Abstraktni podatkovni tipi

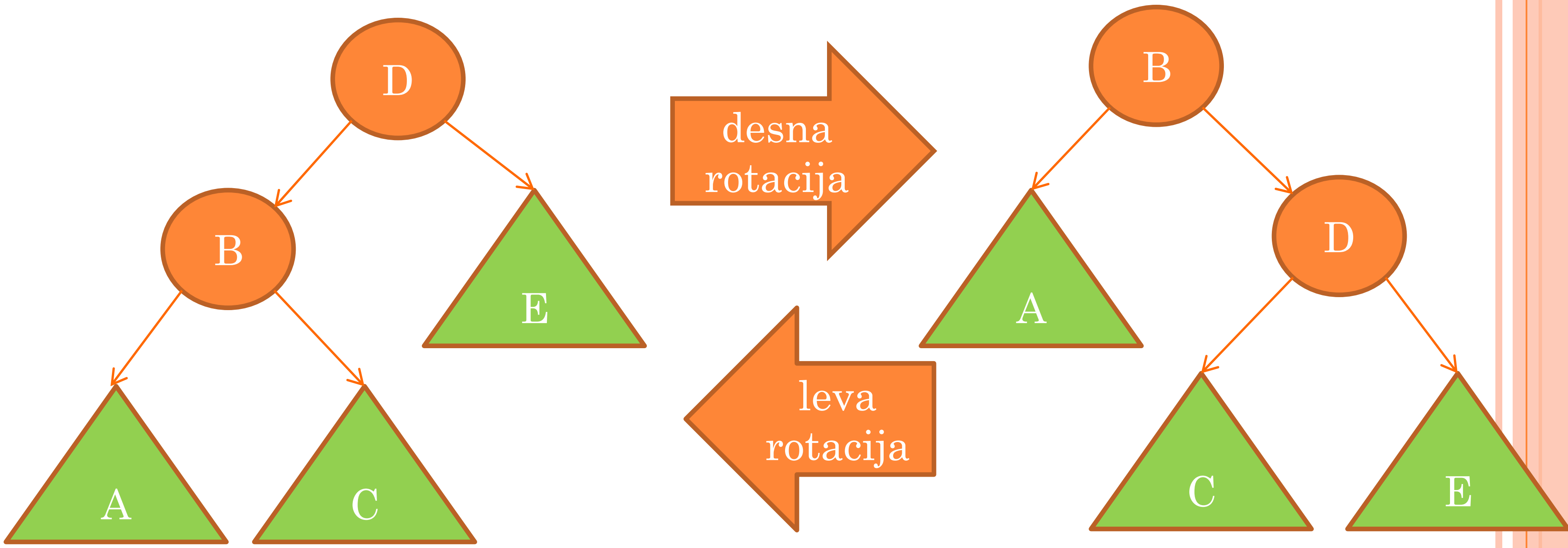
Osnovni abstraktni podatkovni tipi, ki jih potrebujemo za razvoj algoritmov so:

- seznam (list)
- vrsta (queue)
- sklad (stack)
- preslikava (map)
- množica (set)
- drevo (tree)
- slovar (dictionary)

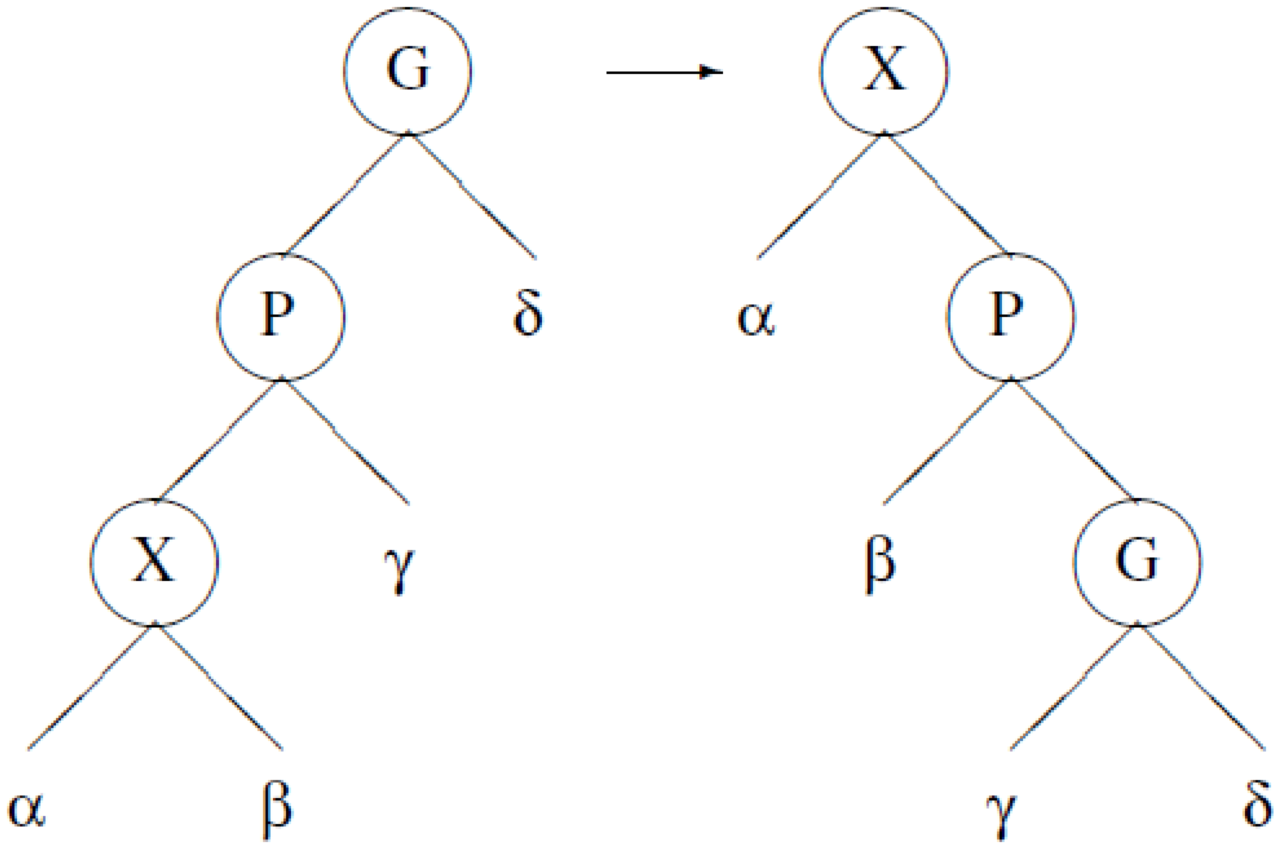
Implementacije ADT slovarja:

- binarno iskalno drevo (BST)
- lomljena drevesa – samo informativno
- rdeče-črno drevo (red-black tree)

