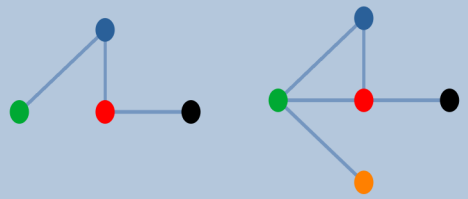
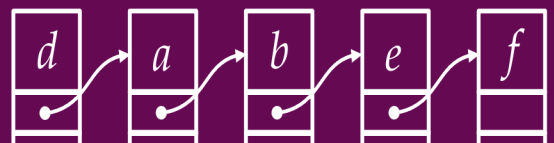
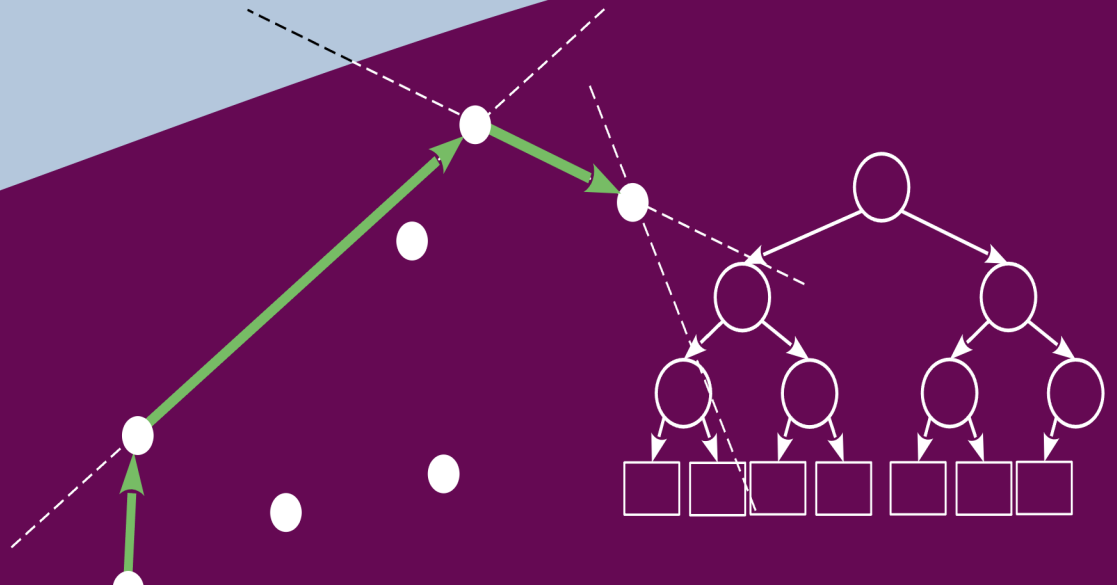
Advanced algorithms and data structures

Week 2: B-trees and Red-Black trees (RB)

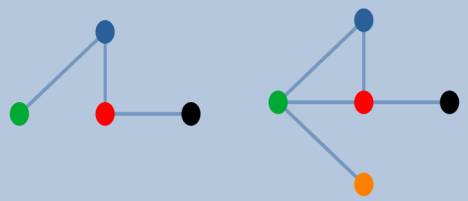


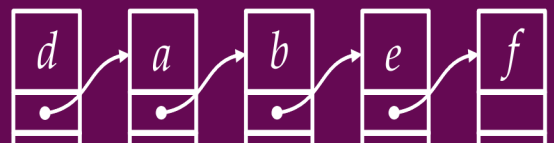


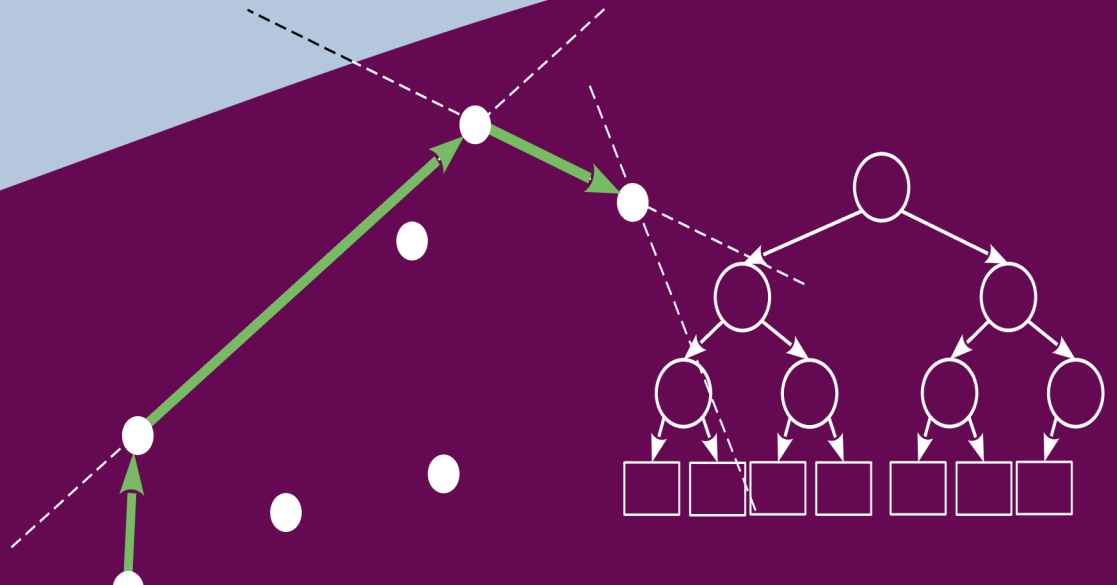


Napredni algoritmi i strukture podataka

Tjedan 2: B-stabla i Crveno-crna stabla (RB)







Creative Commons

you are free to:

to share — to reproduce, distribute and communicate the work to the public, to

adapt the work

under the following conditions:

Attribution: You must acknowledge and attribute the authorship of the work in a way specified by the author or licensor (but not in a way that suggests that you or your use of their work has their direct endorsement).

non-commercial: You may not use this work for commercial purposes.

share under the same conditions: if you modify, transform, or create using this work, you may distribute the adaptation only under a license that is the same or similar to this one.

In the case of further use or distribution, you must make clear to others the license terms of this work. Any of the above conditions may be waived with the permission of the copyright holder.
Nothing in this license infringes or limits the author's moral rights.

The text of the license is taken from http://creativecommons.org/

Creative Commons

slobodno smijete:

dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo

prerađivati djelo

pod sljedećim uvjetima:

imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).

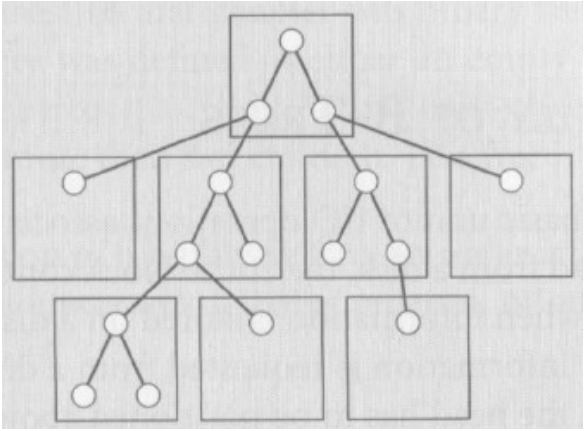
nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.

dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

Motivation

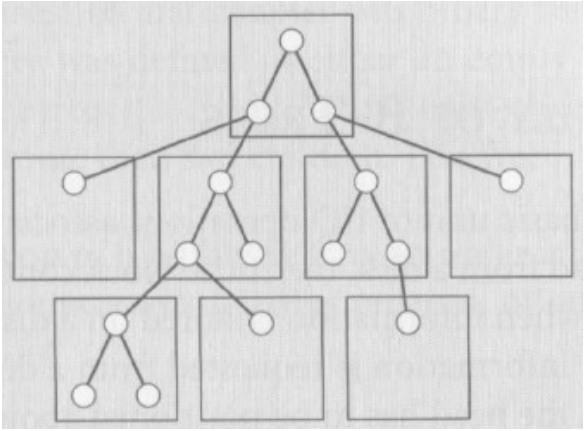
- External memory
  - Sequential reading by blocks
  - Neighboring nodes in the tree can be scattered in distant blocks



- B-trees alleviate the effects of the sequential block read limitation
  - The size of the node adjusts to the size of the block

Motivacija

- Vanjska memorija
  - Sekvencijalno čitanje po blokovima
  - Susjedni čvorovi u stablu mogu biti razasuti u udaljenim blokovima



- B-stabla ublažavaju efekte ograničenja sekvencijalnog blokovskog čitanja
  - Veličina čvora se prilagođava veličini bloka

Characteristics

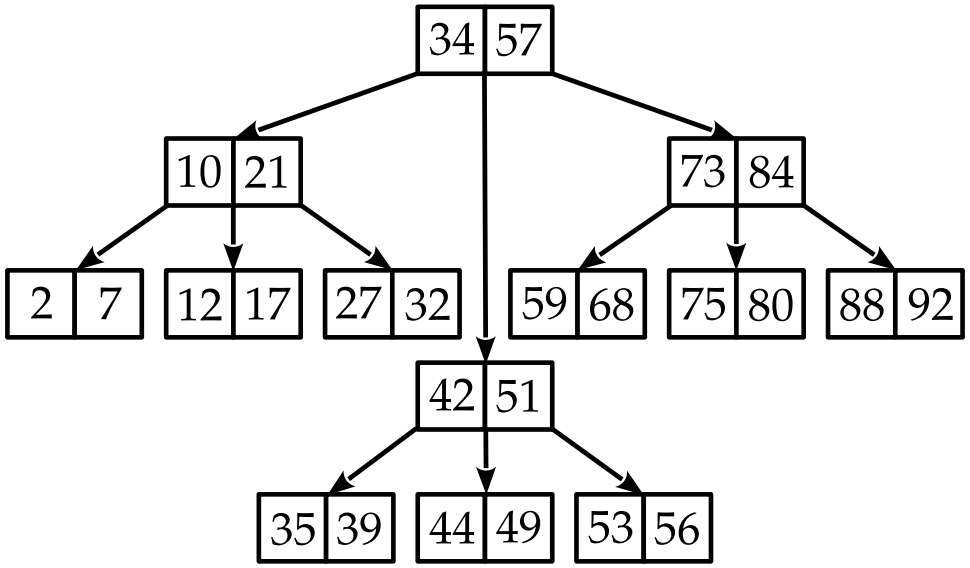
- Complete balance
- Sort data by key value
- Storing a certain number of elements in one node

Karakteristike

- Potpuna balansiranost
- Sortiranje podataka po vrijednosti ključa
- Čuvanje određenog broja elemenata u jednom čvoru

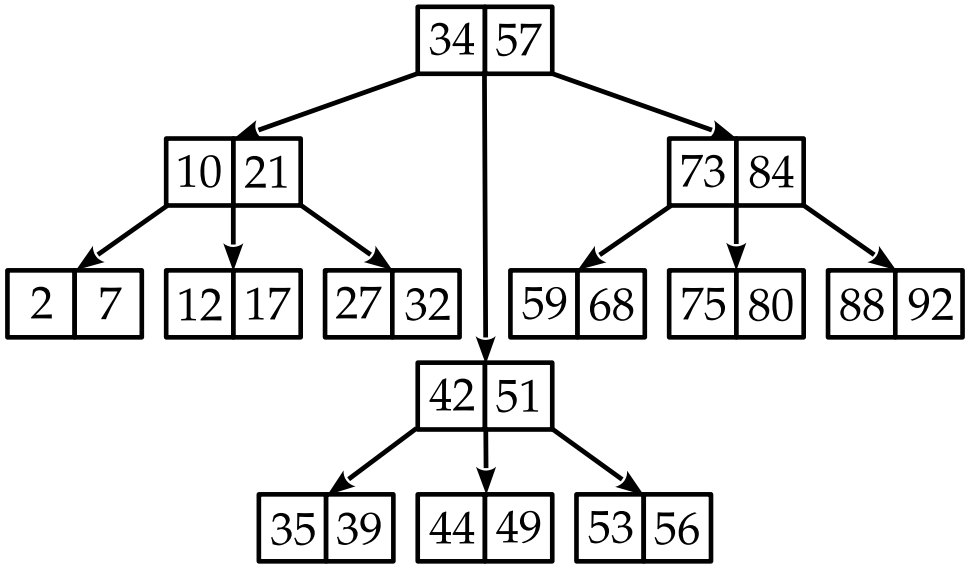
M trees

- M trees (*multiway tree*): trees in which nodes can have an arbitrary number of children
- M tree *m*-of order: M tree in which nodes can have at most *m* children.



M stabla

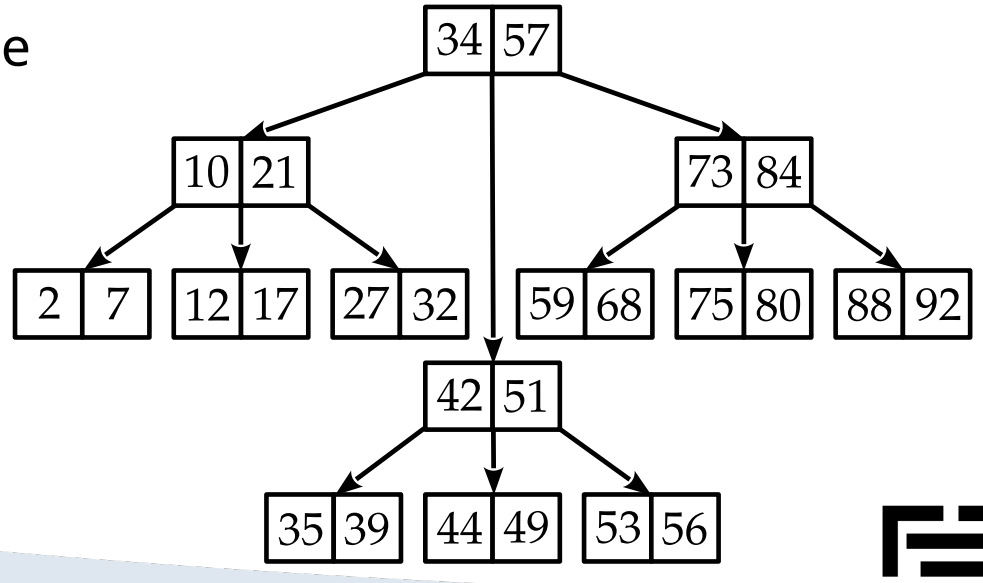
- M stabla (*multiway tree*): stabla u kojima čvorovi mogu imati proizvoljan broj djece
- M stablo *m*-tog reda: M stablo u kojem čvorovi mogu imati najviše *m* djece.



M trees

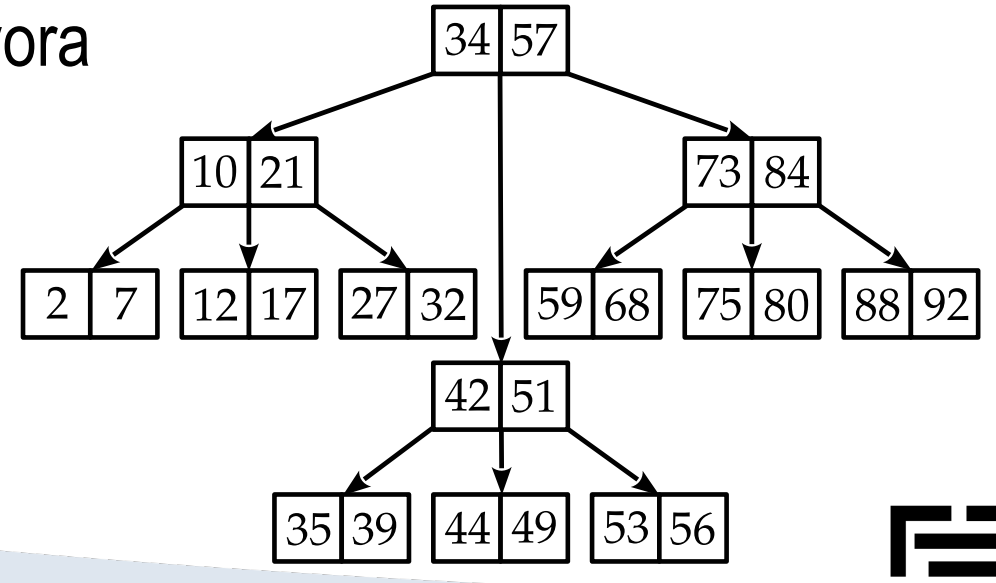
- Properties of an M-tree *m*- of that order:

  1. Each node has a maximum *m* children and *m* – *1* data (keys)
  2. The keys in the nodes are sorted
  3. Keys in the first *and* children of a node are smaller than *and* of the key of the observed node
  4. Keys in the last *me* children of a node are greater than *and* of the key of the observed node



M stabla

- Svojstva M stabla *m*-tog reda:
  1. Svaki čvor ima najviše *m* djece i *m–1* podataka (ključeva)
  2. Ključevi u čvorovima su sortirani
  3. Ključevi u prvih *i* djece nekog čvora su manji od *i*-tog ključa promatranog čvora
  4. Ključevi u zadnjih *m–i* djece nekog čvora su veći od *i*-tog ključa promatranog čvora



B-tree

- B tree *m*-of order is an M-tree with additional properties:

1. The root has at least two children, unless it is also a leaf (the only node in the tree).

Utilization ≥50% - internal

2. Each node, apart from the roots and leaves, contains **at least** *k*–*1* keys and *k* pointers to subtrees (has *k* children), whereby

$$\lceil m/2 \rceil \leq k \leq m$$

Utilization ≥50% - leaves

3. All sheets contain **at least** *k*–*1* keys, where is it

$$\lceil m/2 \rceil \leq k \leq m$$

4. All sheets are on the same level } Perfect balance

B-stablo

- B stablo *m*-tog reda je M-stablo sa dodatnim svojstvima:
  1. Korijen ima najmanje dvoje djece, osim ako je ujedno i list (jedini čvor u stablu)
  2. Svaki čvor, osim korijena i listova, sadrži **barem** *k–1* ključeva i *k* pokazivača na podstabla (ima *k* djece), pri čemu je
     $$\lceil m/2 \rceil \leq k \leq m$$
  3. Svi listovi sadrže **barem** *k–1* ključeva, pri čemu je
     $$\lceil m/2 \rceil \leq k \leq m$$
  4. Svi listovi su na istoj razini

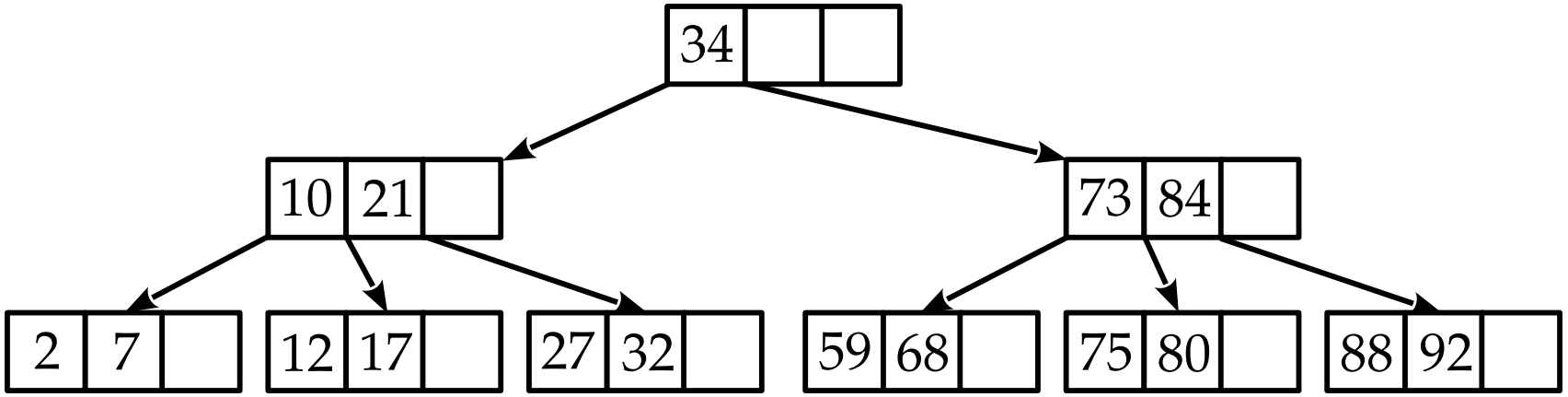
Iskorištenje ≥50% - unut.čv.

Iskorištenje ≥50% - listovi

Savršena uravnoteženost

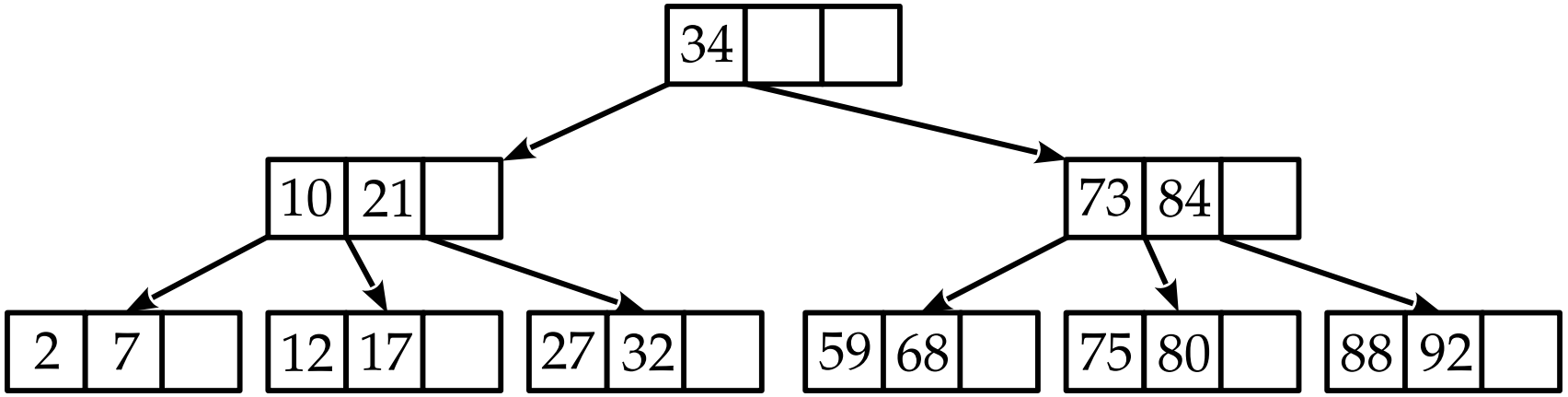
B-tree

- Peculiarities
  - Occupancy at least 50%
  - Perfectly balanced



B-stablo

- Osobitosti
  - Popunjenost barem 50%
  - Savršeno uravnoteženo



B-tree

- Implementation #1
  - A structure (class) with a field of *m-1* keys and field of *m* pointers – in Python, there can be a single field where pointers and keys are exchanged
  - It is possible to add data for easier maintenance (e.g. the number of entered data in the node)

- Implementation #2
  - Each node is a doubly linked list
  - Each key has pointers to children – only the last key uses both pointers

B-stablo

- Implementacija #1
  - Struktura (klasa) s poljem od *m-1* ključeva i poljem od *m* pokazivača – u Pythonu može biti i jedno jedinstveno polje gdje se izmjenjuju pokazivači i ključevi
  - Moguće dodati podatke radi lakšeg održavanja (npr. broj upisanih podataka u čvoru)

- Implementacija #2
  - Svaki čvor je dvostruko povezana lista
  - Svaki ključ ima pokazivače na djecu – samo posljednji ključ koristi oba pokazivača

B-tree search algorithm

1. Enter the node and review the keys in turn as long as the current one is less than the requested one, and there are still unverified ones
   - The first node entered is the root

2. If the 1st step ended due to encountering a key larger than the required one or due to reaching the end of the node, go down to a lower level and repeat the first step

   - If there is no lower level, there is no required key

Algoritam pretraživanja B-stabla

1. Ući u čvor i redom pregledavati ključeve sve dok je trenutni manji od traženog, a još ima neprovjerenih
   - Prvi čvor u koji se ulazi je korijen

2. Ako je 1. korak završio zbog nailaska na ključ veći od traženog ili zbog dolaska do kraja čvora, spusti se na razinu niže i ponovi prvi korak
   - Ako nema niže razine, nema traženog ključa

B-tree search - implementation

```
function BTreeSearch(n, v_s)
    n_v ← starting value of the node n
    while value(n_v) < v_s and next(n_v) is not nil do
        n_v ← next(n_v)
    if value(n_v) = v_s then
        return n
    else if next(n_v) is nil and value(n_v) < v_s then
        if rightChild(n_v) is not nil then
            return BTreeSearch(rightChild(n_v), v_s)
        else
            return no searched key
    else
        if leftChild(n_v) is not nil then
            return BTreeSearch(leftChild(n_v), v_s)
        else
            return no searched key
```

- Remark
  - value(n_c) – value of key n_c– eg an integer
  - next(n_c) – the next key after $n_c$in the node key list – this depends on the node implementation
  - leftChild(n_c) and rightChild(n_c) – left and right child of key n_c

Pretraživanje B-stabla - implementacija

```
function BTreeSearch(n, v_s)
    n_v ← starting value of the node n
    while value(n_v) < v_s and next(n_v) is not nil do
        n_v ← next(n_v)
    if value(n_v) = v_s then
        return n
    else if next(n_v) is nil and value(n_v) < v_s then
        if rightChild(n_v) is not nil then
            return BTreeSearch(rightChild(n_v), v_s)
        else
            return no searched key
    else
        if leftChild(n_v) is not nil then
            return BTreeSearch(leftChild(n_v), v_s)
        else
            return no searched key
```

