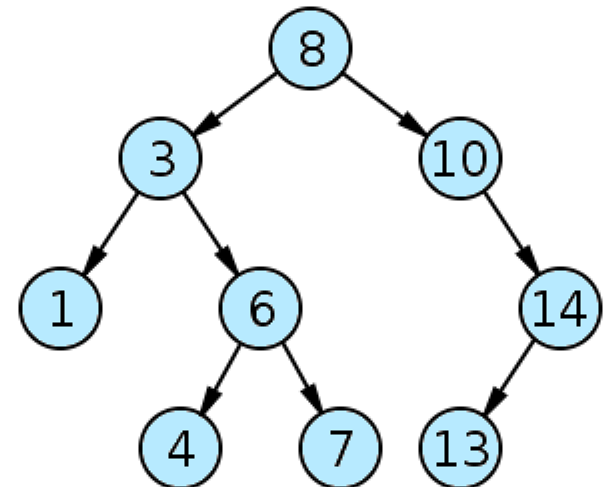
Binarna drevesa

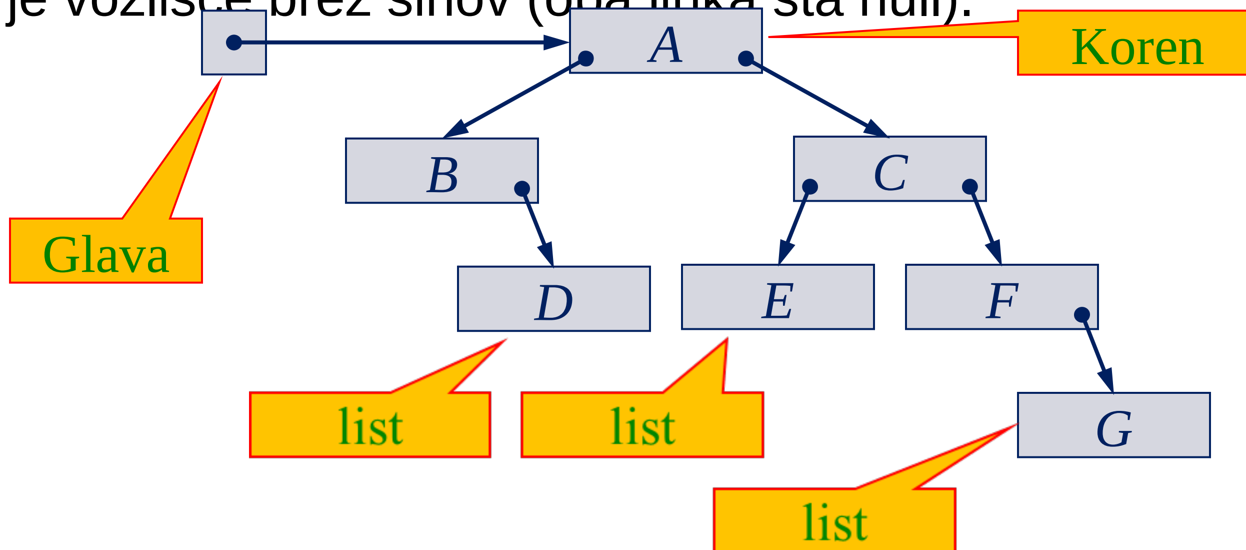
Obravnavali bomo

- Binarna drevesa in binarna iskalna drevesa.
- Iskanje elementov v drevesu
- Vnašanje elementov v drevo.
- Brisanje elementov iz drevesa.
- Prehajanje po drevesu.



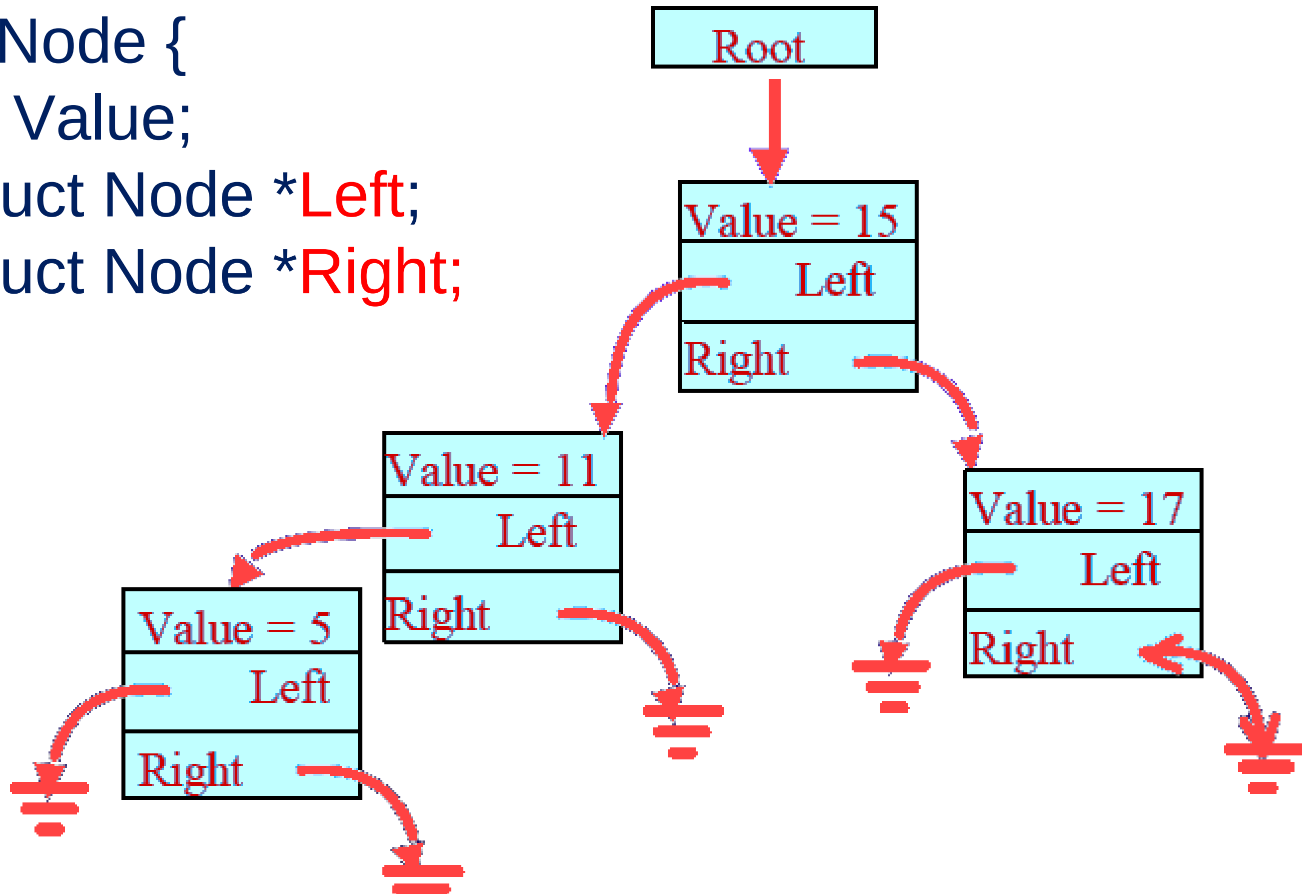
Binarna drevesa (1)

