Adatszerkezetek és algoritmusok

Horváth Géza

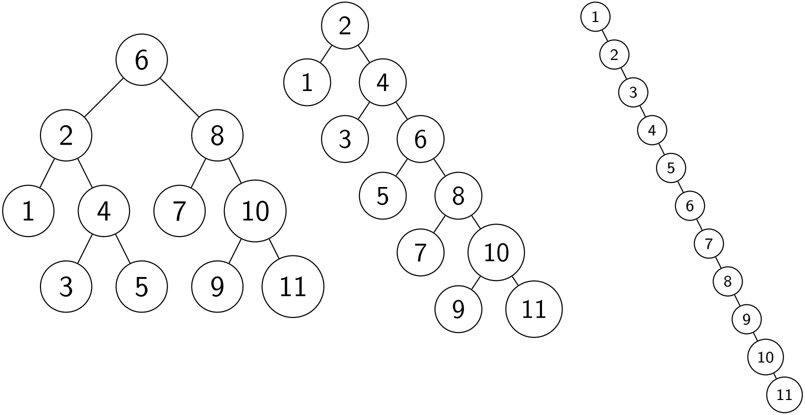
nyolcadik előadás

Előadások témái

1. Az algoritmusokkal kapcsolatos alapfogalmak bevezetése egyszerű példákon keresztül.
2. Az algoritmusok futási idejének aszimptotikus korlátai.
3. Az adatszerkezetekkel kapcsolatos alapfogalmak. A halmaz, a multihalmaz és a tömb adatszerkezet bemutatása.
4. Az adatszerkezetek folytonos és szétszórt reprezentációja. A verem, a sor és a lista.
5. Táblázatok, önátrendező táblázatok, hash függvények és hash táblák, ütközéskezelés.
6. Fák, bináris fák, bináris keresőfák, bejárás, keresés, beszúrás, törlés.
7. Kiegyensúlyozott bináris keresőfák: AVL fák.
8. Piros-fekete fák.
9. B-fák.
10. Gráfok, bejárás, legrövidebb út megkeresése.
11. Párhuzamos algoritmusok.
12. Eldönthetőség és bonyolultság, a P és az NP problémaosztályok.
13. Lineáris idejű rendezés. Összefoglalás.

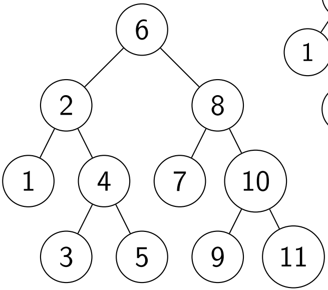
Keresőfák

Mennyi idő a kulcs megtalálása?

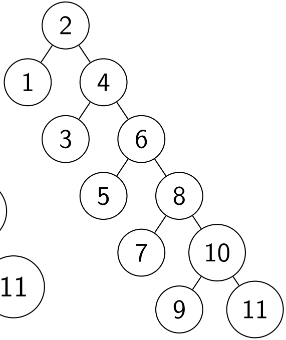


Keresőfák

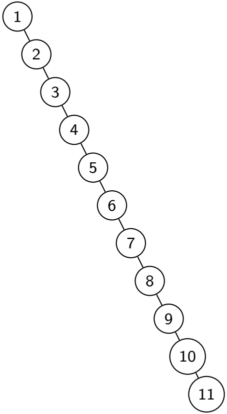
Mennyi idő a kulcs megtalálása?



Logaritmus!

