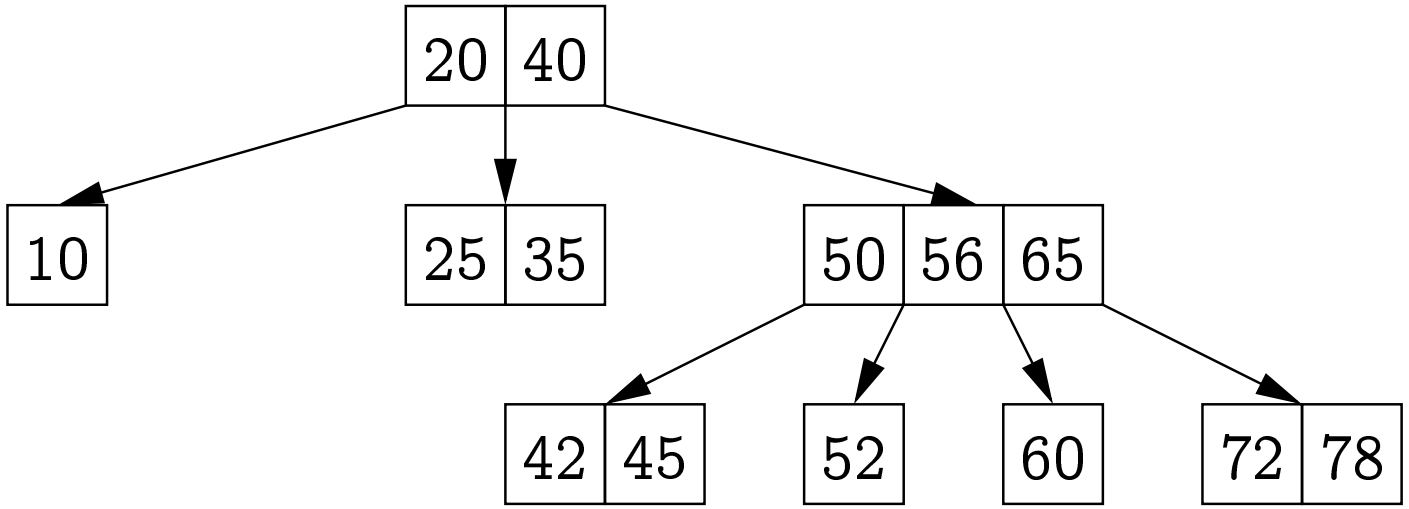
T on *m-rajaline otsimispuu*, kui

- Igas tipus on 0 kuni $m-1$ kirjet võtmetega $k_1 \leqslant k_2 \leqslant \cdots \leqslant k_j$ ning kui see tipp ei ole leht, siis $j+1$ alluvat, mis on alampuude $T_0, \ldots, T_j$ juurteks.
- Kui k on võti alampuus T_0, siis $k \leqslant k_1$.
- Kui k on võti alampuus T_j, siis $k \geqslant k_j$.
- Kui k on võti alampuus T_i $(1 \leqslant i \leqslant j-1)$, siis $k_i \leqslant k \leqslant k_{i+1}$.



Otsimine m-rajalises otsimispuus on sarnane otsimisega kahendotsimise puus.

m-järku B-puu on m-rajaline otsimispuu, kus

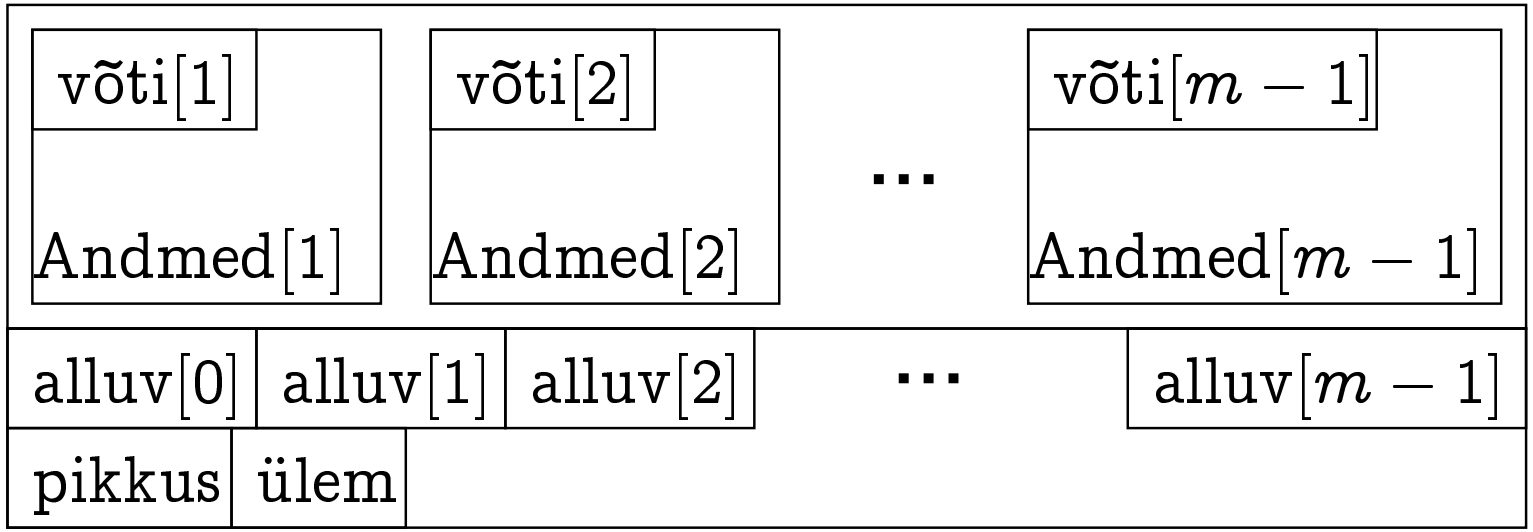
- m on paaritu;
- kõik lehed on juurest samal kaugusel;
- kõigis tippudes, v.a. juures on vähemalt $\lfloor m/2 \rfloor$ kirjet;
- juures on vähemalt üks kirje.

2-3 puu on 3. järku B-puu.

- Sama kasutusvaldkond, mis AVL-puul.

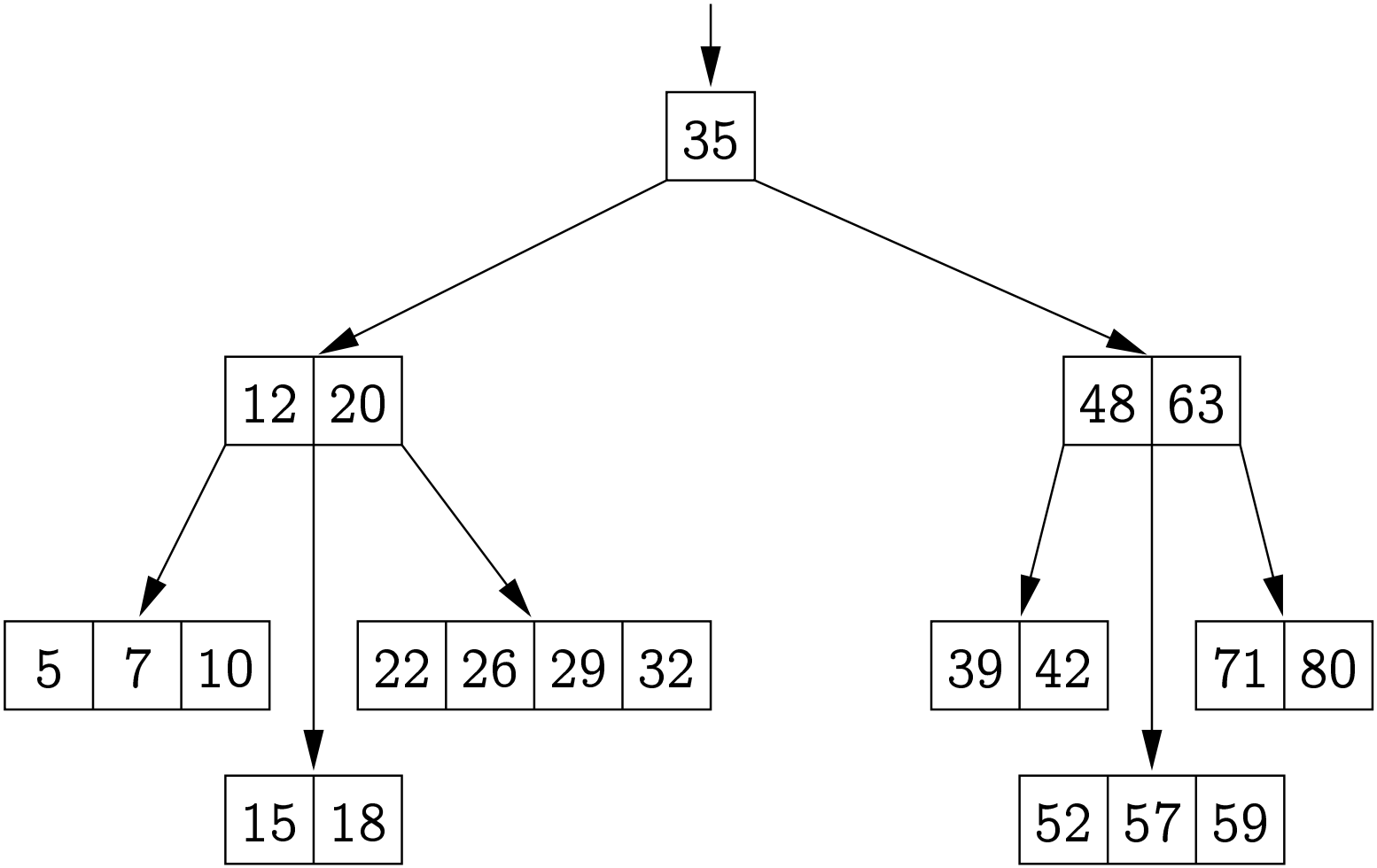
m-järku B-puus kõrgusega h on vähemalt $2\frac{(m/2)^h - 1}{m/2 - 1} = 2^{\Theta(h)}$ tippu.

Kujutagu m-järku B-puu tippu järgmine kirje:



Siin *.pikkus* näitab, mitu kirjet tipus tegelikult on.

Näide 5. järku B-puust



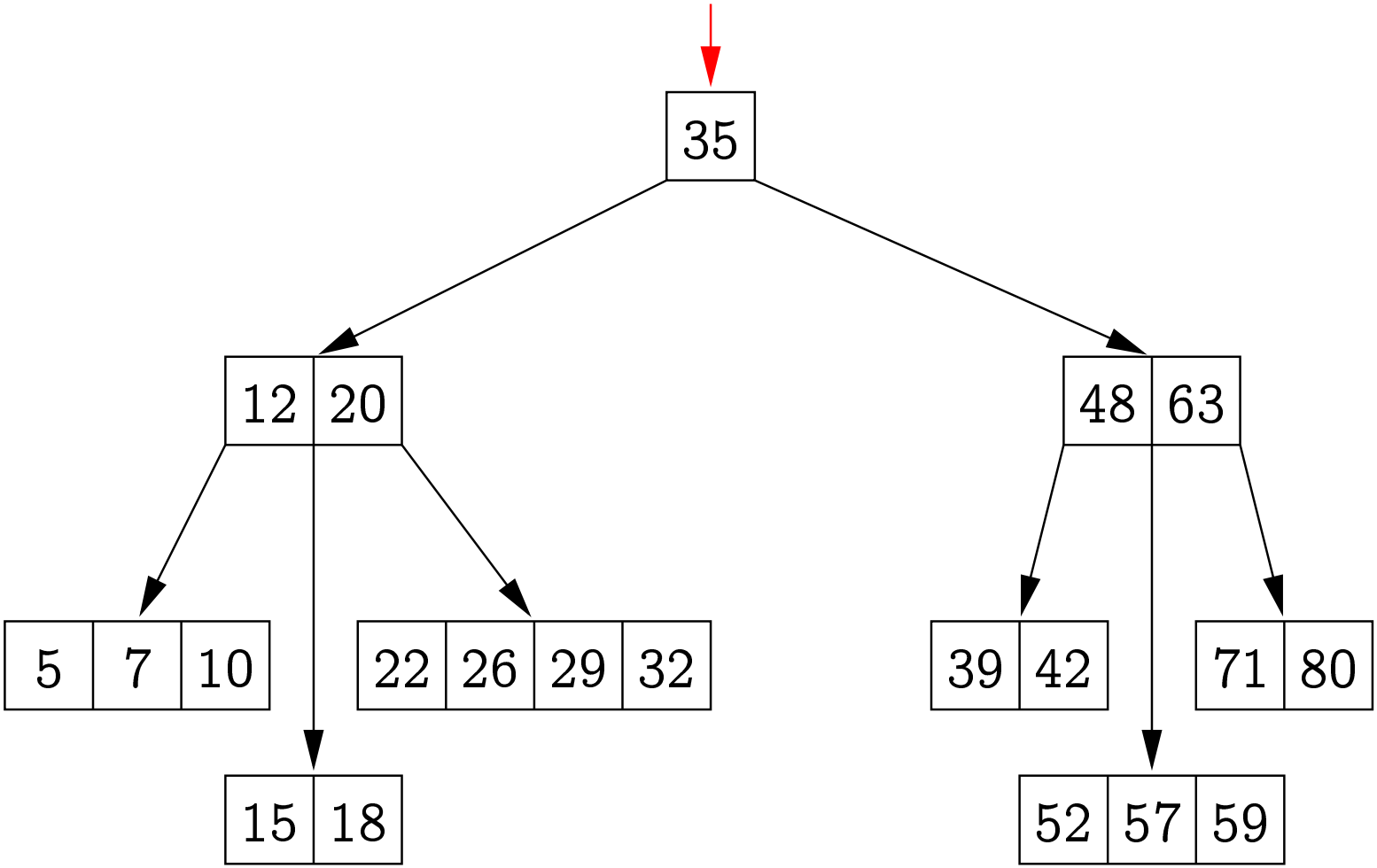
Otsimine B-puus on sarnane otsimisega kahendpuus.

Kui otsitavat võtit x antud tipus p ei ole, siis võrdleme x-i väärtustega $p.võti[1]$ kuni $p.võti[m-1]$.

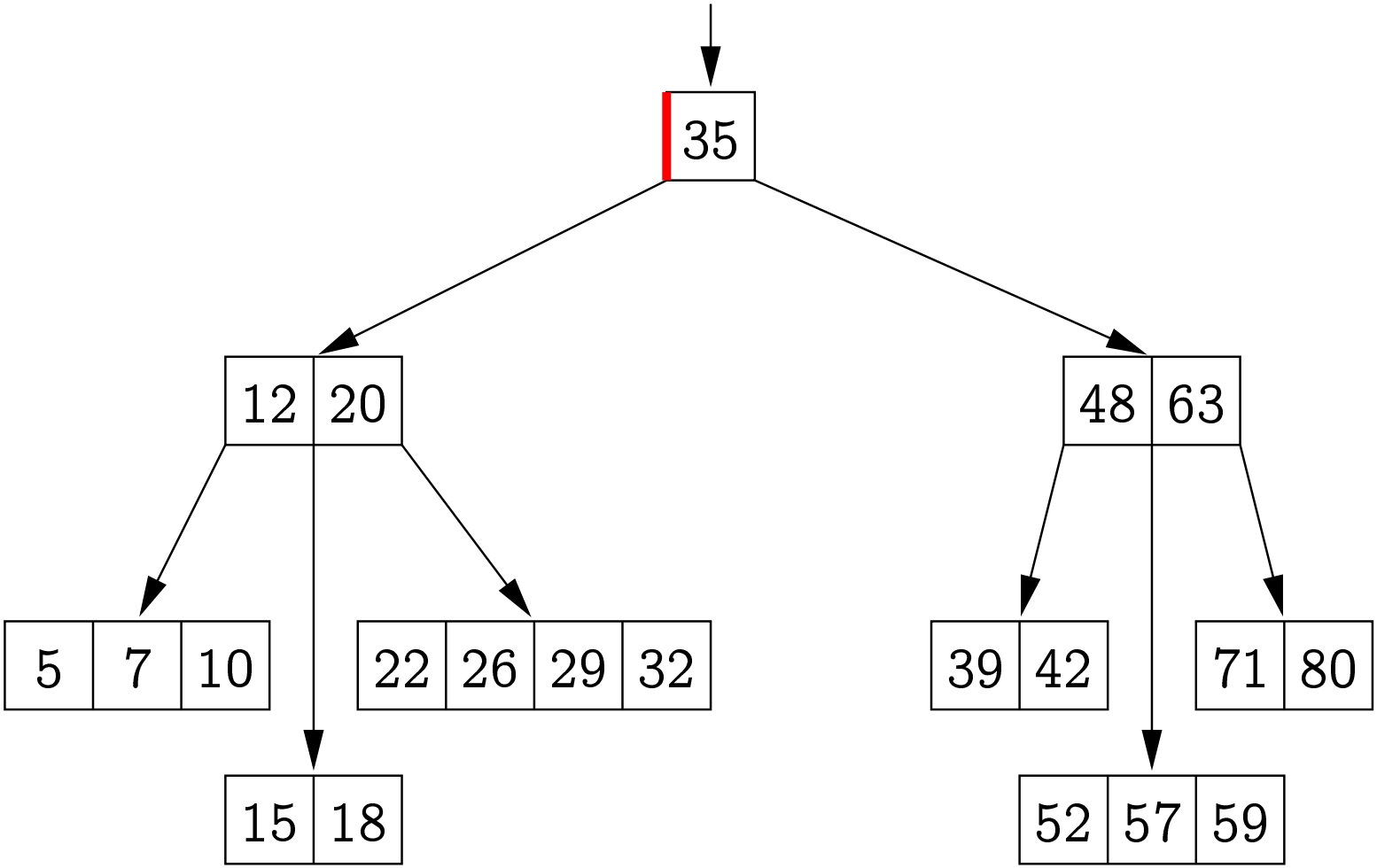
Keerukus: $O(m)$, aga m on konstant.

Võrdlemistulemus määrab, millisest alluvast otsimist jätkame.

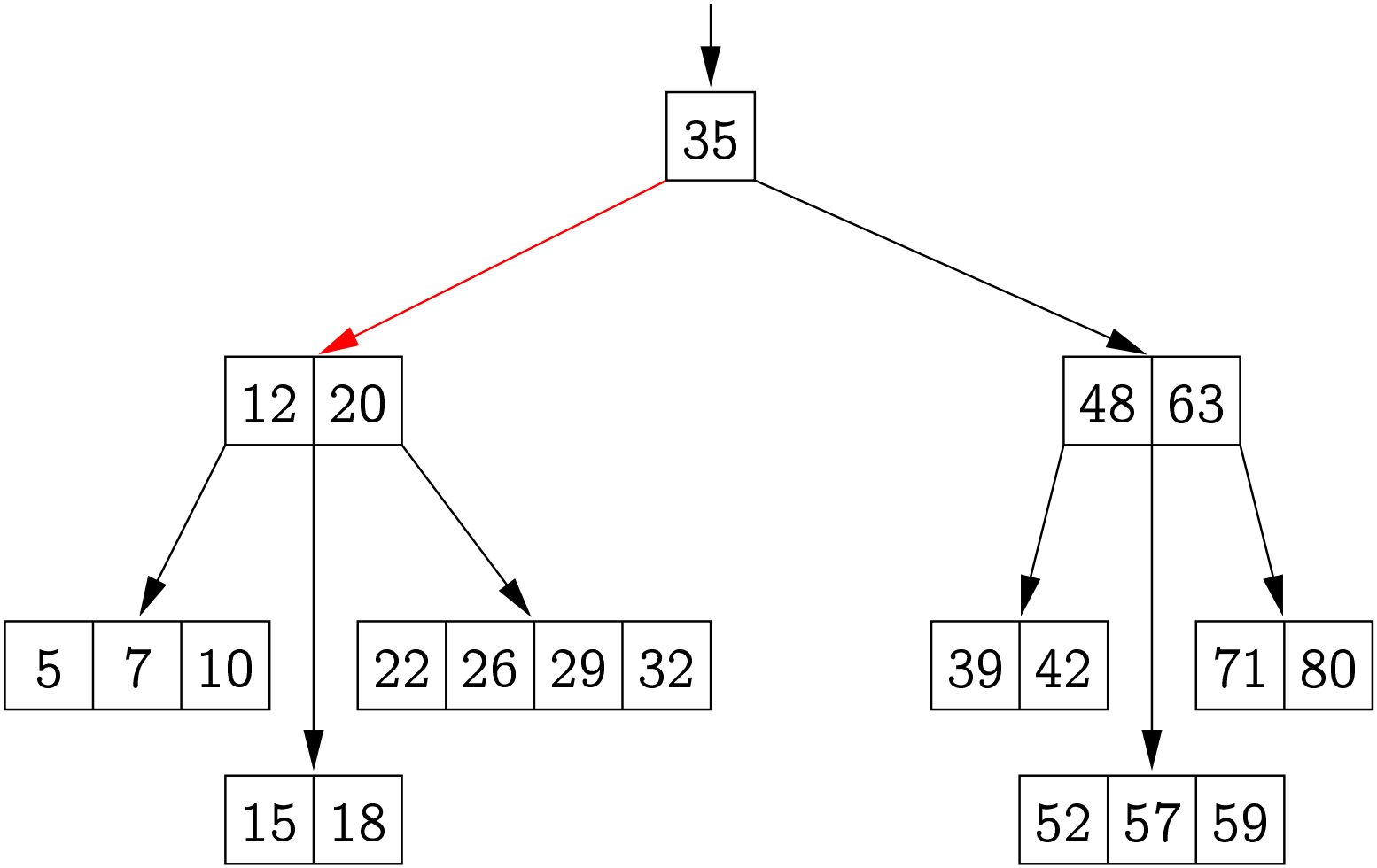
Otsime võtit 18



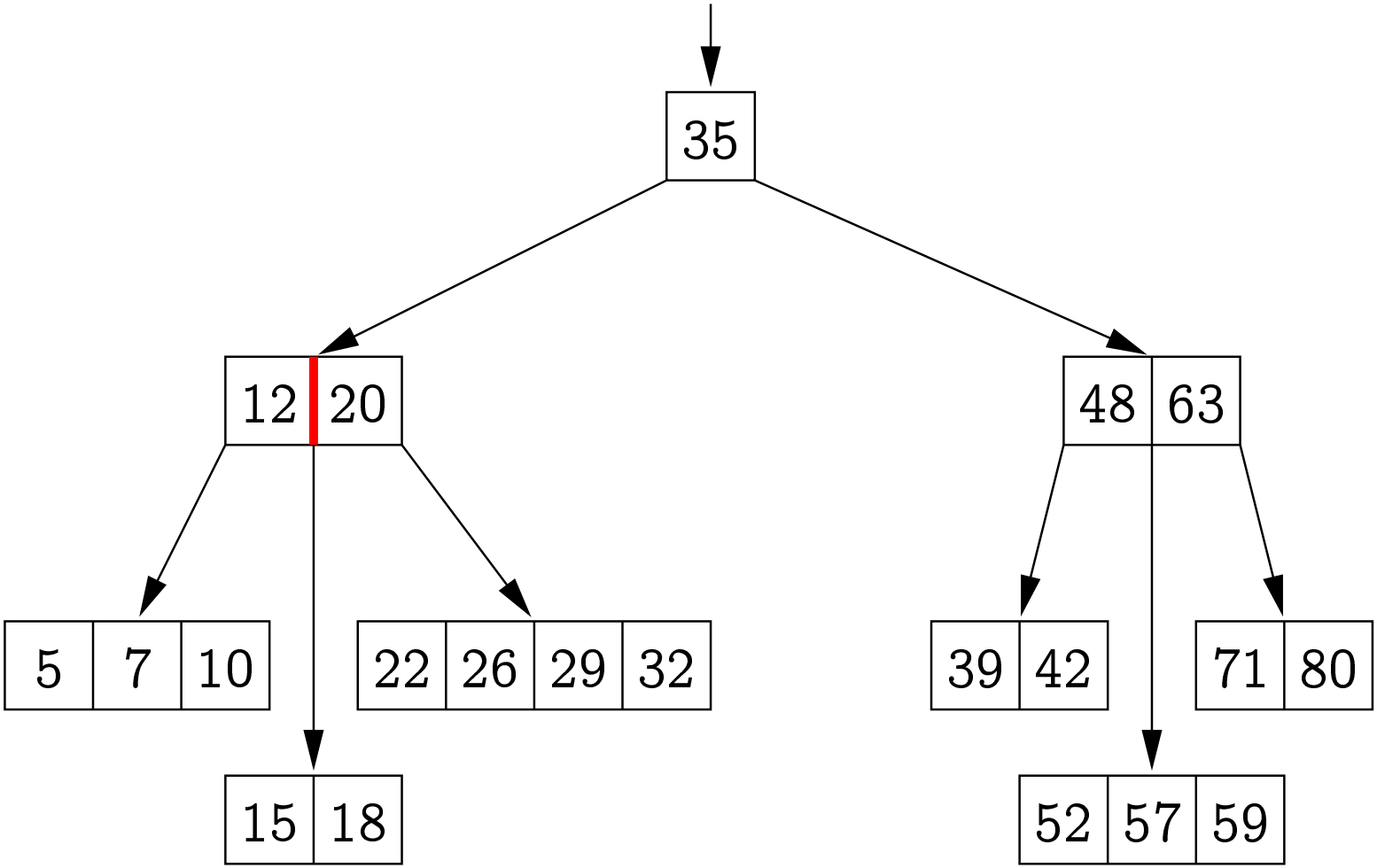
Otsime võtit 18



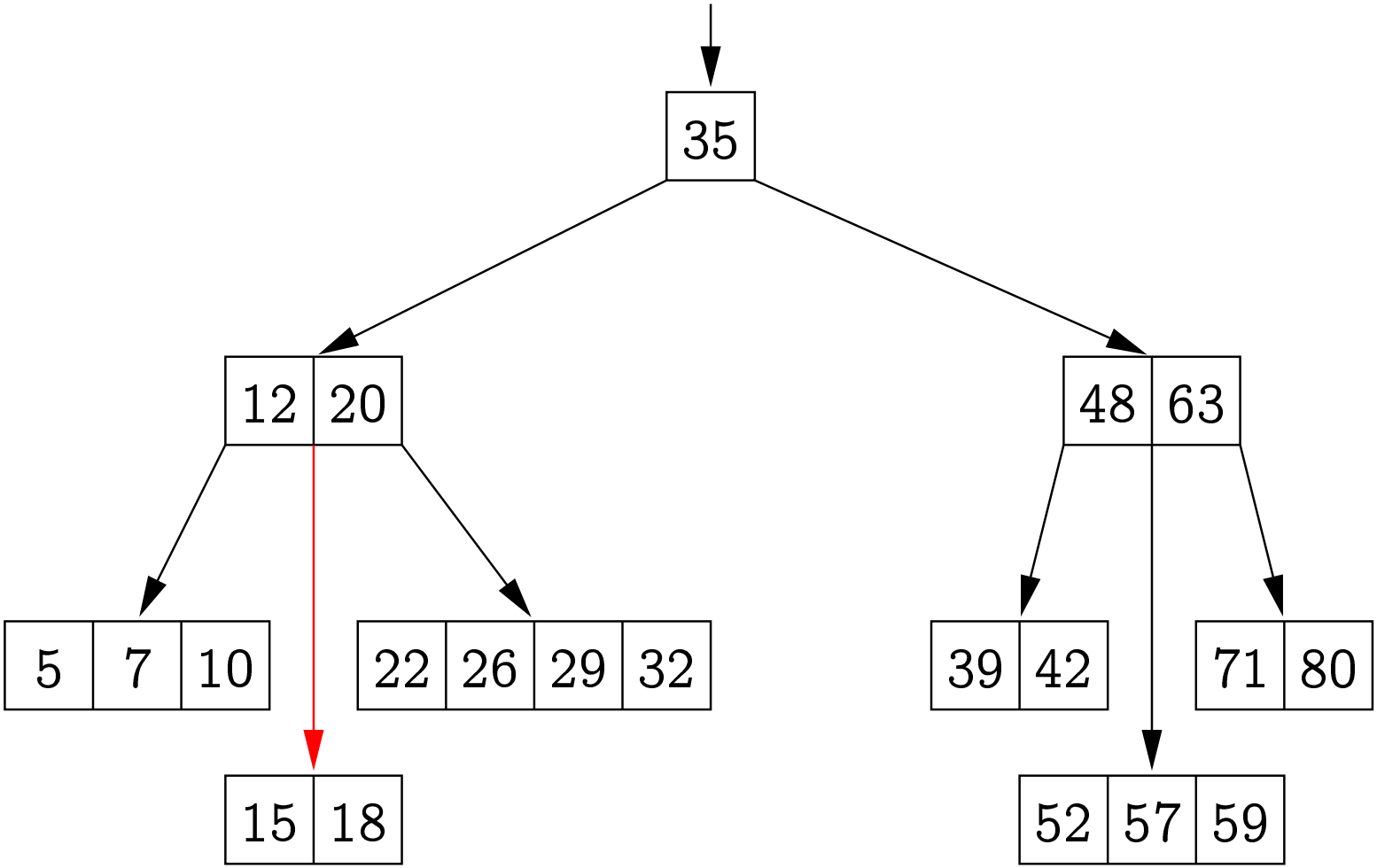
Otsime võtit 18



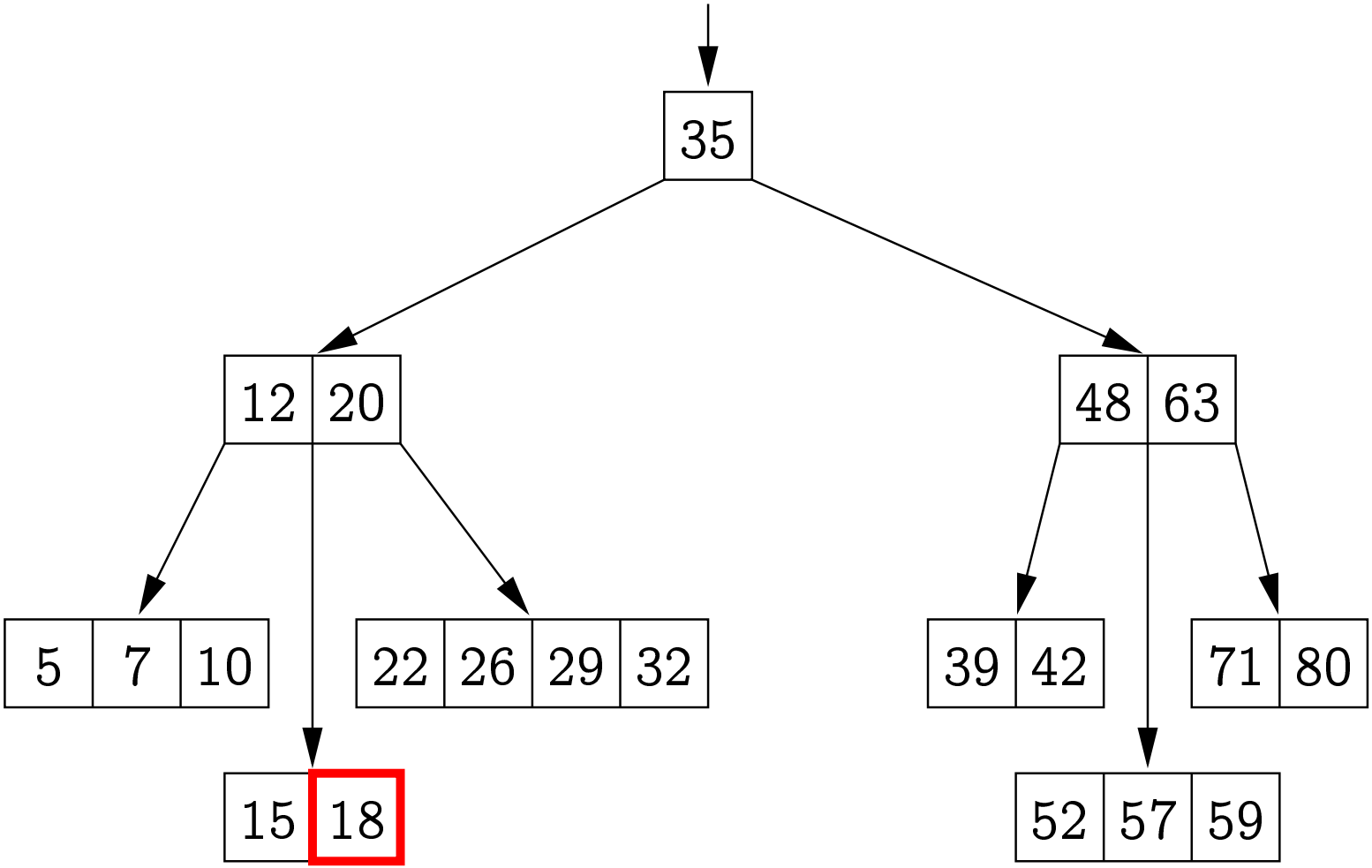
Otsime võtit 18



Otsime võtit 18



Otsime võtit 18



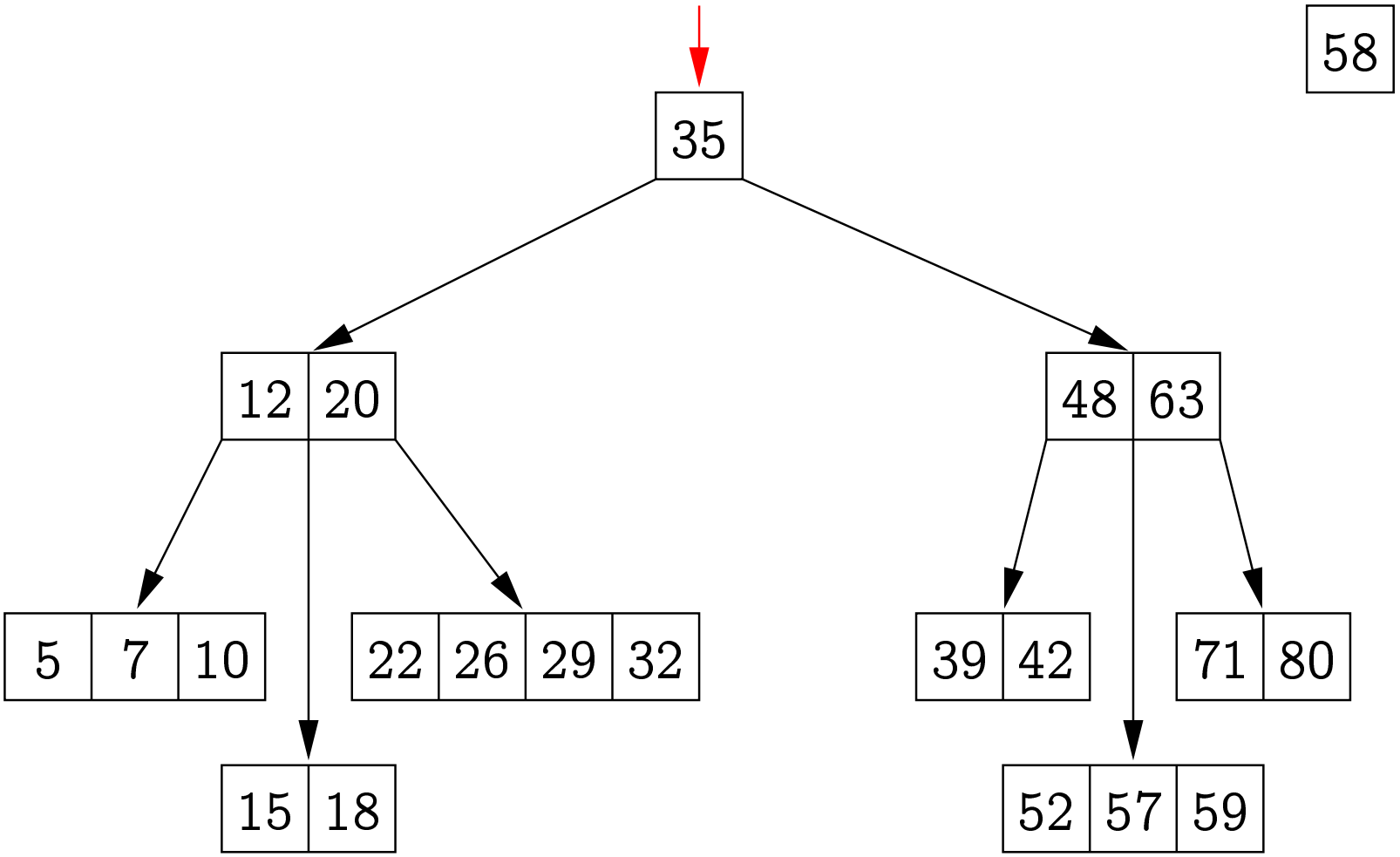
Lisades uut kirjet, eeldame, et vastavat võtit puus veel ei ole.

