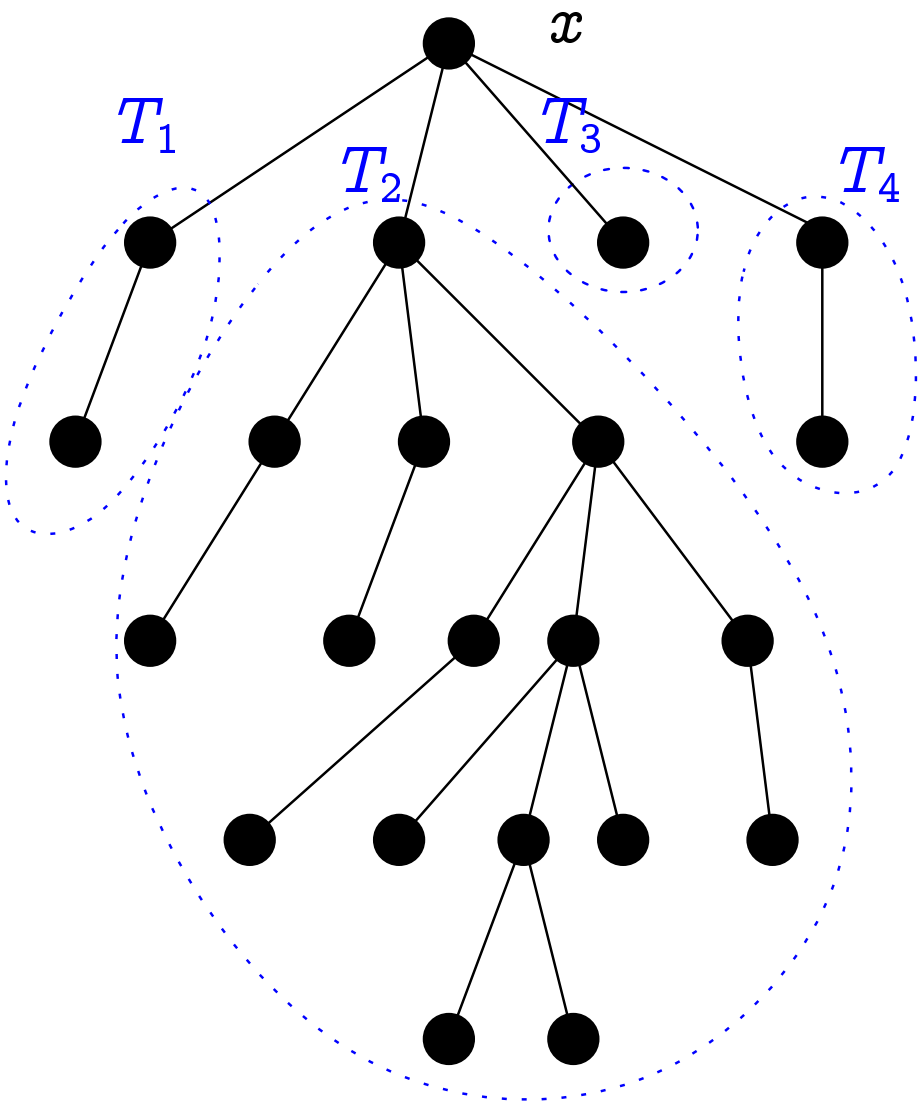
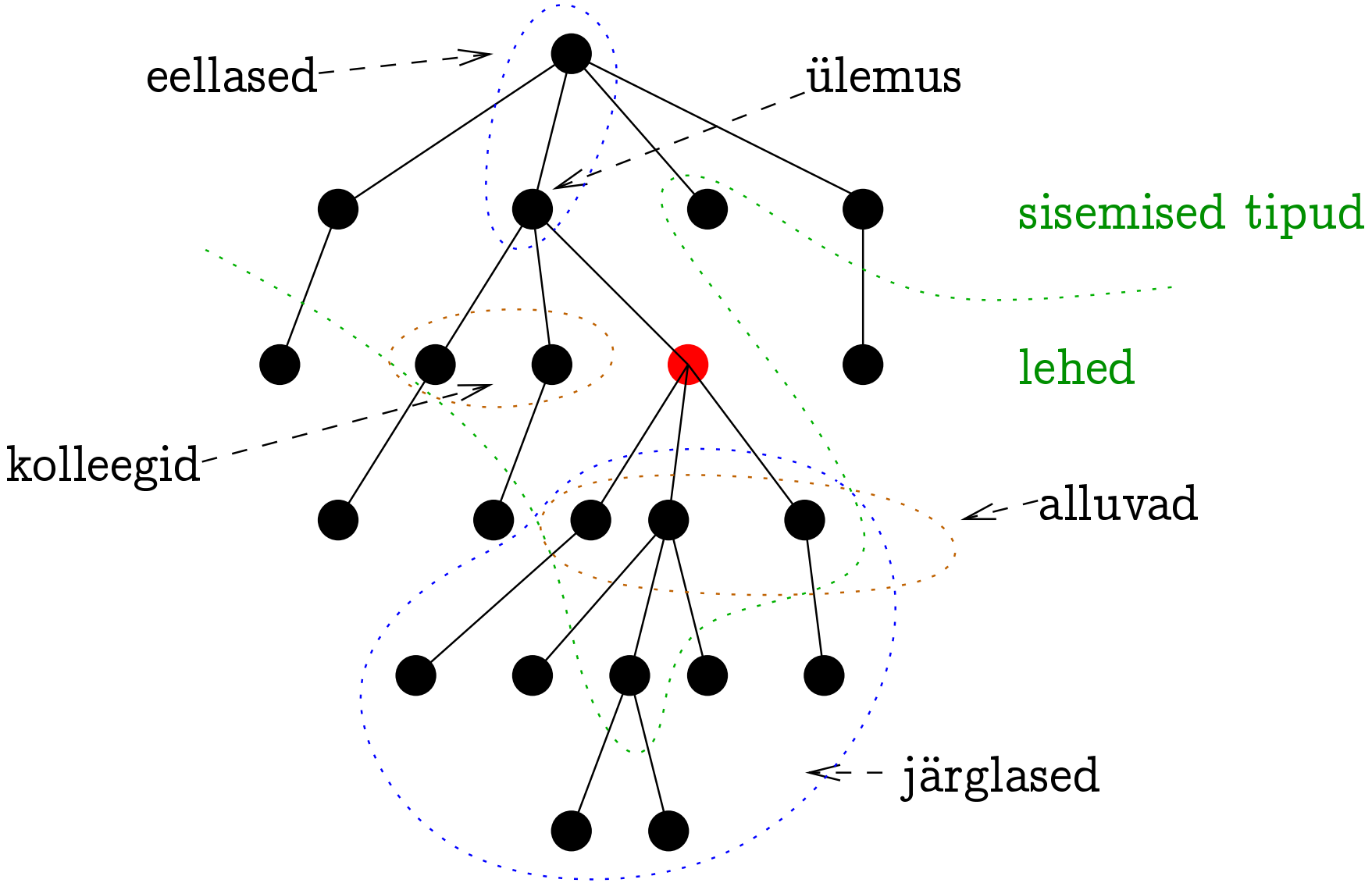
(Juurega) puu T on üks järgmistest variantidest:

- tühi puu;
- tipp x ja mittetühjad juurega puud $T_1, \ldots, T_m$, kus $m \geqslant 0$.


x
T_1
T_2
T_3
T_4


eellased
ülemus
sisemised tipud
lehed
kolleegid
alluvad
järglased

Kahendpuu T on üks järgmistest variantidest:

- tühi puu;
- üksainus tipp x;
- tipp x ning kahendpuud T_{vasak} ja T_{parem}.

Kahendpuu ei ole samaväärne puuga — kui kahendpuu mingil tipul on ainult üks alluv, siis on ta kas vasak või parem alluv. Puu tipu ainsa alluva korral vasak/parem-erisust ei ole.

Lause. Mittetühjas kahendpuus, kus ühegi tipu aste ei ole 1, on lehti ühe võrra rohkem kui sisemisi tippe.

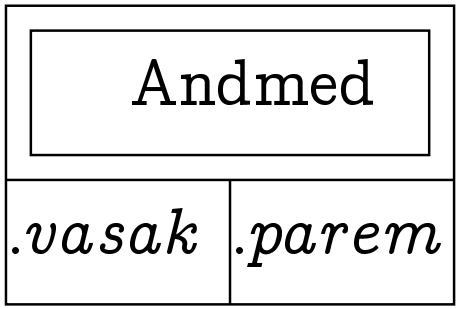
Tõestus. Induktsiooniga üle kahendpuu struktuuri.

Ühetipulises puus olev ainus tipp on leht.

Kui puus T on juurtipp x ja selle alluvatest koosnevad mittetühjad kahendpuud T_v ja T_p, siis olgu l_v ja l_p lehtede arvud nendes. Puus T on siis lehti $l_v + l_p$ ja sisemisi tippe

$$
\begin{aligned}
&l_v - 1 + l_p - 1 + && \text{(puudes } T_v \text{ ja } T_p \text{ vastavalt induktsioonile)} \\
&+ 1 = && \text{(tipp } x\text{)} \\
&= l_v + l_p - 1 \ . &&
\end{aligned}
$$

Kahendpuud võime arvuti põhimälus kujutada struktuurina, kus igale tipule vastab kirje

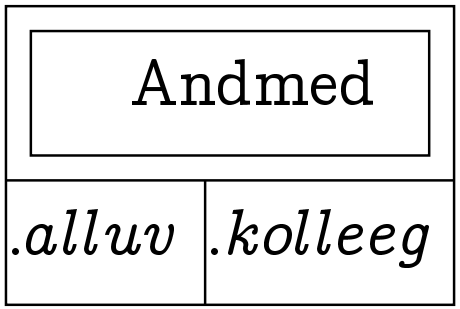


kus väli *.vasak* viitab vasakule ning *.parem* paremale alluvale.

Teatavad algoritmid võivad nõuda täiendavate väljade olemasolu kas osa „Andmed“ sees või väljaspool seda.

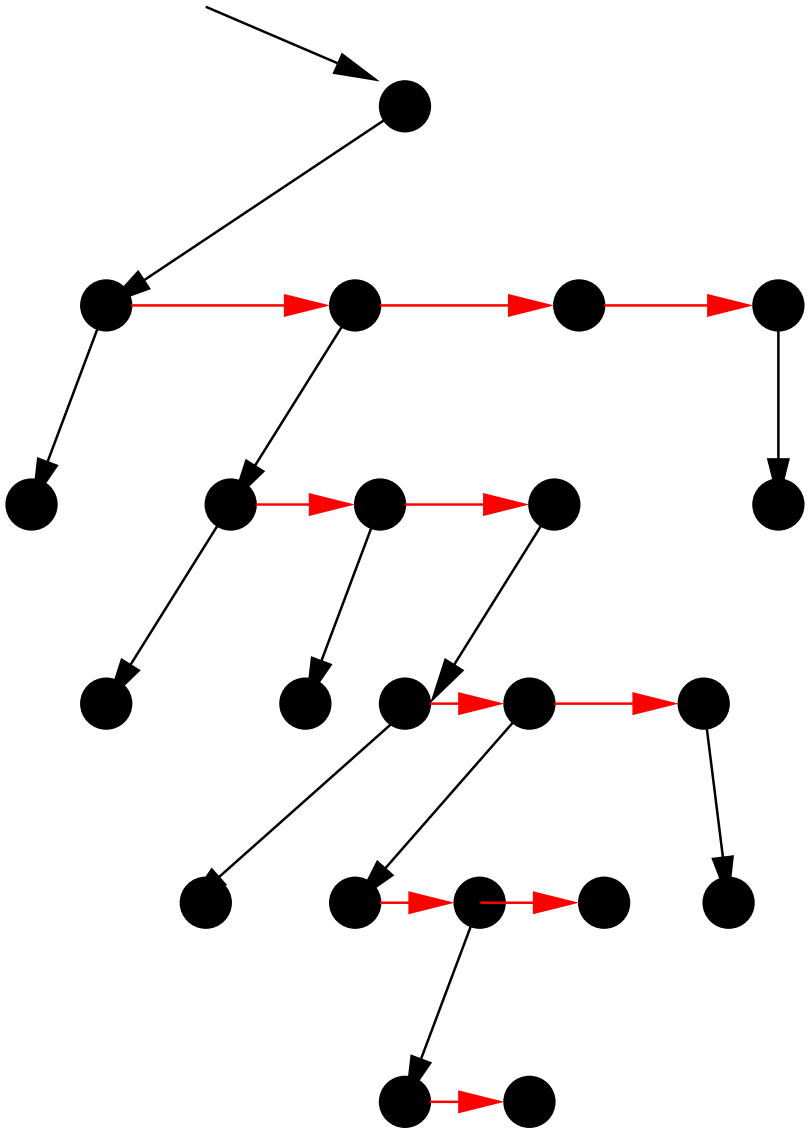
(kahendpuu naturaalesitus)

Puud võime arvuti põhimälus kujutada struktuurina, kus igale tipule vastab kirje



kus väli *.alluv* viitab vasakpoolseimale alluvale ning *.kolleeg* paremalt järgmisele kolleegile.

(puu naturaalesitus)



viidad: alluv / kolleeg

puuduvad viidad ≡ NIL

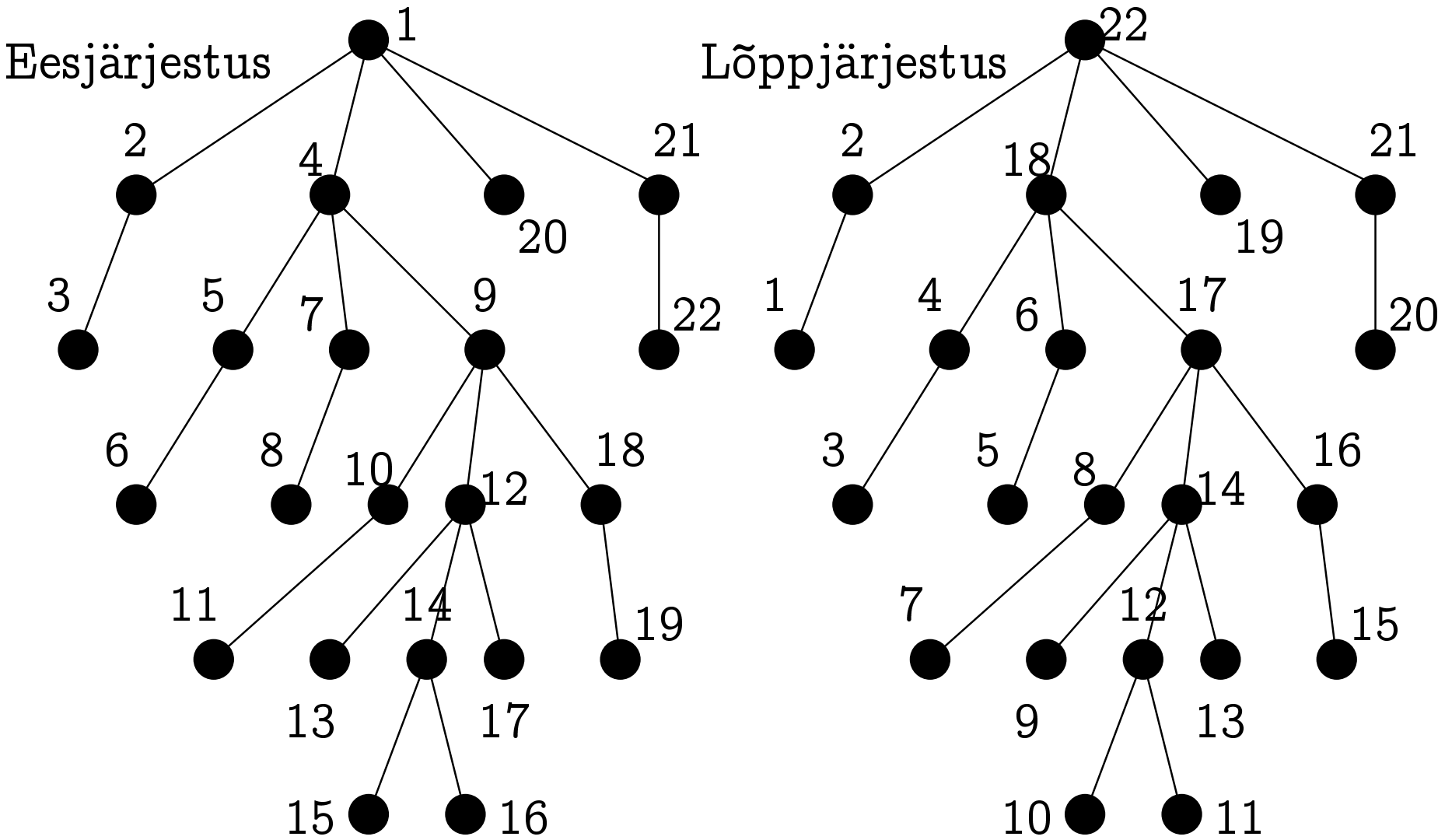
Kui loeme alluv≡vasak ja kolleeg≡parem, siis saame puu naturaalesitusele vastava kahendpuu.

Olgu meil antud mingi protseduur P, mida me puu igal tipul rakendada tahame.