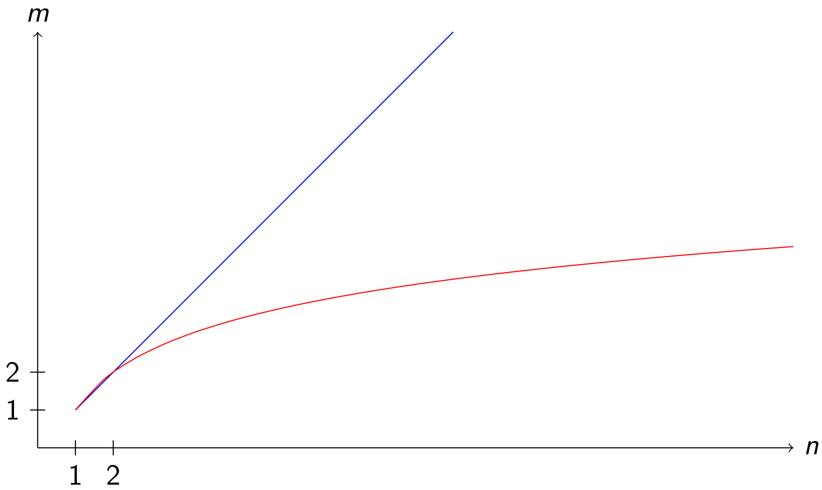


Lineáris.



Lineáris.

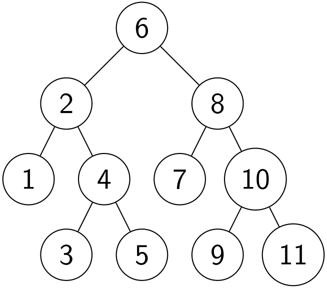
A lineáris és a logaritmus idejű keresés összehasonlítása



n – **input size**

m – **running time**

Search trees

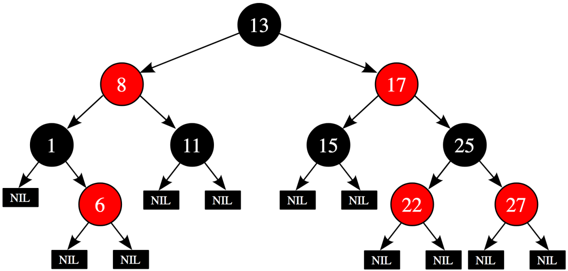


- TREE-SEARCH – $\mathcal{O}(log_2 n)$
- TREE-INSERT – $\mathcal{O}(log_2 n)$
- TREE-DELETE – $\mathcal{O}(log_2 n)$

Remek megoldás: kiegyensúlyozott bináris keresőfa.

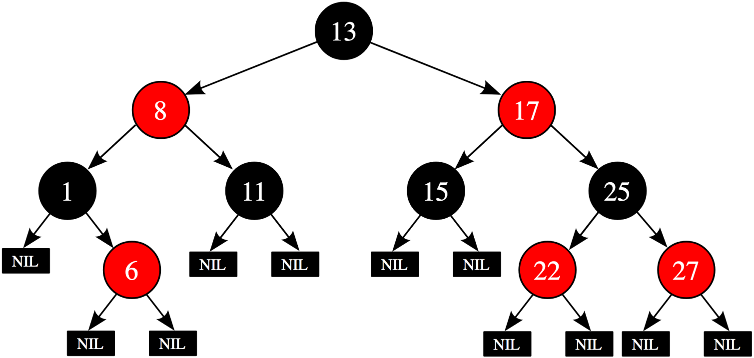
– **Van más megoldás is???**

A piros-fekete fa



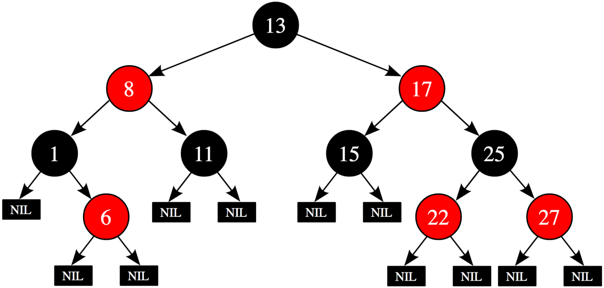
A piros-fekete fa 1972-ben került bemutatásra. A piros-fekete fa a bináris keresőfa minden egyes csúcsához hozzárendel egy plusz bit információt, a csúcs színét, ami lehet piros vagy fekete. A piros-fekete fa használatakor beszúrás és törlés után szükség lehet átszínezésre és elforgatásra, hogy visszaállítsuk a fa megfelelő alakját.

