AVL-puu on kahendotsimise puu, kus iga tipu vasaku ja parema alampuu kõrgus erineb ülimalt ühe võrra.

AVL-puu kõrgus on alati $\Theta(\log n)$, kus n on tippude arv.

AVL-puus on tippudel täiendav väli $.kõrgus$ (ei kuulu ossa „Andmed“), kuhu salvestatakse sellest tipust algava alampuu kõrgus.

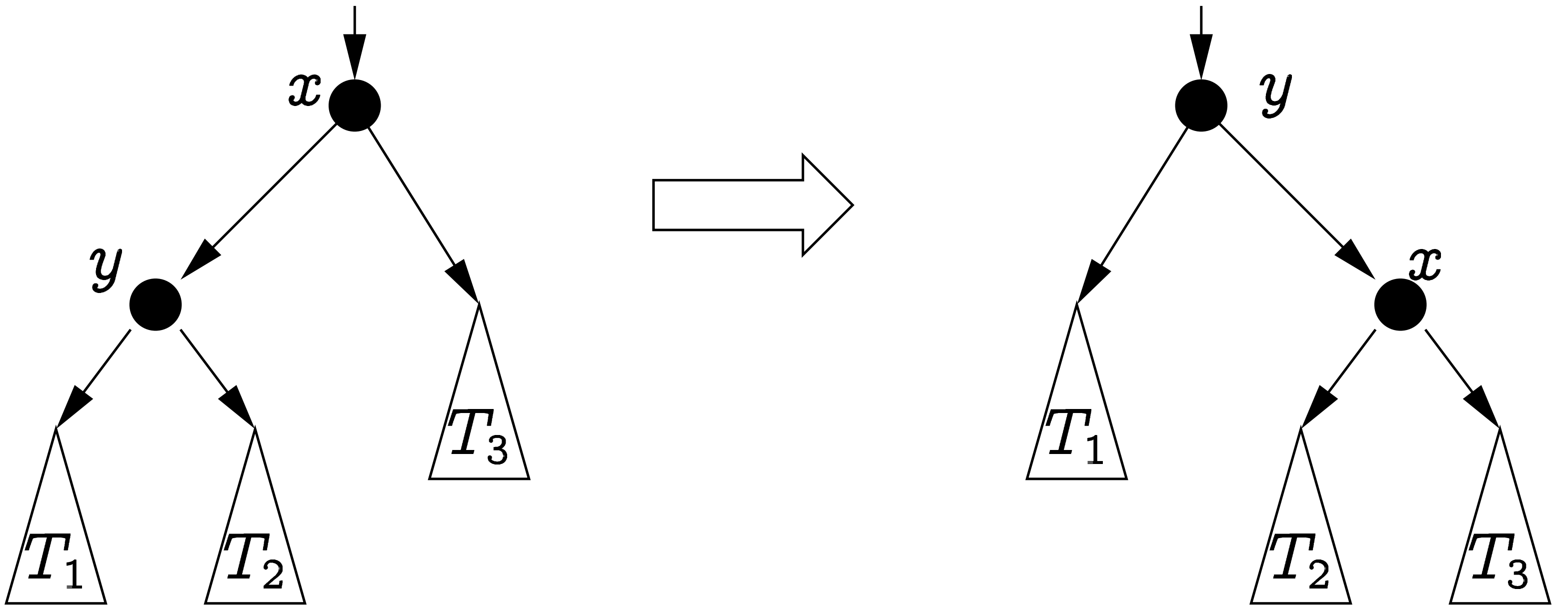
Peale tipu lisamist või eemaldamist tuleb nende väljade väärtusi parandada.

Peale eemaldamist võivad valed kõrgused olla tippudel, mis asuvad ahelal eemaldatud tipu (endisest) ülemusest juureni.

Kõrguste parandamine toimub samaaegselt puu tasakaalustamisega.

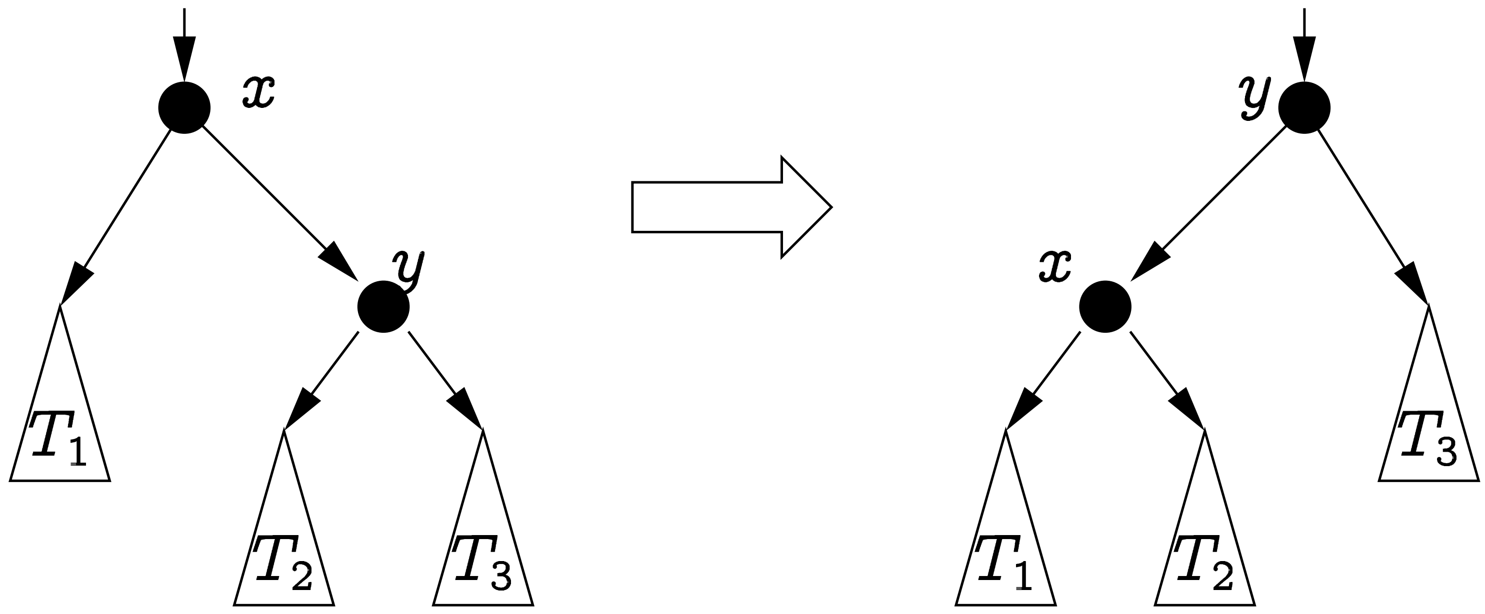
Tasakaalustamisoperatsioonid: pööre_vasakule ja pööre_paremale.

pööre_paremale(p, x), kus p on viit puu juurtipule ja x viit mingile tipule, mille korral $x.vasak \neq$ NIL, muudab alampuud, mille juureks on x, järgmiselt:



pööre_paremale tagastab muudetud puu juurtipu.

pööre_vasakule on vastupidine operatsioon.



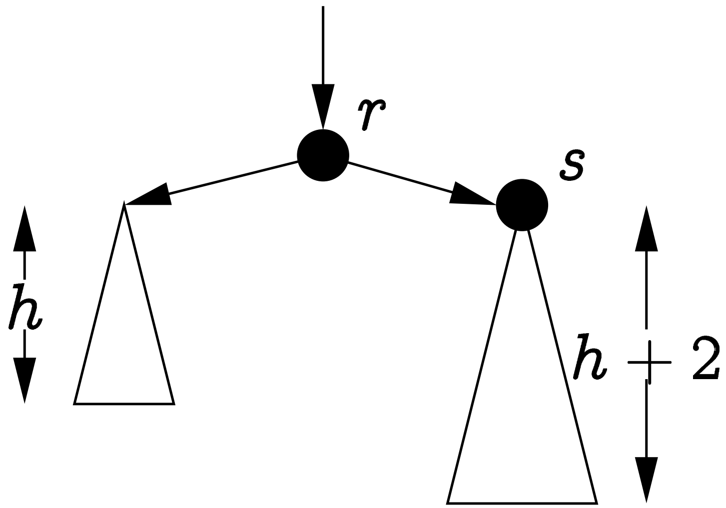
pööre_vasakule tagastab muudetud puu juurtipu.