- Binarno drevo ima **glavo** ter vrsto **vozlišč**, s **povezavami** povezanih v hierarhično strukturo:
  - Vsako vozlišče vsebuje en **element** (vrednost ali objekt) in povezavi na največ dve drugi vozlišči (njegovega levega in desnega sina).
  - Glava vsebuje kazalec na vozlišče, ki je določeno kot **korensko** vozlišče
  - **List** je vozlišče brez sinov (oba linka sta null).



Binarna drevesa: primer

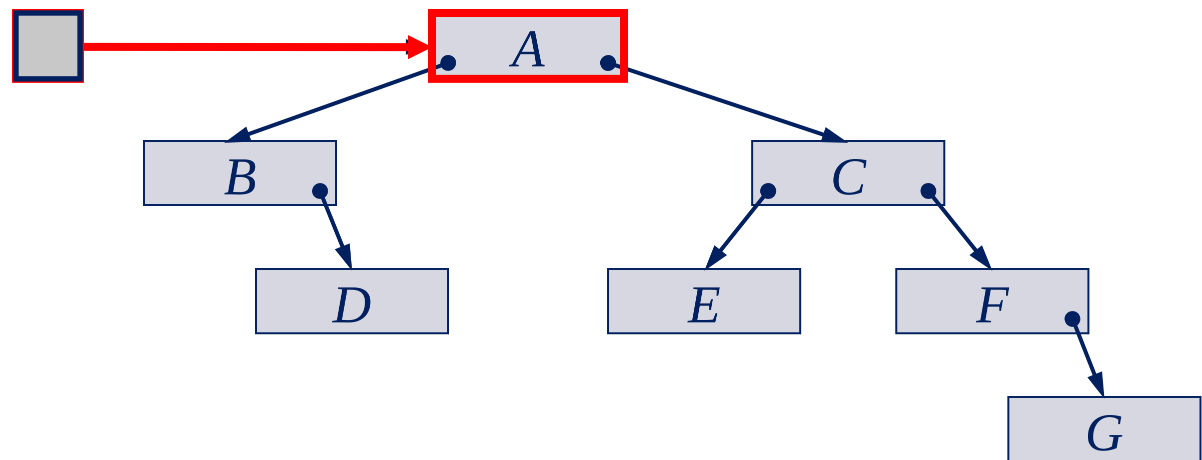
```
struct Node {
    int Value;
    struct Node *Left;
    struct Node *Right;
};
```



Demo Demo Knjižnica

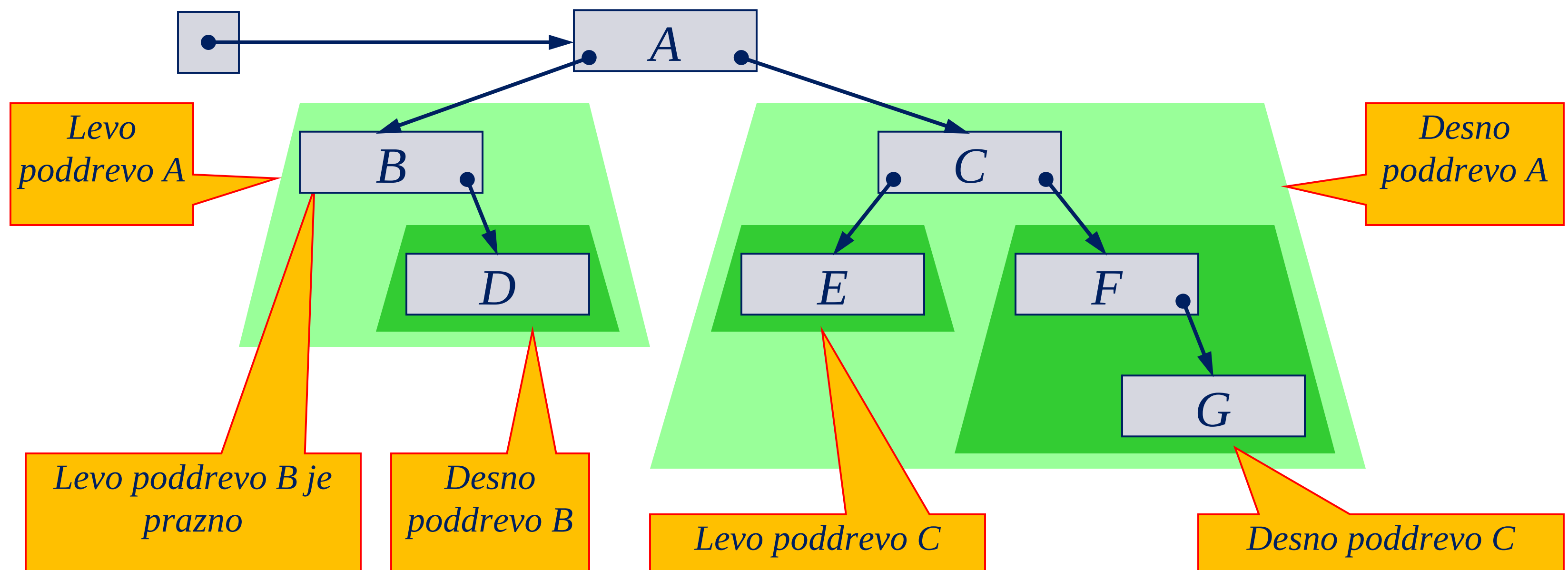
Binarna drevesa (2)

- Vsako vozlišče, razen korena, je levi ali desni sin nekega drugega vozlišča (starša).
- Korensko vozlišče nima starša. Nanj kaže glava (header).
- Velikost binarnega drevesa je število vozlišč.
- Prazno binarno drevo ima velikost nič. Kazalec nanj (glava) je enak null.