A piros-fekete fa



Minden csúcs esetén a következő adatokat tartjuk nyilván:
kulcs, szín, baloldali gyerek, jobboldali gyerek, szülő.
Amennyiben nincs szülő vagy gyerek, akkor az adott érték NIL.

Piros-fekete fa – tulajdonságok

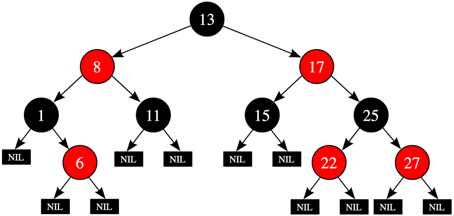


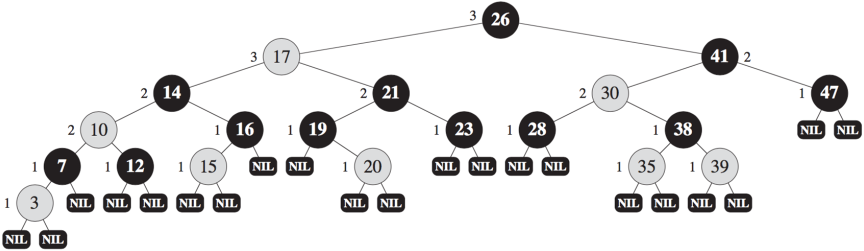
A piros-fekete fa tulajdonságai:

1. Minden csúcs piros vagy fekete.
2. A gyökér fekete.
3. Minden (NIL) levél fekete.
4. Minden piros csúcsnak mindkét gyereke fekete.
5. Minden csúcsra igaz, hogy az alatta lévő levelekhez vezető utakon ugyanannyi fekete csúcs található.

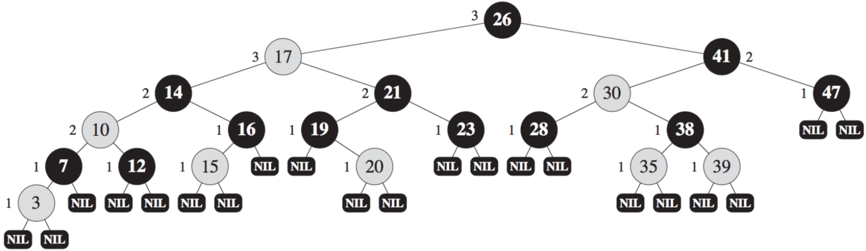
A piros-fekete fa

A piros-fekete fák kiegyensúlyozottak???





A piros-fekete fa

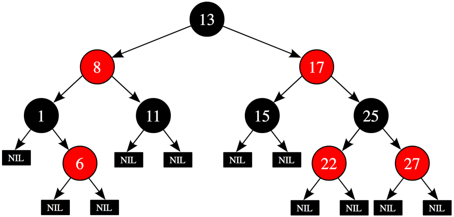


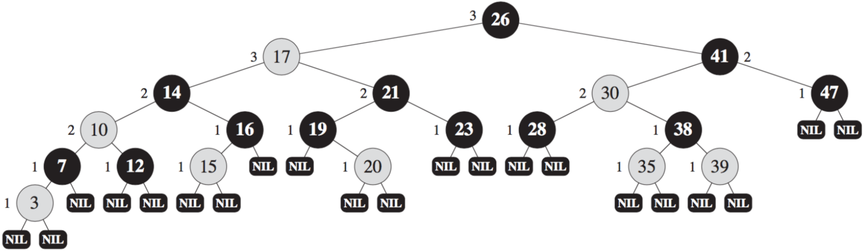
A piros-fekete fa tulajdonságai:

1. Minden csúcs piros vagy fekete.
2. A gyökér fekete.
3. Minden (NIL) levél fekete.
4. Minden piros csúcsnak mindkét gyereke fekete.
5. Minden csúcsra igaz, hogy az alatta lévő levelekhez vezető utakon ugyanannyi fekete csúcs található.