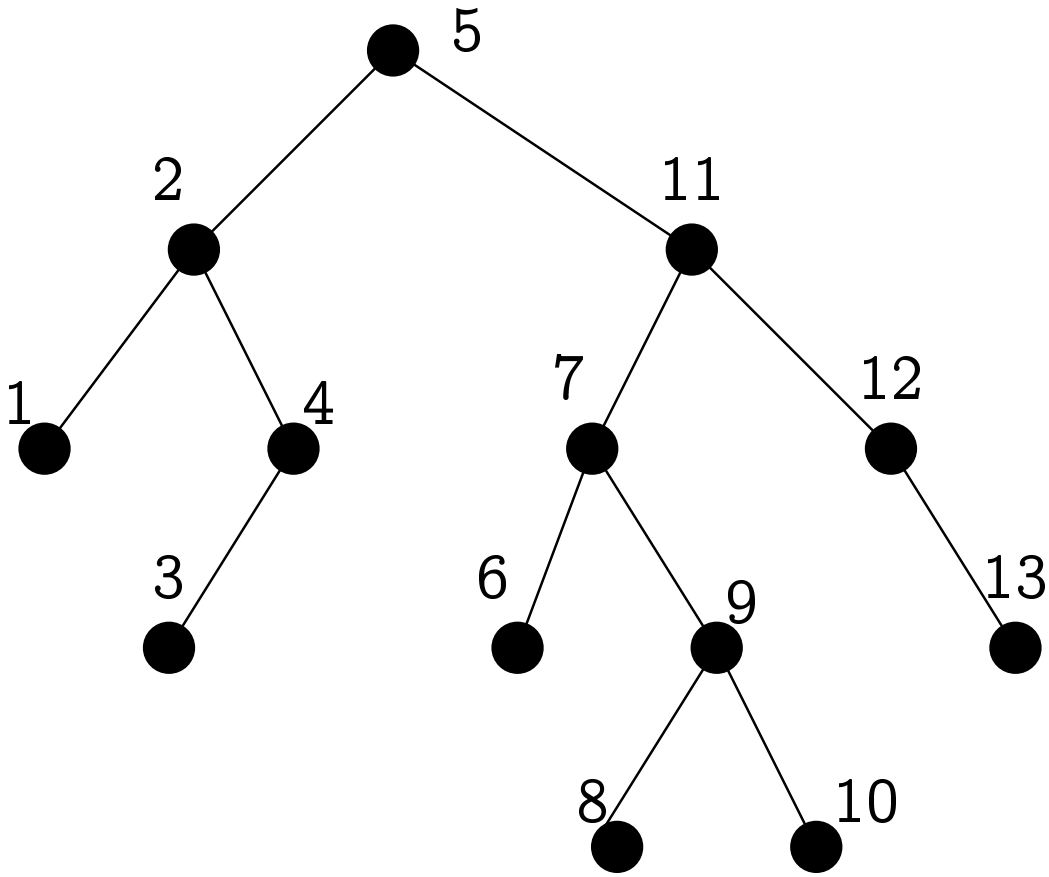
Mis järjekorras rakendada?

Järjestust, kus kõigepealt on puu juur ja seejärel samal viisil järjestatult juure vahetute alampuude tipud, nimetatakse *eesjärjestuseks*.

Järjestust, kus kõigepealt on samal viisil järjestatult juure vahetute alampuude tipud ja lõpuks puu juur, nimetatakse *lõppjärjestuseks*.


Eesjärjestus
1
2
4
20
21
3
5
7
9
22
6
8
10
12
18
11
14
19
13
17
15
16
Lõppjärjestus
22
2
18
19
21
1
4
6
17
20
3
5
8
14
16
7
12
15
9
13
10
11

Kahendpuu tippude järjestust, kus kõigepealt on samal viisil järjestatult juure vasaku vahetu alampuu tipud, seejärel puu juur ja lõpuks samal viisil järjestatult juure vahetu parema alampuu tipud, nimetatakse *keskjärjestuseks*.

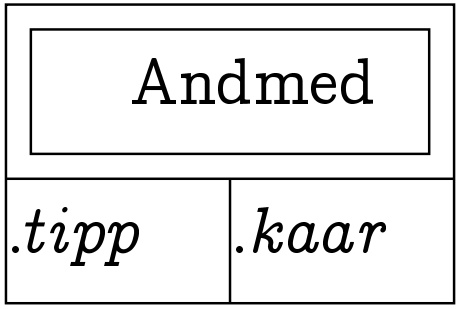

5
2
11
1
4
7
12
3
6
9
13
8
10

Suunatud graaf G koosneb

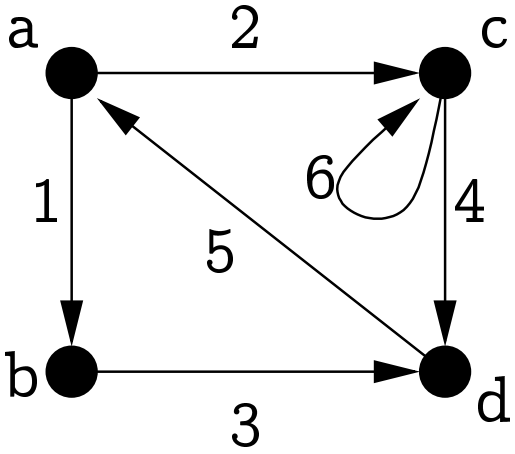
- *tippude* hulgast V;
- *servade* ehk *kaarte* hulgast E;
- *intsidentsusfunktsioonist* $\mathcal{E} : E \longrightarrow V \times V$, mis seab igale kaarele vastavusse tema alg- ja lõpptipu.

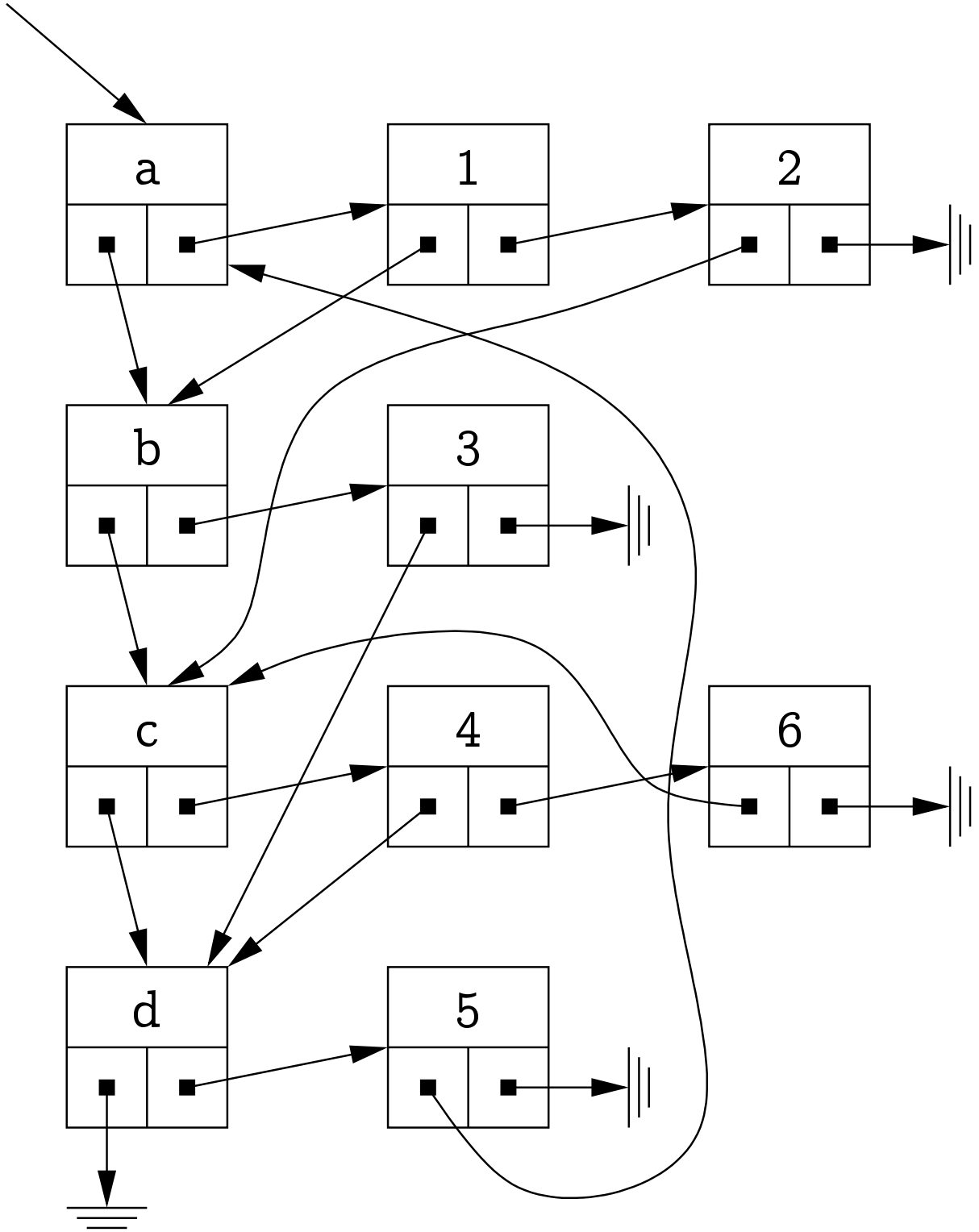
Suunamata serva kujutame üht ja teist pidi suunatud servade paarina.

Graafi võib mälus kujutada struktuurina, kus igale tipule ja igale servale vastab kirje



- Graafi tipud on lihtahelas välja *.tipp* kaudu; viit esimesele tipule on viit kogu graafile.
- Kõik mingist tipust algavad servad on lihtahelas välja *.kaar* kaudu; esimesele lülile selles ahelas viitab selle tipu väli *.kaar*.
- Servadele vastavates kirjetes viitab väli *.tipp* selle serva lõpptipule (vastavale kirjele).


a
2
c
1
6
4
5
b
3
d


a
1
2
b
3
c
4
6
d
5

Dünaamiline järjend: andmetüüp, mille väärtusvaruks on kirjete suvalise pikkusega järjendid.

Magasin: dünaamiline järjend, kus elemente lisatakse alati järjendi lõppu ja elemente võetakse järjendi lõpust.

Järjekord: dünaamiline järjend, kus elemente lisatakse alati järjendi lõppu ja elemente võetakse järjendi algusest.

Elemendi r lisamist järjendisse Q tähistame $Q \Leftarrow r$. Elemendi võtmist järjendist Q ja omistamist muutujale r tähistame $r \Leftarrow Q$.

Operatsioonid: Lisa 1, Lisa 2, Lisa 3, Võta, Lisa 4, Võta, Võta

Magasin:

Võetud:

Järjekord:

Võetud:

Operatsioonid: Lisa 2, Lisa 3, Võta, Lisa 4, Võta, Võta

Magasin: 1

Võetud:

Järjekord: 1

Võetud:

Operatsioonid: Lisa 3, Võta, Lisa 4, Võta, Võta

Magasin:

1	2

Võetud:

Järjekord:

1	2

Võetud:

Operatsioonid: Võta, Lisa 4, Võta, Võta

Magasin:

1	2	3

Võetud:

Järjekord:

1	2	3

Võetud:

Operatsioonid: Lisa 4, Võta, Võta

Magasin:

1	2

Võetud: 3

Järjekord:

2	3

Võetud: 1

Operatsioonid: Võta, Võta

Magasin:

1	2	4

Võetud: 3

Järjekord:

2	3	4

Võetud: 1

Operatsioonid: Võta

Magasin:

1	2

Võetud: 3, 4

Järjekord:

3	4

Võetud: 1, 2

Operatsioonid:

Magasin: 1

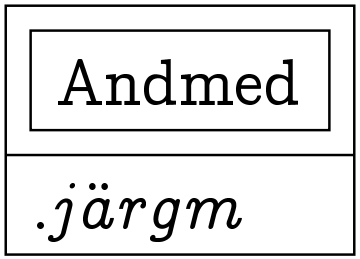
Võetud: 3, 4, 2

Järjekord: 4

Võetud: 1, 2, 3

Magasini võib realiseerida dünaamilise andmestruktuuri abil.

Ühte elementi hoitakse järgmises struktuuris:

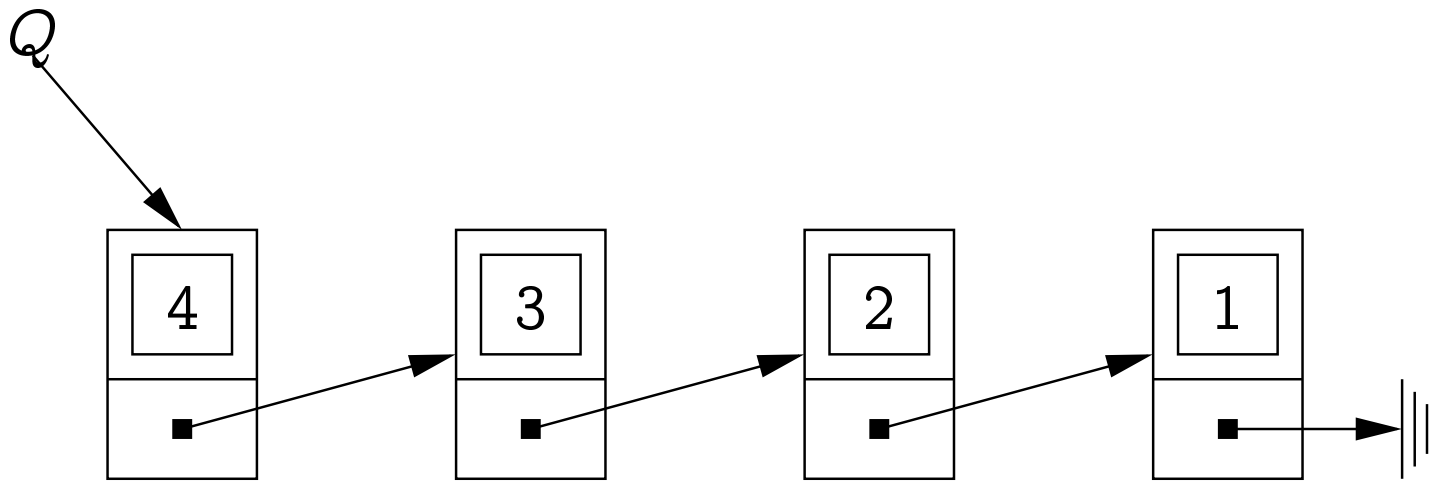


Siin *.järgm* on viit järgmisele (sügavamale) elemendile.

Magasini poole pöördumiseks kasutatakse viita tema esimesele elemendile.

Tühja magasini tähistab nullviit.

Magasin Q, millesse on lisatud elemendid $1, 2, 3, 4$:



$Q \Leftarrow r$ on siis

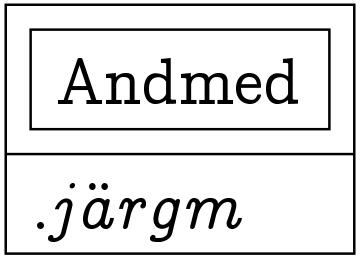
1 $e := \mathbf{new}(\text{magasini element})$
2 $e.\text{Andmed} := r$
3 $e.järgm := Q$
4 $Q := e$

ja $r \Leftarrow Q$ on

1 **if** $Q = \text{NIL}$ **then error**
2 $r := Q.\text{Andmed}$
3 $e := Q$
4 $Q := Q.järgm$
5 vabasta e

Järjekorda võib realiseerida dünaamilise andmestruktuuri abil.

Ühte elementi hoitakse järgmises struktuuris:

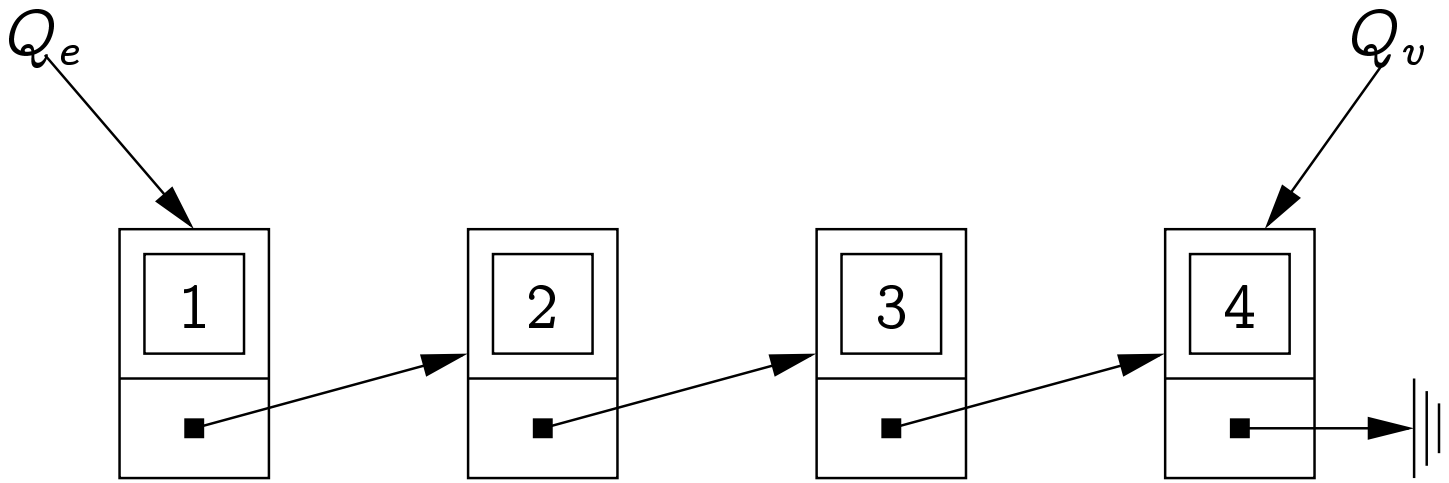


Siin *.järgm* on viit järgmisele (hiljem lisatud) elemendile.