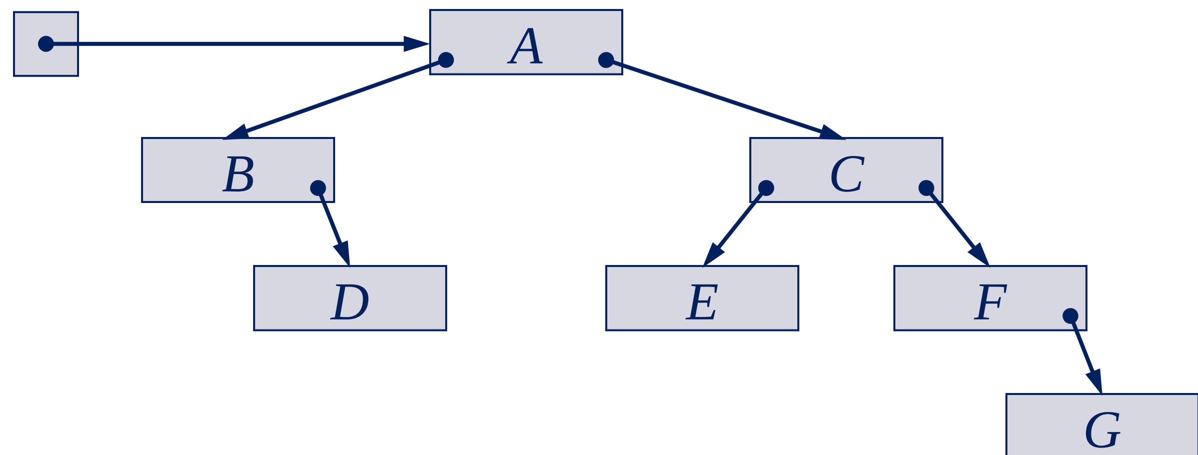
Binarna drevesa in poddrevesa

- Vsako vozlišče ima levo in desno poddrevo (ki pa sta lahko prazni).
- Levo (desno) poddrevo je predstavljeno z levim (desnim) sinom skupaj z njegovimi sinovi, vnuki itd..



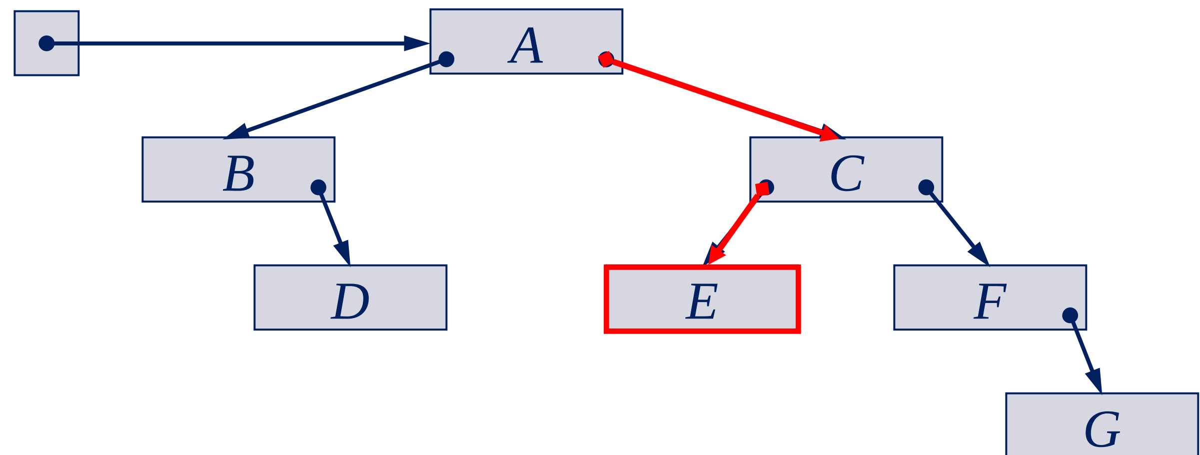
Binarna drevesa in poddrevesa

- Vsako poddrevo je spet drevo.
- Tako dobimo rekurzivno definicijo.
  Binarno drevo je:
  - **prazno**, ali
  - **neprazno**, v tem primeru ima korensko vozlišče, ki vsebuje en element in linka na levo in desno poddrevo

