop, keskmine, (v.x()+w.x())/2 - keskmine.x());»

»
if (juurtipus)»
»
» w jääb juureks, tuua esikohale tippude seas:
kop.esikohale(w);»

return kop;»

void nihutadaH(Kahendpuu kp, Tipp t, int xnihe)
```

Amadeus - <noname59>*

Fail Toimeti Lisada Minna Teisendus Raam Erisoovid Abi

```
spetsifikatsioon
»
» Antud: kop - kahendotsimispuu,
» .        tipp v - allaviidav,
» .        tipp w - ülesviidav,
» .        vy - tipu v ülemus või null , kui kui v on kop juurtipp.
» .        Joonistel [Kiho 2003, joonis 2.4 või selle peegeldus] ja
» .                  [Kiho 2003, joonis 2.5 või selle peegeldus]
» .                  tähistab allaviidavat tippu alla-nooleke ja
» .                  ülesviidavat tippu üles-nooleke.
» Tulemus: puus kop sooritatud tasakaalustamise võte,
» .          vt nt [Kiho 2003, joonis 2.4]
» .          tagasatakse (muudetud) kop
spetsifikatsioon
```

Amadeus - <noname60>*

Fail Toimeti Lisada Minna Teisendus Raam Erisoovid Abi

```
»
void nihutadaH(Kahendpuu kp, Tipp t, int xnihe)  abiks
»
» Antud: kop - kahendpuu, t - selle tipp,
» .        xnihe - nihke suurus;
» Tulemus: antud kp tipust t algava alampuu tippude
» .          x-koordinaadile liidetud xnihe
»
if (t != null)»
»
Tipp vas = kp.vasakAlluv(t);»
Tipp par = kp.paremAlluv(t);»
nihutadaH(kp, vas, xnihe);»
nihutadaH(kp, par, xnihe);»
t.seadaKoordinaadid(t.x()+xnihe, t.y());»
```