ROTACIJA - ENOJNA

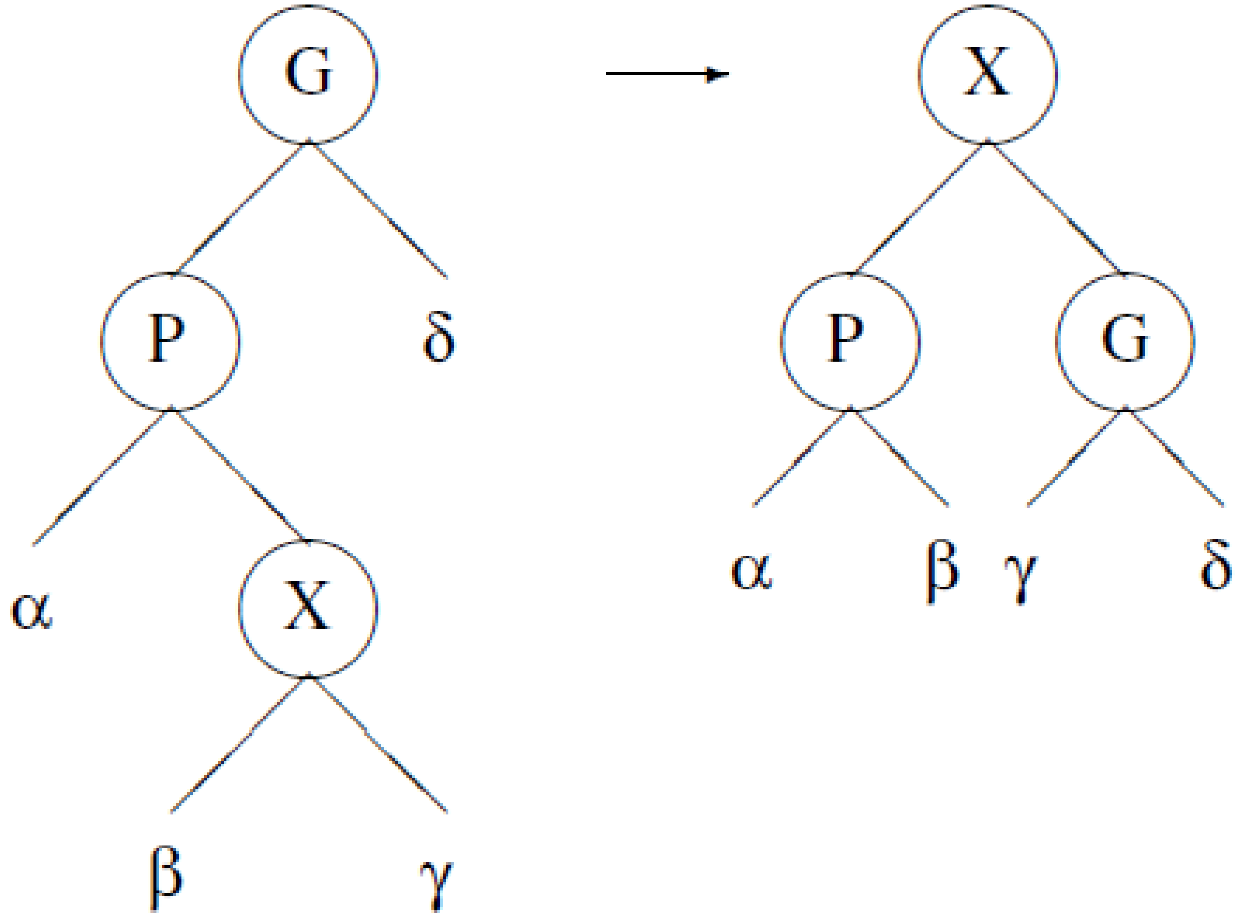


Rotacija – dvojna 1



Rotacija – dvojna 2

Enako kot dve enojni rotaciji…



Lomljena drevesa

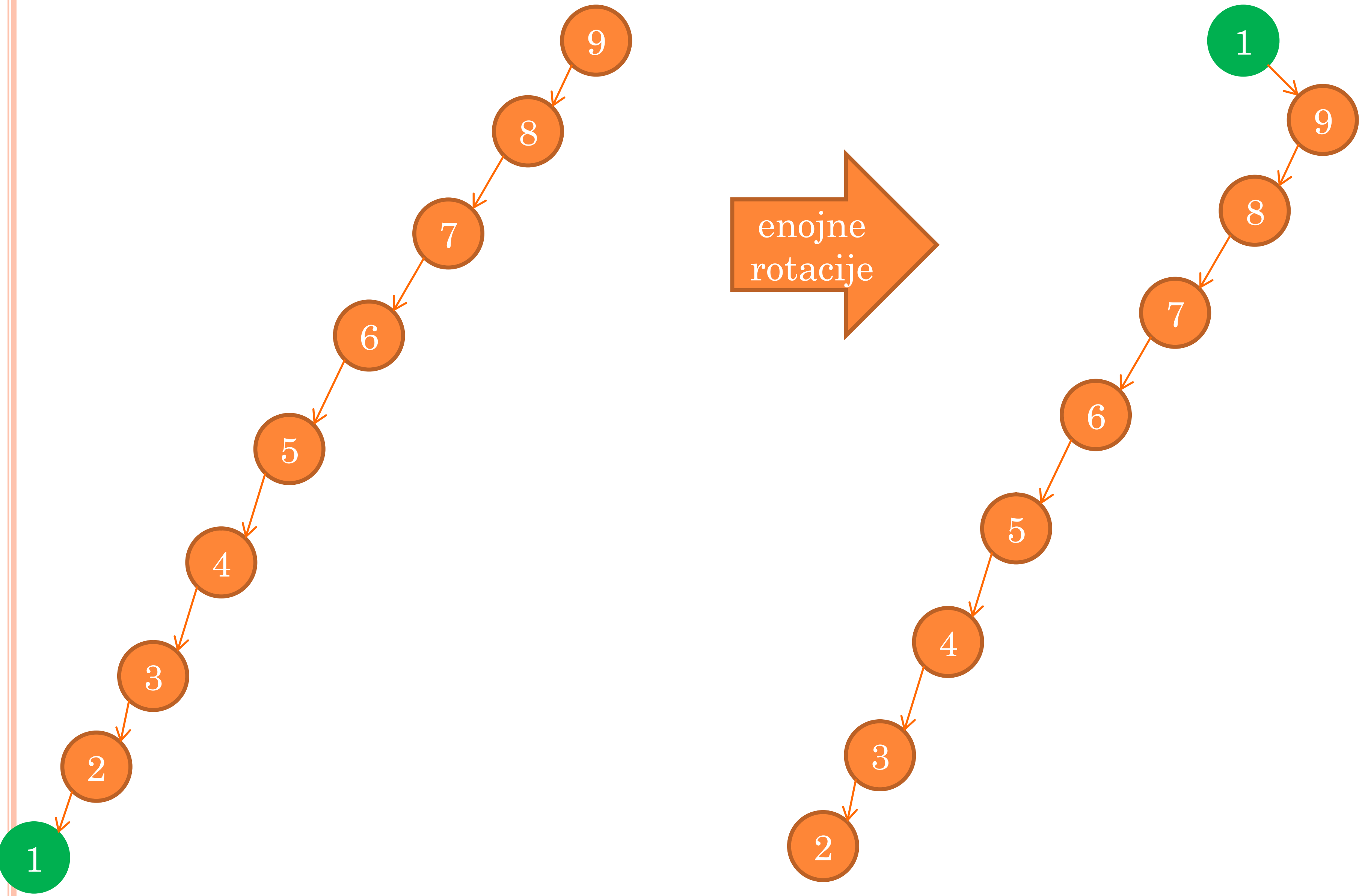
Lomljeno drevo (splay tree):

- Pri vsaki operaciji se drevo spremeni – se lomi.
- Uporablja dvojne rotacije za dvig elementa v koren – lomljenje

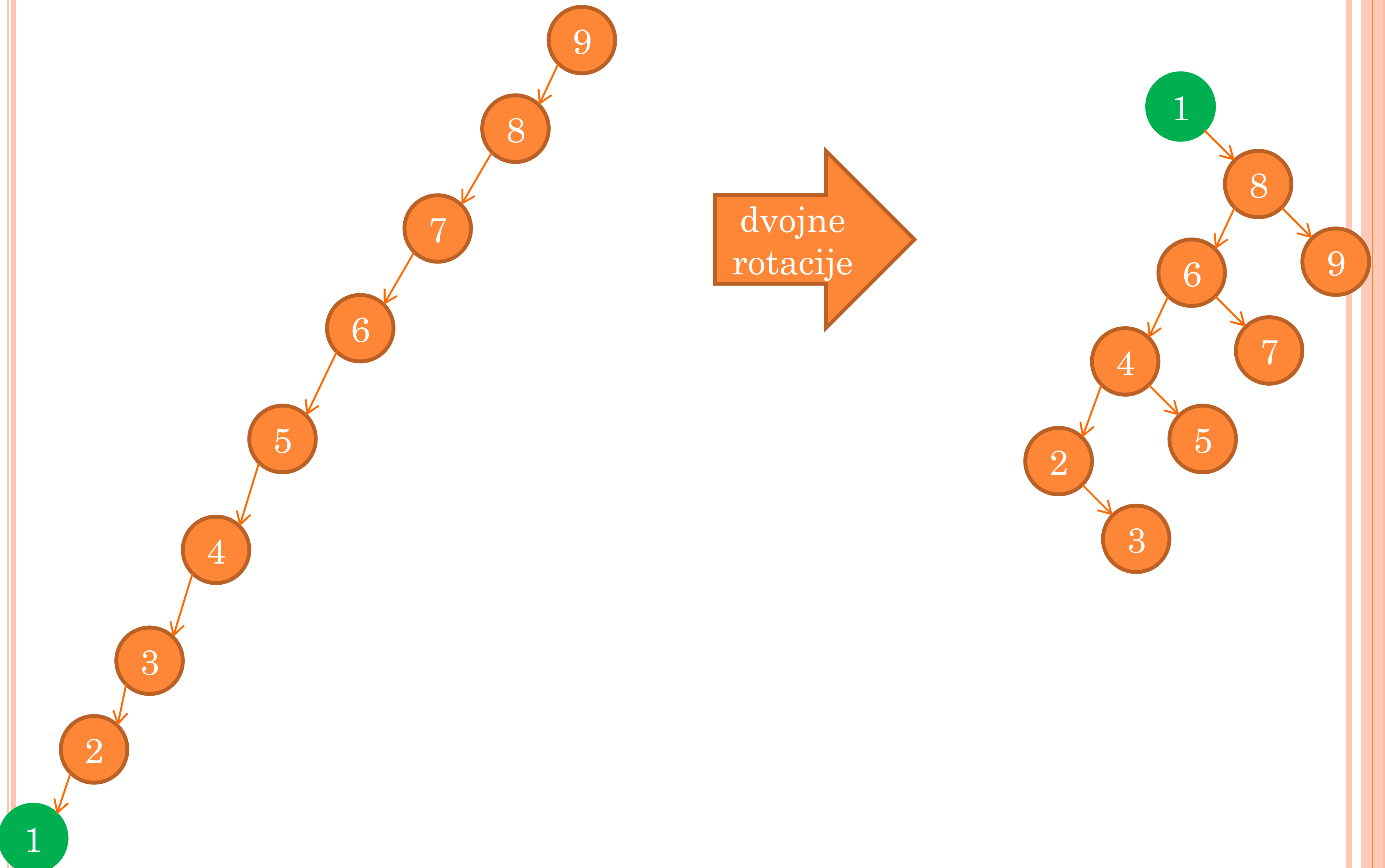
- Pri **iskanju** se najdeni element (ali oče praznega lista, če elementa ni v drevesu) z lomljenjem dvigne v koren.
- Pri **vstavljanju**: element vstavi kot list in ga z lomljenjem dvigne v koren.
- Pri **brisanju**: element najprej z lomljenjem dvignemo v koren, zatem z lomljenjem dvignemo najmanjši element v levem poddrevesu, ki nadomesti izbrisani element v korenu.