- Napomena
  - value(n_v) – vrijednost ključa n_v – npr. cijeli broj
  - next(n_v) – sljedeći ključ nakon n_v u listi ključeva čvora – ovo ovisi o implementaciji čvora
  - leftChild(n_v) i rightChild(n_v) – lijevo i desno dijete ključa n_v

Adding data to the B-tree

- It is simpler to build a B-tree**from the bottom up**

- Algorithm:

1. find the sheet in which the new data should be placed
2. if there is space, enter new data
3. if that sheet is full, "split" (*Split*) ga (create a new sheet, evenly distribute the elements between the two nodes, and write the central element in the parent)
4. if the parent is also full, "split" the parent as well (repeat the procedure from step 3)
5. if the root is also full, "split" it and make a new root

Dodavanje podataka u B-stablo

- Jednostavnije je graditi B-stablo **odozdo prema gore**

- Algoritam:
  1. pronaći list u koji bi trebalo smjestiti novi podatak
  2. ako ima mjesta, upisati novi podatak
  3. ako je taj list pun, “rascijepiti” (*split*) ga (napraviti novi list, ravnomjerno raspodijeliti elemente između dva čvora, a središnji element upisati u roditelja)
  4. ako je i roditelj pun, “rascijepiti” i roditelja (ponavljati proceduru iz koraka 3)
  5. ako je i korijen pun, “rascijepiti” ga i napraviti novi korijen

Adding data to the B-tree

- When inserting new data, 3 situations are possible:

1. the sheet where the new element should go is not full
    - insert a new element in that sheet in the appropriate place, moving the previous content if necessary
2. the leaf where the new element should go is full, but the root of the tree is not
    - the leaf is split (a new node is created) and all elements are distributed evenly, with the central element being written to the parent
3. the leaf where the new element should go is full, and so is the root of the tree
    - when the root is divided, two B-trees are created that need to be united

Dodavanje podataka u B-stablo

- Prilikom ubacivanja novog podatka moguće su 3 situacije:
  1. list u koji treba ići novi element nije pun
     - ubaciti novi element u taj list na odgovarajuće mjesto, pomičući po potrebi prethodni sadržaj
  2. list u koji treba ići novi element je pun, ali korijen stabla nije
     - list se dijeli (stvara se novi čvor) i svi elementi se ravnomjerno raspoređuju, s tim da se središnji element upisuje u roditelja
  3. list u koji treba ići novi element je pun, a isto tako i korijen stabla
     - kad se razdijeli korijen nastaju dva B-stabla koja treba sjediniti

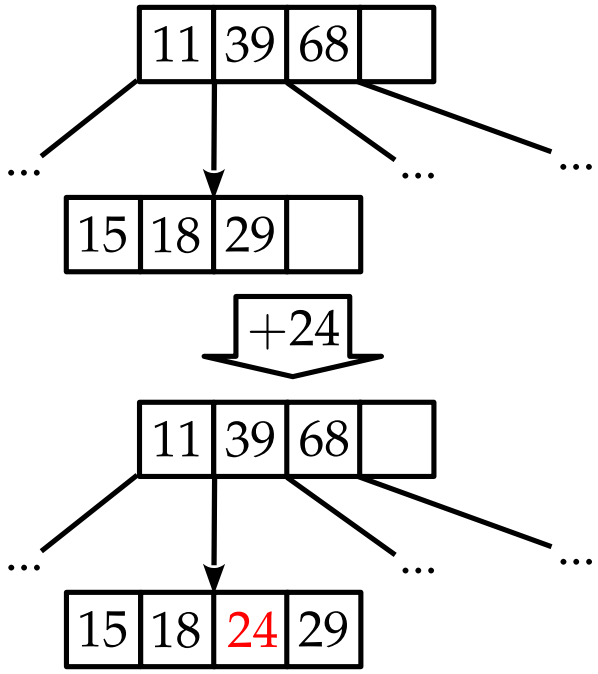
Adding data to the B-tree

- The union in the third case is achieved by creating another node that will be the new root and writing the central element in it
  - It's the only case that ends up raising the tree
  - **The B-tree is always perfectly balanced**

Dodavanje podataka u B-stablo

- Sjedinjenje u trećem slučaju se postiže stvaranjem još jednog čvora koji će biti novi korijen i upisivanjem središnjeg elementa u njega
  - To je jedini slučaj koji završava povisivanjem stabla
  - **B-stablo je uvijek savršeno uravnoteženo**

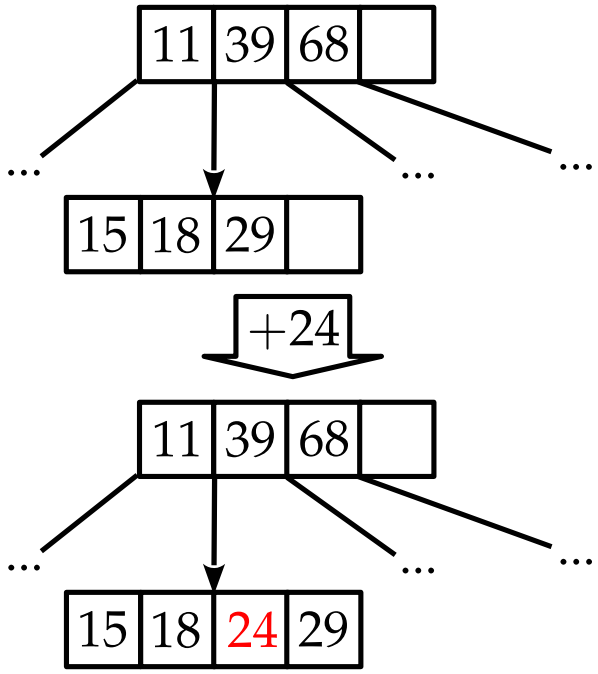
An example of adding data to a B-tree



- **Example 1**: we add 24 to the sheet in which there are less than -1keys. There is no need to restructure the B-tree.

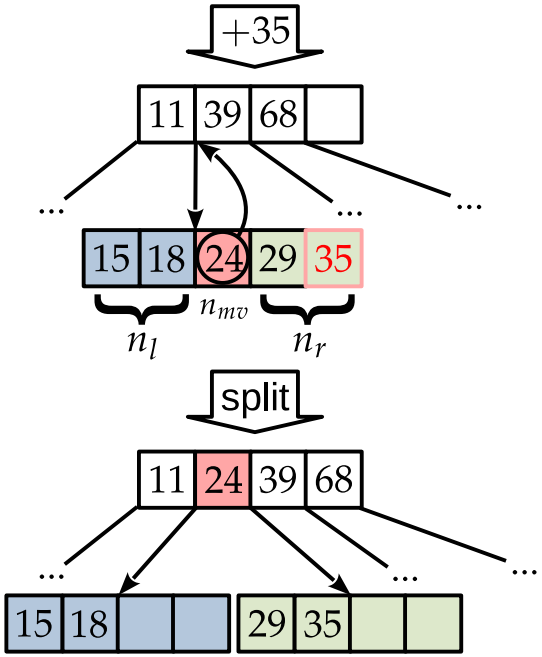
Max. degree = 5

Primjer dodavanja podataka u B-stablo



- **Primjer 1**: dodajemo 24 u list u kojem ima manje od $m - 1$ ključeva. Nema potrebe za restrukturiranjem B-stabla.

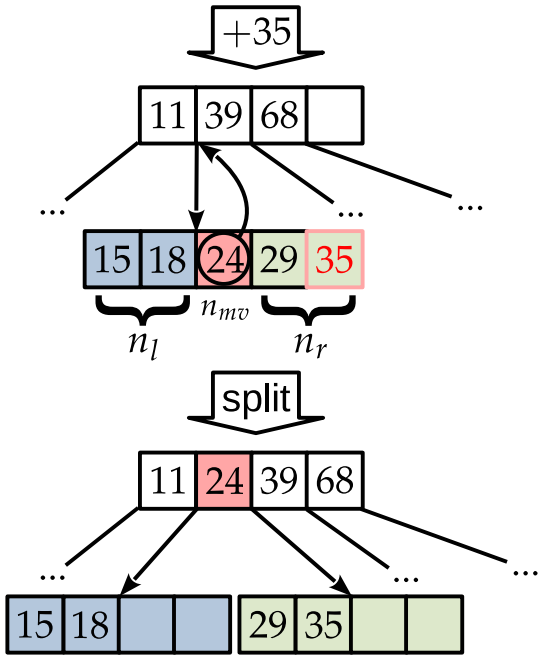
An example of adding data to a B-tree



- **Example 2**: we add 35 to the sheet where it is correct − 1keys and which has a parent node.
  - There is an overflow in the sheet.
  - The leaf is divided into a central key and two parts.
  - Insert the central key into the parent, and split the leaf into two nodes.

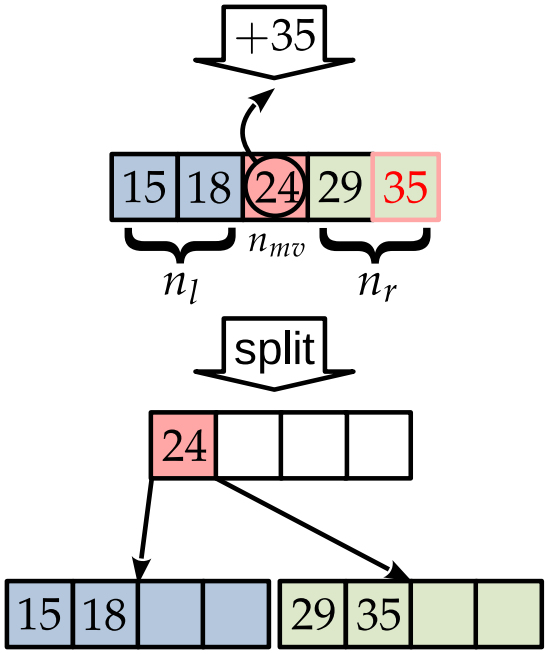
Max. degree = 5

Primjer dodavanja podataka u B-stablo



- **Primjer 2**: dodajemo 35 u list u kojem ima točno $m - 1$ ključeva i koji ima roditeljski čvor.
  - Dešava se preljev u listu.
  - List se dijeli na središnji ključ i dva dijela.
  - Središnji ključ ubacujemo u roditelja, a list razdvajamo na dva čvora.

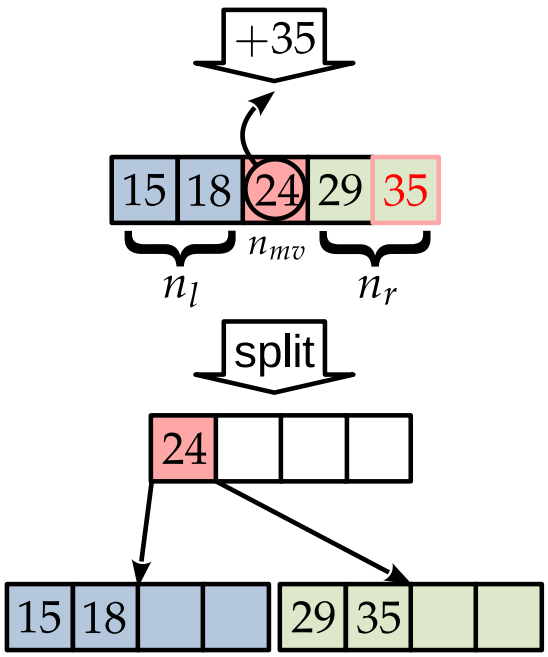
An example of adding data to a B-tree



- **Example 3**: we add 35 to the node where there is exactly − 1keys and which**there is none**the parent node (obviously the root node).
  - There is an overflow in the node.
  - The leaf is divided into a central key and two parts.
  - We use the central key to create a new root node, and the leaf we separate into two nodes.

Max. degree = 5

Primjer dodavanja podataka u B-stablo

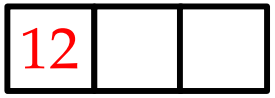


- **Primjer 3**: dodajemo 35 u čvor u kojem ima točno $m - 1$ ključeva i koji **nema** roditeljski čvor (očito je korijenski čvor).
  - Dešava se preljev u čvoru.
  - List se dijeli na središnji ključ i dva dijela.
  - Središnji ključ koristimo za stvaranje novog korijenskog čvora, a list razdvajamo na dva čvora.

Max. degree = 5

An example of adding data to a B-tree

- B-tree of row 4 – 4 hands, 3 keys
- We add the keys in order:
  12,75,34,62,19,25,66,30,33,71,47,21,15, 23,27

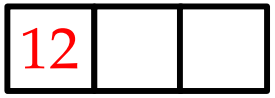


- **Step 1**: We form the root node with the first key**12**

- **Step 2**: We add keys to the node until we can - we add**75**and**34**

Primjer dodavanja podataka u B-stablo

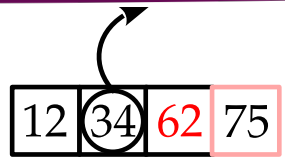
- B-stablo reda 4 – 4 kazaljke, 3 ključa
- Dodajemo redom ključeve: **12,75,34,62,19,25,66,30,33,71,47,21,15, 23,27**

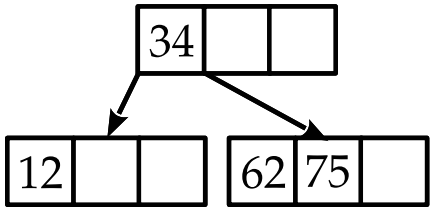


- **Korak 1**: Formiramo korijenski čvor s prvim ključem **12**

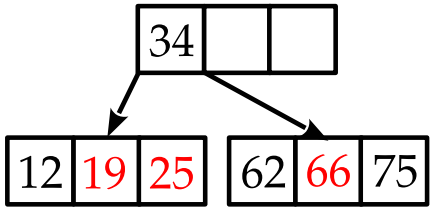
- **Korak 2**: U čvor dodajemo ključeve do dok možemo – dodajemo **75** i **34**

An example of adding data to a B-tree





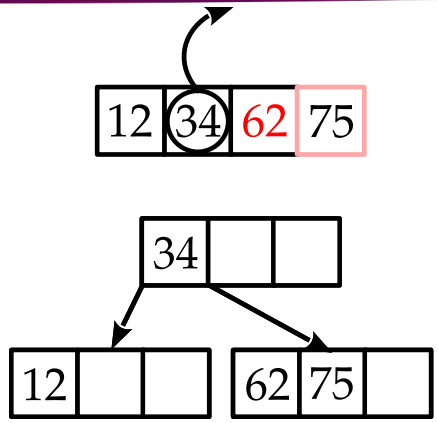
- **Step 3**: By adding a key**62**, there is an overflow in the root node, which we separate with the creation of a new one root node.



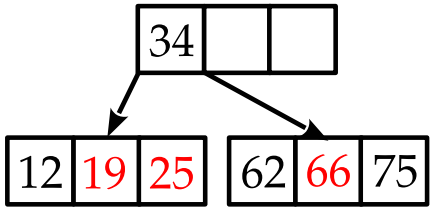
- **Step 4**: We are adding keys**19**,**25**and**66** directly into the leaves.

12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

Primjer dodavanja podataka u B-stablo



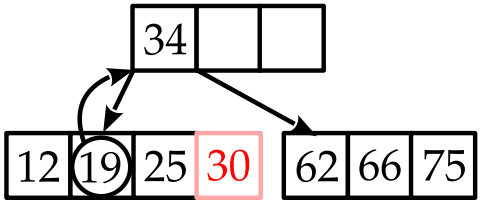
- **Korak 3**: Dodavanjem ključa **62**, dolazi do preljeva u korijenskom čvoru, kojeg razdvajamo uz stvaranje novog korijenskog čvora.

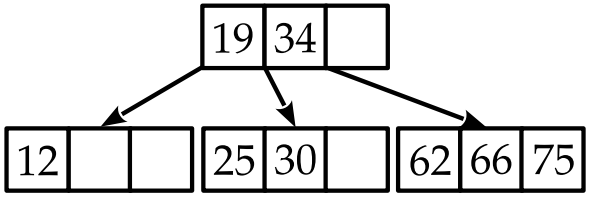


- **Korak 4**: Dodajemo ključeve **19**, **25** i **66** direktno u listove.

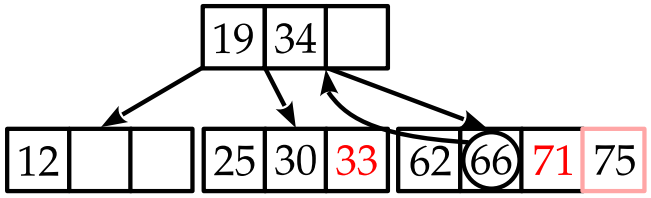
12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

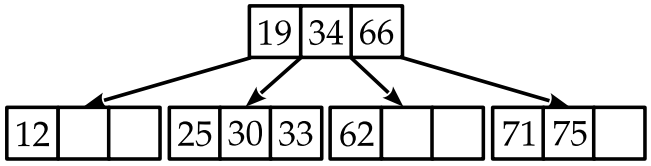
An example of adding data to a B-tree





- **Step 5**: By adding a key**30**, there is an overflow in the left leaf, which we separate by inserting the middle key into the root node.

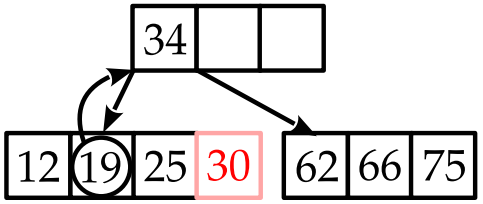


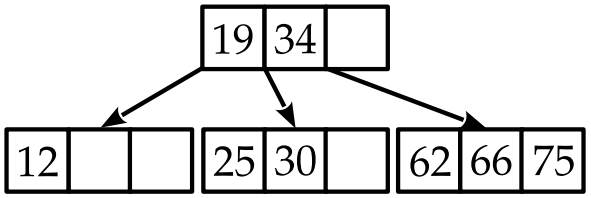


- **Step 6**: We add the key**33**, and then**71**. By adding**71**we cause an overflow in the right leaf, which we separate by inserting the middle key into the root node.

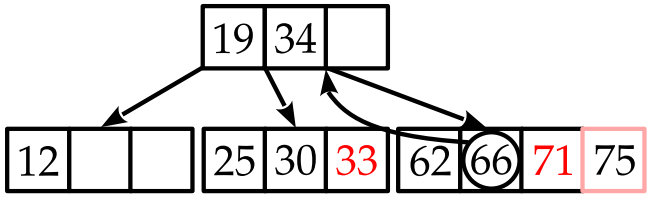
12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

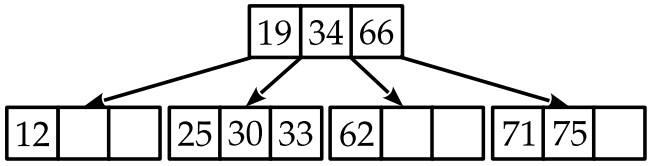
Primjer dodavanja podataka u B-stablo





- **Korak 5**: Dodavanjem ključa **30**, dolazi do preljeva u lijevom listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.

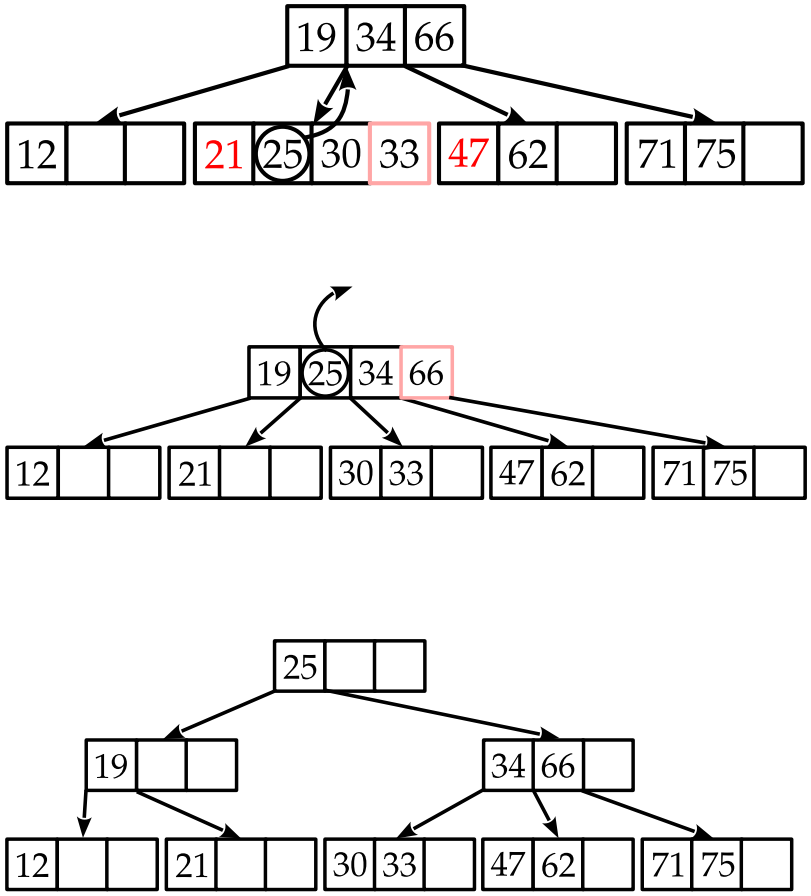




- **Korak 6**: Dodajemo ključ **33**, a zatim **71**. Dodavanjem **71** izazivamo preljev u desnom listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.

12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

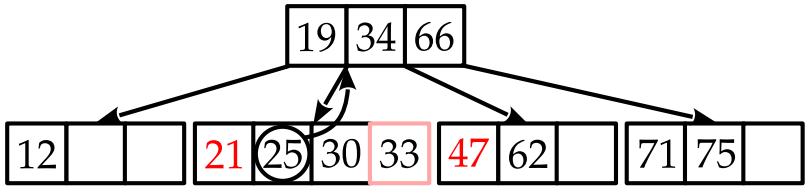
An example of adding data to a B-tree

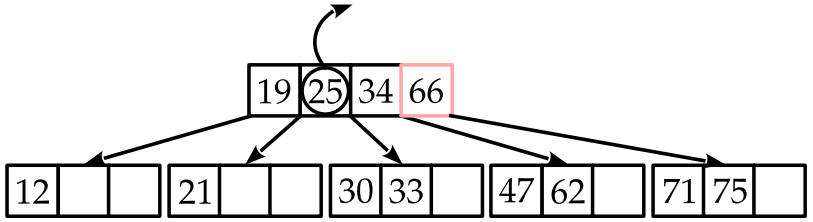


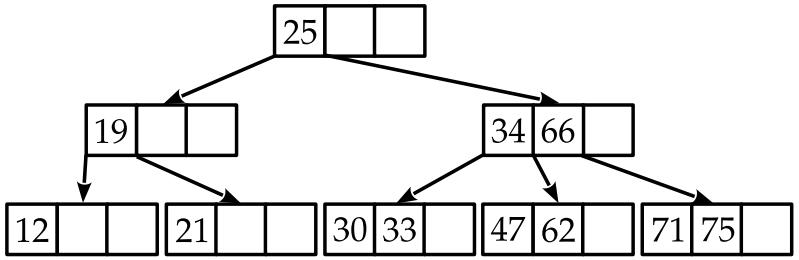
- **Step 7**: We add the key**47**, and then the key**21**.
  - By adding**21**we cause an overflow in the leaf, which we separate by inserting the middle key into the root node.
  - By inserting 25 into the root node, we cause an overflow of the root node, which we separate while creating a new root node.

12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

Primjer dodavanja podataka u B-stablo





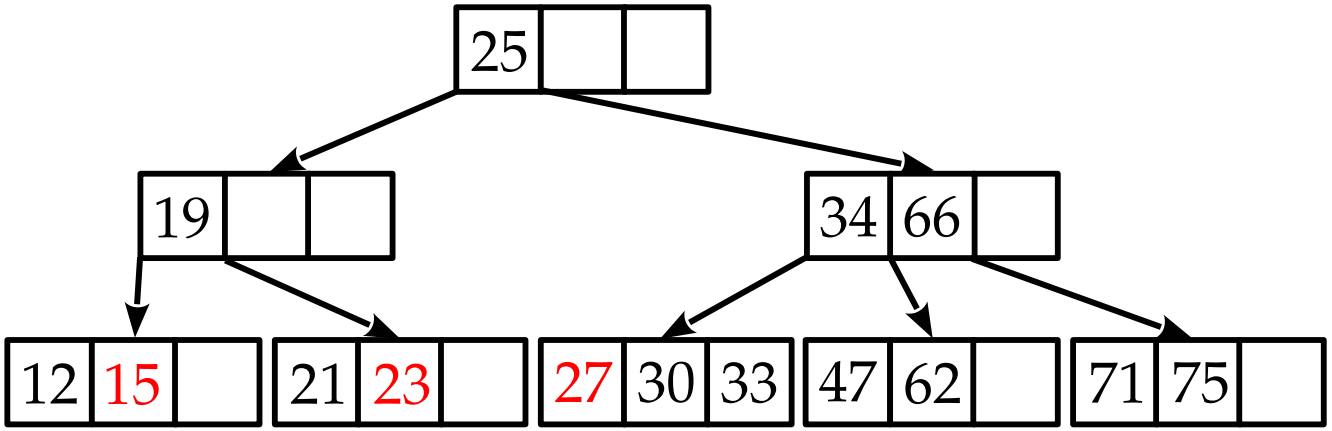


- **Korak 7**: Dodajemo ključ **47**, a zatim ključ **21**.
  - Dodavanjem **21** izazivamo preljev u listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.
  - Ubacivanjem 25 u korijenski čvor izazivamo preljev korijenskog čvora, kojeg razdvajamo uz stvaranje novog korijenskog čvora.