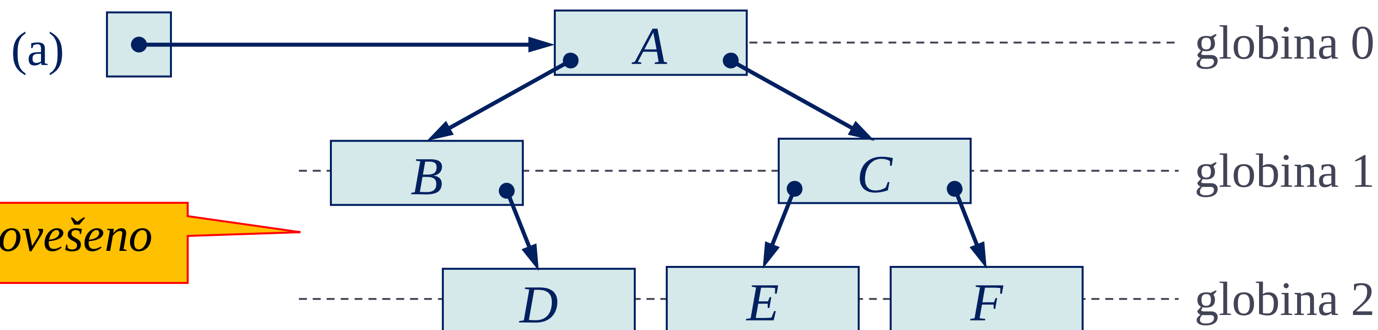


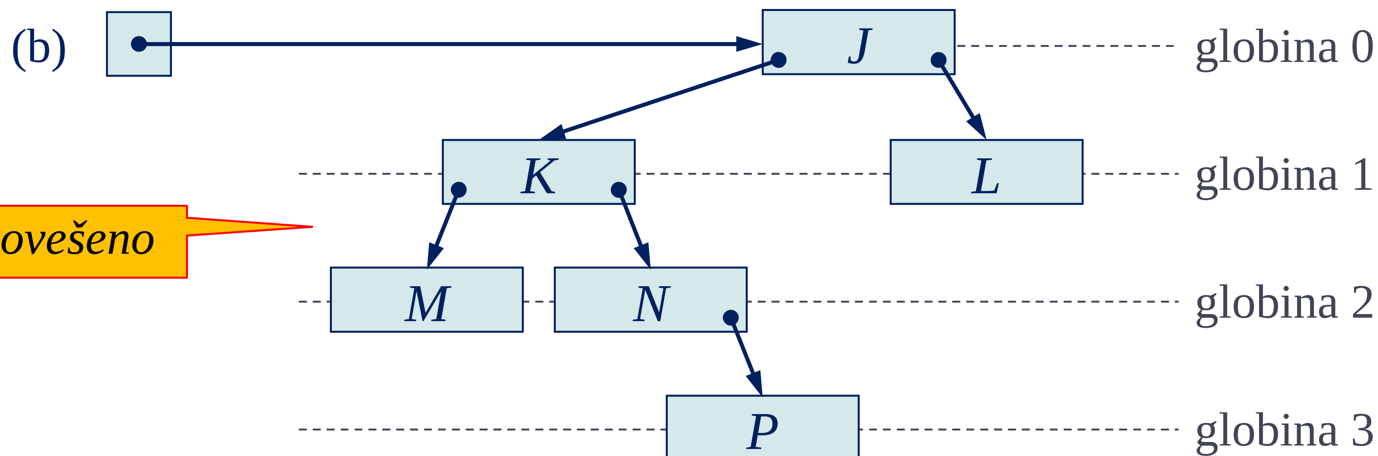
Vozlišča in globina drevesa

- Opazka: Za vsako vozlišče N v drevesu je natančno eno zaporedje povezav med korenskim vozliščem in N.
- **Globina** vozlišča *N* je število povezav med korenskim vozliščem in *N*.
- **Globina** drevesa je globina najglobljega vozlišča v drevesu.
  - Drevo z enim samim vozliščem ima globino 0.
  - Po dogovoru ima prazno drevo globino –1.

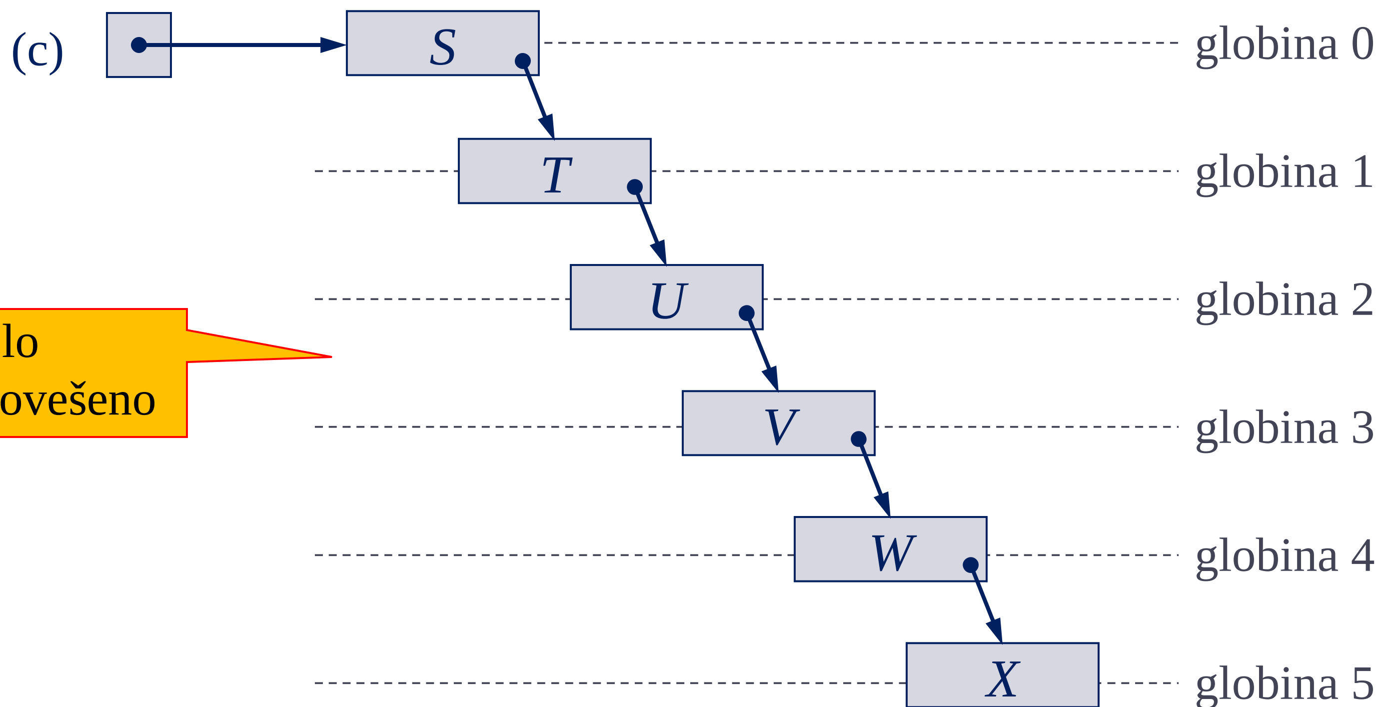


Ponazoritev globine dreves





Ponazoritev globine dreves (2)



Uravnovešena binarna drevesa

- Binarno drevo globine *d* je **uravnovešeno,** če imajo vsa vozlišča na globinah 0, 1, …, *d*–2 dva otroka.
  - Vozlišča na globini *d*–1 imajo lahko 2/1/0 sinov.
  - Vozlišča na globini *d* ne smejo imeti sinov (po definiciji).
  - Binarno drevo globine 0 ali 1 je vedno uravnovešeno.
- Uravnovešeno binarno drevo globine d ima najmanj 2^d in največ $2^{d+1} – 1$ vozlišč.

Binarno drevo: demo

PPT

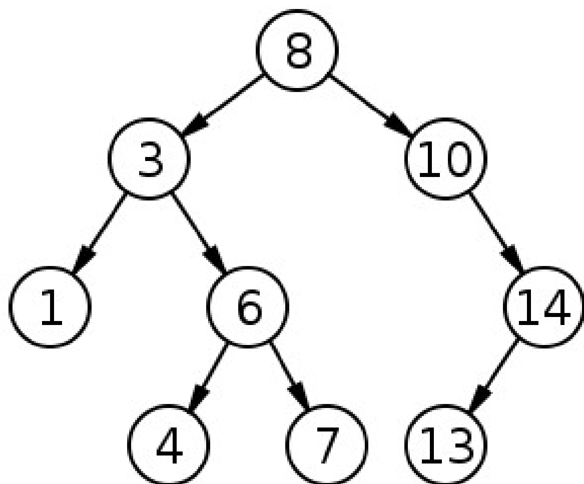
Program prikazuje izgradnjo binarnega drevesa v jeziku C.
Uporablja dinamično alokacijo pomnilnika, kazalce in rekurzijo.

V drevo vstavljamo naključne vrednosti
Na koncu z rekurzijo izpišemo urejeno vsebino drevesa.