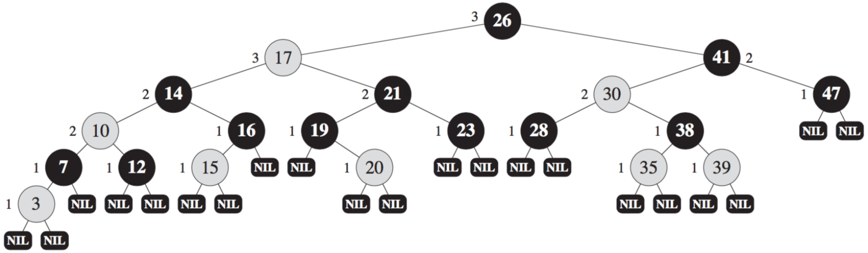
Piros-fekete fa – tulajdonságok

A piros-fekete fák
kiegyensúlyozottak???
Néha igen,
néha nem.





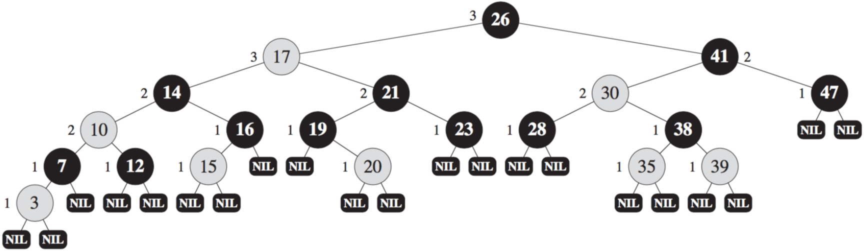
Piros-fekete fa – tulajdonságok



Lemma

Egy n közbenső csúccsal rendelkező piros-fekete fa maximális magassága $2 * log_2(n+1)$.

Piros-fekete fa – tulajdonságok



- TREE-SEARCH – $\mathcal{O}(log_2 n)$
- TREE-INSERT – $\mathcal{O}(log_2 n)$
- TREE-DELETE – $\mathcal{O}(log_2 n)$

Ez a célnak tökéletesen megfelelő, még akkor is, ha nem (mindig) kiegyensúlyozott.

Műveletek piros-fekete fákkal, mint adatszerkezetekkel

Műveletek:

- adatszerkezetek **létrehozása**: folytonos vagy láncolt reprezentációval
- adatszerkezetek **módosítása**
  - elem hozáadása: $\textsc{Tree-Insert} + \underline{\textsc{Recolor} + \textsc{Rotate}}$
  - elem törlése: $\textsc{Tree-Delete} + \underline{\textsc{Recolor} + \textsc{Rotate}}$
  - elem cseréje: nincs
- elem **elérése**: $\textsc{Iterative-Tree-Search}$

Mint látható, a "$\textsc{Rebalance}$" helyett "$\textsc{Recolor} + \textsc{Rotate}$" szerepel, mivel a piros-fekete fák nem feltétlen kiegensúlyozottak. Ezért a piros-fekete fa tulajdonságainak visszaállítása átszínezéssel és szükség esetén forgatással valósul meg.

