Biltré

Bergur Snorrason

13. febrúar 2023

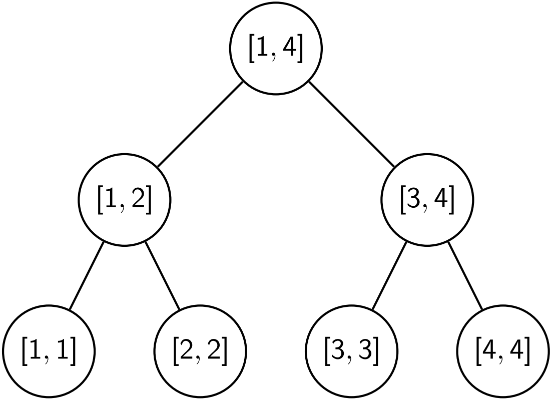
Dæmi

- Gefinn er listi með n tölum.
- Næst koma q fyrirspurnir, þar sem hver er af einni af tveimur gerðum:
  - Bættu k við i-tu töluna.
  - Reiknaðu summu allra talna á bilinu $[i, j]$.
- Einföld útfærlsa á þessum fyrirspurnum gefur okkur $\mathcal{O}(1)$ fyrir þá fyrri og $\mathcal{O}(n)$ fyrir þá seinni.
- Þar sem allar (eða langflestar) fyrirspurnir gætu verið af seinni gerðin yrði lausnin í heildina $\mathcal{O}(qn)$.
- Það er þó hægt að leysa þetta dæmi hraðar.
- Algengt er að nota til þess *biltré* (e. *segment tree*).

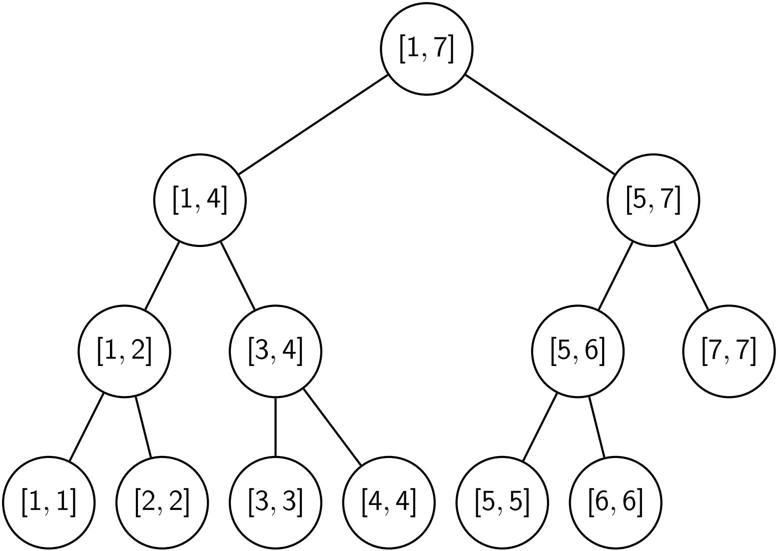
Biltré

- Biltré er tvíundartré sem geymir svör við vissum fyrirspurnum af seinni gerðinni.
- Rótin geymir svar við fyrirspurninni `1 n` og ef hnútur geymir svarið við `i j` þá geyma börn hans svör við `i m` og `(m + 1) j`, þar sem m er miðja heiltölubilsins $[i, j]$.
- Þeir hnútar sem geyma svar við fyrirspurnum af gerðinni `i i` eru lauf trésins.
- Takið eftir að laufin geyma þá gildin í listanum og aðrir hnútar geyma summu barna sinna.
- Þegar við útfærum tréð geymum við það eins og hrúgu.

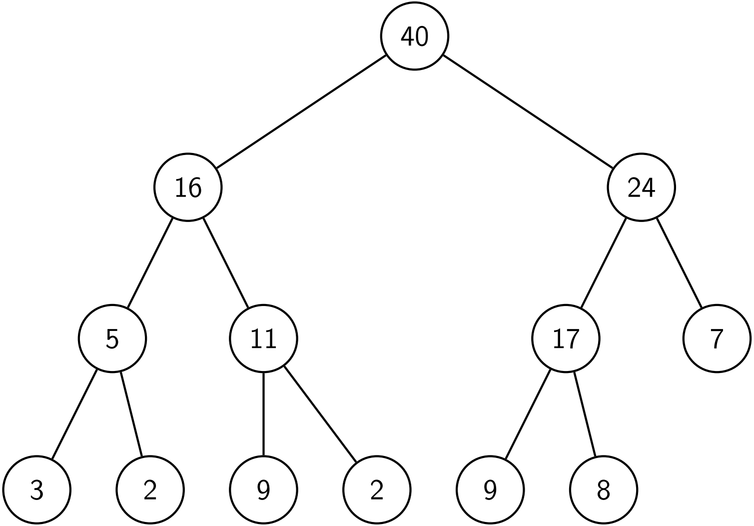
Mynd af biltré, $n = 4$



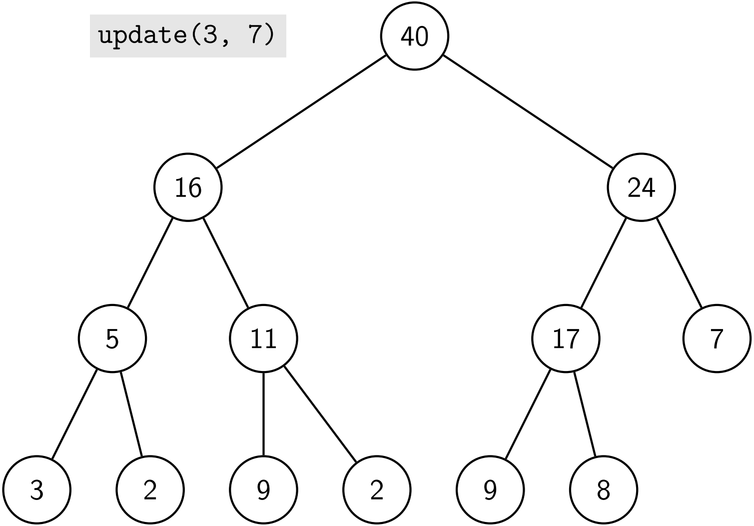
Mynd af biltré, $n = 7$



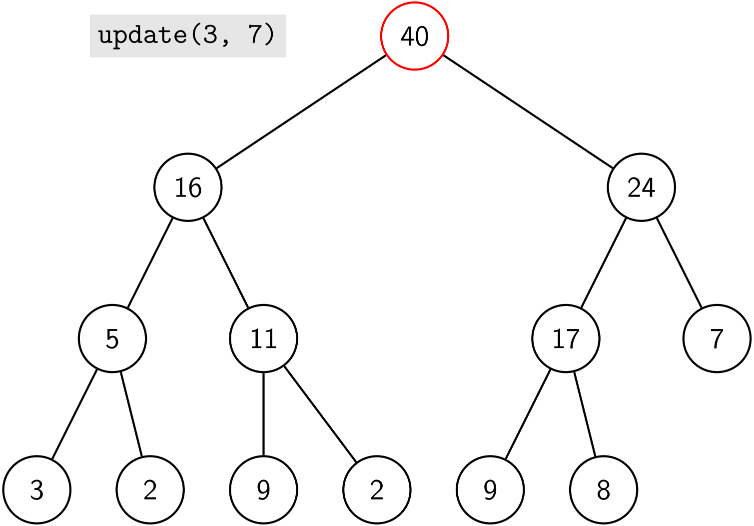
- Gerum ráð fyrir að við höfum biltré eins og lýst er að ofan og látum H tákna hæð trésins.
- Hvernig getum við leyst fyrirspurnirnar á glærunni á undan, og hver er tímaflækjan?
- Fyrri fyrirspurnin er einföld.
- Ef við eigum að bæta k við i-ta stakið finnum við fyrst laufið sem svarar til fyrirspurnar `i i`, bætum k við gildið þar og förum svo upp í rót í gegnum foreldrin og uppfærum á leiðinni gildin í þeim hnútum sem við lendum í.
- Þar sem við heimsækjum bara þá hnúta sem eru á veginum frá rót til laufs (mest H hnútar) er tímaflækjan á fyrri fyrirspurninni $\mathcal{O}(H)$.