Kui on, siis liidame lisatava kirje võtmele ε-i.

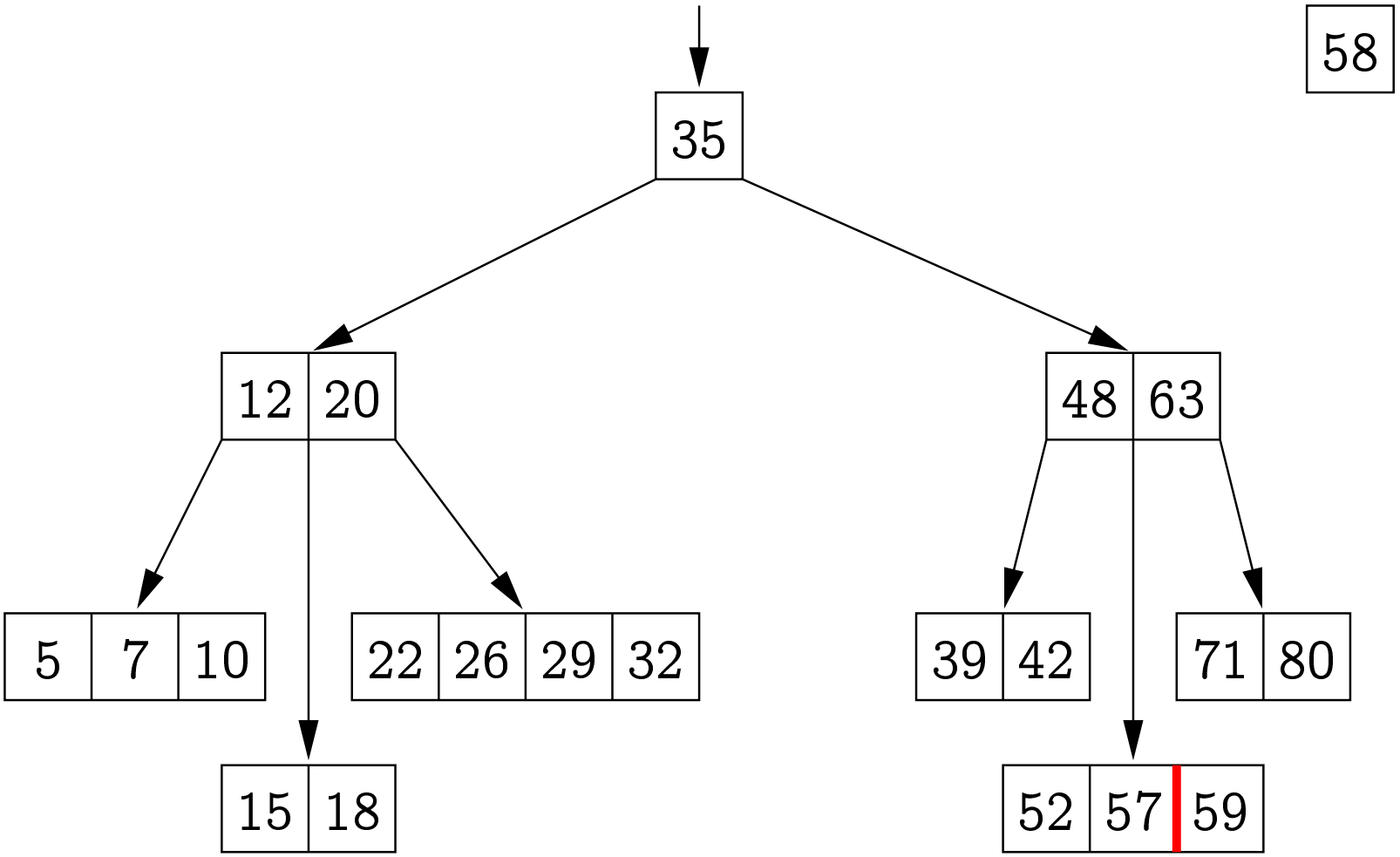
Lisamisel leitakse kõigepealt koht, kuhu vastav kirje käib. See koht on alati mingis lehes.

Kui selles lehes on vähem kui $m - 1$ kirjet, siis suurendatakse lihtsalt seda lehte.

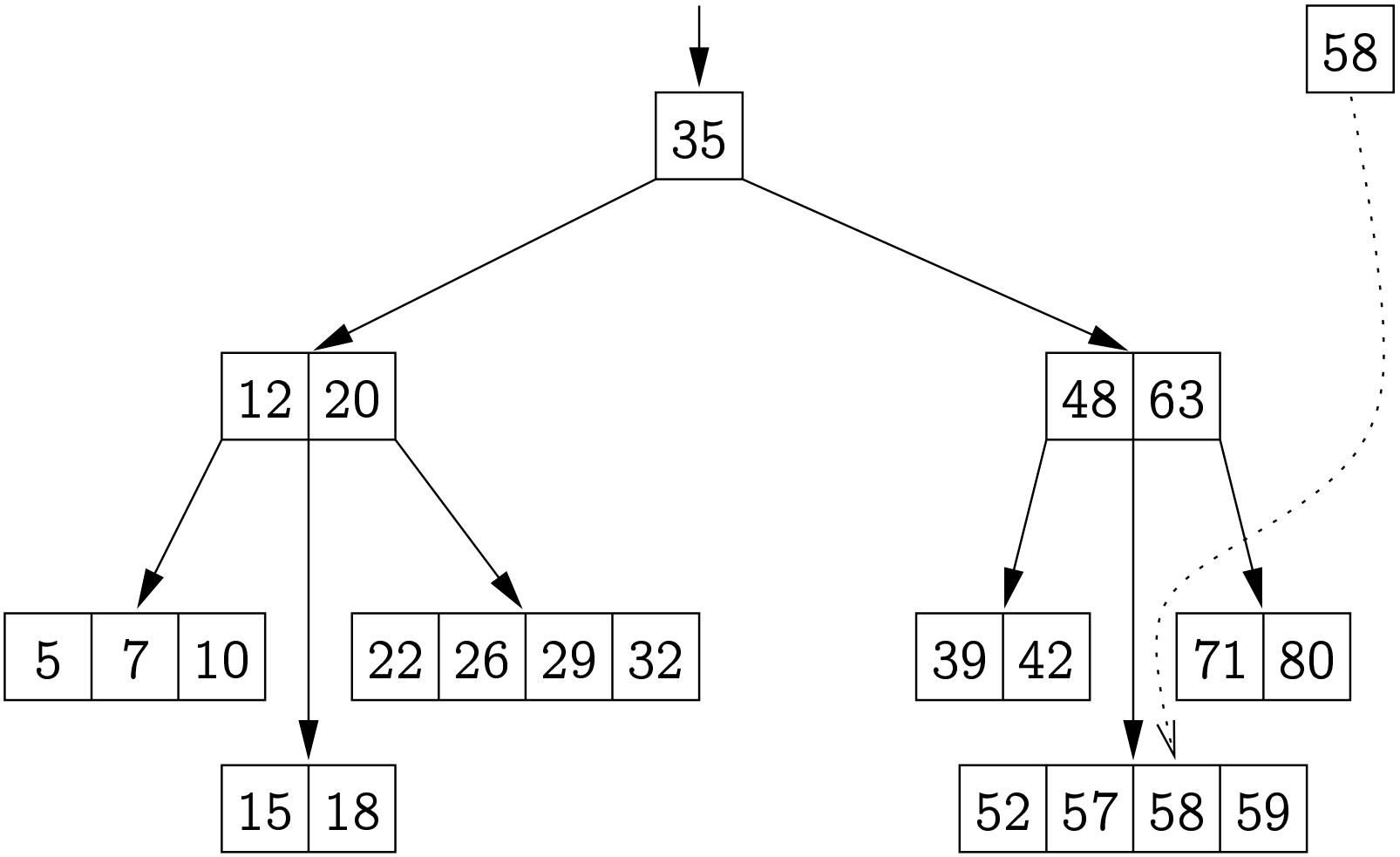
Lisame võtme 58



Lisame võtme 58



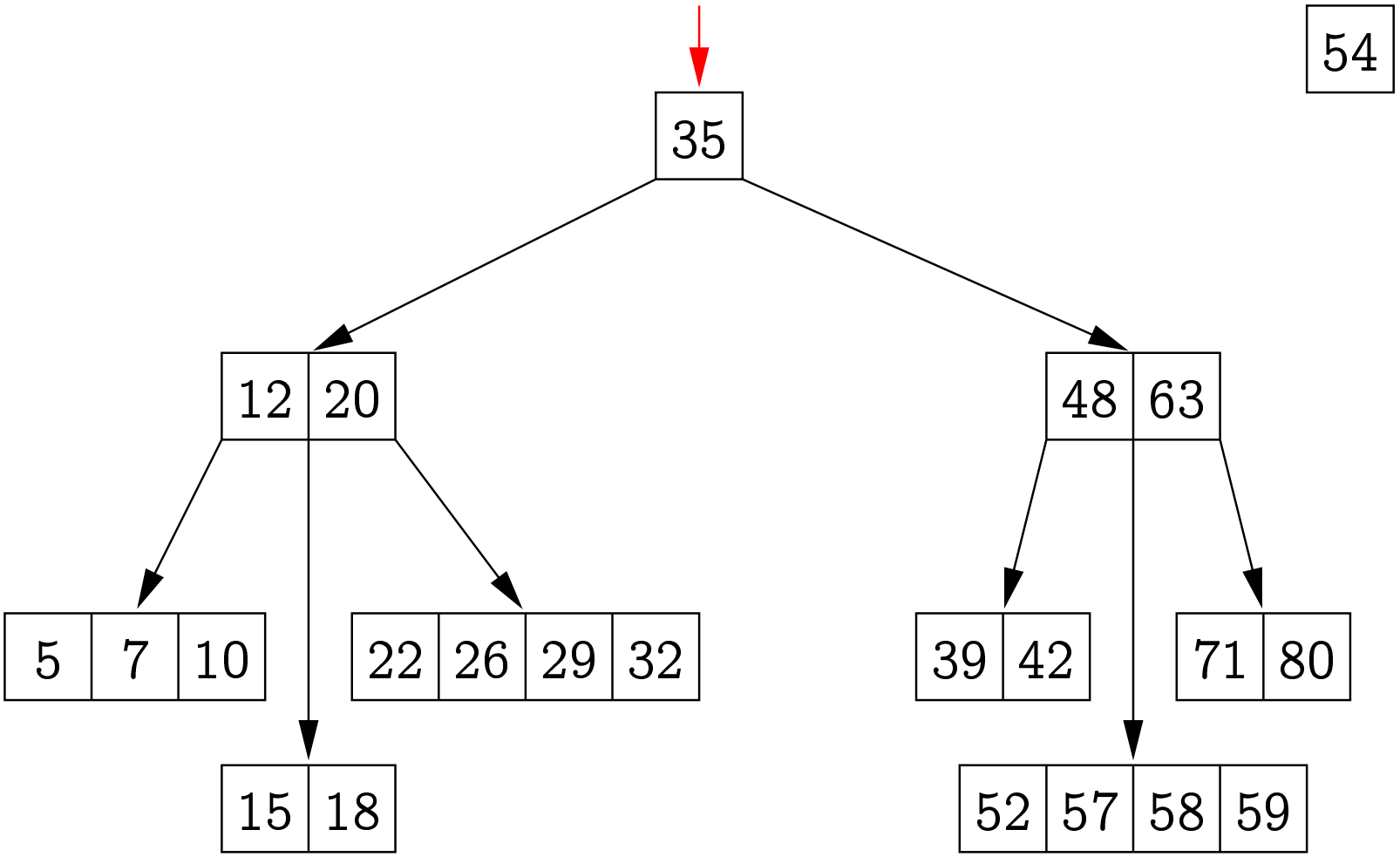
Lisame võtme 58



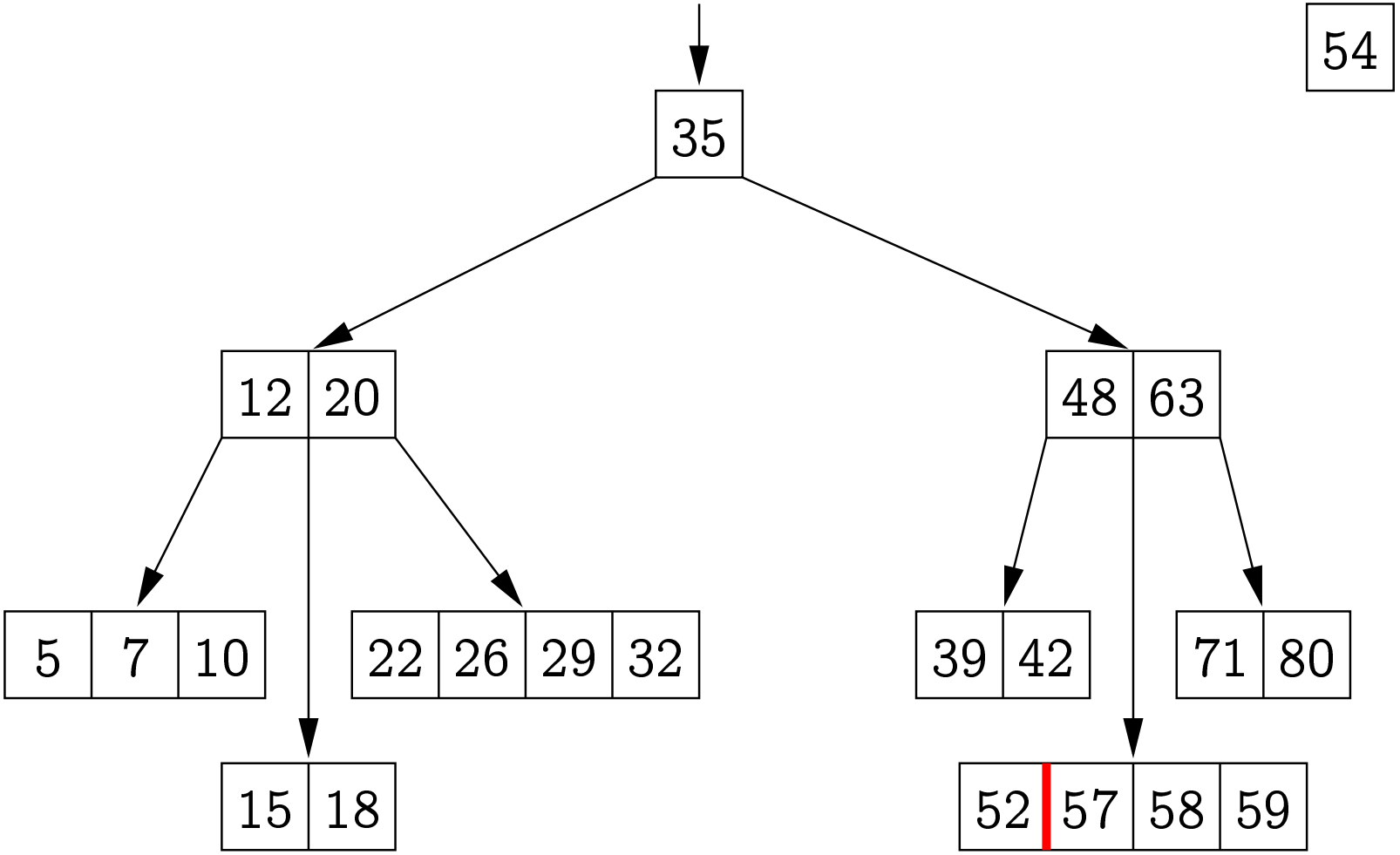
Mingi tipu ületäitumisel (kui sinna tuleb m kirjet) viiakse keskmine kirje ülemusse.

Tipu vasakust ja paremast poolest tehakse kaks tippu, milles on $\lfloor m/2 \rfloor$ kirjet.