Järjekorra poole pöördumiseks kasutatakse viitasid tema esimesele ja viimasele elemendile.

Tühja järjekorda tähistab kaks nullviita.

Järjekord Q, kuhu on lisatud elemendid $1, 2, 3, 4$:



Siin Q_e on võtmise ja Q_v lisamise koht.

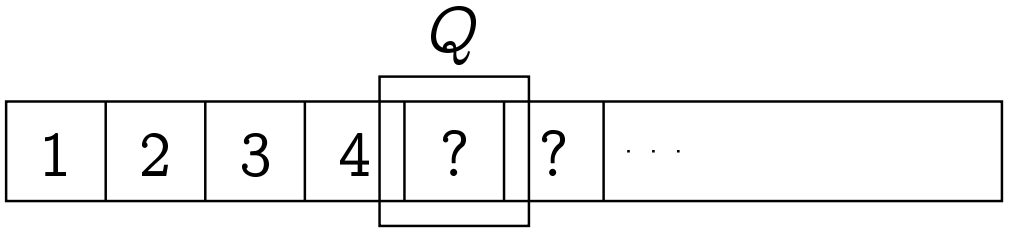
$Q \Leftarrow r$ on siis

```
1   e := new(järjekorra element)
2   e.Andmed := r
3   e.järgm := NIL
4   if Q_e = Q_v = NIL then
5       Q_e := Q_v := e
6   else
7       Q_v.järgm := e
8   Q_v := e
```

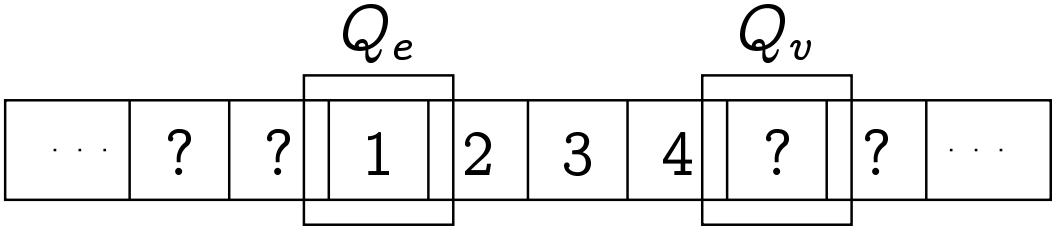
```
ja r ⇐ Q on
1   if Q_e = Q_v = NIL then error
2   r := Q_e.Andmed
3   e := Q_e
4   if Q_e = Q_v then
5       Q_e := Q_v := NIL
6   else
7       Q_e := Q_e.järgm
8   vabasta e
```

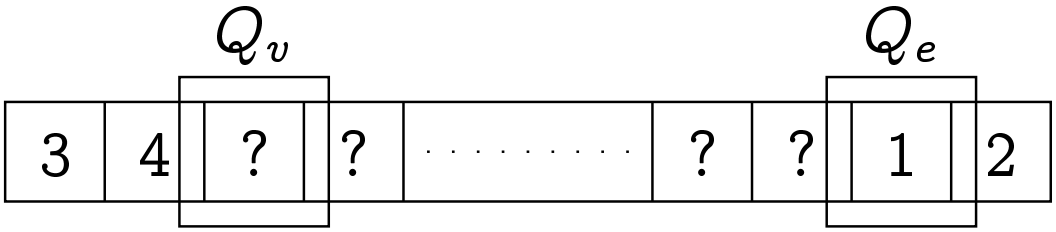
Kui on teada elementide suurim võimalik arv magasinis / järjekorras, siis saab teda realiseerida ka massiivi $a = [a_1, \ldots, a_n]$ abil.

Magasin, kuhu on lisatud elemendid $1, 2, 3, 4$:



Järjekord (2 varianti), kus on elemendid $1, 2, 3, 4$:





Olgu antud protseduur P, mida tahame välja kutsuda antud graafi G kõigil tippudel.

Kaks võimalikku järjekorda, mis mingis mõttes vastavad graafi struktuurile, on graafi *laiuti läbimine* ja *sügavuti läbimine*.

Olgu p viit graafi G esimesele tipule. Eeldame, et graafi kõik tipud on tipust p alates mööda servi liikudes saavutatavad.

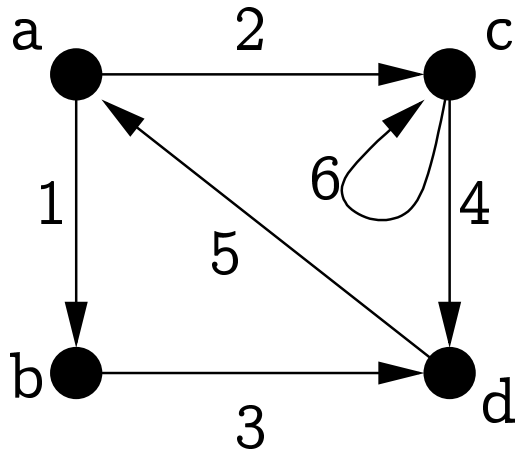
Olgu tipu-/servastruktuuris täiendav väli *.läbitud* (ei kuulu ossa „Andmed“).

Laiuti läbimine: Olgu Q järjekord (esialgu tühi)

```
 1  q := p;
 2  while q ≠ NIL do
 3      q.läbitud := false; q := q.tipp
 4  p.läbitud := true; Q ⇐ p
 5  while ¬tühi?(Q) do
 6      q ⇐ Q; P(q);
 7      r := q.kaar
 8      while r ≠ NIL do
 9          if ¬(r.tipp.läbitud) then
10              Q ⇐ r.tipp
11              r.tipp.läbitud := true
12          r := r.kaar
```

Näide:

r



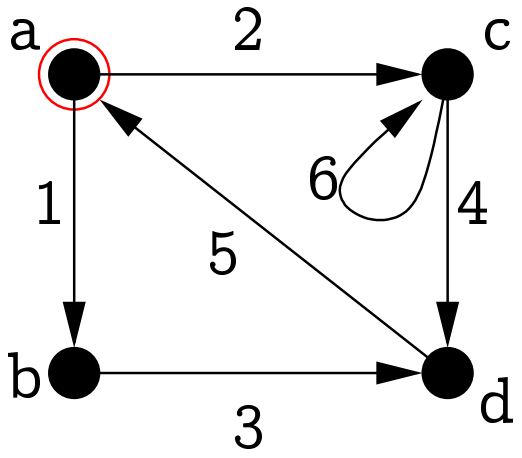
Q :

q

$P(\cdot)$ väljakutsed:

Näide:

r



q

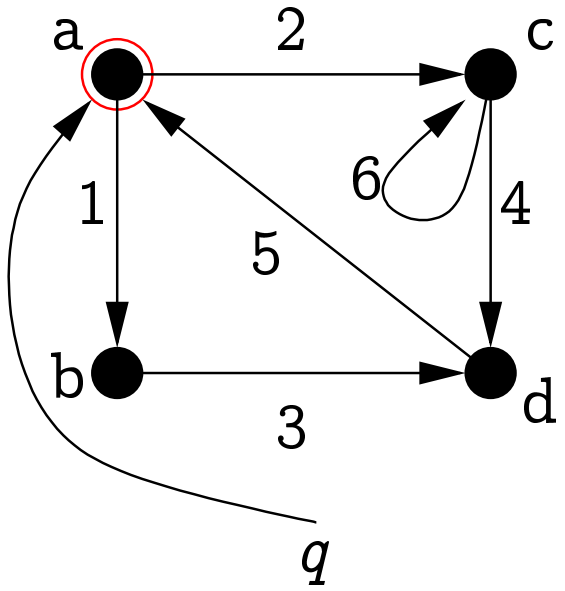
Q :

a

$P(\cdot)$ väljakutsed:

Näide:

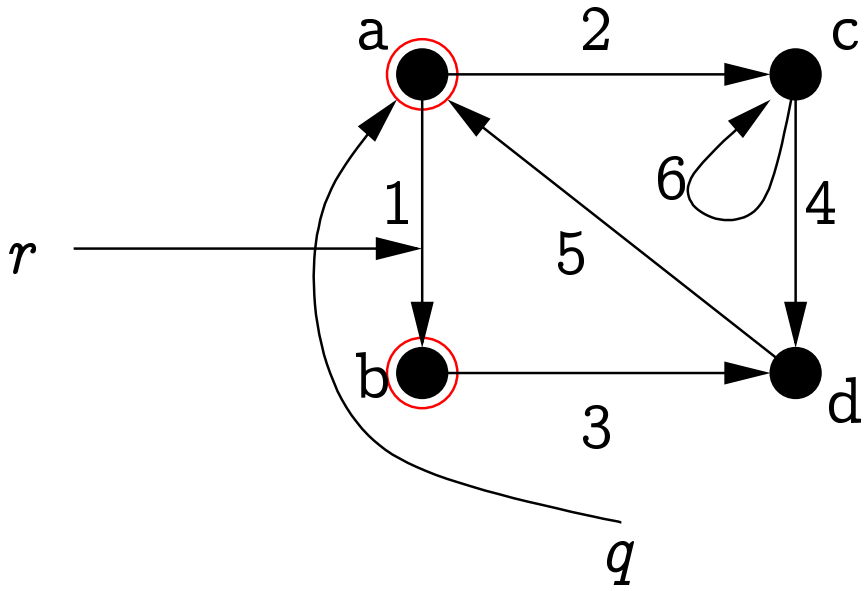
r



Q :

$P(\cdot)$ väljakutsed: a

Näide:

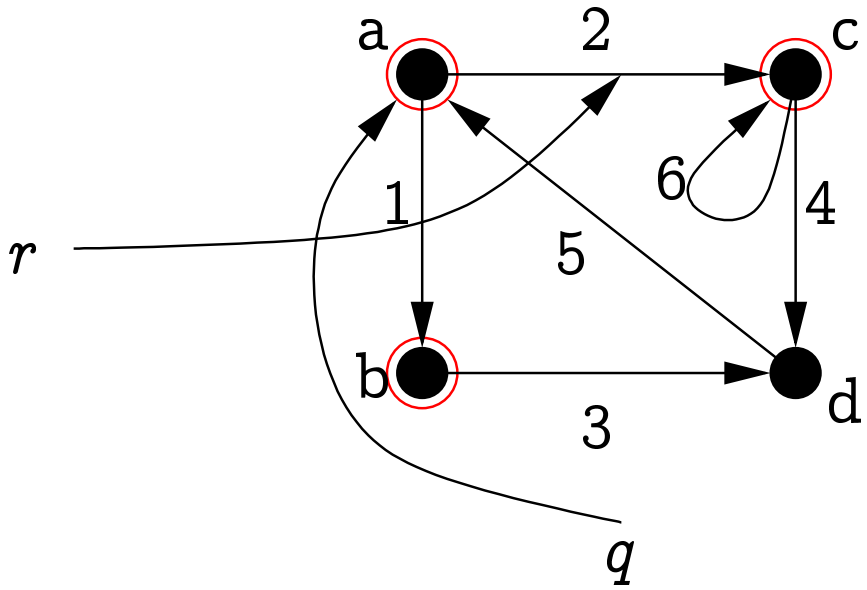


Q :

b

$P(\cdot)$ väljakutsed: a

Näide:

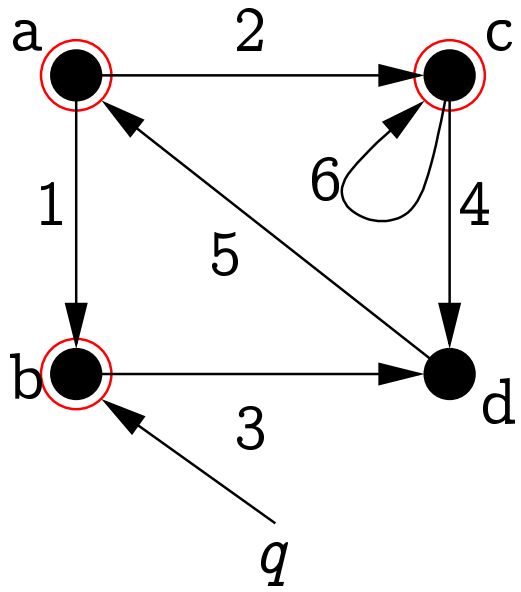


Q :

b c

$P(\cdot)$ väljakutsed: a

Näide:



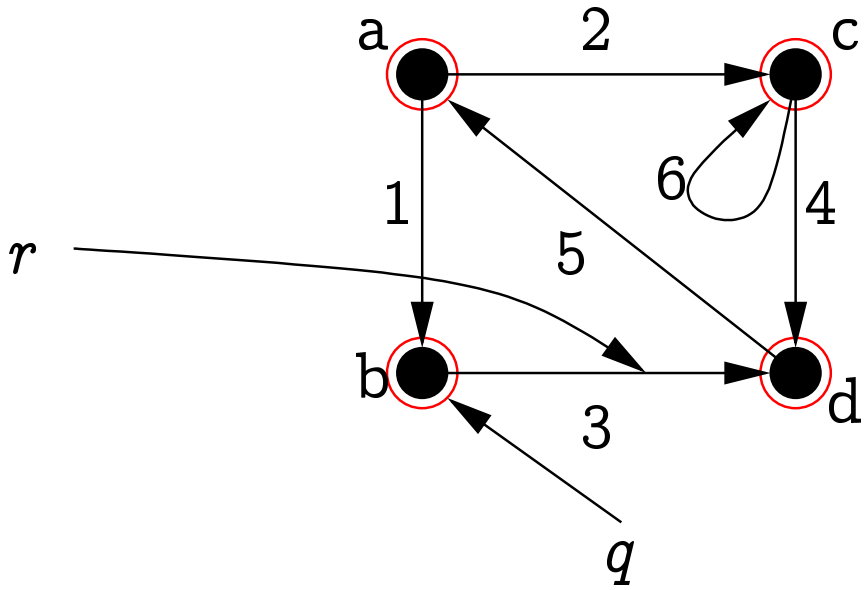
r

Q :

c

$P(\cdot)$ väljakutsed: a b

Näide:

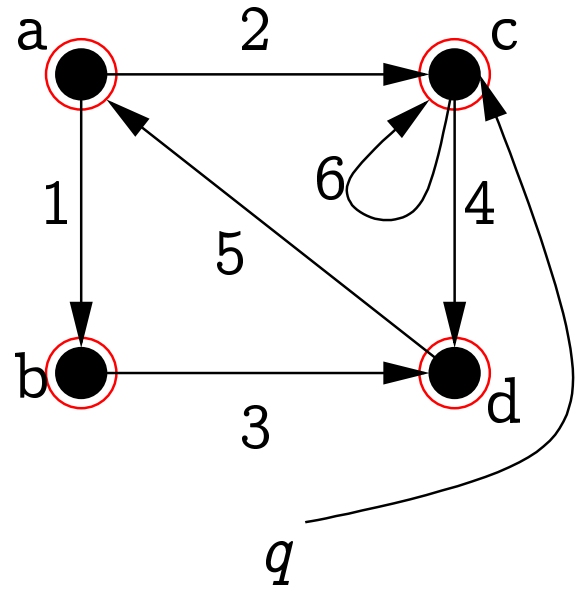


Q :

c d

$P(\cdot)$ väljakutsed: a b

Näide:



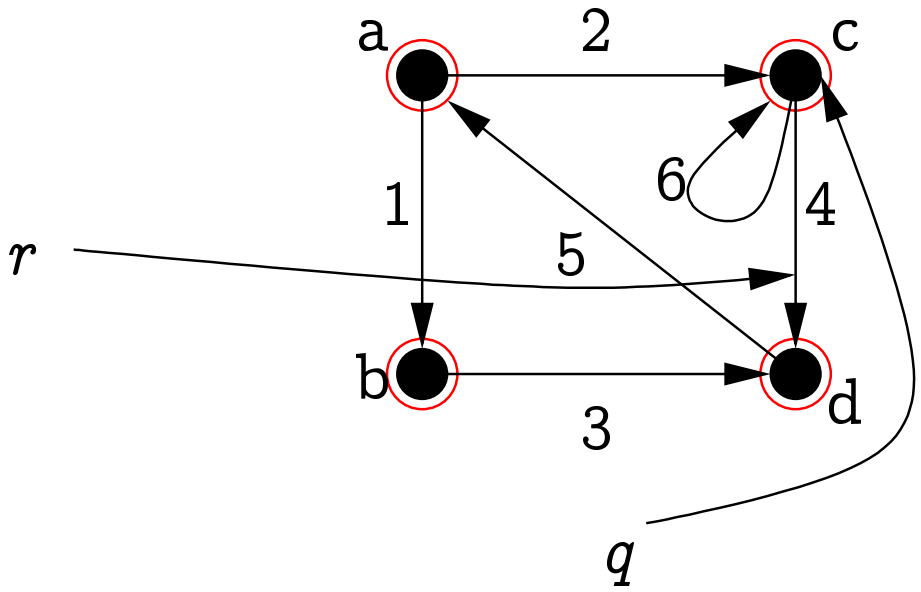
r

Q :

d

$P(\cdot)$ väljakutsed: a b c

Näide:

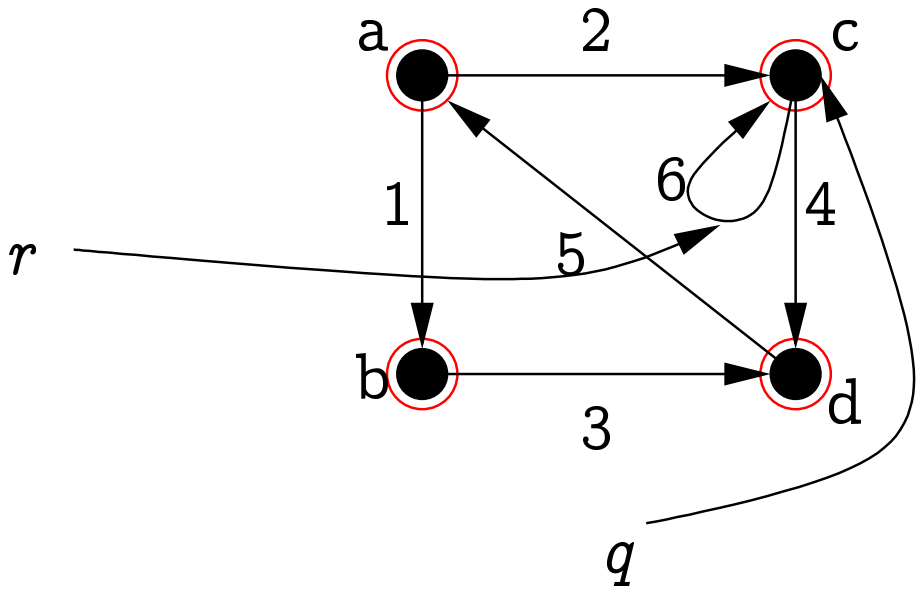


Q :

d

$P(\cdot)$ väljakutsed: a b c

Näide:



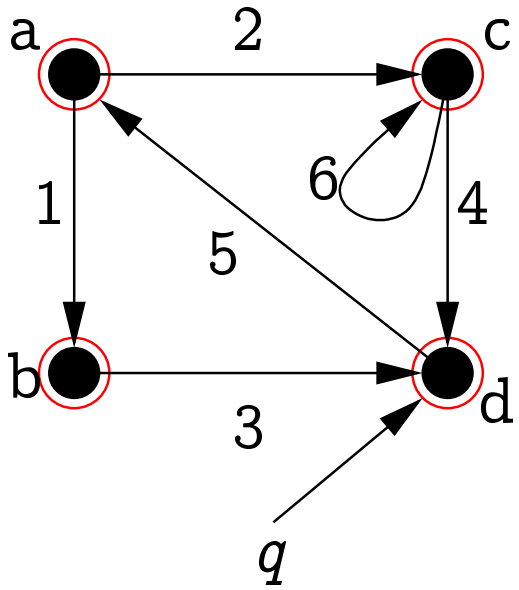
Q :

d

$P(\cdot)$ väljakutsed: a b c

Näide:

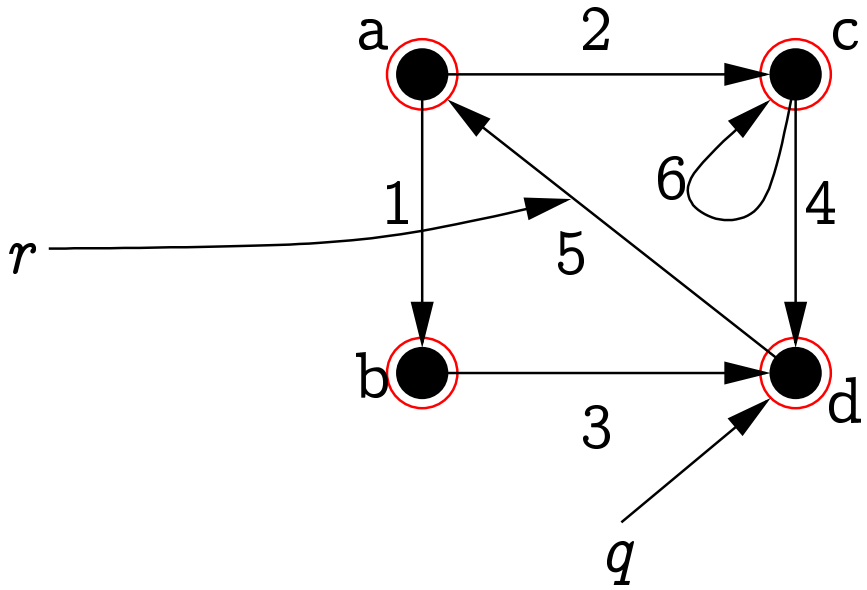
r



Q :

$P(\cdot)$ väljakutsed: a b c d

Näide:

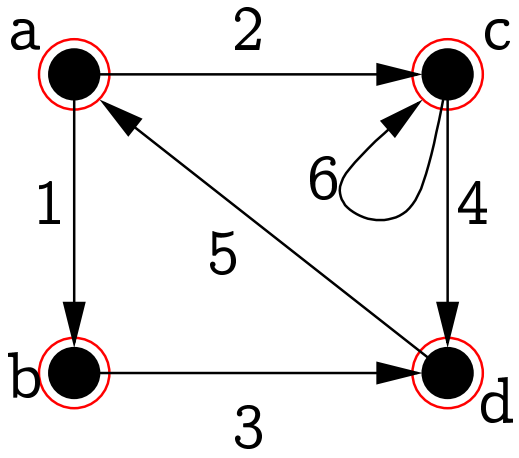


Q :

$P(\cdot)$ väljakutsed: a b c d

Näide:

r



Q :

q

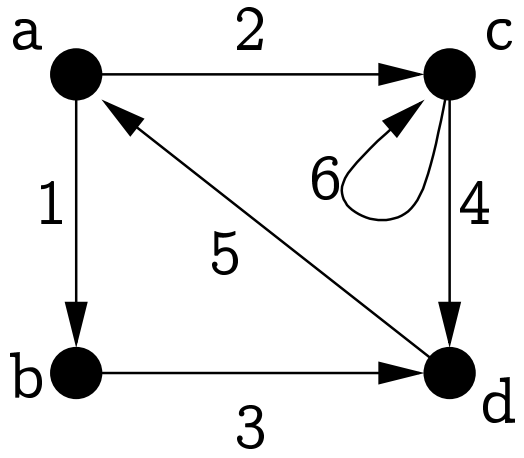
$P(\cdot)$ väljakutsed: a b c d

Sügavuti läbimine: Olgu Q magasin (esialgu tühi)

1..12 sama programm, mis laiuti läbimisel

Näide:

r



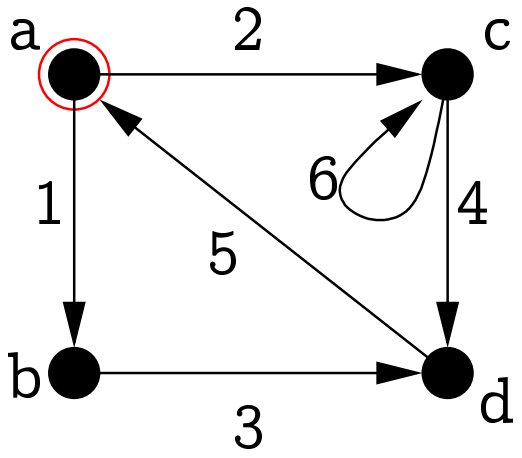
Q :

q

$P(\cdot)$ väljakutsed:

Näide:

r



q

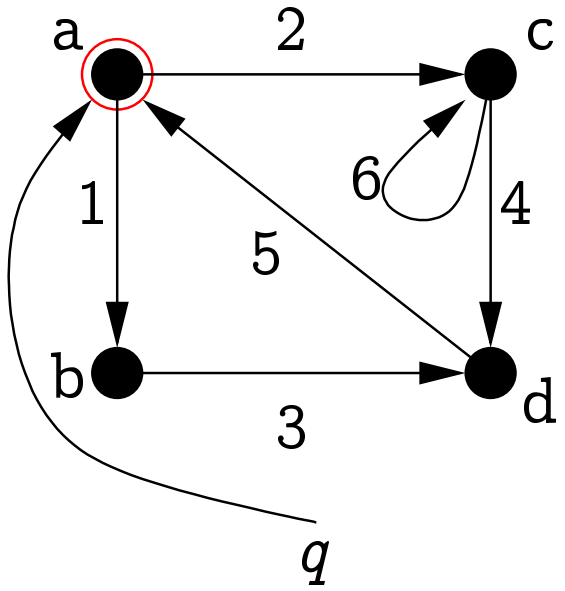
Q :

a

$P(\cdot)$ väljakutsed:

Näide:

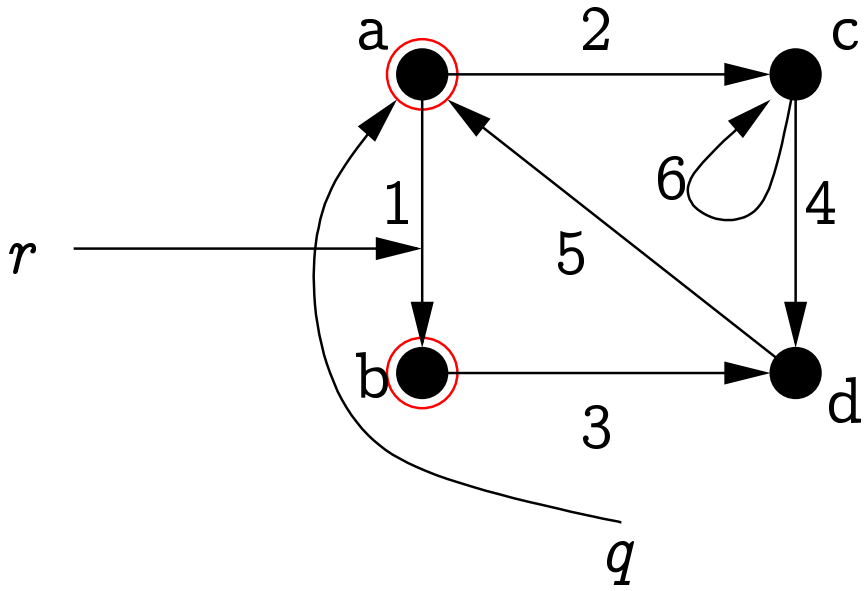
r



Q :

$P(\cdot)$ väljakutsed: a

Näide:

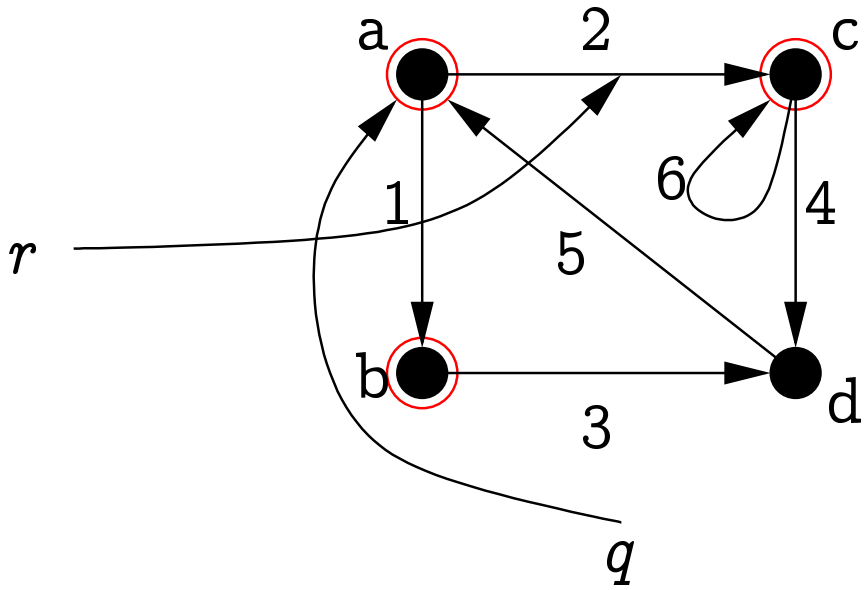


Q :

b

$P(\cdot)$ väljakutsed: a

Näide:

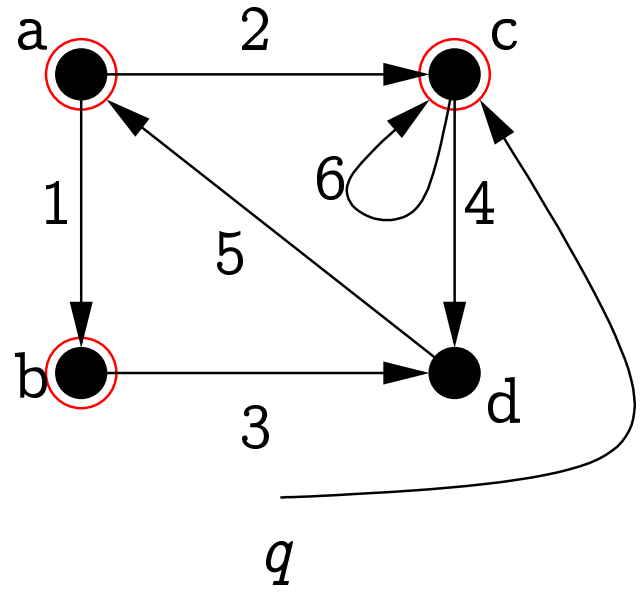


Q :

b c

$P(\cdot)$ väljakutsed: a

Näide:



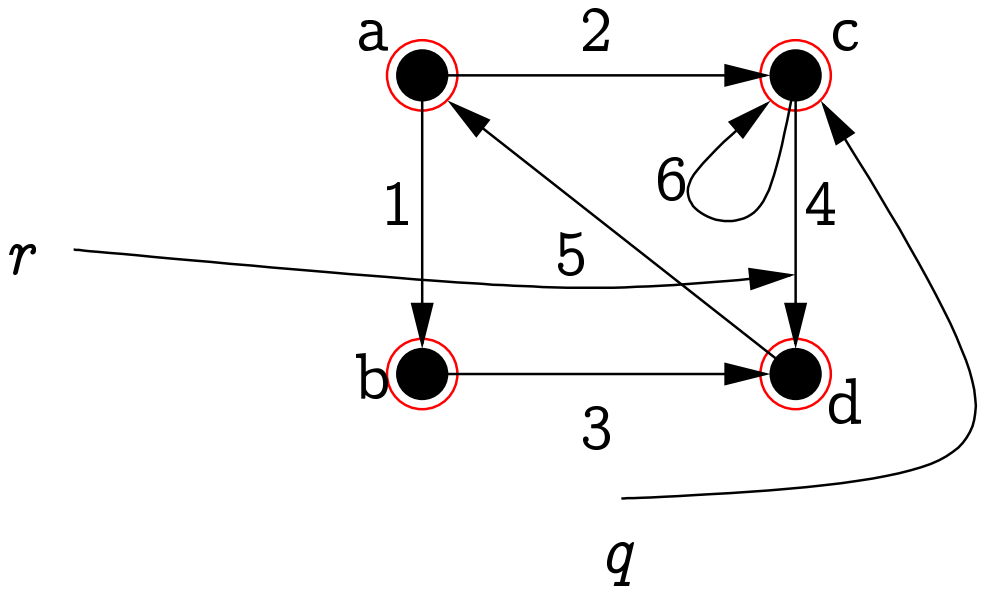
r

Q :

b

$P(\cdot)$ väljakutsed: a c

Näide:

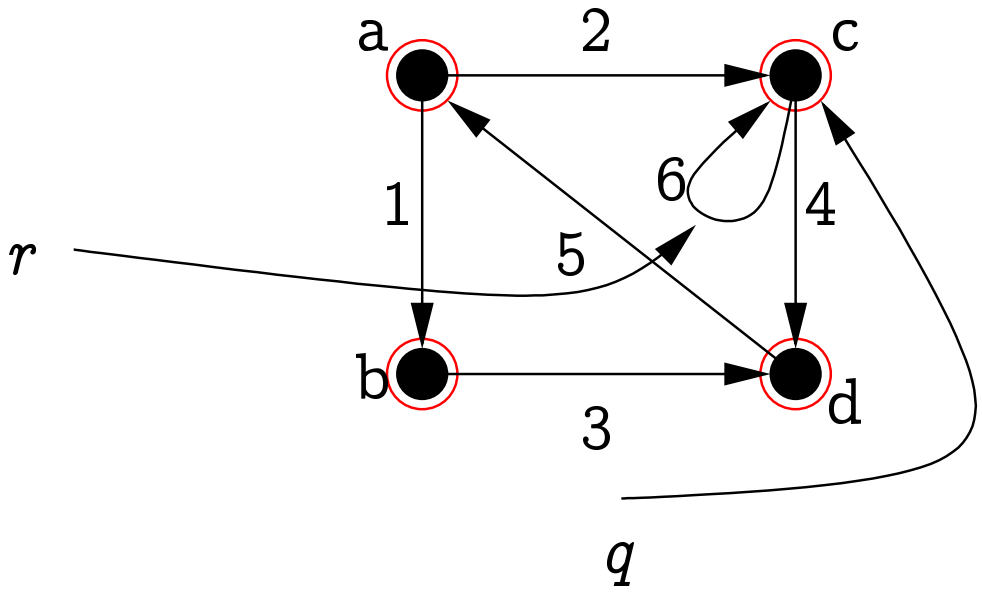


Q :

b d

$P(\cdot)$ väljakutsed: a c

Näide:

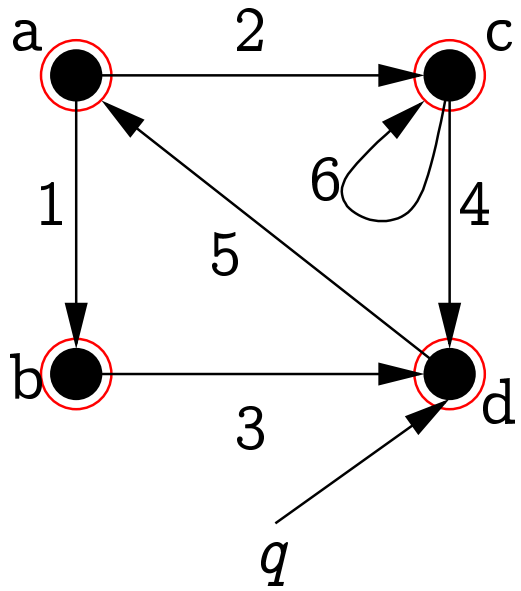


Q :

b d

$P(\cdot)$ väljakutsed: a c

Näide:



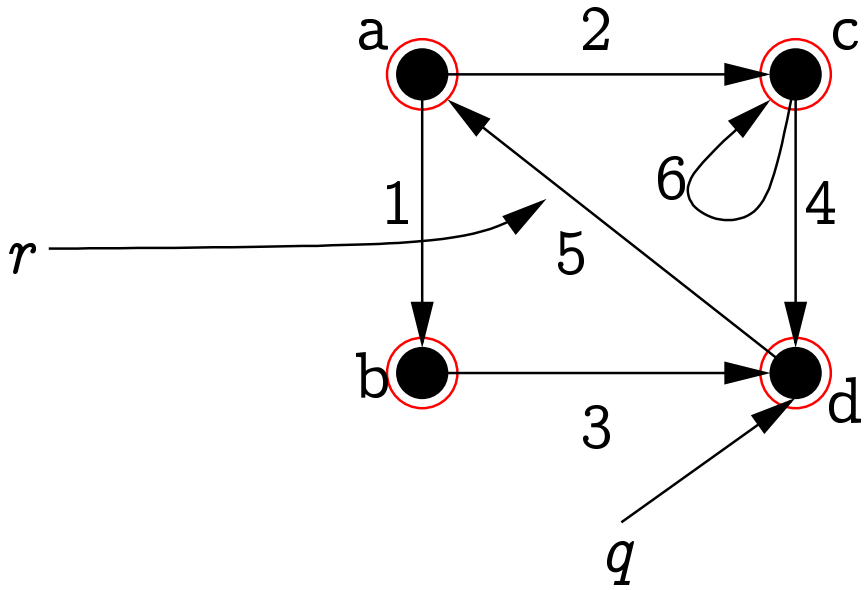
r

Q :

b

$P(\cdot)$ väljakutsed: a c d

Näide:



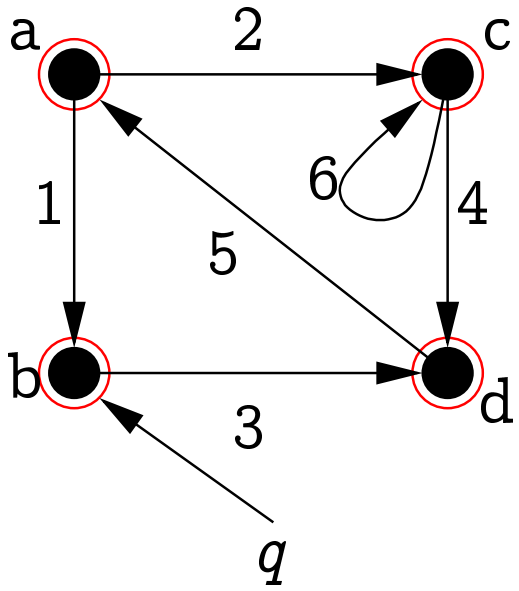
Q :

b

$P(\cdot)$ väljakutsed: a c d

Näide:

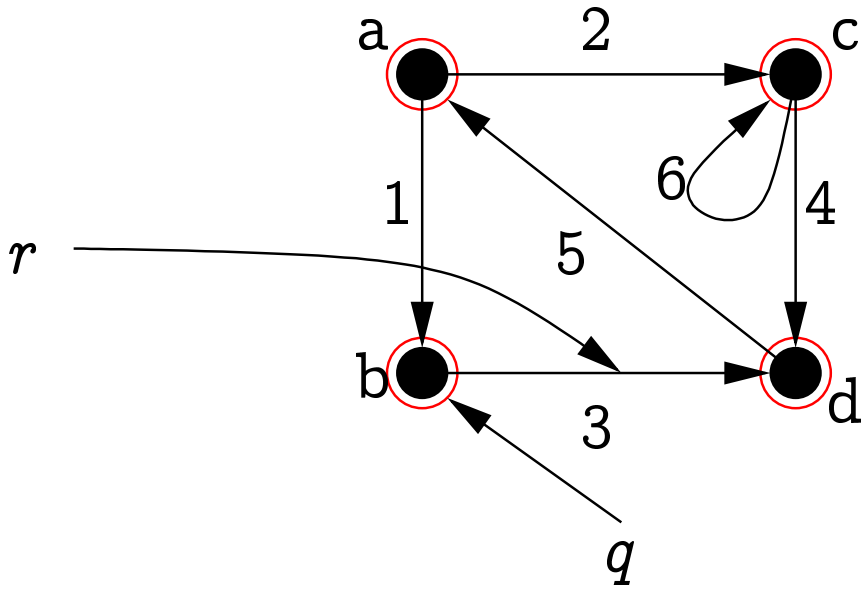
r



Q :

$P(\cdot)$ väljakutsed: a c d b

Näide:

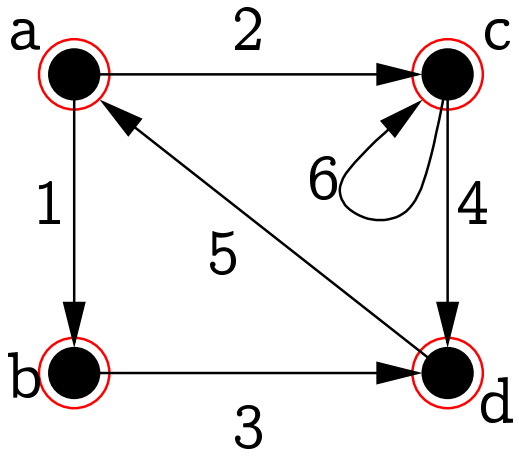


Q :

$P(\cdot)$ väljakutsed: a c d b

Näide:

r



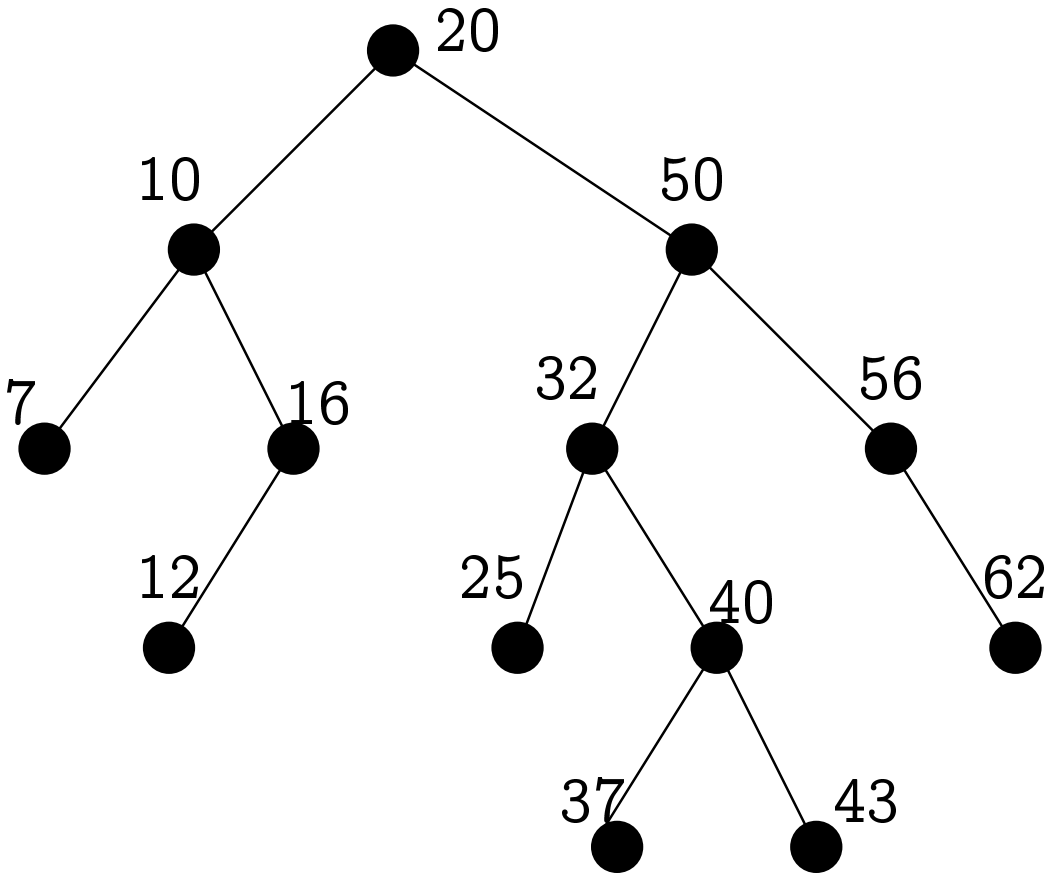
q

Q :

$P(\cdot)$ väljakutsed: a c d b

Kahendotsimise puu on kahendpuu, kus igal tipul on täiendav väli *.võti* (kuulub ossa „Andmed“), mille väärtustel on defineeritud täielik järjestus, ning kui puu ei ole tühi, siis

- juure välja *.võti* väärtus pole väiksem kui tema vasaku alampuu mistahes tipu välja *.võti* väärtus;
- juure välja *.võti* väärtus pole suurem kui tema parema alampuu mistahes tipu välja *.võti* väärtus;
- juure vasak ja parem alampuu on mõlemad kahendotsimise puud.


20
10
50
7
16
32
56
12
25
40
62
37
43

Lause. Võttes kahendotsimise puu tipud keskjärjestuses, on nende väljade *.võti* väärtused mittekahanevas järjekorras.

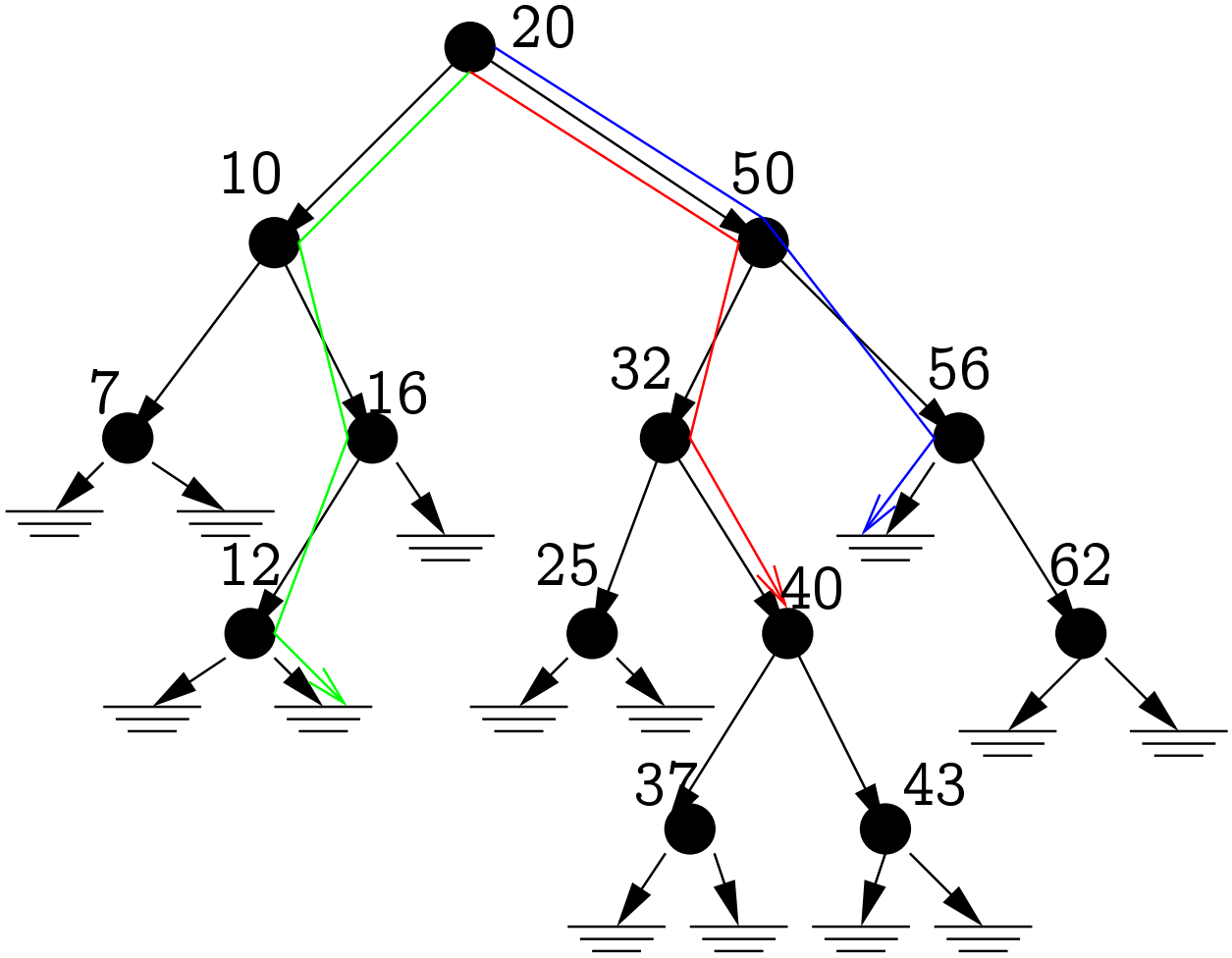
Tõestus. Induktsiooniga üle puu struktuuri. (vt. Kiho konspektist)

Kahendotsimise puust etteantud võtmeväärtusega tipu otsimine. Olgu p viit puu juurtipule ning a otsitav võtmeväärtus. Kui tippu ei leita, siis tagastatakse NIL.

Algoritm: otsi(p, a), kus $otsi(p, x)$ on

```
1  if p = NIL then
2      return NIL
3  if p.võti = x then
4      return p
5  if x < p.võti then
6      return otsi(p.vasak, x)
7  else                          --- x > p.võti
8      return otsi(p.parem, x)
```

Otsimisteed, kui otsitavaks võtmeks on 15, 40 või 52.

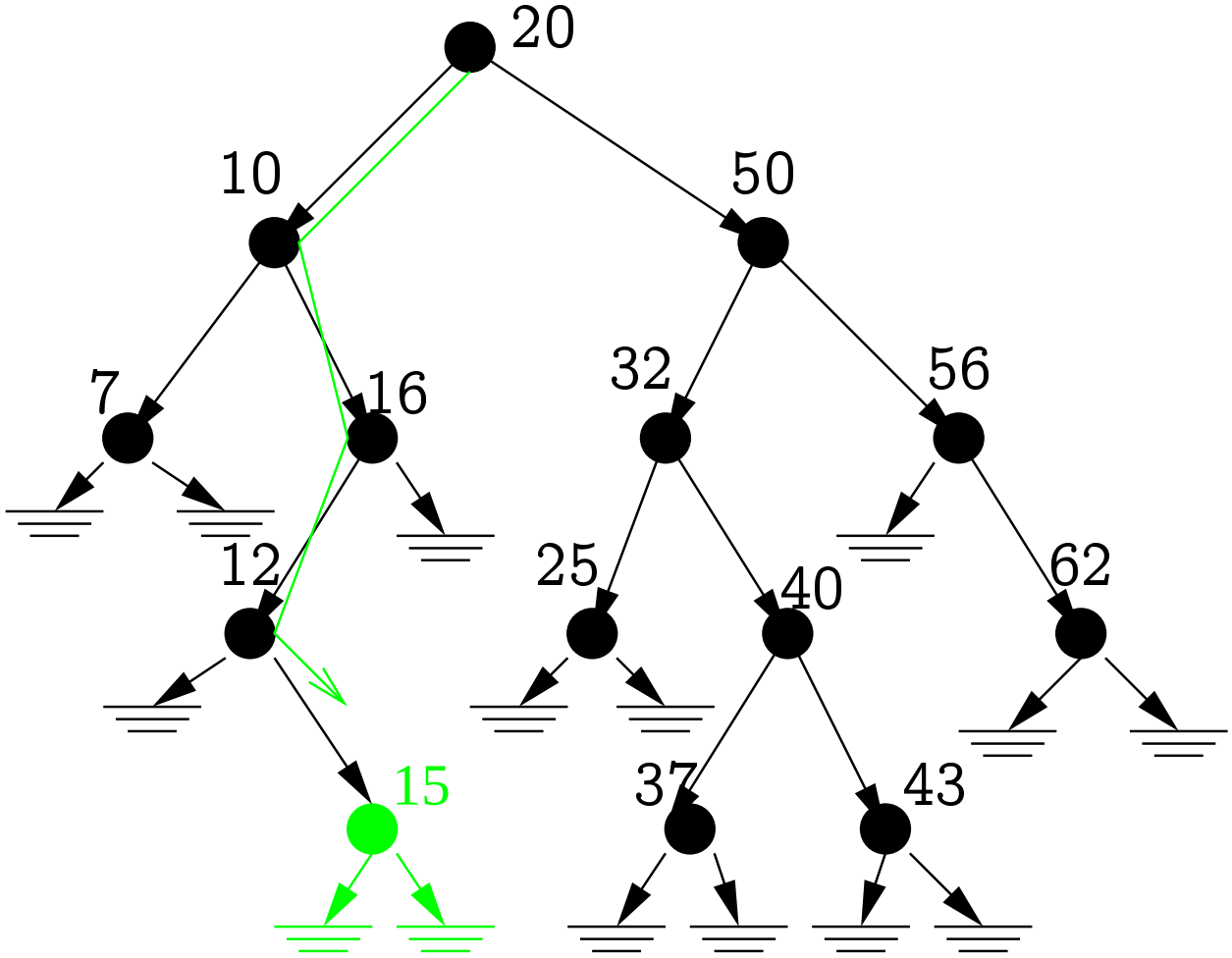


Uue kirje (samade väljadega kui osas „Andmed“) lisamine kahendotsimise puusse. Olgu p viit puu juurtipule ja R lisatav kirje. Tagastatakse lisatud tipp.