12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

An example of adding data to a B-tree

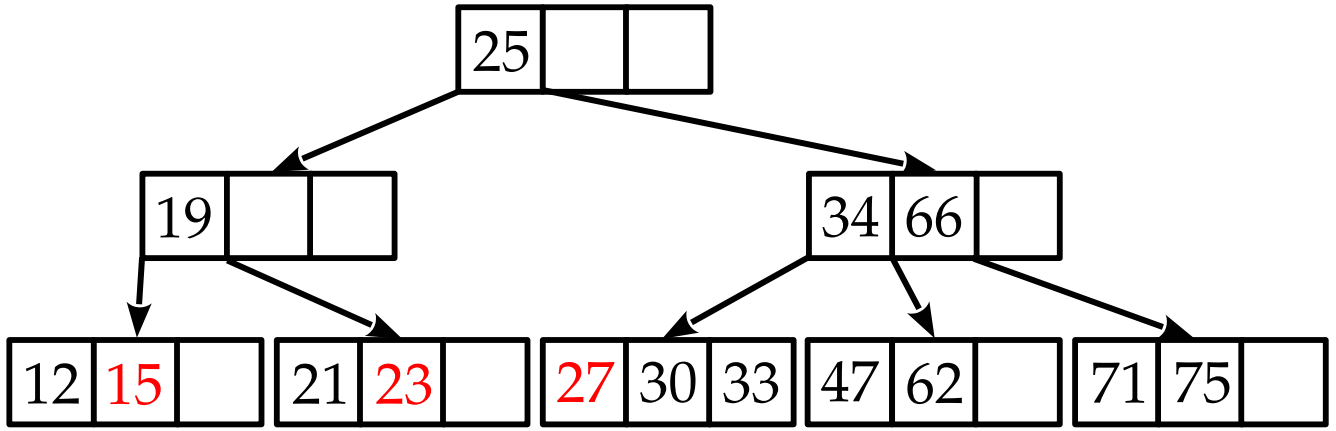
- **Step 8**: We are adding keys**15**,**23**and**27**directly into Bstable sheets without restructuring.



12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

Primjer dodavanja podataka u B-stablo

- **Korak 8**: Dodajemo ključeve **15**, **23** i **27** direktno u listove B-stabla bez restrukturiranja.

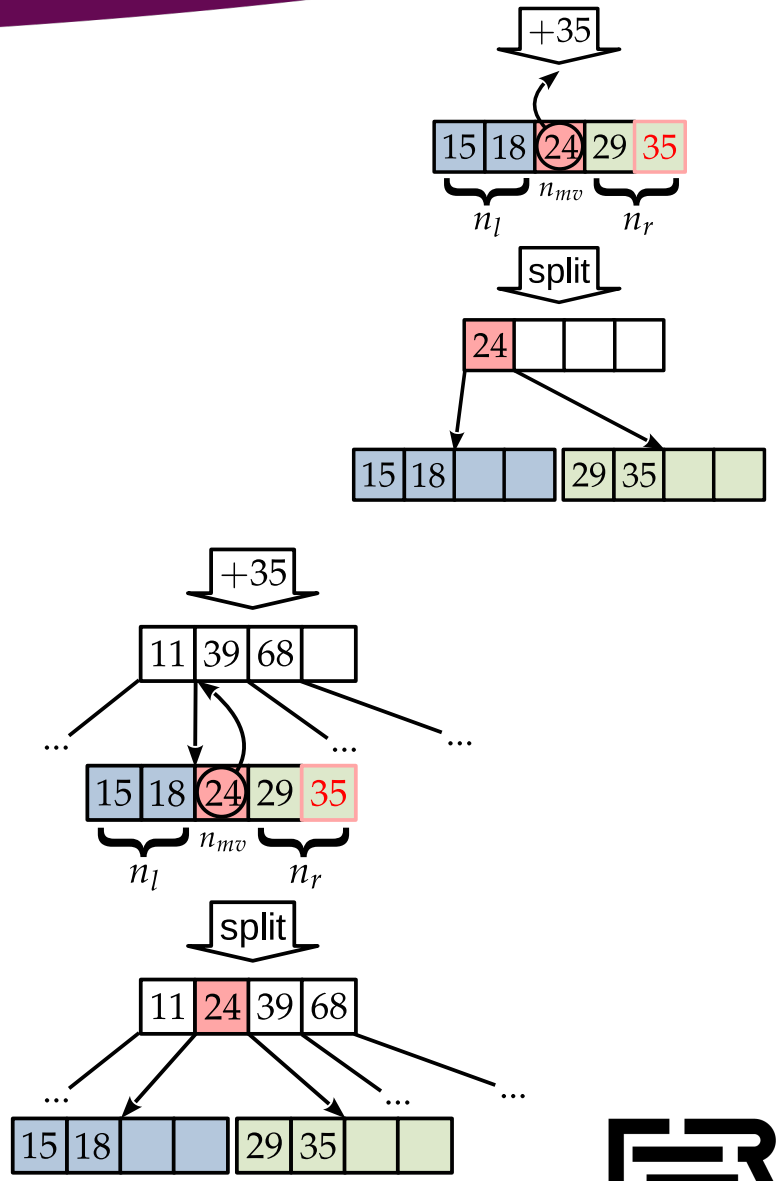


12,75,34,62,19,25,66,30,33,71,47,21,15,23,27

Implementation of B-tree addition

procedure BTreeSplit($btree, n$)
  if $|n| == m$ **then**
    $n_l \leftarrow$ new node having first $\lceil m/2 \rceil - 1$ values in the node n
    $n_{mv} \leftarrow$ the next value in the node n
    $n_r \leftarrow$ new node having the rest of values from the node n
    $n_{last} \leftarrow$ the last value in n_l
    $rightChild(n_{last}) \leftarrow leftChild(n_m)$
    if $parent(n)$ is nil **then**
      $n_{root} \leftarrow$ new node
      $n_{mv}' \leftarrow$ insert $value(n_{mv})$ into the node n_{root}
      $root$ of $btree \leftarrow n_{root}$
      $leftChild(n_{mv}') \leftarrow n_l$
      $rightChild(n_{mv}') \leftarrow n_r$
    else
      $n_{mv}' \leftarrow$ insert $value(n_{mv})$ into the parent node of n
      $leftChild(n_{mv}') \leftarrow n_l$
      $rightChild(n_{mv}') \leftarrow n_r$
      BTreeSplit($btree$,parent of n)

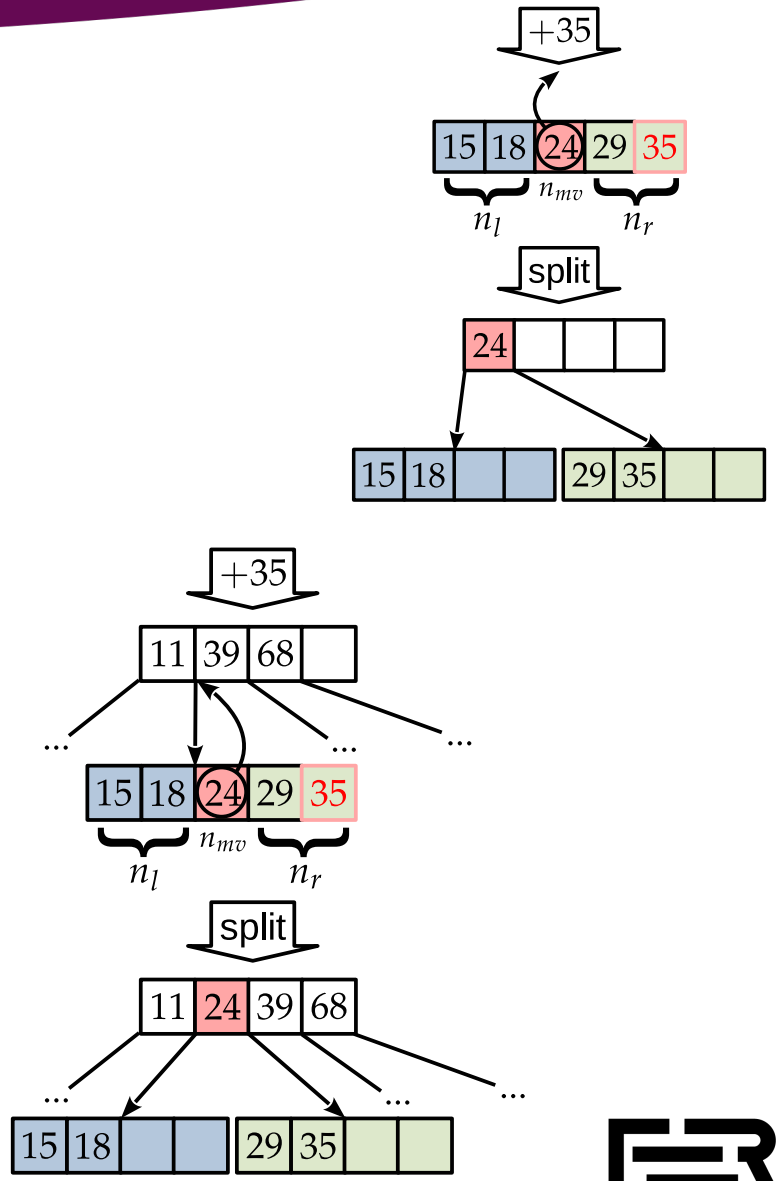
procedure BTreeInsert($btree, v_n$)
  $(n, n_v) \leftarrow$BTreeSearch($root$ of $btree, v_n$)
  insert v_n into the node n
  BTreeSplit($btree, n$)



Implementacija dodavanja u B-stablo

procedure BTreeSplit($btree, n$)
  if $|n| == m$ **then**
    $n_l \leftarrow$ new node having first $\lceil m/2 \rceil - 1$ values in the node n
    $n_{mv} \leftarrow$ the next value in the node n
    $n_r \leftarrow$ new node having the rest of values from the node n
    $n_{last} \leftarrow$ the last value in n_l
    $rightChild(n_{last}) \leftarrow leftChild(n_m)$
    if $parent(n)$ is nil **then**
      $n_{root} \leftarrow$ new node
      $n_{mv}' \leftarrow$ insert $value(n_{mv})$ into the node n_{root}
      $root$ of $btree \leftarrow n_{root}$
      $leftChild(n_{mv}') \leftarrow n_l$
      $rightChild(n_{mv}') \leftarrow n_r$
    else
      $n_{mv}' \leftarrow$ insert $value(n_{mv})$ into the parent node of n
      $leftChild(n_{mv}') \leftarrow n_l$
      $rightChild(n_{mv}') \leftarrow n_r$
      BTreeSplit($btree$,parent of n)

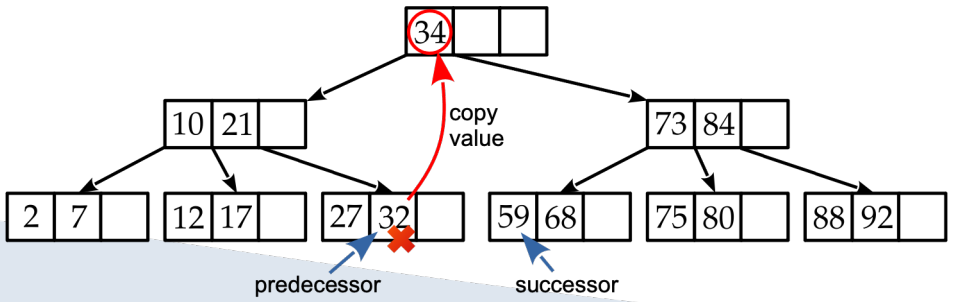
procedure BTreeInsert($btree, v_n$)
  $(n, n_v) \leftarrow$BTreeSearch($root$ of $btree, v_n$)
  insert v_n into the node n
  BTreeSplit($btree, n$)

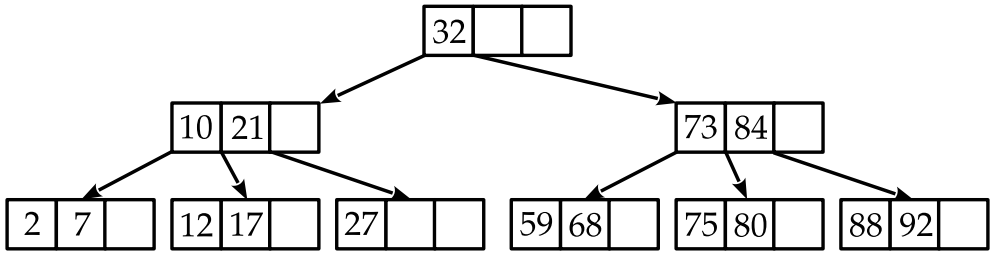


Deleting data in the B-tree

- 2 cases:

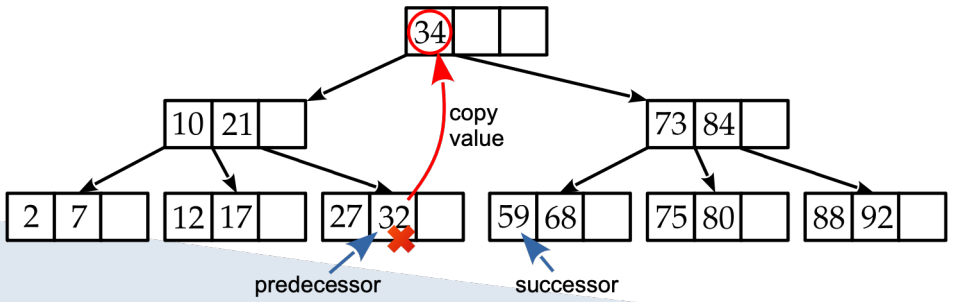
  1. deleting an element in the tree leaf
  2. deleting an element in a node
     - it boils down to deleting an element from the list
       - in the place of the element to be deleted, its immediate predecessor (which can only be in the list) is written, then the overwritten element is deleted from the list using the standard procedure for deleting a list

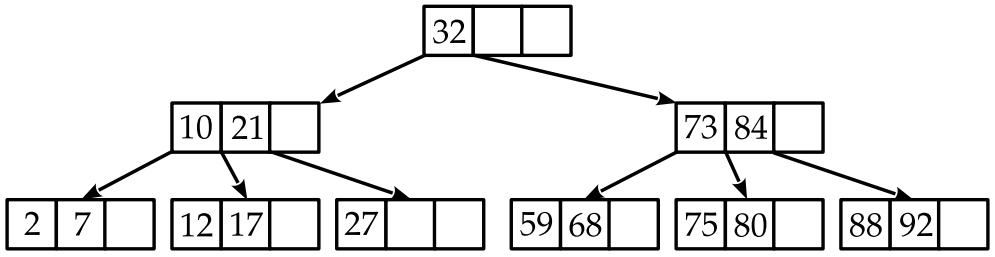




Brisanje podataka u B-stablu

- 2 slučaja:
  1. brisanje elementa u listu stabla
  2. brisanje elementa u čvoru
     - svodi se na brisanje elementa iz lista
       - na mjesto elementa koji treba izbrisati upisuje se njegov neposredni prethodnik (koji može biti samo u listu), potom se u listu briše prepisani element standardnim postupkom za brisanje lista





Deleting data in the B-tree

1. leaf even after deleting the element has ≥⌈*m/2*⌉-*1*keys;**END**
2. number of remaining elements (keys) <⌈*m/2*⌉-*1*
    1. if there is a neighbor with > on the left or right⌈*m/2*⌉-*1*keys
        - distribute leaf elements, neighbor elements, and the central element from the parent evenly into the leaf and neighbor, and write the central element of the united set (union) of elements as a new central element in the parent;**END**

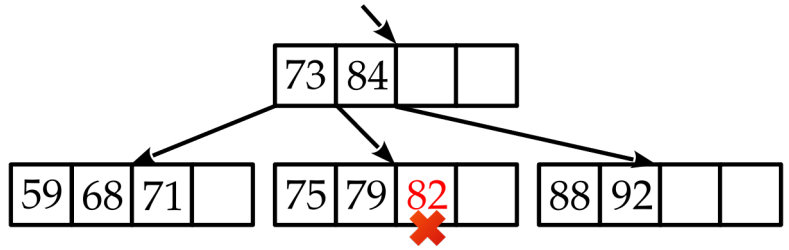
    2. the leaf and neighbor are merged (all elements of the leaf and neighbor + the central element from the parent are written into the leaf, and the neighbor is deleted);**CONTINUE with parent**
    3. by procedure 2.2 we get to the root:
        - if root has more than 1 element: merge current node and neighbor as in 2.2;**END**
        - otherwise: all leaf, neighbor and root elements are written into 1 node which becomes the new root, and 2 nodes are deleted from the tree;**END**

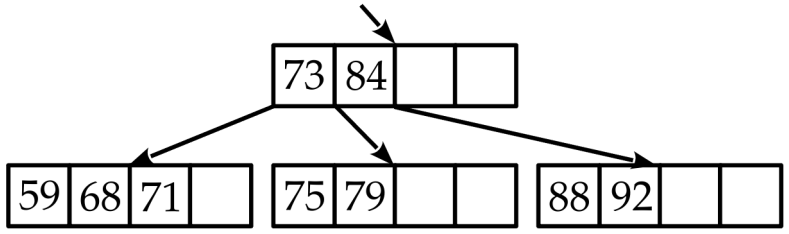
Brisanje podataka u B-stablu

1. list i nakon brisanja elementa ima ≥ [*m/2*] *-1* ključeva; **KRAJ**
2. broj preostalih elemenata(ključeva) < [*m/2*] *-1*
   1. ako lijevo ili desno postoji susjed s > [*m/2*] *-1* ključeva
      - elemente lista, elemente susjeda i središnji element iz roditelja ravnomjerno rasporediti u list i susjeda, a kao novi središnji element u roditelja upisati središnji element ujedinjenog skupa (unije) elemenata; **KRAJ**
   2. list i susjed se sjedinjuju (svi elementi lista i susjeda + središnji element iz roditelja se upisuju u list, a susjed se briše); **NASTAVITI s roditeljem**
   3. postupkom 2.2 dolazimo do korijena:
      - ako korijen ima više od 1 elementa: sjediniti trenutni čvor i susjeda kao u 2.2; **KRAJ**
      - inače: sve elemente lista, susjeda i korijena upisati u 1 čvor koji postaje novi korijen, a 2 čvora se brišu iz stabla; **KRAJ**

An example of deleting data from the B-tree

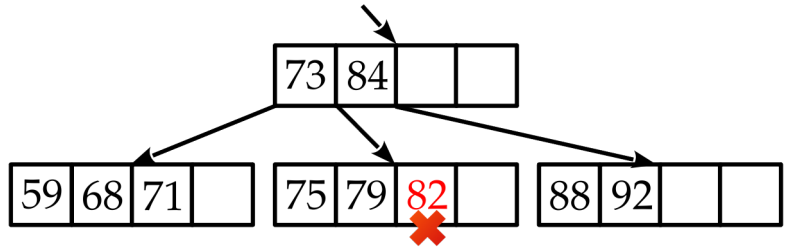
- **Example 1**: we delete the key**82**from the sheet. After deletion, the sheet is still filled≥50% because there is≥ ⌈m/2⌉-1keys. There is no need to restructure the B-tree.

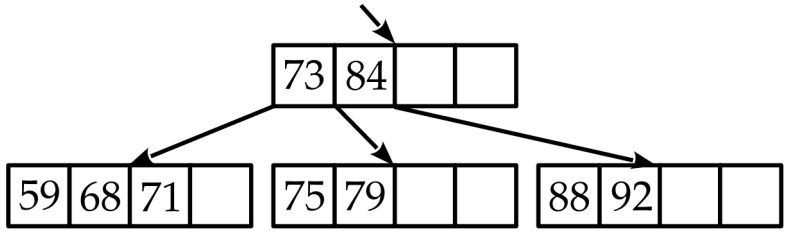




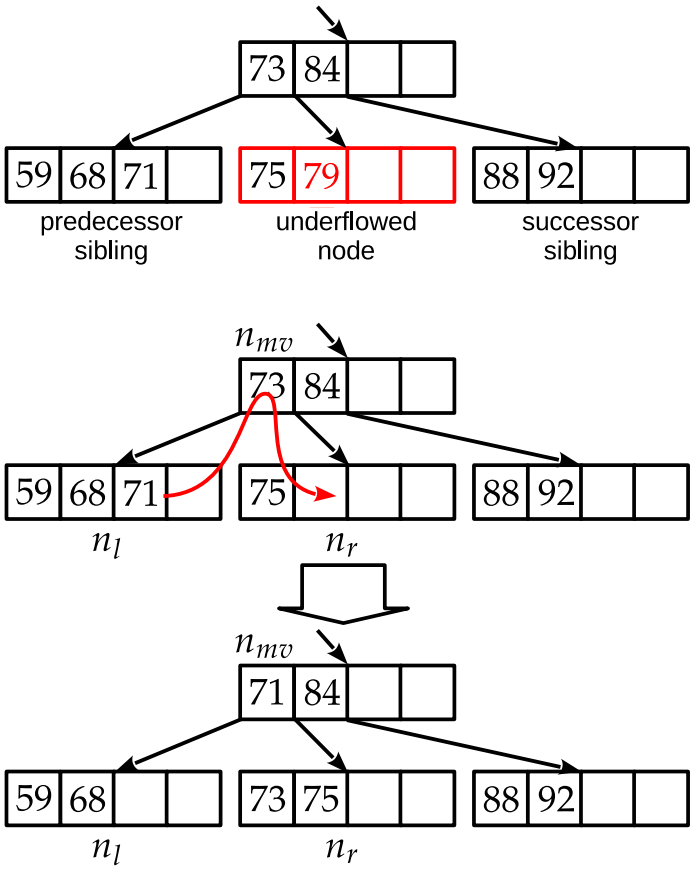
Primjer brisanja podataka iz B-stabla

- **Primjer 1**: brišemo ključ **82** iz lista. List je nakon brisanja još uvijek popunjen ≥50% zato jer ima ≥ [m/2]-1 ključeva. Nema potrebe za restrukturiranjem B-stabla.





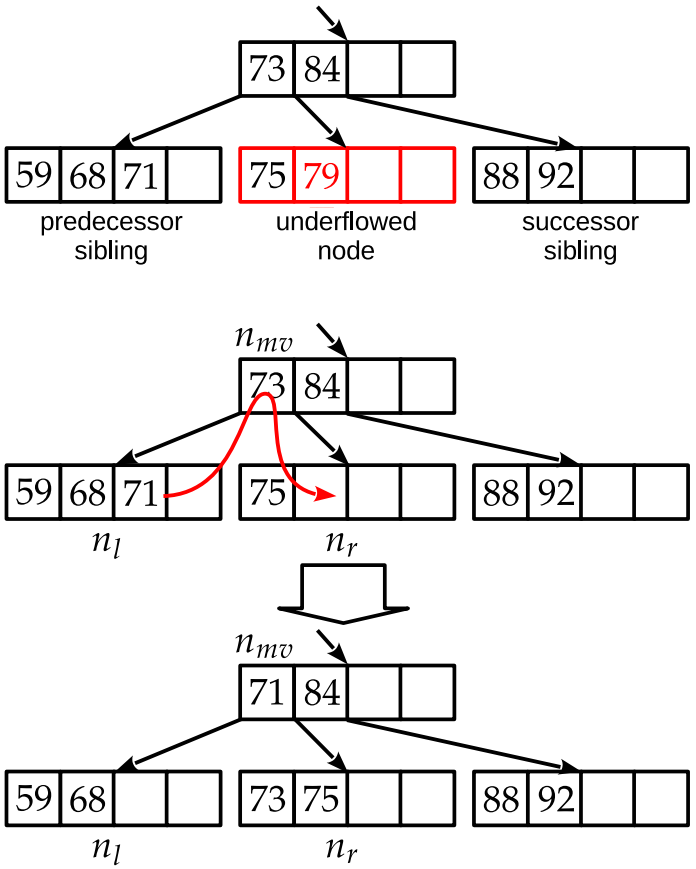
An example of deleting data from the B-tree



- **Example 2**: we delete the key**79**from the sheet.