Tipu eemaldamisel AVL-puust teeme kõigepealt tavalise eemaldamise, seejärel teeme tasakaalustamisi teel eemaldatud tipust juureni.

eemaldaAVL(p, q) on

$(p, q) := \text{eemalda}(p, q)$
$r := q.\textit{ülem}$
while $r \neq \text{NIL}$ **do**
    $h_v := r.vasak = \text{NIL}\ ?\ 0 : r.vasak.k\tilde{o}rgus$
    $h_p := r.parem = \text{NIL}\ ?\ 0 : r.parem.k\tilde{o}rgus$
    if $|h_v - h_p| = 2$ **then break**
    $r.k\tilde{o}rgus := \max(h_v, h_p) + 1$
    $r := r.\textit{ülem}$
if $r = \text{NIL}$ **then return** (p, q)
$\ldots$

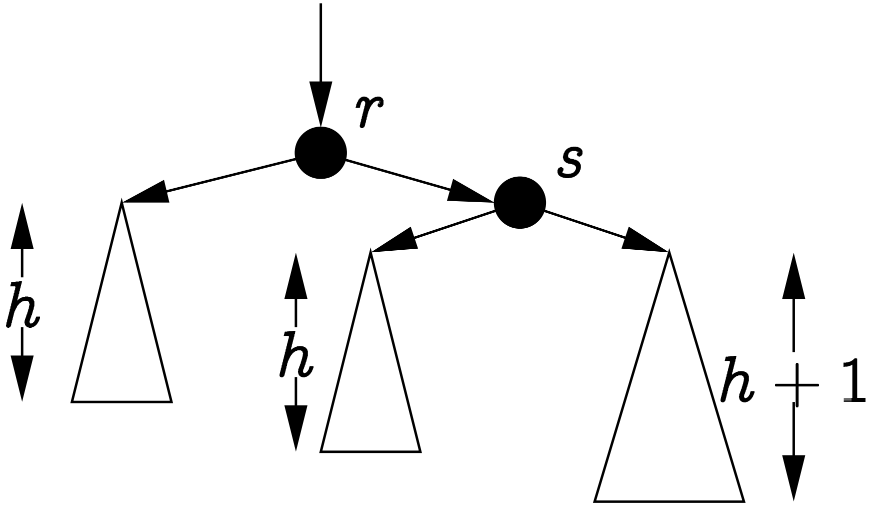
r viitab tasakaalustamata tipule. Vaatame juhtu, kus tema parempoolne alampuu on kõrgem kui vasakpoolne (teistpidine juht on sümmeetriline).



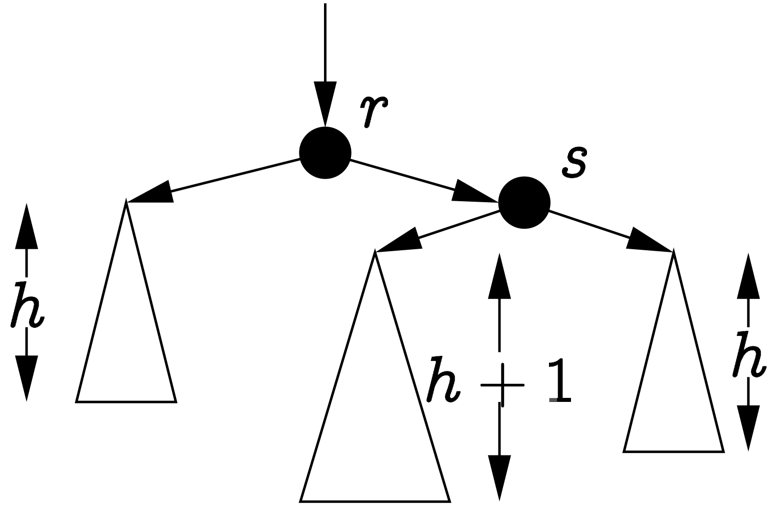
r-i kõrgus: $h + 3$
(enne eemaldamist: $h + 3$)

if $h_p > h_v$ **then**
    $s := r.parem$
    $\ldots$

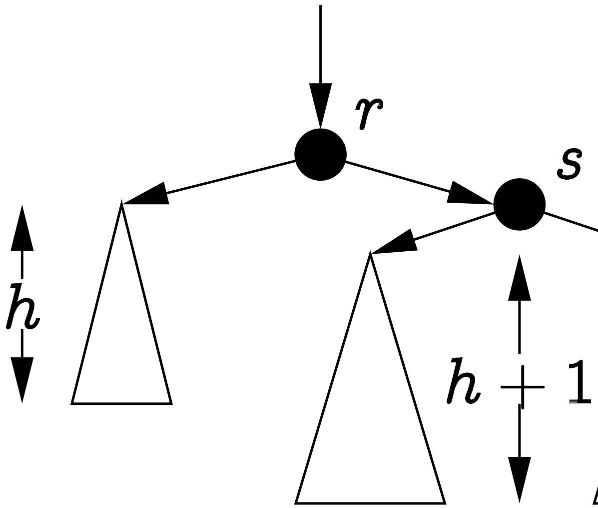
Tipp eemaldati vasakust alampuust. Kolm varianti:



1. variant



2. variant



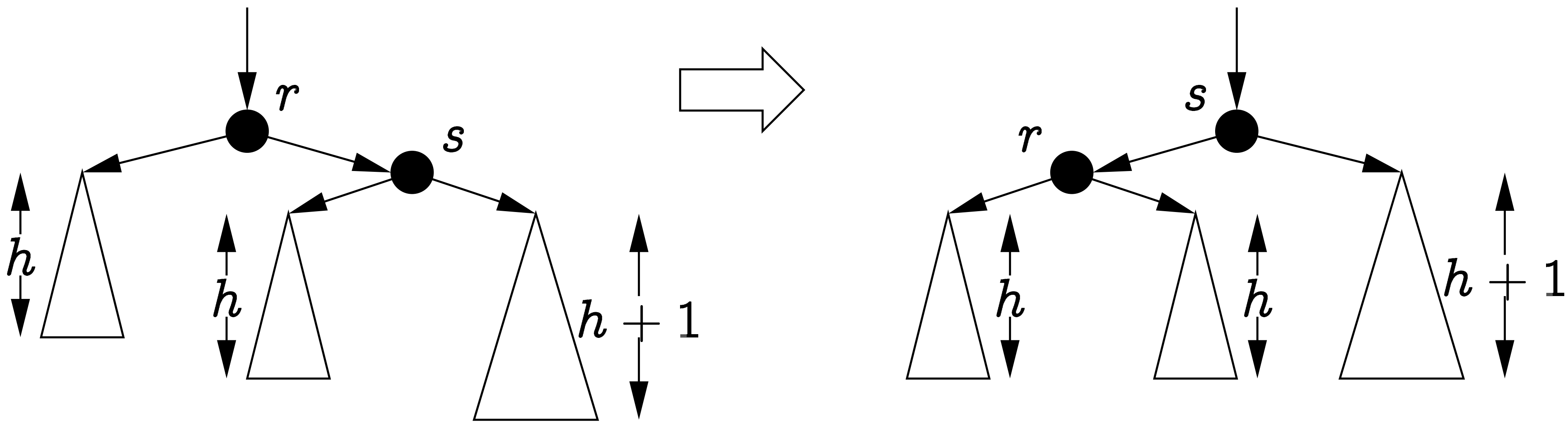
3. variant

$h_{sv} := s.vasak = \mathrm{NIL}\ ?\ 0 : s.vasak.kõrgus$

$h_{sp} := s.parem = \mathrm{NIL}\ ?\ 0 : s.parem.kõrgus$

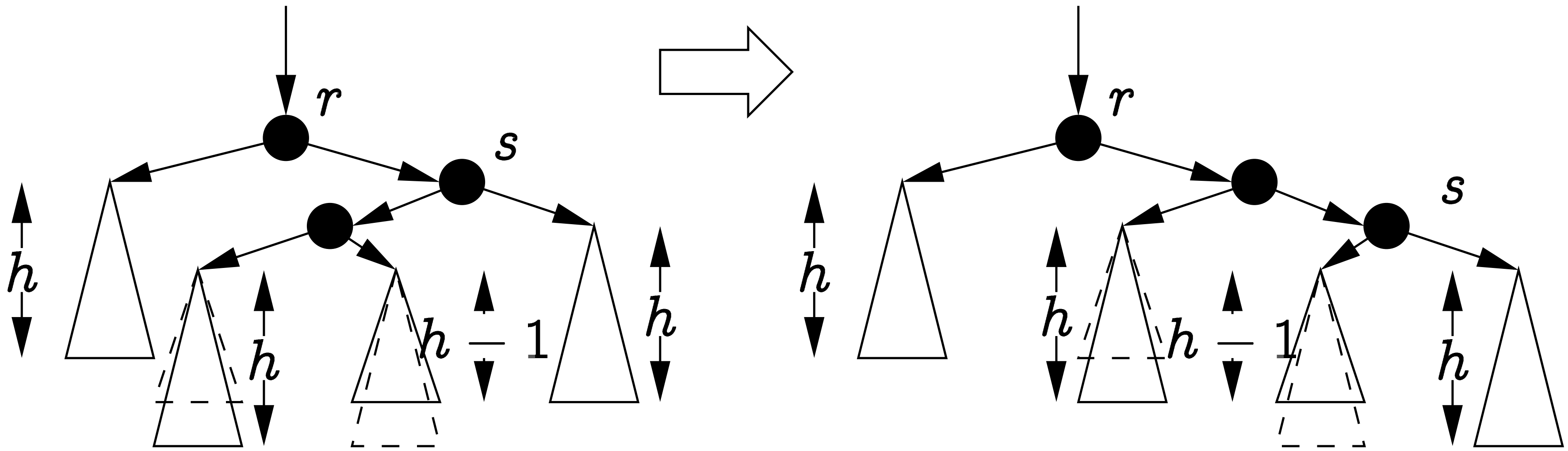
...

1. variandil pöörame r-i vasakule.

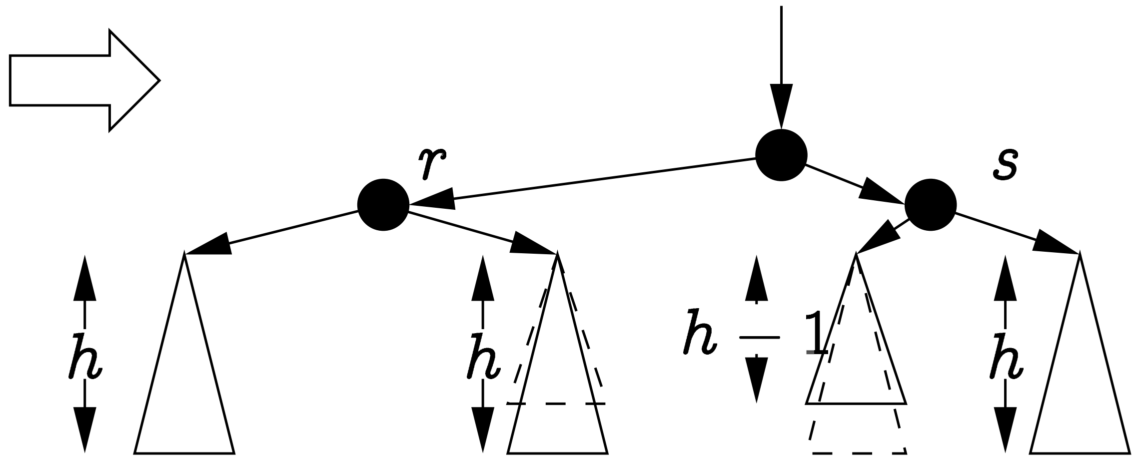


Kogu vaadeldava alampuu kõrgus on nüüd $h+2$ — vähenes ühe võrra. Seega peame me jätkame juure pool asuvate tippude tasakaalustamisi.

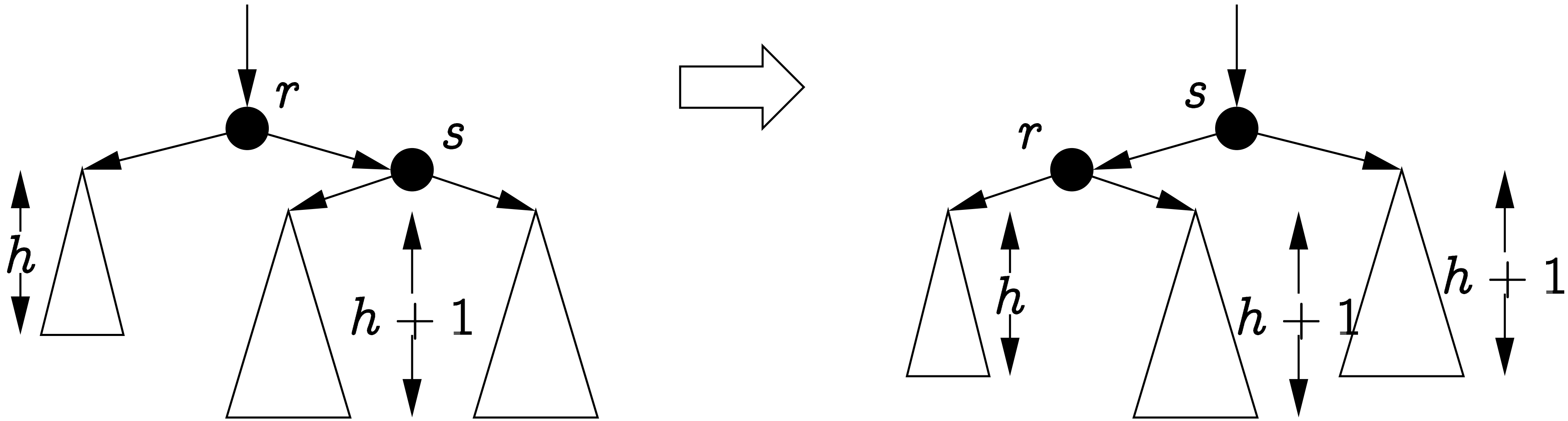
2. variandil pöörame s-i paremale



saame 1. variandi (s.t. pöörame r-i vasakule).



3. variandil pöörame r-i vasakule.



Kogu vaadeldava alampuu kõrgus on nüüd $h + 3$ — sama, mis enne eemaldamist. Seega on nüüd juurepoolsed tipud tasakaalus.

if $h_{sv} = h_{sp}$ **then**
    $p :=$ pööra_vasakule(p, r)
    $r.kõrgus := h_v + 2$; $s.kõrgus := h_v + 3$
    return (p, q)
if $h_{sv} > h_{sp}$ **then**
    $p :=$ pööra_paremale(p, s); $s.kõrgus := h_v + 1$
$p :=$ pööra_vasakule(p, r)
$r.kõrgus := h_v + 1$; $r.ülem.kõrgus := h_v + 2$
$r := r.ülem.ülem$; **goto** 3

$\ldots$

23 **else** --- $h_v > h_p$
24 $s := r.vasak$
25 $h_{sv} := s.vasak = \mathrm{NIL}\ ?\ 0 : s.vasak.k\tilde{o}rgus$
26 $h_{sp} := s.parem = \mathrm{NIL}\ ?\ 0 : s.parem.k\tilde{o}rgus$
27 **if** $h_{sv} = h_{sp}$ **then**
28 $p := \mathrm{p\ddot{o}\ddot{o}ra\_paremale}(p, r)$
29 $r.k\tilde{o}rgus := h_p + 2$; $s.k\tilde{o}rgus := h_p + 3$
30 **return** (p, q)
31 **if** $h_{sv} < h_{sp}$ **then**
32 $p := \mathrm{p\ddot{o}\ddot{o}ra\_vasakule}(p, s)$; $s.k\tilde{o}rgus := h_p + 1$
33 $p := \mathrm{p\ddot{o}\ddot{o}ra\_paremale}(p, r)$
34 $r.k\tilde{o}rgus := h_p + 1$; $r.\ddot{u}lem.k\tilde{o}rgus := h_p + 2$
35 $r := r.\ddot{u}lem.\ddot{u}lem$; **goto** 3

Puus *kehtib kuhjaomadus*, kui ühegi tipu võti pole suurem kui tema ülemuse võti.