40
16
24
5
11
17
7
3
2
9
2
9
8

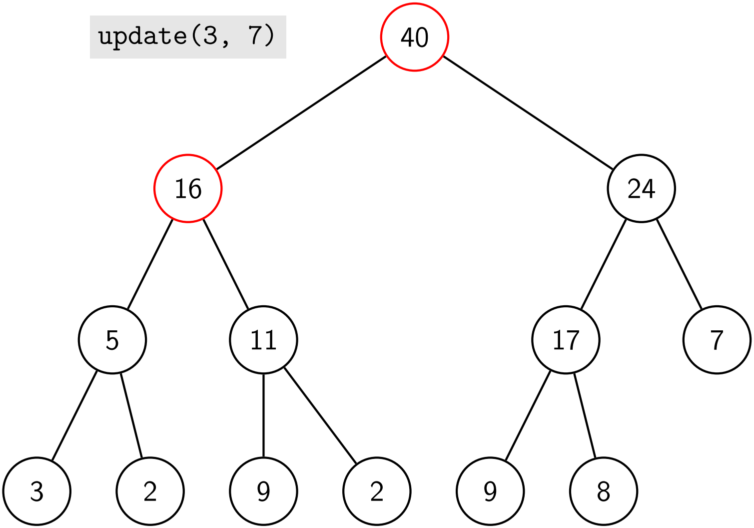
`update(3, 7)`



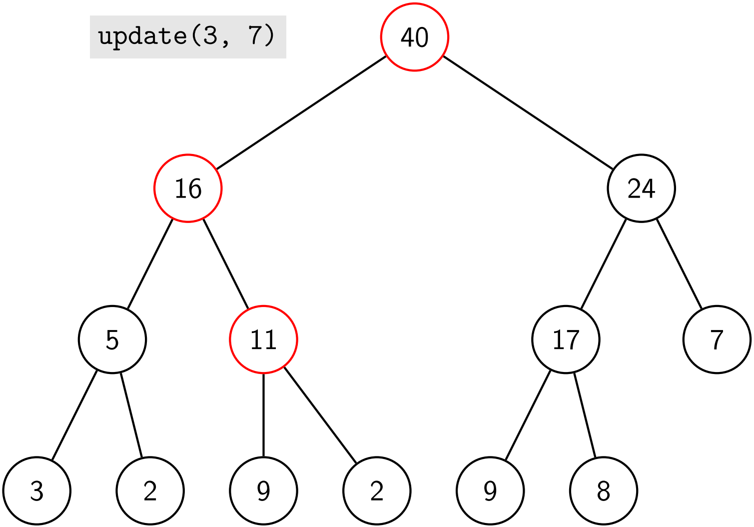
`update(3, 7)`



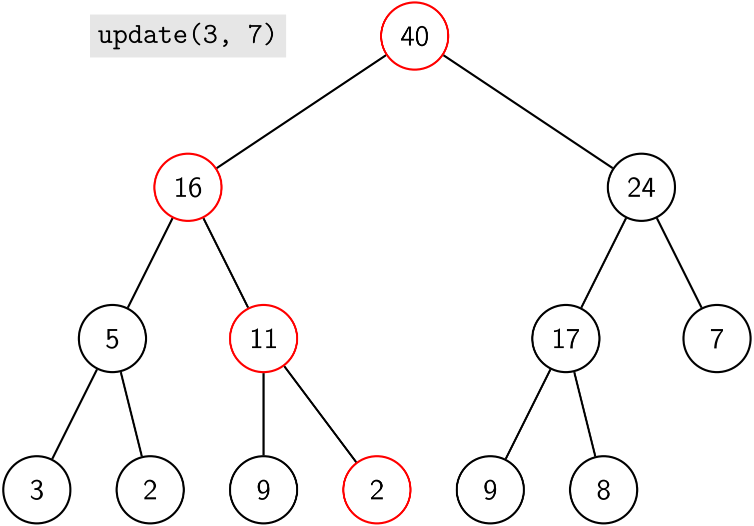
`update(3, 7)`



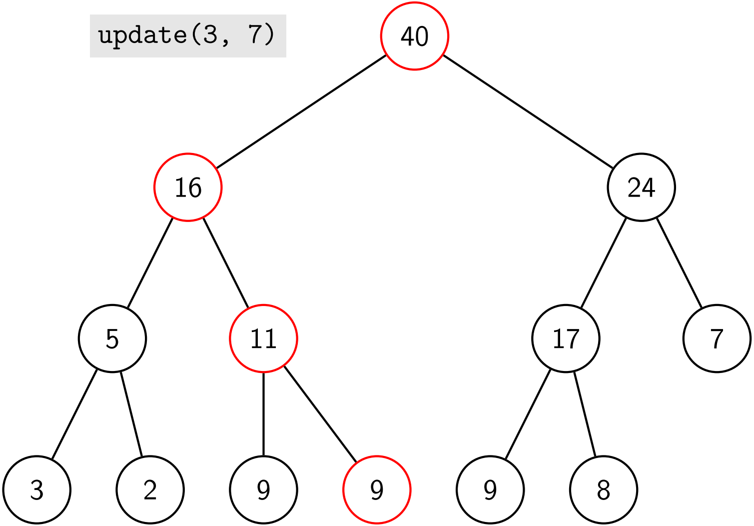
`update(3, 7)`



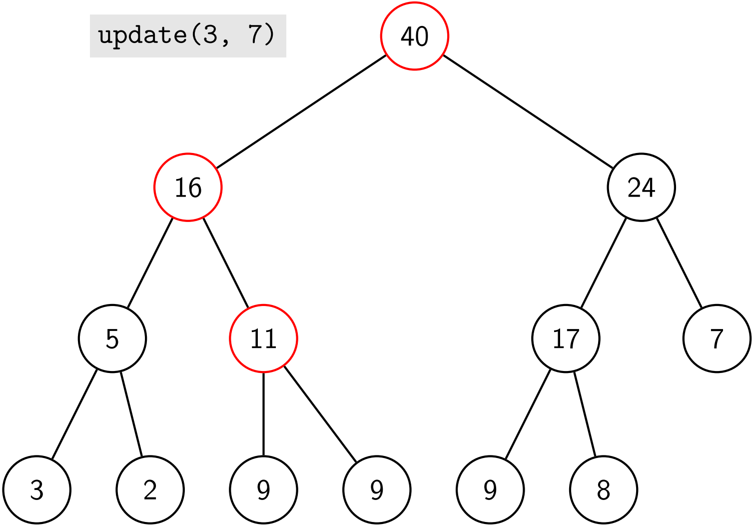
`update(3, 7)`



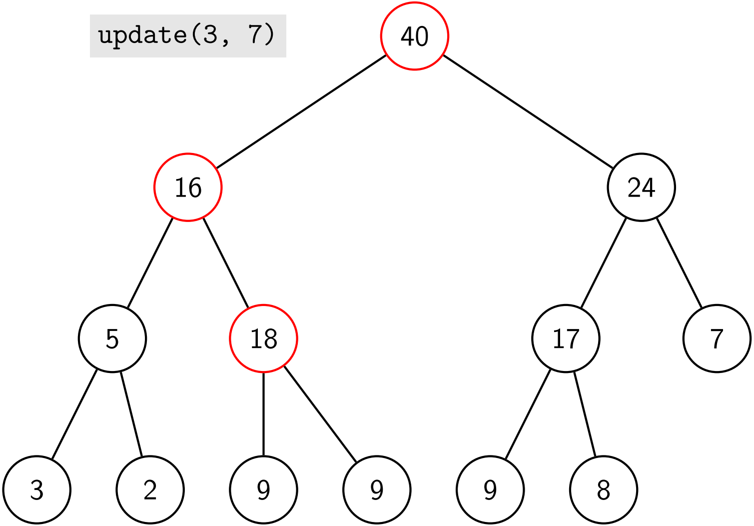
`update(3, 7)`



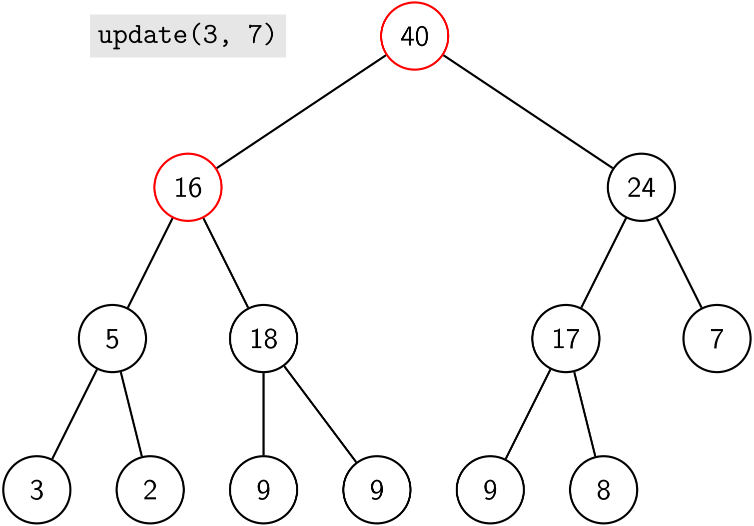
`update(3, 7)`



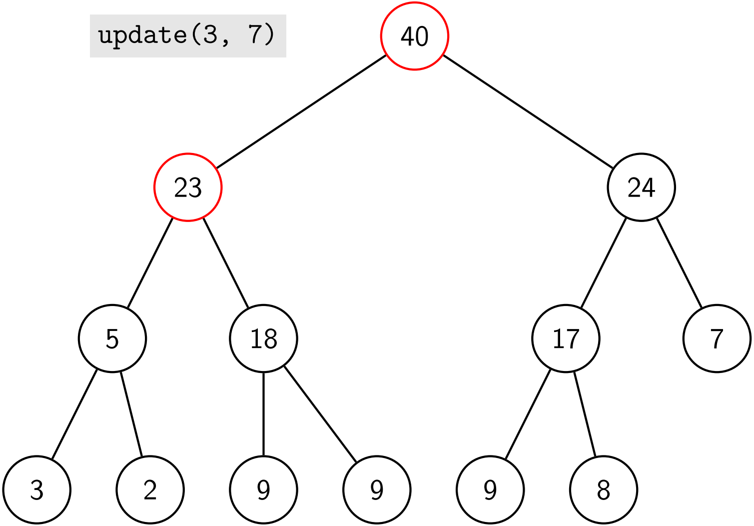
`update(3, 7)`



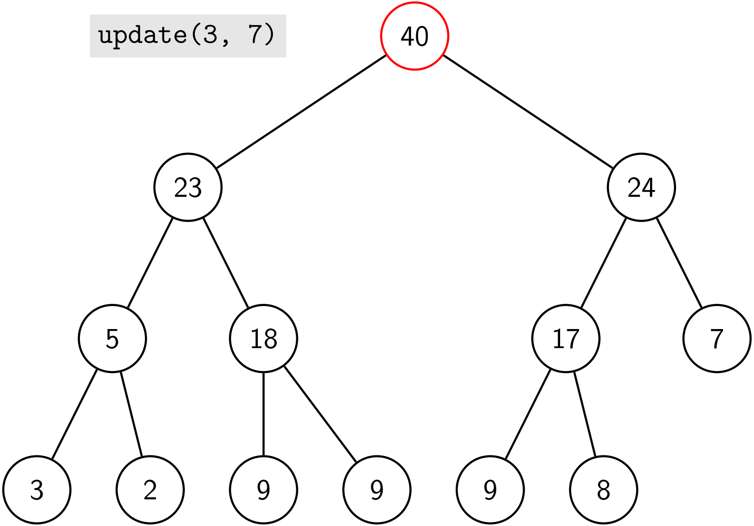
```
update(3, 7)
```



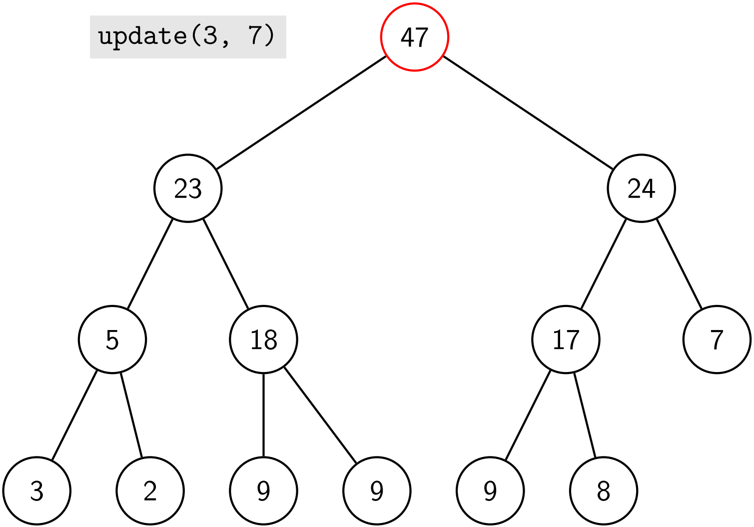
`update(3, 7)`