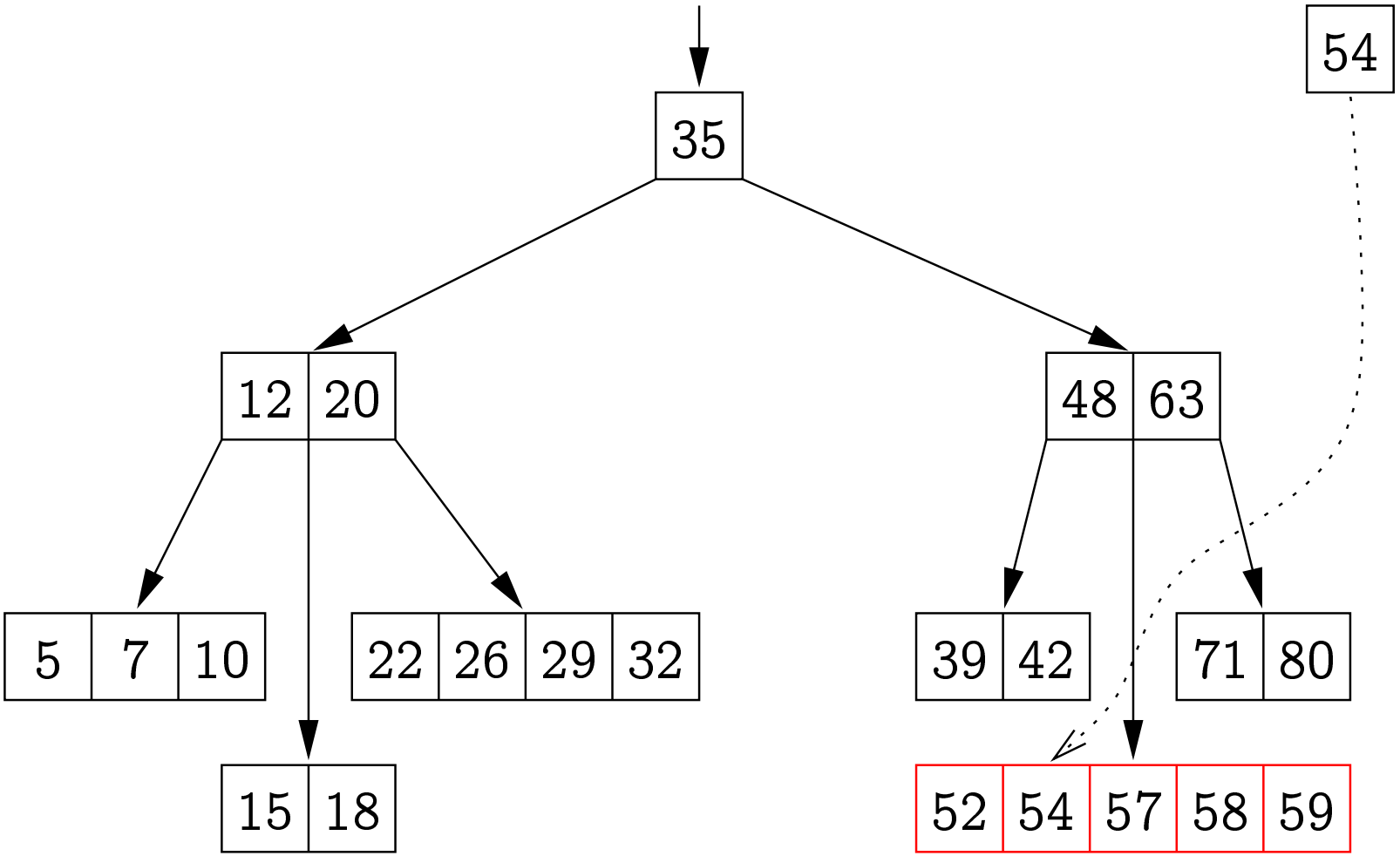
Lisame võtme 54



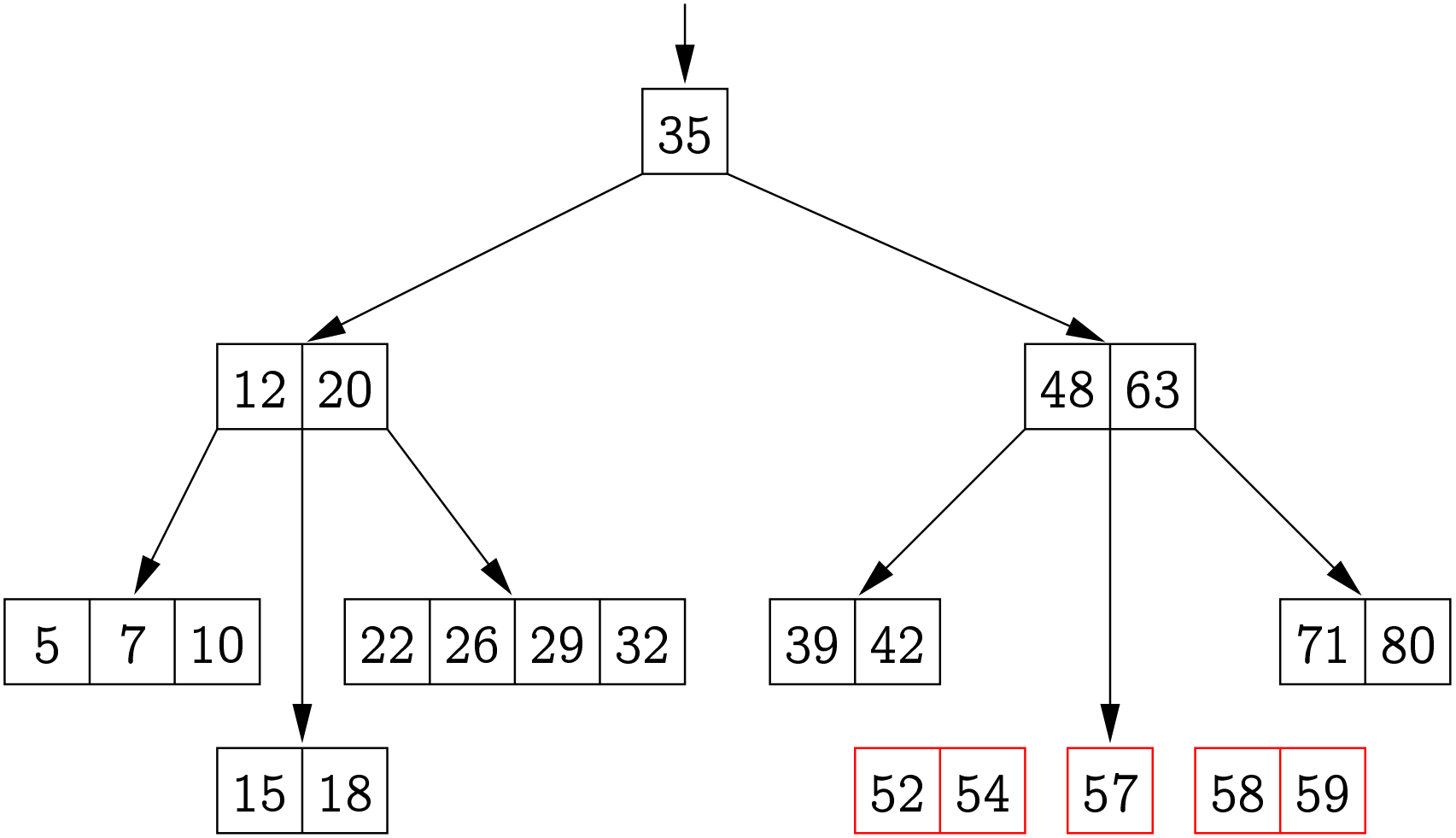
Lisame võtme 54



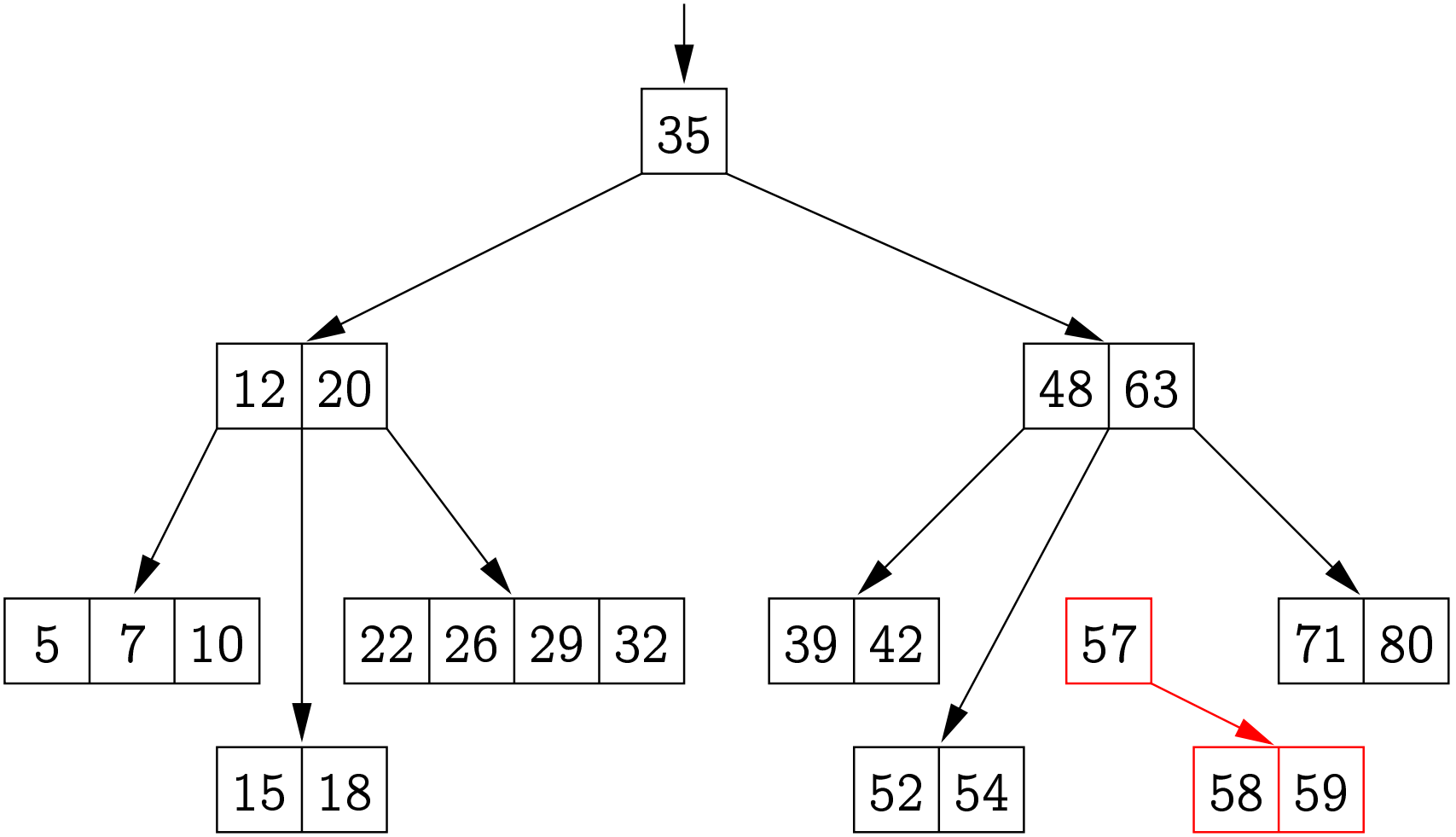
Lisame võtme 54



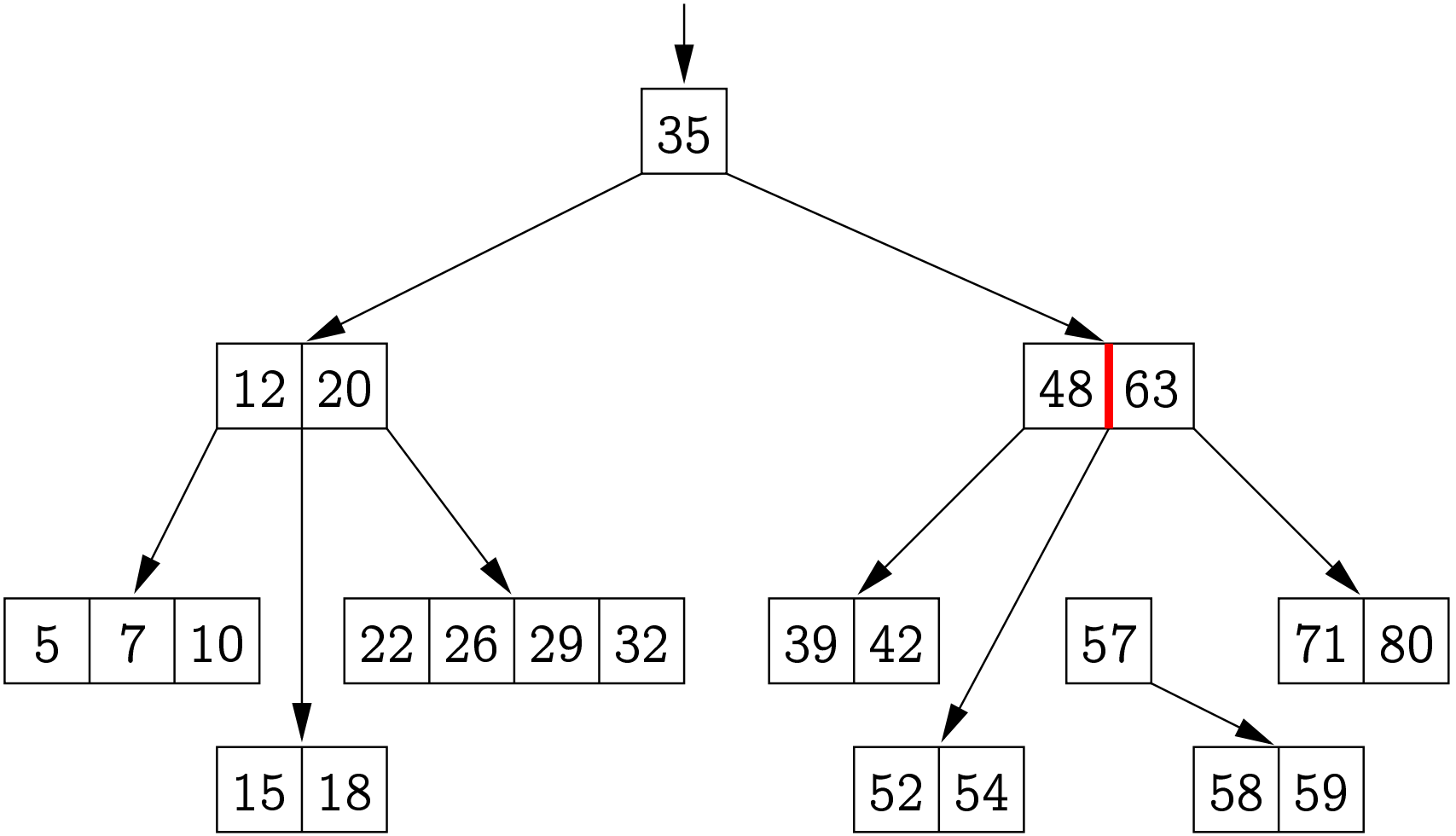
Lisame võtme 54



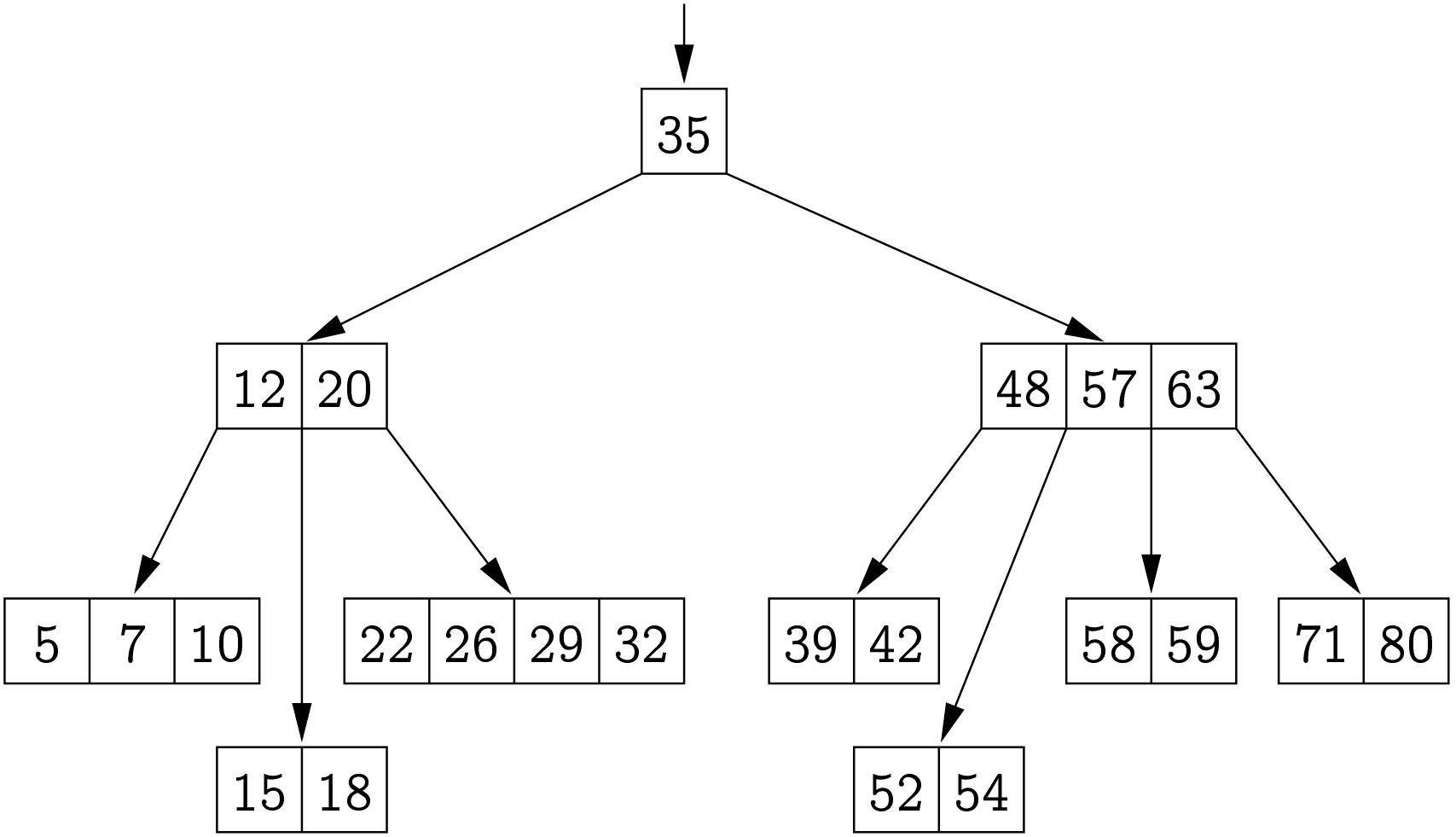
Lisame võtme 54



Lisame võtme 54

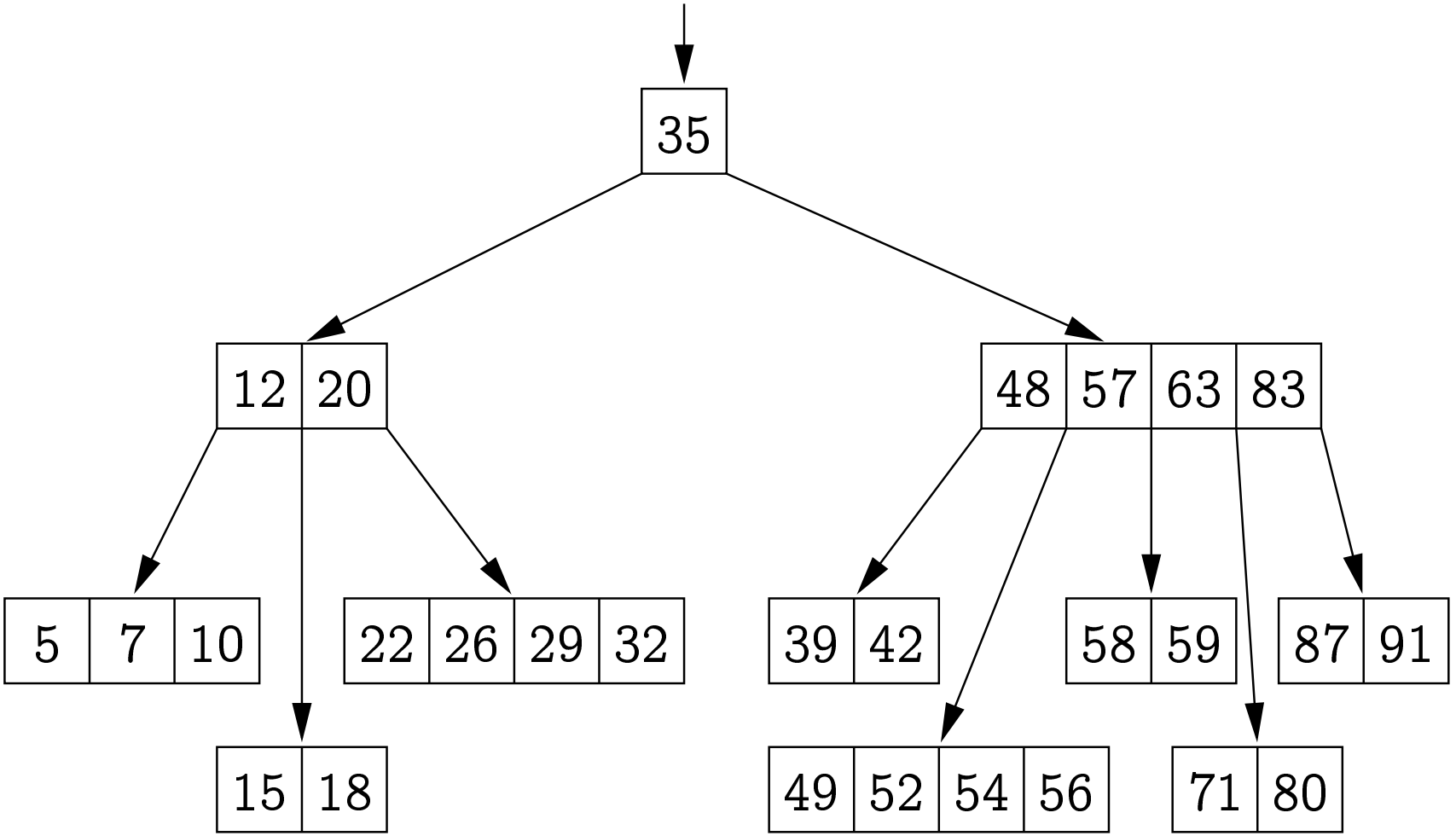


Lisame võtme 54

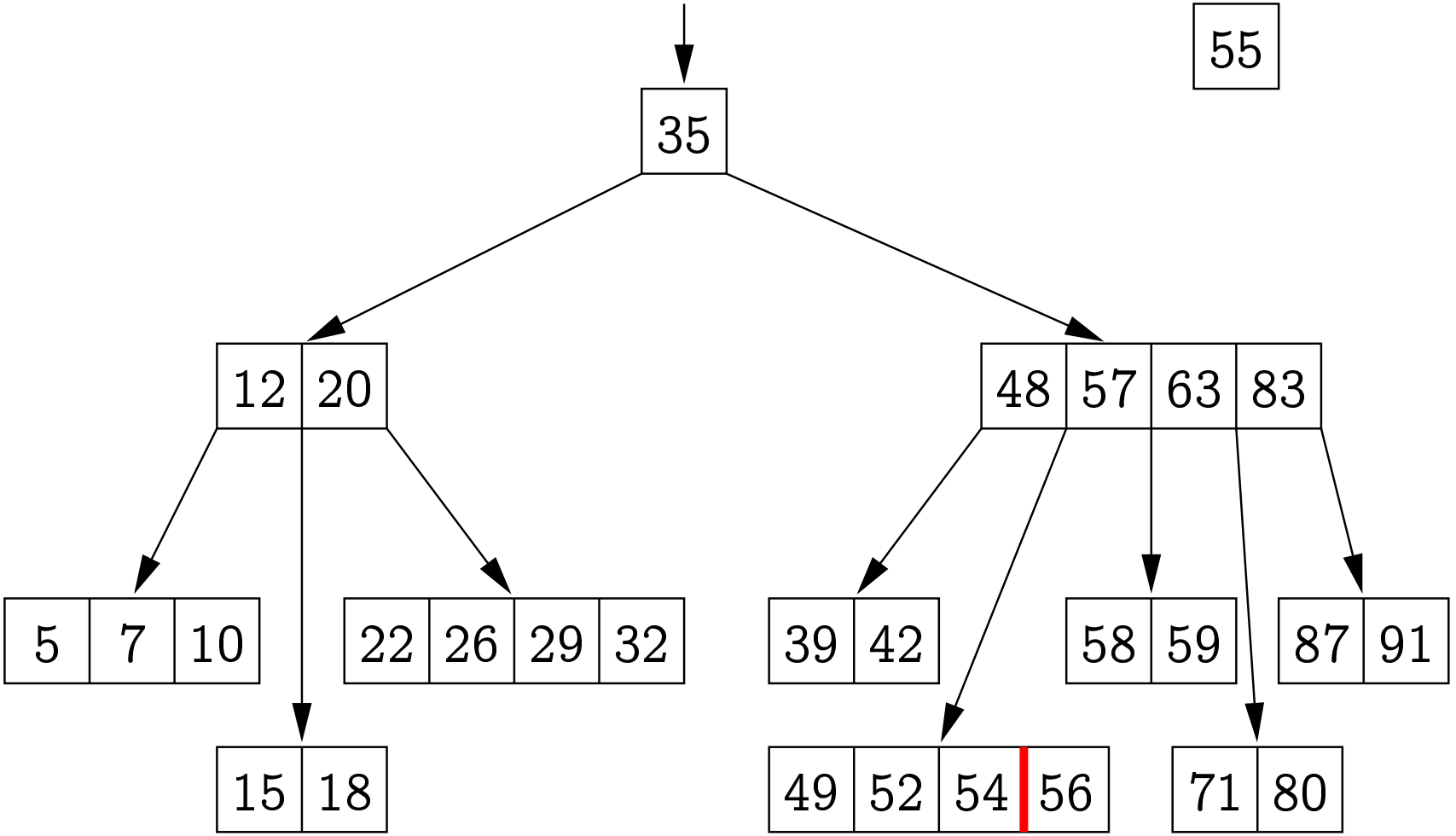


Ülemusse tipu lisamisel võib omakorda ülemus ületäituda.

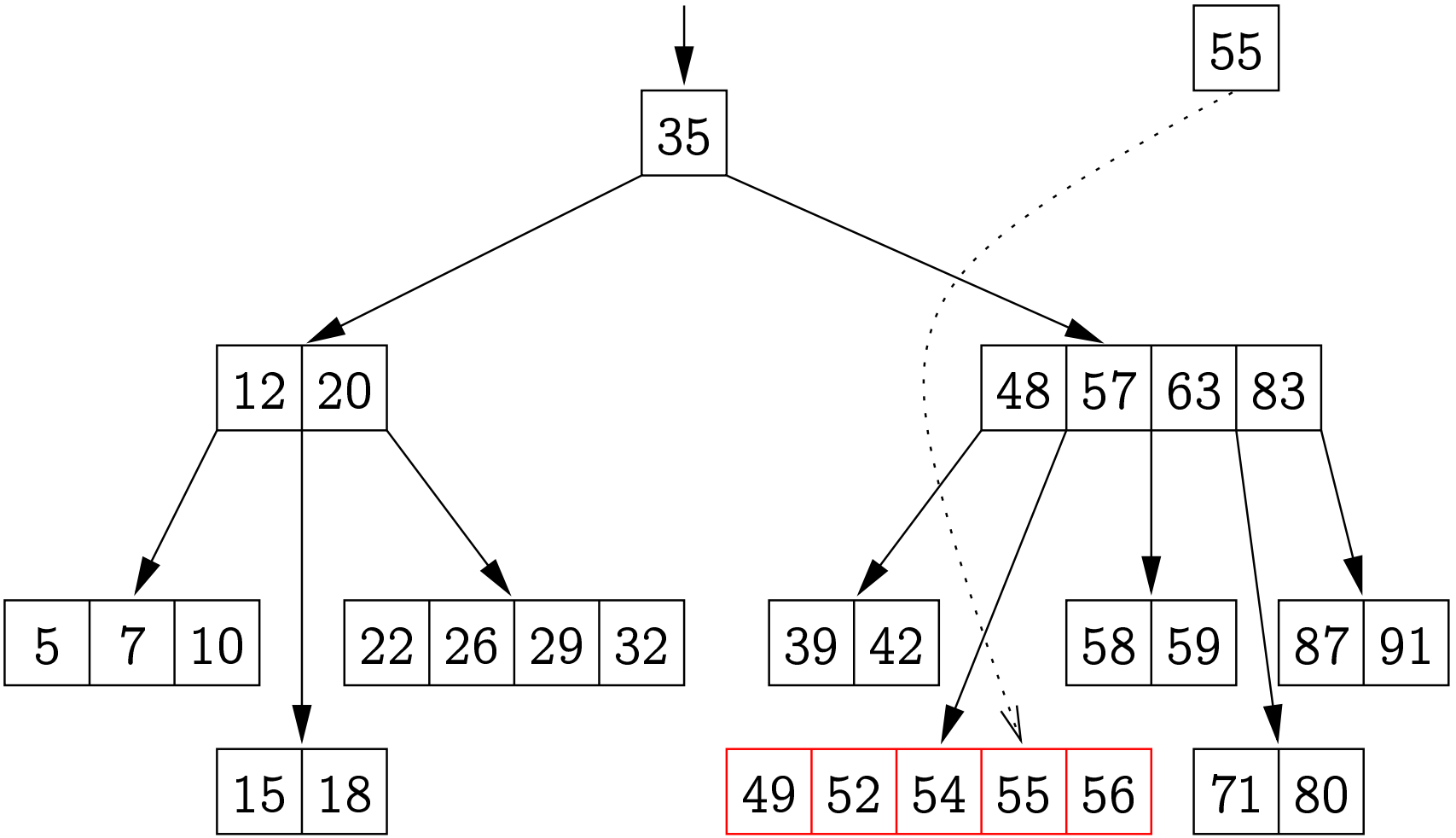
. . .


35
12
20
48
57
63
83
5
7
10
22
26
29
32
39
42
58
59
87
91
15
18
49
52
54
56
71
80