Binarna iskalna drevesa

Binarno iskalno drevo (binary search tree ,BST) je urejeno binarno drevo z naslednjimi lastnostmi:

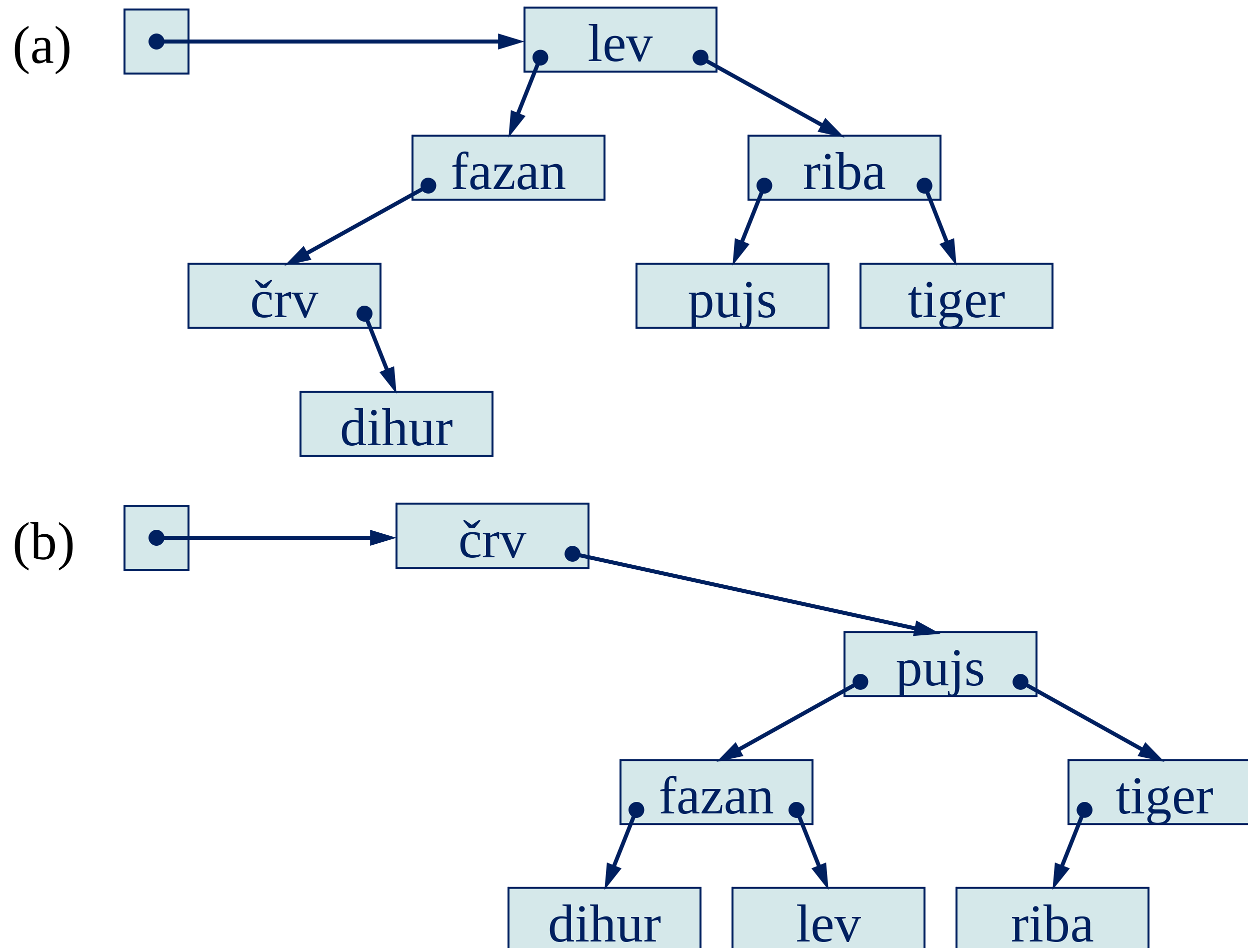
- Levo poddrevo nekega vozla vsebuje le vozle z vrednostmi ključev manjšimi od vrednosti ključa tega vozla.
- Desno poddrevo nekega vozla vsebuje le vozle z vrednostmi ključev, ki so večje od vrednosti ključa tega vozla.
- Tako levo kot desno poddrevo morata spet biti binarni iskalni drevesi.



bstTest.c
Bst.c
Bst.h

DEMO

Primeri binarnih iskalnih dreves



Implementacija vozlišča BST drevesa v Javi

```
public class BSTNode {
    protected Comparable element;
    protected BSTNode left, right;

    protected BSTNode (Comparable elem) {
        element = elem;
        left = null;  right = null;
    }
    …
}
```

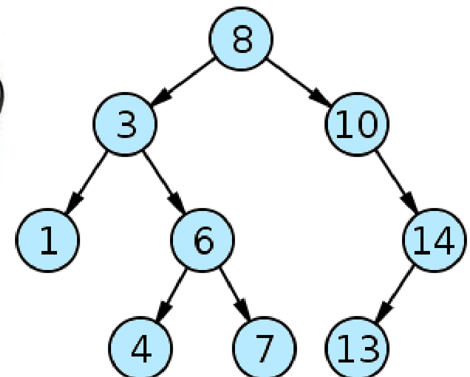
Implementacija BST drevesa v Javi

```
public class BST {
    private BSTNode root;

    public BST () {
    // konstruktor za prazno drevo.
        root = null;
    }
    …
}
```