Piros-fekete fa – forgatás

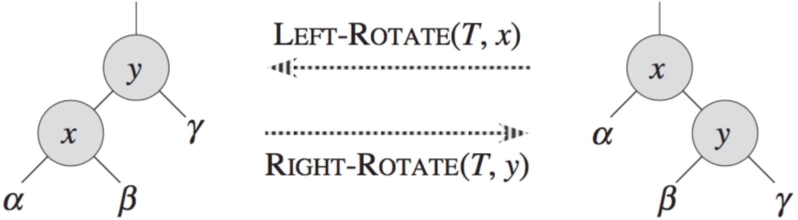
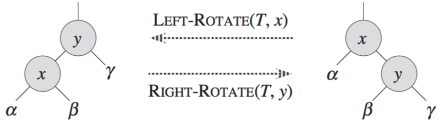


Figure 13.2 The rotation operations on a binary search tree. The operation LEFT-ROTATE(T, x) transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation RIGHT-ROTATE(T, y) transforms the configuration on the left into the configuration on the right. The letters α, β, and γ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in α precede $x.key$, which precedes the keys in β, which precede $y.key$, which precedes the keys in γ.

Piros-fekete fa – forgatás



LEFT-ROTATE(T, x)

```
y = x.right                    // set y
x.right = y.left               // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                      // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                     // put x on y's left
x.p = y
```

Piros-fekete fa – forgatás

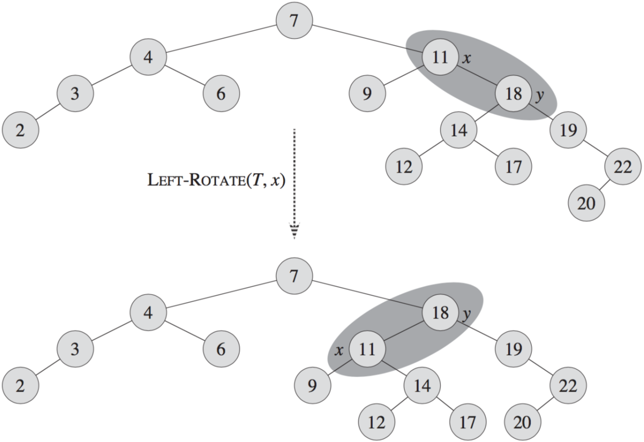


Figure 13.3 An example of how the procedure LEFT-ROTATE(T, x) modifies a binary search tree. Inorder tree walks of the input tree and the modified tree produce the same listing of key values.

Piros-fekete fa – beszúrás

Mindig piros csúcsot szúrunk be.
Melyik piros-fekete fa tulajdonság sérülhet új elem beszúrásakor?

- Az első, a harmadik és az ötödik tulajdonság nem sérülhet,
- csak a második és a negyedik tulajdonság sérülhet.

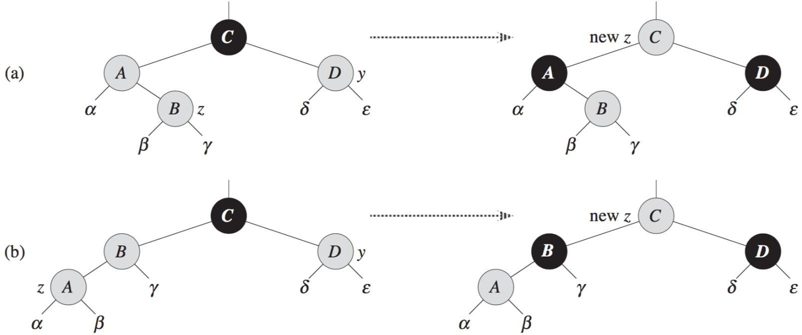
2. A gyökér fekete. – Ha a gyökér piros, színezzük át feketére!