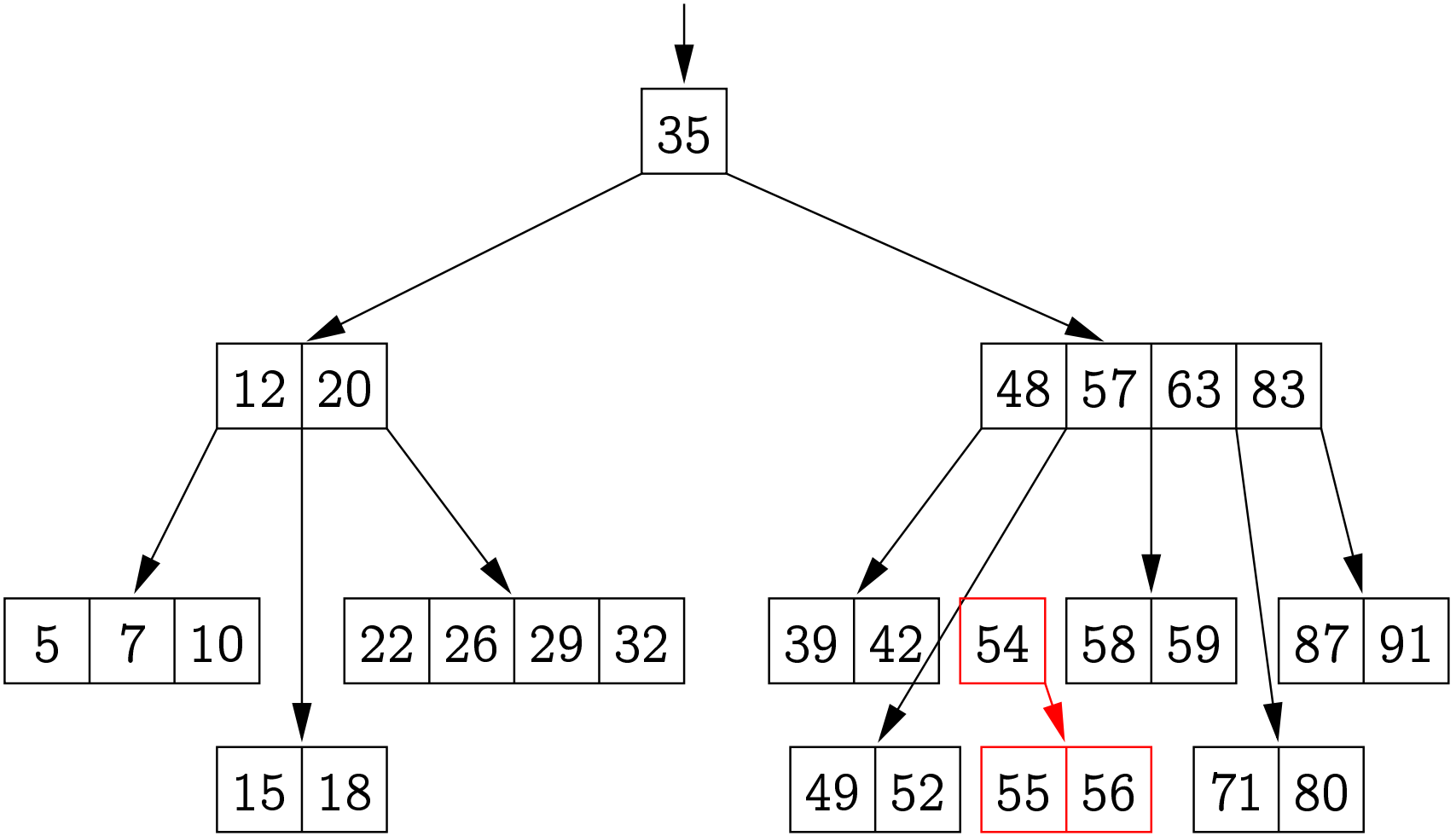
Lisame võtme 55



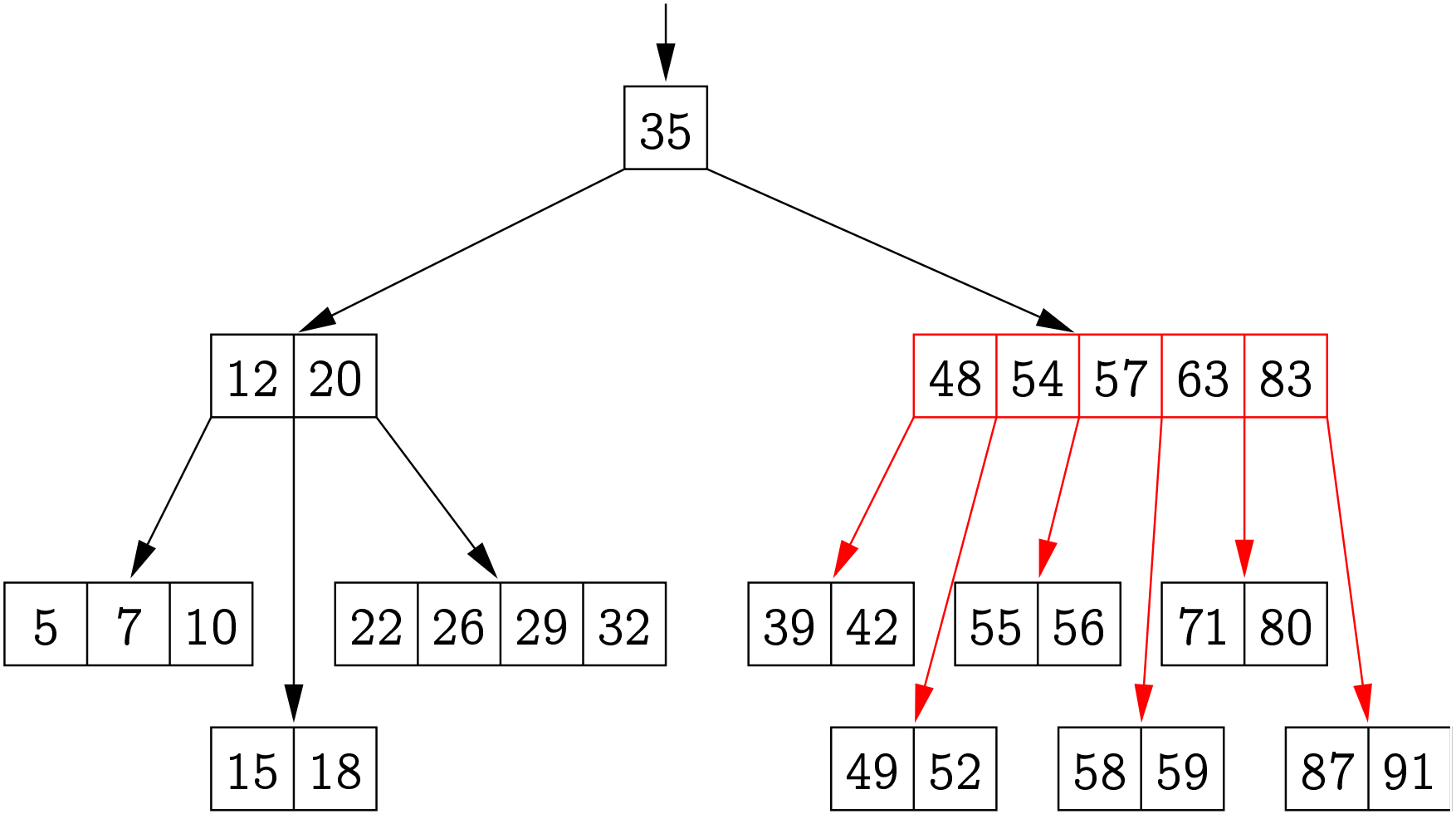
Lisame võtme 55



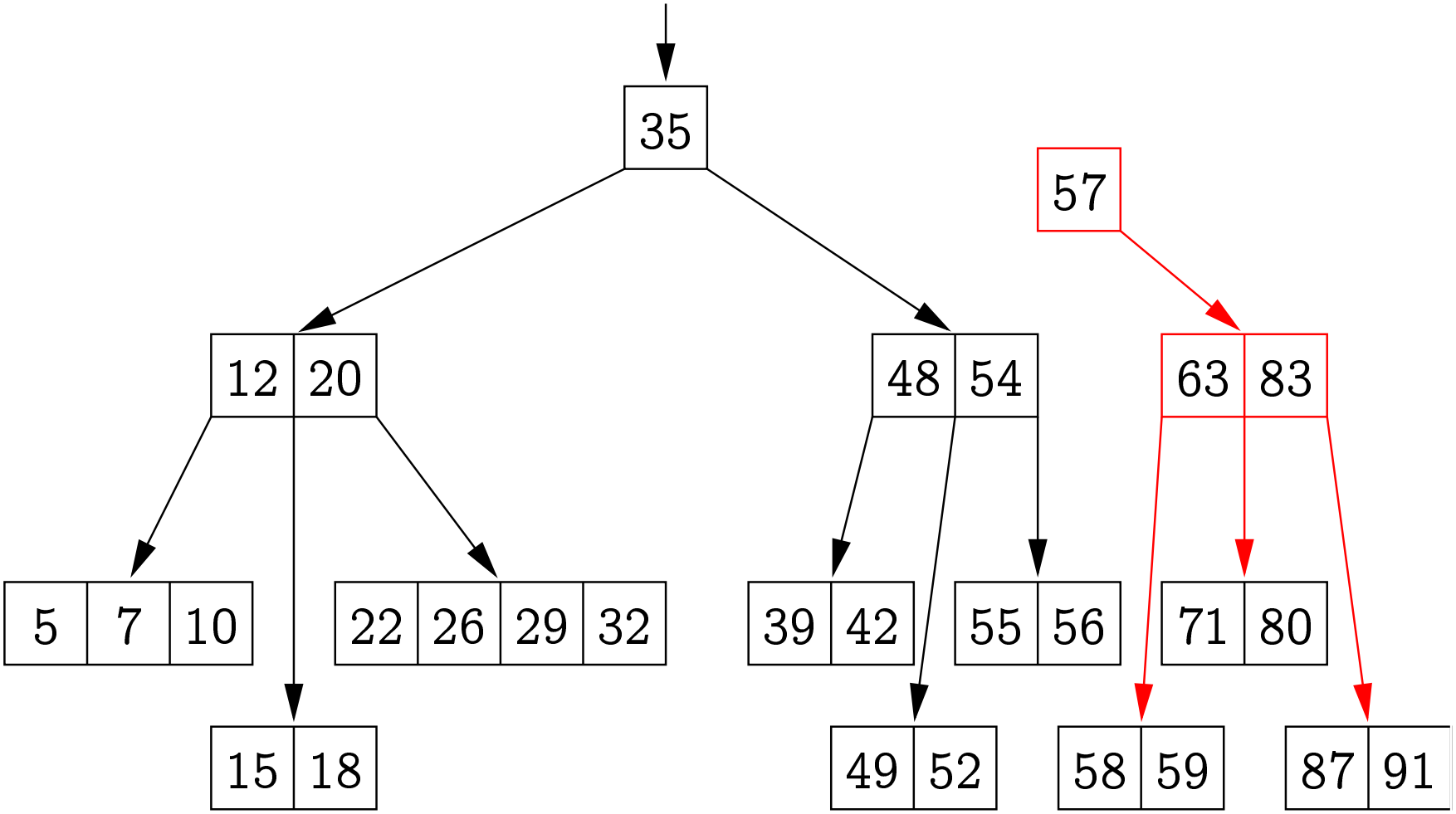
Lisame võtme 55



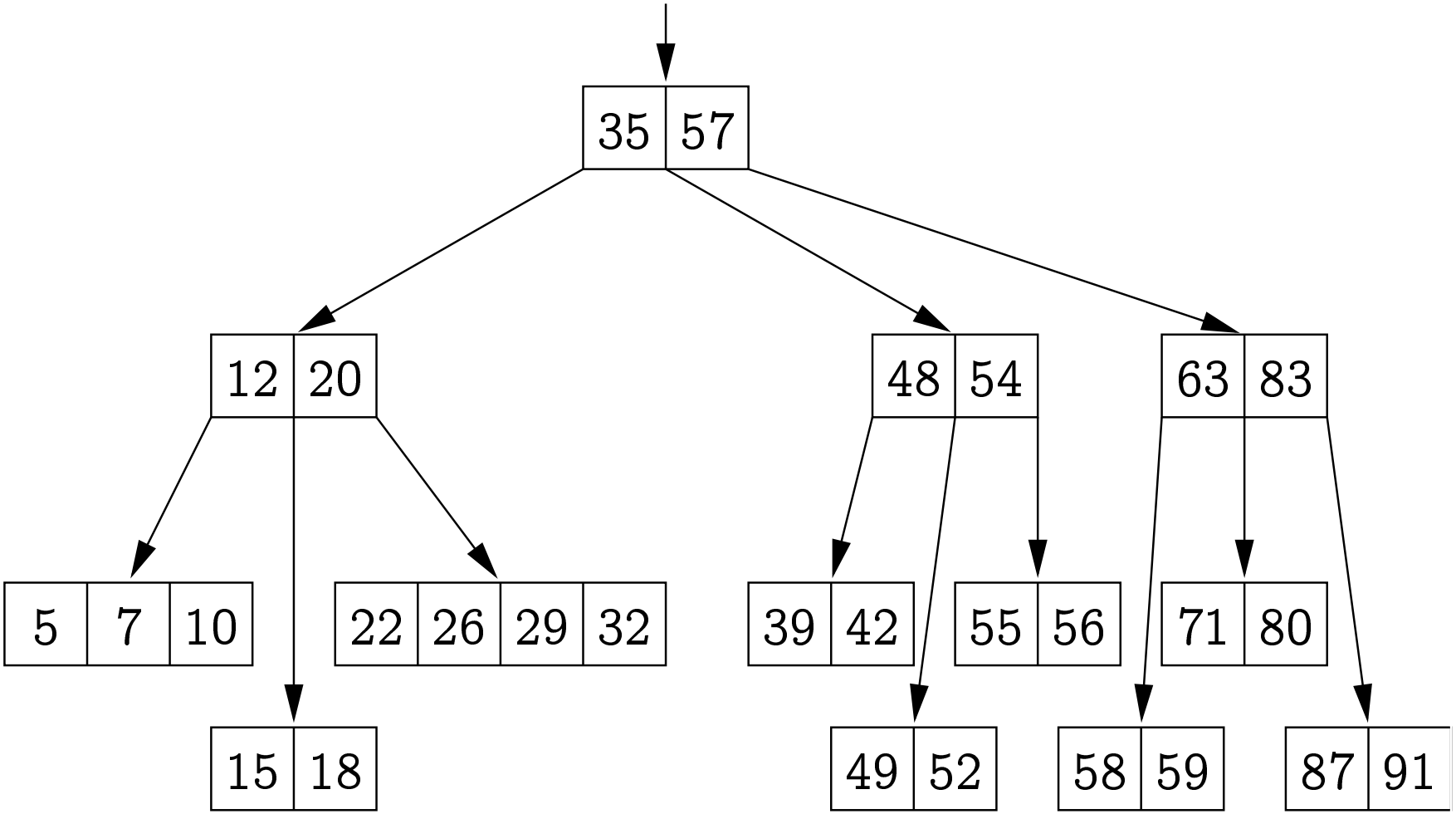
Lisame võtme 55



Lisame võtme 55



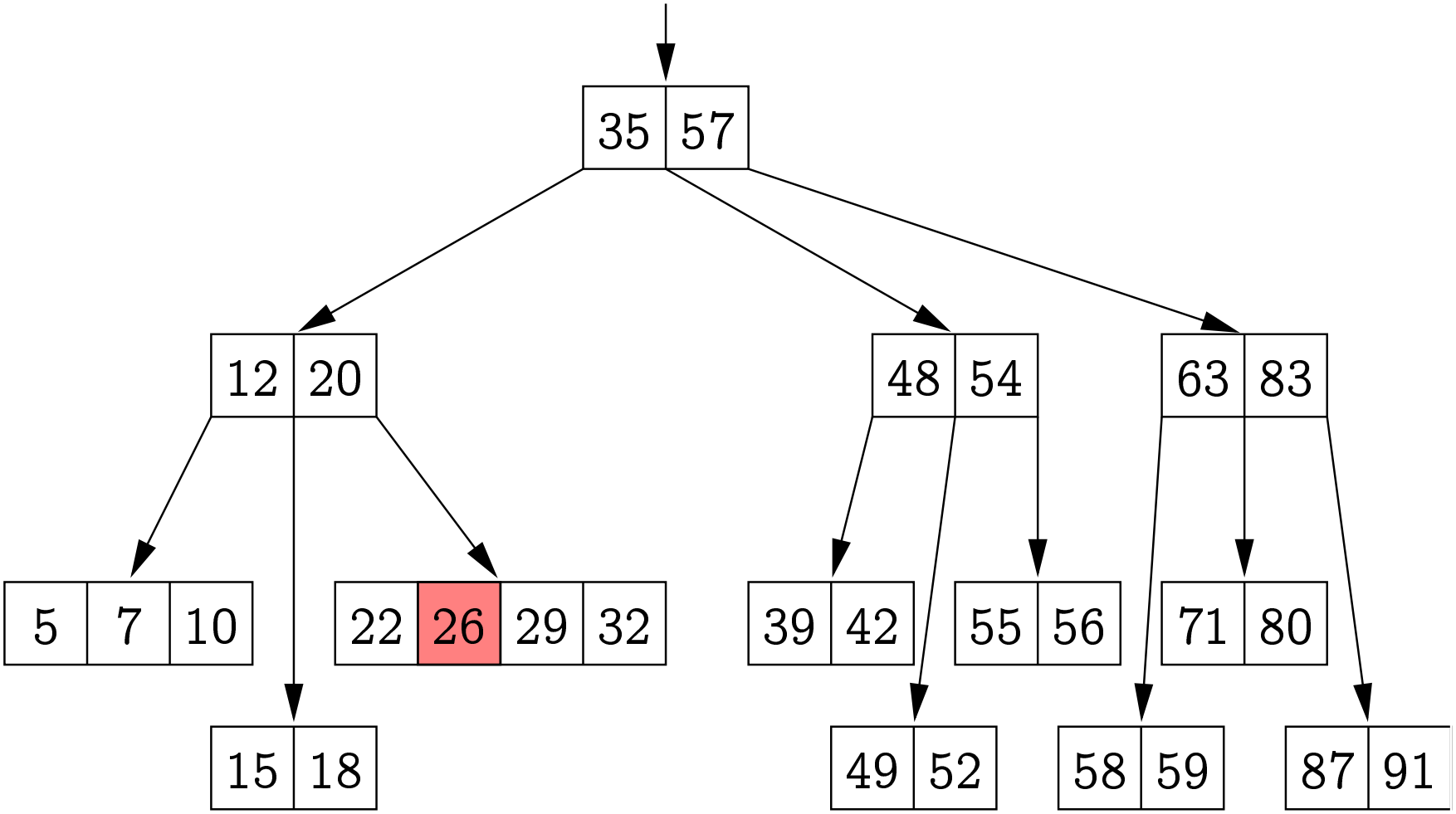
Lisame võtme 55



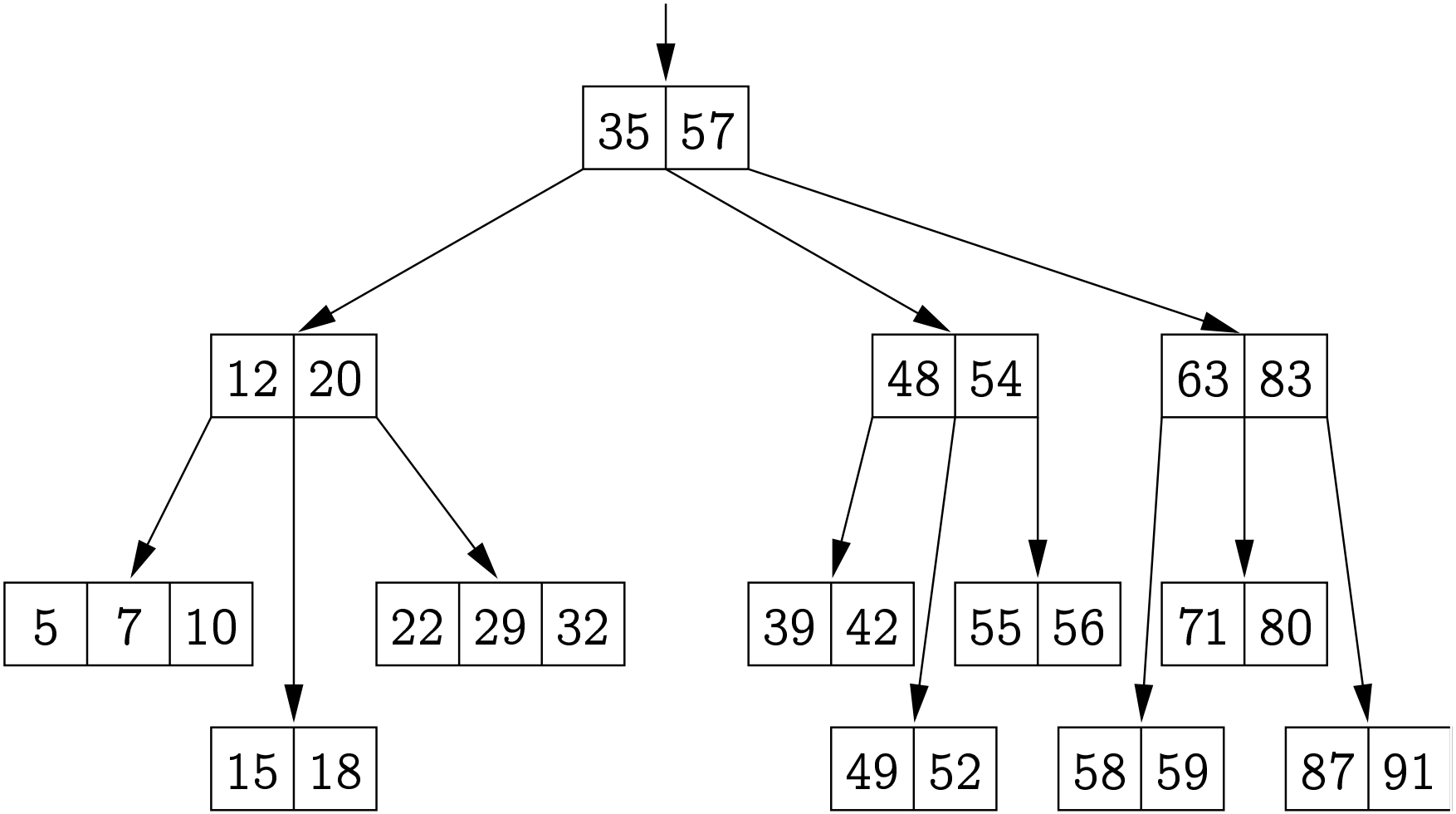
Juure ületäitumisel tekib uus juur.

Kirje eemaldamisel lehest, kus on rohkem kui $\lfloor m/2 \rfloor$ kirjet, teeme selle lehe lihtsalt väiksemaks.

Eemaldame võtme 26

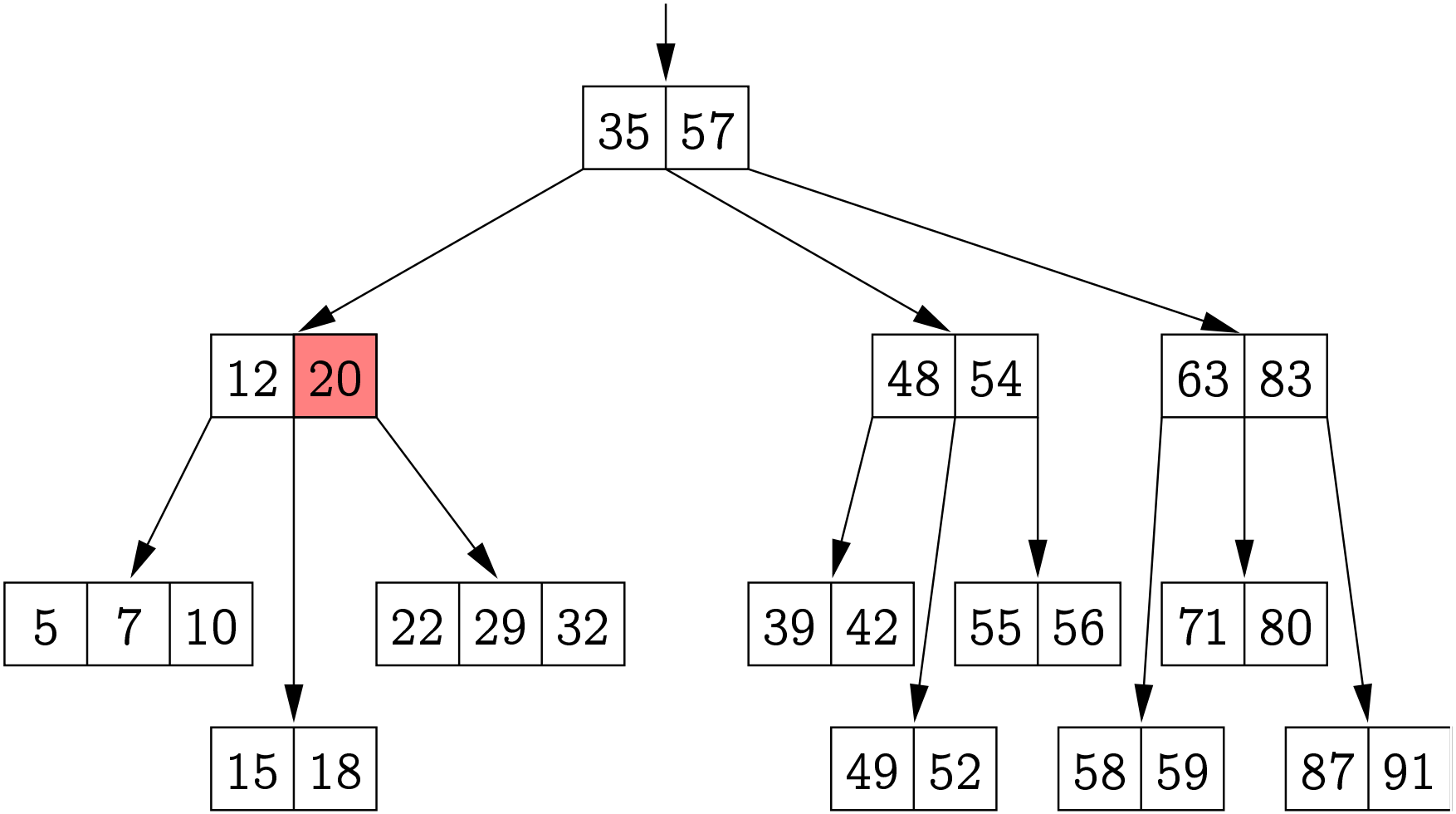


Eemaldame võtme 26

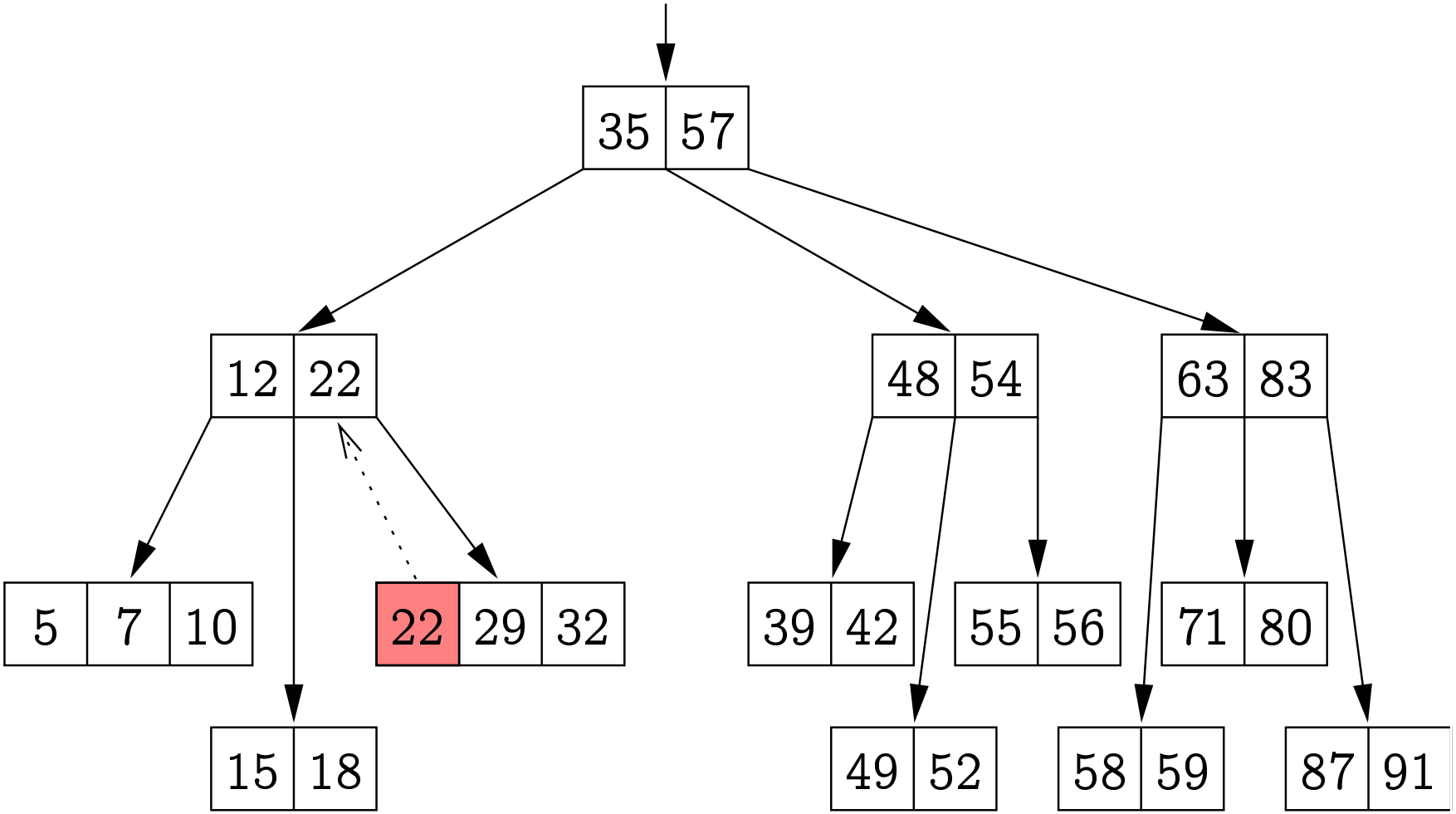


Kirje eemaldamisel sisemisest tipust kirjutame ta üle talle vahetult järgneva kirjega (see asub lehes) ja kustutame hoopis selle kirje.

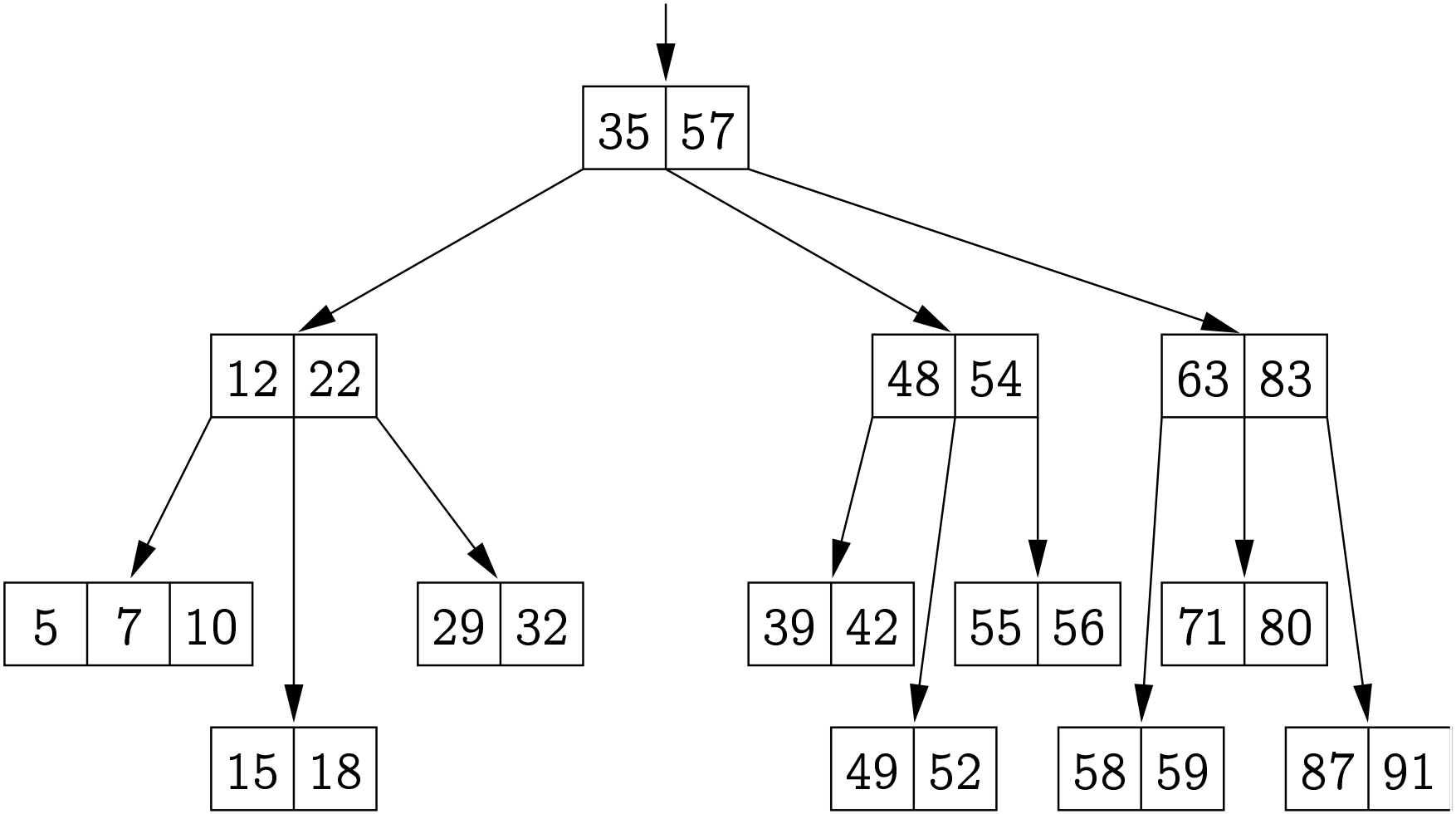
Eemaldame võtme 20



Eemaldame võtme 20

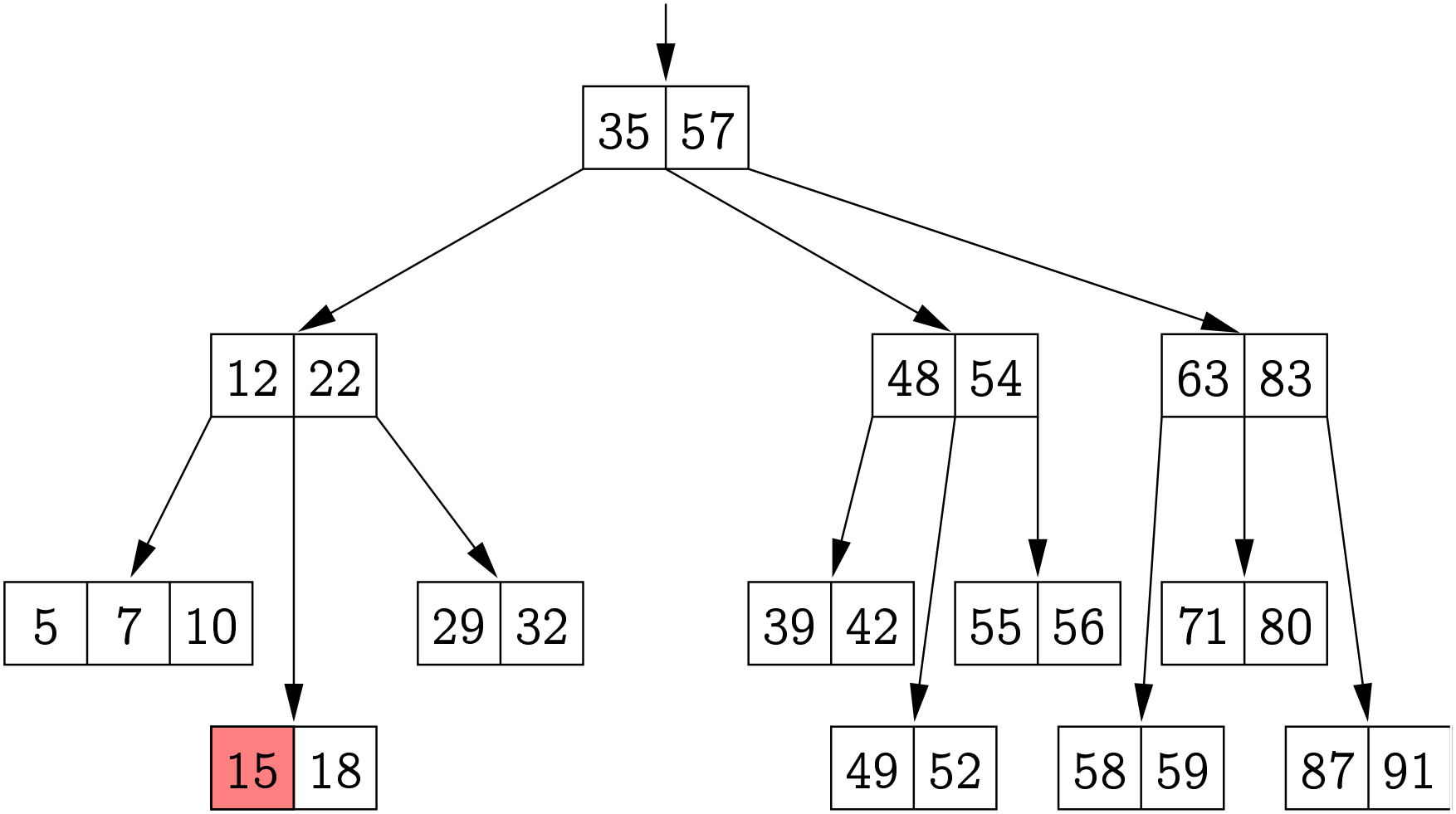


Eemaldame võtme 20

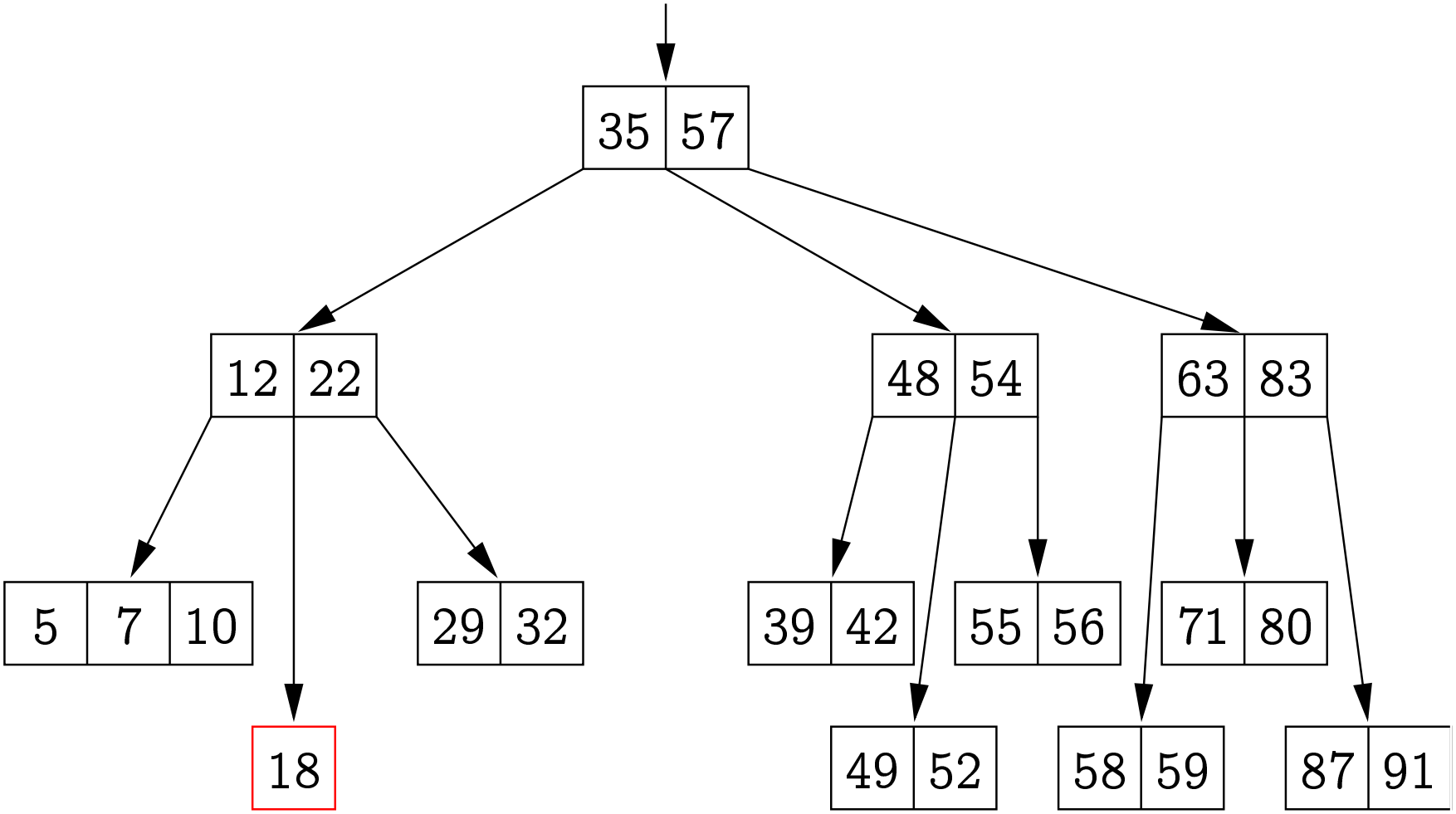


Kui kirje eemaldamisel jääb mõnda tippu vähem kui $\lfloor m/2 \rfloor$ kirjet ja selle tipu vasakus või paremas naabris on rohkem kui $\lfloor m/2 \rfloor$ kirjet, siis võtame sellest naabertipust ühe kirje (ülemuse kaudu).