- Na dolgi rok drevo ne more ostati izrojeno – v povprečju so vse operacije učinkovite: O(log n)

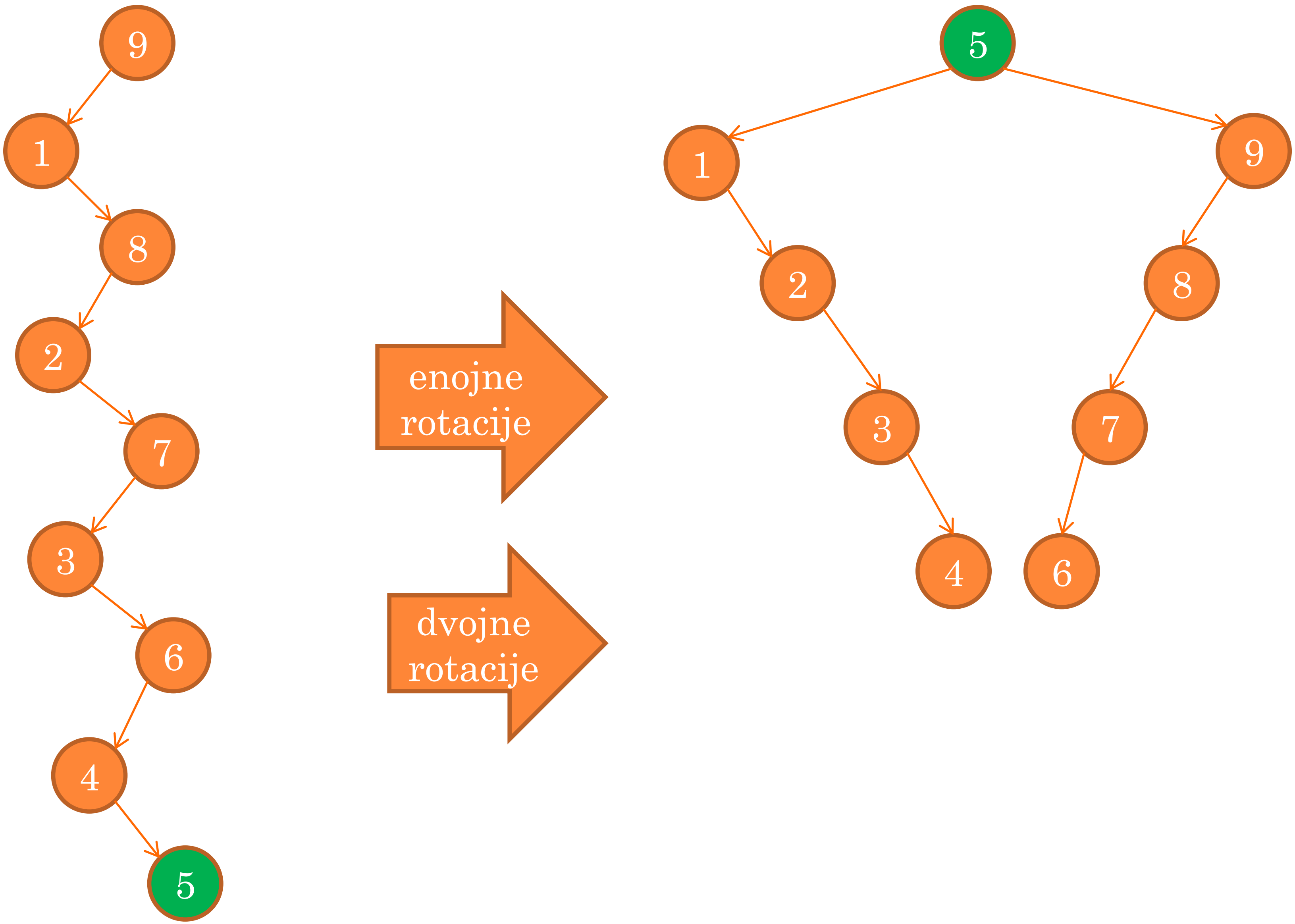
Lomljena drevesa

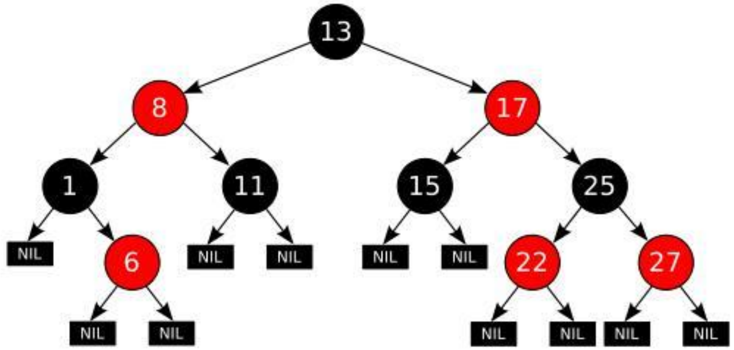


Lomljena drevesa



Lomljena drevesa



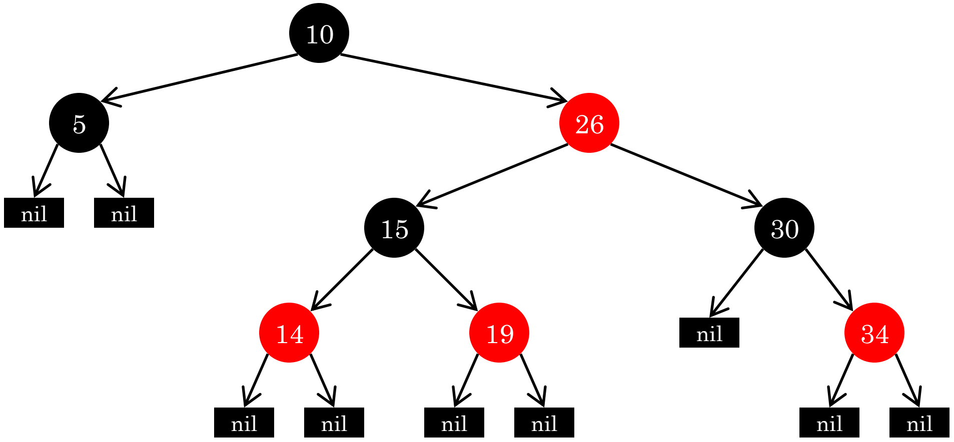


RDEČE-ČRNO DREVO (angl. *red-black tree*)

Rdeče-črno drevo

Binarno iskalno drevo za katerega velja:

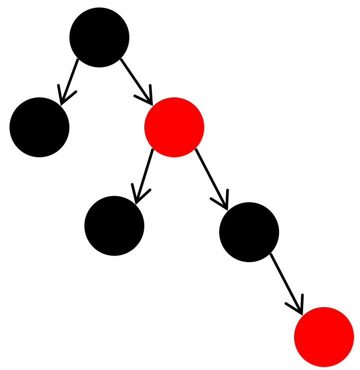
- vsako vozlišče je bodisi **rdeče** bodisi **črne** barve
- **rdeče** vozlišče ima lahko samo **črna** sinova
- za **vsako** vozlišče velja, da **vsaka pot** od vozlišča do praznega poddrevesa (nil) vsebuje **enako** število **črnih** vozlišč (**črna višina** je konstantna)



Višina rdeče-črnega drevesa

- višina rdeče-črnega drevesa z n vozlišči je največ $2\log_2(n + 1)$
  (dokaz v knjigi, stran 177)

- rdeče-črno drevo je **vedno** delno poravnano
    - višina drevesa je največ dvakrat večja od poravnanega drevesa z istim številom vozlišč
    - najdaljša pot od korena do listov je kvečjemu dvakrat daljša od najkrajše poti od korena do listov



IMPLEMENTACIJA RB DREVES

K običajnemu BST vozlišču dodamo še:
- kazalec na očeta
- podatek o barvi vozlišča

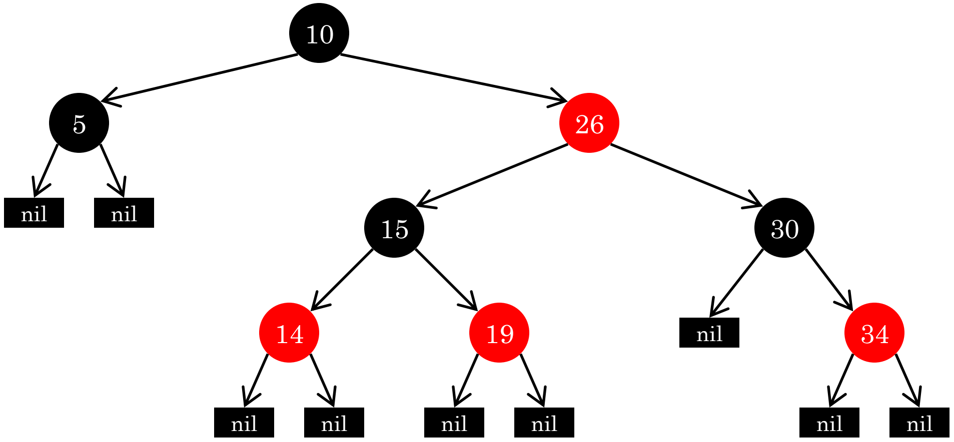
```
public class RBTreeNode extends BSTreeNode {
  RBTreeNode parent ;
  int color ;
  } // class RBTreeNode
```

```
public class BSTreeNode {
  Comparable key ;
  BSTreeNode left, right ;
} // class BSTreeNode
```

Rdeče-črno drevo

Zagotavlja časovno zahtevnost osnovnih operacij reda *O(log n)*:

- **iskanje**: enako kot pri običajnem BST → *O(log n)*
- **dodajanje**: dodamo **rdeči** list; eventuelno potrebno popravljanje, ki se v najslabšem primeru nadaljuje vse do korena →
  → *O(2log n)* = *O(log n)*
- **brisanje**: nadomestimo element z minimalnim iz desnega poddrevesa (ali z maksimalnim iz levega poddrevesa), in če je minimalni (zbrisani) **črn**, je potrebno popravljanje, ki se v najslabšem primeru nadaljuje do korena → *O(2log n)* = *O(log n)*



Dodajanje elementa v rdeče-črno drevo

1. Element dodamo v list drevesa kot pri navadnem BST.
2. Dodano vozlišče (list) pobarvamo **rdeče**.
3. Če je oče dodanega lista **rdeč**, je potrebno drevo popraviti:

   3.1 oče je koren drevesa → postopek se zaključi

   3.2 stric je **rdeč** → stari oče postane **rdeč** → **ponovi 3.** pri starem očetu

   3.3 stric **ni rdeč** → postopek se zaključi

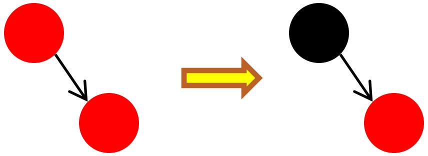
Dodajanje elementa v rdeče-črno drevo

1. Element dodamo v list drevesa kot pri navadnem BST.

2. Dodano vozlišče (list) pobarvamo **rdeče**.

3. Če je oče dodanega lista **rdeč**, je potrebno drevo popraviti:

   3.1 Če je oče koren drevesa, ga pobarvamo s **črno** barvo in končamo.