Iskanje v binarnem iskalnem drevesu

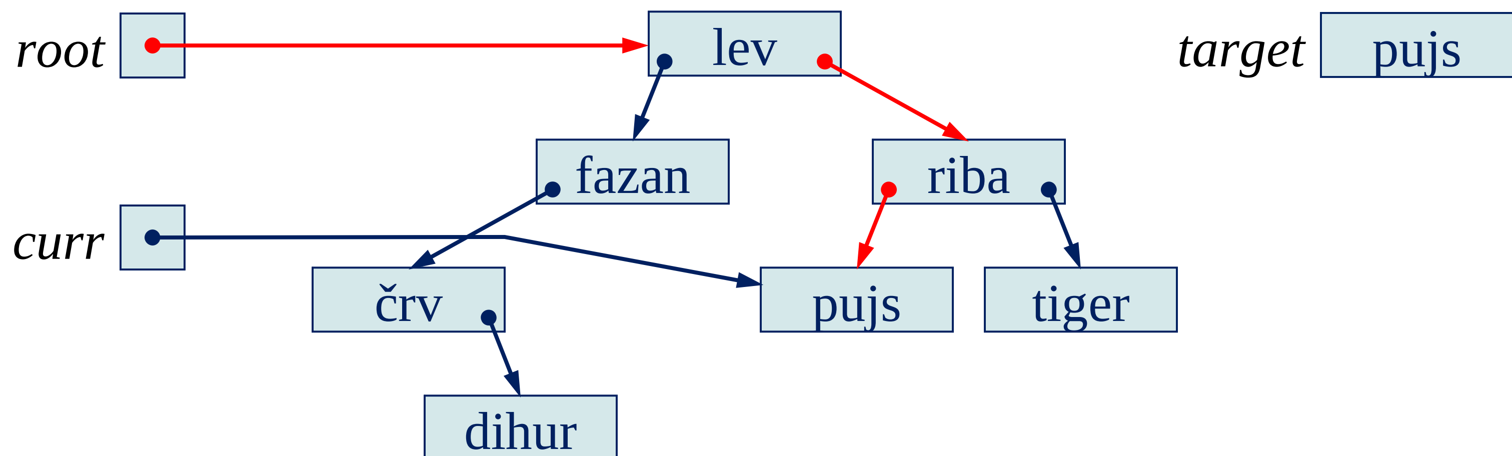
- Problem: V BST iščemo **podano ciljno vrednost**.
- Ideja: Ciljno vrednost primerjamo z vrednostjo elementa v korenu.
  - Če je ciljna vrednost **enaka**, smo iskanje uspešno končali.
  - Če je ciljna vrednost **manjša**, nadaljujemo iskanje v levem poddrevesu.
  - Če je ciljna vrednost **večja**, nadaljujemo iskanje v desnem poddrevesu.
  - Če je poddrevo **prazno**, je iskanje ne



Uspešno iskanje (animacija)

To find which if any node of a BST contains an element equal to *target*:

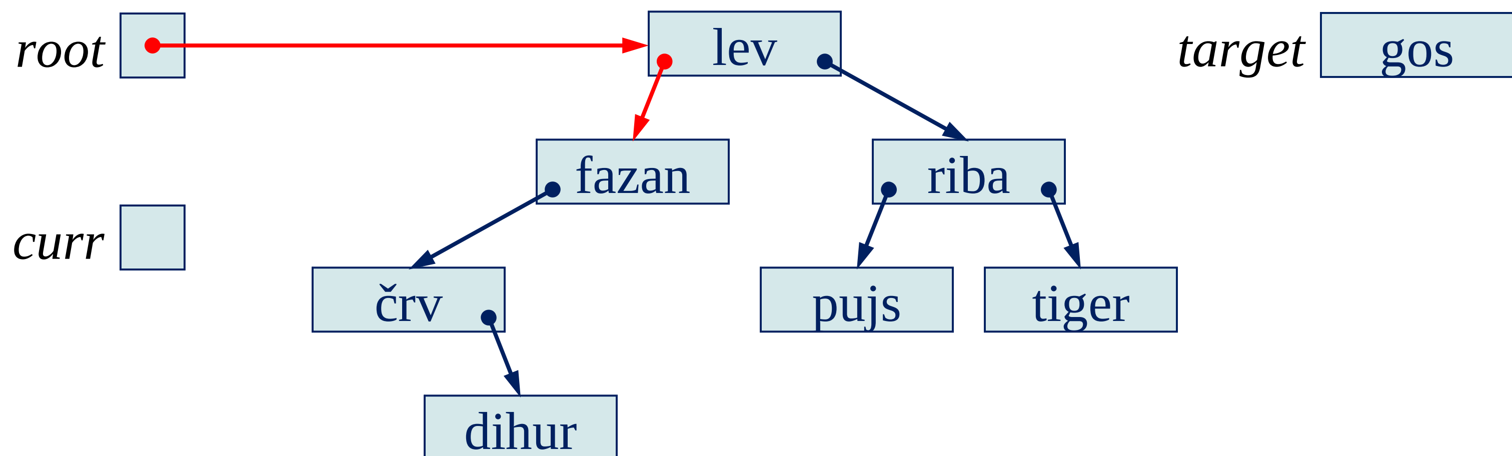
1. Set *curr* to the BST's root.
2. Repeat:
   - 2.1. If *curr* is null, terminate with answer *none*.
   - 2.2. Otherwise, if *target* is equal to *curr*'s element, terminate with answer *curr*.
   - 2.3. Otherwise, if *target* is less than *curr*'s element, set *curr* to *curr*'s left child.
   - 2.4. Otherwise, if *target* is greater than *curr*'s element, set *curr* to *curr*'s right child.



Neuspešno iskanje (animacija)

To find which if any node of a BST contains an element equal to *target*:

1. Set *curr* to the BST's root.
2. Repeat:
   2.1. If *curr* is null, terminate with answer *none*.
   2.2. Otherwise, if *target* is equal to *curr*'s element, terminate with answer *curr*.
   2.3. Otherwise, if *target* is less than *curr*'s element, set *curr* to *curr*'s left child.
   2.4. Otherwise, if *target* is greater than *curr*'s element, set *curr* to *curr*'s right child.



Kompleksnost iskanja

- Analiza (štetje primerjav):

  Naj bo velikost BST *n* (drevo ima n vozlišč).

  Če je d globina BST, je število primerjav največ *d*+1.

- Če je BST uravnovešeno, ima globino floor($\log_2 n$):

  Primerjav je največ floor($\log_2 n$) + 1

  Kompleksnost najboljšega primera je $O(\log n)$.

- Če je drevo neuravnovešeno, ima globino največ n–1:

  Primerjav je največ *n*

  Kompleksnost v najslabšem primeru je is $O(n)$.

Implementacija iskanja v Javi

```
public BSTNode  search (Comparable target) {
    int direction = 0;
    BSTNode curr = root;
    for (;;) {
        if (curr == null)  return null;
        direction =  target.compareTo(curr.element);
        if (direction > 0) curr = curr.right;
        else if (direction < 0)  curr = curr.left;
        else return curr;
    }
}
```

Vstavljanje elementov v drevo

Zamisel:

Nov element vnašamo v BST tako, kot bi v drevesu ta element iskali.

Če elementa še ni, nas iskanje pripelje do povezave null.

To prazno povezavo nadomestimo z listom, ki vsebuje naš element.