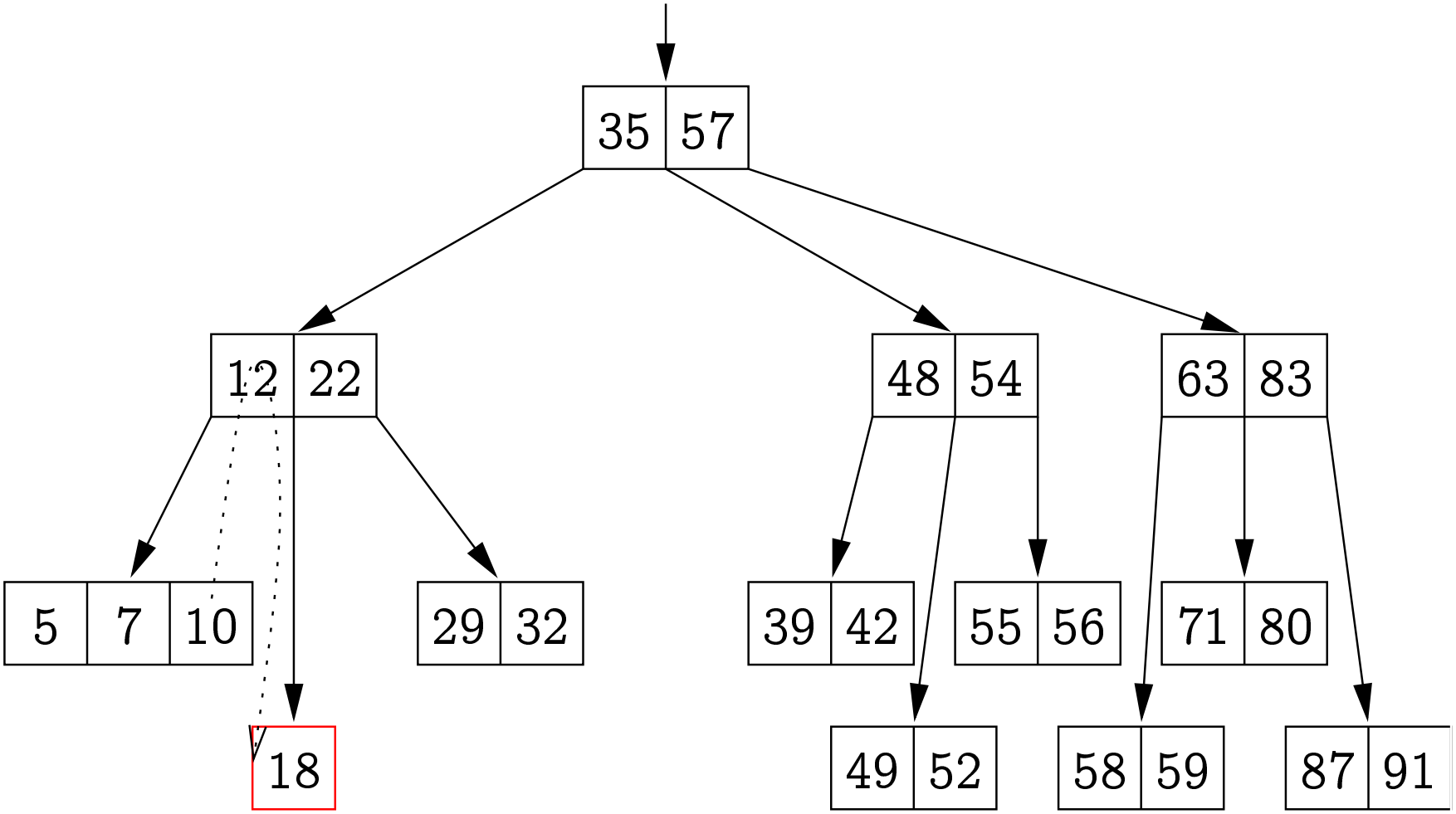
Eemaldame võtme 15



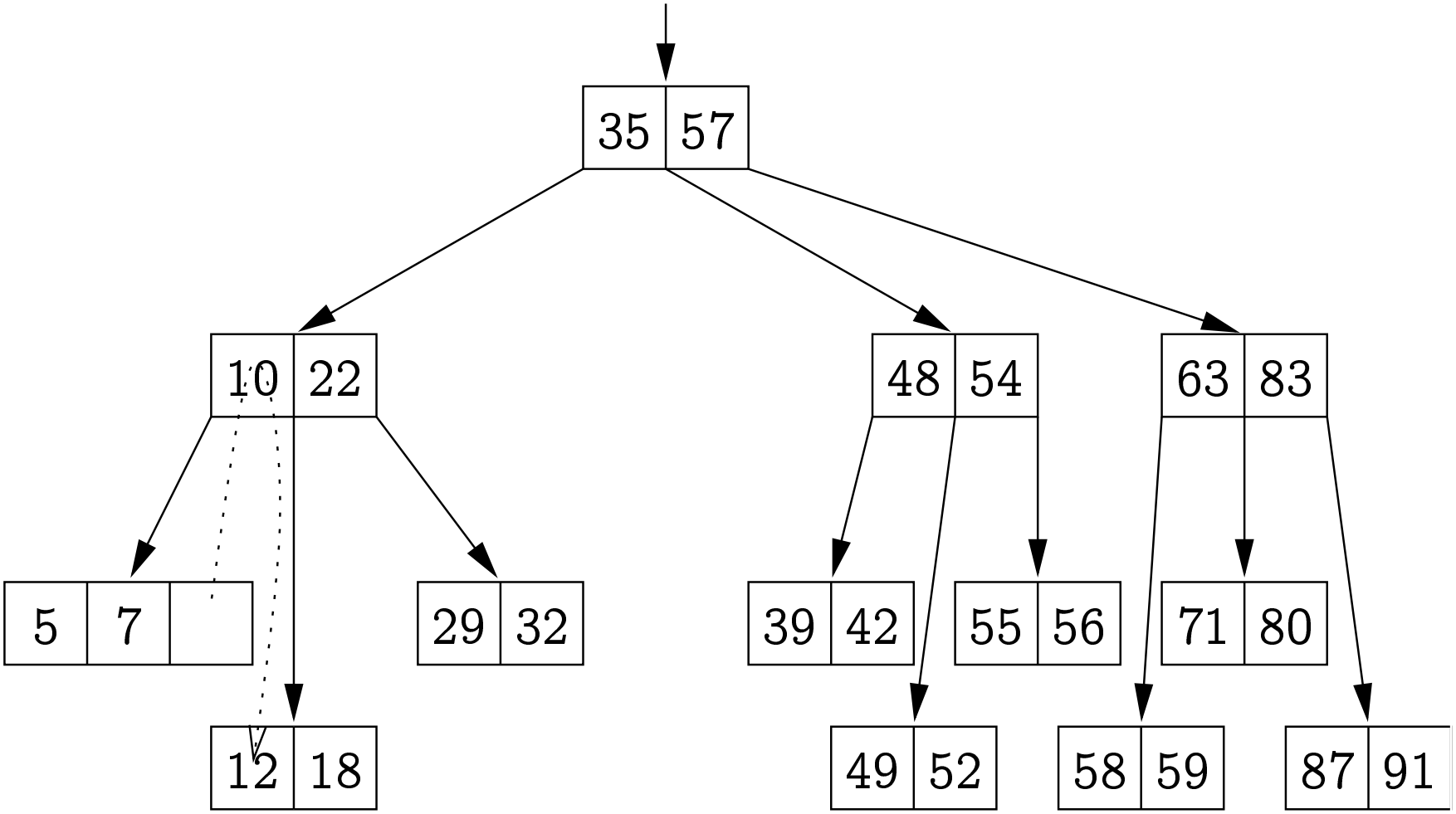
Eemaldame võtme 15



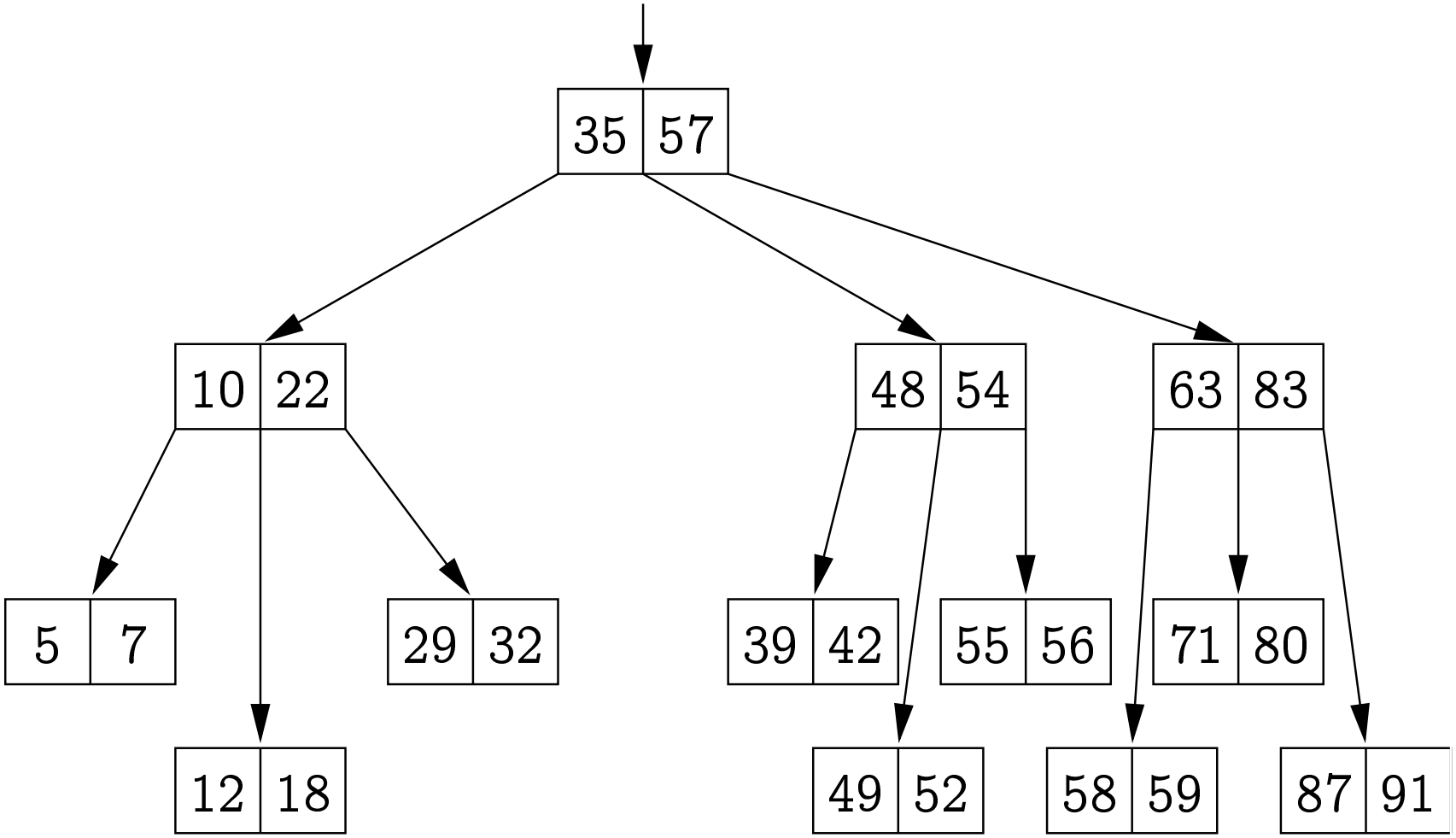
Eemaldame võtme 15



Eemaldame võtme 15



Eemaldame võtme 15



Kui kirje eemaldamisel jääb mõnda tippu vähem kui $\lfloor m/2 \rfloor$ kirjet ja selle tipu vasakus ja paremas naabris on täpselt kui $\lfloor m/2 \rfloor$ kirjet, siis ühendame tipu, tema naabertipu ja nendevahelise kirje ülemusest.

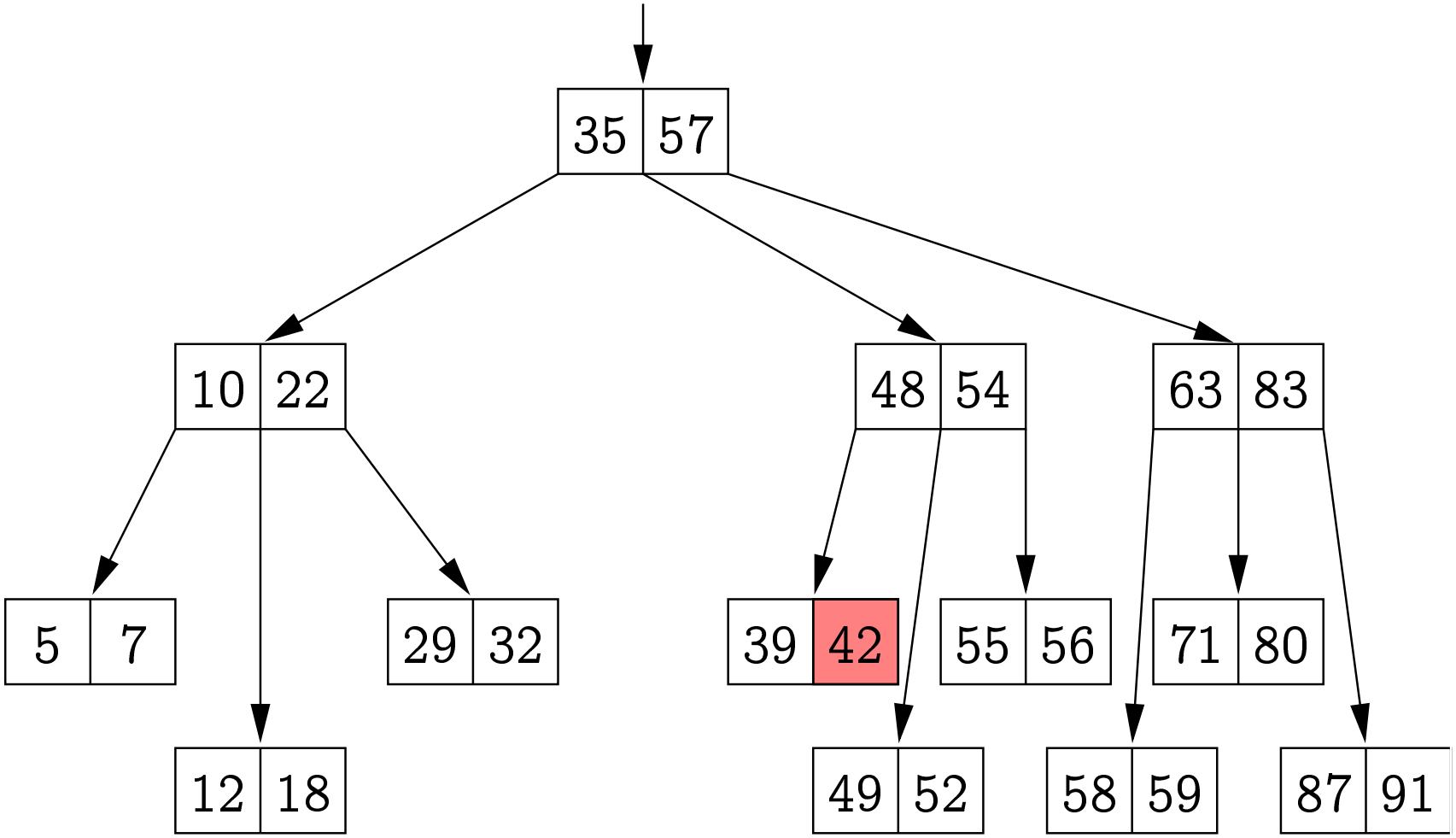
Saame tipu, kus on

$$\lfloor m/2 \rfloor - 1 + \lfloor m/2 \rfloor + 1 = 2\lfloor m/2 \rfloor = m - 1$$

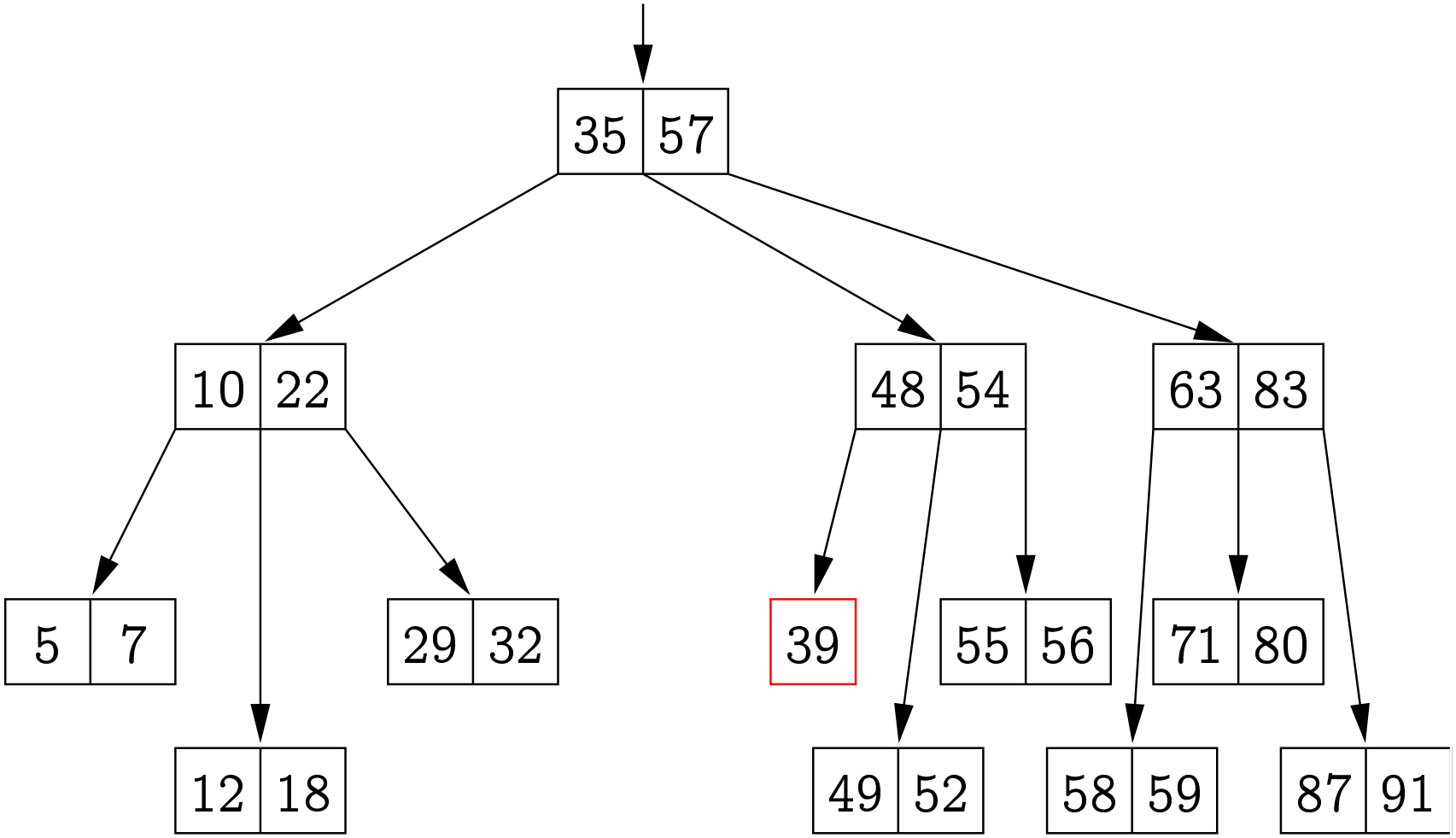
kirjet.

Seejuures väheneb kirjete arv ülemuses ühe võrra.