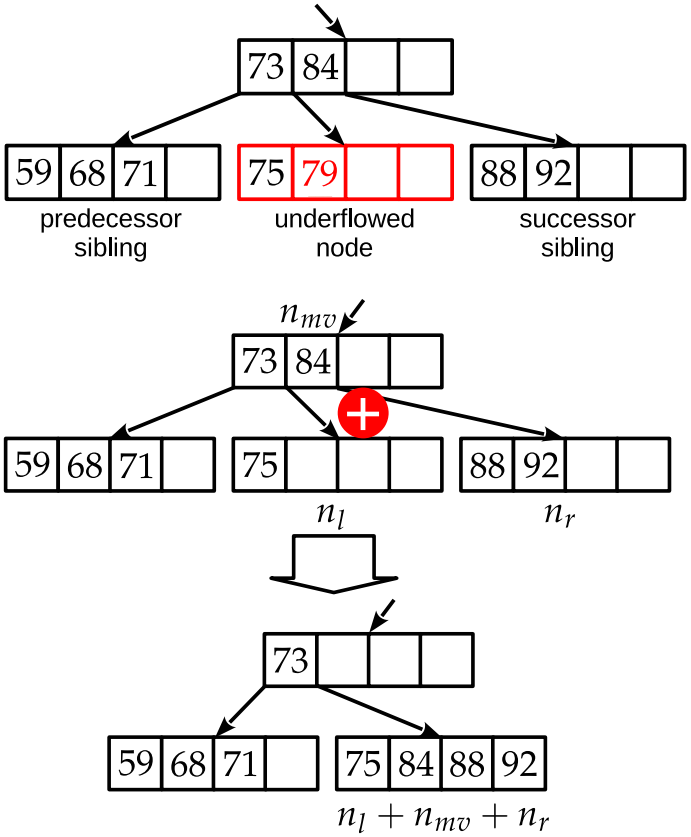
  After deletion, the sheet is no longer filled ≥50% because it has <⌈m/2⌉-1 keys.
- If we look at the left neighbor (twin), we see that it is filled >⌈m/2⌉-1
  - We do the restructuring by transferring the keys from the left neighbor to the node that is in the subflow
  - In doing so, we pay attention to the shared key in the parent node

Primjer brisanja podataka iz B-stabla



- **Primjer 2**: brišemo ključ **79** iz lista. List nakon brisanja nije više popunjen $\geq$50% zato jer ima $<$ [m/2]-1 ključeva.
- Gledamo li lijevog susjeda (blizanca), vidimo da je on popunjen $>$ [m/2]-1
  - Radimo restrukturiranje tako da prebacujemo ključeve iz lijevog susjeda u čvor koji je u podljevu
  - Pri tome pazimo na zajednički ključ u čvoru roditelja

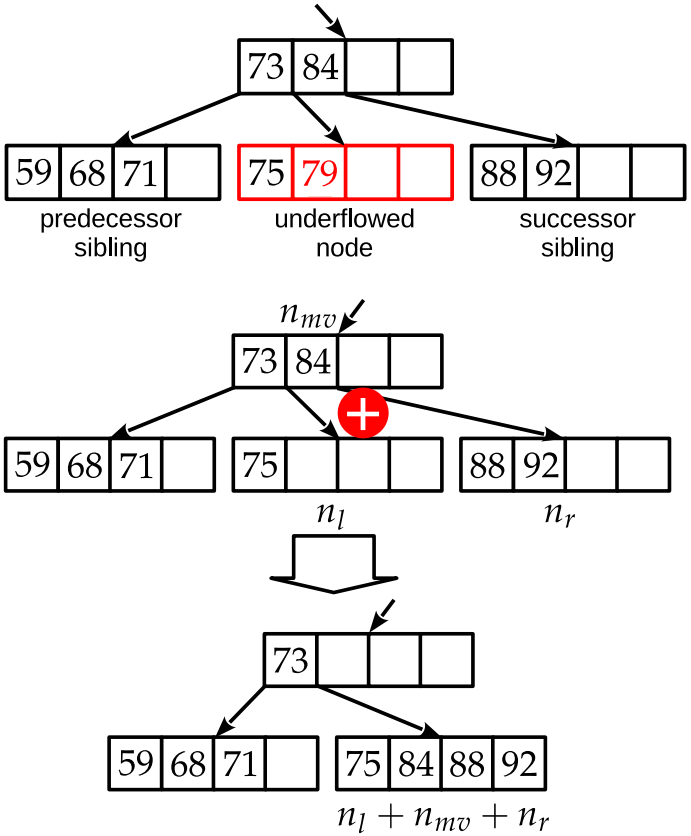
An example of deleting data from the B-tree



- **Example 3**: we delete the key**79**from the sheet.

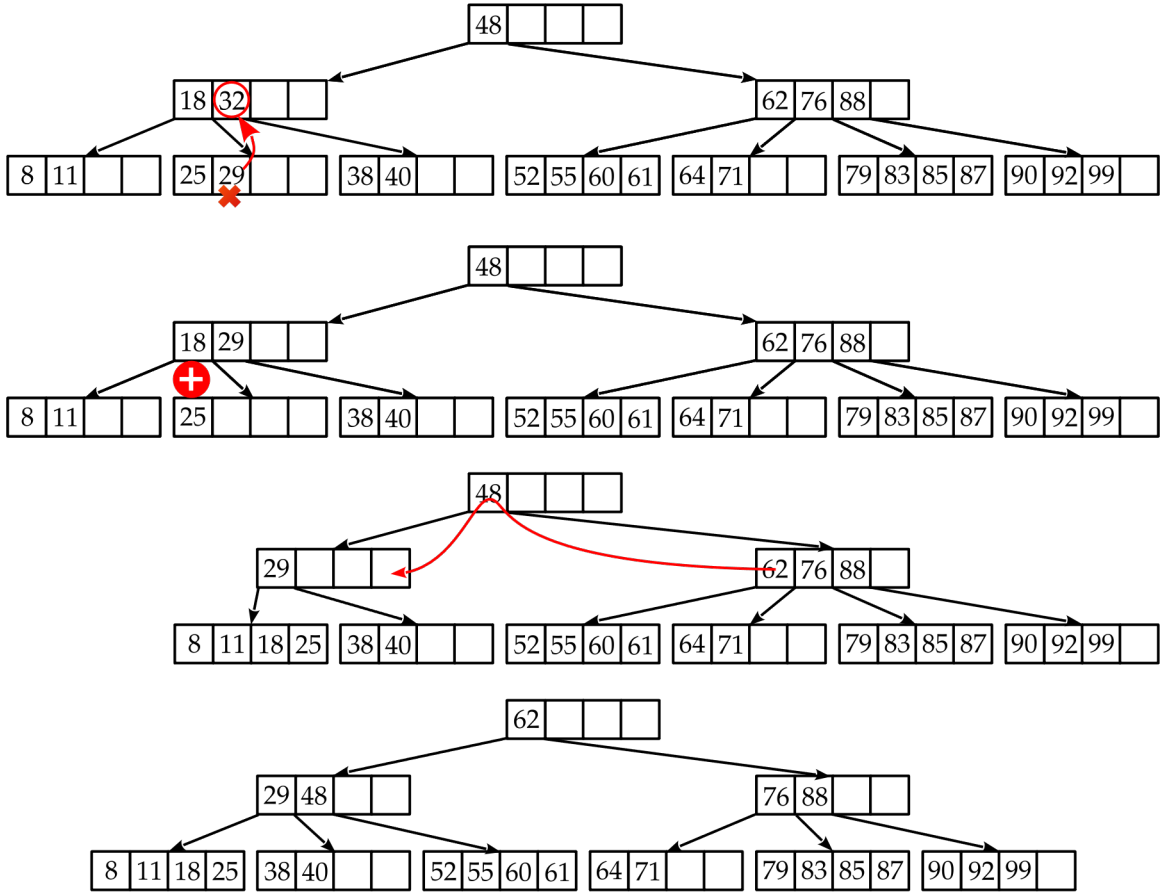
  After deletion, the sheet is no longer filled ≥50% because it has <⌈m/2⌉-1 keys.
- If we look at the right neighbor (twin), we see that it is filled with =⌈m/2⌉-1
  - We connect the right neighbor and the node that is in the underflow (*underflow*)
- Notice that now the parent node is in the subflow, which we have to solve recursively

Primjer brisanja podataka iz B-stabla



- **Primjer 3**: brišemo ključ **79** iz lista. List nakon brisanja nije više popunjen ≥50% zato jer ima < [m/2]-1 ključeva.
- Gledamo li desnog susjeda (blizanca), vidimo da je on popunjen = [m/2]-1
  - Radimo spajanje desnog susjeda i čvora koji je u podljevu (*underflow*)
- Primijetimo da je sada roditeljski čvor u podljevu, što moramo riješiti rekurzivno

An example of deleting data from the B-tree

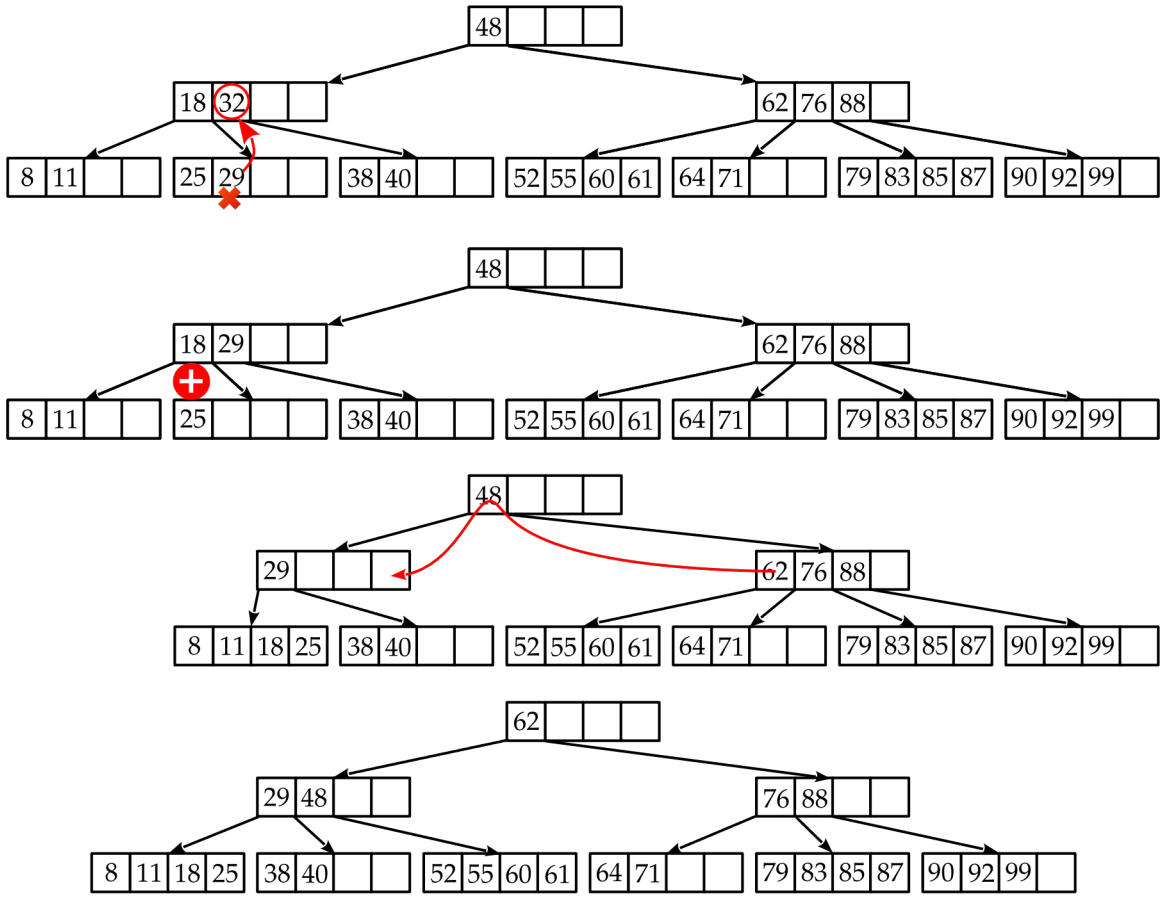


- We delete the keys from the initial B-tree

  32, 76, 48, 25 and 11

- **Step 1**: we delete the key **32** by copying process
  - We connect the leaf in the base with the left neighbor
  - This causes the underflow of the internal node
  - We restructure the right neighbor and the internal node in the subflow

Primjer brisanja podataka iz B-stabla

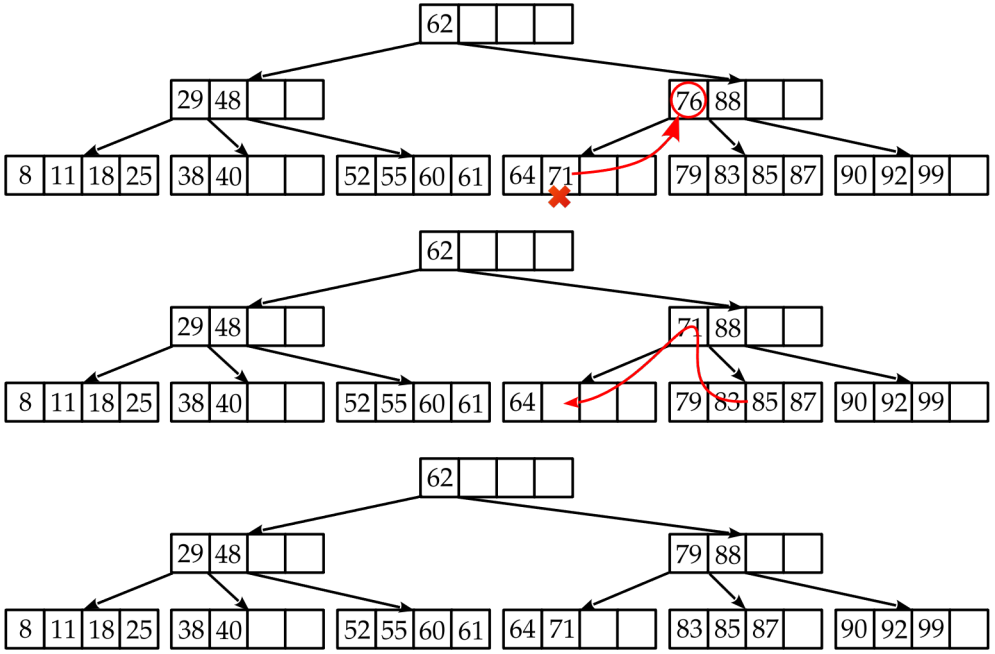


- Iz početnog B-stabla brišemo ključeve

  32, 76, 48, 25 i 11

- **Korak 1**: brišemo ključ **32** postupkom kopiranja
  - List u podljevu spajamo s lijevim susjedom
  - To uzrokuje podljev unutarnjeg čvora
  - Restrukturiramo desnog susjeda i unutarnji čvor u podljevu

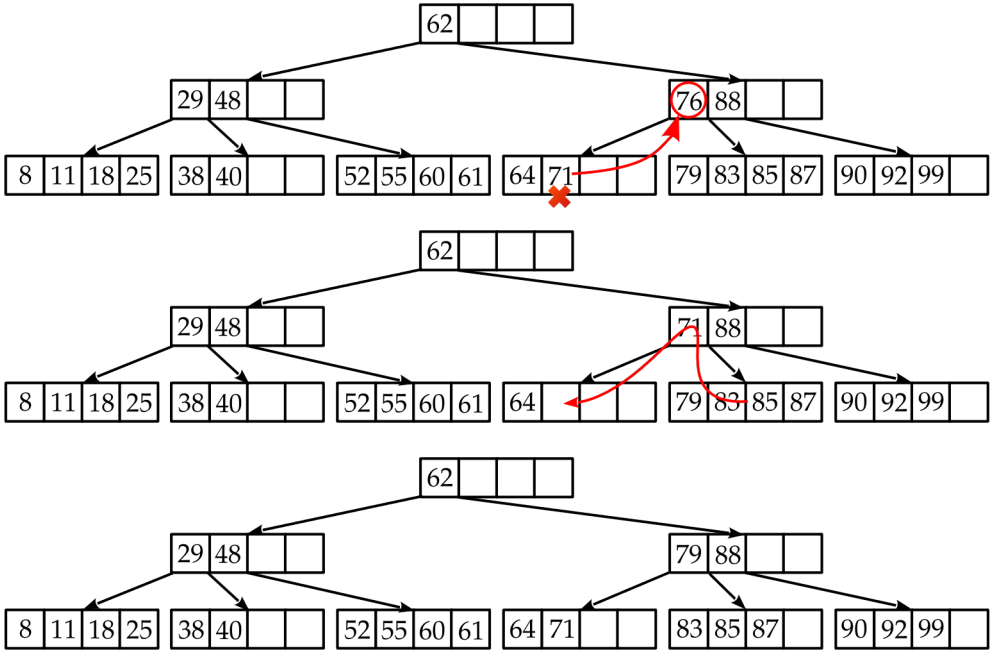
An example of deleting data from the B-tree



- **Step 2**: we delete the key**76**by copying process. The sheet where we deleted the replacement key is in the footer.
  - The right neighbor to the node in the underflow has 100% occupancy
  - We restructure the right neighbor and the node in the subflow

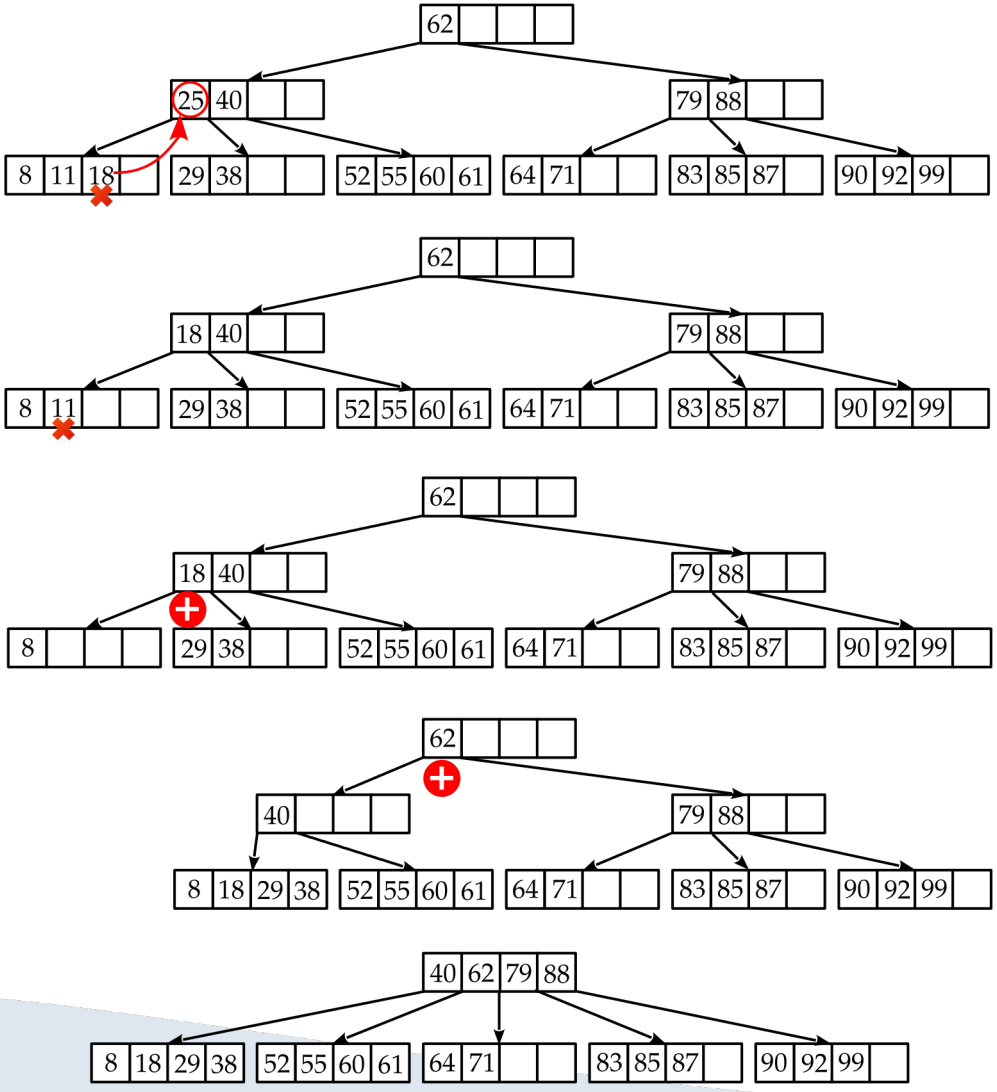
32, 76, 48, 25 and 11

Primjer brisanja podataka iz B-stabla



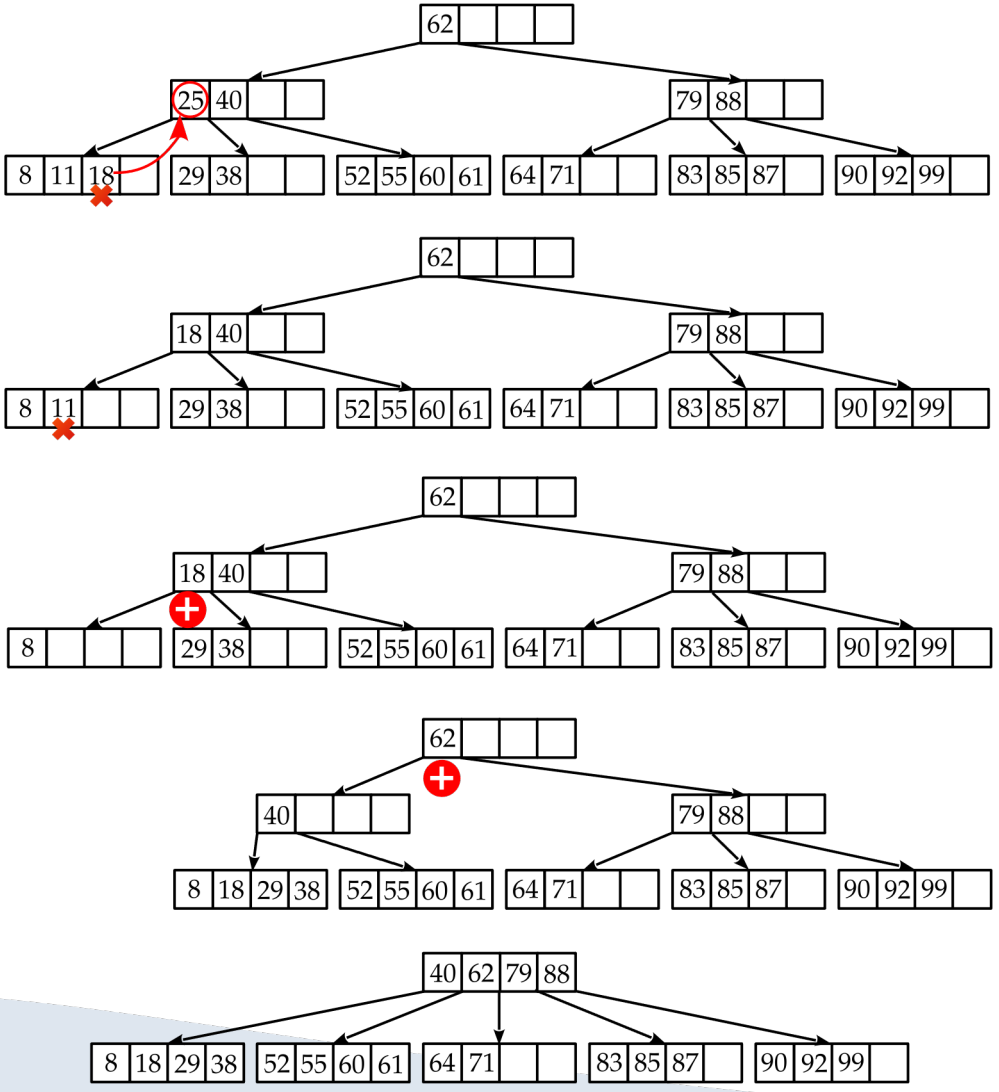
- **Korak 2**: brišemo ključ **76** postupkom kopiranja. List u kojem smo obrisali zamjenski ključ je u podljevu.
  - Desni susjed do čvora u podljevu ima 100% popunjenost
  - Restrukturiramo desnog susjeda i čvor u podljevu

An example of deleting data from the B-tree



- **Step 3**: we delete the key**25**by the copying process, and then we delete the key**11**. The sheet where we deleted 18 and 11 is now in the underfill.
  - The right neighbor to the node in the underflow has exactly 50% occupancy, so we connect the node in the underflow to the right neighbor
  - Now the internal node is in the underflow, and its right neighbor has exactly 50% occupancy, so we connect the node in the underflow with the right neighbor
  - With the previous merge, the old root node disappears, and the newly merged node becomes the root node. Tree depth is reduced by 1.

Primjer brisanja podataka iz B-stabla



- **Korak 3**: brišemo ključ **25** postupkom kopiranja, a zatim brišemo ključ **11**. List u kojem smo obrisali 18 i 11 je sada u podljevu.
  - Desni susjed do čvora u podljevu ima točno 50% popunjenost, te čvor u podljevu spajamo s desnim susjedom
  - Sada je unutarnji čvor u podljevu, a njegov desni susjed ima točno 50% popunjenost, te čvor u podljevu spajamo s desnim susjedom
  - Prethodnim spajanjem nestaje stari korijenski čvor, a novi spojeni čvor postaje korijenski. Dubina stabla smanjuje se za 1.