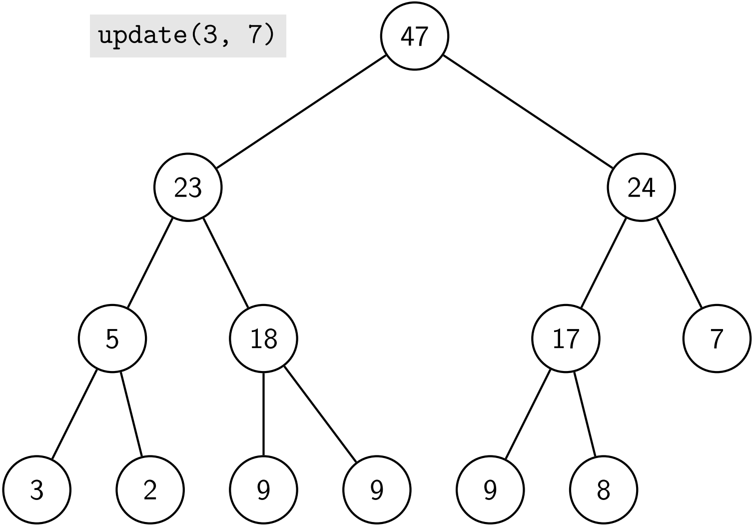
```
update(3, 7)
```



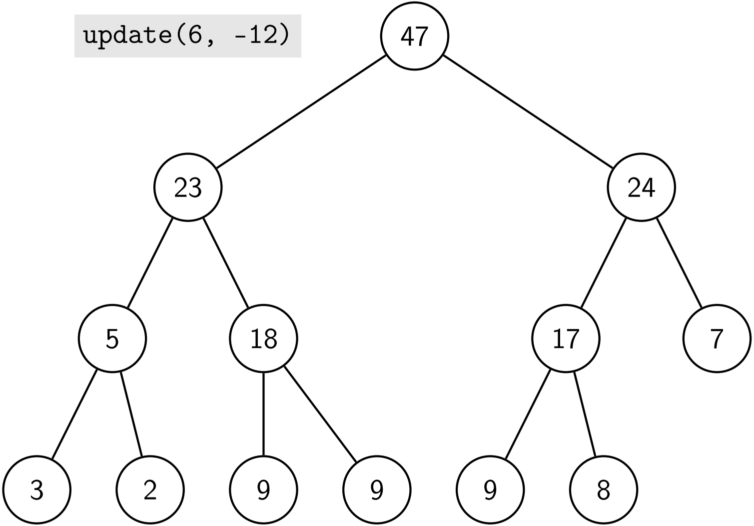
`update(3, 7)`



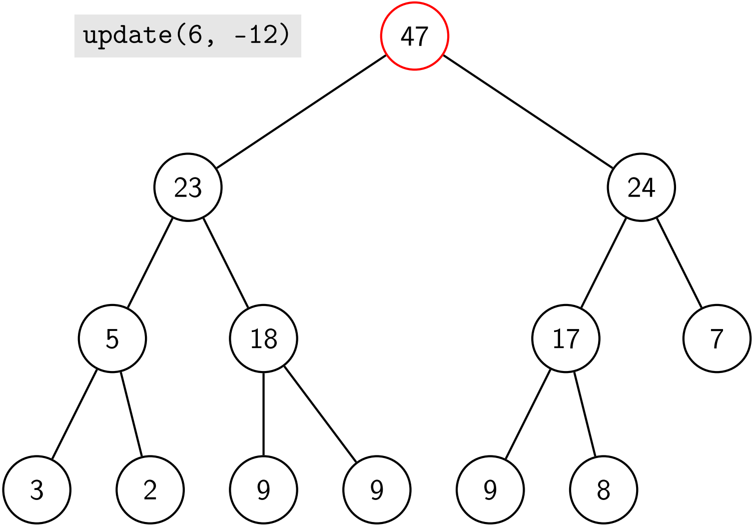
`update(3, 7)`



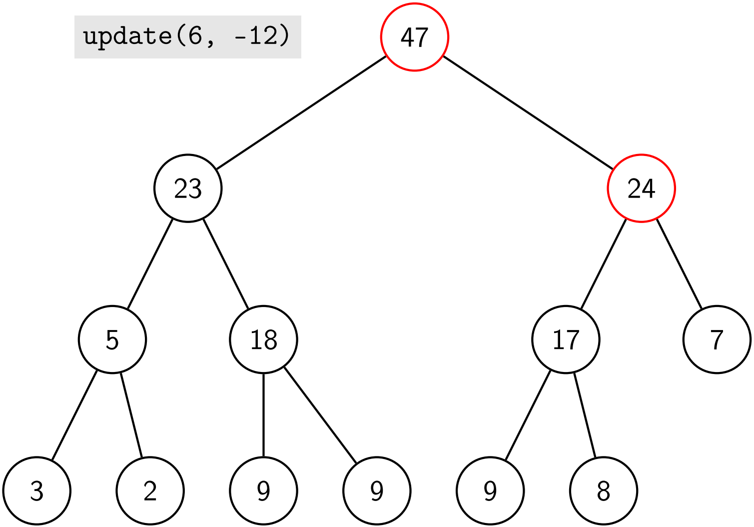
`update(6, -12)`



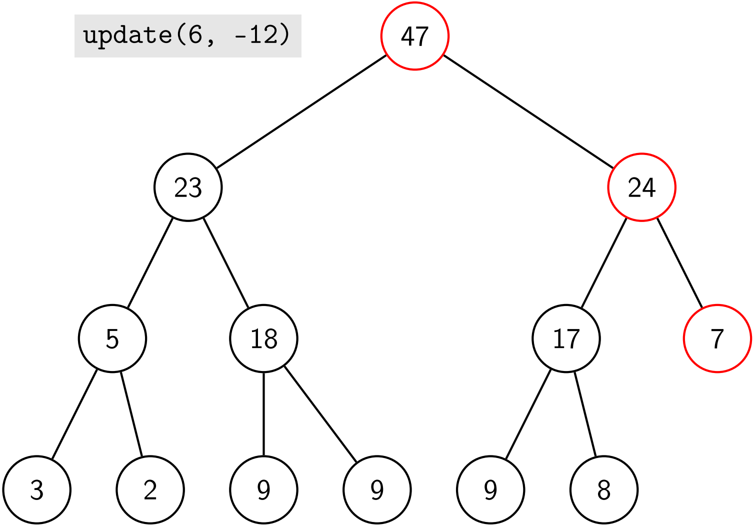
`update(6, -12)`



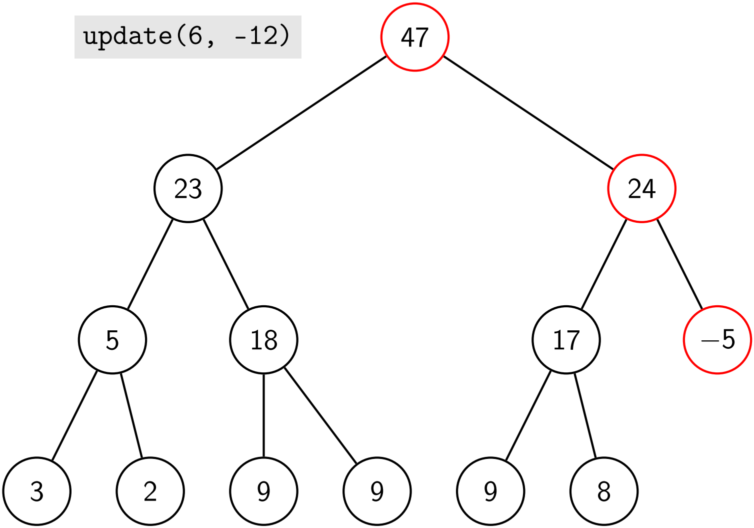
`update(6, -12)`



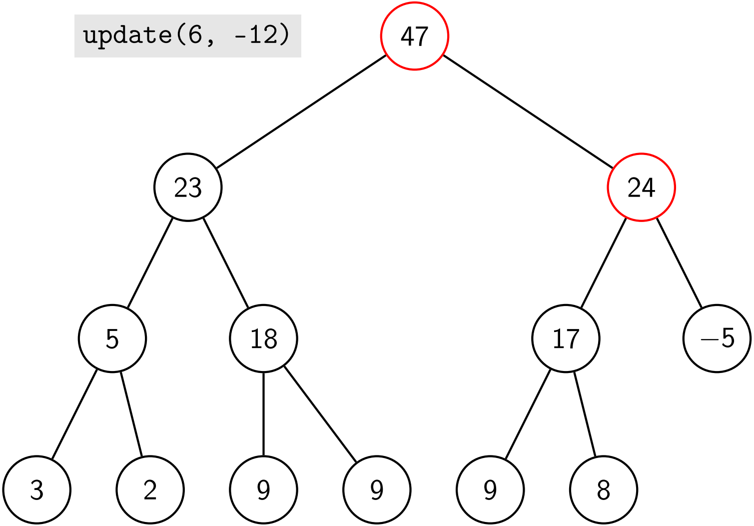
`update(6, -12)`



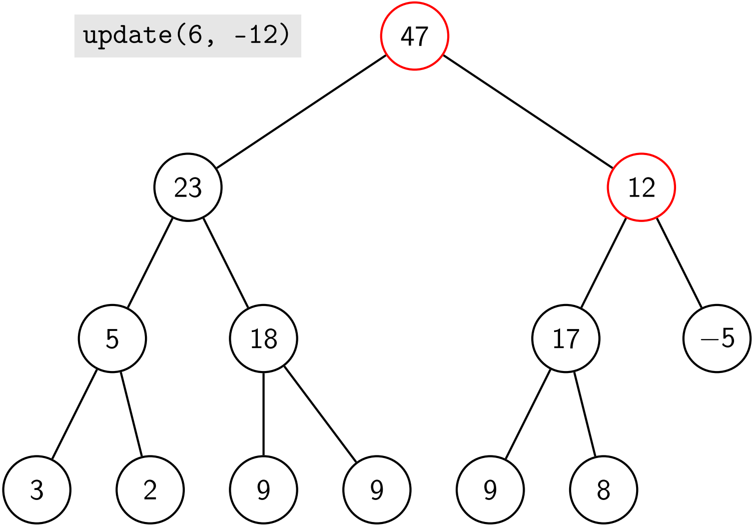
`update(6, -12)`



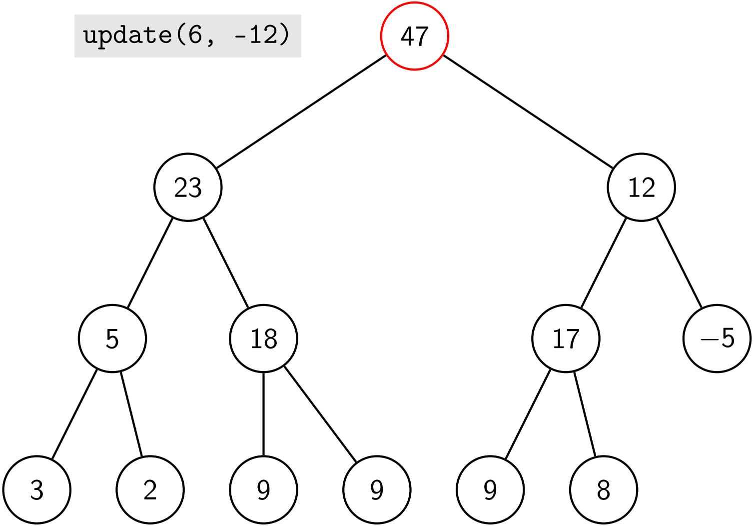
`update(6, -12)`



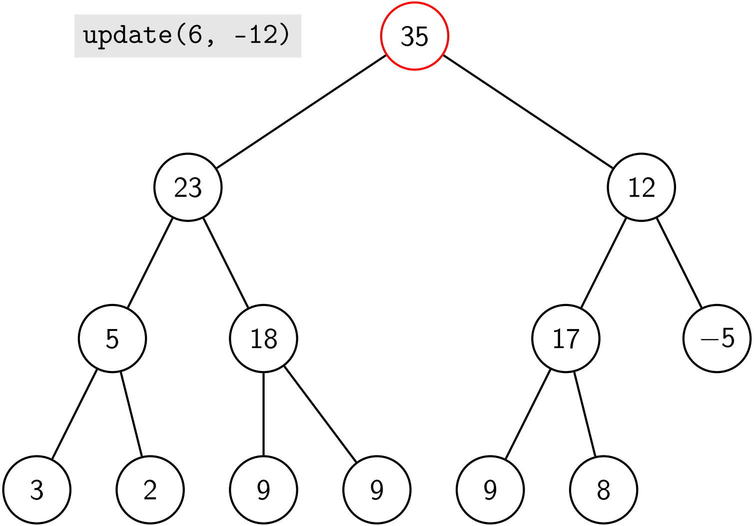
`update(6, -12)`



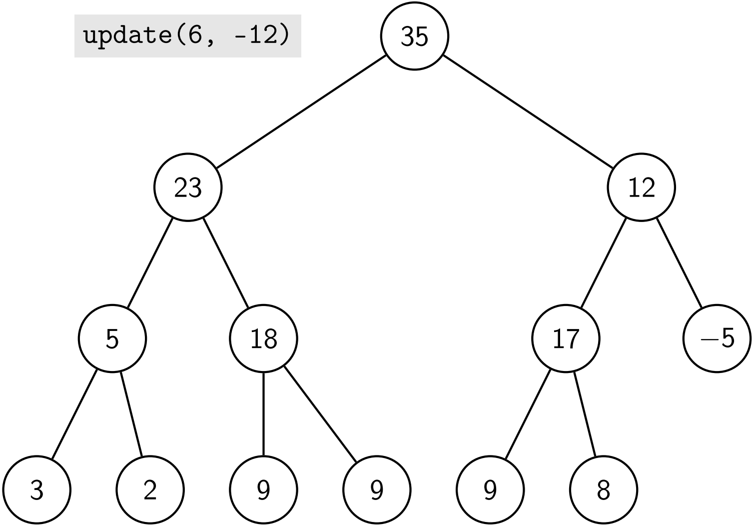