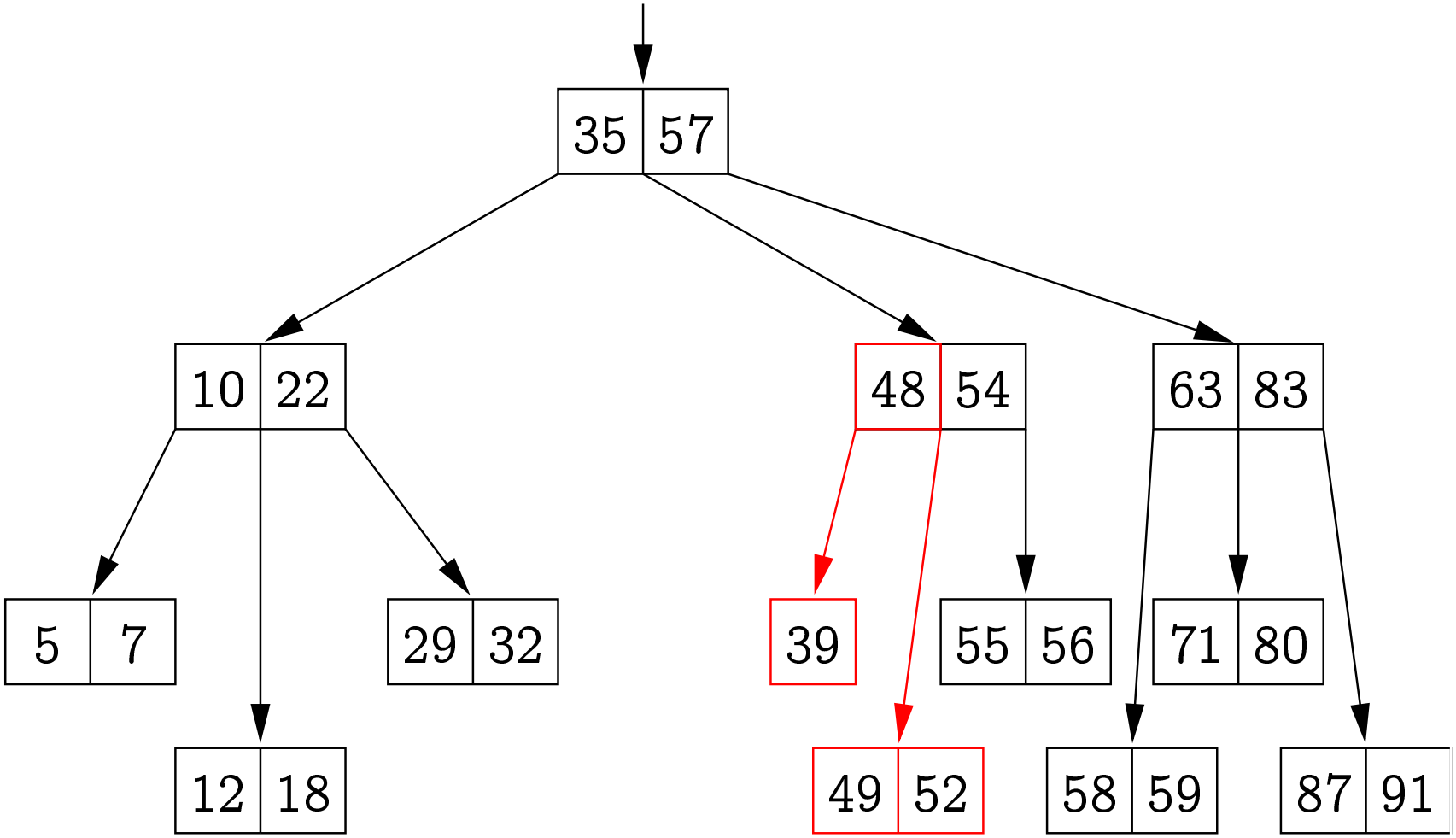
Eemaldame võtme 42



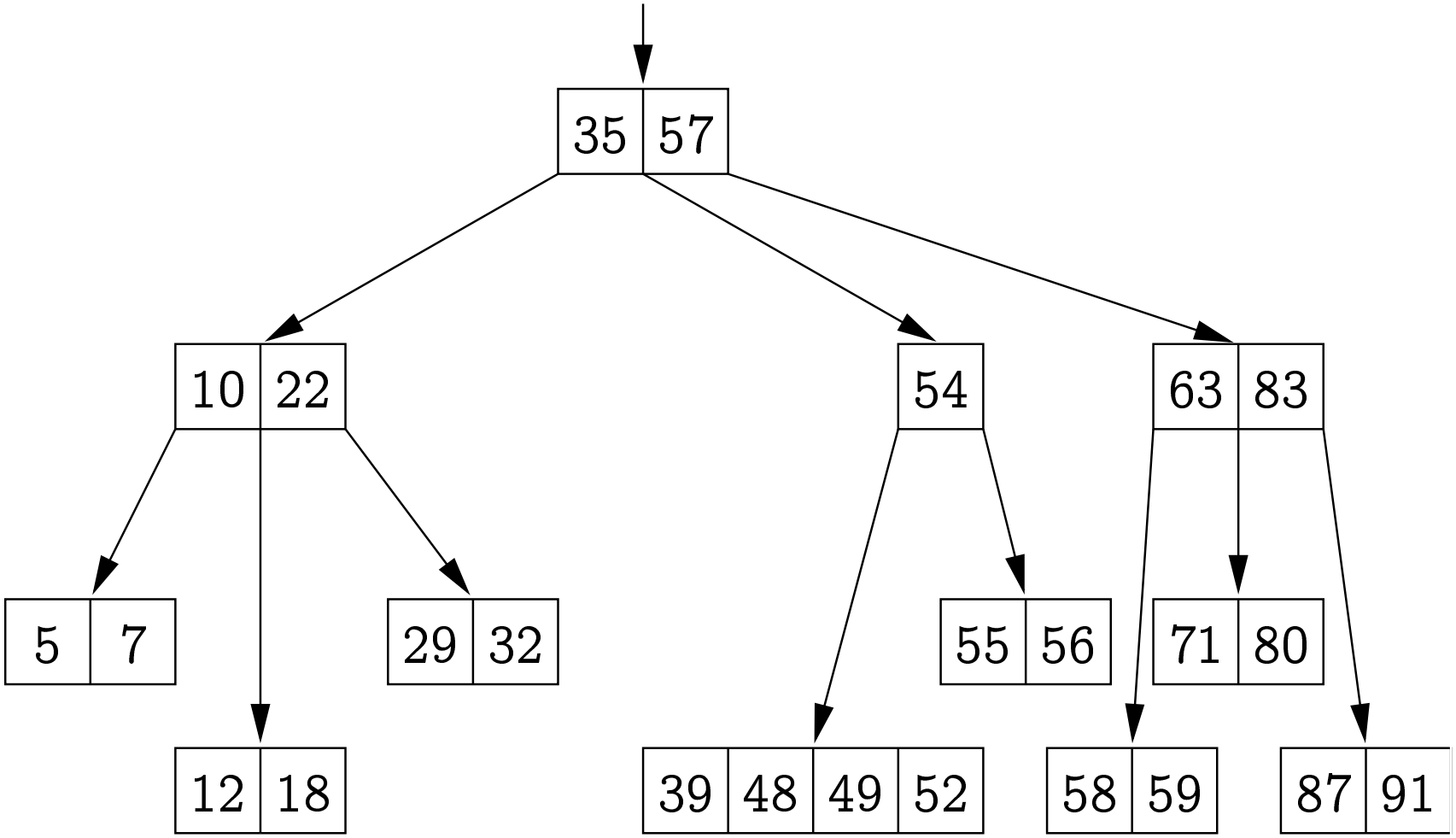
Eemaldame võtme 42



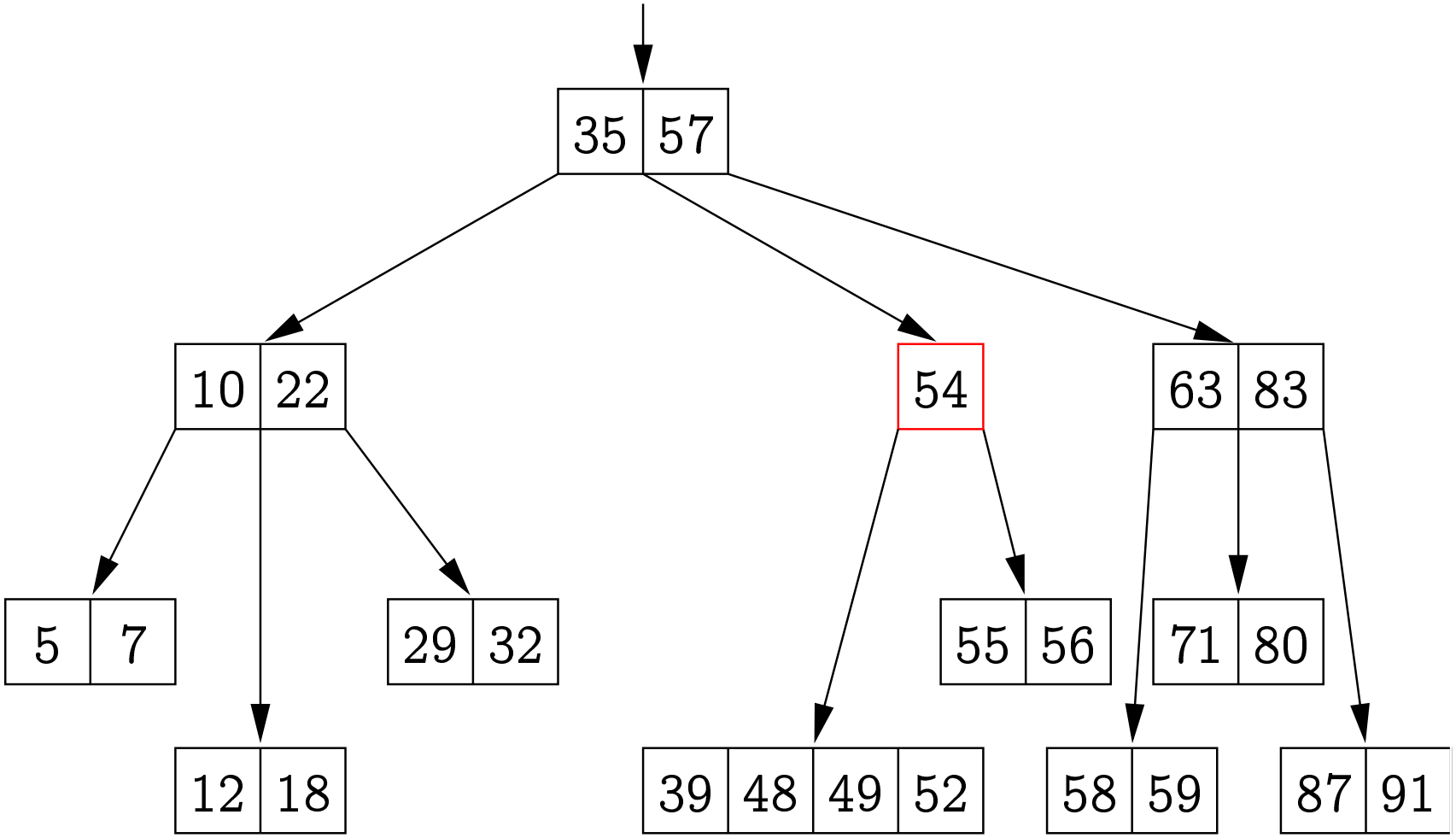
Eemaldame võtme 42



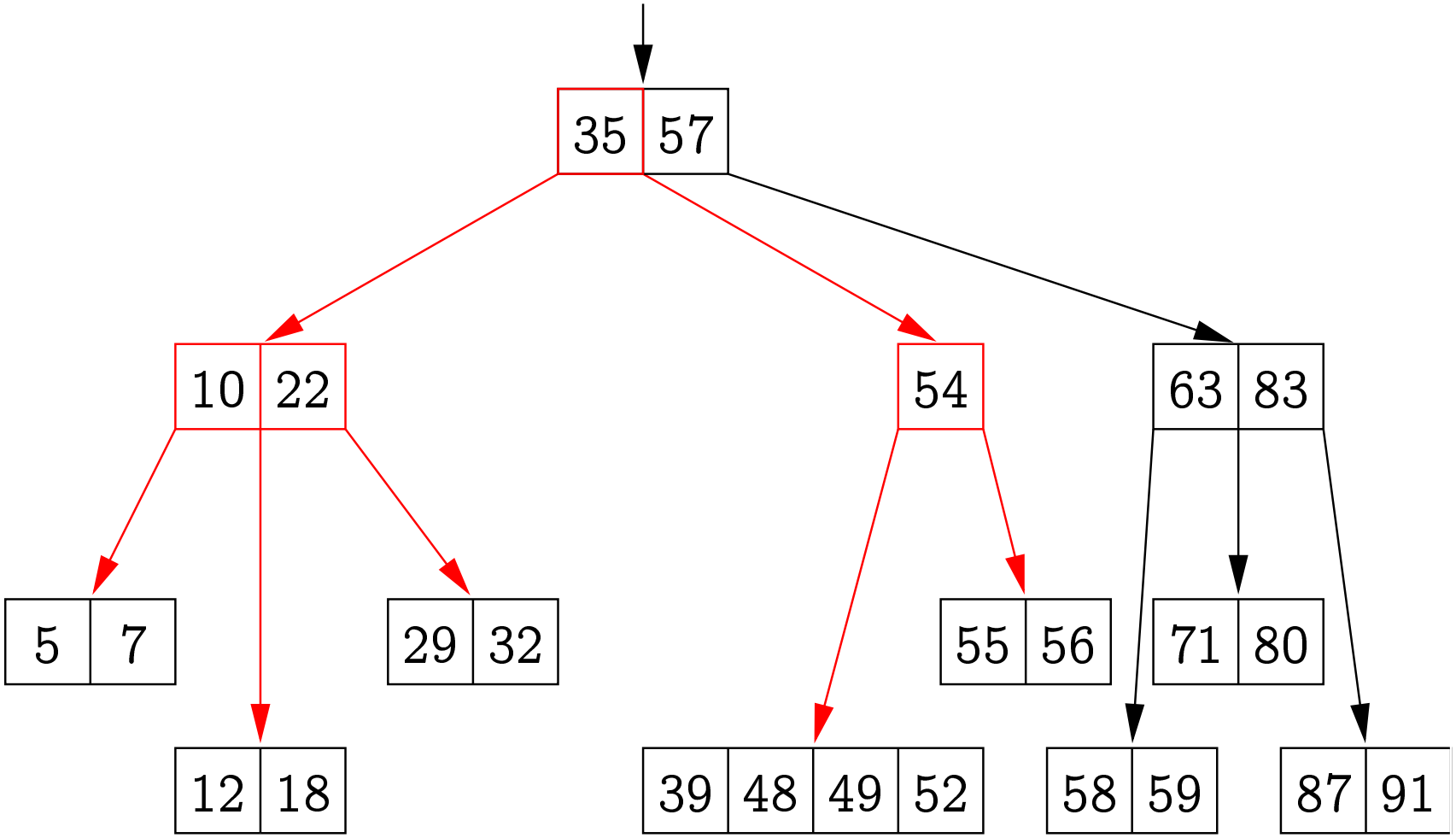
Eemaldame võtme 42



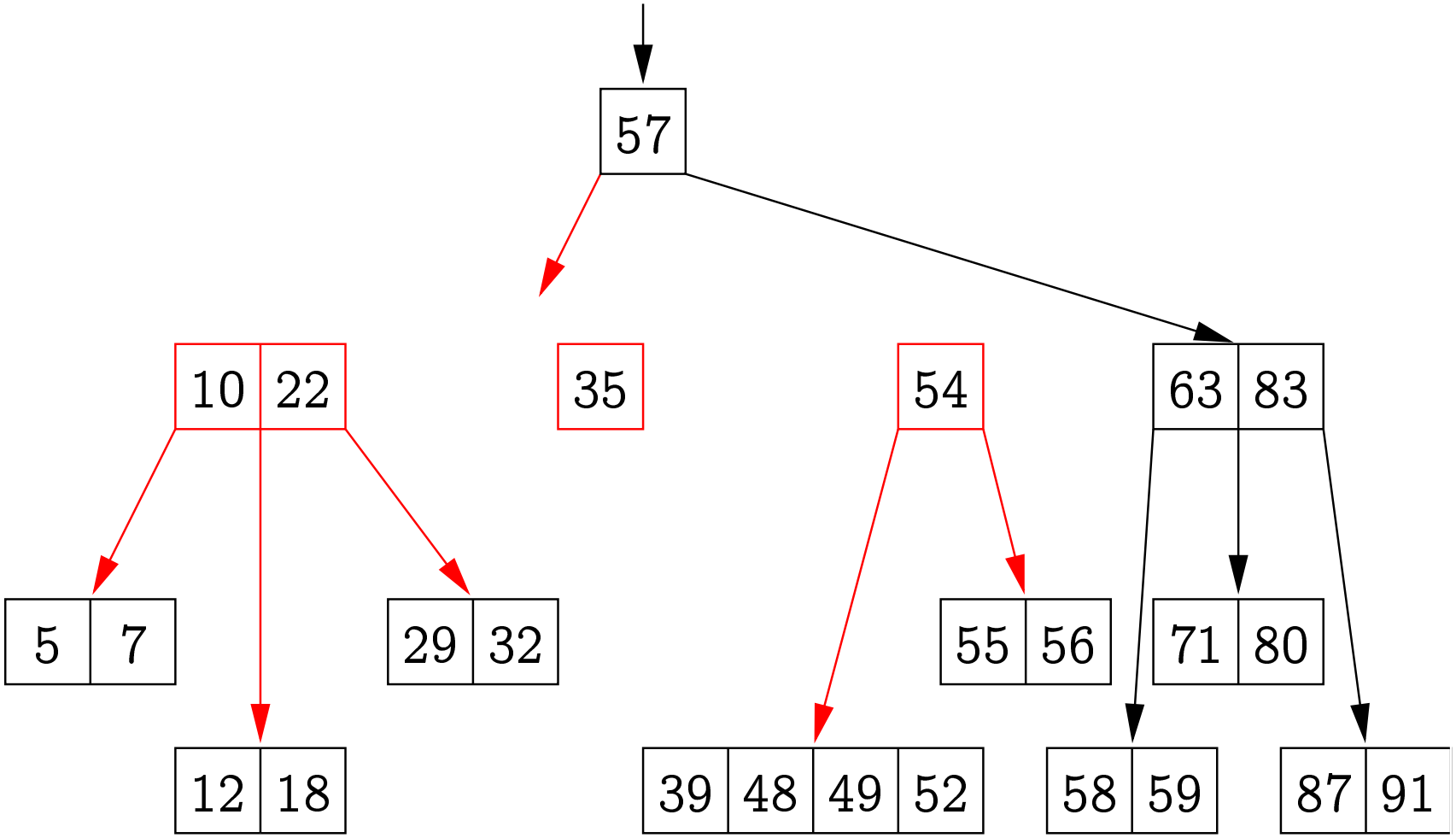
Eemaldame võtme 42



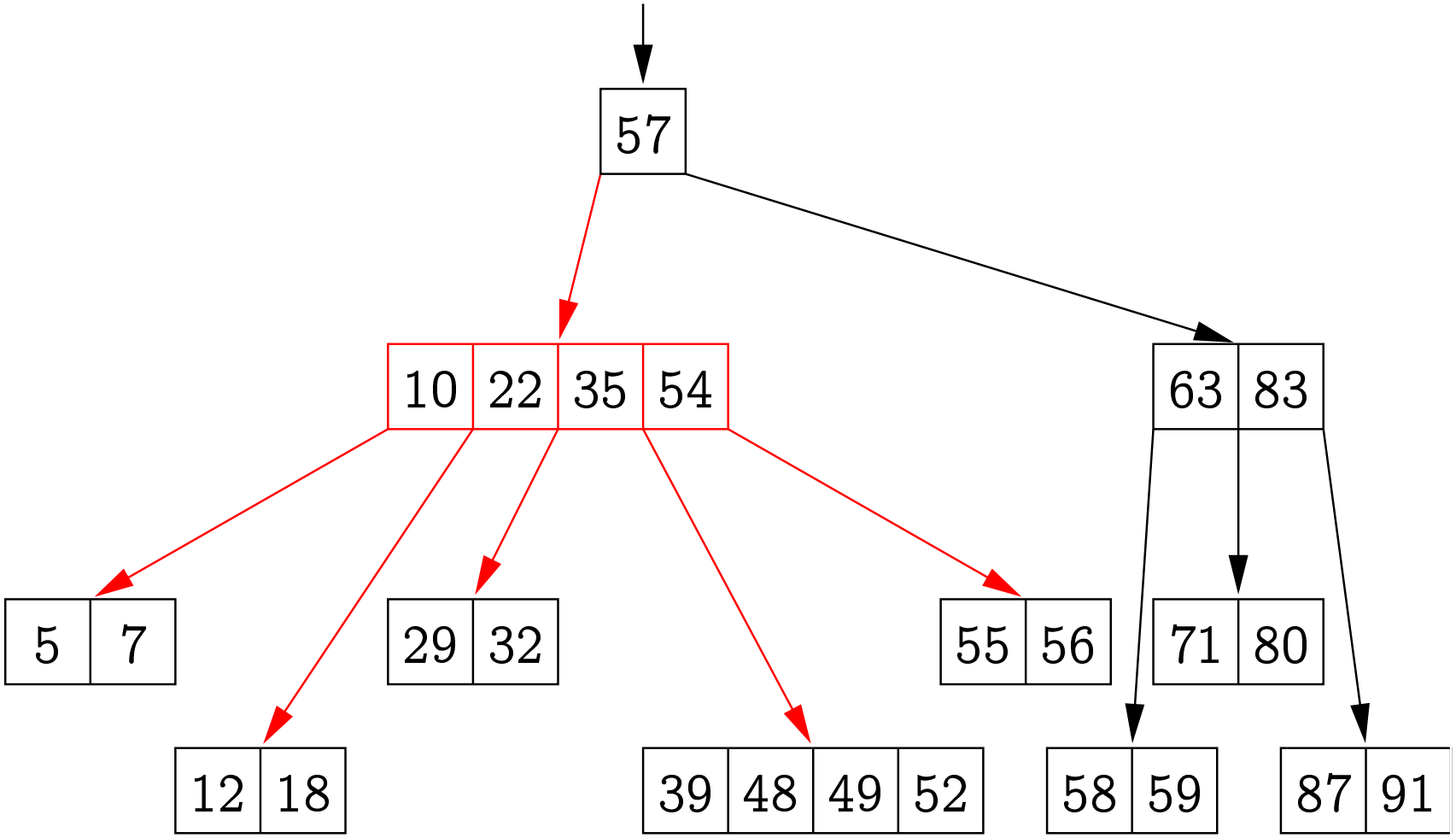
Eemaldame võtme 42



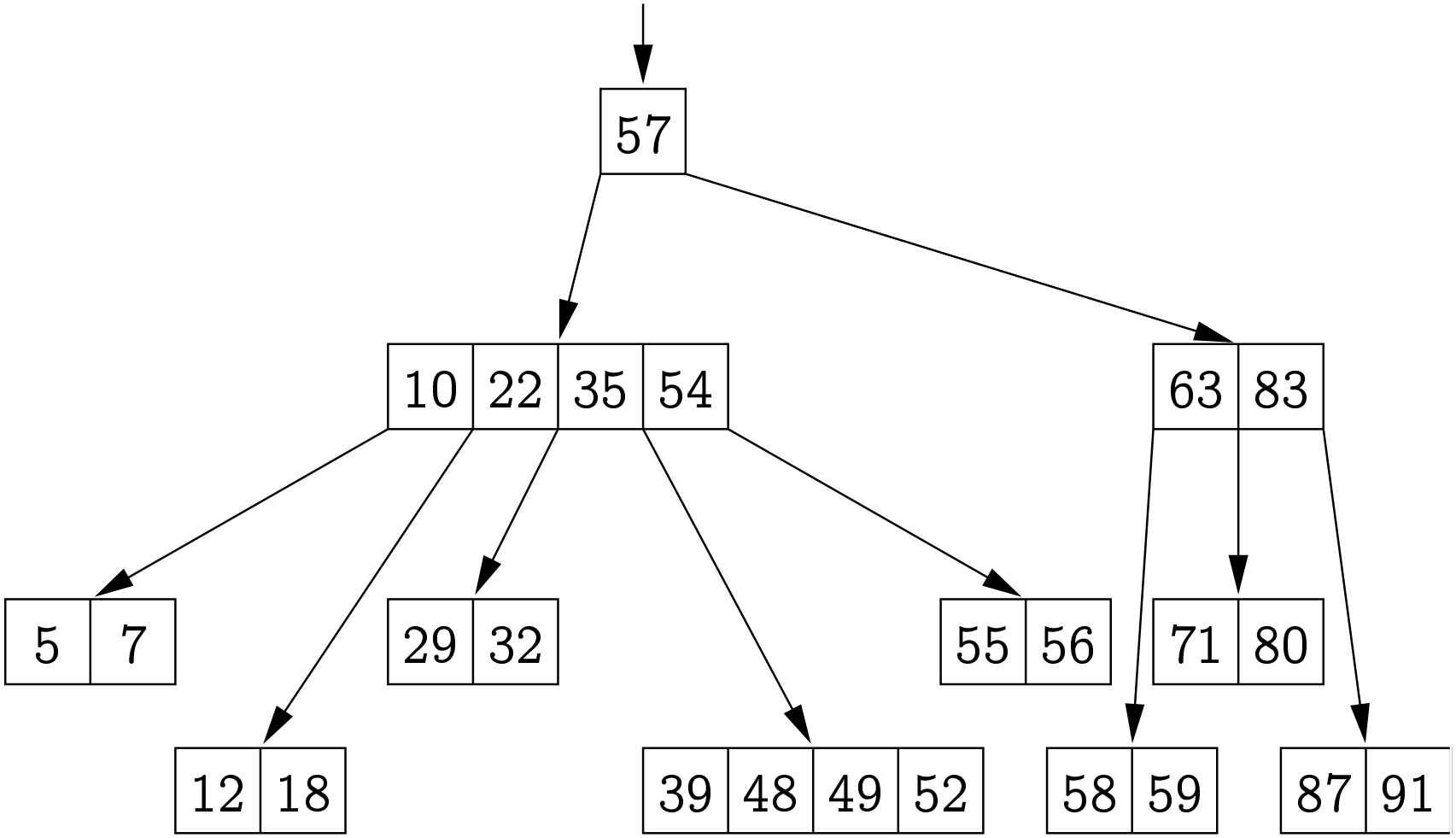
Eemaldame võtme 42



Eemaldame võtme 42

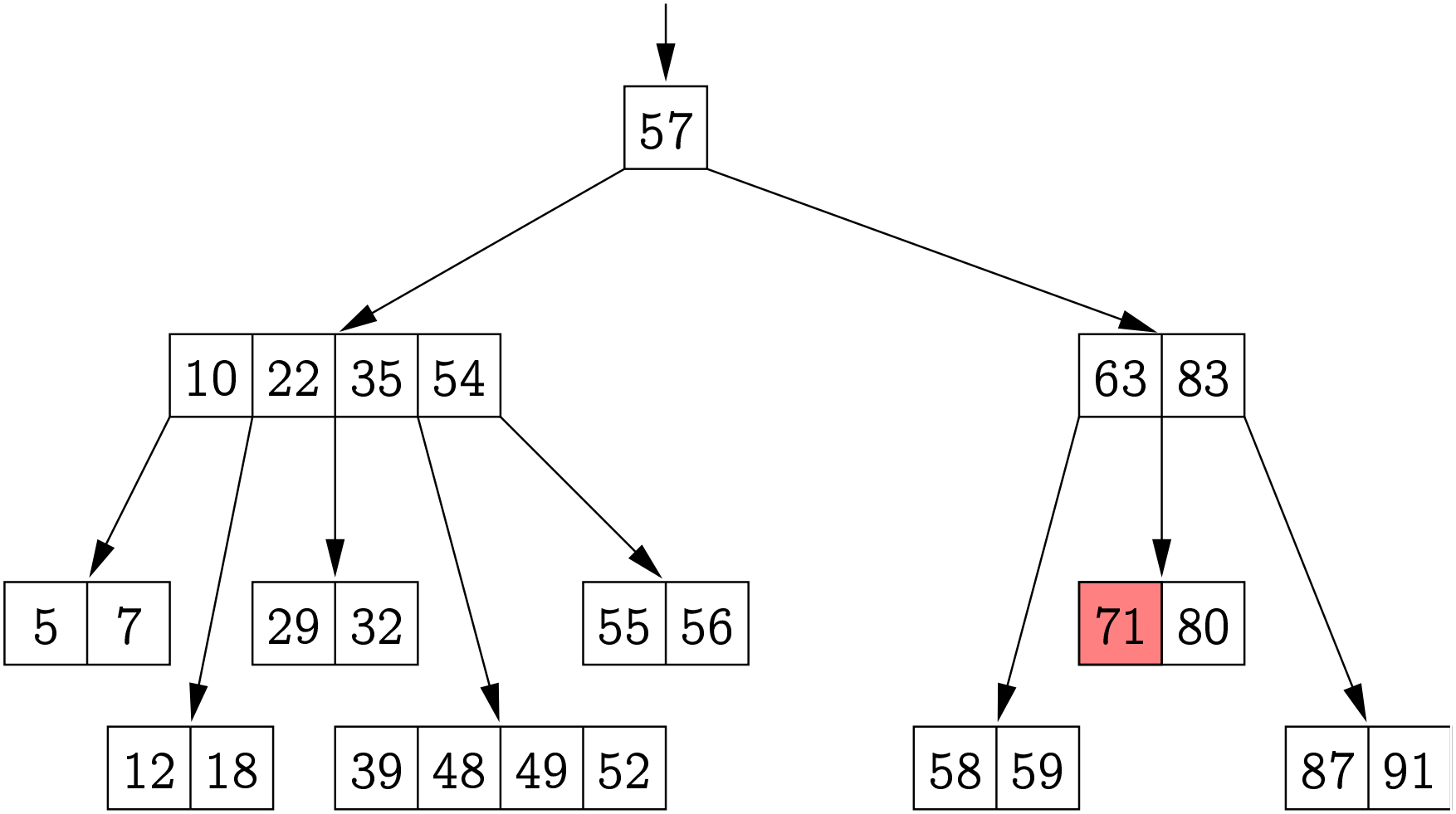


Eemaldame võtme 42

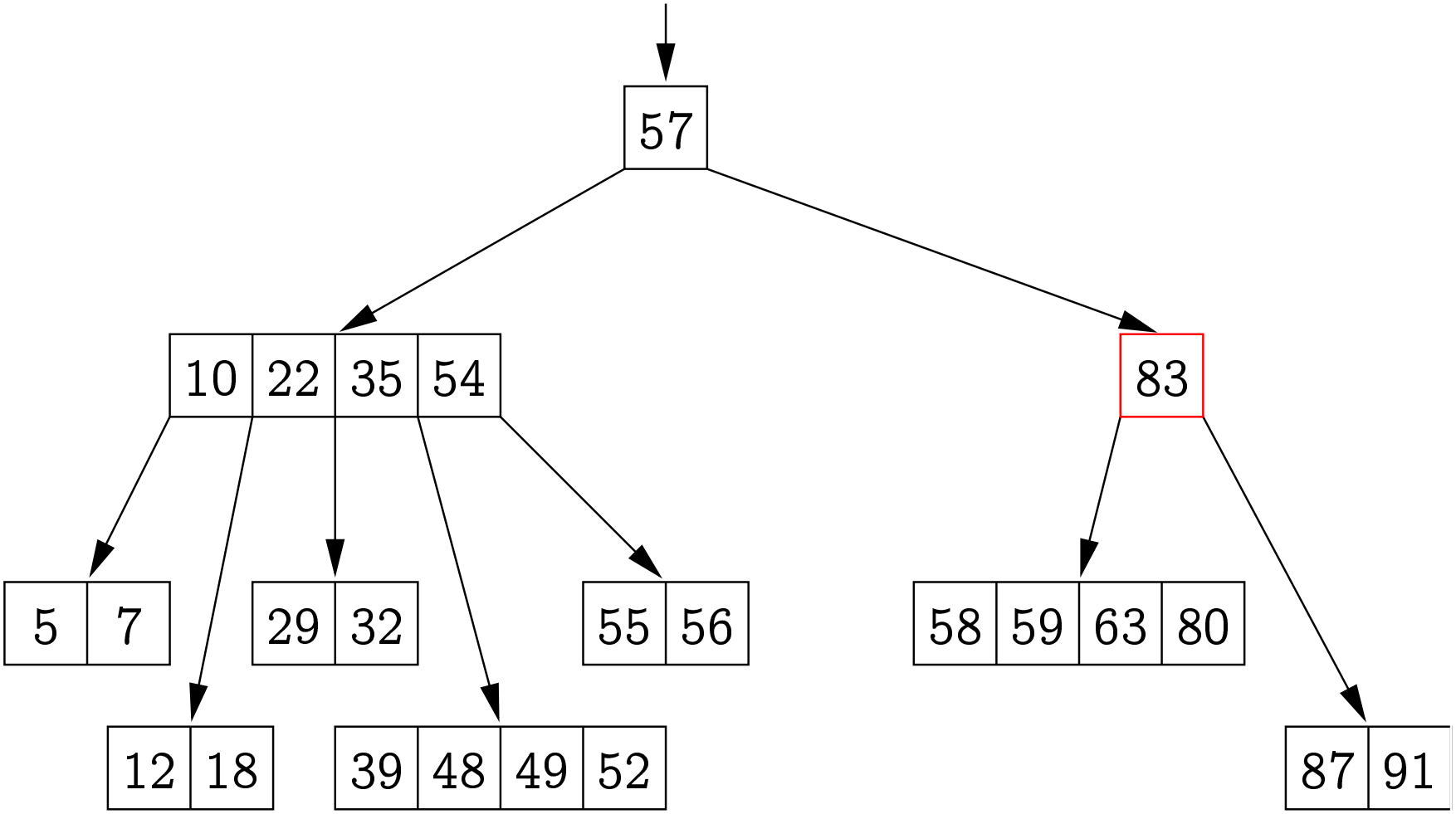


Veel üks näide naabrilt laenamisest.

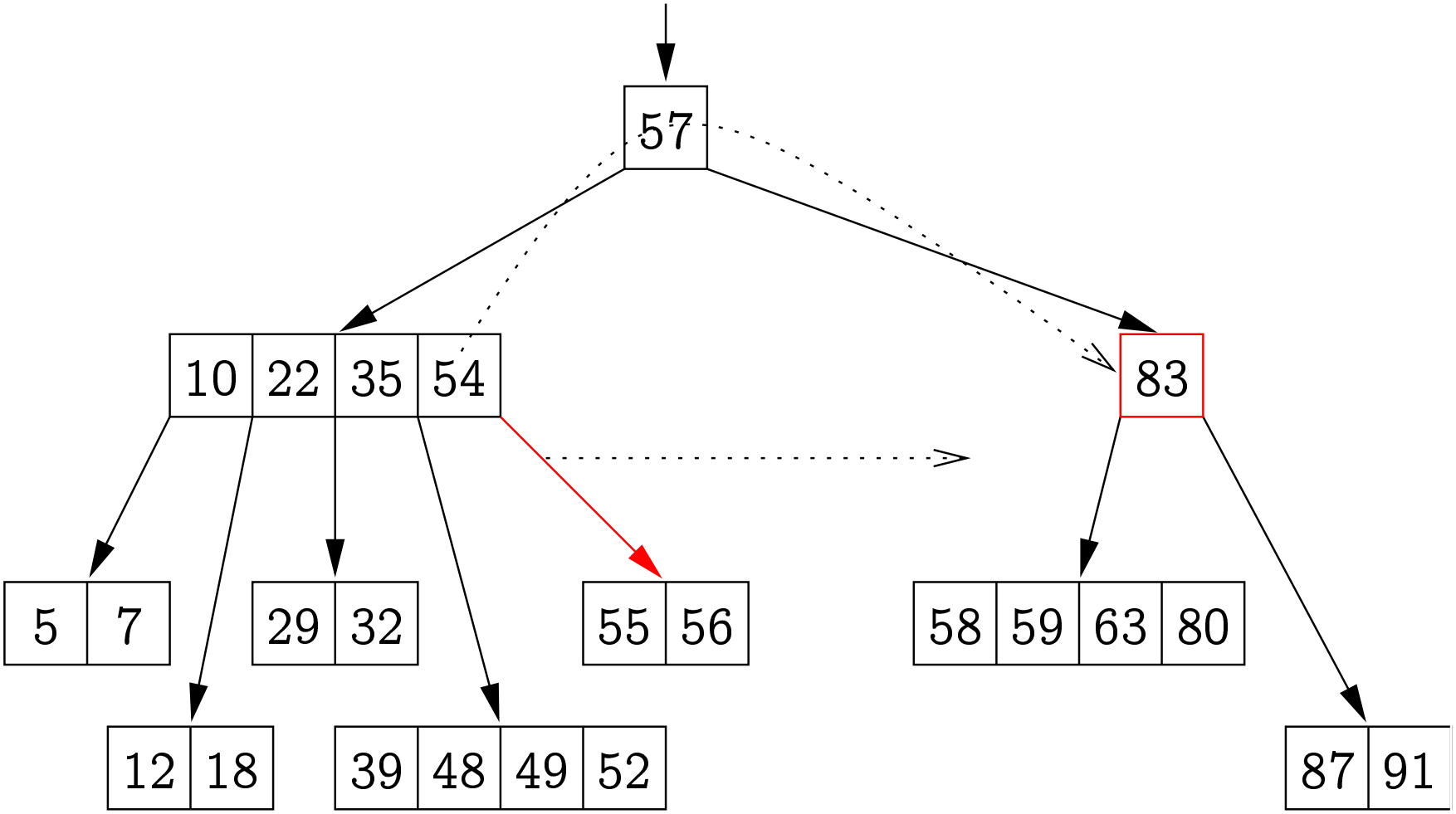
Eemaldame võtme 71



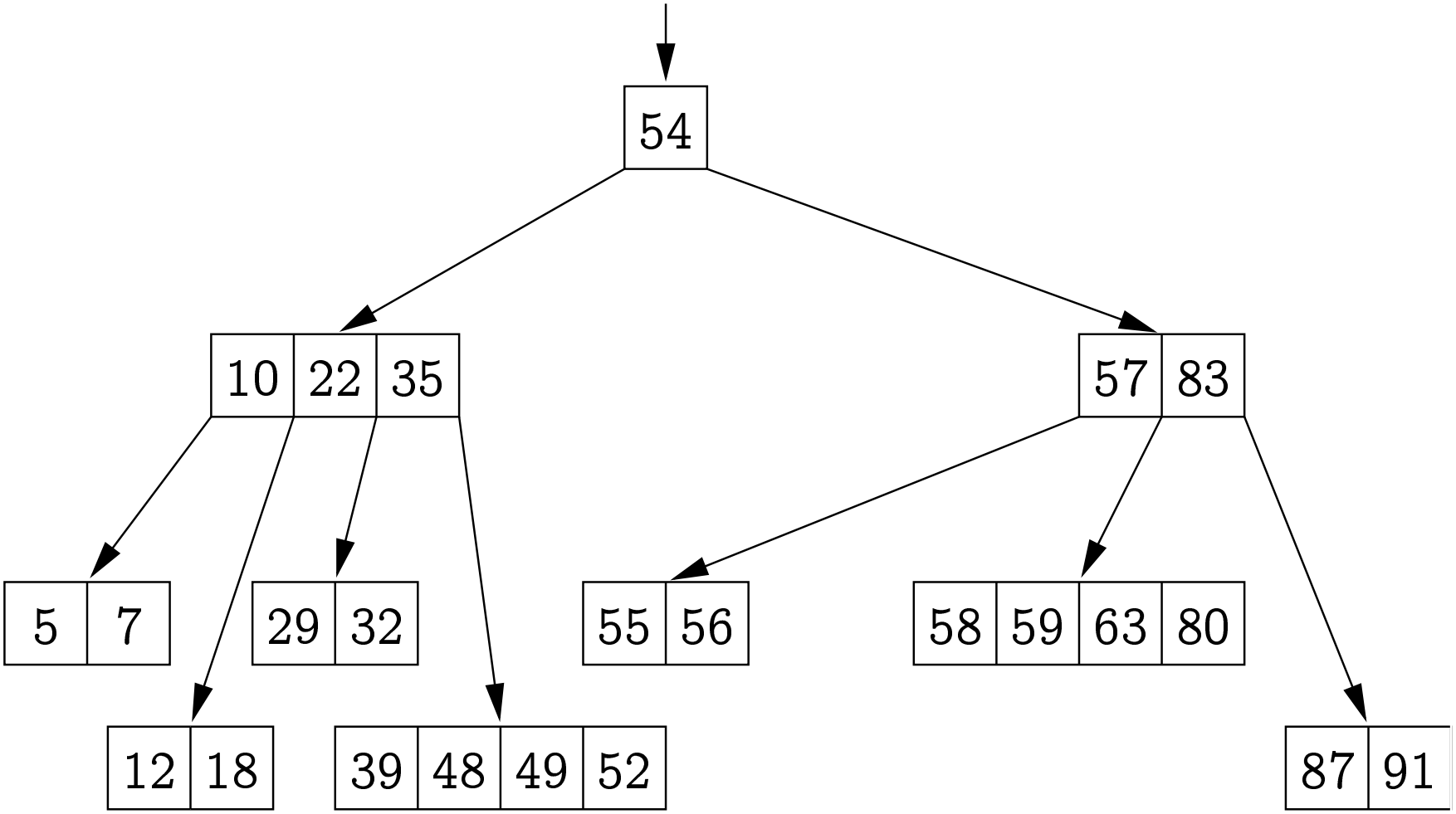
Eemaldame võtme 71



Eemaldame võtme 71



Eemaldame võtme 71



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

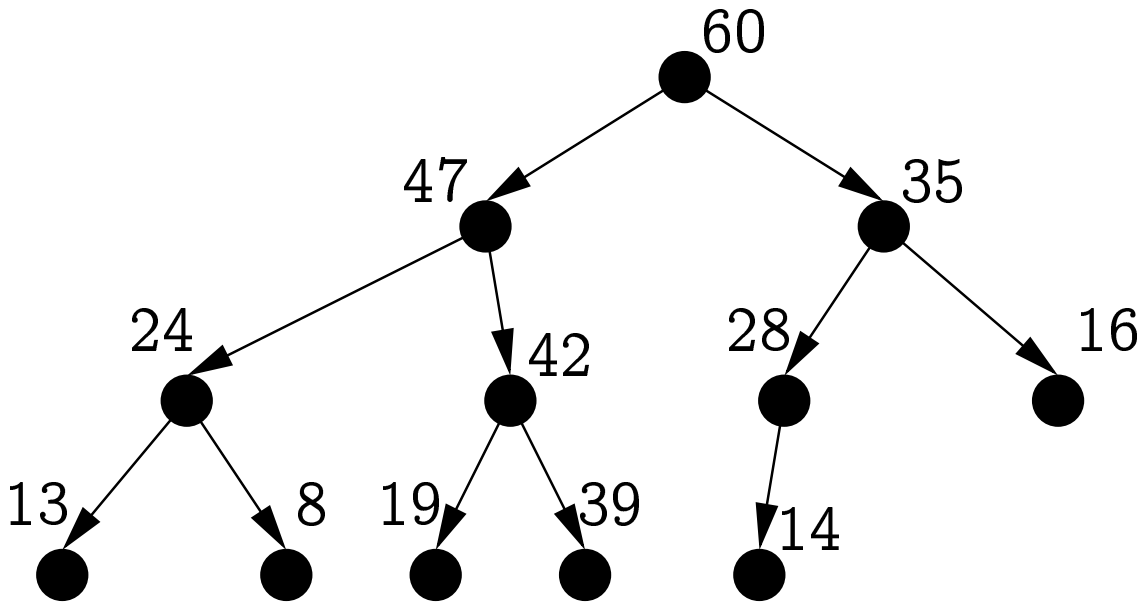
Kahendpuu on *täielik*, kui kõik tema lehed on juurest sama kaugel ja kõigi sisemiste tippude aste on 2.

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, tipp, uus_võti)	$\Theta(\log n)$
vähenda_võti(K, tipp, uus_võti)	$\Theta(\log n)$
kustuta(K, tipp)	$\Theta(\log n)$

Realisatsioon vajab kompaktse kahendpuu esitamist.

Kasutame selleks siis massiivi a. Olgu n hetkel massiivis a olevate elementide arv. Eeldame, et me ei lisa a-sse kunagi nii palju elemente, et enam ei mahu.

- Tipu i (s.t. kohal $a[i]$) vasak alluv on siis $2i$ (s.t. kohal $a[2i]$).
- Tipu i parem alluv on $2i + 1$.
- Tipu i ülemus on $\lfloor i/2 \rfloor$.

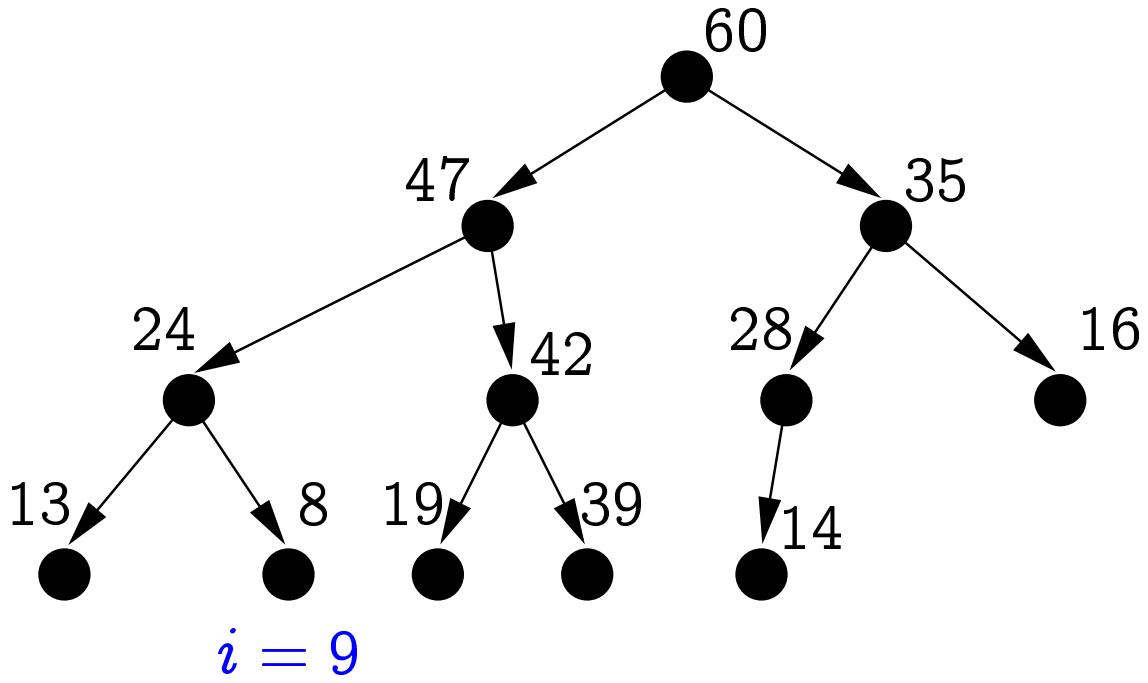
Kui mõni neist arvudest ei ole vahemikus $1..n$, siis vastavat alluvat või ülemust ei ole.