4. Minden piros csúcsnak mindkét gyereke fekete. – Ez lehet az egyetlen komolyan felmerülő probléma.
Ebben az esetben három különböző esetet különböztetünk meg:

- 1. eset: z nagybácsikája piros,
- 2. eset: z nagybácsikája fekete és háromszögünk van,
- 3. eset: z nagybácsikája fekete és egyenesünk van,

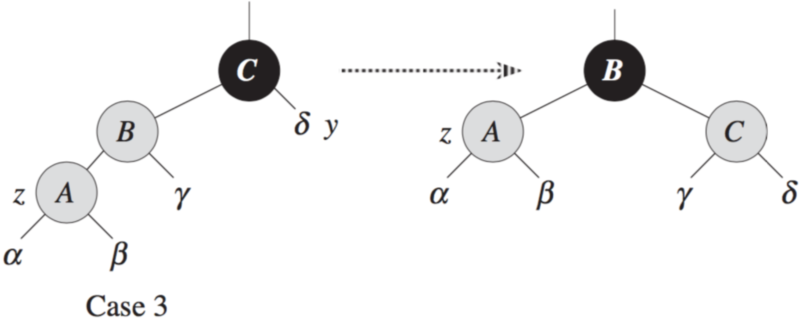
ahol z a problémás csúcs.

Piros-fekete fa – beszúrás – a nagybácsi piros



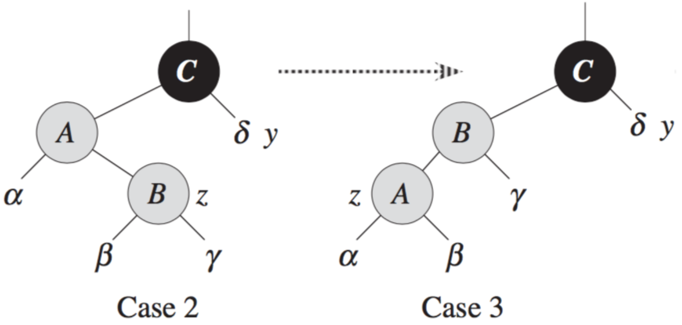
Megoldás: A szülő, a nagyszülő és a nagybácsi átszínezése.

Piros-fekete fa – beszúrás – egyenesünk van



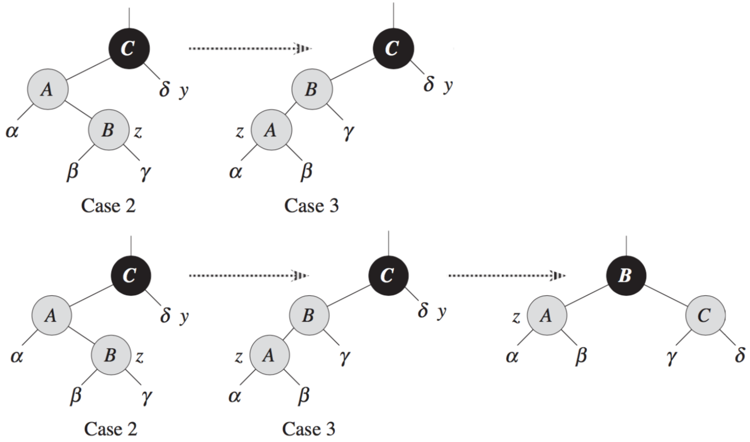
Megoldás: Egy forgatás, majd pedig a szülő és a testvér átszínezése.

Piros-fekete fa – beszúrás – háromszögünk van



Megoldás: Egy forgatás.