Implementation of element deletion in B-tree

```
procedure BTreeRemoval(btree, val)
    (n_rem, n_v) ← BTreeSearch(root of btree, val)
    if n_v is not nil then
        remove value val from the node n_rem
        BTreeRemovalConsolidation(btree, n_rem)

procedure BTreeRemovalConsolidation(btree, n)
    if |n| < ⌈degree of btree/2⌉ − 1 then
        ps ← the predecessor sibling
        n_root ← nil
        if ps is not nil then
            n_mv ← the shared parent value between ps and n
            if |ps| > ⌈degree of btree/2⌉ − 1 then
                BTreeRedistribute(btree, ps, n_mv, n)
            else
                n_root ← BTreeMerge(btree, ps, n_mv, n)
        else
            ss ← the successor sibling
            n_mv ← the shared parent value between ss and n
            if |ss| > ⌈degree of btree/2⌉ − 1 then
                BTreeRedistribute(btree, n, n_mv, ss)
            else
                n_root ← BTreeMerge(btree, n, n_mv, ss)
        if n_root is nil and parent of n exists then
            BTreeRemovalConsolidation(btree, parent of n)
```

Implementacija brisanja elementa u B-stablu

```
procedure BTreeRemoval(btree, val)
    (n_rem, n_v) ← BTreeSearch(root of btree, val)
    if n_v is not nil then
        remove value val from the node n_rem
        BTreeRemovalConsolidation(btree, n_rem)

procedure BTreeRemovalConsolidation(btree, n)
    if |n| < ⌈degree of btree/2⌉ − 1 then
        ps ← the predecessor sibling
        n_root ← nil
        if ps is not nil then
            n_mv ← the shared parent value between ps and n
            if |ps| > ⌈degree of btree/2⌉ − 1 then
                BTreeRedistribute(btree, ps, n_mv, n)
            else
                n_root ← BTreeMerge(btree, ps, n_mv, n)
        else
            ss ← the successor sibling
            n_mv ← the shared parent value between ss and n
            if |ss| > ⌈degree of btree/2⌉ − 1 then
                BTreeRedistribute(btree, n, n_mv, ss)
            else
                n_root ← BTreeMerge(btree, n, n_mv, ss)
        if n_root is nil and parent of n exists then
            BTreeRemovalConsolidation(btree, parent of n)
```

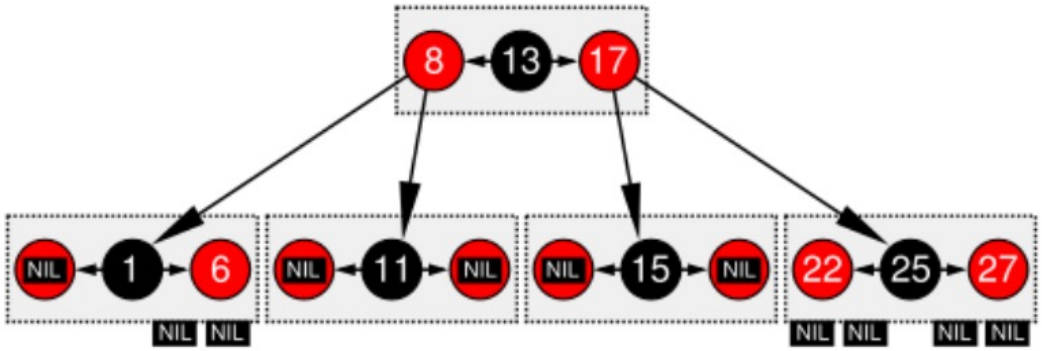
Red and black trees

Crveno-crna stabla

Red-black tree (*Red-black tree*)

- A binary tree that**conceptually**arises from a B-tree of order 4 if its node elements are considered colored according to strict rules

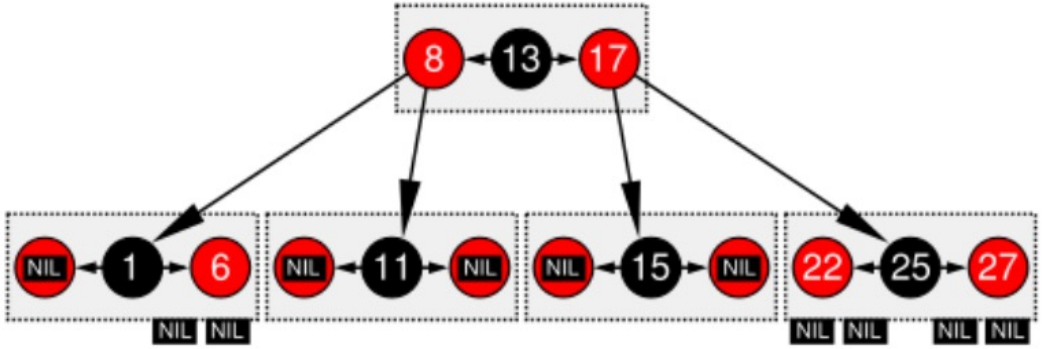
- Comparison with B-tree:
  - lower memory consumption
  - maintains balance
  - the complexity is the same



Crveno-crno stablo (*Red-black tree*)

- Binarno stablo koje **idejno** proizlazi iz B-stabla 4. reda ako mu se elementi čvorova smatraju obojanima prema strogim pravilima

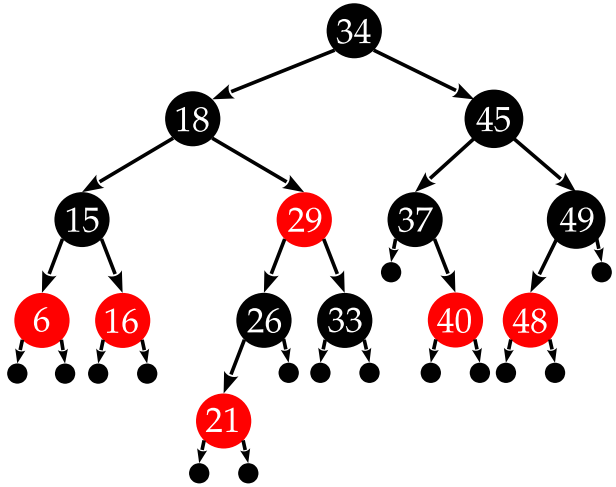
- Usporedba s B-stablom:
  - manji utrošak memorije
  - zadržava uravnoteženost
  - složenost ista



Definition rules

1. each node is **red** or **black**
2. is the root **black** (optional but common)
3. each sheet* is **black**
4. both descendants **red** nodes are **black**
5. every path from a node to (any) leaf that is its descendant passes through the same one

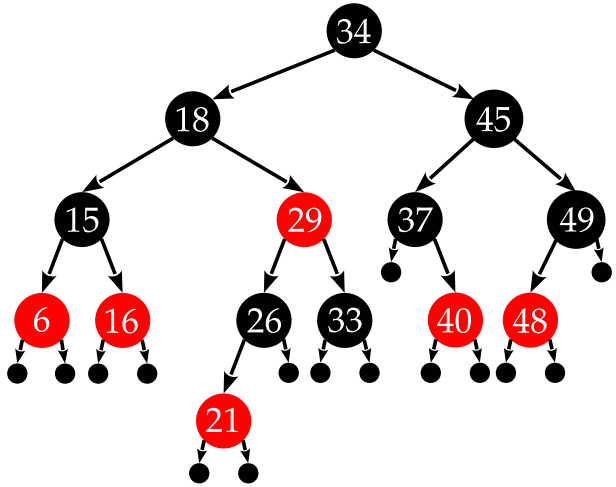
   by the number of black nodes



- * Leaves in a red-black (RB) tree do not contain information, so they do not have to exist, but parents can have NULL pointers or all point to the same special node, the sentinel

Definicijska pravila

1. svaki čvor je **crven** ili **crn**
2. korijen je **crn** (neobavezno, ali uobičajeno)
3. svaki list* je **crn**
4. oba potomka crvenog čvora su **crna**
5. svaka staza od nekog čvora do (bilo kojeg) lista koji je njegov potomak prolazi istim brojem crnih čvorova



- *Listovi u crveno-crnom (RB) stablu ne sadrže informacije pa ne moraju ni postojati, nego roditelji mogu imati NULL pokazivače ili svi pokazivati isti poseban čvor, sentinel

Red and black tree height

- We differentiate red and **black** tree height:
  - rh(x), bh(x)
    - the number of nodes of a certain color on the path from node x to the leaf that is its descendant (x is not counted).

- Key property for RB tree balance:
  - the longest path from the root to a leaf is at most twice as long as the shortest path from the root to a (other) leaf
  - i.e. the longest path is at most twice as long as the shortest.

Crvena i crna visina stabla

- Razlikujemo crvenu i **crnu** visinu stabla:
  - rh(x), bh(x)
    - broj čvorova određene boje na putu od čvora x do lista koji mu je potomak (x se ne broji).

- Ključno svojstvo za uravnoteženost RB stabla:
  - najduži put od korijena do nekog lista najviše je dvostruko duži od najkraćeg puta od korijena do nekog (drugog) lista
  - tj. najduži put je najviše dvostruko duži od najkraćeg.

Theorem

- The height of the RB-tree s of internal nodes is $h \le 2^{!}(+ 1)$ **Evidence:**

  Binary height tree*h*has the most $= 2^{"}- 1$of nodes Due to the 4th rule, at least half the height is black height so it is $h \ge h/2$.Since n is greater than or equal to the number of black nodes on the path from the root to the lowest leaf, it follows:

  $$\ge 2^{"\#} - 1 \ge 2^{"}- 1, \text{and} \overline{!} \text{ from that directly}$$

  $$h \le 2^{!}(+ 1)$$

- Searching a binary tree is of complexity O(h), so the complexity of searching an RB-tree $( )$!

Teorem

- Visina RB-stabla s n unutarnjih čvorova je

  $$h \leq 2log_2(n+1)$$

  Dokaz:

  Binarno stablo visine h ima najviše $n = 2^h - 1$ čvorova
  Zbog 4. pravila, barem polovica visine je crna visina pa je $hb \geq h/2$. Budući da je n veći ili jednak broju crnih čvorova na putu od korijena do najnižeg lista, slijedi:

  $n \geq 2^{hb} - 1 \geq 2^{\frac{h}{2}} - 1$, a iz toga izravno

  $$h \leq 2log_2(n+1)$$

- Pretraživanje binarnog stabla je složenosti O(h) pa je složenost pretraživanja RB-stabla $O(log_2 n)$

Adding a node to the RB-tree

- For easier analysis, terms are introduced
  - node-uncle (uncle) which is denoted by**IN**, and means the twin of the parent of the observed node (parent's brother/sister)
  - the grandfather node denoted by s**MR**(grandparent), meaning the parent of a parent

1. insert a new node as in any other binary search tree and assign itredcolor
2. restructure the tree (by applying rotations and coloring nodes) to satisfy the definition rules

Dodavanje čvora u RB-stablo

- Radi lakše analize, uvode se pojmovi
  - čvor-ujak (uncle) koji se označava s **U**, a znači blizanac roditelja promatranog čvora (roditeljev brat/sestra)
  - čvor-djed koji se označava s **G** (grandparent), a znači roditelj roditelja

1. ubaciti novi čvor kao u svako drugo binarno search stablo i pridijeliti mu crvenu boju
2. restrukturirati stablo (primjenom rotacija i bojanjem čvorova) da bi zadovoljilo definicijska pravila

Adding a node to the RB-tree

- Definition rules 1, 3 and 5 are always satisfied when adding a new node, and 2 and 4 can be compromised (not simultaneously) in the following ways:
    - rule 2 if the new node is the root
    - rule 4 if it is the parent of the new nodered
        - In both cases, restructuring is necessary
- Restructuring:

1.the new node is the root:

- recolor it in**black**(Rule 5 remains satisfied because it is an additional black node in all paths in the tree)

Dodavanje čvora u RB-stablo

- Definicijska pravila 1, 3 i 5 su uvijek zadovoljena kod dodavanja novog čvora, a 2 i 4 mogu biti ugrožena (ne istodobno) na sljedeće načine:
  - pravilo 2 ako je novi čvor korijen
  - pravilo 4 ako je roditelj novog čvora crven

    - U oba slučaja potrebno je restrukturiranje

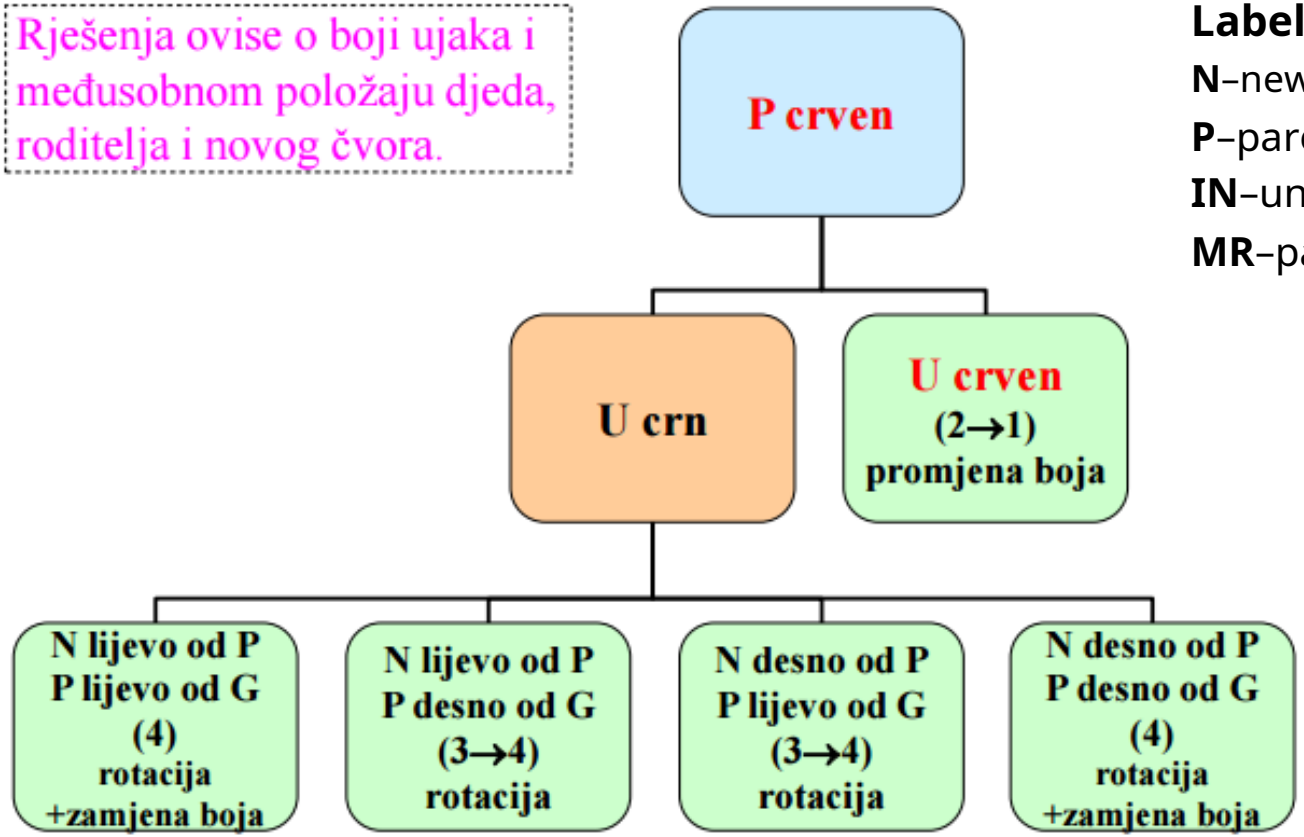
- Restrukturiranje:
  1. novi čvor je korijen:
     - prebojati ga u **crno** (5. pravilo ostaje zadovoljeno jer je to dodatni crni čvor u svim putevima u stablu)

Adding a node to the RB-tree

- Restructuring:

  2. the parent of the new node isred(step 1):

Rješenja ovise o boji ujaka i međusobnom položaju djeda, roditelja i novog čvora.

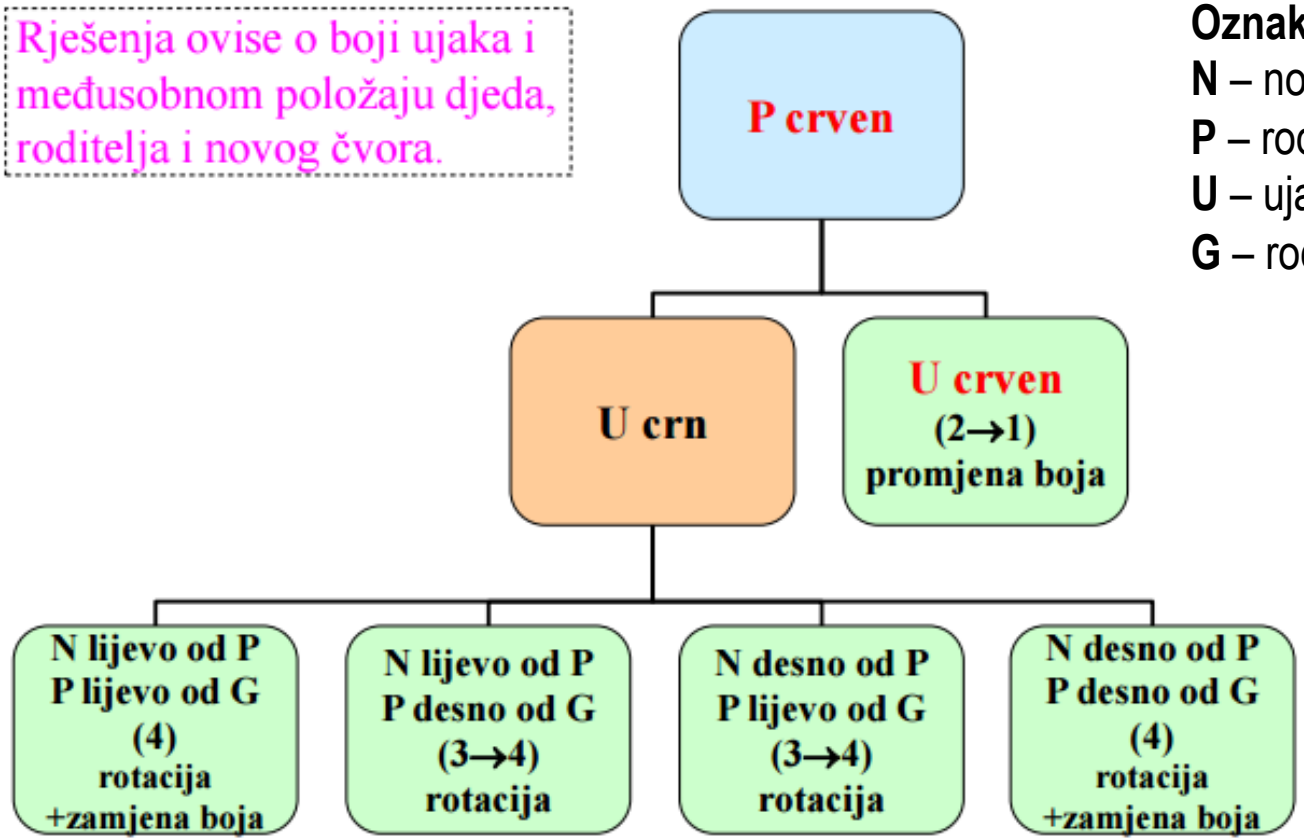


Labels:
N–new node
P–parent of N
IN–uncle (twin of P)
MR–parent of P

Dodavanje čvora u RB-stablo

- Restrukturiranje:
  2. roditelj novog čvora je crven (korak 1):

Rješenja ovise o boji ujaka i međusobnom položaju djeda, roditelja i novog čvora.

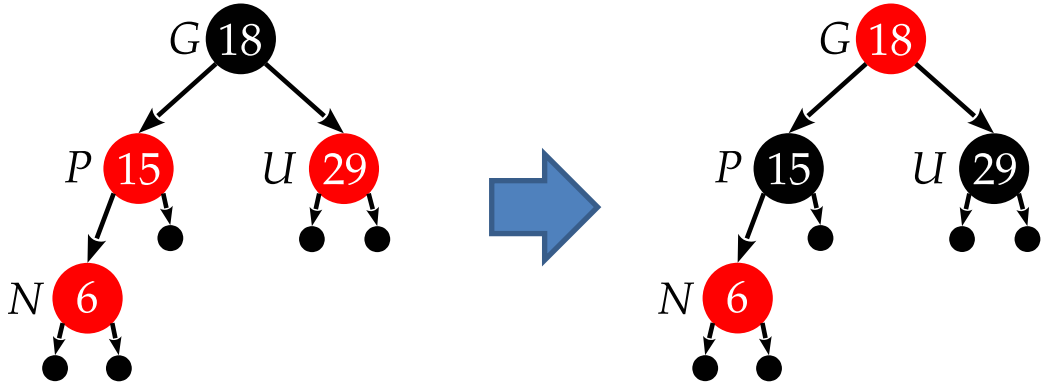


Oznake:
N – novi čvor
P – roditelj od N
U – ujak (blizanac od P)
G – roditelj od P

Adding a node to the RB-tree

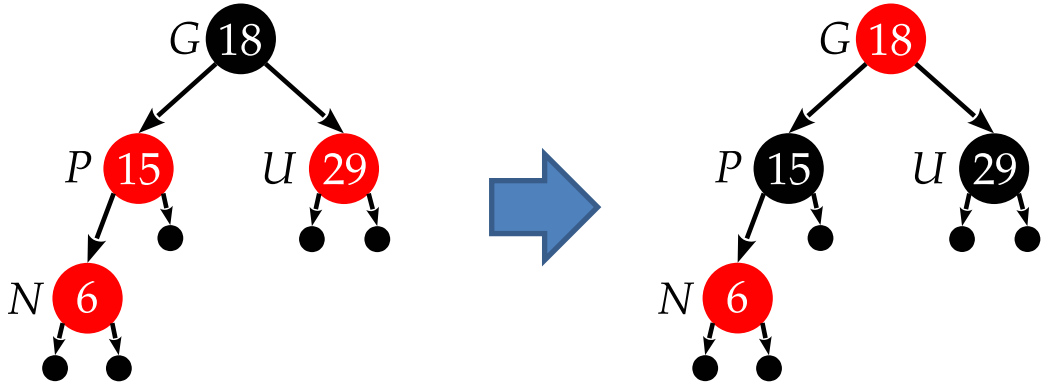
2.They are a parent and an uncleRed

- Rule 4 violated (two reds strung together; P and N)
  - repaint P and U in black (solves the 4th rule), and G inred (preservation of rule 5) - now G can violate rule 4 if it has redof the parent or the 2nd rule if it is the root
  - **Continue**with a check considering G as a new node (N)

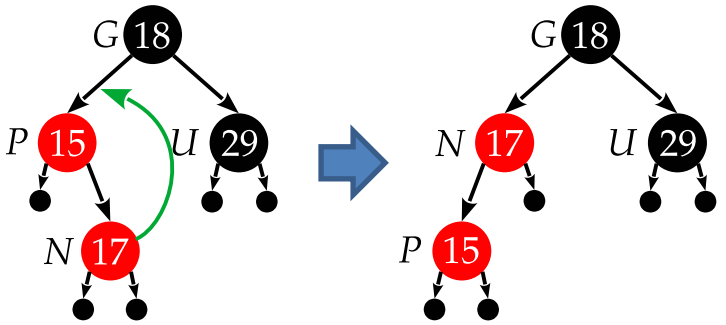


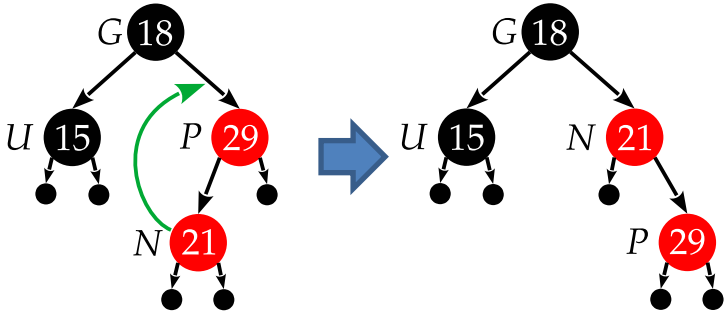
Dodavanje čvora u RB-stablo

2. Roditelj i ujak su crveni
   - Narušeno 4. pravilo (nanizana dva crvena; P i N)
     - prebojati P i U u crno (rješava 4. pravilo), a G u crveno (očuvanje 5. pravila) - sada G može narušavati 4. pravilo ako ima crvenog roditelja ili 2. pravilo ako je korijen
     - **Nastaviti** s provjerom promatrajući G kao novi čvor (N)



Adding a node to the RB-tree

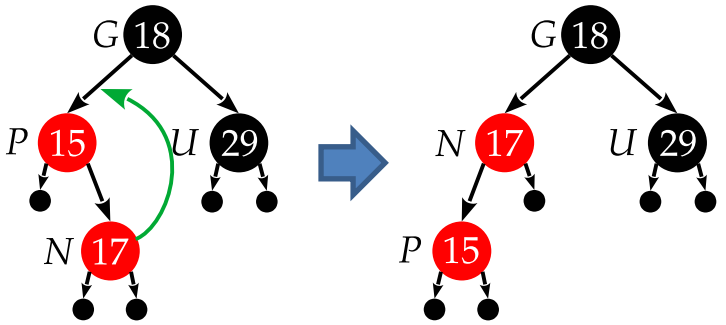


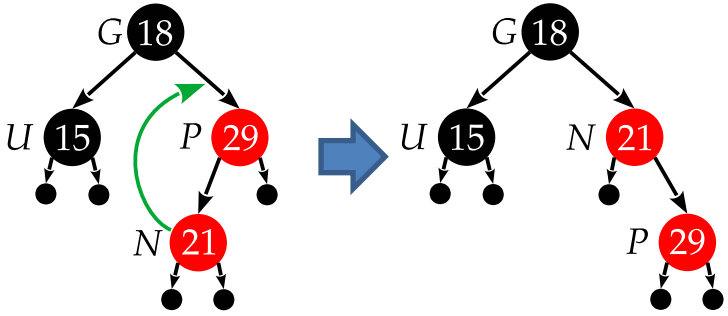


3.Parentredand Uncle Black ("broken" order of N, P and G)

- Two symmetrical cases:
  - N right child of P and P left child of G
  - N is the left child of P and P is the right child of G
- Solution:
  - the rotation of N around P, which translates the state into an “aligned order” of N, P and G which is solved in the 4th check
  - **Continue**with check (4), assigning Pu role to Na