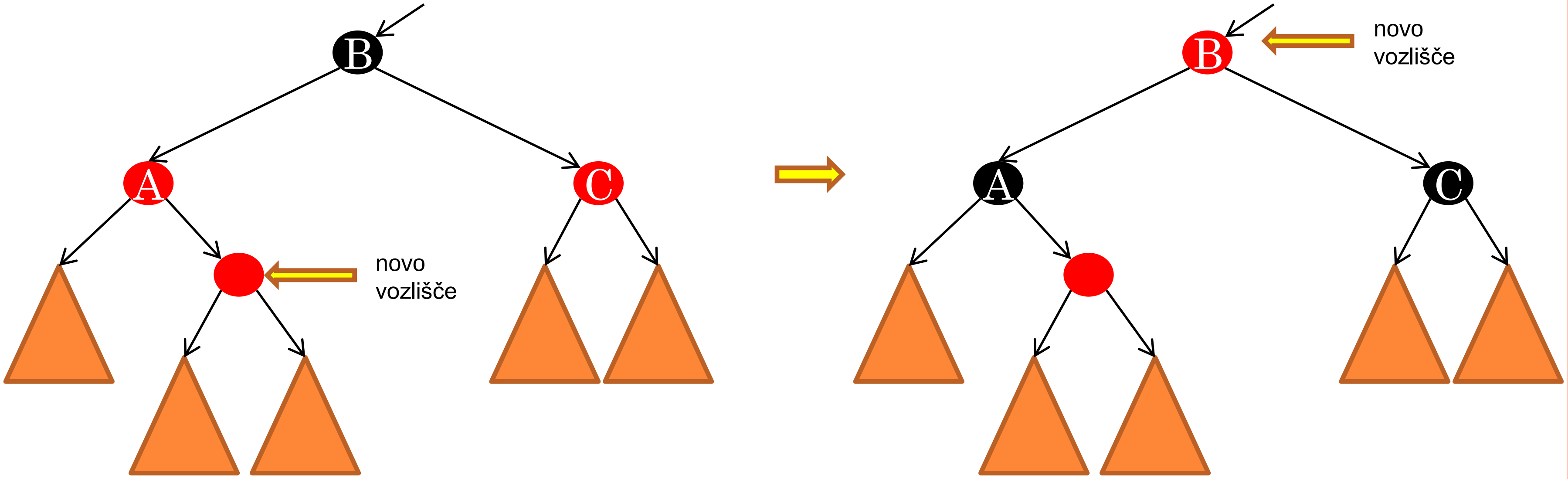
Dodajanje elementa v rdeče-črno drevo

1. Element dodamo v list drevesa kot pri navadnem BST.

2. Dodano vozlišče (list) pobarvamo **rdeče**.

3. Če je oče dodanega lista **rdeč**, je potrebno drevo popraviti:

   3.2 Če je stric C novega vozlišča **rdeče** barve, potem očeta A in strica C pobarvamo s **črno** barvo, starega očeta B pa z **rdečo**. Če ima stari oče **rdečega** očeta, postopek 3. rekurzivno ponovimo tako, da starega očeta proglasimo za novo vozlišče.



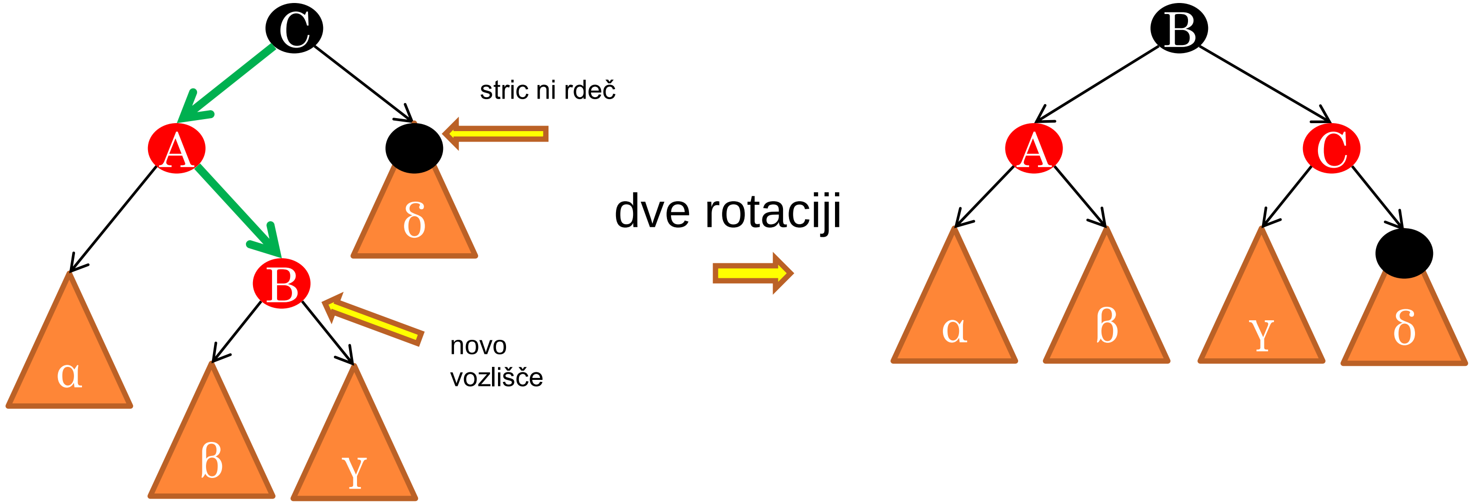
Dodajanje elementa v rdeče-črno drevo

1. Element dodamo v list drevesa kot pri navadnem BST.

2. Dodano vozlišče (list) pobarvamo **rdeče**.

3. Če je oče dodanega lista **rdeč**, je potrebno drevo popraviti:

   3.3 Če je stric **črne** barve ali ne obstaja, potem izvedemo eno ali dve rotaciji na poddrevesu starega očeta C in končamo.



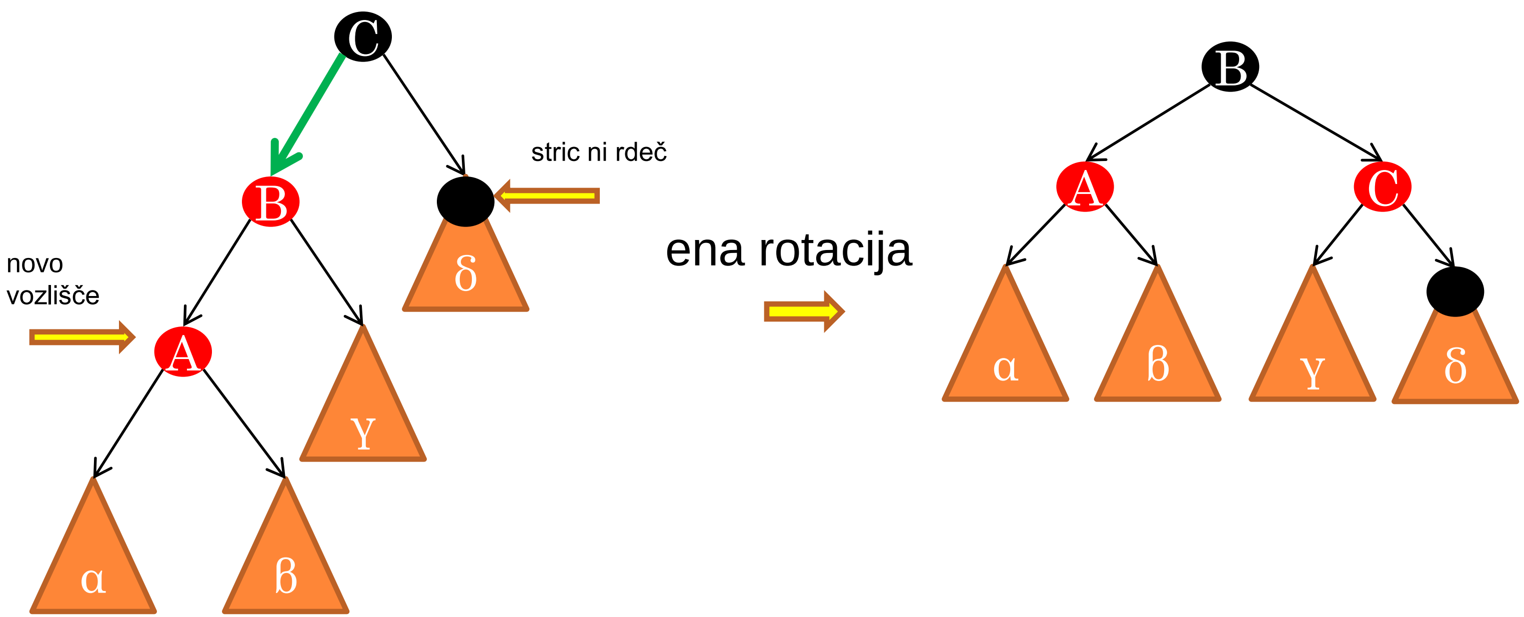
Dodajanje elementa v rdeče-črno drevo

1. Element dodamo v list drevesa kot pri navadnem BST.

2. Dodano vozlišče (list) pobarvamo **rdeče**.

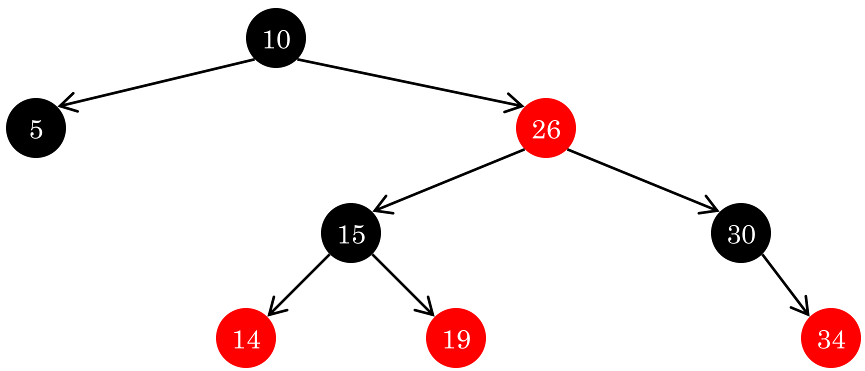
3. Če je oče dodanega lista **rdeč**, je potrebno drevo popraviti:

   3.3 Če je stric **črne** barve ali ne obstaja, potem izvedemo eno ali dve rotaciji na poddrevesu starega očeta C in končamo.