`update(6, -12)`

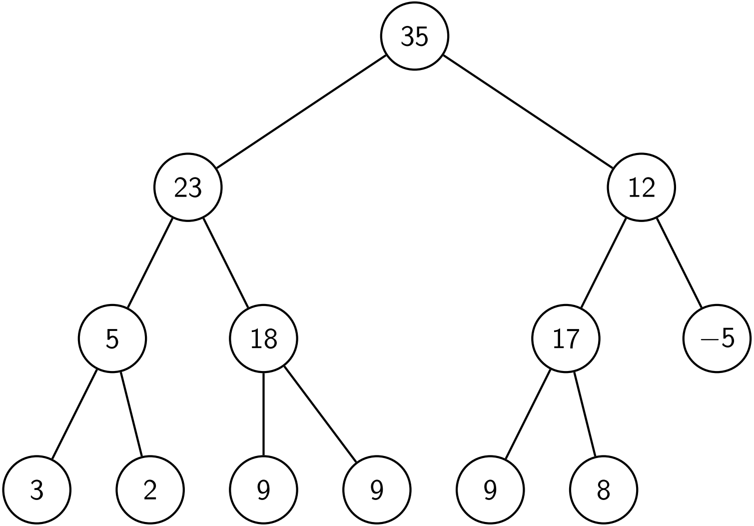


`update(6, -12)`



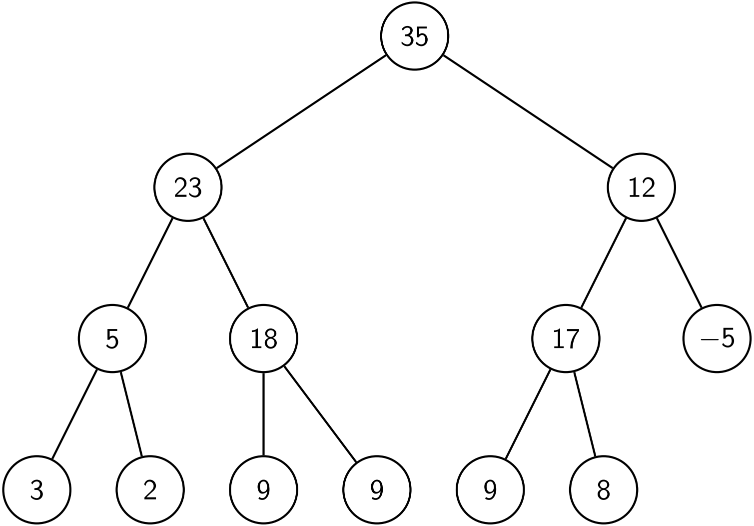
`update(6, -12)`



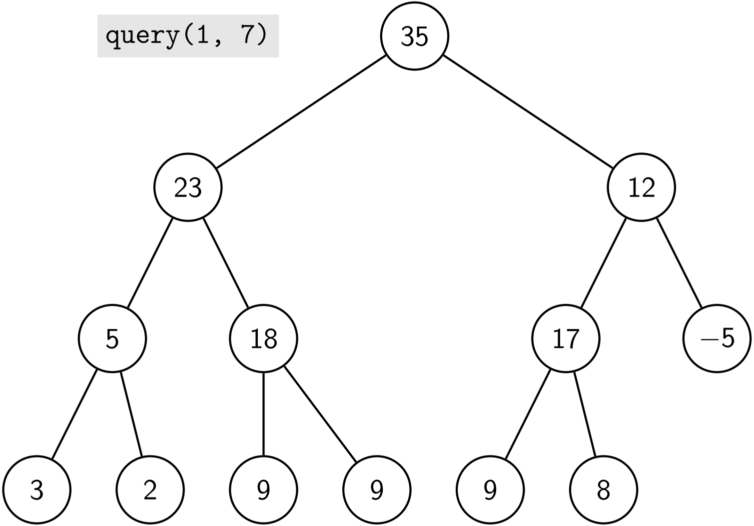

35
23
12
5
18
17
−5
3
2
9
9
9
8

```
void urec(int i, int j, int x, int y, int e)
{
    if (i == j) p[e] += y;
    else
    {
        int m = (i + j)/2;
        if (x <= m) urec(i, m, x, y, LEFT(e));
        else urec(m + 1, j, x, y, RIGHT(e));
        p[e] = p[LEFT(e)] + p[RIGHT(e)];
    }
}
void update(int x, int y)
{
    return urec(0, p[0] - 1, x, y, 1);
}
```

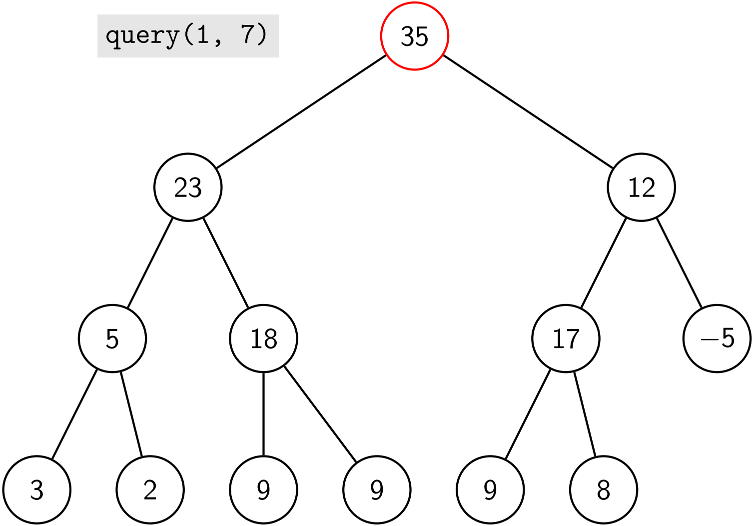
- Seinni fyrirspurnin er ögn flóknari.
- Auðveldast er að ímynda sér að við förum niður tréð og leitum að hvorum endapunktinum fyrir sig.
- Á leiðinni upp getum við svo pússlað saman svarinu, eftir því hvort við erum að skoða hægri eða vinstri endapunktinn.
- Til dæmis, ef við erum að leita að hægri endapunkti x og komum upp í bil $[i, j]$ þá bætum við gildinu í hnút $[i, m]$ við það sem við höfum reiknað hingað til ef $x \in [m + 1, j]$, en annars bætum við engu við (því x er hægri endapunkturinn).
- Við göngum svona upp þar til við lendum í bili sem inniheldur hinn endapunktinn.
- Með sömu rökum og áðan er tímaflækjan $\mathcal{O}(H)$.