Vstavljanje v prazno drevo (animacija)

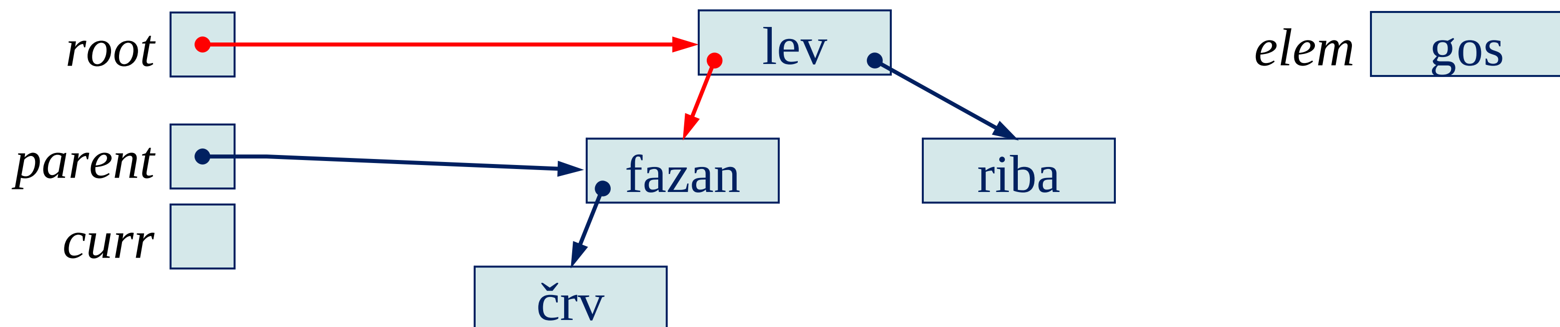
To insert the element *elem* into a BST:

1. Set *parent* to null, and set *curr* to the BST's root.
2. Repeat:
   2.1. If *curr* is null, replace the null link from which *curr* was taken by a link to a newly-created leaf node with element *elem*, and terminate.
   2.2. Otherwise, if *elem* is equal to *curr*'s element, terminate.
   2.3. Otherwise, if *elem* is less than *curr*'s element, set *parent* to *curr* and set *curr* to *curr*'s left child.
   2.4. Otherwise, if *elem* is greater than *curr*'s element, set *parent* to *curr* and set *curr* to *curr*'s right child.

Vstavljanje v neprazno drevo (animacija)

To insert the element *elem* into a BST:

1. Set *parent* to null, and set *curr* to the BST's root.
2. Repeat:
   2.1. If *curr* is null, replace the null link from which *curr* was taken by a link to a newly-created leaf node with element *elem*, and terminate.
   2.2. Otherwise, if *elem* is equal to *curr*'s element, terminate.
   2.3. Otherwise, if *elem* is less than *curr*'s element, set *parent* to *curr* and set *curr* to *curr*'s left child.
   2.4. Otherwise, if *elem* is greater than *curr*'s element, set *parent* to *curr* and set *curr* to *curr*'s right child.



Kompleksnost vstavljanja

- Analiza (štetje primerjav):

  Število primerjav je enako kot pri iskanju v BST.
- Če je BST uravnovešeno:

  Maks. število primerjav= floor($\log_2 n$) + 1

  Kompleksnost je v najboljšem primeru $O(\log n)$.
- Če je BST neuravnovešeno:

  Maks. št. Primerjav = n

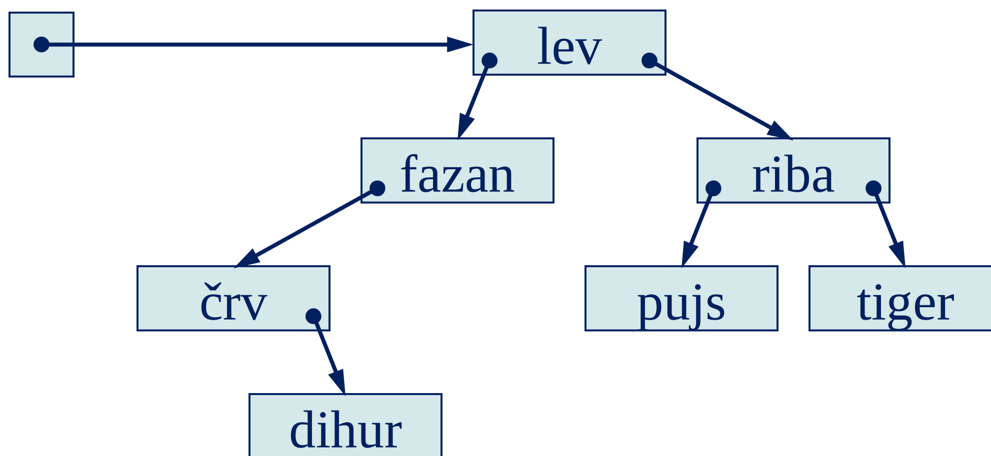
  Kompleksnost je v najslabšem primeru $O(n)$.

Implementacija vstavljanja v Javi

```
public void insert (Comparable elem) {
    int direction = 0;
    BSTNode parent = null, curr = root;
    for (;;) {
        if (curr == null) {
            BSTNode ins = new BSTNode(elem);
            if (root == null)  root = ins;
            else if (direction < 0)
                parent.left = ins;
            else  parent.right = ins;
            return;
        }
        direction = elem.compareTo(curr.element);
        if (direction == 0)  return;
        parent = curr;
        if (direction < 0)  curr = curr.left;
        else  curr = curr.right;
    }
}
```