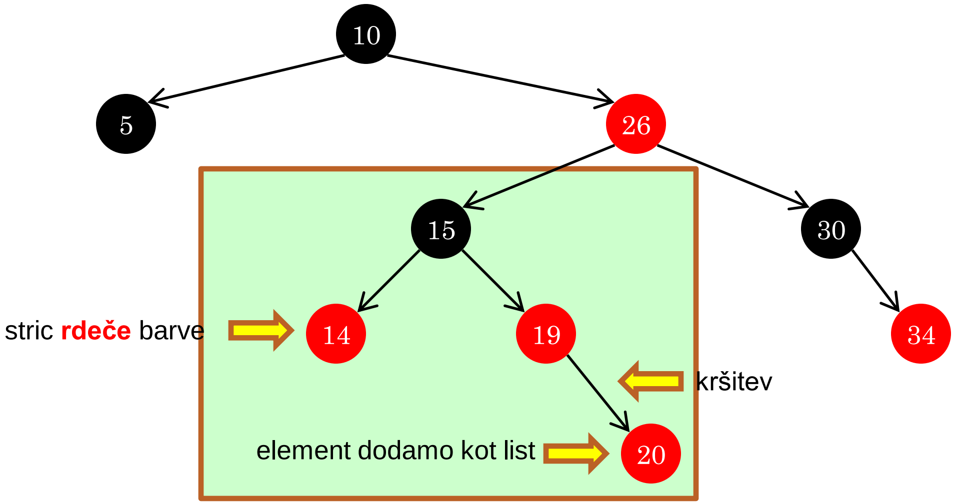
PRIMER (1/4)

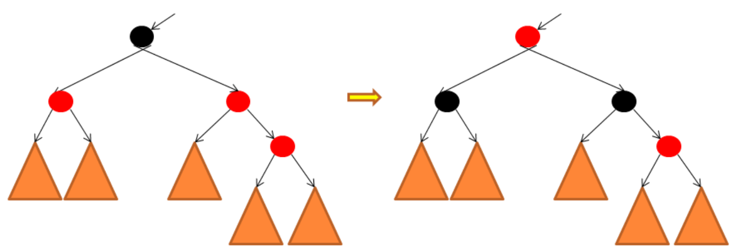
Podano je rdeče-črno drevo:



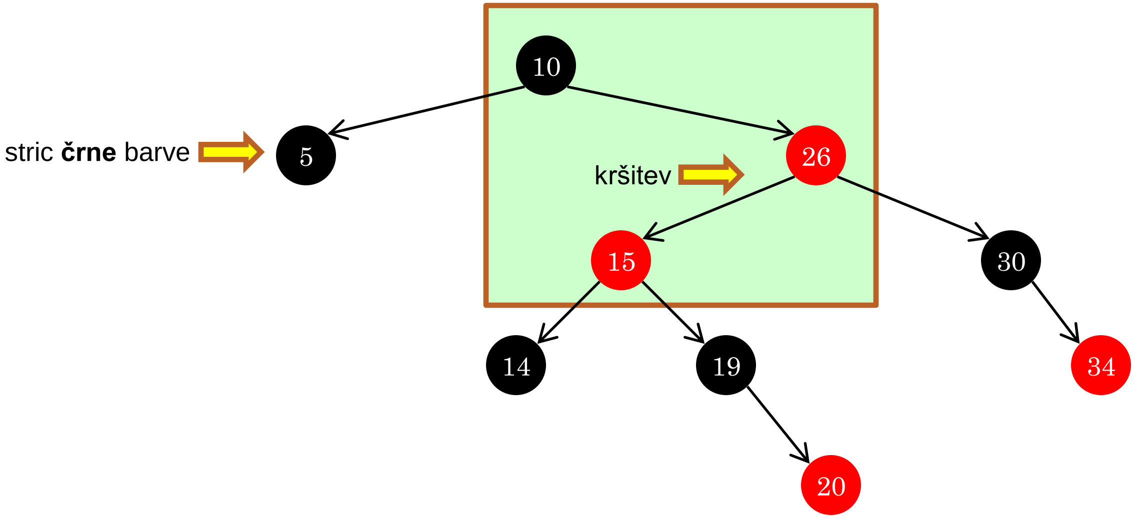
PRIMER (2/4)

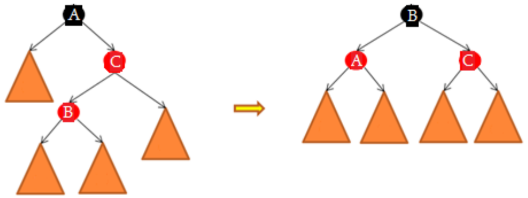
Dodamo element 20:



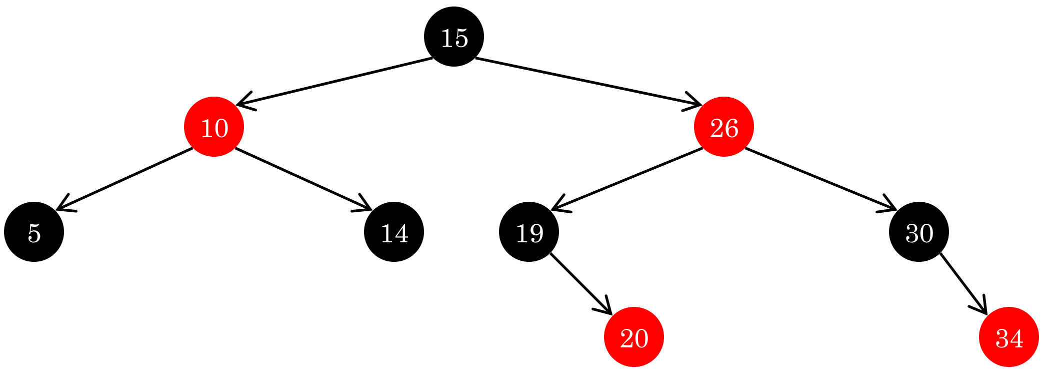


Primer (3/4)





Primer (4/4)



Brisanje elementa iz RB drevesa

1. Element zbrišemo iz drevesa kot pri navadnem BST:
   1.1 če je element list drevesa, ga enostavno zbrišemo,
   1.2 če ima element samo enega sina, ga zbrišemo ter na njegovo mesto postavimo njegovega sina,
   1.3 če ima element dva sina, zbrišemo največji element iz levega poddrevesa ali najmanjši element iz desnega poddrevesa, ki nadomesti dejansko zbrisano vozlišče.

   Vozlišče, ki ga brišemo, ima kvečjemu enega sina.

2. a) Če je zbrisano **rdeče** vozlišče, končamo.

   b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti (**črna višina danega poddrevesa se je znižala za ena**)

Brisanje elementa iz RB drevesa

2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

   1. Če je koren problematičnega poddrevesa **rdeč** → zaključi

   2. Če je zbrisan koren drevesa → konec

   3. Preurejanje drevesa:

      3.1 **rdeč** brat → **črn** brat →3.2

      3.2 **črn** brat in ni rdečega nečaka → ponovi cel postopek pri očetu

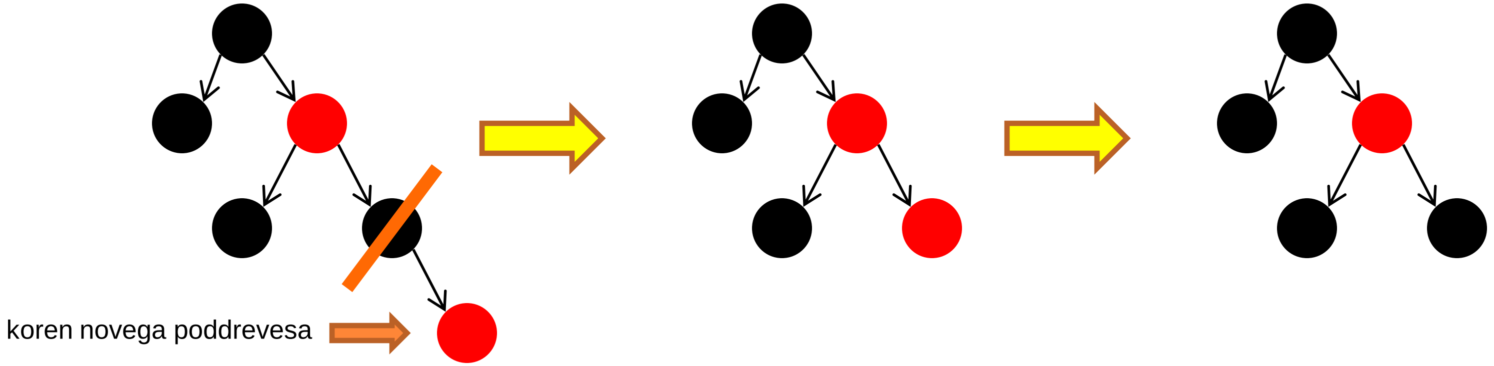
      3.3. **črn** brat in **črn** zunanji nečak → **rdeč** zunanji nečak → 3.4