35
23
12
5
18
17
−5
3
2
9
9
9
8

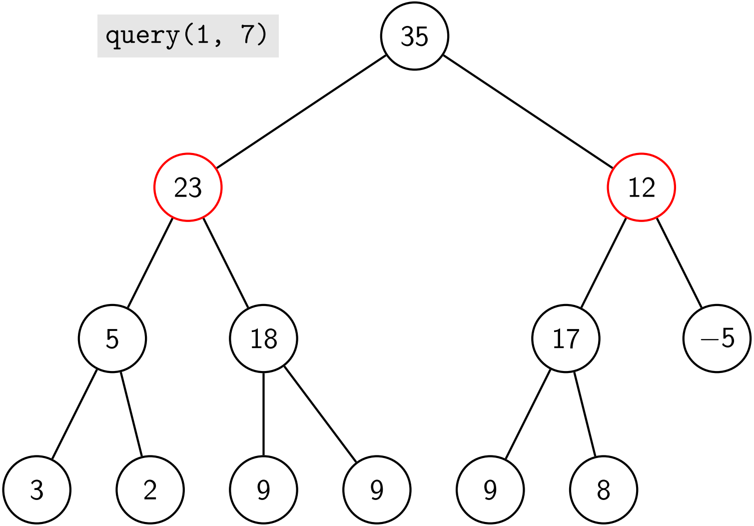
`query(1, 7)`



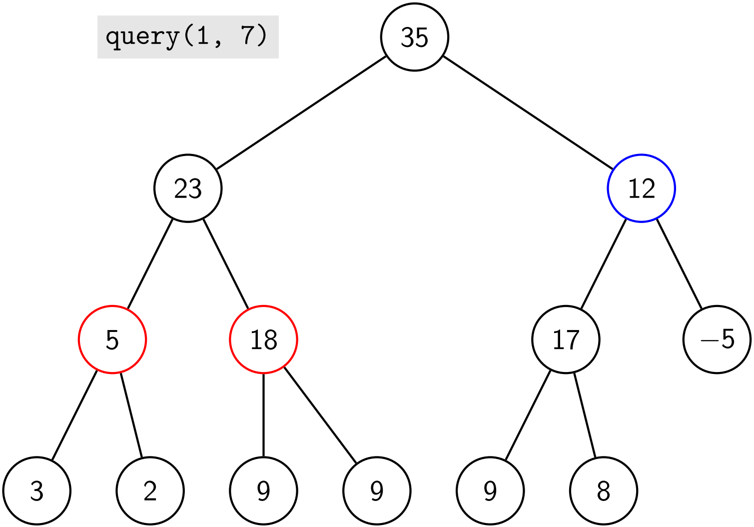
`query(1, 7)`



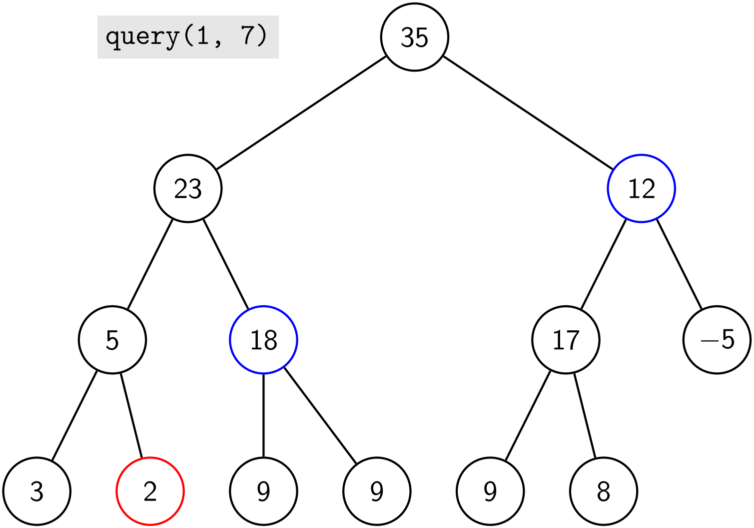
`query(1, 7)`



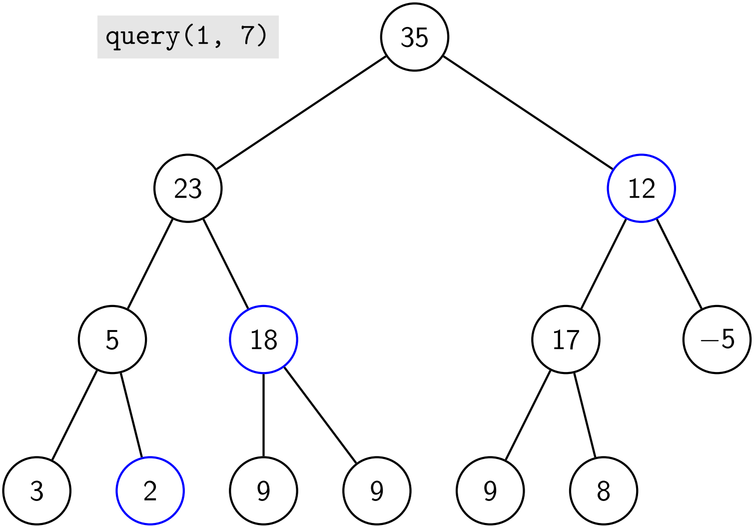
`query(1, 7)`



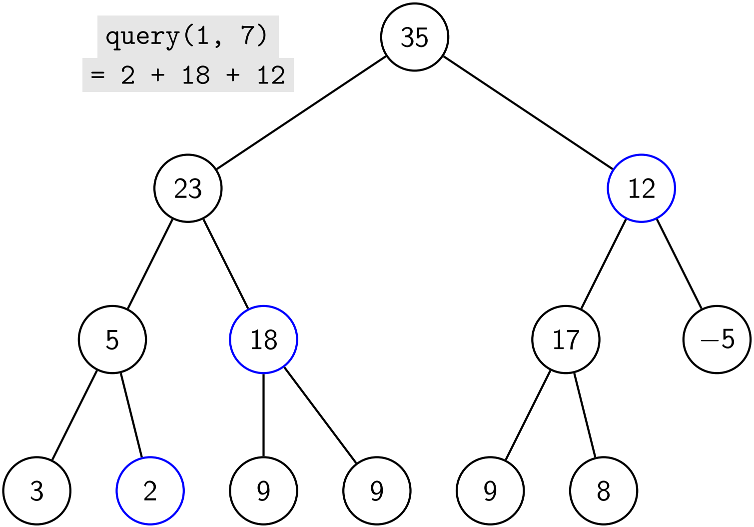
`query(1, 7)`

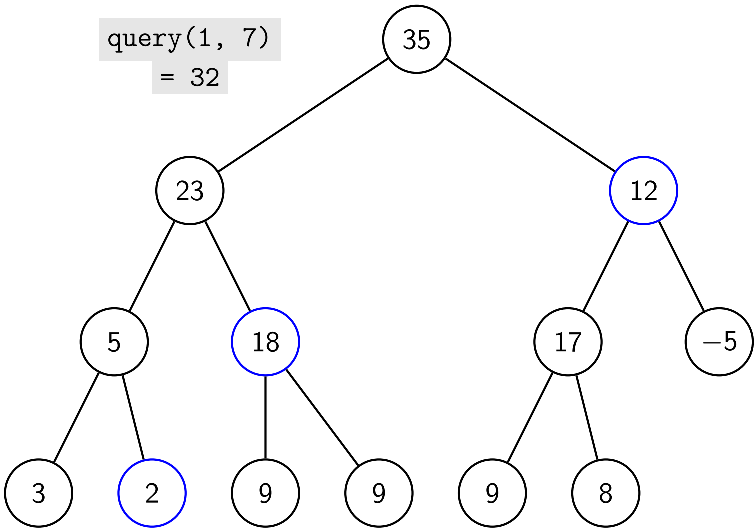


`query(1, 7)`

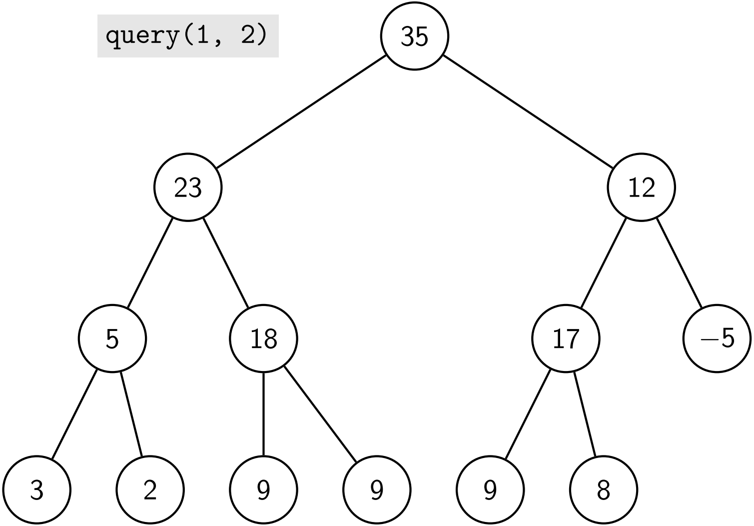


```
query(1, 7)
= 2 + 18 + 12
```

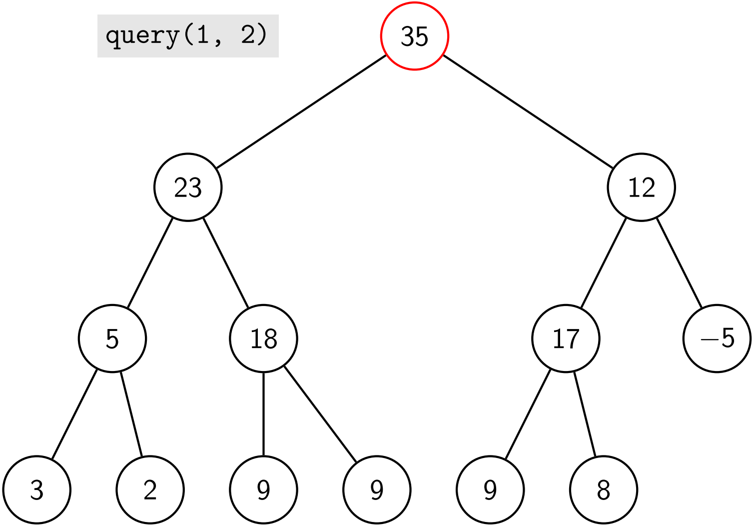