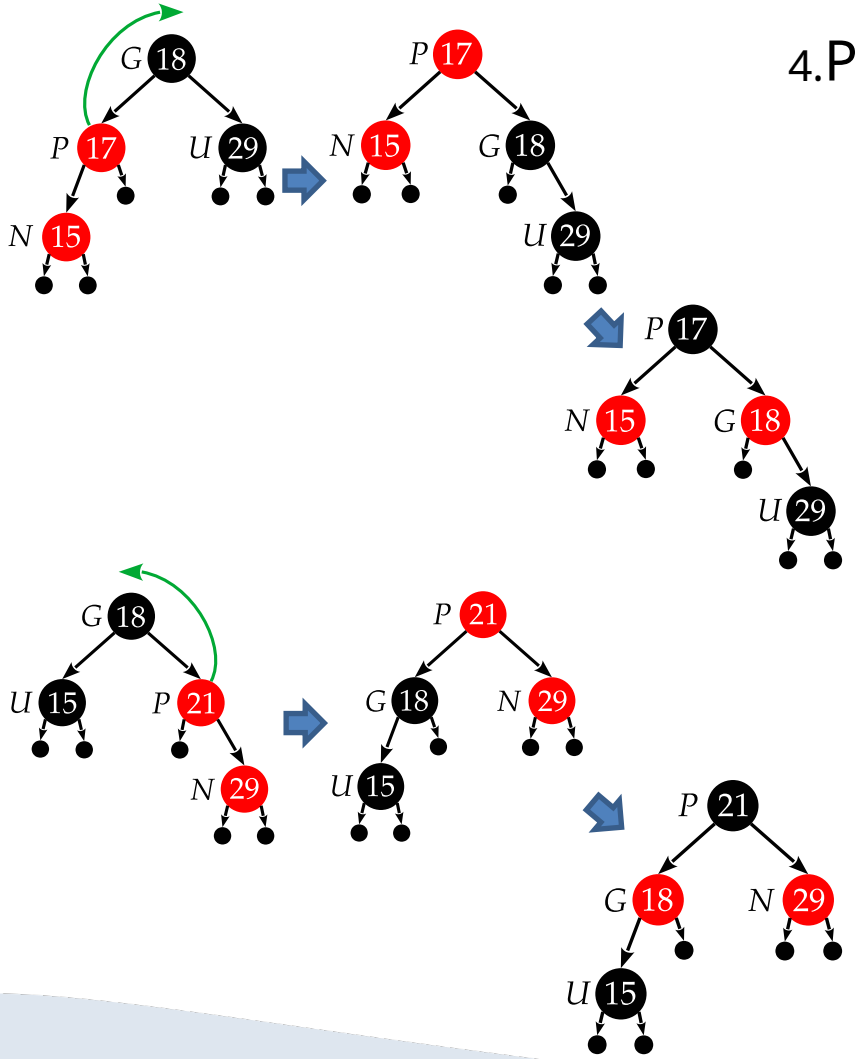
Dodavanje čvora u RB-stablo





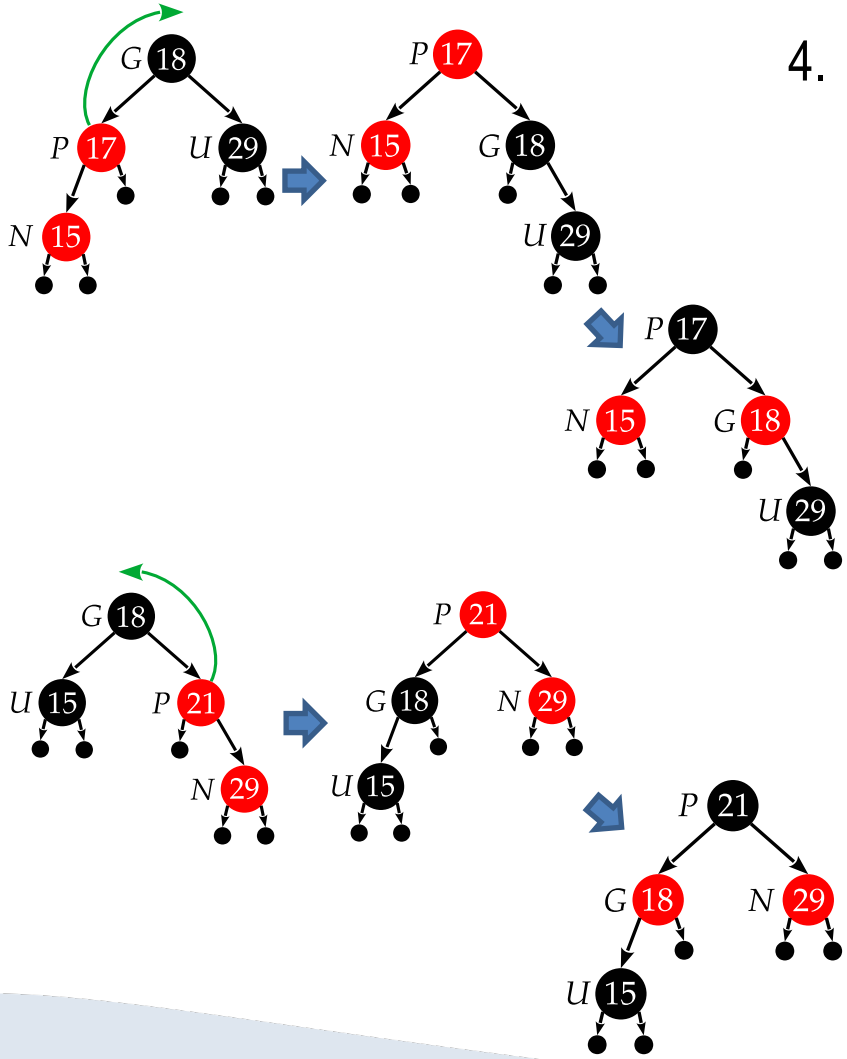
3. Roditelj crven i ujak crn (“izlomljeni” poredak N, P i G)
   - Dva simetrična slučaja:
     - N desno dijete od P i P lijevo dijete od G
     - N lijevo dijete od P i P desno dijete od G
   - Rješenje:
     - rotacija N oko P, čime se stanje prevodi u “izravnati poredak” N, P i G koji se rješava u 4. provjeri
     - **Nastaviti** s provjerom (4), pridajući P-u ulogu N-a

Adding a node to the RB-tree



4. Parent red and uncle black ("linear" order N, P and G)
   - Two symmetrical cases:
     - N is the left child of P and P is the left child of G
     - N right child of P and P right child of G
   - Solution:
     - *rotation* Around G
     - *color swap* P and G (we know that G is black because otherwise P would not be could be red); **END**

Dodavanje čvora u RB-stablo

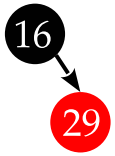


4. Roditelj crven i ujak crn (“linijski” poredak N, P i G)
   - Dva simetrična slučaja:
     - N lijevo dijete od P i P lijevo dijete od G
     - N desno dijete od P i P desno dijete od G
   - Rješenje:
     - *rotacija* P oko G
     - *zamjena boja* P i G (znamo da je G crn jer u protivnom P ne bi mogao biti crven); **KRAJ**

Example of adding data to the RB-tree

- We add the keys in order:**16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40**

- **Step 1**: We form the root node with the first key**16**. After adding, the node is red, so let's turn it into black (rule 2).

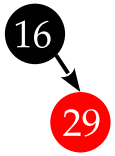


- **Step 2**: We add a key to the tree **29**. No RB-tree restructuring.

Primjer dodavanja podataka u RB-stablo

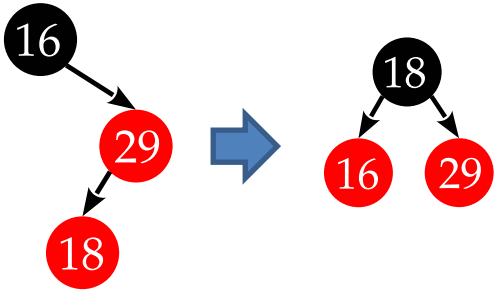
- Dodajemo redom ključeve: **16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40**

- **Korak 1**: Formiramo korijenski čvor s prvim ključem **16**. Nakon dodavanja, čvor je crveni, pa ga pretvorimo u crni (pravilo 2).

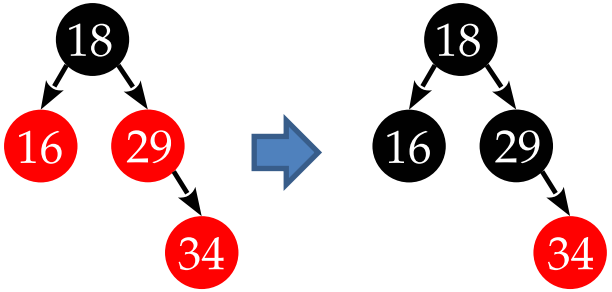


- **Korak 2**: U stablo dodajemo ključ **29**. Nema restrukturiranja RB-stabla.

Example of adding data to the RB-tree



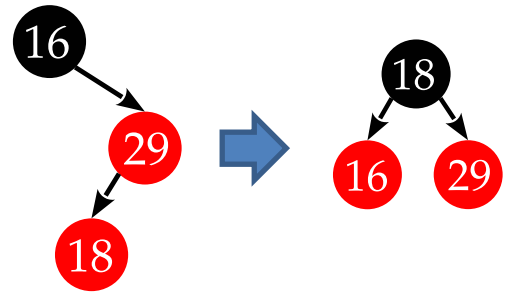
- **Step 3**: We add a key to the tree**18**.
  - **Case 3**: Right rotation**18**eye**29**
  - **Case 4**: Left rotation**18**eye**16**+ color swap.



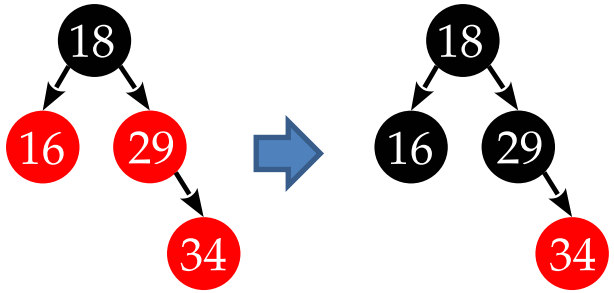
- **Step 4**: We add a key to the tree**34**.
  - **Case 2**: Set**18**in red, a**16**and **29**in black
  - **Case 1**: Set the root**18**in black

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Primjer dodavanja podataka u RB-stablo

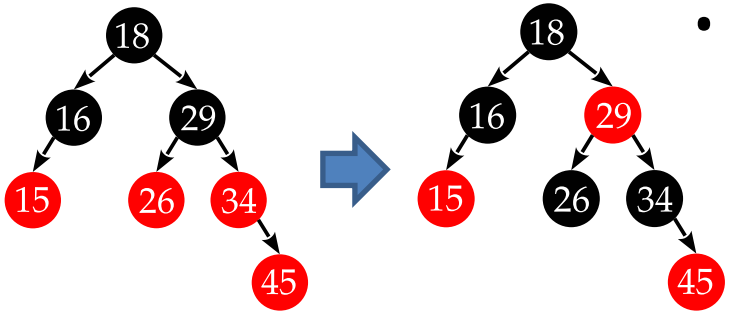


- **Korak 3**: U stablo dodajemo ključ **18**.
  - **Slučaj 3**: Desna rotacija **18** oko **29**
  - **Slučaj 4**: Lijeva rotacija **18** oko **16** + zamjena boja.

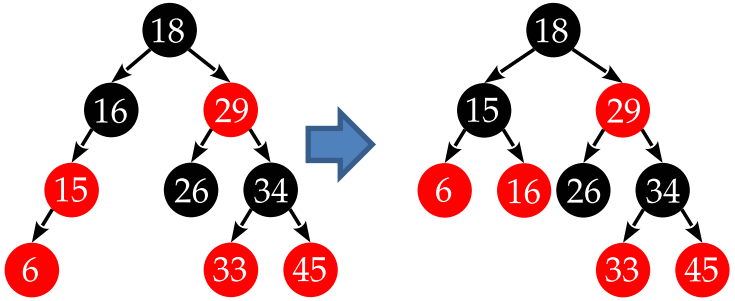


- **Korak 4**: U stablo dodajemo ključ **34**.
  - **Slučaj 2**: Postavi **18** u crveno, a **16** i **29** u crno
  - **Slučaj 1**: Postavi korijen **18** u crno

Example of adding data to the RB-tree



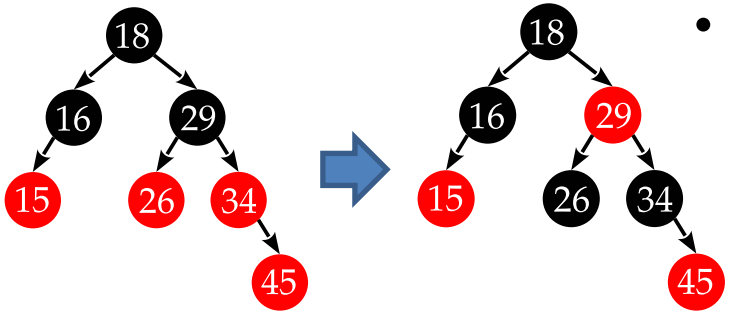
- **Step 5**: We add keys to the tree**26**, **15** and**45**.
  - After adding**45**we have**case 2**: Set**29**in red, a**26**and**34**in black.



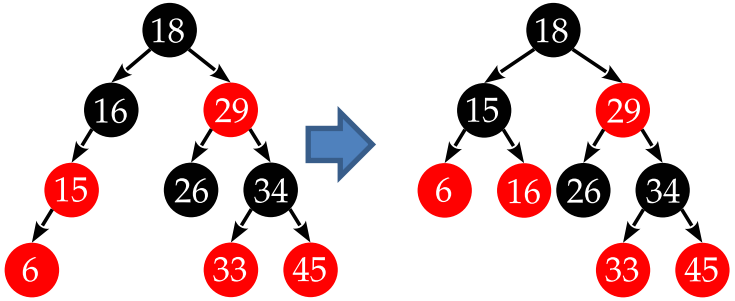
- **Step 6**: We add keys to the tree**33** and**6**.
  - After adding**6**we have**case 4**: right rotation**15**eye**16**and swapping colors in between**15**and**16**

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Primjer dodavanja podataka u RB-stablo



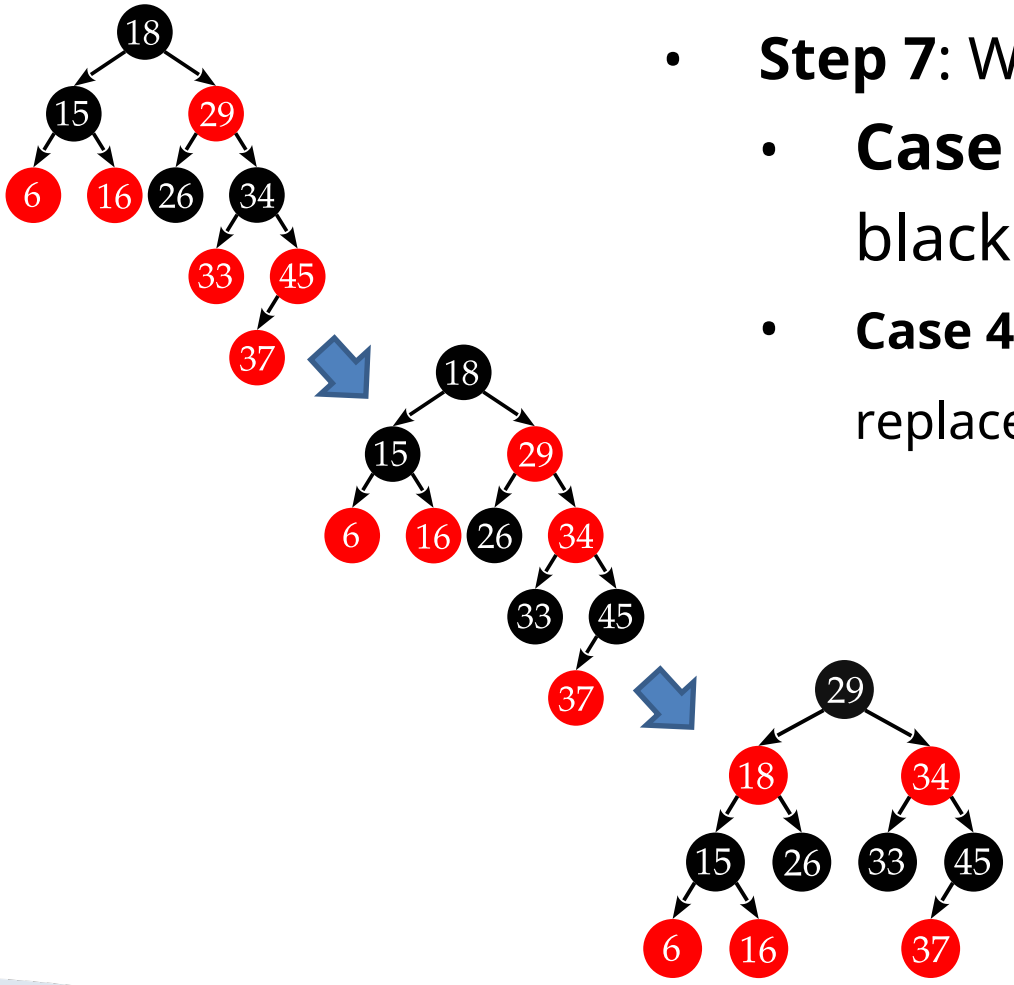
- **Korak 5**: U stablo dodajemo ključeve **26**, **15** i **45**.
  - Nakon dodavanja **45** imamo **slučaj 2**: Postavi **29** u crveno, a **26** i **34** u crno.



- **Korak 6**: U stablo dodajemo ključeve **33** i **6**.
  - Nakon dodavanja **6** imamo **slučaj 4**: desna rotacija **15** oko **16** i zamjena boja između **15** i **16**

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

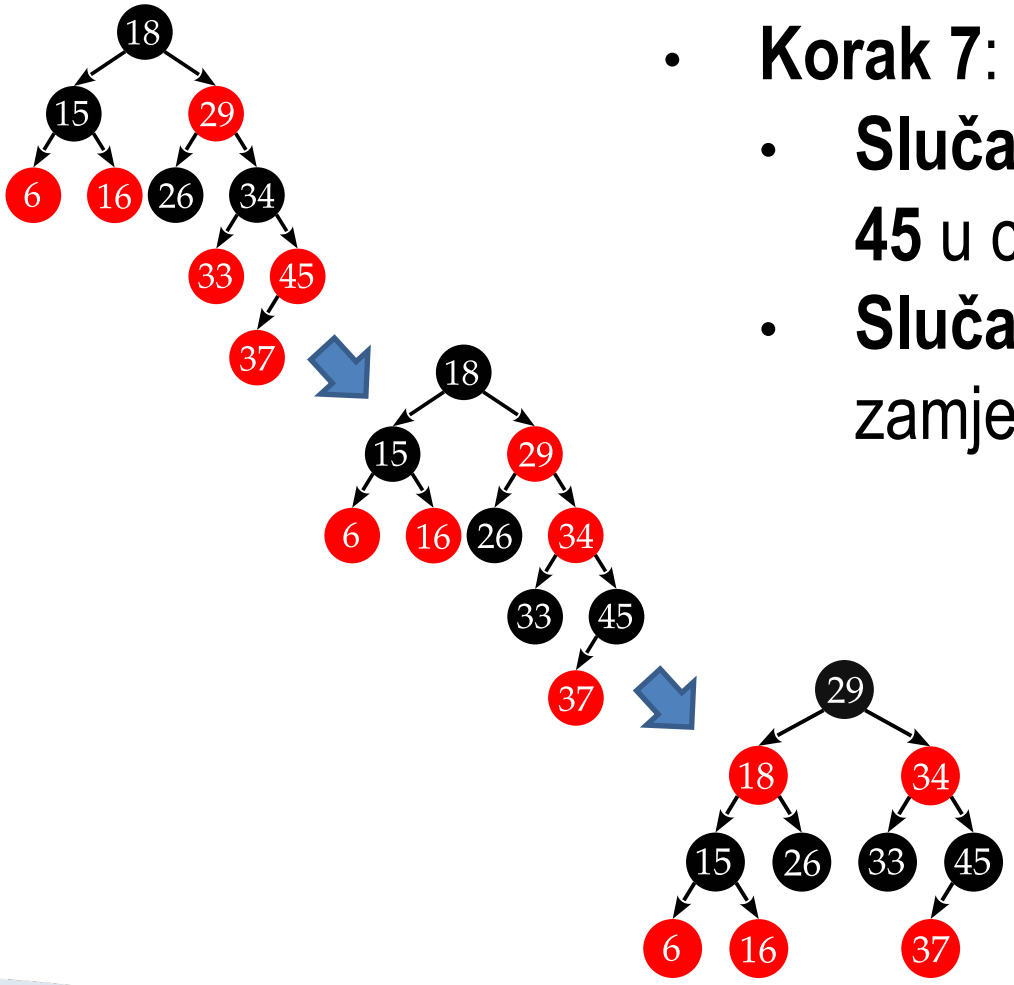
Example of adding data to the RB-tree



- **Step 7**: We add a key to the tree**37**.
  - **Case 2**: Set**34**in red, a**33**and **45**in black.
  - **Case 4**: Left rotation**29**eye**18**and color replacement**18**and**29**.

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

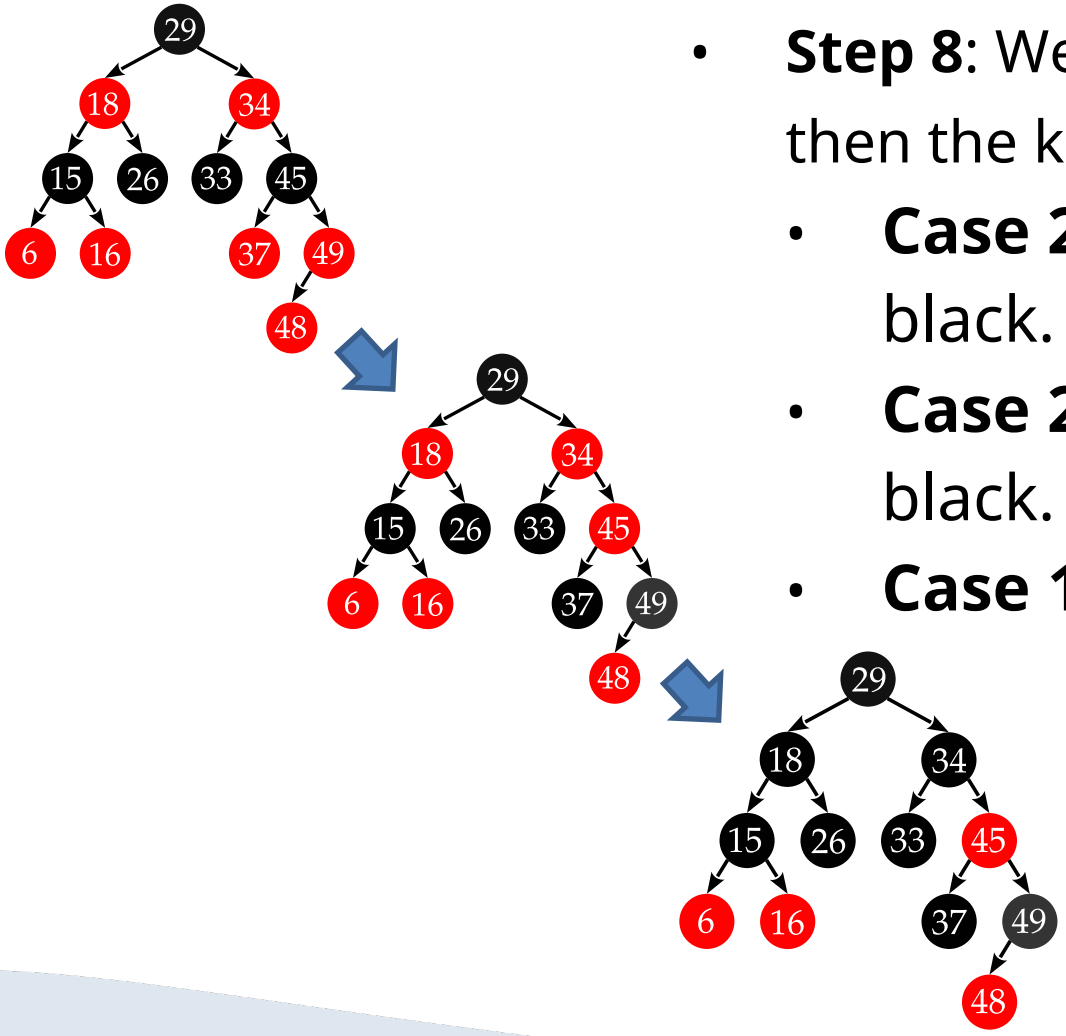
Primjer dodavanja podataka u RB-stablo



- **Korak 7**: U stablo dodajemo ključ **37**.
  - **Slučaj 2**: Postavi **34** u crveno, a **33** i **45** u crno.
  - **Slučaj 4**: Lijeva rotacija **29** oko **18** i zamjena boja **18** i **29**.