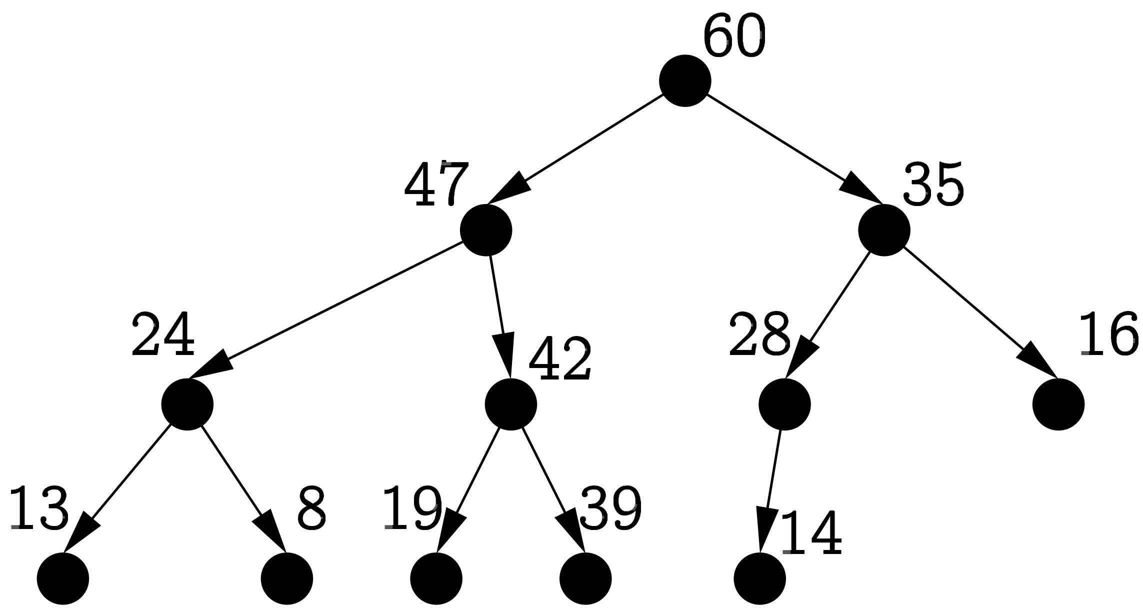
Kahendpuu on *täielik*, kui kõik tema lehed on juurest sama kaugel ja kõigi sisemiste tippude aste on 2.

Täielikus kahendpuus kõrgusega h on $2^h - 1$ tippu.

Kahendpuu on *kompaktne*, kui ta on saadud täielikust kahendpuust sealt nulli või enama parempoolseima lehe kustutamisel.

Kahendkuhi on kompaktne kahendpuu, kus kehtib kuhjaomadus.

Kahendkuhja võimalik esitus: massiivis, puu tasemete kaupa, juurest alates.



Esitus massiivina:

60	47	35	24	42	28	16	13	8	19	39	14

Kahendkuhi K (suurusega n) toetab järgmisi operatsioone:

tee_kuhi	$\Theta(1)$
lisa(K, x)	$\Theta(\log n)$
leia_max(K)	$\Theta(1)$
võta_max(K)	$\Theta(\log n)$
suurenda_võti$(K, \text{tipp}, \text{uus_võti})$	$\Theta(\log n)$
vähenda_võti$(K, \text{tipp}, \text{uus_võti})$	$\Theta(\log n)$
kustuta(K, tipp)	$\Theta(\log n)$

Realisatsioon — vaata Kiho konspektist.

Realisatsioon vajab tabelit a, millel on väli $.n$ — elementide arv tabelis.

Tabel peab toetama järgmisi operatsioone, kõiki konstantse ajaga:

- tee_tabel — loob uue tabeli, milles elemente pole.
- suurenda_tabel(a, x) — lisab tabeli lõppu uue välja, mille väärtuseks võtab x.
- vähenda_tabel(a) — kustutab tabeli lõpust ühe välja.
- muuda_element(a, i, x) — võtab $a[i]$ väärtuseks x.

Massiiv toetab kõiki neid operatsioone (välja $.n$ tuleb kuskil eraldi hoida), v.a. suurendamine juhul, kui elementide arv on võrdne massiivi pikkusega $a.maxn$.

Kui me maksimaalset elementide arvu ette teame, siis saame protseduuris tee_tabel piisava suurusega massiivi luua.

Kui ei tea, siis loome juhul, kui massiiv on täis saanud, uue, suurema massiivi ja kopeerime kõik sinna ümber...

Sel juhul võib tee_tabel lihtsalt
üheelemendilise massiivi teha.

suurenda_tabel(a, x) on

```
if a.n = a.maxn then
    b := new(kirjetüüp[2 * a.maxn])
    b.maxn := 2 * a.maxn
    for i := 1 to a.maxn do
        b[i] := a[i]
    b.n := a.n
    free(a)
    a := b
a.n := a.n + 1
a[a.n] := x
return a
```

Aga see ei ole konstantse keerukusega.

Siiski, kui palju aega võtab n suurenda_tabel-i väljakutset (samal tabelil)?

Olgu c_i suurenda_tabel-i jooksuaeg, kui esilgse tabeli suurus on i. Siis

$$c_i \leqslant \begin{cases} i & \text{kui } i \text{ on kahe aste} \\ 1 & \text{muidu,} \end{cases}$$

kui võtame ajaühiku piisavalt suure.

n väljakutse aeg:

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=1}^{\lfloor \log n \rfloor} 2^j < n + 2 \cdot 2^{\lfloor \log n \rfloor} \leqslant n + 2n = 3n \quad .$$

Ühe väljakutse aeg — <u>keskmiselt</u> ülimalt 3 ajaühikut.

Samuti võiks tabelit vähendades massiivi vähese täituvuse korral elemendid väiksemasse massiivi ümber kopeerida.

vähenda_tabel(a) on

```
a.n := a.n − 1
if a.n ⩽ a.maxn/4 then
    b := new(kirjetüüp[a.maxn/2])
    b.maxn := a.maxn/2
    for i := 1 to a.n do
        b[i] := a[i]
    b.n := a.n
    free(a)
    a := b
return a
```

Üldiselt:

Olgu meil mingi andmestruktuur, mis toetab operatsioone $m_1, \ldots, m_k$ (igal neist operatsioonidest võivad olla ka mingid argumendid).

Olgu M kõigi lubatud operatsioonijadadade hulk (alustades loomata andmestruktuurist). S.t. mingi $\vec{m} \in M$ on kujul $m_{i_1}(\ldots)m_{i_2}(\ldots)\cdots m_{i_{|\vec{m}|}}(\ldots)$ ning m_1 peab olema andmestruktuuri loomine.

Kui $\vec{m} \in M$ ja $1 \leqslant j \leqslant \vec{m}$, siis olgu $[\vec{m}]_j$ andmestruktuuri(de) (kogu)suurus peale $\vec{m}$ esimese j operatsiooni rakendamist.

Olgu $f_1, \ldots, f_k$ naturaalarvudel määratud positiivsete väärtustega funktsioonid.