      3.4 **črn** brat in **rdeč** zunanji nečak → postopek se zaključi

Brisanje elementa iz RB drevesa

2\. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

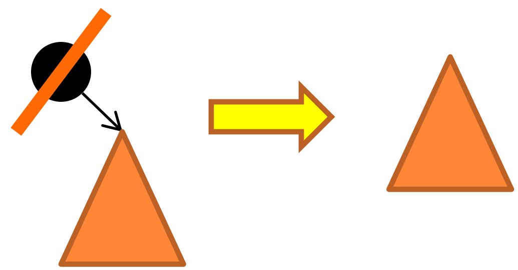
1. če je bil sin zbrisanega vozlišča **rdeč** (sedaj je to koren problematičnega poddrevesa), ga pobarvamo **črno** in končamo (črna višina je urejena)



Brisanje elementa iz RB drevesa

2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

   2. če je zbrisan koren drevesa, ki je imelo eno samo poddrevo, je postopek končan

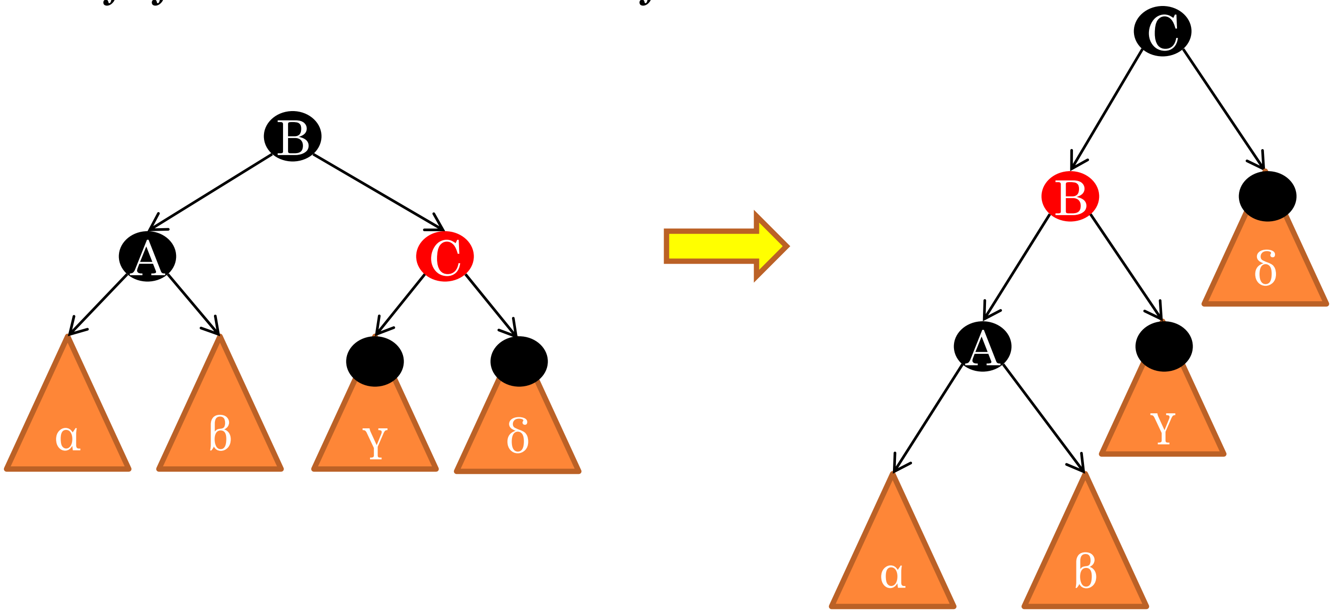


Brisanje elementa iz RB drevesa

2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

3.1 če je koren (A) problematičnega poddrevesa **črn** in je njegov brat (C) **rdeč** (kar pomeni, da je oče B **črn**), rotiramo brata in očeta ter zamenjamo njuni barvi.

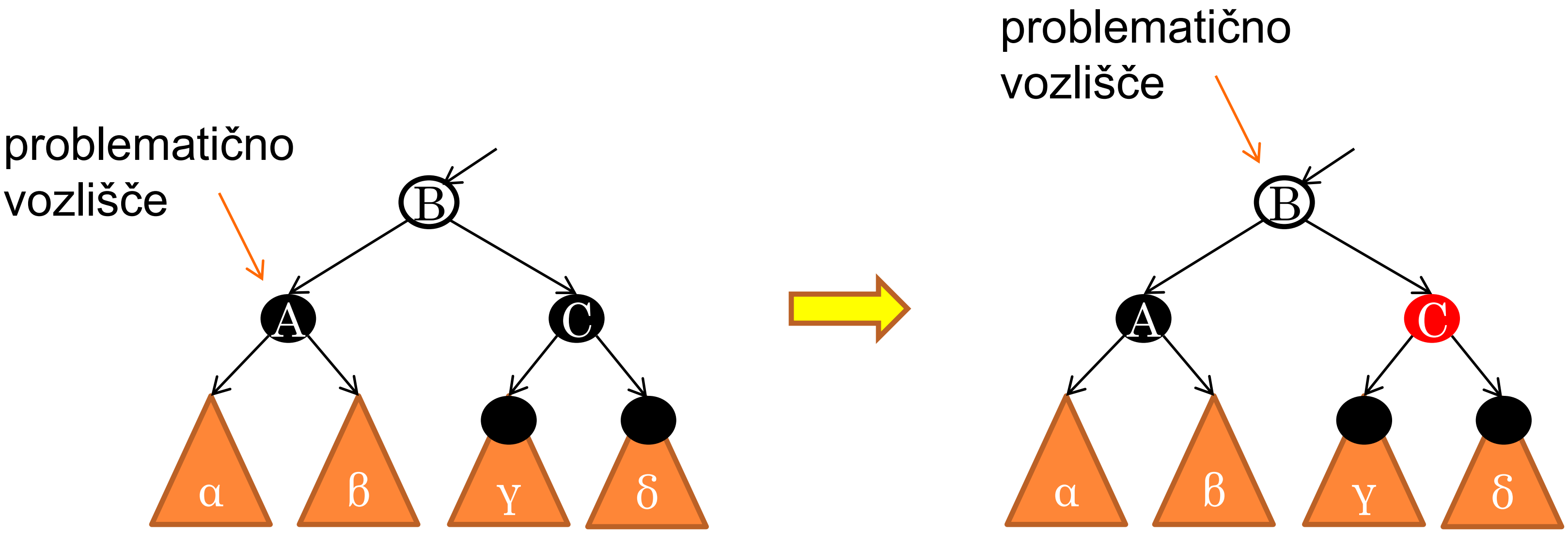
Ta korak je predpriprava! Dobili smo **črnega** brata. Nadaljujemo s transformacijo 3.2



Brisanje elementa iz RB drevesa

2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

   3.2 Če je brat C **črn** in sta nečaka **črna**, pobarvamo brata v **rdeče** in rekurzivno ponovimo celoten postopek z novim vozliščem B.

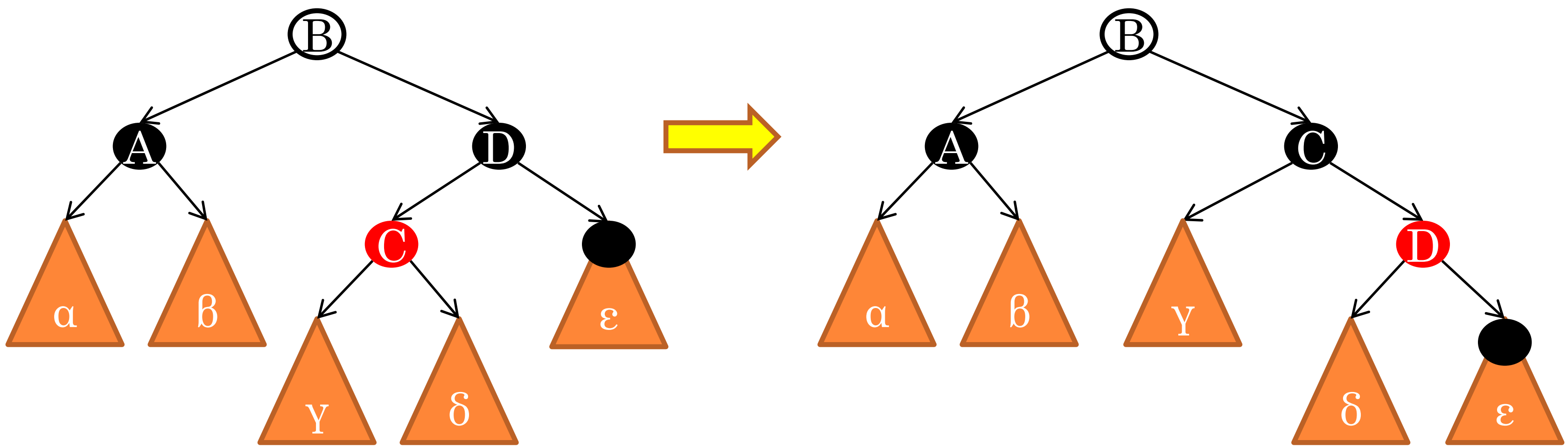


Brisanje elementa iz RB drevesa

2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

3.3 Če je brat D **črn** in je zunanji nečak **črn**, rotiramo brata in notranjega nečaka in zamenjamo njuni barvi.