Piros-fekete fa – beszúrás – háromszögünk van



Piros-fekete fa – beszúrás – példa

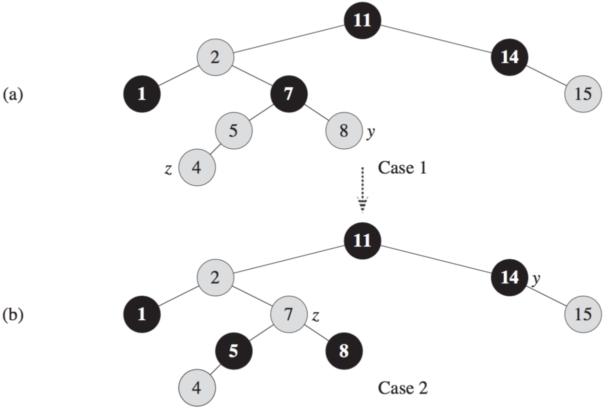


Figure 13.4 The operation of RB-INSERT-FIXUP. **(a)** A node z after insertion. Because both z and its parent $z.p$ are red, a violation of property 4 occurs. Since z's uncle y is red, case 1 in the code applies. We recolor nodes and move the pointer z up the tree, resulting in the tree shown in **(b)**. Once again, z and its parent are both red, but z's uncle y is black. Since z is the right child of $z.p$, case 2 applies. We perform a left rotation, and the tree that results is shown in **(c)**. Now, z is the left child of its parent, and case 3 applies. Recoloring and right rotation yield the tree in **(d)**, which is a legal red-black tree.

Piros-fekete fa – beszúrás – példa

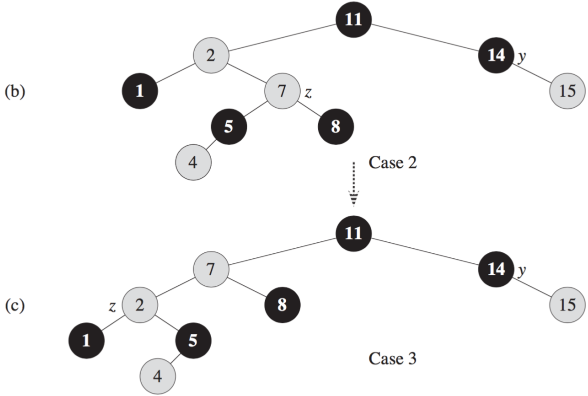


Figure 13.4 The operation of RB-INSERT-FIXUP. **(a)** A node z after insertion. Because both z and its parent $z.p$ are red, a violation of property 4 occurs. Since z's uncle y is red, case 1 in the code applies. We recolor nodes and move the pointer z up the tree, resulting in the tree shown in **(b)**. Once again, z and its parent are both red, but z's uncle y is black. Since z is the right child of $z.p$, case 2 applies. We perform a left rotation, and the tree that results is shown in **(c)**. Now, z is the left child of its parent, and case 3 applies. Recoloring and right rotation yield the tree in **(d)**, which is a legal red-black tree.

Piros-fekete fa – beszúrás – példa

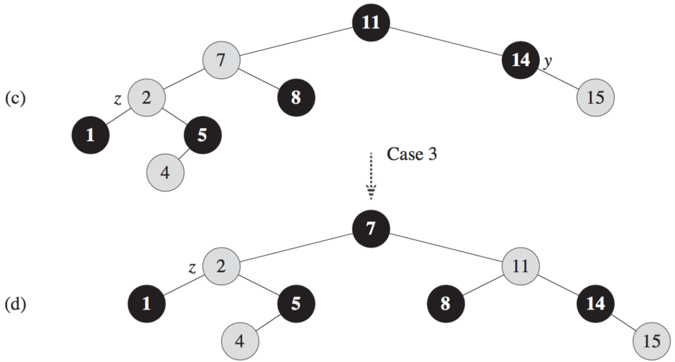


Figure 13.4 The operation of RB-INSERT-FIXUP. **(a)** A node z after insertion. Because both z and its parent $z.p$ are red, a violation of property 4 occurs. Since z's uncle y is red, case 1 in the code applies. We recolor nodes and move the pointer z up the tree, resulting in the tree shown in **(b)**. Once again, z and its parent are both red, but z's uncle y is black. Since z is the right child of $z.p$, case 2 applies. We perform a left rotation, and the tree that results is shown in **(c)**. Now, z is the left child of its parent, and case 3 applies. Recoloring and right rotation yield the tree in **(d)**, which is a legal red-black tree.

Irodalomjegyzék