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

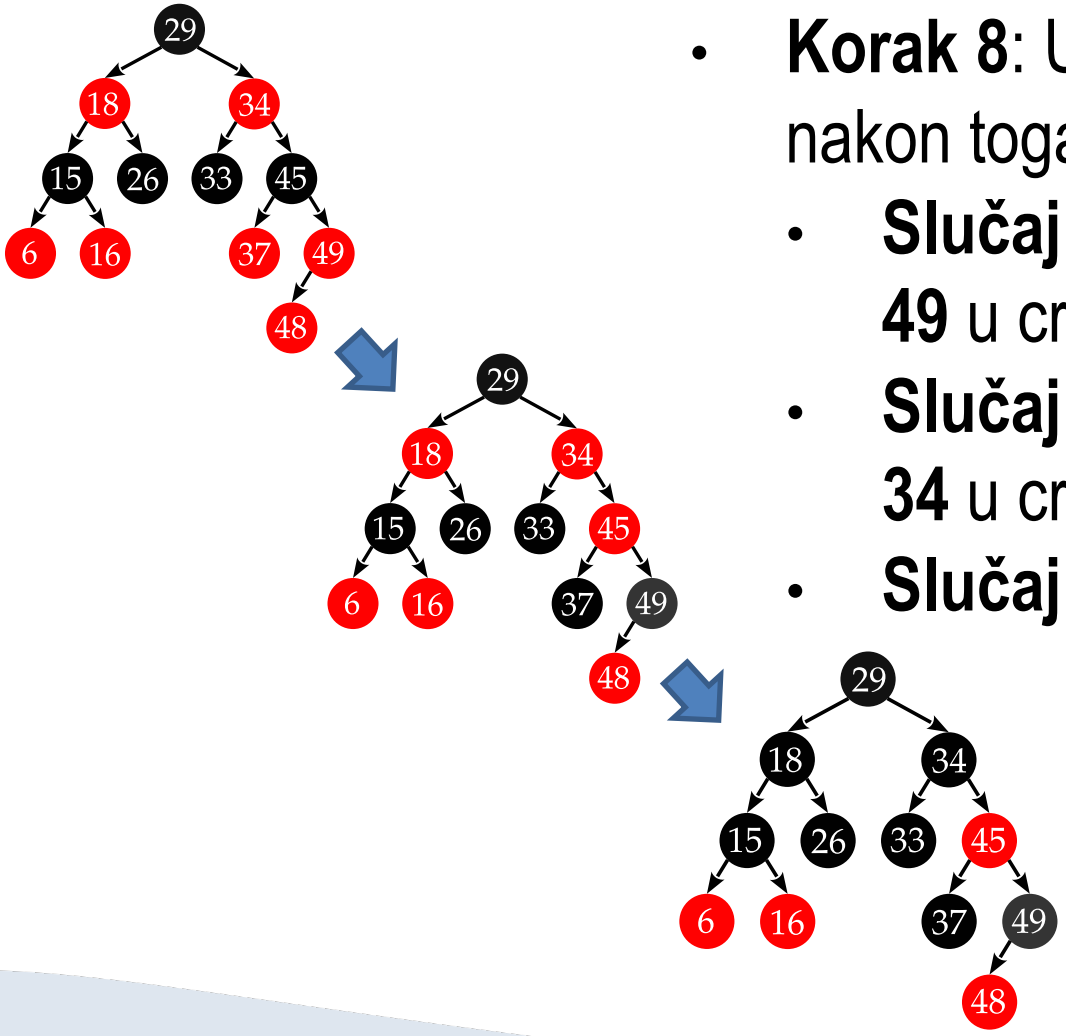
Example of adding data to the RB-tree



- **Step 8**: We add a key to the tree**49**, and then the key**48**.
  - **Case 2**: Set**45**in red, a**37**and **49**in black.
  - **Case 2**: Set**29**in red, a**18**and **34**in black.
  - **Case 1**: Set the root**29**in black.

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Primjer dodavanja podataka u RB-stablo

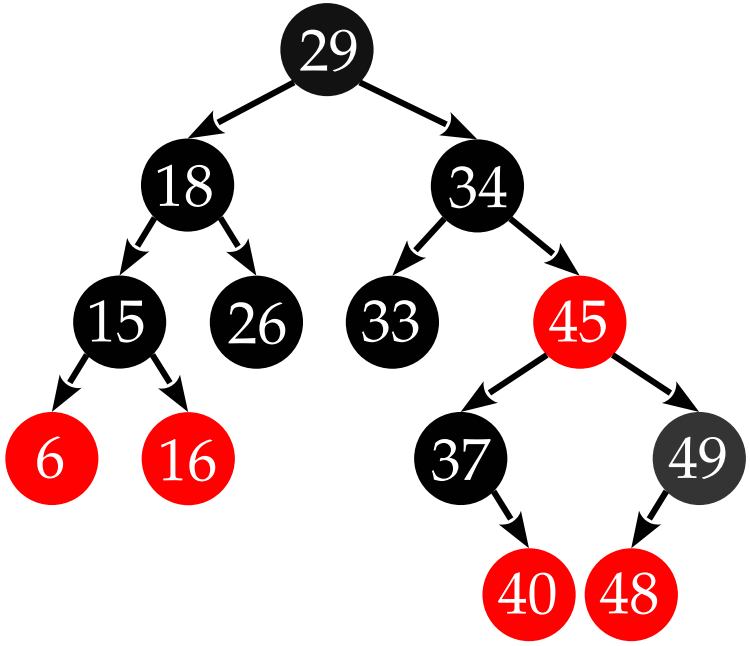


- **Korak 8**: U stablo dodajemo ključ **49**, te nakon toga ključ **48**.
  - **Slučaj 2**: Postavi **45** u crveno, a **37** i **49** u crno.
  - **Slučaj 2**: Postavi **29** u crveno, a **18** i **34** u crno.
  - **Slučaj 1**: Postavi korijen **29** u crno.

16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Example of adding data to the RB-tree

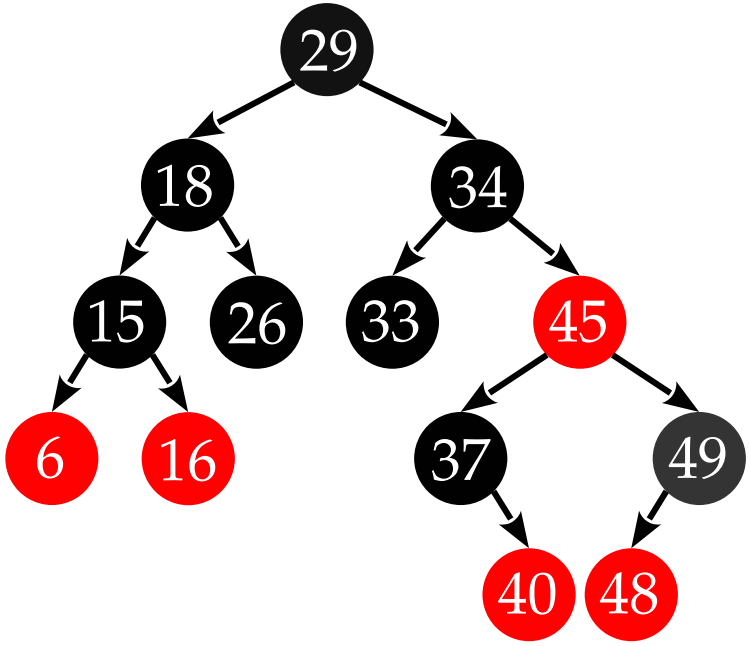
- **Step 9**: We add a key to the tree**40**



16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Primjer dodavanja podataka u RB-stablo

- **Korak 9**: U stablo dodajemo ključ **40**



16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Implementation of adding a node to the RB-tree

```
procedure RBTreeInsert(rbtree, N)
    P ← parent(N)
    G ← parent(P)
    while P is ● do
        if P is the left child then
            case ← LL
            if N is the right child then
                case ← LR
            U ← the right child of G
        else
            case ← RR
            if N is the left child then
                case ← RL
            U ← the left child of G
        if U and P are ● then
            P ← U ← ●
            G ← ●
            N ← G
        else if P is ● and U is ● then
            if case ∈ {LL, RR} then                ▷ straight cases
                if case is LL then
                    right rotate P around G
                else
                    left rotate P around G
                switch P and G colors
                break
            else                                   ▷ broken cases
                if case is LR then
                    right rotate N around P
                else
                    left rotate N around P
                N ← P
        P ← parent(N)
        G ← parent(P)
    root(rbtree) ← ●                               ▷ Rule 2
```

Implementacija dodavanja čvora u RB-stablo

```
procedure RBTreeInsert(rbtree, N)
    P ← parent(N)
    G ← parent(P)
    while P is ● do
        if P is the left child then
            case ← LL
            if N is the right child then
                case ← LR
            U ← the right child of G
        else
            case ← RR
            if N is the left child then
                case ← RL
            U ← the left child of G
        if U and P are ● then
            P ← U ← ●
            G ← ●
            N ← G
        else if P is ● and U is ● then
            if case ∈ {LL, RR} then                ▷ straight cases
                if case is LL then
                    right rotate P around G
                else
                    left rotate P around G
                switch P and G colors
                break
            else                                   ▷ broken cases
                if case is LR then
                    right rotate N around P
                else
                    left rotate N around P
                N ← P
        P ← parent(N)
        G ← parent(P)
    root(rbtree) ← ●                               ▷ Rule 2
```

Deleting a node in the RB-tree

- Algorithm:
  1. Delete by copying (substitute node; hereafter labeled X)
  2. Remove replacement node; he can have at most one child, so the problem is simplified

- If the replacement node is:
  - red: the properties of the RB-tree are not violated, the procedure is finished
  - **black**: a more complex procedure

Brisanje čvora u RB-stablu

- Algoritam:
  1. Brisanje kopiranjem (zamjenski čvor; u nastavku oznaka X)
  2. Ukloniti zamjenski čvor; on može imati najviše jedno dijete pa je problem pojednostavnjen

- Ako je zamjenski čvor:
  - crven: svojstva RB-stabla nisu narušena, postupak je gotov
  - **crn**: složeniji postupak

Blackhead removal

- There are 3 possible problems after removing a black node:
  1. if the root is removed, it could have only one child (N) which becomes the new root, and that can be ired
     - violation of the 2nd rule (the root is black)
  2. after removing X, its child N and parent P are in a child-parent relationship and if bothRed
     - violation of the 4th rule (childrenredare black)
  3. removing black X means reducing the black height of all its predecessors (ancestors)
     - violation of the 5th rule
- For the first case, it is enough to recolor N in black and everything is solved, because by changing the color of the root, the black height of all nodes of the tree changes equally. The other two cases depend on the color of node N.

Uklanjanje crnog čvora

- 3 su moguća problema nakon uklanjanja crnog čvora:
  1. ako je uklonjen korijen, mogao je imati samo jedno dijete (N) koje postaje novi korijen, a ono može biti i crveno
     - povreda 2. pravila (korijen je crn)
  2. nakon uklanjanja X, njegovo dijete N i roditelj P su u odnosu dijete-roditelj i ako su oboje crveni
     - povreda 4. pravila (djeca crvenog su crna)
  3. uklanjanje crnog X znači smanjenje crne visine svih njegovih prethodnika (predaka)
     - povreda 5. pravila
- Za prvi slučaj dovoljno je prebojati N u crno i sve je riješeno jer se mijenjanjem boje korijena jednako mijenja crna visina svim čvorovima stabla. Ostala dva slučaja ovise o boji čvora N.

Blackhead removal

- Let's imagine that we can somehow transfer the blackness of X to N. Then by removing X we would not lose it and the RB rules would not be violated:
  - If N was previouslyred, will become red-black and contribute 1 to the black height.
  - If N was previously black, it will become double black and contribute 2 to the black height.

- Solution:
  - If it is red-black, it is enough to repaint it in pure black.
  - If it is double black, the idea is to pass the excess black to the predecessor and thus raise that excess until it reaches a place where we can permanently incorporate it into the tree or until it reaches the root where we can ignore it.

Uklanjanje crnog čvora

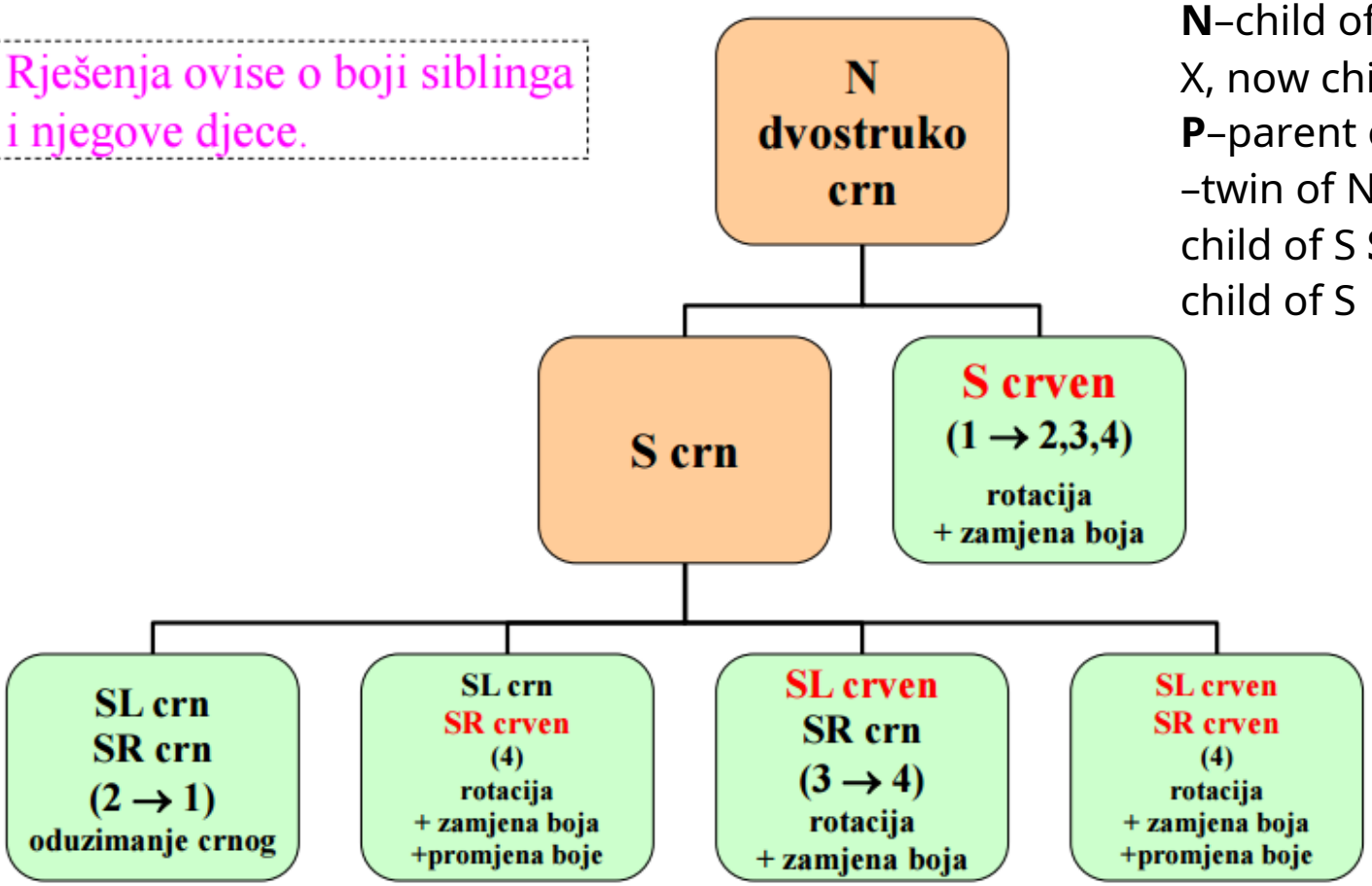
- Zamislimo da možemo nekako prenijeti crninu X-a na N. Tada ju uklanjanjem X ne bismo izgubili i RB pravila ne bi bila prekršena:
  - Ako je N prethodno bio crven, postat će crveno-crn i crnoj visini doprinositi 1.
  - Ako je N prethodno bio crn, postat će dvostruko crn i crnoj visini doprinositi 2.

- Rješenje:
  - Ako je crveno-crn, dovoljno je prebojati ga u čisto crno.
  - Ako je dvostruko crn, ideja je proslijediti višak crnog prethodniku i tako taj višak podizati sve dok ne dođe na mjesto gdje ga možemo trajno ugraditi u stablo ili dok ne dođe u korijen gdje ga možemo zanemariti.

Black node removal (double black)

- 4 (+4 symmetrical) are possible cases, and they depend on the color of the twin node (child of the same parent as N) and its children

Rješenja ovise o boji siblinga i njegove djece.

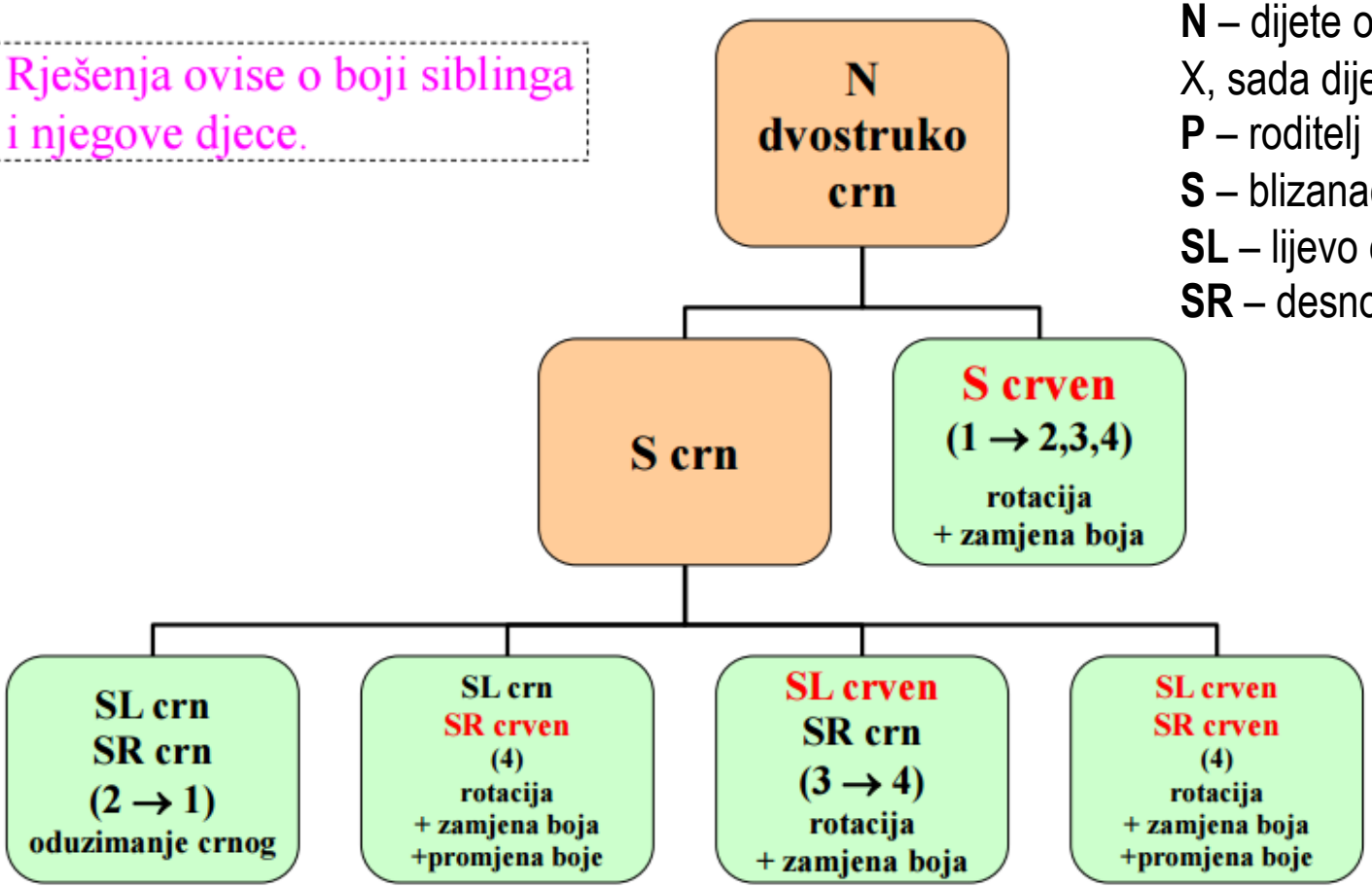


Labels:
N–child of removed X, now child of P
P–parent of N **WITH** –twin of N **FIG**–left child of S **SR**–right child of S

Uklanjanje crnog čvora (dvostruko crn)

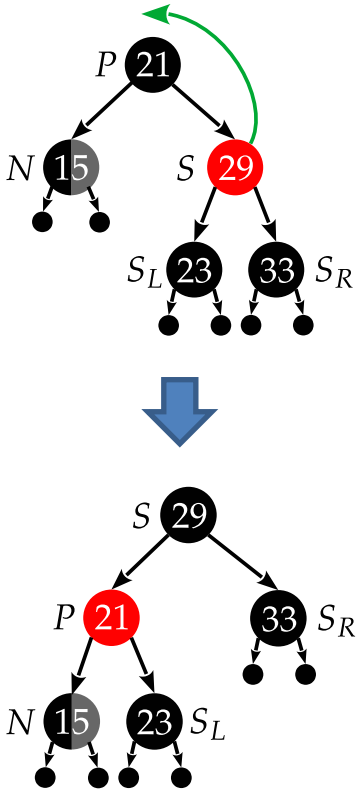
- 4 (+4 simetrična) su moguća slučaja, a ovise o boji čvora blizanca (dijete istog roditelja kao i N) i njegove djece

Rješenja ovise o boji siblinga i njegove djece.



Oznake:
N – dijete od uklonjenog X, sada dijete od P
P – roditelj od N
S – blizanac od N
SL – lijevo dijete od S
SR – desno dijete od S

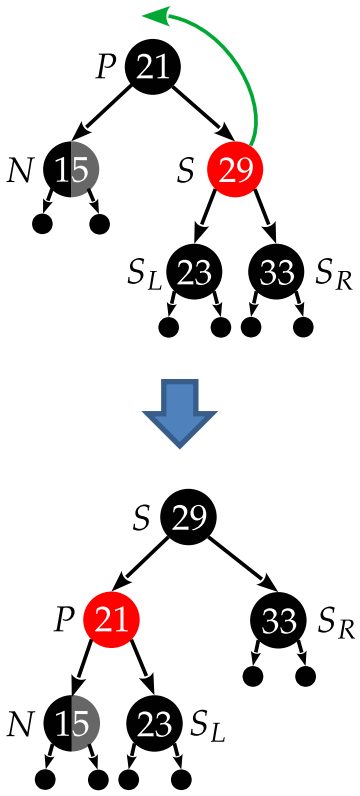
Black node removal (double black)



1. Twin S is red

- P must be black because it has red child
- After deleting X the black height of the left subtree of P is one less than the black height of the right subtree (ie N double black)
- Solution: rotate S around P (symmetry) and swap the colors of P and S
- **CONTINUATION** balancing from N

Uklanjanje crnog čvora (dvostruko crn)



1. Blizanac S je crven
   - P je sigurno crn jer ima crveno dijete
   - Nakon brisanja X crna visina lijevog podstabla od P za jedan je manja od crne visine desnog podstabla (tj. N dvostruko crn)
   - Rješenje: rotirati S oko P (simetrija) pa zamijeniti boje P i S
   - **NASTAVAK** uravnotežavanja iz N

Black node removal (double black)

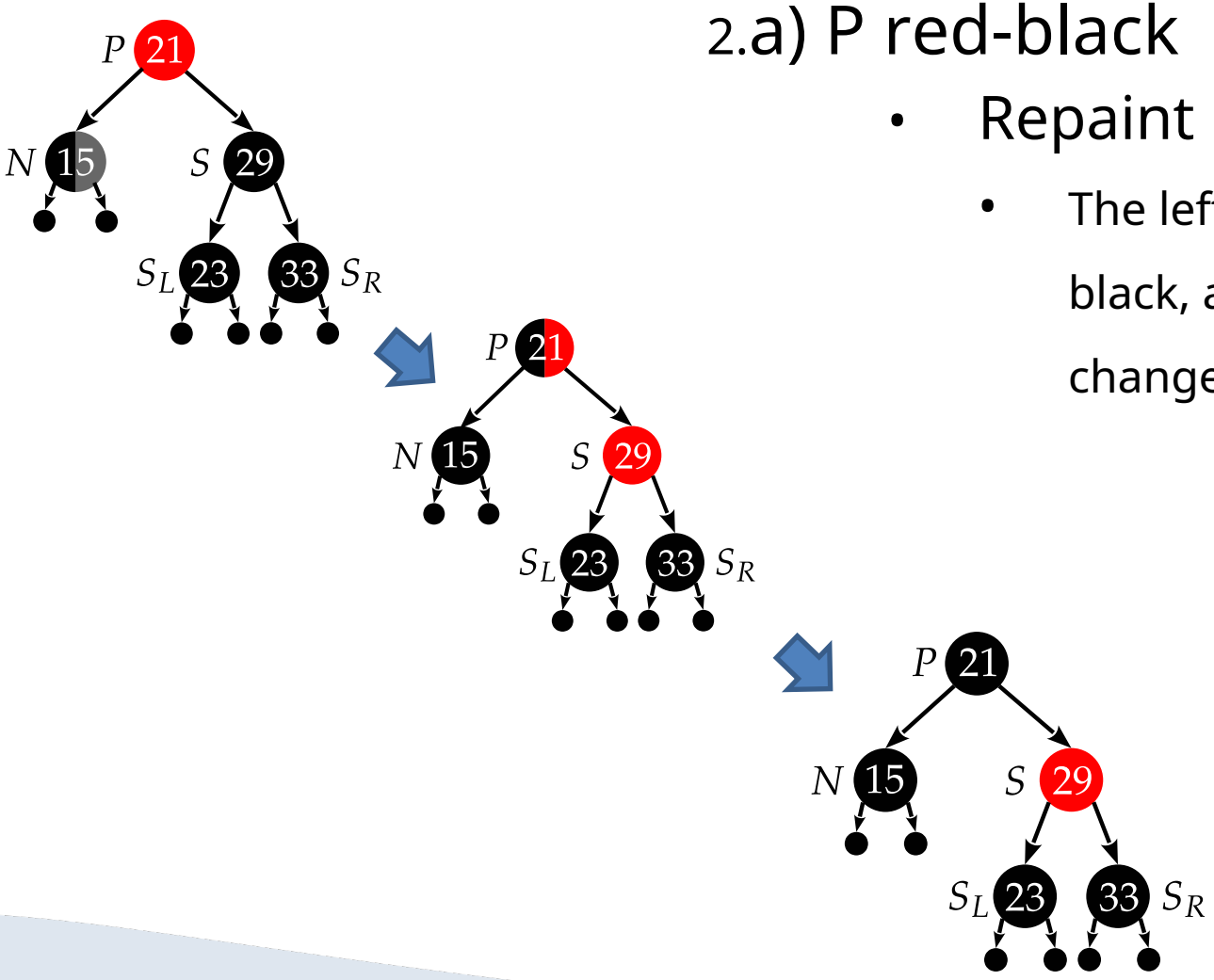
2. S black, children of S black

- Take away one black Nu and Su; N remains solid black and S becomesred
- Pass that excess black to a higher level (convergence!), i.e. Pu, which thereby becomes either red-black or double black
- The procedure after the intervention depends on Pu (cases 2a and 2b):