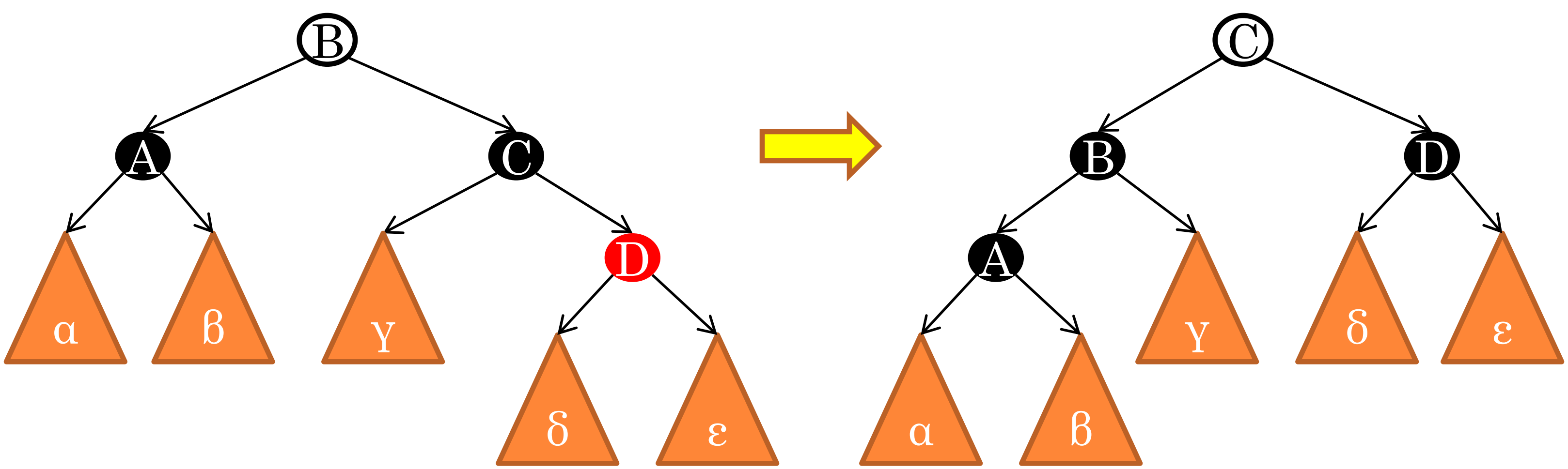
Ta korak je predpriprava! Dobili smo **rdečega** zunanjega nečaka. Nadaljujemo s transformacijo 3.4.



Brisanje elementa iz RB drevesa

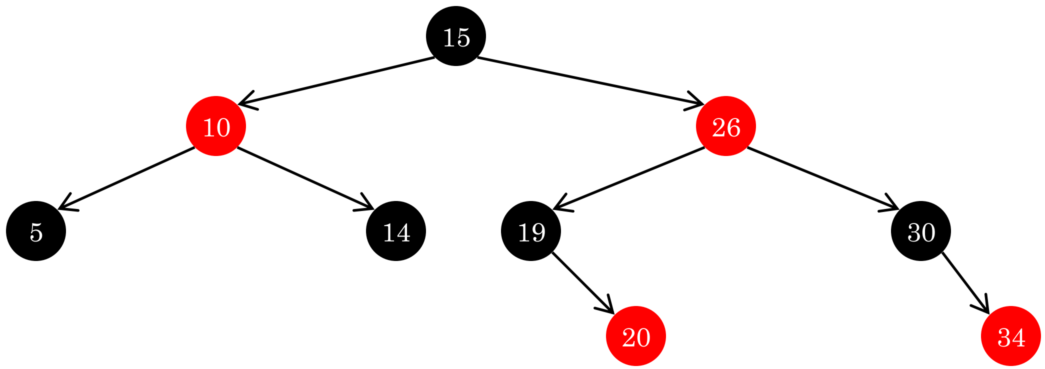
2. b) Če je zbrisano **črno** vozlišče, je potrebno drevo popraviti:

   3.4 Če je brat C **črn** in je zunanji nečak **rdeč**, izvedemo enojno rotacijo med očetom in bratom in končamo, ker se je črna višina problematičnega poddrevesa povečala za ena.



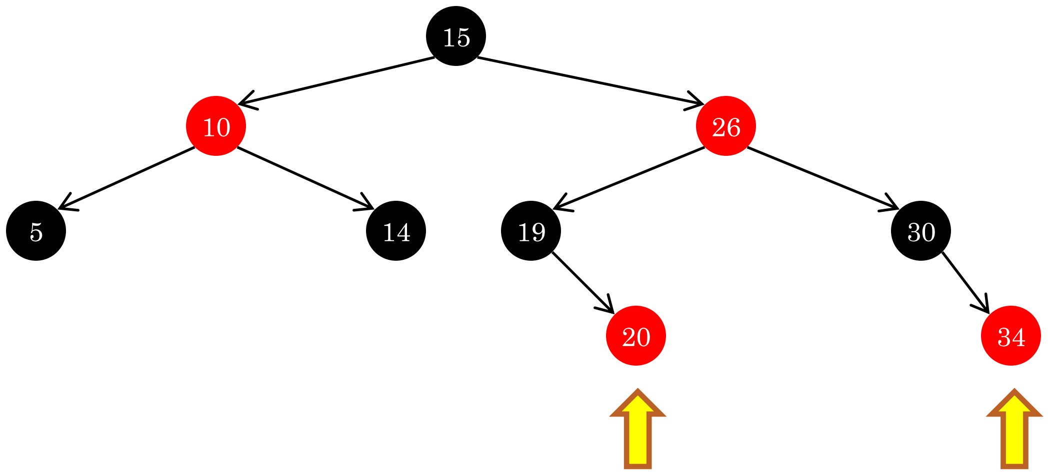
Primer (1/13)

Podano je rdeče-črno drevo.
 Izbriši elemente 34, 20, 30, 26 in 19 v tem vrstnem redu.



PRIMER (2/13)

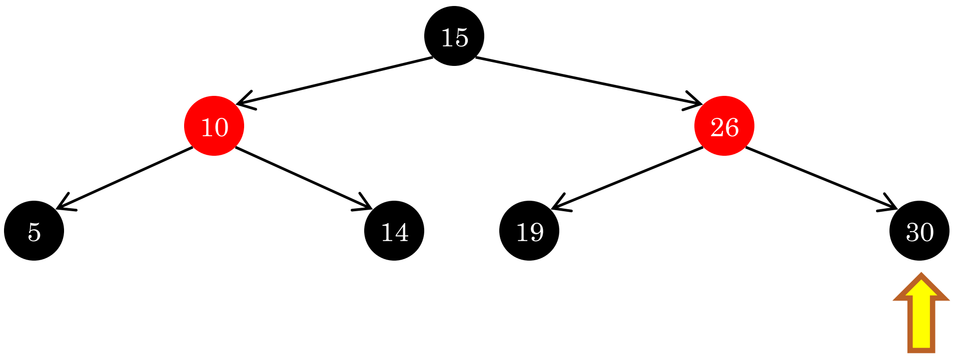
Brišemo elementa 34 in 20…



elementa 34 in 20 sta **rdeča** lista, zato ju samo zbrišemo iz drevesa.

Primer (3/13)

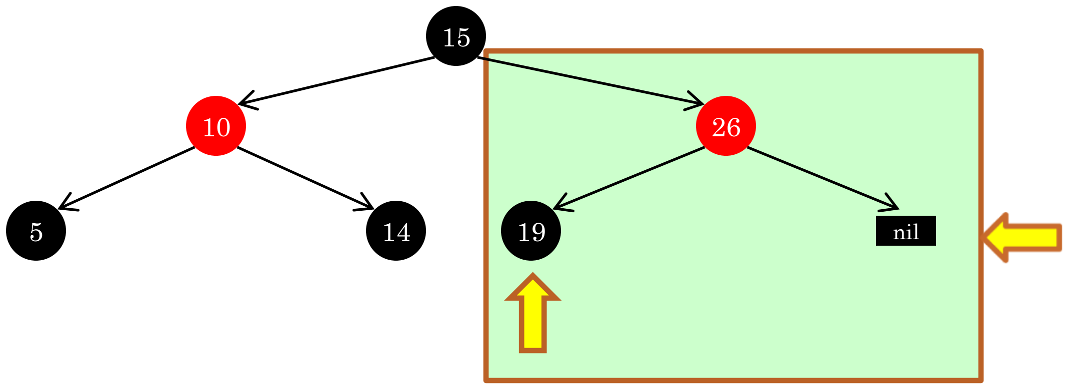
Brišemo element 30…



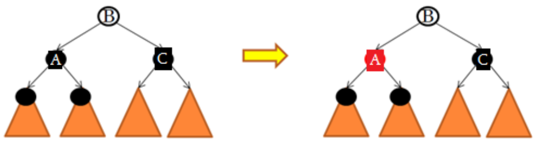
vozlišče 30 zamenjamo s praznim drevesom (nil),
ki je po definiciji **črne** barve

PRIMER (4/13)