Algoritm: lisa(p, R), kus lisa(p, R) on

```
 1  if R.x < p.x then
 2      if p.vasak = NIL then
 3          q := new(tipukirje); q.Andmed := R; p.vasak := q
 4          return q
 5      else
 6          return lisa(p.vasak, R)
 7  else
 8      if p.parem = NIL then
 9          q := new(tipukirje); q.Andmed := R; p.parem := q
10          return q
11      else
12          return lisa(p.parem, R)
```

Tipu võtmeväärtusega 15 lisamine kahendotsimise puusse.



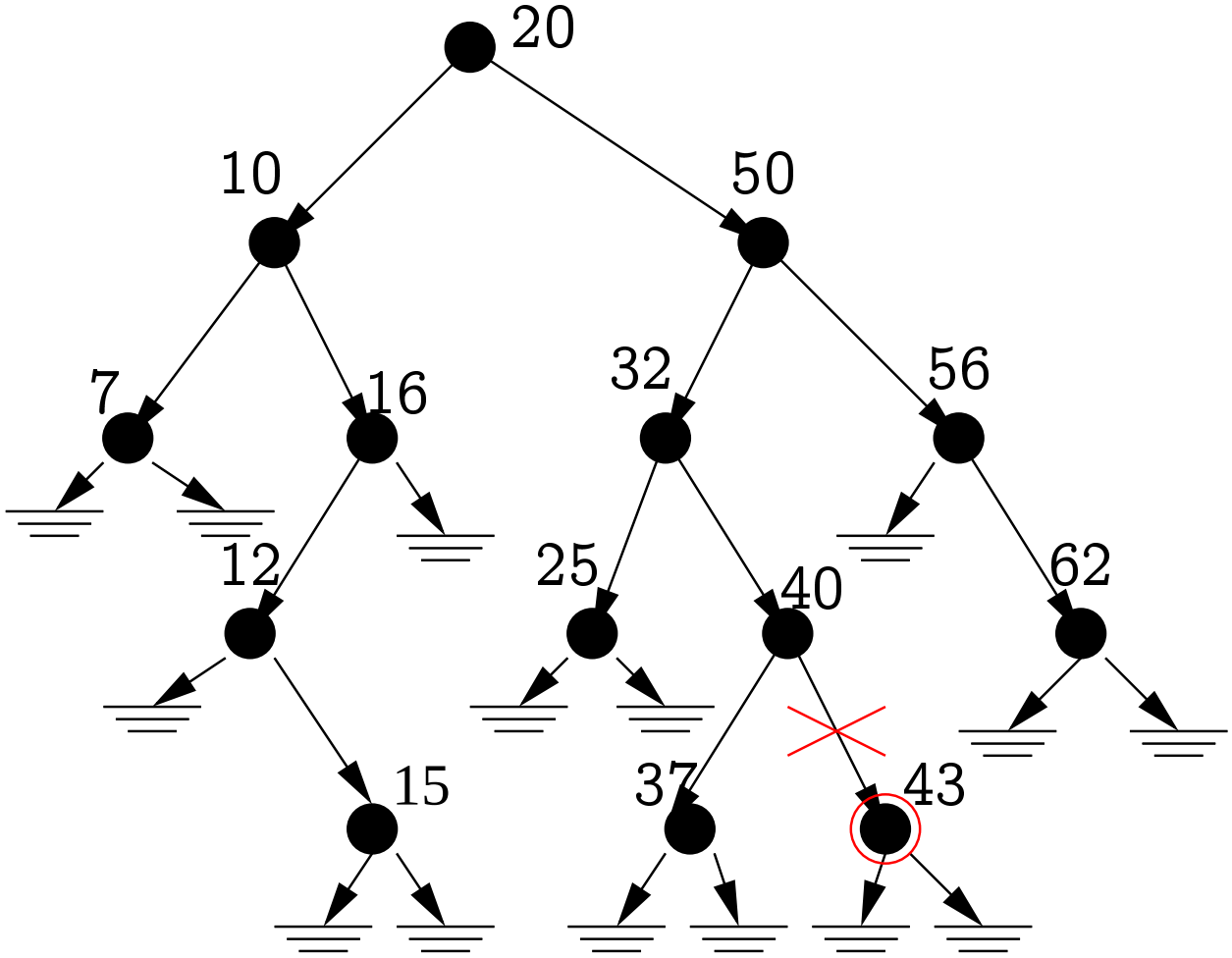
Tipu eemaldamine kahendotsimise puust.

Olgu tippudel väli $.\ddot{u}lem$ (ei kuulu ossa „Andmed“), mis viitab käesoleva tipu ülemusele.

Lisamisalgoritmis tuleb siis uue tipu väli $.\ddot{u}lem$ ka initsialiseerida, s.t. 3. ja 9. rida tuleb täiendada omistamisega $q.\ddot{u}lem := p$.

Olgu p viit vaadeldava kahendpuu juurtipule ja q viit eemaldatavale tipule. Algoritm eemalda(p, q) kustutab puust kirje q.Andmed. Ta tagastab väärtuste paari: viida muudetud puu juurtipule ning viida tegelikult eemaldatud tipule.

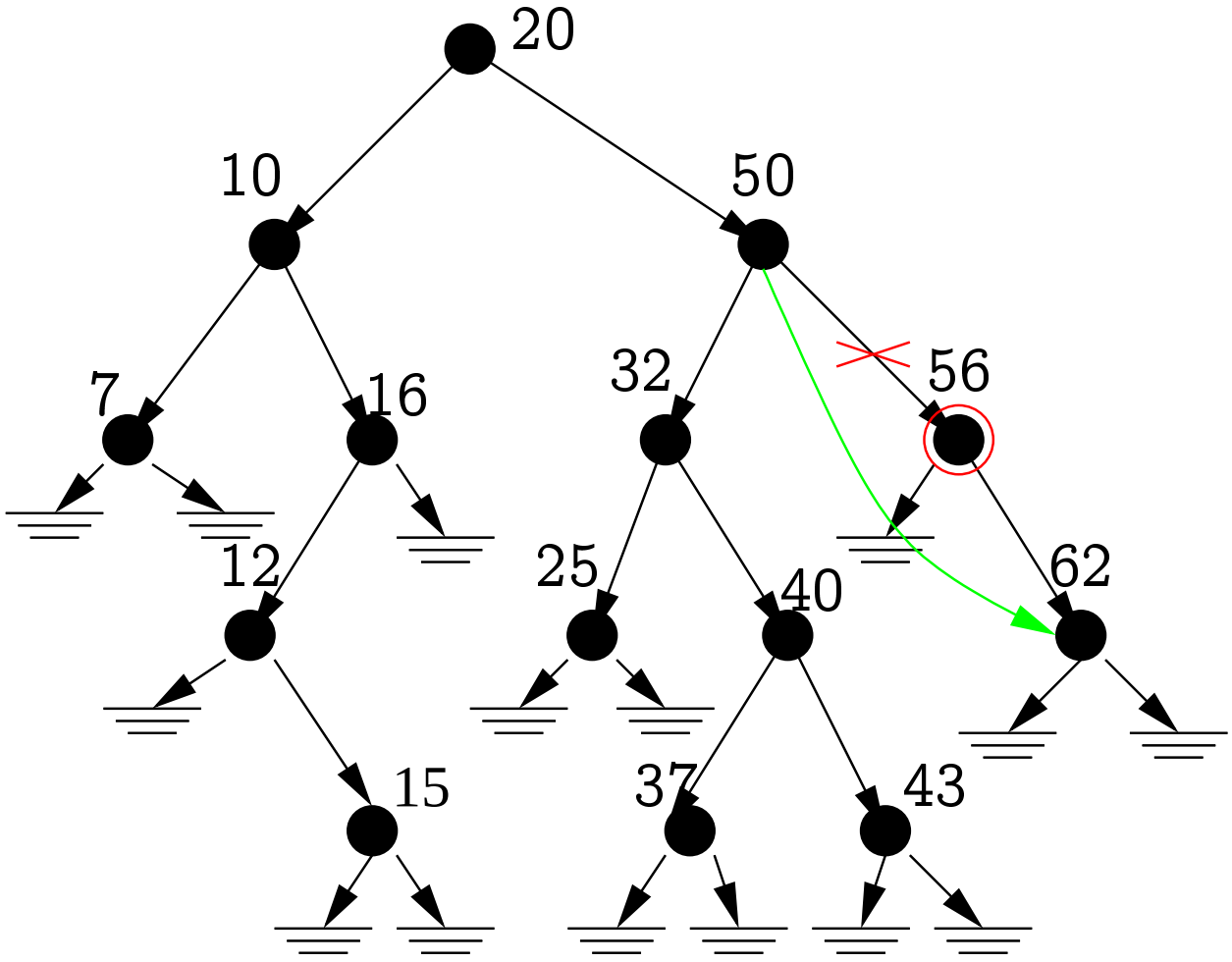
Kui tipul q pole alluvaid, siis lihtsalt kustutame ta.



eemalda(p, q) on

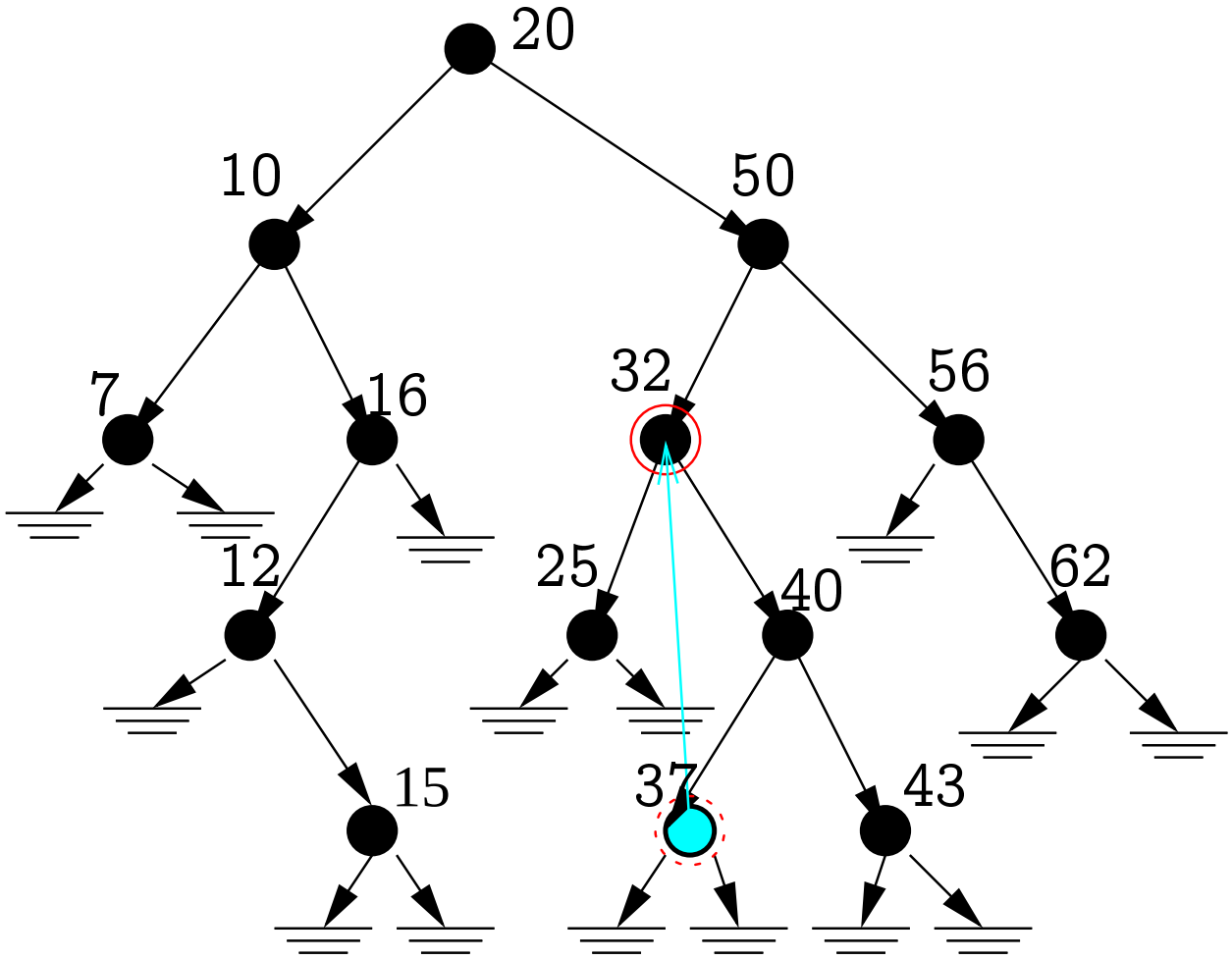
```
1   if q.vasak = NIL and q.parem = NIL then
2       if q.ülem = NIL
3           return (NIL, q)
4       else if q.ülem.vasak = q
5           q.ülem.vasak := NIL
6       else
7           q.ülem.parem := NIL
8       return (p, q)
    ...
```

Kui tipul q on üks alluv, siis paneme q ülemuse viitama sellele alluvale.



```
 9   else if (q.vasak = NIL and q.parem ≠ NIL)
           or (q.vasak ≠ NIL and q.parem = NIL) then
10     r := q.vasak = NIL ? q.parem : q.vasak
11     if q.ülem = NIL
12       return (r, q)
13     else if q.ülem.vasak = q
14       q.ülem.vasak := r
15     else
16       q.ülem.parem := r
17     return (p, q)
     ...
```

Kui tipul q on mõlemad alluvad, siis leiame q-st keskjärjestuses järgmise tipu r, kopeerime tema andmed tippu q ja kustutame hoopis r-i.



Keskjärjestuses järgmine tipp: parempoolse alampuu vasakpoolseim tipp.

```
18  else
19      r := q.parem
20      while r.vasak ≠ NIL do
21          r := r.vasak
22      q.Andmed := r.Andmed
23      return eemalda(p, r).
```

Kahendotsimise puu operatsioonide ajaline keerukus on $\Theta(h)$, kus h on puu kõrgus.

Puus kõrgusega h on vähemalt h ja ülimalt $2^h - 1$ tippu.

Seega on kahendpuu operatsioonide ajaline keerukus halvimal juhul $O(n)$, kus n on tippude arv.

Tahaksime keerukust $O(\log n)$. Selleks tuleb puud peale muutmist tasakaalustada.

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

Lühend „AVL“ tuleb autorite nimedest.

Olgu A_h minimaalne tippude arv AVL-puus kõrgusega h. Siis $A_1 = 1$, $A_2 = 2$ ja $A_h = A_{h-1} + A_{h-2} + 1$.

Fibonacci arvud: $F_0 = 0$, $F_1 = 1$, $F_h = F_{h-1} + F_{h-2}$. Seega $A_h = F_{h+2} - 1$. (tõestus induktsiooniga üle h)

Kuna $F_h = \left[\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^h \right]$, kus $[\cdot]$ tähistab ümardamist täisarvuni, siis $A_h = 2^{\Theta(h)}$ ja 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.