Ütleme, et operatsioonide $m_1, \ldots, m_k$ *amortiseeritud ajalised keerukused* on $O(f_1), \ldots, O(f_k)$, kui

- leiduvad $c_1, \ldots, c_k > 0$ nii, et
- iga $\vec{m} \in M$ jaoks
  - kus $\vec{m} = m_{i_1}(\ldots)m_{i_2}(\ldots)\cdots m_{i_{|\vec{m}|}}(\ldots)$

  vajab $\vec{m}$-i operatsioonide rakendamine ülimalt

$$\sum_{j=1}^{|\vec{m}|} c_{i_j} f_{i_j}([\vec{m}]_j)$$

  sammu.

Seega võivad hilisemad operatsioonid ära kasutada aja, mis varasematest operatsioonidest üle jäi.

Vastupidi (aega võlgu võtta) ei või, sest mingi lubatud operatsioonide jada iga prefiks on ka lubatud operatsioonide jada.

Operatsioonide amortiseeritud ajalist keerukust võib leida näiteks:

- Otseselt kokku lugedes, kui palju arvutussamme operatsioonide jadad teevad.
- *Potentsiaalide* meetodil.

Näitame, et suvaline (lubatud) n-elemendiline suurenda_tabel-i ja vähenda_tabel-i väljakutsete jada vajab $O(n)$ aega, s.t. üks operatsioon vajab keskmiselt $O(1)$ aega.

Kui a on tabel, siis olgu

$$\alpha(a) := a.n/a.maxn \qquad \text{(täituvus)}$$

$$\Phi(a) := \begin{cases} 2 \cdot a.n - a.maxn & \text{kui } \alpha(a) \geqslant 1/2 \\ a.maxn/2 - a.n & \text{kui } \alpha(a) \leqslant 1/2 \end{cases} \qquad \text{(potentsiaal)}$$

Tabelit muutva operatsiooni amortiseeritud ajaliseks keerukuseks võime võtta tema tegeliku ajalise keerukuse, millele on liidetud tulemustabeli ja esialgse tabeli potentsiaalide vahe.

Kui $\Phi(a) = 0$, siis a-le rakendatud operatsioonide jada tegelik keerukus on tõkestatud selle operatsioonijada amortiseeritud keerukusega.

Eelmine väide järeldub sellest, et potentsiaal on alati mittenegatiivne.

suurenda_tabel(a, x) amortiseeritud keerukus on ülimalt

- Kui $\alpha(a) < 1/2$, siis

  $$1 + (a.maxn/2 - (a.n + 1)) - (a.maxn/2 - a.n) = 0 \ .$$

- Kui $\alpha(a) \geqslant 1/2$, kuid $a.n < a.maxn$, siis

  $$1 + (2 \cdot (a.n + 1) - a.maxn) - (2 \cdot a.n - a.maxn) = 3 \ .$$

- Kui $a.n = a.maxn$, siis

  $$a.n + (2 \cdot (a.n + 1) - 2 \cdot a.maxn) - (2 \cdot a.n - a.maxn) =$$
  $$a.n + (2 \cdot a.n + 2 - 2 \cdot a.n) - (2 \cdot a.n - a.n) = 2 \ .$$

Kokkuvõttes seega $O(1)$.

vähenda_tabel(a) amortiseeritud keerukus on ülimalt

- Kui $a.n \leqslant a.maxn/4$, siis

$$a.n+(a.maxn/(2\cdot 2)-(a.n-1))-(a.maxn/2-a.n) = a.n - a.maxn/4 + 1 \leqslant 1 \ .$$

- Kui $1/4 < \alpha(a) \leqslant 1/2$, siis

$$1 + (a.maxn/2 - (a.n - 1)) - (a.maxn/2 - a.n) = 2 \ .$$

- Kui $\alpha(a) > 1/2$, siis

$$1 + (2 \cdot (a.n - 1) - a.maxn) - (2 \cdot a.n - a.maxn) = -1 \ .$$

Kokkuvõttes seega $O(1)$.

Kiho konspektis joonisel 3.11 on toodud algoritm kompaktse kahendpuu teisendamiseks kahendkuhjaks. Algoritmiks on

1. Tee vasakust alampuust kahendkuhi.
2. Tee paremast alampuust kahendkuhi.
3. Vii juurtipp allapoole.

Näitame, et see algoritm töötab ajas $O(n)$.

Allapoole viimist kutsutakse (otse algoritmist) välja iga tipu jaoks üks kord.

viia-alla väljakutsete arv mingi tipu jaoks on piiratud selle tipu kaugusega lehtedest.

Väljakutseid on ülimalt…

$$
\begin{aligned}
1\cdot\frac{n}{2}+2\cdot\frac{n}{4}+3\cdot\frac{n}{8}+4\cdot\frac{n}{16}+\cdots &= \\
\left(\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots\right)&+ \\
\left(\frac{n}{4}+\frac{n}{8}+\frac{n}{16}+\cdots\right)&+ \\
\left(\frac{n}{8}+\frac{n}{16}+\frac{n}{32}+\cdots\right)&+\cdots= \\
n+\frac{n}{2}+\frac{n}{4}+\cdots&=2n\quad.
\end{aligned}
$$

i-ndat järku binomiaalpuu B_i on defineeritud järgmiselt:

- B_0 koosneb ühestainsast tipust.
- B_i saadakse B_{i-1}-st, lisades tema juurele täiendava vasakpoolseima alampuu B_{i-1}.

Teisisõnu, B_i koosneb juurtipust ja selle alampuudest $B_{i-1}, B_{i-2}, \ldots, B_1, B_0$.

Binomiaalkuhi on binomiaalpuude hulk, kus kõigi puude järk on erinev ja kõik puud rahuldavad kuhjaomadust.

Seega määrab tippude arv binomiaalkuhjas, mitmendat järku puud sellesse kuhja kuuluvad.

Binomiaalkuhi K (suurusega n) toetab järgmisi operatsioone:

tee_kuhi	$\Theta(1)$
lisa(K, x)	$\Theta(\log n)$
leia_max(K)	$\Theta(\log n)$
võta_max(K)	$\Theta(\log n)$
suurenda_võti(K, tipp, uus_võti)	$\Theta(\log n)$
vähenda_võti(K, tipp, uus_võti)	$\Theta(\log n)$
kustuta(K, tipp)	$\Theta(\log n)$
ühenda(K_1, K_2)	$\Theta(\log n)$