Tähistame $v(i)$, $p(i)$, $y(i)$.

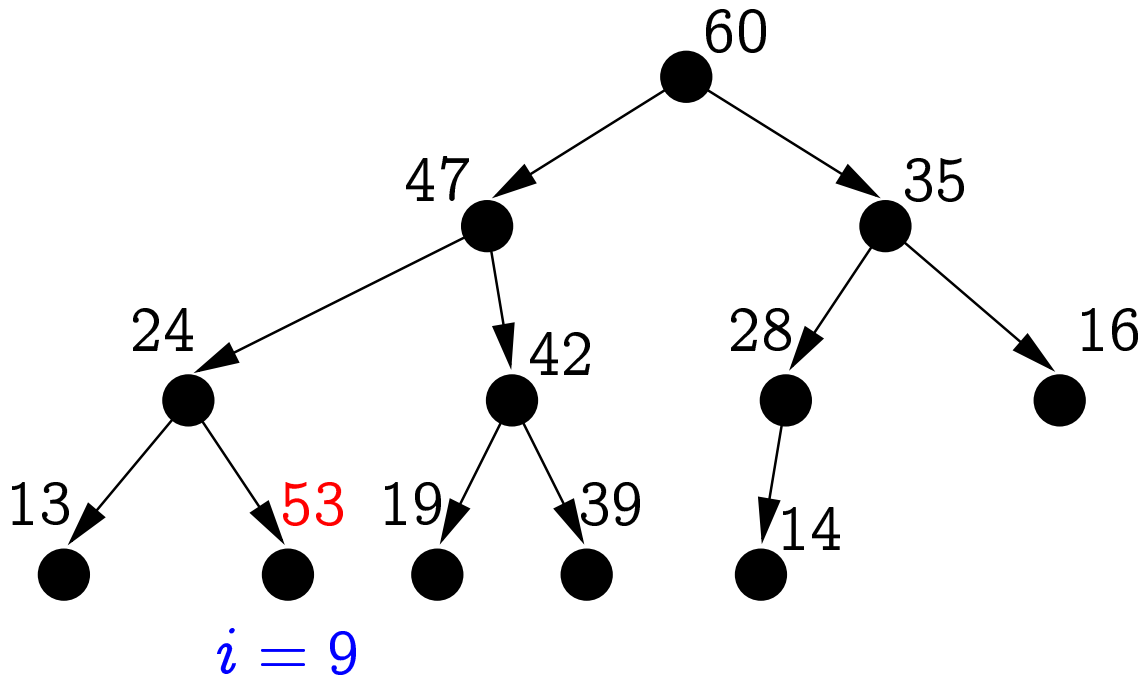
tee_kuhi: $n := 0$.

leia_max: $\mathbf{return}\ n > 0\ ?\ a[1] : \mathbf{error}$

Võtme suurendamine:



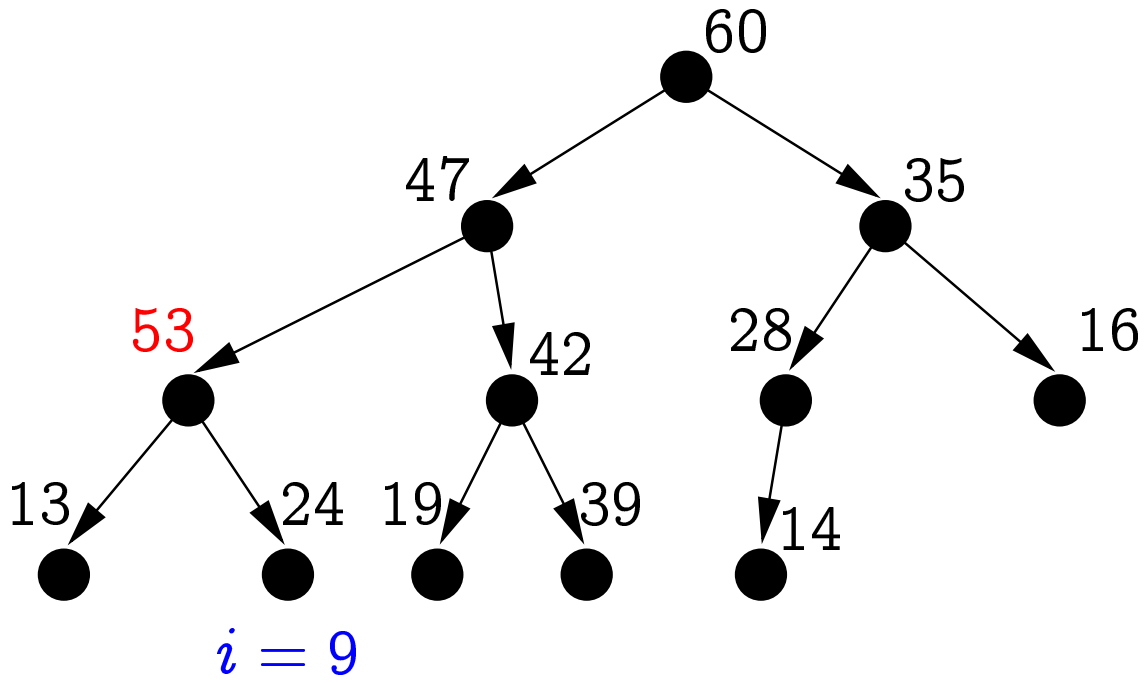
Võtame $a[9]$ uueks võtmeks 53.



Kutsume välja vii_üles(9)

vii_üles(i) on

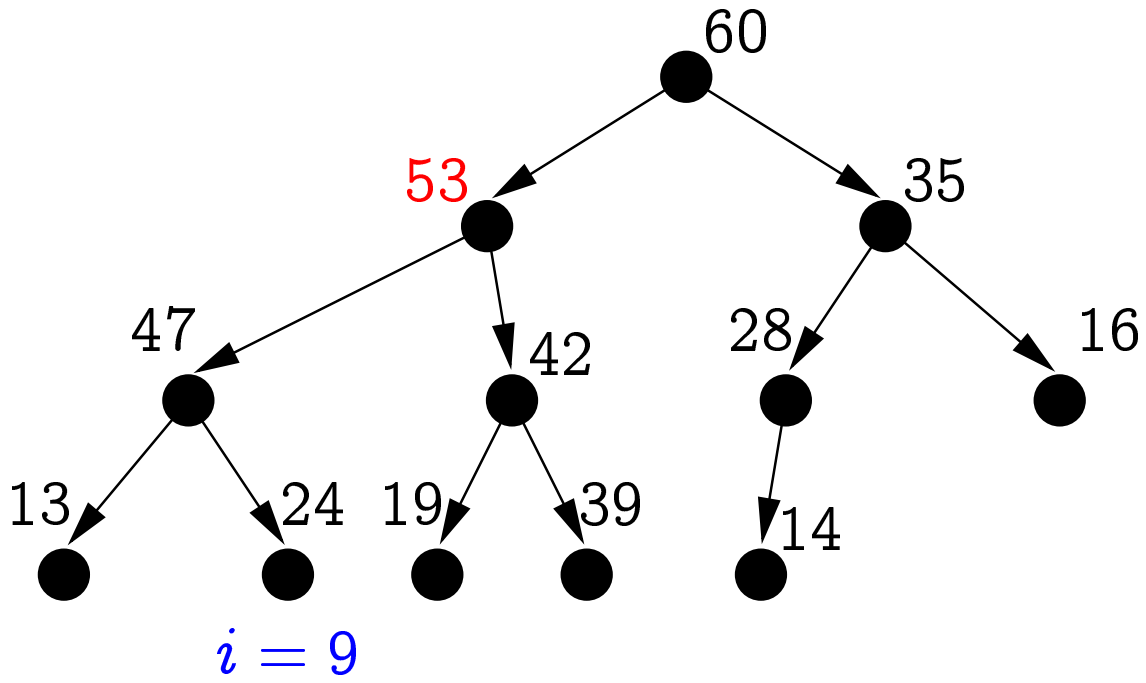
```
1   if i = 1 then return
2   if a[i] ⩽ a[y(i)] then
3       return
4   else
5       a[i] :=: a[y(i)]
6       vii_üles(y(i))
```



Kutsume välja vii_üles(4)

vii_üles(i) on

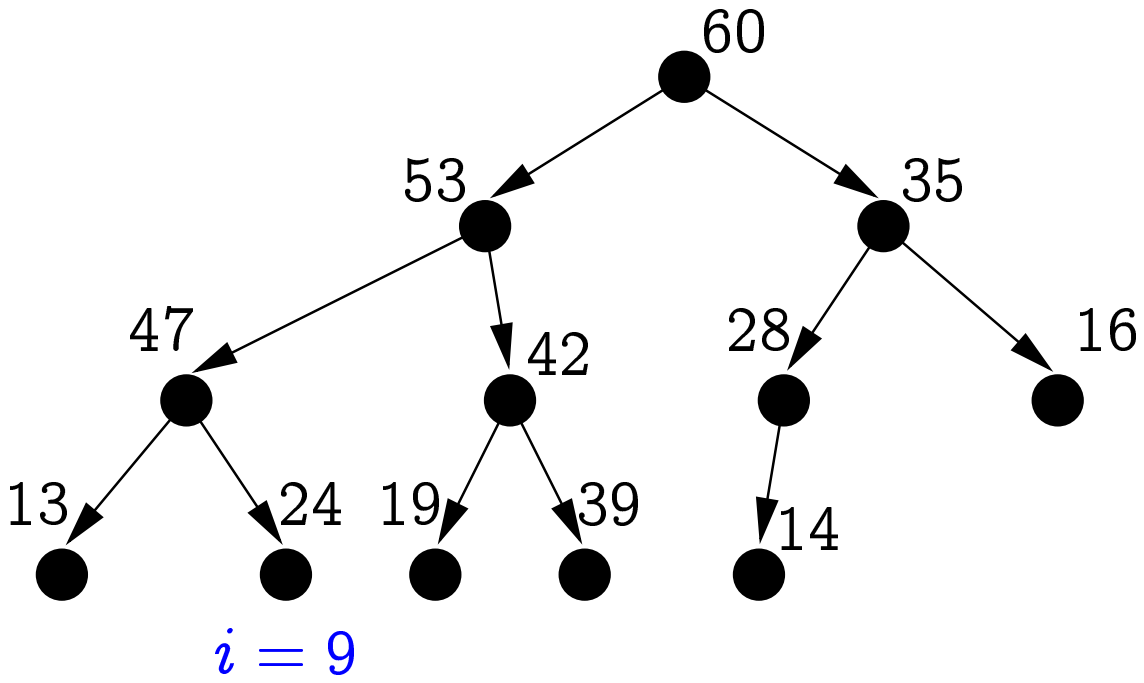
```
1   if i = 1 then return
2   if a[i] ⩽ a[y(i)] then
3       return
4   else
5       a[i] :=: a[y(i)]
6       vii_üles(y(i))
```



Kutsume välja vii_üles(2)

vii_üles(i) on

```
1   if i = 1 then return
2   if a[i] ⩽ a[y(i)] then
3       return
4   else
5       a[i] :=: a[y(i)]
6       vii_üles(y(i))
```



vii_üles(i) on

```
1   if i = 1 then return
2   if a[i] ⩽ a[y(i)] then
3       return
4   else
5       a[i] :=: a[y(i)]
6       vii_üles(y(i))
```

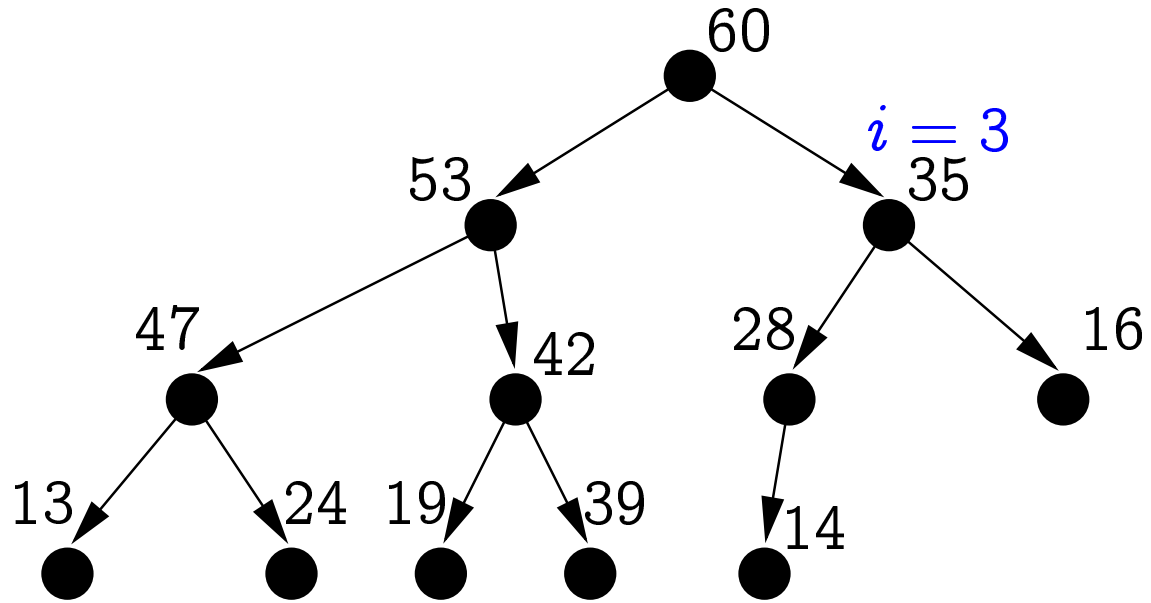
suurenda_võti(K, tipp, uus_võti) on siis

1. a[tipp].võti := uus_võti
2. vii_üles(tipp)

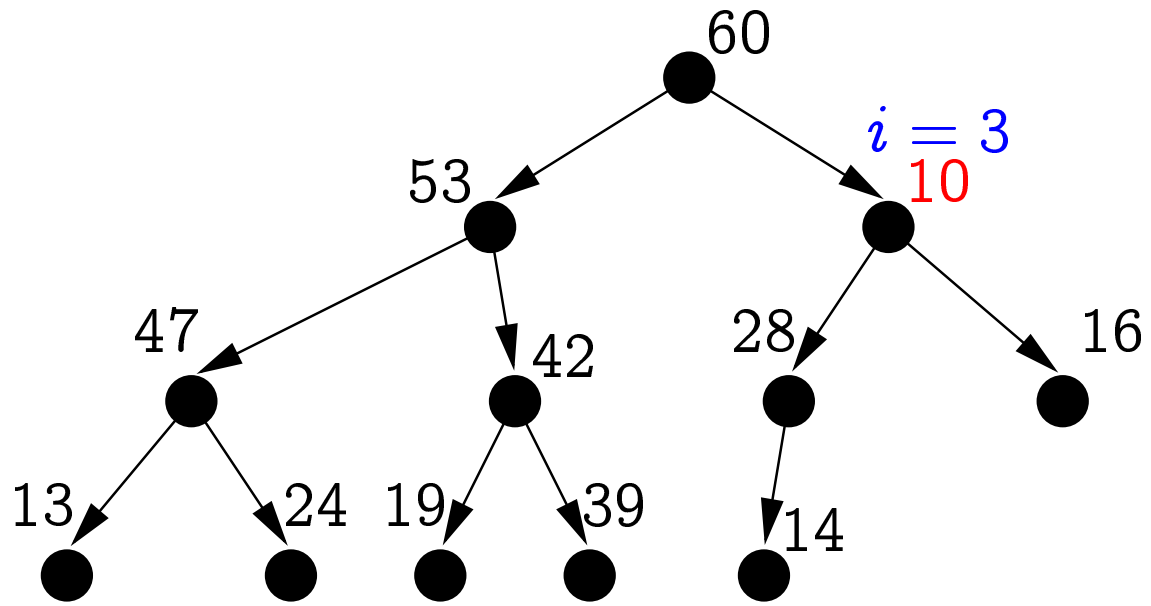
Keerukus:

- vii_üles rekursiivsete väljakutsete arv pole suurem kui puu kõrgus.
- vii_üles mitterekursiivne osa on konstantse keerukusega.
- Puu kõrgus on $\Theta(\log n)$.
- Ajaline keerukus halvimal juhul on ka $\Theta(\log n)$.

Võtme vähendamine:



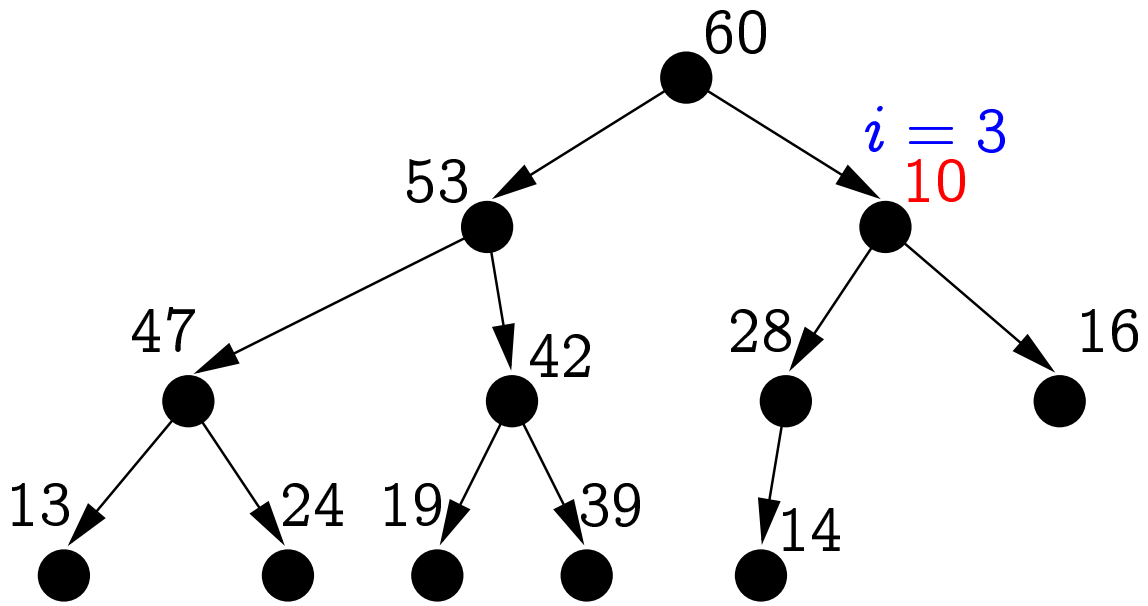
Võtame $a[3]$ uueks võtmeks 10.



Kutsume välja vii_alla(3)

vii_alla(i) on

```
 1   if v(i) > n then return
 2   if p(i) > n or
             a[v(i)] > a[p(i)] then
 3       x := v(i)
 4   else
 5       x := p(i)
 6   if a[i] ⩾ a[x] then
 7       return
 8   else
 9       a[i] :=: a[x]
10       vii_alla(x)
```

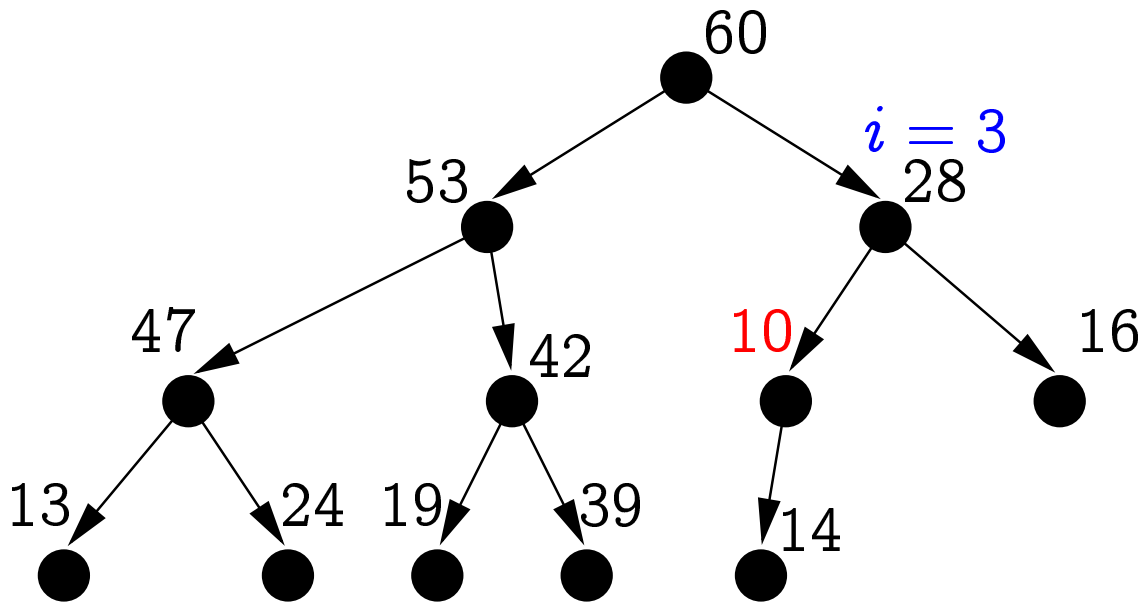


Kutsume välja vii_alla(3)

$x = 6$

vii_alla(i) on

```
 1   if v(i) > n then return
 2   if p(i) > n or
             a[v(i)] > a[p(i)] then
 3       x := v(i)
 4   else
 5       x := p(i)
 6   if a[i] ⩾ a[x] then
 7       return
 8   else
 9       a[i] :=: a[x]
10       vii_alla(x)
```

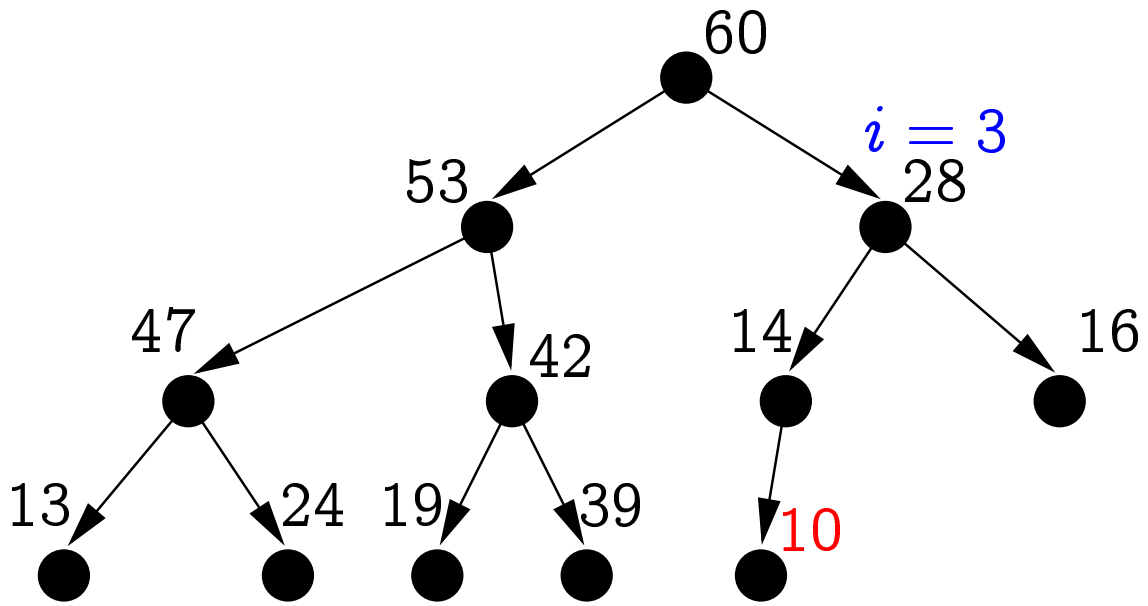


Kutsume välja vii_alla(6)

$x = 12$

vii_alla(i) on

```
 1   if v(i) > n then return
 2   if p(i) > n or
             a[v(i)] > a[p(i)] then
 3       x := v(i)
 4   else
 5       x := p(i)
 6   if a[i] ⩾ a[x] then
 7       return
 8   else
 9       a[i] :=: a[x]
10       vii_alla(x)
```

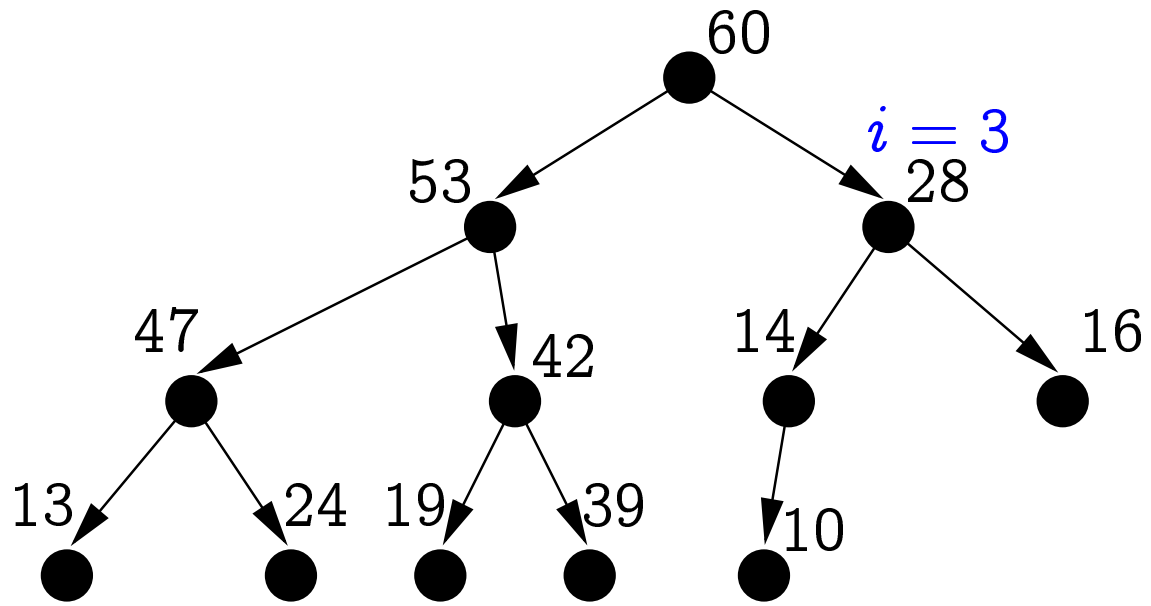


Kutsume välja vii_alla(12)

leht

vii_alla(i) on

```
 1   if v(i) > n then return
 2   if p(i) > n or
             a[v(i)] > a[p(i)] then
 3       x := v(i)
 4   else
 5       x := p(i)
 6   if a[i] ⩾ a[x] then
 7       return
 8   else
 9       a[i] :=: a[x]
10       vii_alla(x)
```



Kutsume välja vii_alla(12)

vii_alla(i) on

```
 1   if v(i) > n then return
 2   if p(i) > n or
             a[v(i)] > a[p(i)] then
 3       x := v(i)
 4   else
 5       x := p(i)
 6   if a[i] ⩾ a[x] then
 7       return
 8   else
 9       a[i] :=: a[x]
10       vii_alla(x)
```

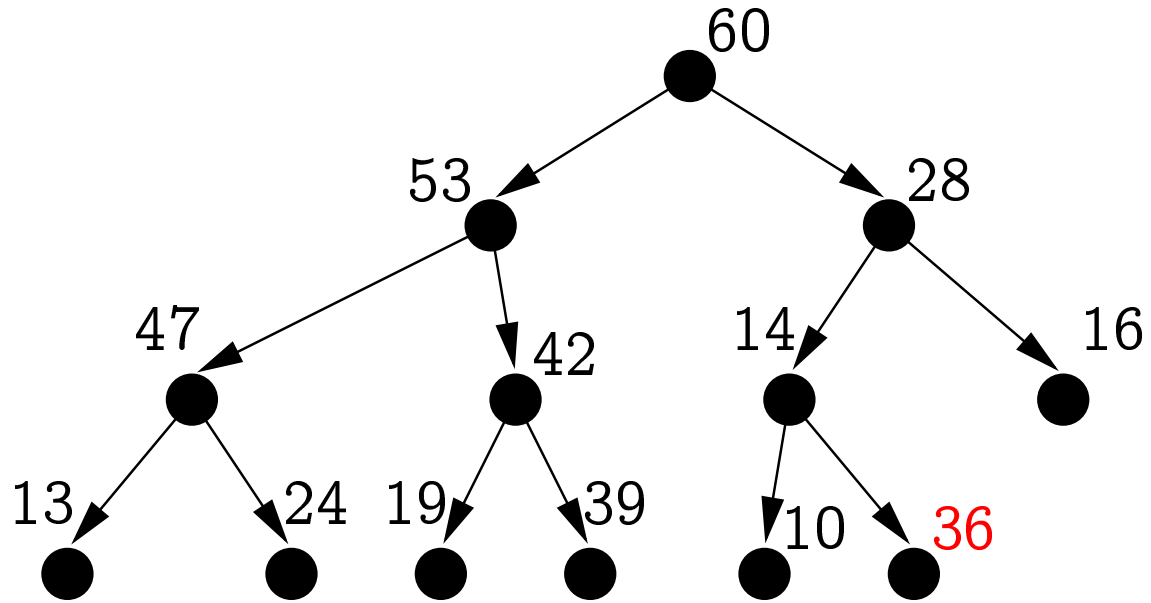
vähenda_võti(K, tipp, uus_võti) on siis

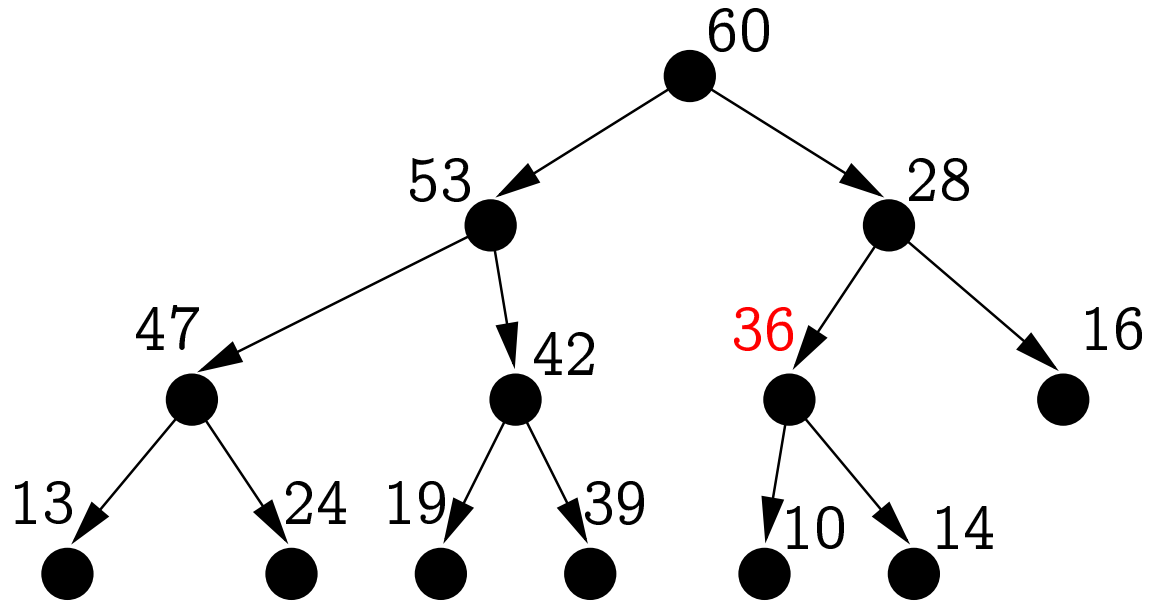
1. a[tipp].võti := uus_võti
2. vii_alla(tipp)

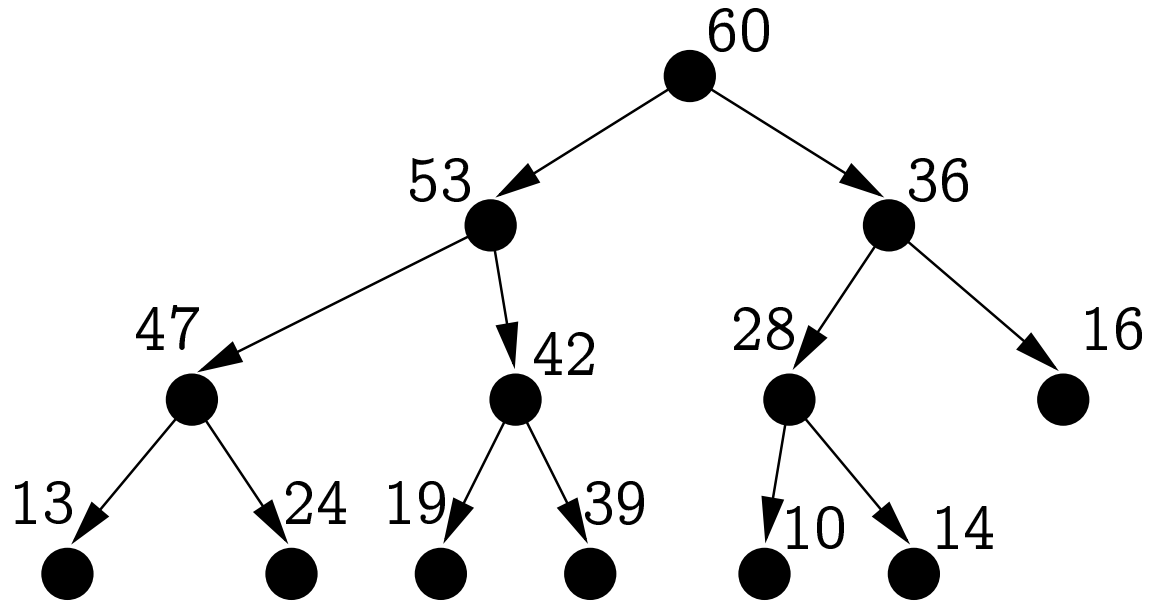
Keerukus: $\Theta(\log n)$ nagu võtme suurendamiselgi.