query(1, 7)
= 32
35
23
12
5
18
17
−5
3
2
9
9
9
8

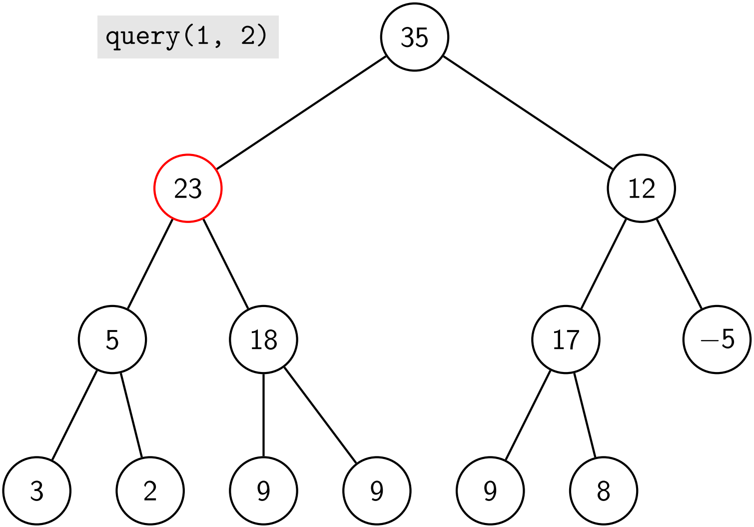
`query(1, 2)`



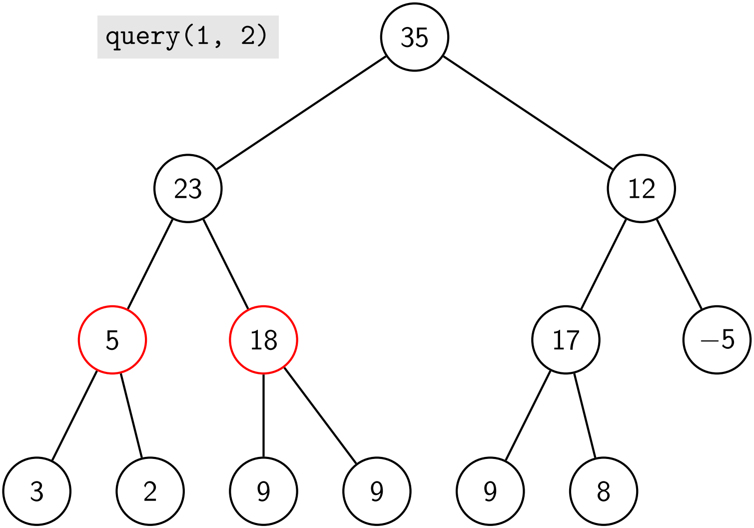
`query(1, 2)`



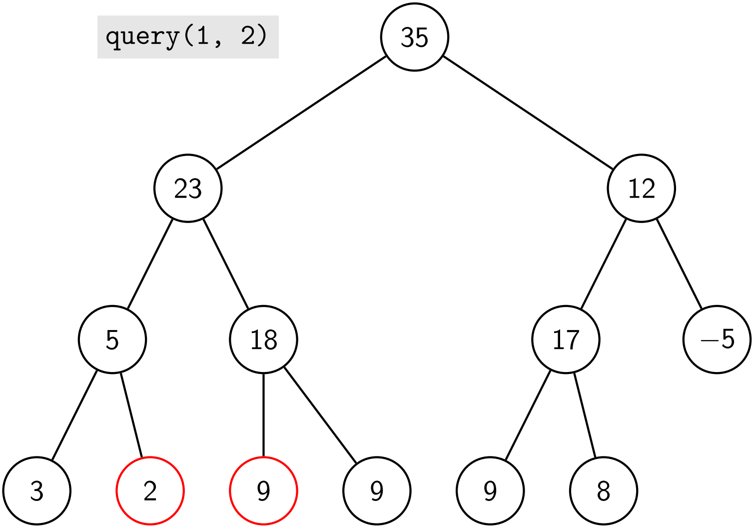
`query(1, 2)`



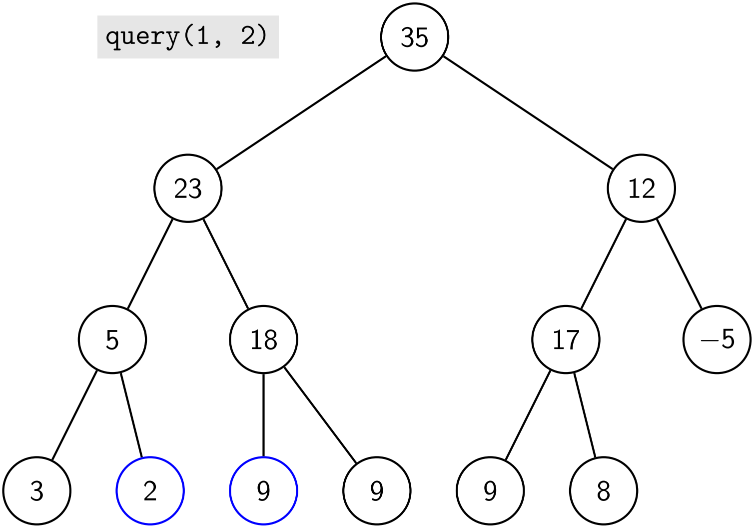
`query(1, 2)`



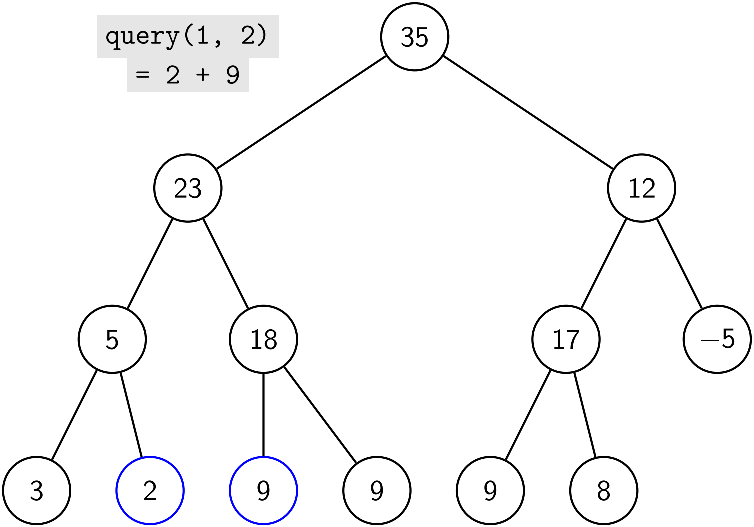
`query(1, 2)`



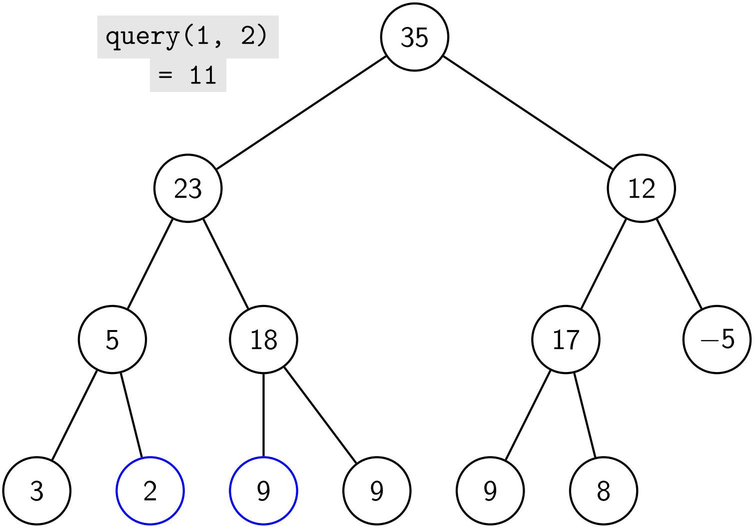
`query(1, 2)`

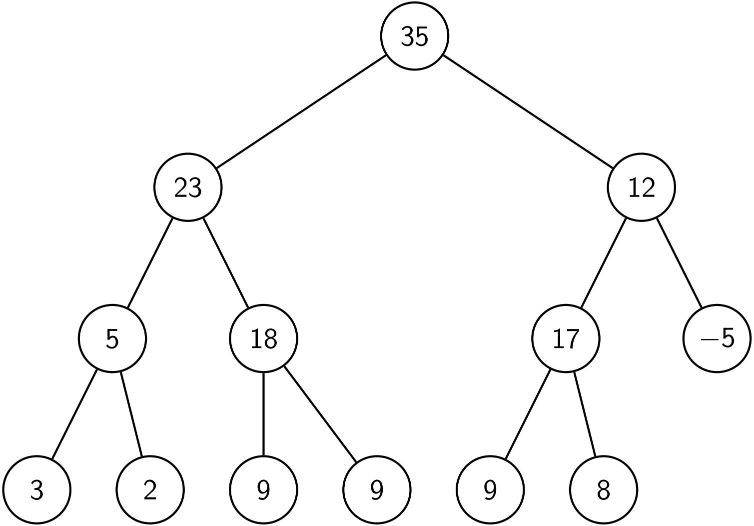


```
query(1, 2)
 = 2 + 9
```



```
query(1, 2)
= 11
```