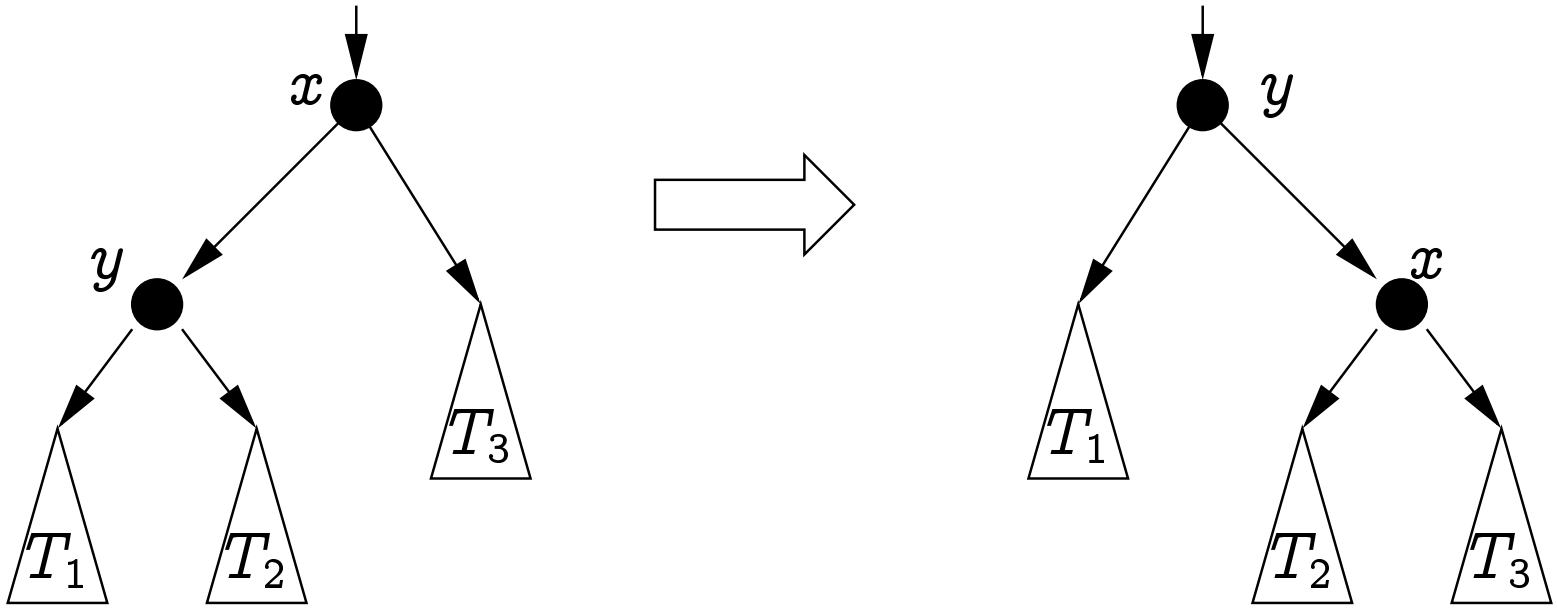
Peale lisamist võivad valed kõrgused olla tippudel, mis asuvad ahelal lisatud tipust juureni.

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:

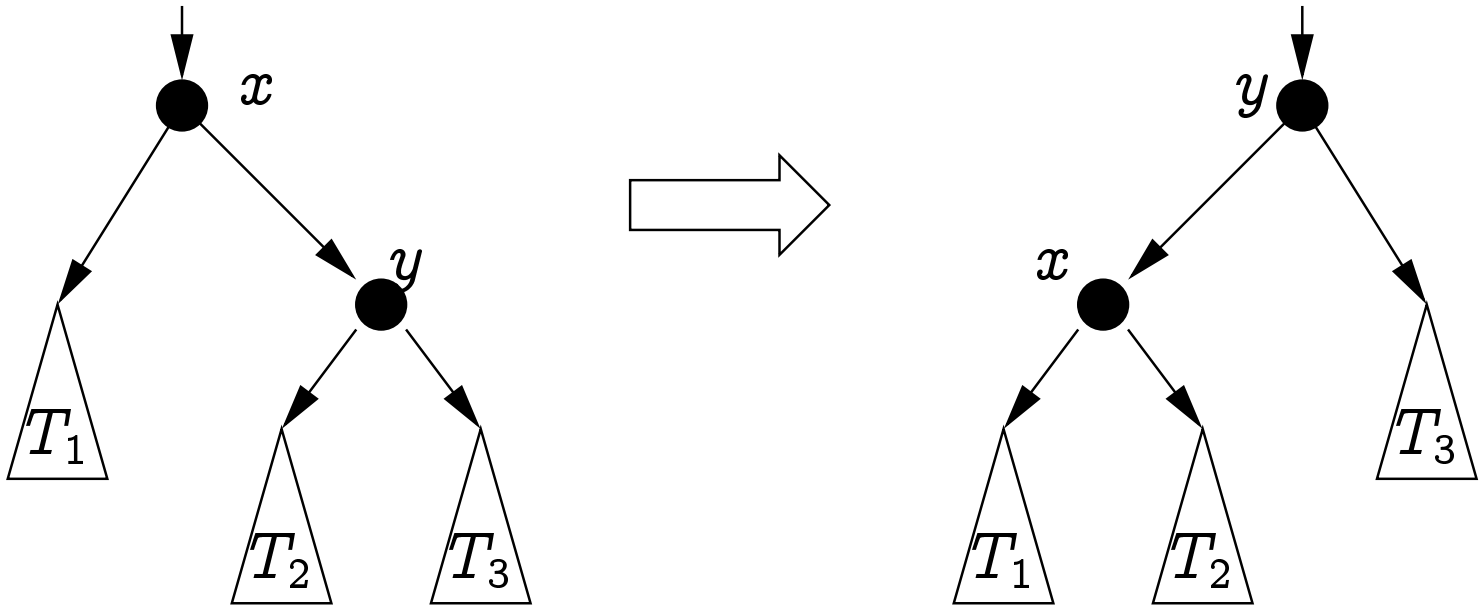


pööre_paremale tagastab muudetud puu juurtipu.

pöörе_paremale(p, x) on

```
1   y := x.vasak; u := x.ülem
2   T1 := y.vasak; T2 := y.parem; T3 := x.parem
3   y.vasak := T1; y.parem := x; x.vasak := T2; x.parem := T3
4   if u = NIL then
5       return y
6   else
7       (u.vasak = x ? u.vasak : u.parem) := y
8       return p
```

pööre_vasakule on vastupidine operatsioon.



pööre_vasakule tagastab muudetud puu juurtipu.

pööre_vasakule(p, x) on

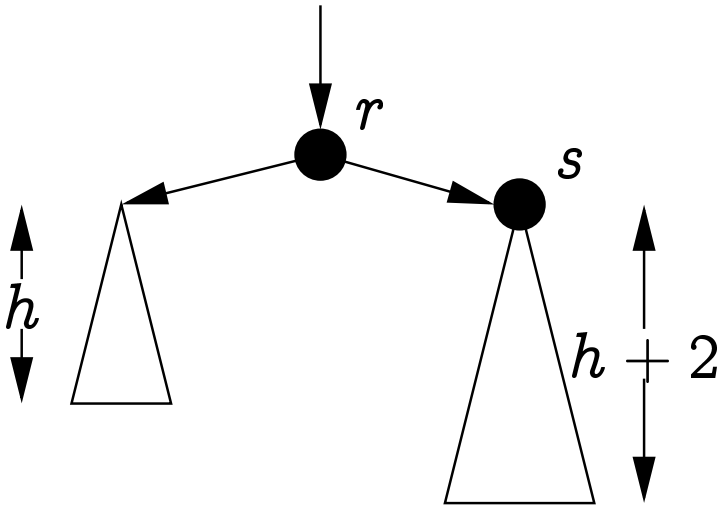
```
1  y := x.parem; u := x.ülem
2  T1 := x.vasak; T2 := y.vasak; T3 := y.parem
3  y.vasak := x; y.parem := T3; x.vasak := T1; x.parem := T2
4  if u = NIL then
5     return y
6  else
7     (u.vasak = x ? u.vasak : u.parem) := y
8     return p
```

Tipu lisamisel AVL-puusse teeme kõigepealt tavalise lisamise ja hakkame lisatud tipust alates kõrgusi parandama, kuni jõuame tasakaalustamata tipuni...

lisaAVL(p, R), mis tagastab paari uuest puu juurest ja lisatud tipust, on

```
1  q := lisa(p, R)
2  r := q
3  while r ≠ NIL do
4      h_v := r.vasak = NIL ? 0 : r.vasak.kõrgus
5      h_p := r.parem = NIL ? 0 : r.parem.kõrgus
6      if |h_v − h_p| = 2 then break
7      r.kõrgus := max(h_v, h_p) + 1
8      r := r.ülem
9  if r = NIL then return (p, q)
   ...
```

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

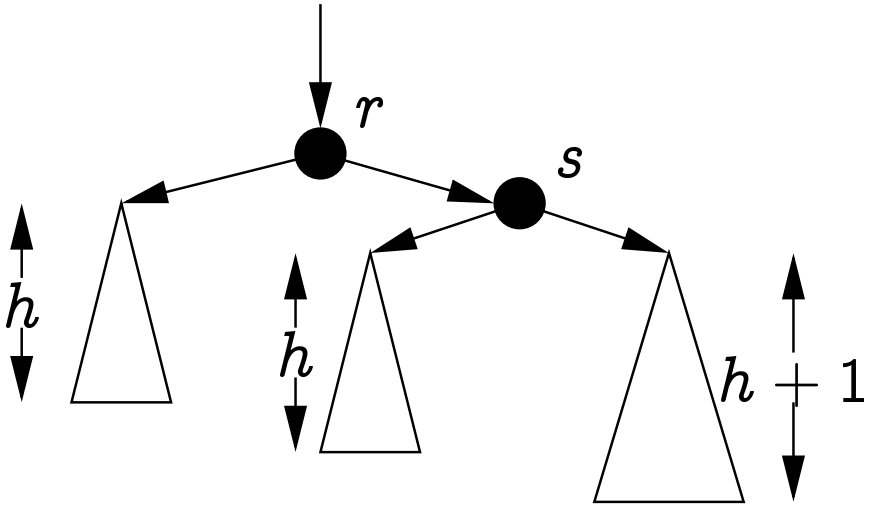


r-i kõrgus: $h + 3$

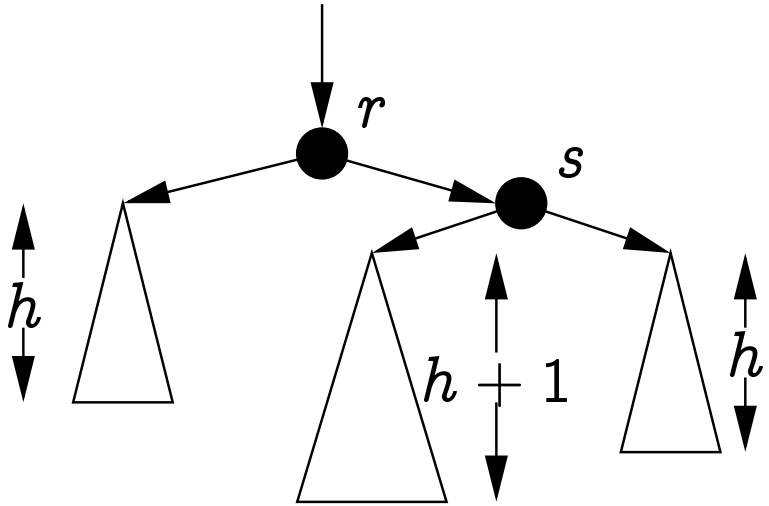
(enne lisamist: $h + 2$)

```
10   if h_p > h_v then
11       s := r.parem
     ...
```

Uus tipp lisati paremasse alampuusse. Kui s-i vasak ja parem alampuu oleks peale lisamist võrdse kõrgusega, siis oleks üks alampuudest juba enne lisamist selle kõrgusega olnud ja s-i kõrgus poleks muutunud. Seega oleks r juba enne lisamist tasakaalustamata olnud. Järelikult on kaks varianti:



1. variant



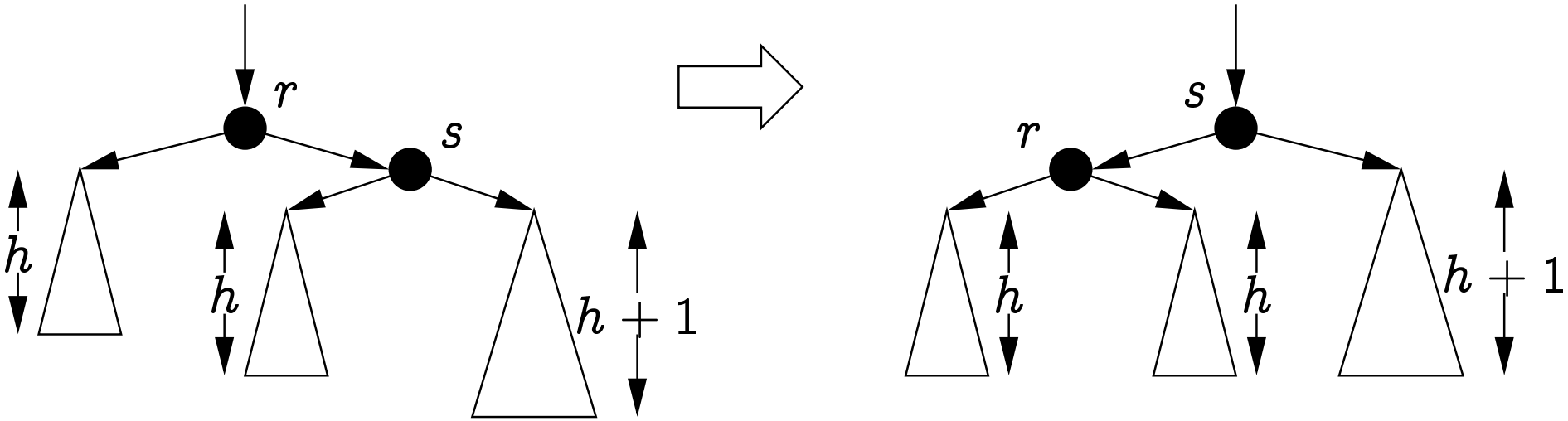
2. variant

12 $\quad h_{sv} := s.vasak = \text{NIL} \; ? \; 0 : s.vasak.kõrgus$

13 $\quad h_{sp} := s.parem = \text{NIL} \; ? \; 0 : s.parem.kõrgus$

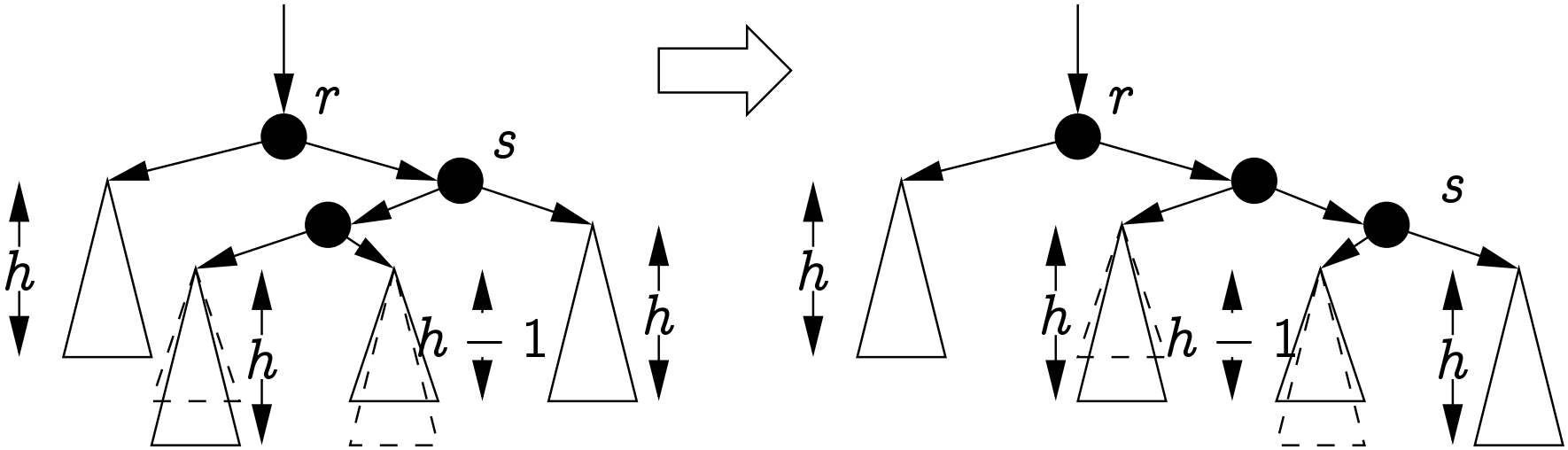
$\quad \ldots$

1. variandil pöörame r-i vasakule.

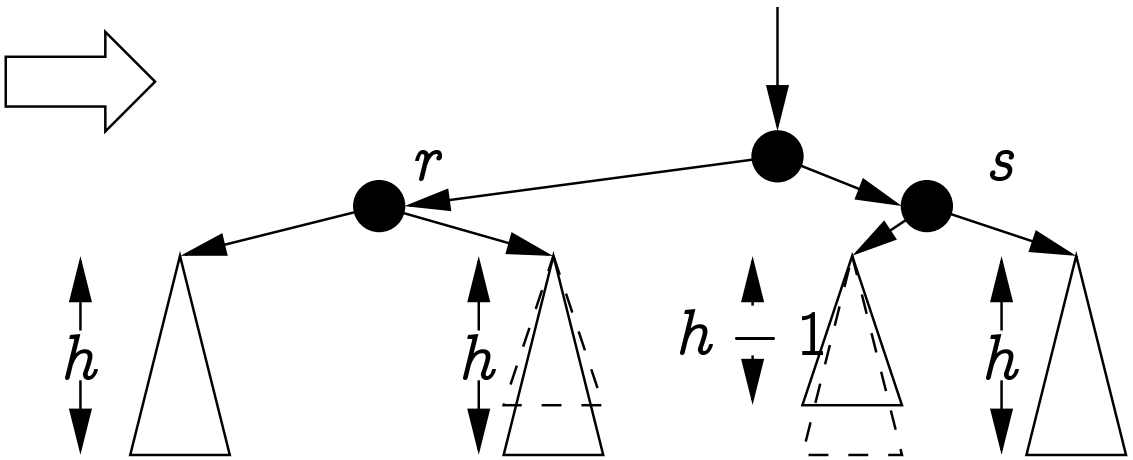


Kogu vaadeldava alampuu kõrgus on nüüd $h + 2$ — sama, mis enne uue tipu lisamist. Kõrgemalasuvate tippude kõrgused seega ei muutunud — puu on tasakaalus.