Binomiaalkuhja võib mälus kujutada
struktuurina, kus igale tipule vastab kirje

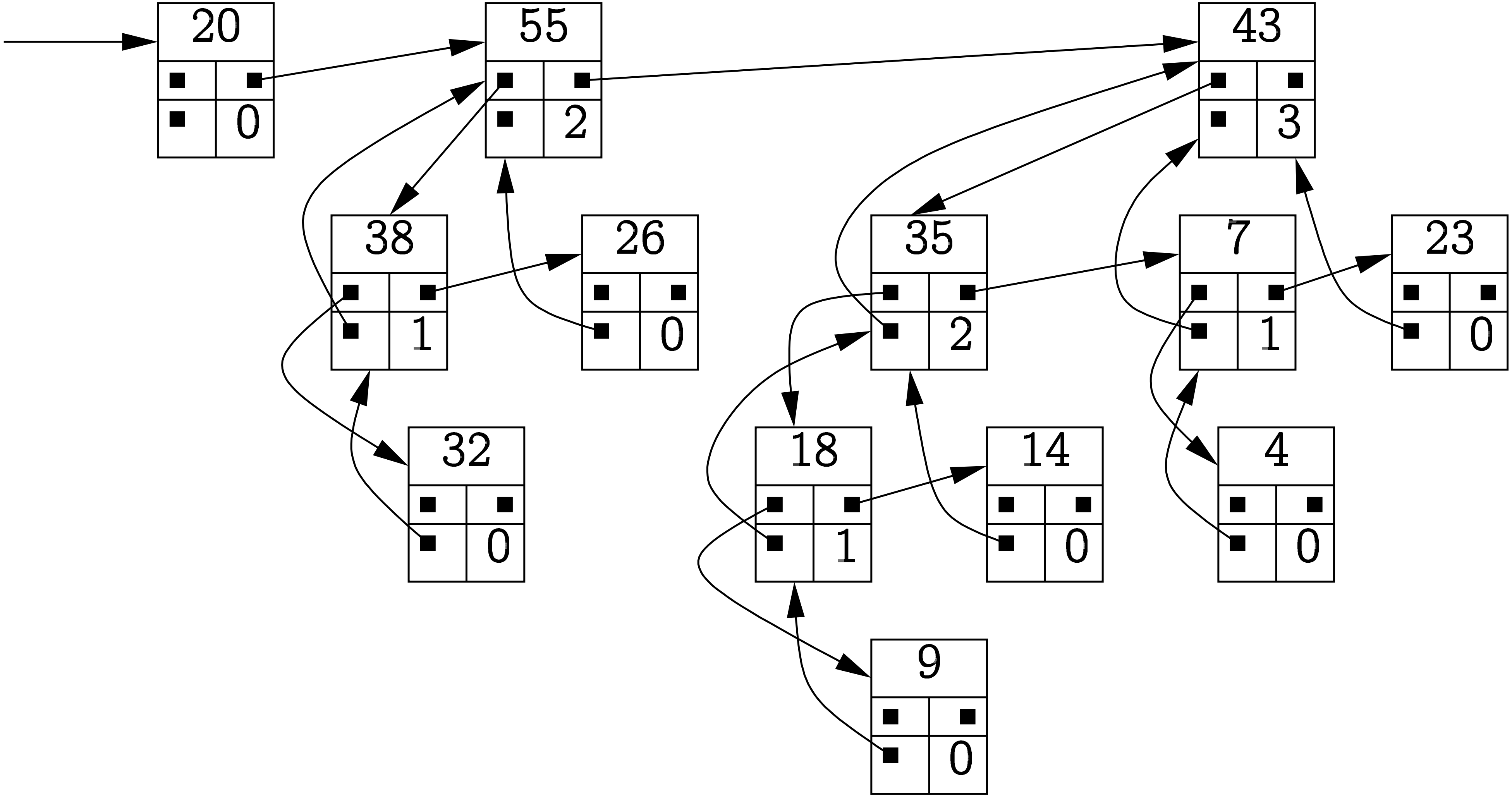
<table>
<tr><td colspan="2">.võti
Andmed</td></tr>
<tr><td>.alluv</td><td>.kolleeg</td></tr>
<tr><td>.ülem</td><td>.järk</td></tr>
</table>

- Väli *.järk* näitab, mitmendat järku binomiaalpuu juureks antud tipp on.

- B_i juurtipu alluvateks on vasakult paremale lugedes B_{i-1} juurtipp, B_{i-2} juurtipp,... B_0 juurtipp.

- Puude juurtipud on seotud *juurahelasse* viida *.kolleeg* kaudu, väiksemad puud esinevad eespool. Viidaks kuhjale on viit väikseima puu juurtipule.

Näide: 13-tipuline binomiaalkuhi.



Binomiaalkuhja maksimaalne element asub mõne puu juurtipus:

leia_max(K) on

$R := K.\mathrm{Andmed}$
$p := K.kolleeg$
while $p \neq \mathrm{NIL}$ **do**
  if $p.v\tilde{o}ti > R.v\tilde{o}ti$ **then**
    $R := p.\mathrm{Andmed}$
  $p := p.kolleeg$
return R

Keerukus: proportsionaalne juurahela pikkusega, mis on kuni $\lfloor \log n \rfloor + 1$. Seega on keerukus $\Theta(\log n)$.

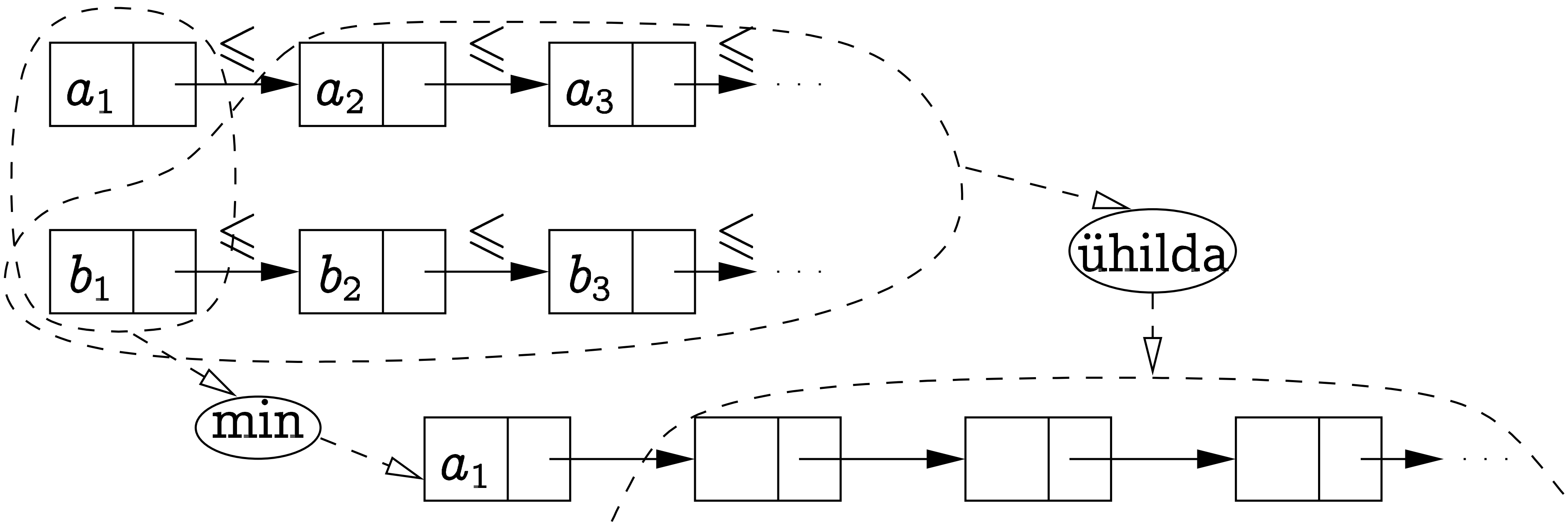
Protseduur ühenda_puud(p, q) teeb kahest i-ndat järku puust $(i+1)$-st järku puu. (Eeldus: $p.v\tilde{o}ti \leqslant q.v\tilde{o}ti$)

$p.\ddot{u}lem := q$
$p.kolleeg := q.alluv$
$q.alluv := p$
$q.j\ddot{a}rk := q.j\ddot{a}rk + 1$

Keerukus: konstantne. Seda protseduuri kasutame alamprogrammina kuhjade ühendamisel.

ühenda(K_1, K_2) teeb kõigepealt puude juurahelatest üheainsa ahela, mis on puude järkude järgi sorteeritud.

Sorteeritud listide ühildamine:



Listide pikkusega l_1 ja l_2 ühildamise keerukus: Meil tuleb

- leida kahest elemendist väiksem — konstantne keerukus;
- ühildada listid pikkusega $l_1 - 1$ ja l_2 või l_1 ja $l_2 - 1$;
- lisada leitud vähim element konstrueeritud listi esimeseks elemendiks — konstantne keerukus.

Kokkuvõttes on keerukus $\Theta(l_1 + l_2)$.

ühilda_binomiaalkuhjad(K_1, K_2) on

```
if K1 = NIL then
    return K2
else if K2 = NIL then
    return K1
else if K1.järk < K2.järk then
    K1.kolleeg := ühilda_binomiaalkuhjad(K1.kolleeg, K2)
    return K1
else
    K2.kolleeg := ühilda_binomiaalkuhjad(K1, K2.kolleeg)
    return K2
```

Kui mõlemas kuhjas on ülimalt n elementi, siis on keerukus $\Theta(\log n)$.