35
23
12
5
18
17
−5
3
2
9
9
9
8

Biltré í `C`

```
int qrec(int i, int j, int x, int y, int e)
{
    if (x == i && y == j) return p[e];
    int m = (i + j)/2;
    if (y <= m) return qrec(i, m, x, y, LEFT(e));
    if (x > m) return qrec(m + 1, j, x, y, RIGHT(e));
    return qrec(i, m, x, m, LEFT(e)) + qrec(m + 1, j, m + 1, y, RIGHT(e));
}
int query(int x, int y)
{
    return qrec(0, p[0] - 1, x, y, 1);
}
```

Tímaflækja biltrjáa

- Þar sem lengd hvers bils sem hnútur svarar til helmingast þegar farið er niður tréð er $\mathcal{O}(H) = \mathcal{O}(\log n)$.
- Við erum því komin með lausn á upprunalega dæminu sem er $\mathcal{O}(q \cdot \log n)$.
- Þetta væri nógu hratt ef, til dæmis, $n = q = 10^6$.
- Tökum annað dæmi.

Annað dæmi

- Fyrsta lína inntaksins inniheldur tvær jákvæðar heiltölur, n og m, minni en 10^5.
- Næsta lína inniheldur n heiltölur, á milli -10^9 og 10^9.
- Næstu m línur innihalda fyrirspurnir, af tveimur gerðum.
- Fyrri gerðin hefst á `1` og inniheldur svo tvær tölur, x og y. Hér á að setja x-tu töluna sem y.
- Seinni gerðin hefst á `2` og inniheldur svo tvær tölu, x og y. Hér á að prenta út stærstu töluna á hlutbilinu $[x, y]$ í talnalistanum.
- Hvernig leysum við þetta?

- Við getum leyst þetta með biltrjám.
- Í stað þess að láta hnúta (sem eru ekki lauf) geyma summu barna sinna, þá geyma þeir stærra stak barna sinna.

Lausn

```
int qrec(int i, int j, int x, int y, int e)
{
    if (x == i && y == j) return p[e];
    int m = (i + j)/2;
    if (y <= m) return qrec(i, m, x, y, LEFT(e));
    if (x > m) return qrec(m + 1, j, x, y, RIGHT(e));
    return max(qrec(i, m, x, m, LEFT(e)), qrec(m + 1, j, m + 1, y, RIGHT(e)));
}
int query(int x, int y)
{
    return qrec(0, p[0] - 1, x, y, 1);
}

void urec(int i, int j, int x, int y, int e)
{
    if (i == j) p[e] = y;
    else
    {
        int m = (i + j)/2;
        if (x <= m) urec(i, m, x, y, LEFT(e));
        else urec(m + 1, j, x, y, RIGHT(e));
        p[e] = max(p[LEFT(e)], p[RIGHT(e)]);
    }
}
void update(int x, int y)
{
    return urec(0, p[0] - 1, x, y, 1);
}
```

- Leysa má ýmis dæmi af þessari gerð, með biltrjám.
- Þessi dæmi eru yfirleitt kölluð *punkt-uppfærslur, bil-fyrirspurnir* (e. *point-update, range-query*).
- Algengt er að sýna næst hvernig nota megi biltré til að leysa *bil-uppfærslur, punkt-fyrirspurnir* (e. *range-update, point-query*).
- Þetta er, í grófum dráttum, gert með því að snúa trjánum við.
- Við munum ekki skoða þetta.
- Við tökum frekar fyrir *lygn biltré*.
- Þau leyfa okkur að leysa *bil-uppfærslur, bil-fyrirspurnir* (e. *range-update, range-query*).

Lygn dreifing

- Sem beinagrind munum við nota biltrjáa útfærsluna sem við notuðum til að leysa fyrsta dæmið.
- Við munum nú láta fyrri fyrirspurnina, `i j k`, þýða “Bættu k við allar tölur á bilinu $[i, j]$”.
- Uppfærslan er framkvæmd á svipaðan hátt og fyrirspurnirnar eru.
- Við geymum í öðrum tréi þær uppfærslur sem við eigum eftir að framkvæma.
- Í hverri endurkvæmni (bæði uppfærslum og fyrirspurnum) dreifum við uppfærslunum í hnútnum niður á við.
- Þetta kallast *lygn dreifing* (e. *lazy propagation*), því við framkvæmum hana bara þegar nauðsyn krefur.
- Ef biltré hefur lygna dreifingu köllum við það *lygnt biltré* (e. *segment tree with lazy propagation*).

- Látum $i < j$ vera heiltölur þannig að bilið $[i, j]$ svara til hnúts í biltréi og m vera miðpunkt heiltölu bilsins $[i, j]$.
- Gerum ráð fyrir að við eigum eftir að framkvæma uppfærslu `i j k`.
- Næst þegar við köllum á `query_rec(i, j, ...)` eða `update_rec(i, j, ...)` þá munum við dreifa uppfærslunni `i j k`.
- Eftir dreifinguna munum við ekki eiga eftir uppfærslu á bilinu $[i, j]$, en við munum eiga eftir uppfærslurnar `i m k` og `(m + 1) j k`.
- Þegar við dreifum uppfærslunni `i i k` þá nægir að uppfæra tilheyrandi lauf í biltrénu.

- Áðan var sagt “laufin geyma þá gildin í listanum og aðrir hnútar geyma summu barna sinna”.
- Þetta gildir ekki fyrir lygn biltré.
- Hnútar lygna biltrjáa þurfa að geyma summu barna sinna, ásamt því að geyma þá summu sem fengist eftir allar óframkvæmdar uppfærslur afkomenda hans.
- Þegar við ferðumst í gegnum tréð til að finna hvert við eigum að setja uppfærsluna uppfærum við tréð jafn óðum.
- Til dæmis, ef við viljum framkvæma uppfærsluna `i j k` þá þurfum við að bæta $k \cdot (j - i + 1)$ við rót biltrésins, því rótin geymir summu allra stakana.

```
void prop(int x, int y, int e)
{
    p[e] += (y - x + 1)*o[e];
    if (x != y) o[LEFT(e)] += o[e], o[RIGHT(e)] += o[e];
    o[e] = 0;
}

int qrec(int i, int j, int x, int y, int e)
{
    prop(i, j, e);
    if (x == i && y == j) return p[e];
    int m = (i + j)/2;
    if (y <= m) return qrec(i, m, x, y, LEFT(e));
    else if (x > m) return qrec(m + 1, j, x, y, RIGHT(e));
    return qrec(i, m, x, m, LEFT(e)) + qrec(m + 1, j, m + 1, y, RIGHT(e));
}
int query(int x, int y)
{
    return qrec(0, p[0] - 1, x, y, 1);
}

void urec(int i, int j, int x, int y, int z, int e)
{
    prop(i, j, e);
    if (x == i && y == j) { o[e] = z; return; }
    int m = (i + j)/2;
    p[e] += (y - x + 1)*z;
    if (y <= m) urec(i, m, x, y, z, LEFT(e));
    else if (x > m) urec(m + 1, j, x, y, z, RIGHT(e));
    else urec(i, m, x, m, z, LEFT(e)), urec(m + 1, j, m + 1, y, z, RIGHT(e));
}
void update(int x, int y, int z)
{
    urec(0, p[0] - 1, x, y, z, 1);
}
```

- Nú hefur `query(...)` sömu tímaflækju og í hefðbundnum biltrjám, það er að segja $\mathcal{O}(\log n)$.
- Loks fæst (með sömu rökum og gefa tímaflækju `query(...)`) að `update(...)` er $\mathcal{O}(\log n)$.