2. variandil pöörame s-i paremale



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



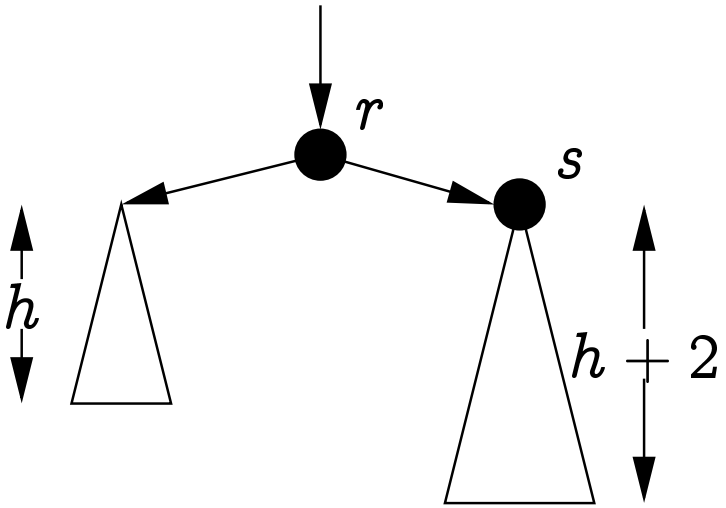
```
14      if h_sv > h_sp then
15          p := pööra_paremale(p, s); s.kõrgus := h_v + 1
16      p := pööra_vasakule(p, r)
17      r.kõrgus := h_v + 1; r.ülem.kõrgus := h_v + 2
18      return (p, q)
19  else                              --- h_v > h_p
20      s := r.vasak
21      h_sv := s.vasak = NIL ? 0 : s.vasak.kõrgus
22      h_sp := s.parem = NIL ? 0 : s.parem.kõrgus
23      if h_sv < h_sp then
24          p := pööra_vasakule(p, s); s.kõrgus := h_p + 1
25      p := pööra_paremale(p, r)
26      r.kõrgus := h_p + 1; r.ülem.kõrgus := h_p + 2
27      return (p, q)
```

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

eemaldaAVL(p, q) on

1 $(p, q) :=$ eemalda(p, q)
2 $r := q.\ddot{u}lem$
3 **while** $r \neq$ NIL **do**
4 $\quad h_v := r.vasak =$ NIL ? 0 : $r.vasak.k\tilde{o}rgus$
5 $\quad h_p := r.parem =$ NIL ? 0 : $r.parem.k\tilde{o}rgus$
6 $\quad$ **if** $|h_v - h_p| = 2$ **then break**
7 $\quad r.k\tilde{o}rgus := \max(h_v, h_p) + 1$
8 $\quad r := r.\ddot{u}lem$
9 **if** $r =$ NIL **then return** (p, q)
$\quad \ldots$

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

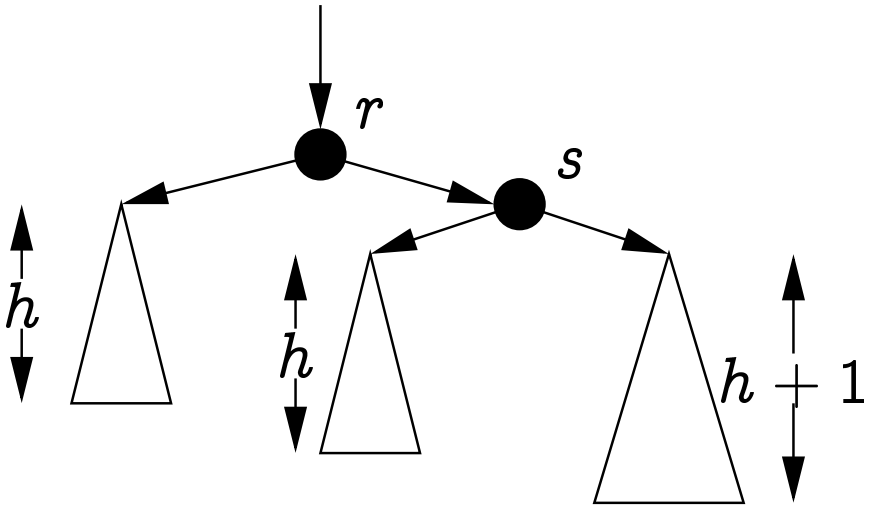


r-i kõrgus: $h + 3$

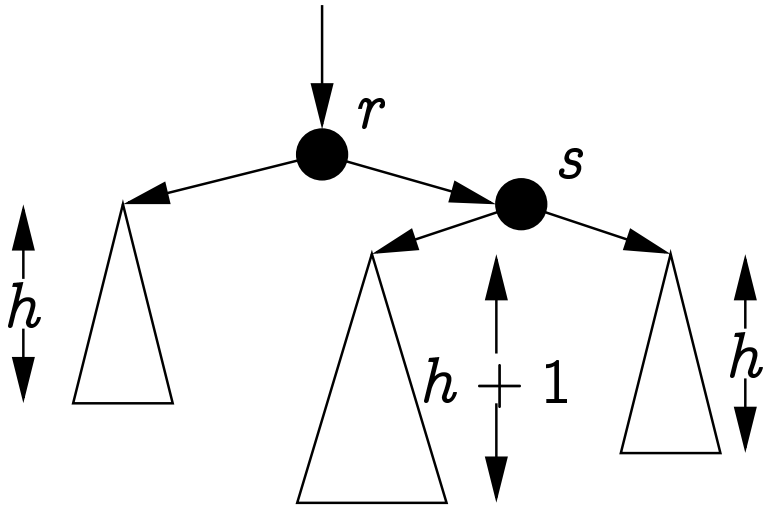
(enne eemaldamist: $h + 3$)

```
10   if h_p > h_v then
11       s := r.parem
     ...
```

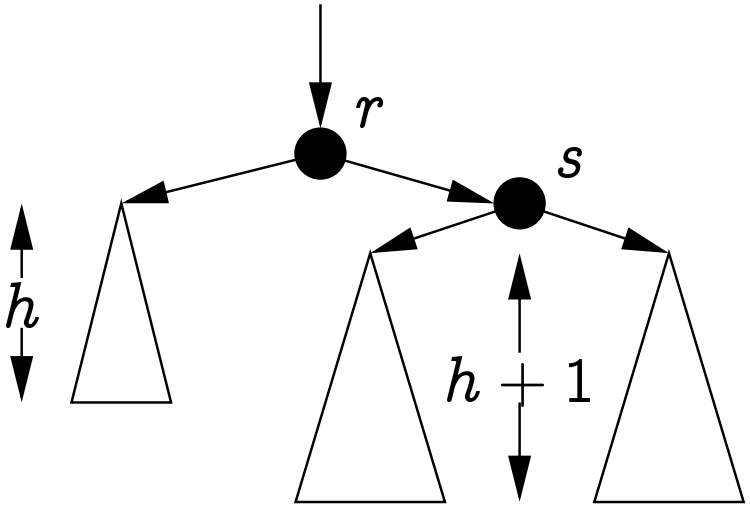
Tipp eemaldati vasakust alampuust. Kolm varianti:



1. variant



2. variant



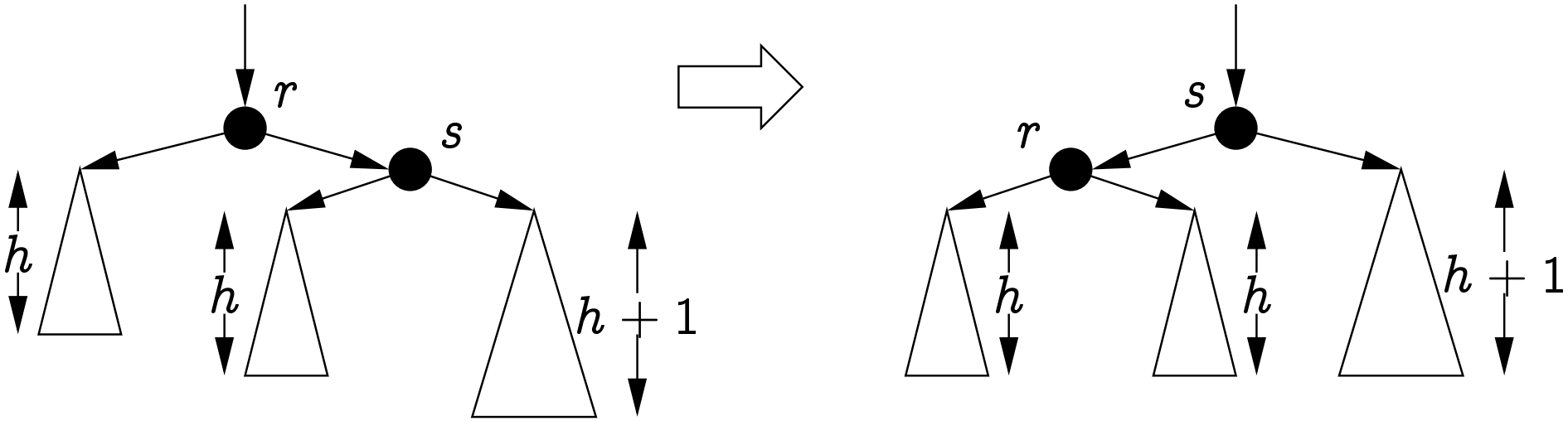
3. variant

12 $h_{sv} := s.vasak = \text{NIL} \; ? \; 0 : s.vasak.kõrgus$

13 $h_{sp} := s.parem = \text{NIL} \; ? \; 0 : s.parem.kõrgus$

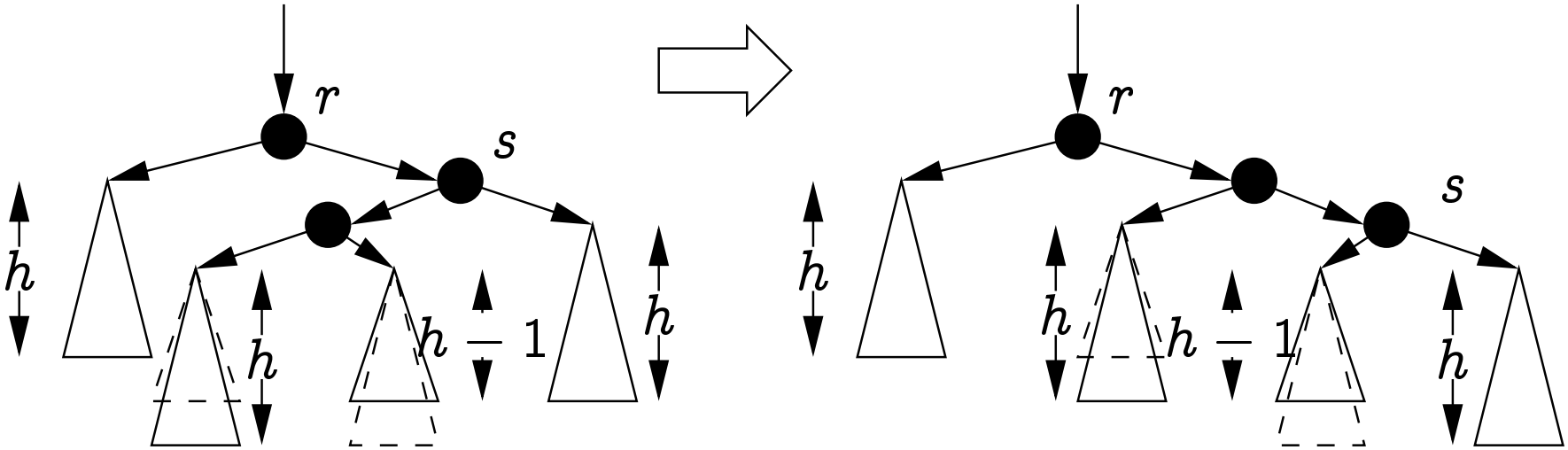
...

1. variandil pöörame r-i vasakule.

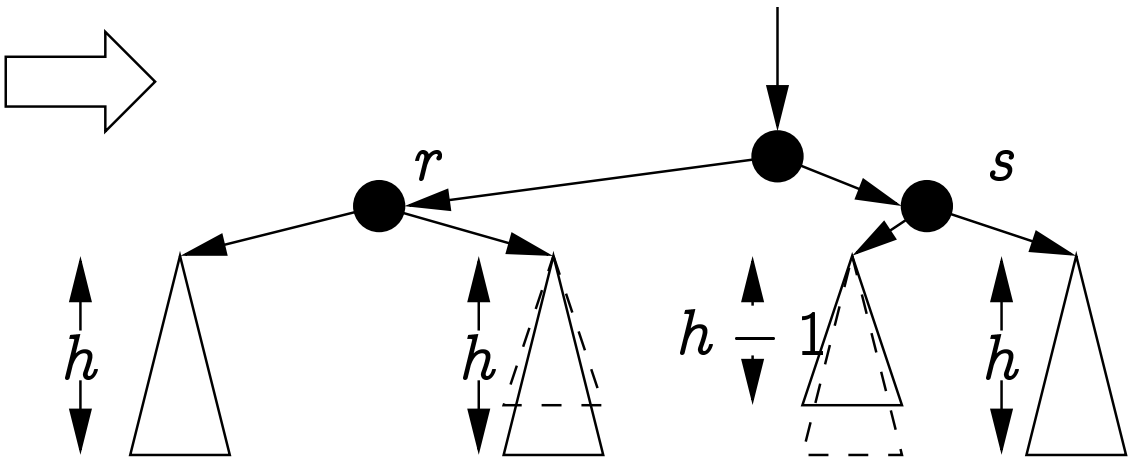


Kogu vaadeldava alampuu kõrgus on nüüd $h+2$ — vähenes ühe võrra. Seega peame me jätkame juure pool asuvate tippude tasakaalustamisi.

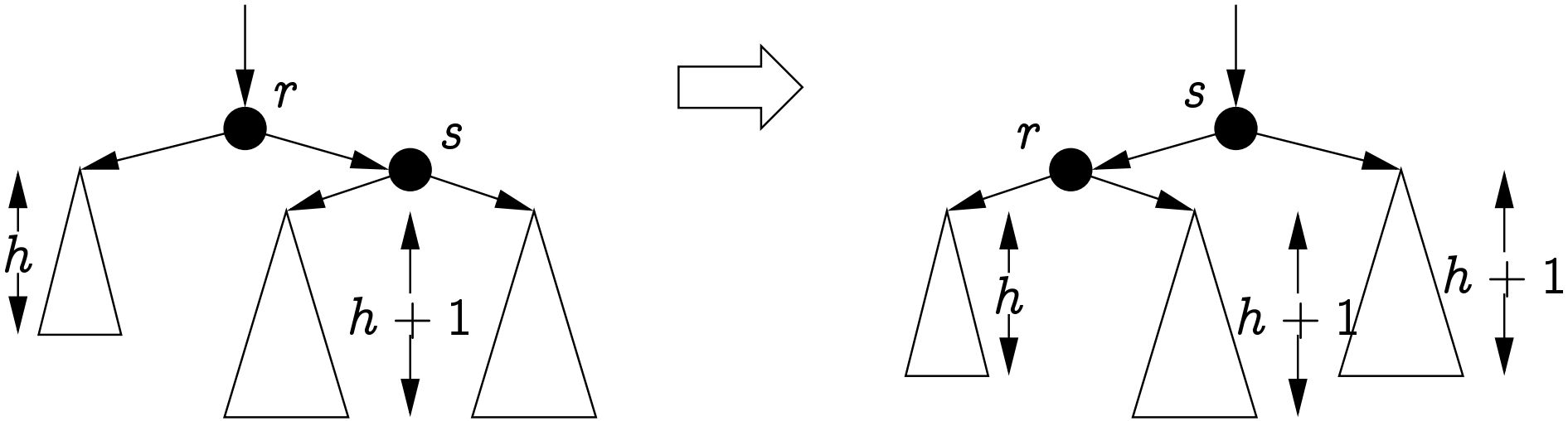
2. variandil pöörame s-i paremale



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.

```
14      if h_sv = h_sp then
15          p := pööra_vasakule(p, r)
16          r.kõrgus := h_v + 2; s.kõrgus := h_v + 3
17          return (p, q)
18      if h_sv > h_sp then
19          p := pööra_paremale(p, s); s.kõrgus := h_v + 1
20      p := pööra_vasakule(p, r)
21      r.kõrgus := h_v + 1; r.ülem.kõrgus := h_v + 2
22      r := r.ülem.ülem; goto 3
    ...
```

```
23  else                                  --- h_v > h_p
24      s := r.vasak
25      h_sv := s.vasak = NIL ? 0 : s.vasak.kõrgus
26      h_sp := s.parem = NIL ? 0 : s.parem.kõrgus
27      if h_sv = h_sp then
28          p := pööra_paremale(p, r)
29          r.kõrgus := h_p + 2; s.kõrgus := h_p + 3
30          return (p, q)
31      if h_sv < h_sp then
32          p := pööra_vasakule(p, s); s.kõrgus := h_p + 1
33      p := pööra_paremale(p, r)
34      r.kõrgus := h_p + 1; r.ülem.kõrgus := h_p + 2
35      r := r.ülem.ülem; goto 3
```