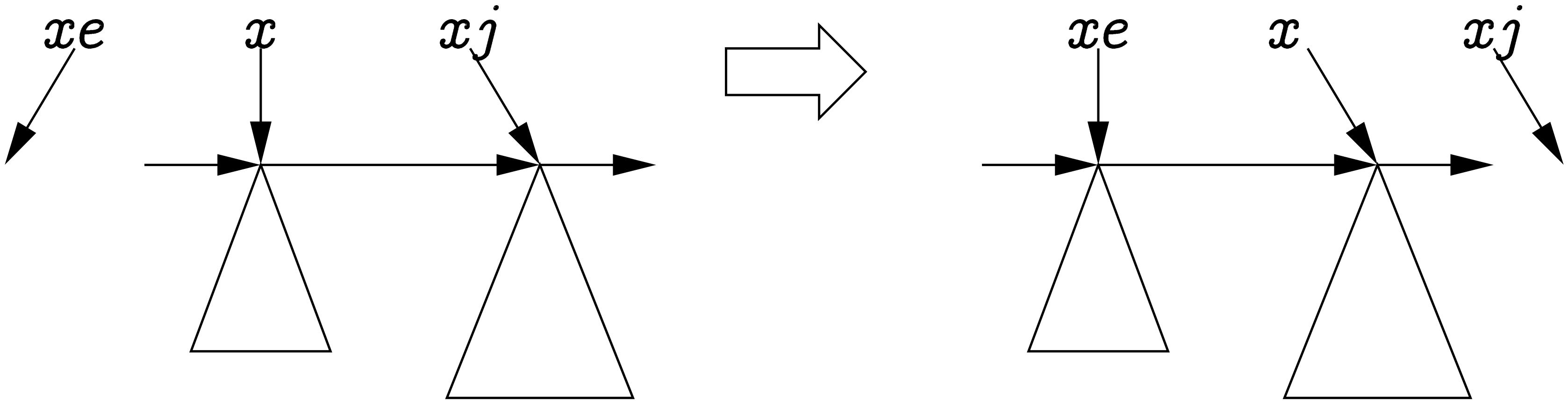
ühenda(K_1, K_2) on

$K :=$ ühilda_binomiaalkuhjad(K_1, K_2)
$x := K$; $xe :=$ NIL; $xj := x.kolleeg$
while $xj \neq$ NIL **do**
$\dots$

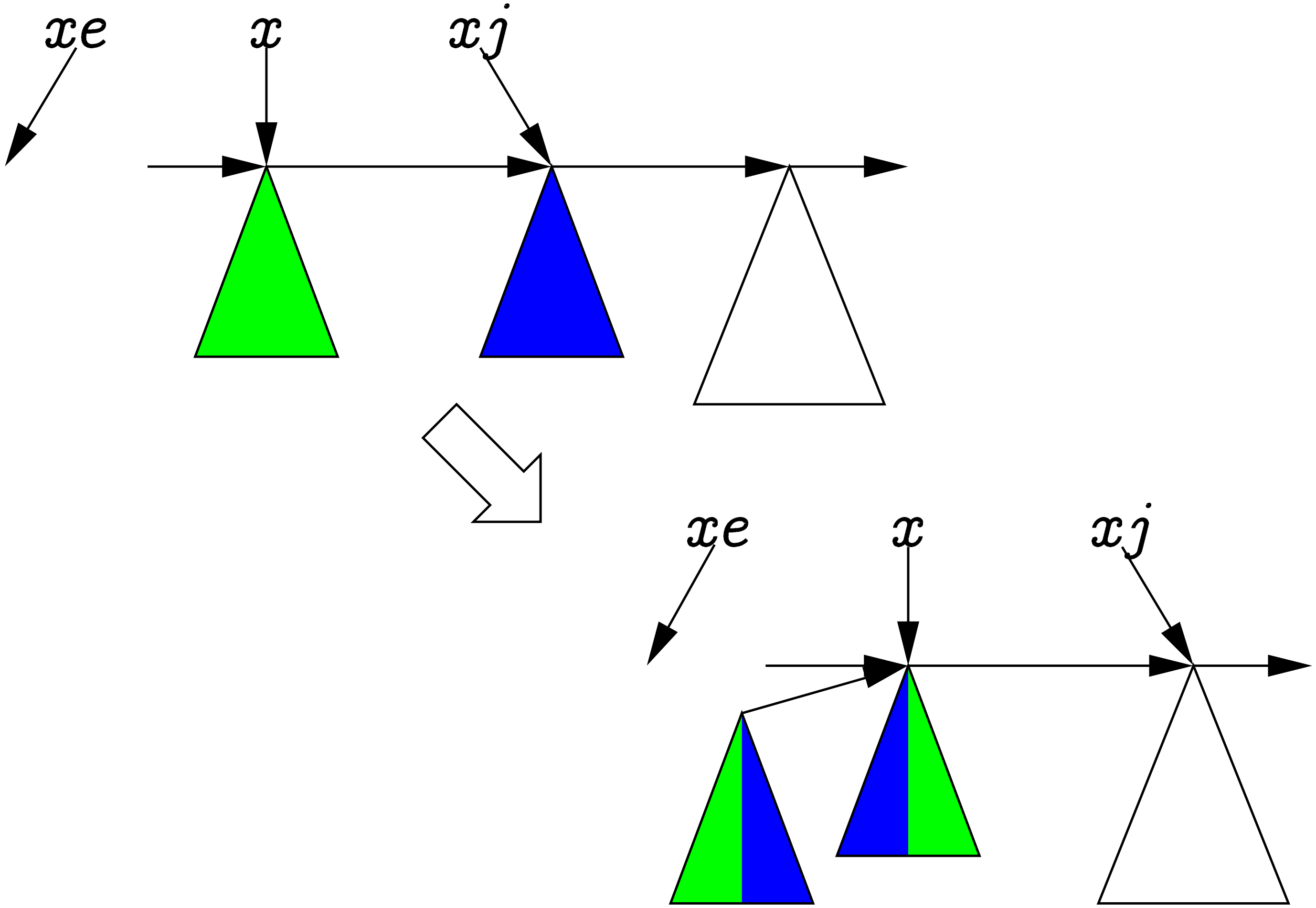
Tsükli alguses x viitab mingile binomiaalpuule, mille järk on suurem kui sellele puule eelneva puu järk. xe viitab eelmisele ja xj järgmisele puule.

Sama järku puid on ahelas 1 kuni 3. Need on järgmine ja ülejärgmine puu. Soovime, et alles jääks ülimalt üks seda järku puu.

Kui on 1 puu, siis liigume lihtsalt ahelas edasi.