Brišemo element 30…

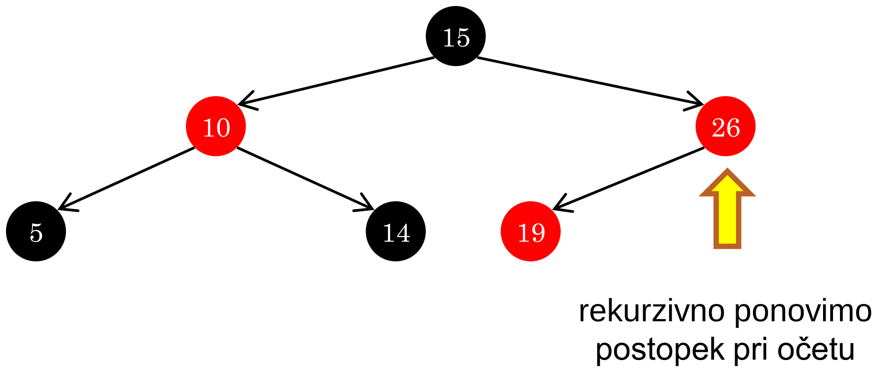


brat je **črne** barve,
nečaka (oba nil) sta **črne** barve



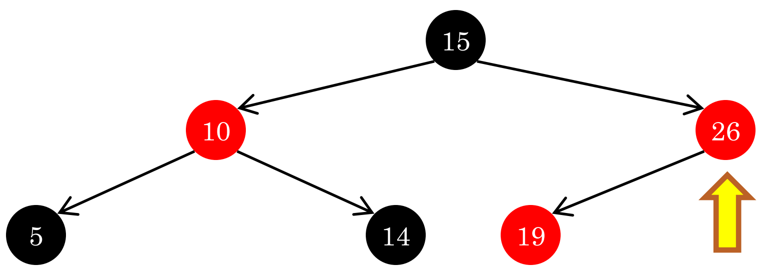
Primer (5/13)

Brišemo element 30…



Primer (6/13)

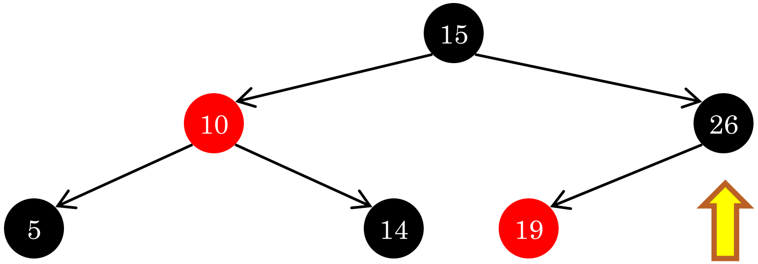
Brišemo element 30…



koren novega poddrevesa je **rdeč**, zato ga pobarvamo **črno** in končamo

Primer (7/13)

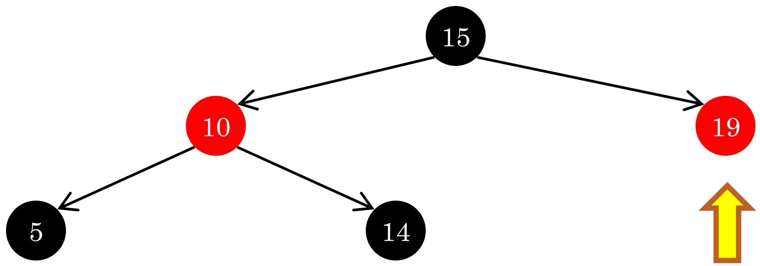
Brišemo element 26…



vozlišče 26 zamenjamo z edinim sinom

PRIMER (8/13)

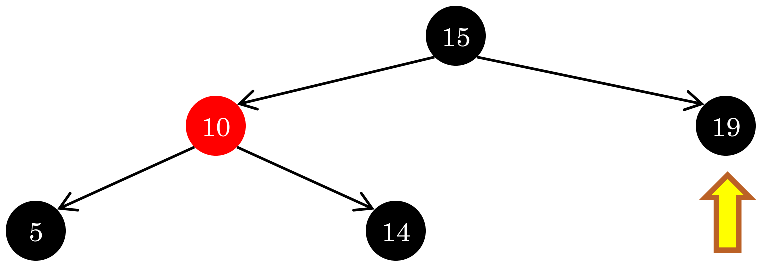
Brišemo element 26…



koren novega poddrevesa je **rdeč**,
zato ga pobarvamo **črno** in končamo

PRIMER (9/13)

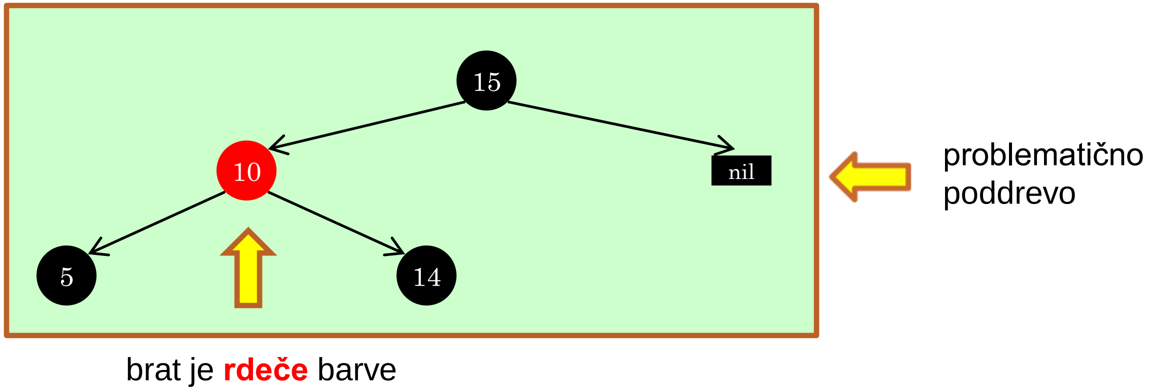
Brišemo element 19…

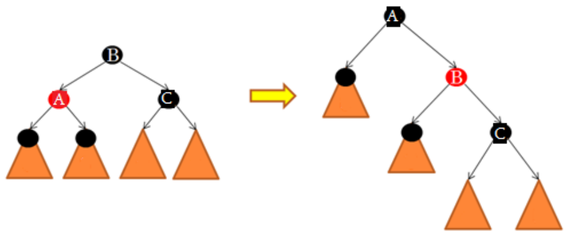


vozlišče 19 zamenjamo s praznim drevesom (nil), ki je po definiciji **črne** barve

Primer (10/13)

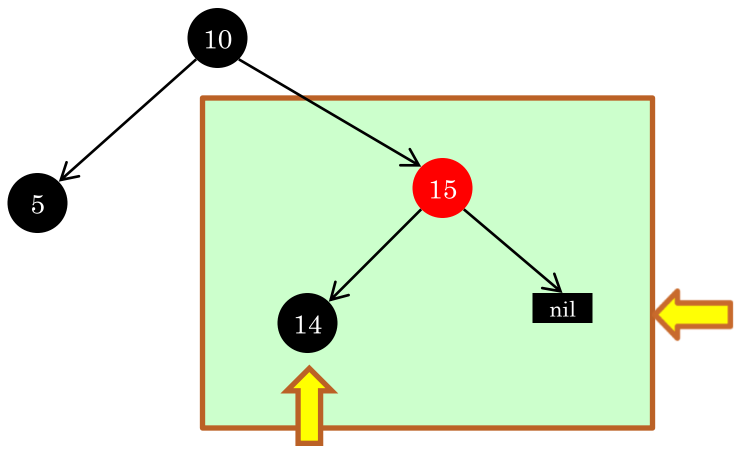
Brišemo element 19…





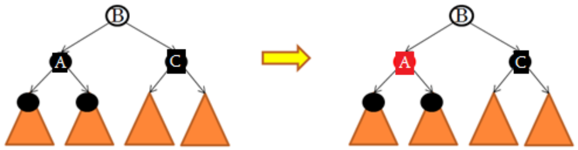
Primer (11/13)

Brišemo element 19…



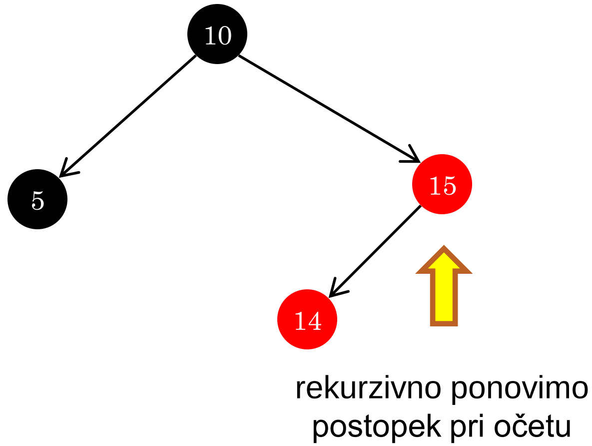
problematično
poddrevo

brat je **črne** barve,
nečaka (oba nil) sta **črne** barve



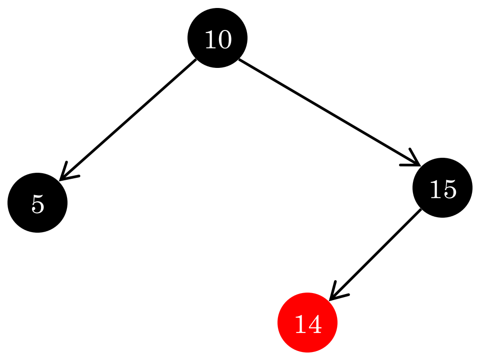
PRIMER (12/13)

Brišemo element 19…



Primer (13/13)

Končno drevo:



POVZETEK: RB-DREVO

Zagotavlja časovno zahtevnost osnovnih operacij reda *O(log n)*:

- **iskanje**: enako kot pri običajnem BST → *O(log n)*
- **dodajanje**: dodamo **rdeči** list; eventuelno potrebno popravljanje, ki se v najslabšem primeru nadaljuje vse do korena →
  → *O(2log n)* = *O(log n)*
- **brisanje**: nadomestimo element z minimalnim iz desnega poddrevesa (ali z maksimalnim iz levega poddrevesa), in če je minimalni (zbrisani) **črn**, je potrebno popravljanje, ki se v najslabšem primeru nadaljuje do korena → *O(2log n)* = *O(log n)*