Uklanjanje crnog čvora (dvostruko crn)

2. S crn, djeca od S crna
   - Oduzeti jedno crno N-u i S-u; N ostaje jednostruko crn, a S postaje crven
   - Taj višak crnoga proslijediti višoj razini (konvergencija!), tj. P-u koji time postaje ili crveno-crn ili dvostruko crn
   - O P-u ovisi postupanje nakon intervencije (slučajevi 2a i 2b):

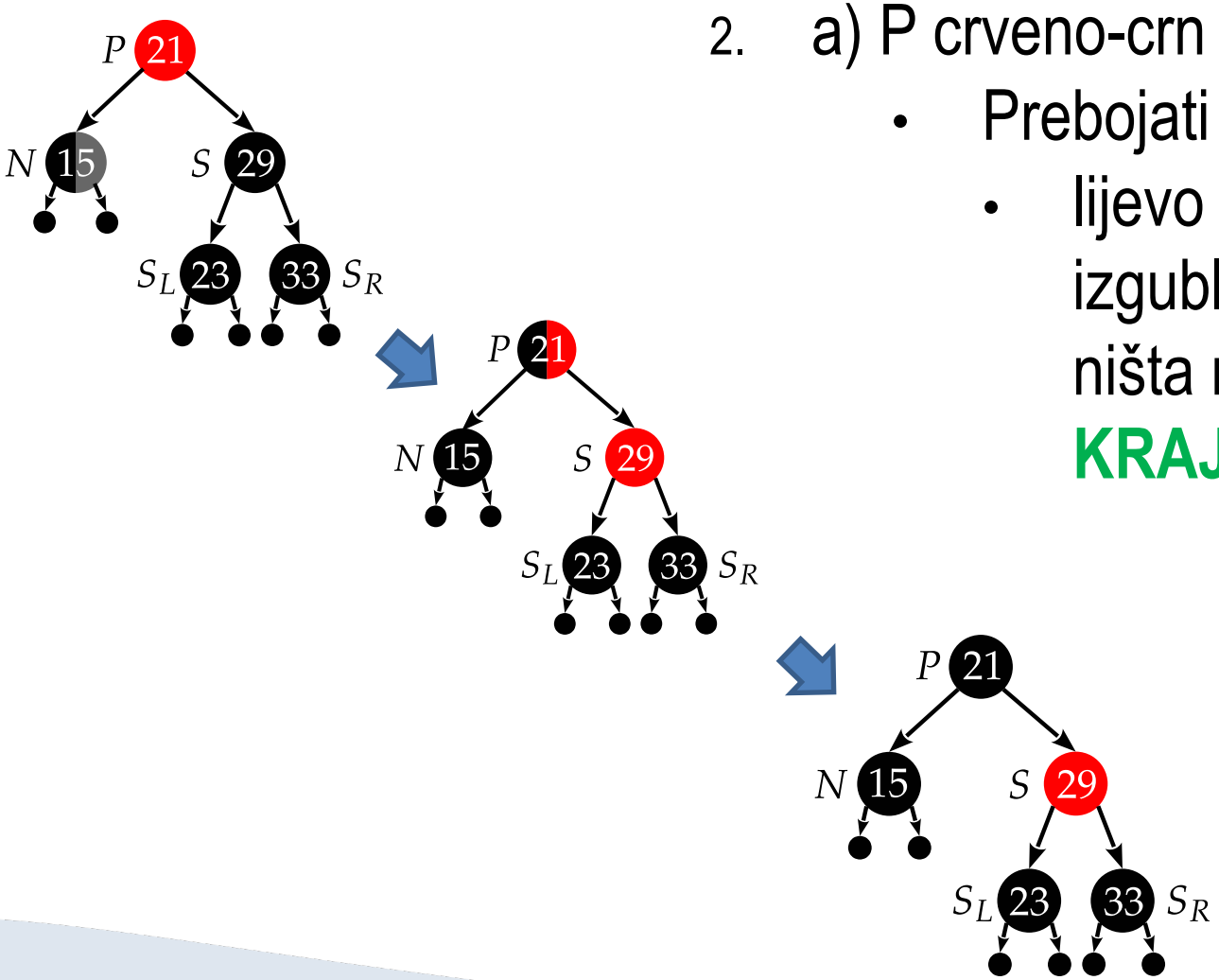
Black node removal (double black)



2.a) P red-black

- Repaint P black
  - The left subtree thus gets the lost black, and the right one does not change anything because Sred; **END**

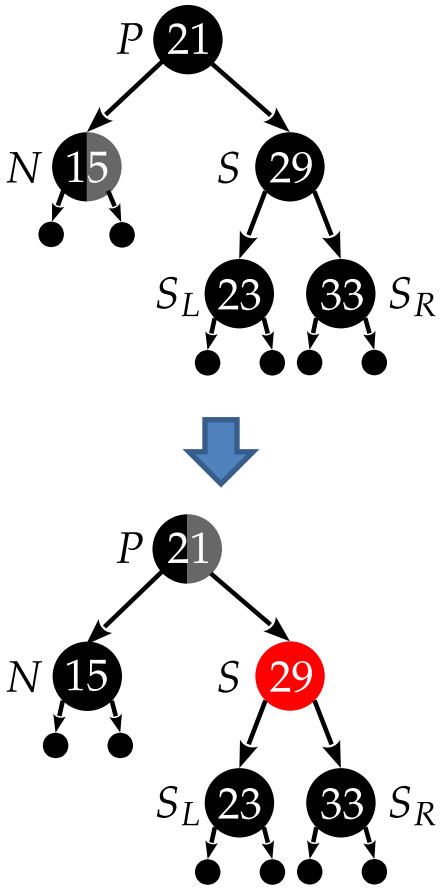
Uklanjanje crnog čvora (dvostruko crn)



2. a) P crveno-crn
   - Prebojati P u crno
     - lijevo podstablo time dobiva izgubljeno crno, a desnom se ništa ne mijenja jer je S crven;

KRAJ

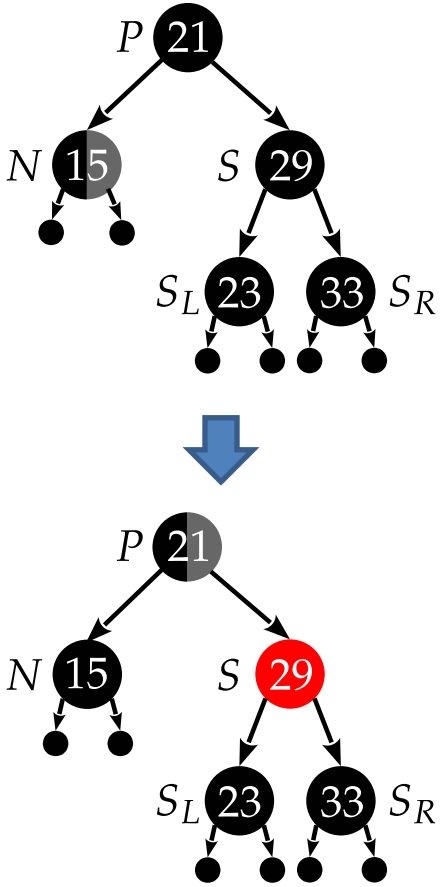
Black node removal (double black)



2.b) P double black

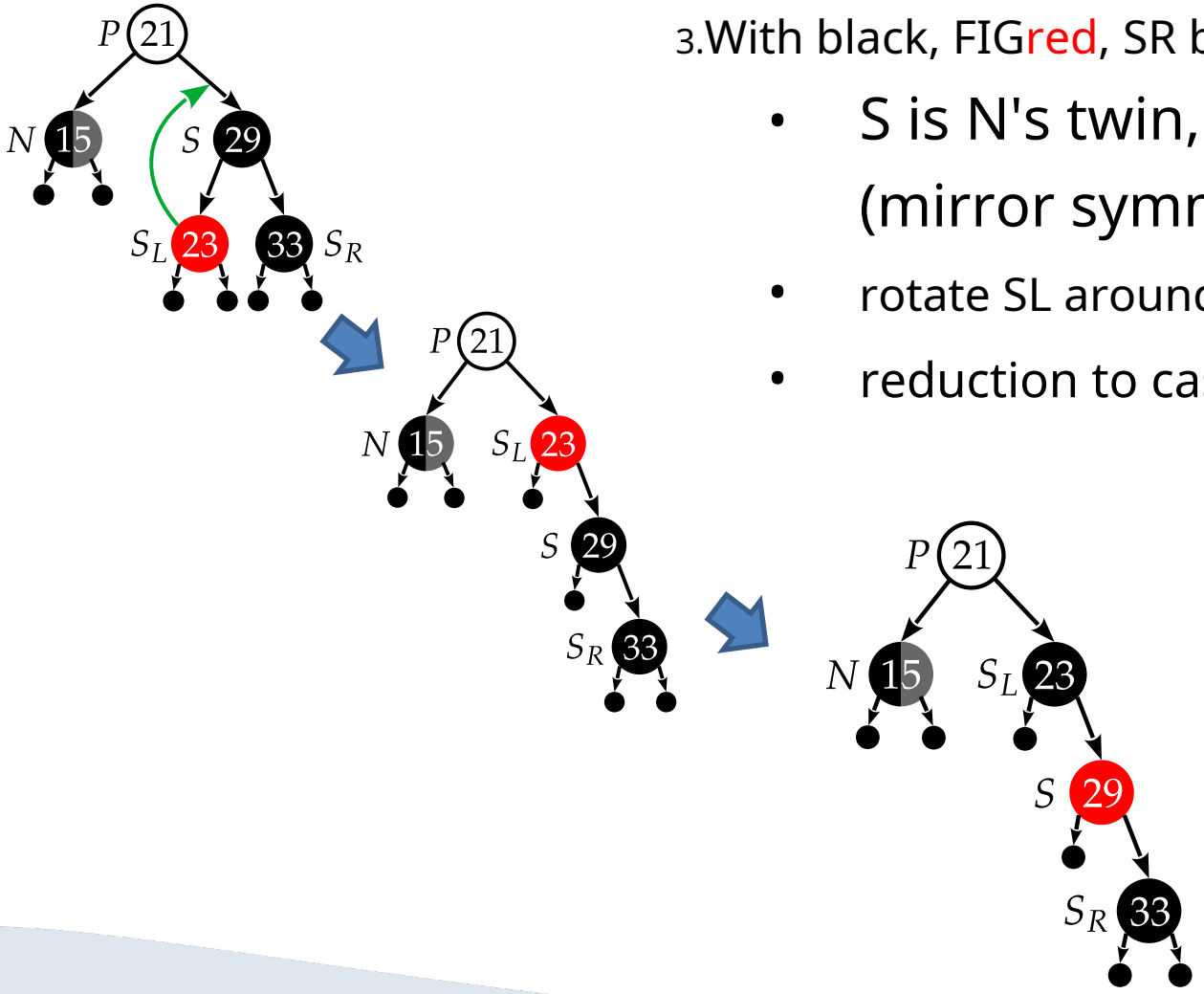
- P is the root
  - excess black is discarded;**END**
- P is not a root
  - back to case 1 viewing P as N; **CONTINUATION**
  - the problem is a level higher (convergence!)

Uklanjanje crnog čvora (dvostruko crn)



2. b) P dvostruko crn
   - P je korijen
     - višak crnog se odbacuje; **KRAJ**
   - P nije korijen
     - natrag na slučaj 1 promatrajući P kao N; **NASTAVAK**
     - problem je razinu više (konvergencija!)

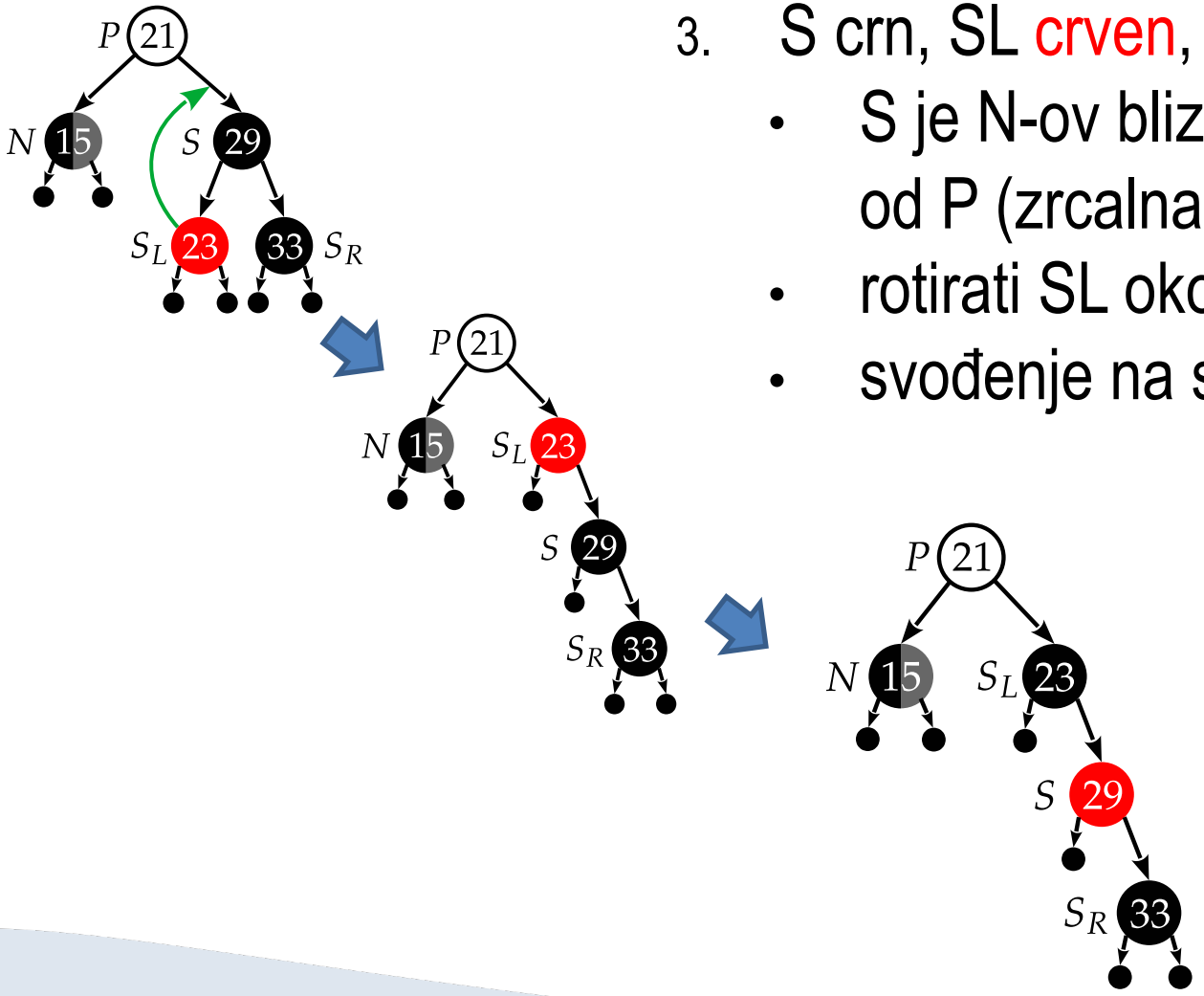
Black node removal (double black)



3.With black, FIGred, SR black, P unimportant

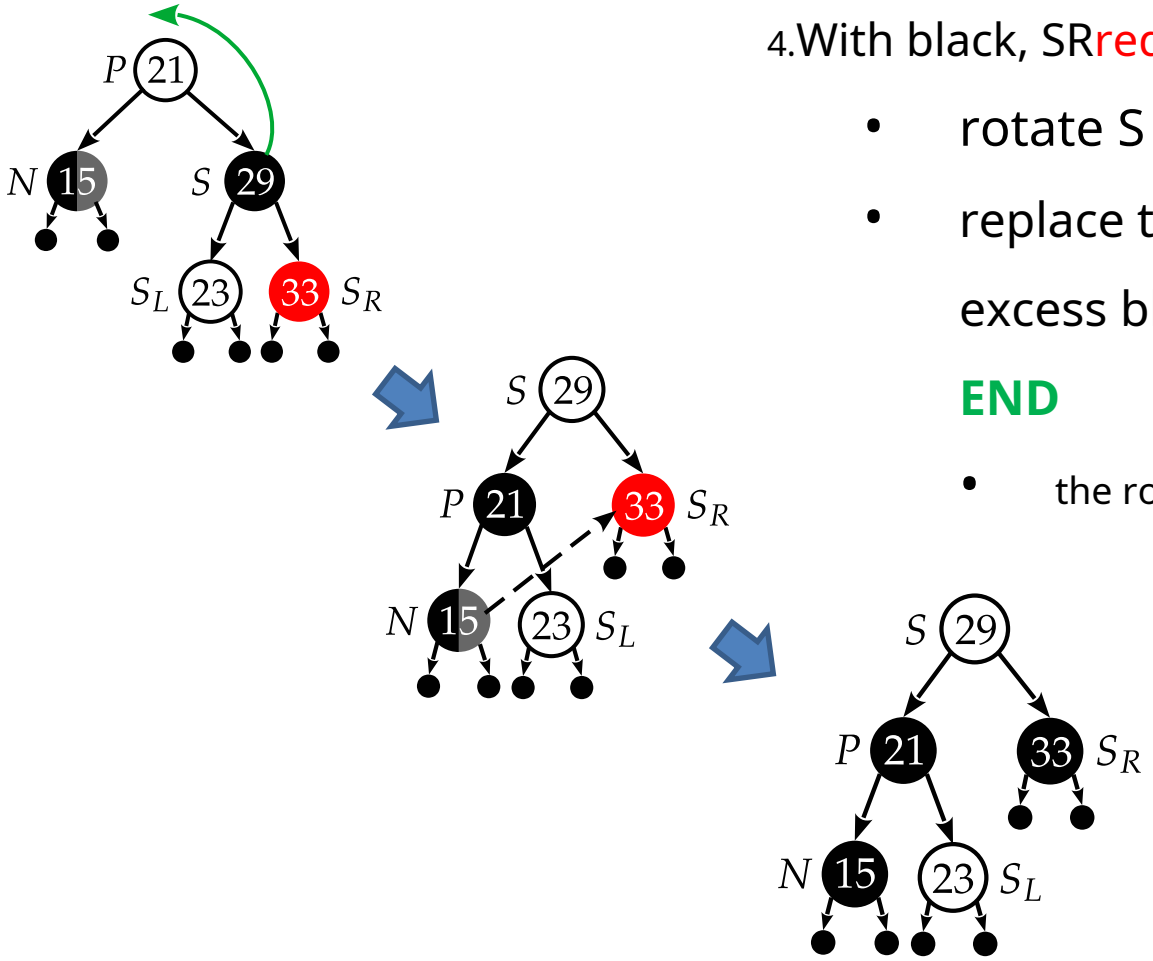
- S is N's twin, and N isleft child of P (mirror symmetry!)
- rotate SL around S and swap their colors
- reduction to case 4;**CONTINUATION**

Uklanjanje crnog čvora (dvostruko crn)



3. S crn, SL crven, SR crn, P nevažan
   - S je N-ov blizanac, a N je <u>lijevo</u> dijete od P (zrcalna simetrija!)
   - rotirati SL oko S i zamijeniti im boje
   - svođenje na slučaj 4; **NASTAVAK**

Black node removal (double black)



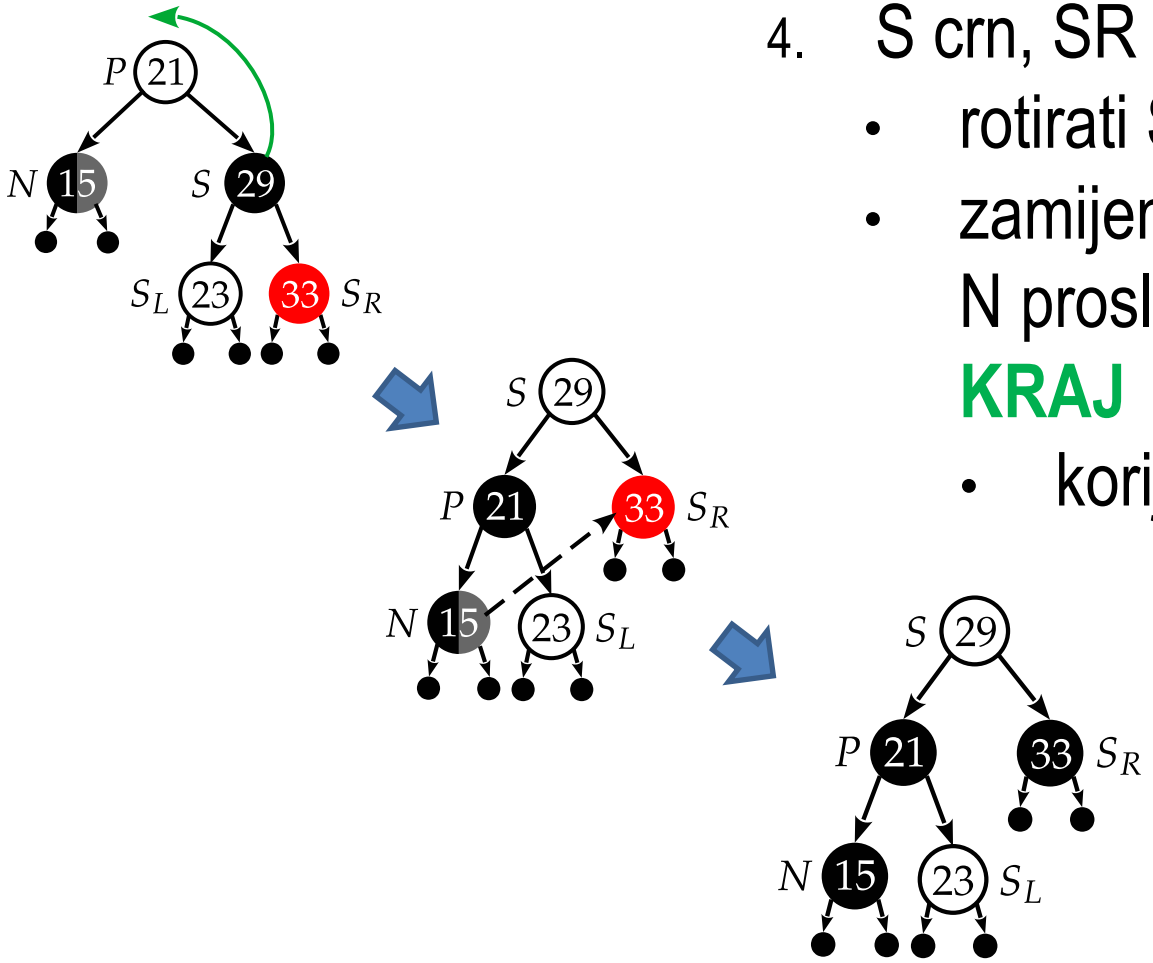
4.With black, SRred, P and SL unimportant

- rotate S around P (mirror symmetry!)
- replace the colors S and P, and transfer the excess black from N to SR (repaint in black);

END

- the root of the subtree remains the same color

Uklanjanje crnog čvora (dvostruko crn)



4. S crn, SR crven, P i SL nevažni
   - rotirati S oko P (zrcalna simetrija!)
   - zamijeniti boje S i P, a višak crnog iz N proslijediti u SR (prebojati u crno); **KRAJ**
   - korijen podstabla ostaje iste boje

Implementation of node deletion in the RB-tree

```
procedure RBTreeRemove(rbtree, N)
    while N is not root(rbtree) and N is ● do
        P ← the parent of N
        if N is the left child of P then
            S ← the right child of P
            S_L, S_R ← children of S
            if S is ● then
                S ← ●
                P ← ●
                left rotate S around P
            if S_L is ● and S_R is ● then
                S ← ●
                N ← the parent of N
            else
                if S_R is ● then
                    S_L ← ●
                    S ← ●
                    right rotate S_L around S
                color of S ← color of P
                P ← S_R ← ●
                left rotate S around P
                N ← root(rbtree)
        else                          ▷ implement the symmetrical cases
    N ← ●
```

Implementacija brisanja čvora u RB-stablu

```
procedure RBTreeRemove(rbtree, N)
    while N is not root(rbtree) and N is ● do
        P ← the parent of N
        if N is the left child of P then
            S ← the right child of P
            S_L, S_R ← children of S
            if S is ● then
                S ← ●
                P ← ●
                left rotate S around P
            if S_L is ● and S_R is ● then
                S ← ●
                N ← the parent of N
            else
                if S_R is ● then
                    S_L ← ●
                    S ← ●
                    right rotate S_L around S
                color of S ← color of P
                P ← S_R ← ●
                left rotate S around P
                N ← root(rbtree)
        else                          ▷ implement the symmetrical cases
    N ← ●
```