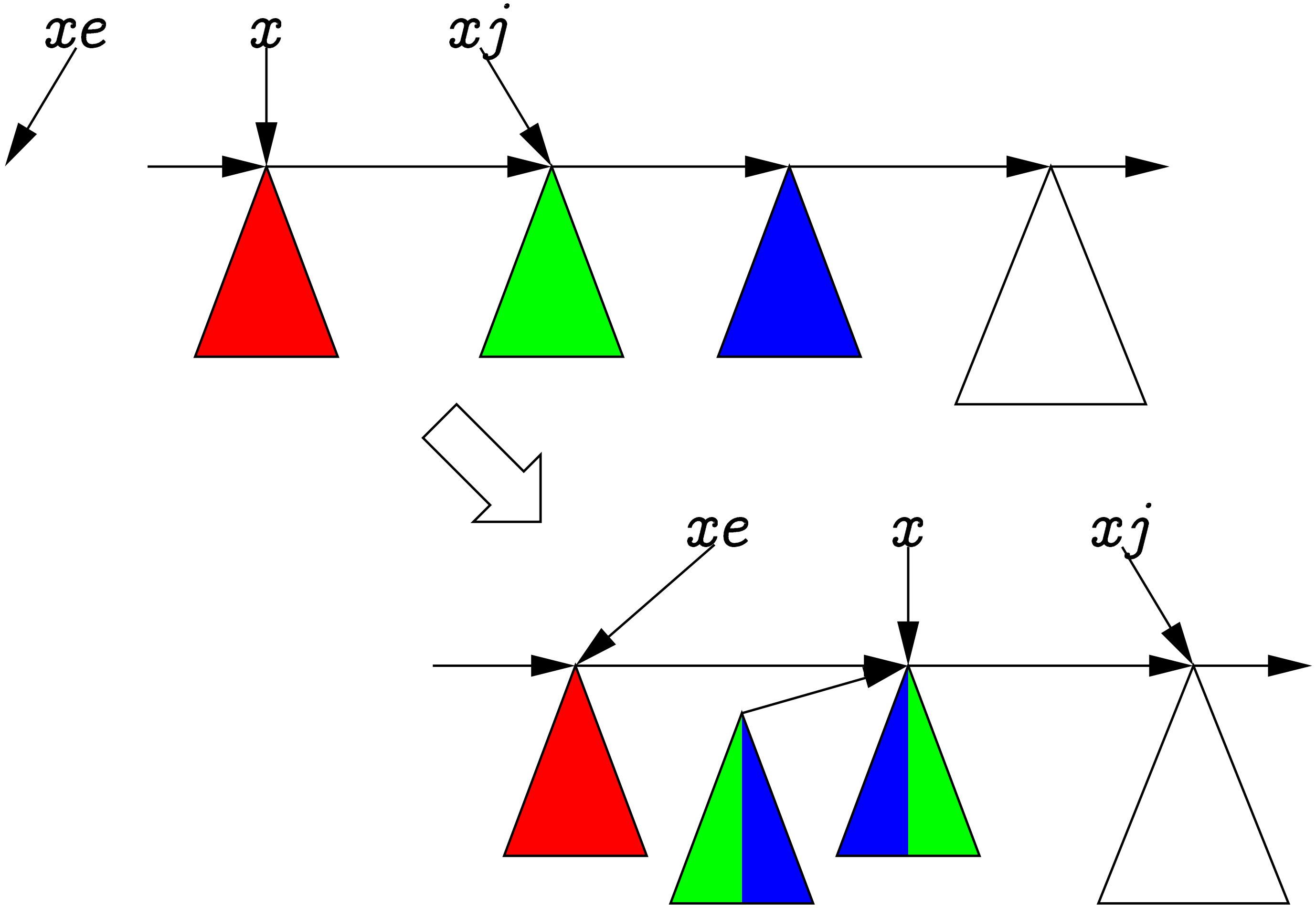


Kui on 2 puud, siis ühendame nad.



Me peame nende juurte võtmeid võrdlema.

Kui on 3 puud, siis ühendame kaks parempoolset.



Võime lihtsalt sammu võrra edasi liikuda ja siis käituda, nagu oleks kaks puud.

```
if x.järk ≠ xj.järk then
    xe := x; x := xj; xj := xj.kolleeg
    continue
if xj.kolleeg ≠ NIL and xj.järk = xj.kolleeg.järk then
    xe := x; x := xj; xj := xj.kolleeg
if x.võti < xj.võti then
    ühenda_puud(x, xj)
    (xe = NIL ? K : xe.kolleeg) := xj
    x := xj
else
    x.kolleeg := xj.kolleeg
    ühenda_puud(xj, x)
xj := x.kolleeg
```

Ühildamine võtab aega $\Theta(\log n)$.

Tsükli igal sammul väheneb x-i viidatava puu kaugus juurahela lõpust.

Juurahela pikkus on $\Theta(\log n)$ ja ühel tsüklisammul tehtava töö hulk on piiratud konstandiga.

Seega on kuhjade ühendamise keerukus $\Theta(\log n)$.

Kuhjast suurima elemendi võtmiseks:

1. Leiame puu, mille juureks on suurim element, eemaldame ta kuhjast.

2. Selle puu juurtipu alampuudest teeme uue kuhja.

   - Selleks tuleb kolleegide järjekord ümber pöörata.

3. Saadud kuhja ühendame esialgse kuhjaga.

Suurima elemendi võtmine tagastab paari sellest elemendist ja muudetud kuhjast.

võta_max(K) on

```
R := K.Andmed; q := K; qq := NIL
p := K.kolleeg; pp := K
while p ≠ NIL do
    if p.võti > R.võti then
        R := p.Andmed; q := p; qq := pp
    pp := p; p := p.kolleeg
(qq = NIL ? K : qq.kolleeg) := q.kolleeg
r := q.alluv; re := NIL
while r ≠ NIL do
    rj := r.kolleeg
    r.kolleeg := re
    re := r; r := rj
free(q)
return (R, ühenda(K, re))
```

Keerukus:

- Suurima juurega puu leidmine ja eemaldamine:
  - Proportsionaalne juurahela pikkusega — $\Theta(\log n)$.
- Kolleegide järjekorra ümberpööramine:
  - Proportsionaalne kolleegide arvuga — $\Theta(\log n)$ (Kiho konspekt, järeldus lk. 39).
- Kuhjade ühendamine — $\Theta(\log n)$.

Kokku seega $\Theta(\log n)$.

Mõne tipu võtme muutmiseks võime kasutada protseduure viia_üles ja viia_alla, tegutsedes ainult seda tippu sisaldava puu piires.

viia_üles(q) on

if $q.\textit{ülem} = \mathrm{NIL}$ **or** $q.\textit{võti} \leqslant q.\textit{ülem}.\textit{võti}$ **then return**
$q.\mathrm{Andmed} :=: q.\textit{ülem}.\mathrm{Andmed}$
viia_üles($q.\textit{ülem}$)

viia_üles keerukus on:

- Puus on kuni n tippu.
- Seega on puus kuni $\log n + 1$ taset, mis võib-olla kõik läbi käia tuleb.
- Igal tasemel kulub konstantne aeg.

Keerukus on $\Theta(\log n)$.

suurenda_võti(K, q, a) on

1 $q.v\tilde{o}ti := a$
2 viia_üles(q)

keerukusega $\Theta(\log n)$

```
viia_alla(q) on
if q.alluv = NIL then return
m := q.alluv; p := m.kolleeg
while p ≠ NIL do
    if p.võti > m.võti then
        m := p
    p := p.kolleeg
if q.võti ⩾ m.võti then return
q.Andmed :=: m.Andmed
viia_alla(m)
```

viia_alla keerukus on:

- Puus on kuni $\log n + 1$ taset, mis võib-olla kõik läbi käia tuleb.
- Igal tasemel kulub aeg, mis on proportsionaalne alluvate arvuga.
- Max. tööaeg: $\log n + (\log n - 1) + \cdots + 2 + 1 = \Theta(\log^2 n)$.

Aga meie lubasime ennist, et vähenda_võti keerukuseks on $\Theta(\log n)$.

Kirje lisamiseks loome sellest kirjest üheelemendilise kuhja ja ühendame olemasolevaga.

lisa(K, R) on

$L := \textbf{new}(\text{kirjetüüp})$
$L.\text{Andmed} := R$; $L.alluv := \text{NIL}$
$L.kolleeg := \text{NIL}$; $L.ülem := \text{NIL}$
$L.järk := 0$
$\textbf{return}$ ühenda(K, L)

Keerukus: ühendamise keerukus ja veel mingi konstant. Seega $\Theta(\log n)$.

Kirje eemaldamiseks tehakse tema võti hästi suureks ja võetakse seejärel kuhja max. element.

kustuta(K, q) on

$q.v\tilde{o}ti := \infty$
viia_üles(q)
$(\_, K') :=$ võta_max(K); **return** K'

Keerukus: viia_üles keerukus ja ühendamise keerukus (mõlemad $\Theta(\log n)$). Seega $\Theta(\log n)$.

vähenda_võti(K, q, a) on

$R := q.$Andmed
$K :=$ kustuta(K, q)
$R.v\tilde{o}ti := a$
return lisa(K, R)

S.t. eemaldame kirje ja lisame ta uuesti. Keerukus on kustutamise ja lisamise keerukuse summa, s.t. $\Theta(\log n)$.