Kirje lisamine: lisame uue kirje massiivi a lõppu ja viime ta üles.

```
1  n := n + 1
2  a[n] := uus kirje
3  vii_üles(n)
```


60
53
28
47
42
14
16
13
24
19
39
10
36


60
53
28
47
42
36
16
13
24
19
39
10
14

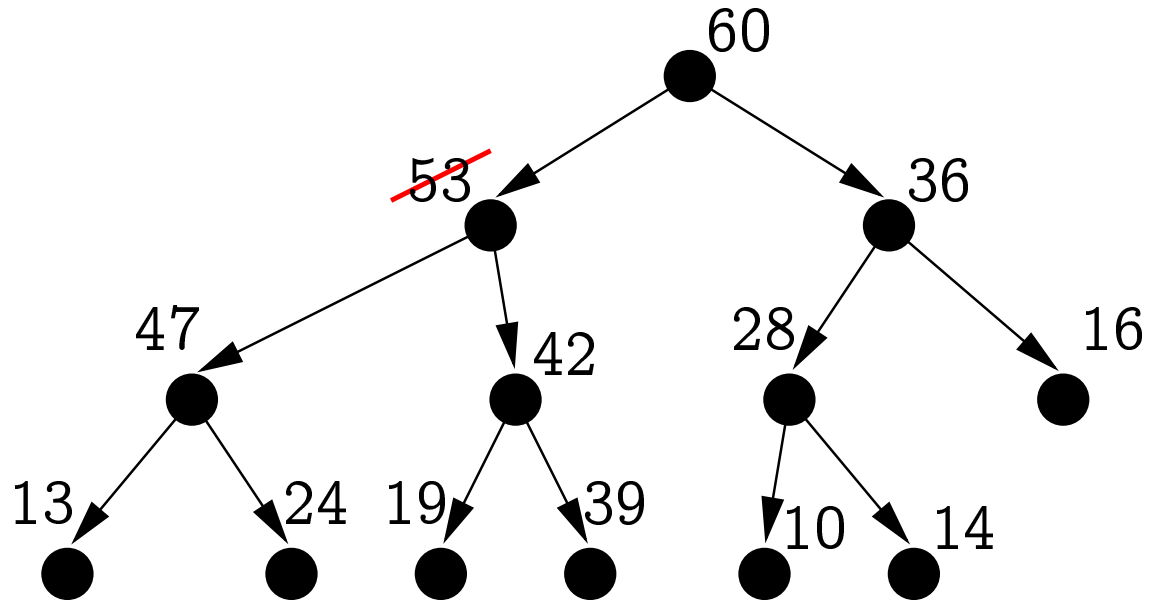

60
53
36
47
42
28
16
13
24
19
39
10
14

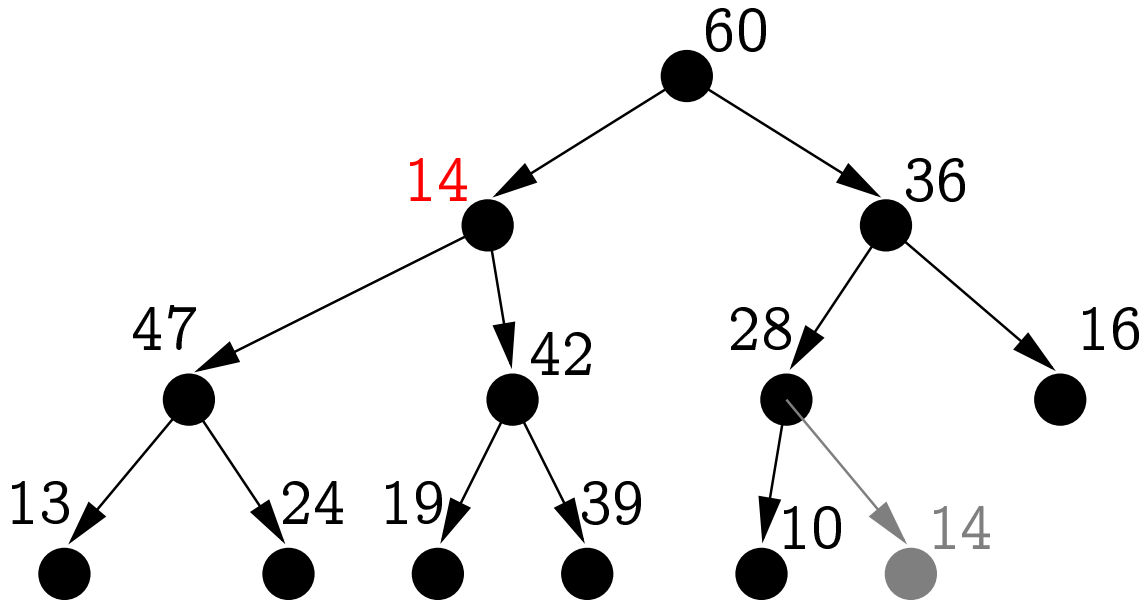
Kirje i eemaldamine: kirjutame ta üle viimase kirjega, viskame viimase kirje ära ning viime muudetud kirje õigele kohale (üles või alla).

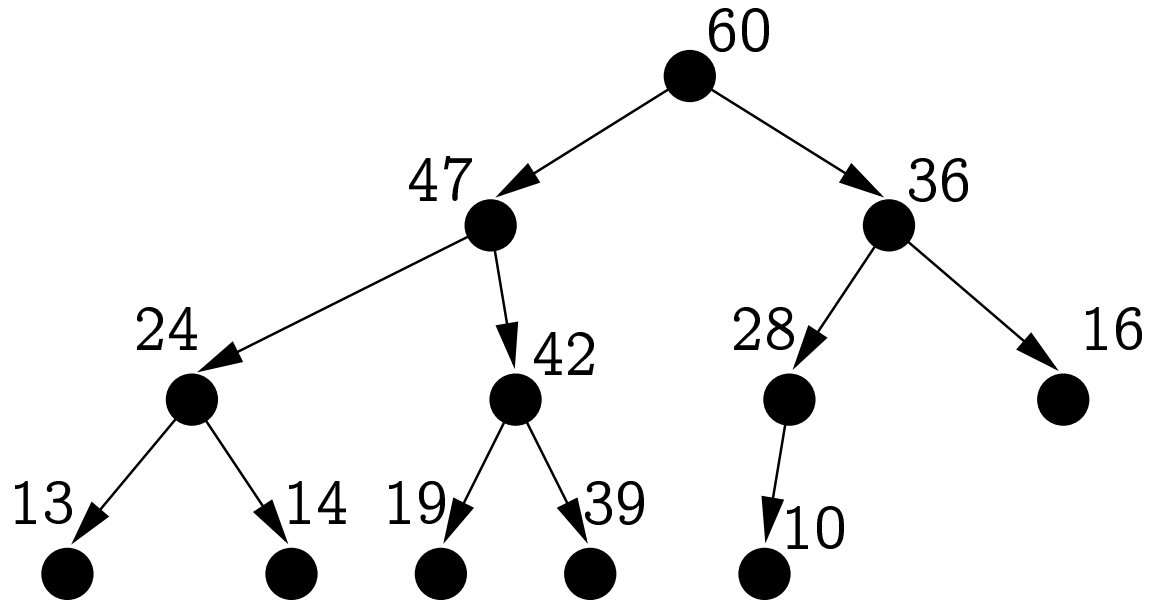
Kustuta(K, i) on

```
1  if i = n then n := n − 1; return
2  v := a[i].võti
3  a[i] := a[n]
4  n := n − 1
5  if a[i] > v then
6      vii_üles(i)
7  else
8      vii_alla(i)
```

Maksimaalse elemendi võtmine — leidmine ja eemaldamine.


60
53
36
47
42
28
16
13
24
19
39
10
14


60
14
36
47
42
28
16
13
24
19
39
10
14


60
47
36
24
42
28
16
13
14
19
39
10

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.

vii_alla väljakutsete arv mingi tipu jaoks on piiratud selle tipu kaugusega lehtedest.

Väljakutseid on ülimalt...

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

Kahendkuhja 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

```
 1  if a.n = a.maxn then
 2      b := new(kirjetüüp[2 * a.maxn])
 3      b.maxn := 2 * a.maxn
 4      for i := 1 to a.maxn do
 5          b[i] := a[i]
 6      b.n := a.n
 7      free(a)
 8      a := b
 9  a.n := a.n + 1
10  a[a.n] := x
11  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:

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

Ühe väljakutse aeg — keskmiselt ülimalt 3 ajaühikut.

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

vähenda_tabel(a) on

```
 1  a.n := a.n − 1
 2  if a.n ⩽ a.maxn/4 then
 3      b := new(kirjetüüp[a.maxn/2])
 4      b.maxn := a.maxn/2
 5      for i := 1 to a.n do
 6          b[i] := a[i]
 7      b.n := a.n
 8      free(a)
 9      a := b
10  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 operatsioonijadade 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.

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

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

  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 $O(1)$.

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 $B_{i-1}, B_{i-2}, \ldots, B_1, B_0$.

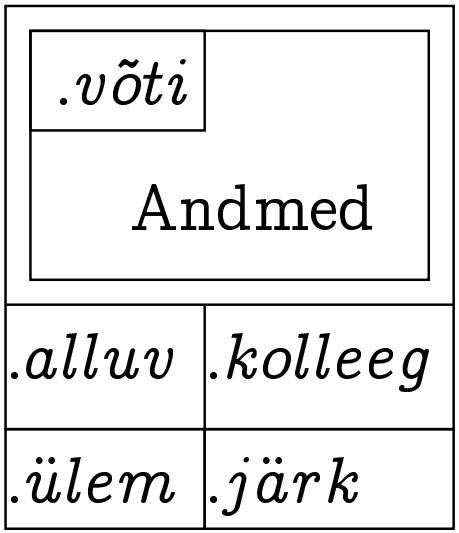
Binomiaalkuhi on binomiaalpuude hulk, kus kõigi puude järk on erinev ja kõik puud rahuldavad kuhjaomadust.

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

Binomiaalkuhi K (suurusega n) toetab järgmisi operatsioone:

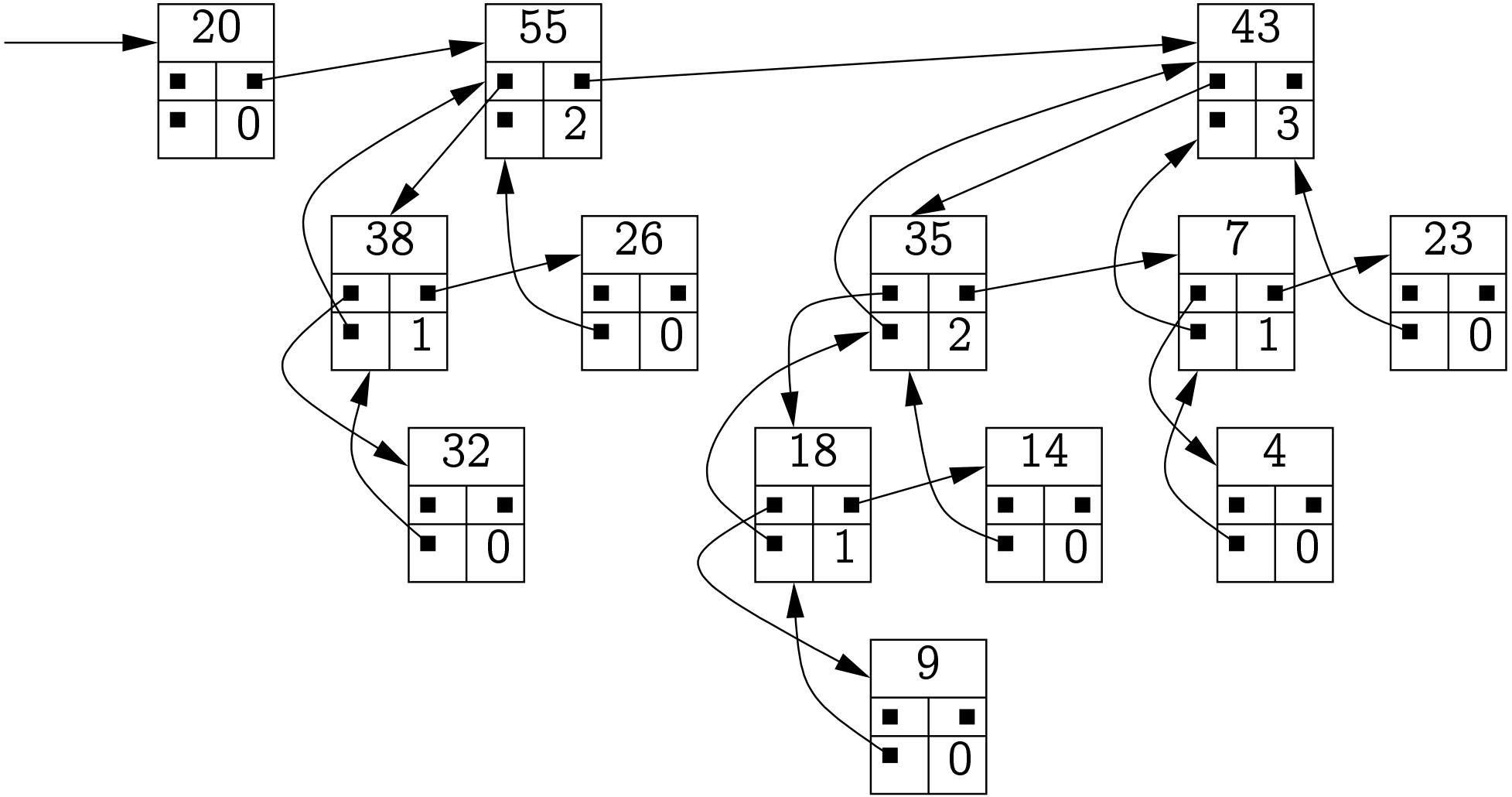
tee_kuhi	$\Theta(1)$
lisa(K, x)	$\Theta(\log n)$
leia_max(K)	$\Theta(\log n)$
võta_max(K)	$\Theta(\log n)$
suurenda_võti$(K, \text{tipp}, \text{uus_võti})$	$\Theta(\log n)$
vähenda_võti$(K, \text{tipp}, \text{uus_võti})$	$\Theta(\log n)$
kustuta(K, tipp)	$\Theta(\log n)$
ühenda(K_1, K_2)	$\Theta(\log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

Binomiaalkuhja võib mälus kujutada struktuurina, kus igale tipule vastab kirje



- 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

```
1  R := K.Andmed
2  p := K.kolleeg
3  while p ≠ NIL do
4      if p.võti > R.võti then
5          R := p.Andmed
6      p := p.kolleeg
7  return R
```

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

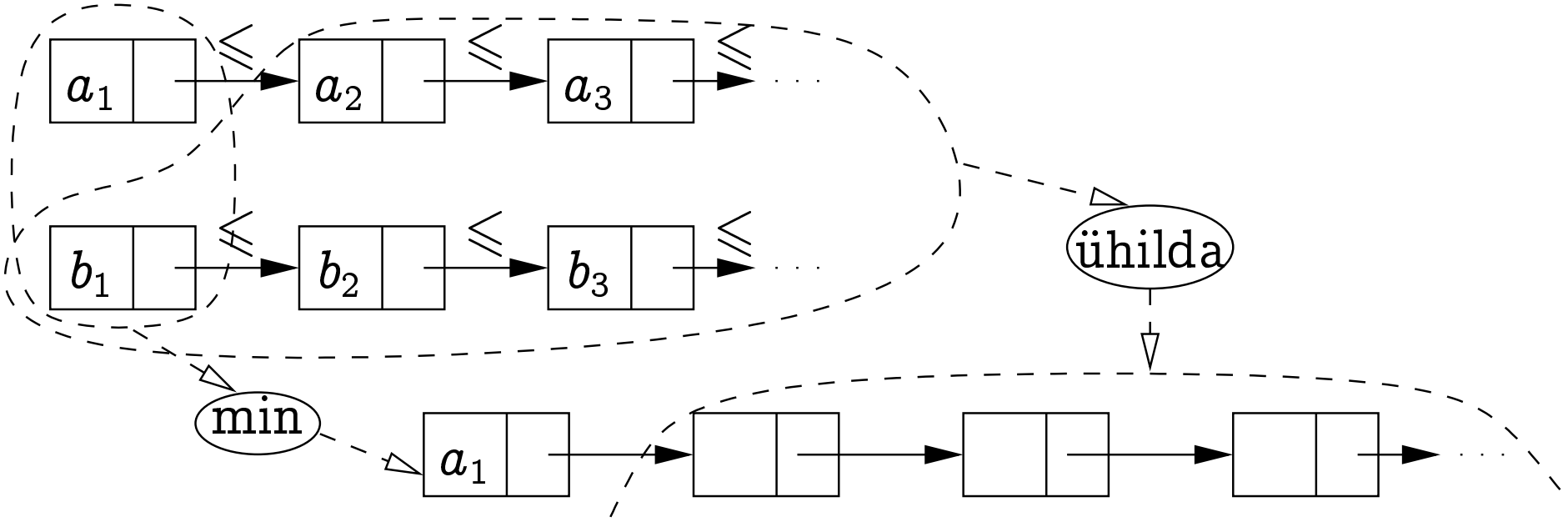
Protseduur ühenda_puud(p, q) teeb kahest i-ndat järku puust $(i+1)$-st järku puu. (Eeldus: $p.võti \leqslant q.võti$)

```
1  p.ülem := q
2  p.kolleeg := q.alluv
3  q.alluv := p
4  q.järk := q.järk + 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

```
 1  if K_1 = NIL then
 2      return K_2
 3  else if K_2 = NIL then
 4      return K_1
 5  else if K_1.järk < K_2.järk then
 6      K_1.kolleeg := ühilda_binomiaalkuhjad(K_1.kolleeg, K_2)
 7      return K_1
 8  else
 9      K_2.kolleeg := ühilda_binomiaalkuhjad(K_1, K_2.kolleeg)
10      return K_2
```

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

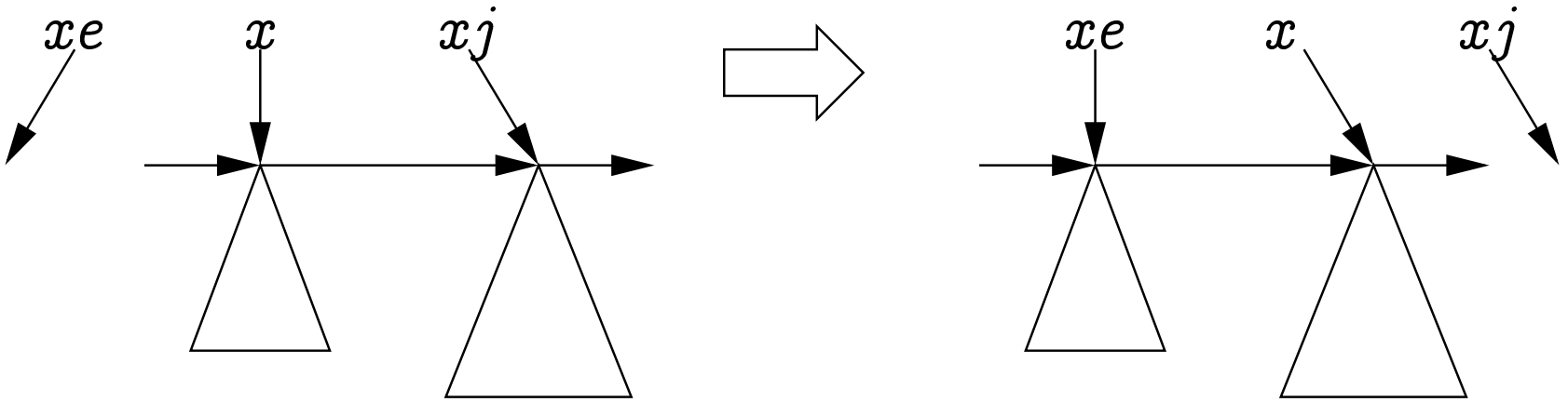
ühenda(K_1, K_2) on

```
1   K := ühilda_binomiaalkuhjad(K1, K2)
2   x := K; xe := NIL; xj := x.kolleeg
3   while xj ≠ NIL do
    ...
```

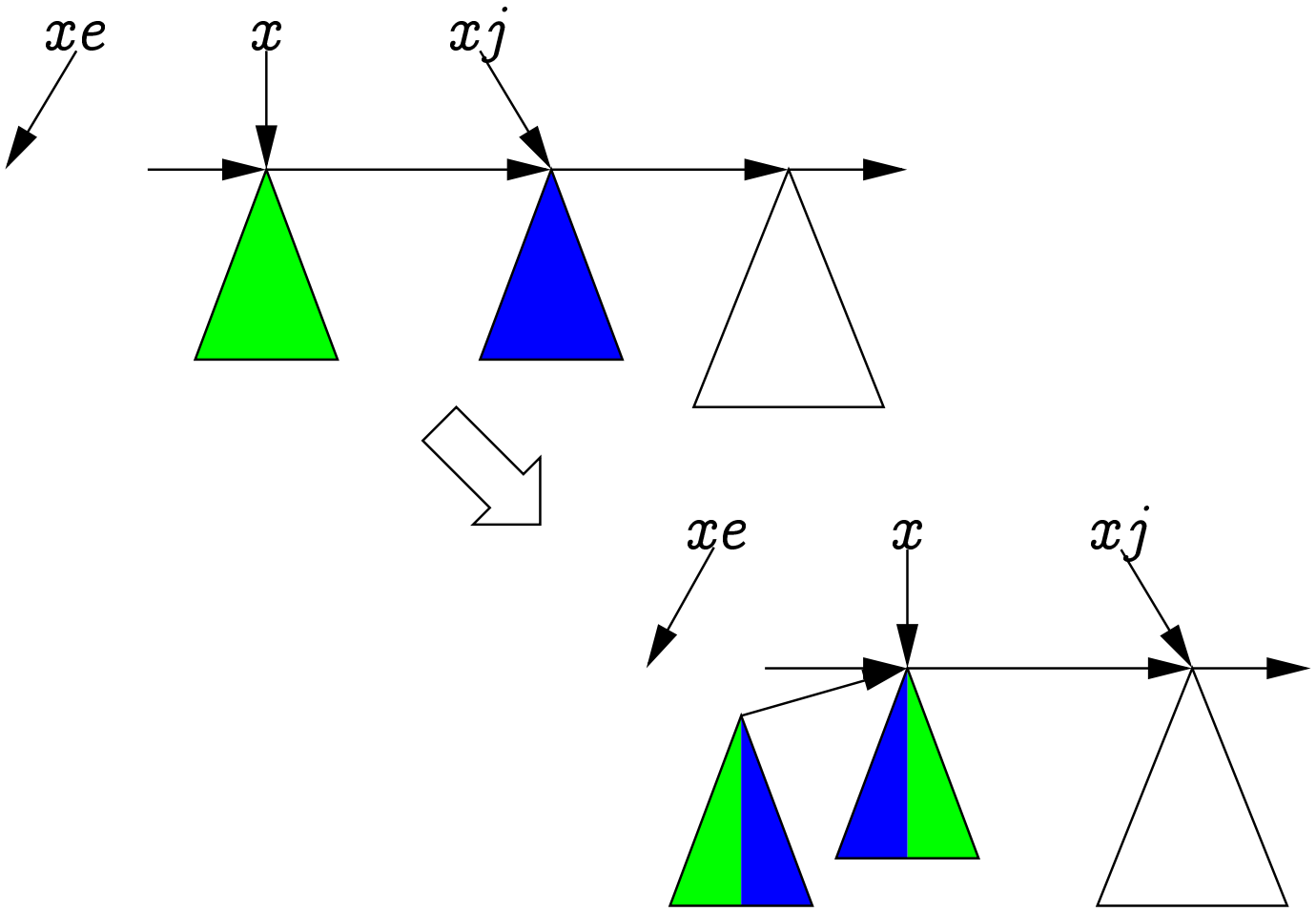
Tsükli alguses x viitab mingile binomiaalpuule, mille järk on suurem kui sellele puule eelneva puu järk. xe viitab eelmisele ja xj järgmisele puule.

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.

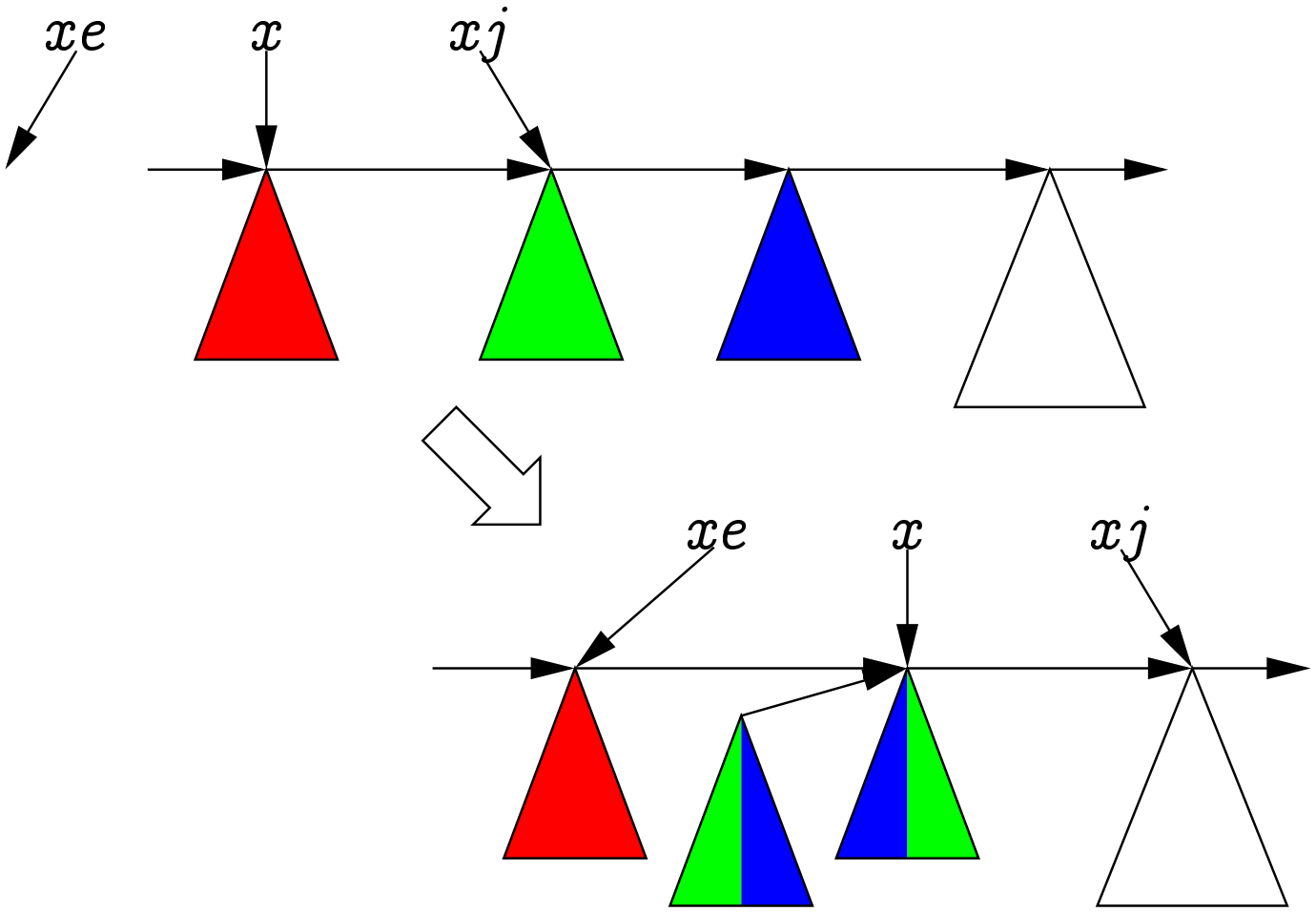


Kui on 2 puud, siis ühendame nad.



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.

```
4     if x.järk ≠ xj.järk then
5         xe := x; x := xj; xj := xj.kolleeg
6         continue
7     if xj.kolleeg ≠ NIL and xj.järk = xj.kolleeg.järk then
8         xe := x; x := xj; xj := xj.kolleeg
9     if x.võti < xj.võti then
10        ühenda_puud(x, xj)
11        (xe = NIL ? K : xe.kolleeg) := xj
12        x := xj
13    else
14        x.kolleeg := xj.kolleeg
15        ühenda_puud(xj, x)
16    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
 1   R := K.Andmed; q := K; qq := NIL
 2   p := K.kolleeg; pp := K
 3   while p ≠ NIL do
 4       if p.võti > R.võti then
 5           R := p.Andmed; q := p; qq := pp
 6       pp := p; p := p.kolleeg
 7   (qq = NIL ? K : qq.kolleeg) := q.kolleeg
 8   r := q.alluv; re := NIL
 9   while r ≠ NIL do
10       rj := r.kolleeg
11       r.kolleeg := re
12       re := r; r := rj
13   free(q)
14   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

```
1  if q.ülem = NIL or q.võti ⩽ q.ülem.võti then return
2  q.Andmed :=: q.ülem.Andmed
3  viia_üles(q.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 $q.võti := a$
2 viia_üles(q)

keerukusega $\Theta(\log n)$

viia_alla(q) on

```
1  if q.alluv = NIL then return
2  m := q.alluv; p := m.kolleeg
3  while p ≠ NIL do
4      if p.võti > m.võti then
5          m := p
6      p := p.kolleeg
7  if q.võti ⩾ m.võti then return
8  q.Andmed :=: m.Andmed
9  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

1 $L := \mathbf{new}(\text{kirjetüüp})$
2 $L.\text{Andmed} := R$; $L.alluv := \text{NIL}$
3 $L.kolleeg := \text{NIL}$; $L.ülem := \text{NIL}$
4 $L.järk := 0$
5 **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

1 $q.võti := \infty$
2 viia_üles(q)
3 $(\_, K') :=$ 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

```
1  R := q.Andmed
2  K := kustuta(K, q)
3  R.võti := a
4  return lisa(K, R)
```

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