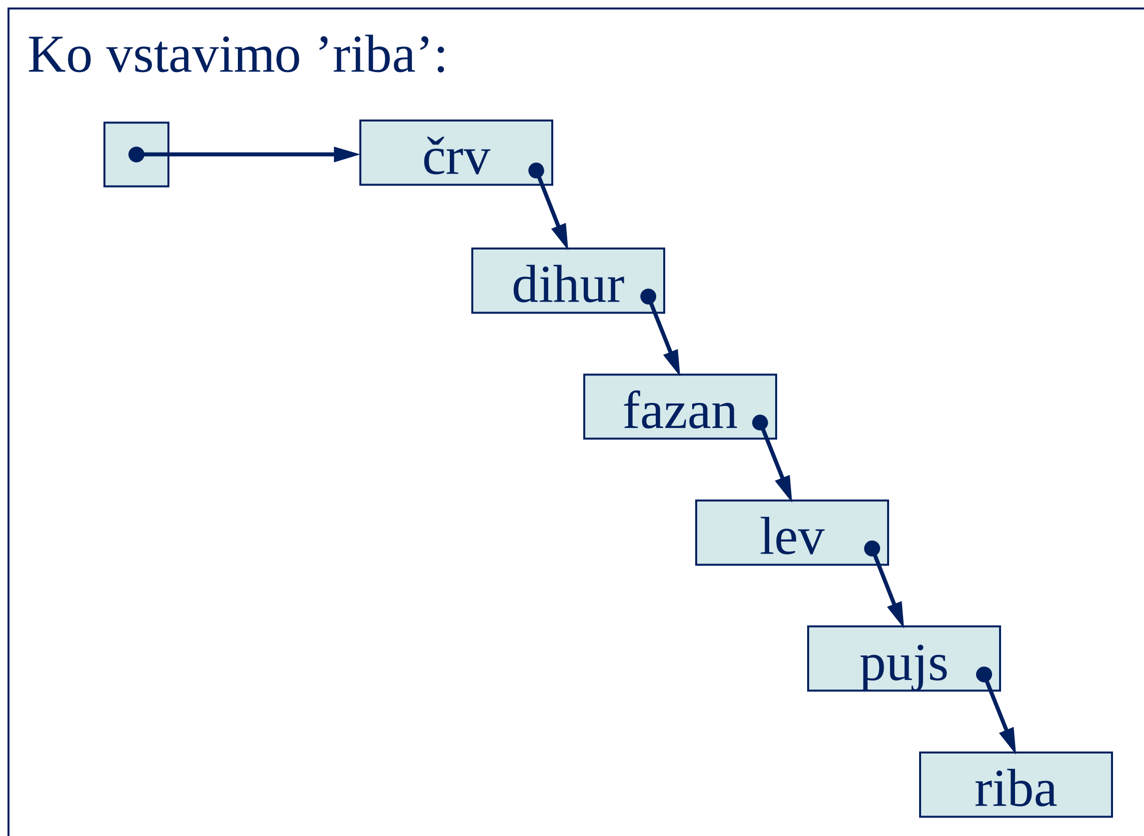
Primer zaporednih vstavljanj

Animacija (vstavljamo ‘lev’, ’fazan’, ’riba’, ’črv’, ’pujs’, ’dihur’, ‘tiger’):



Primer zaporednih vstavljanj (2)

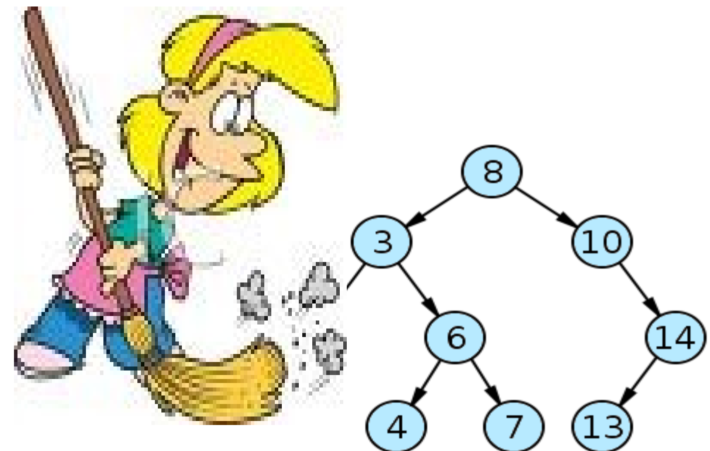
Animacija (vstavljamo ’črv’, ’dihur’, ’fazan’, ‘lev’, ’pujs’, ’riba’):



Brisanje elementov v binarnem iskalnem drevesu

Primeri:

- Brisanje najbolj levega elementa v poddrevesu.
- Brisanje najvišjega elementa v poddrevesu.
- Brisanje poljubnega elementa v poddrevesu.



Brisanje vozla

- Ko brišemo vozel, moramo nekaj narediti z njegovimi otroki.
- Če otrok ni, ni problem. Vozel enostavno zbrišemo.
- Če imamo le levega otroka, spet ni problem; odstranimo vozel in na njegovo mesto postavimo njegovega levega otroka.
- Isto je samo z desnim otrokom: otroka postavimo na mesto brisanega vozla.
- Problem nastane pri brisanju vozla z levim in desnim otrokom. Na mesto brisanega vozla lahko damo tako levega kot desnega otroka, toda kaj naj naredimo z drugim otrokom in njegovim poddrevesom?
- Rešitev je naslednja: poiščemo logičnega naslednika brisanega vozla. Primer: Imejmo drevo s celimi števili in brišemo vozel z vrednostjo 35. Logični naslednik je naslednje večje število. Če bi namreč imeli premi prehod (drevesa), bi bil to element po tistem vozlu, ki ga bomo zbrisali.

Brisanje najbolj levega elementa*(1)*

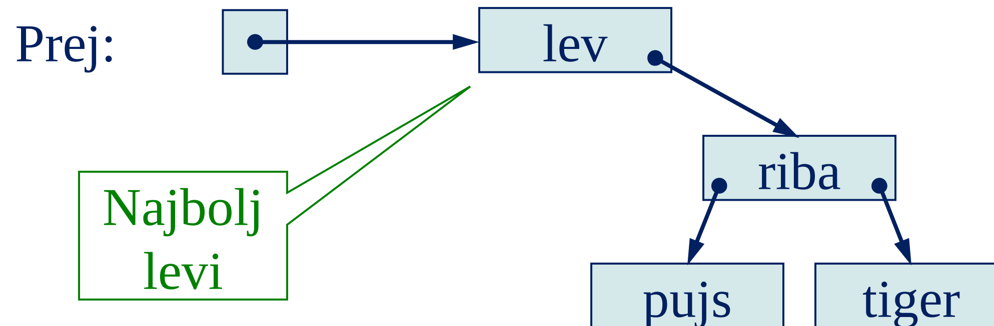
Možna sta dva primera:

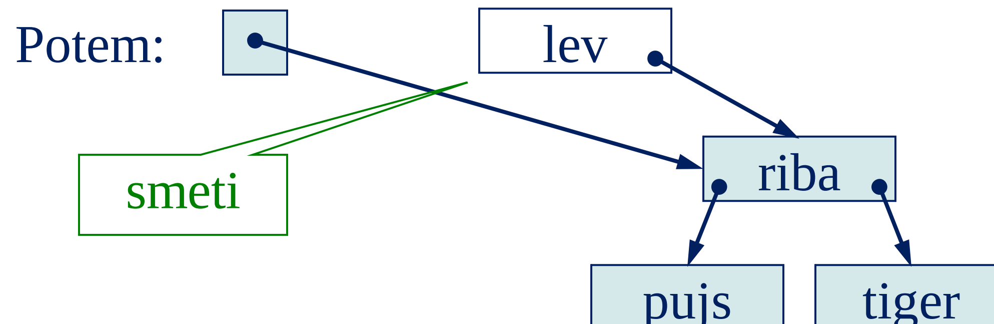
1) Najvišje vozlišče poddrevesa nima levega sina.
2) Najvišje vozlišče poddrevesa ima levega sina.

Opomba: najbolj levo vozlišče po definiciji ne more imeti levega sina.

Brisanje najbolj levega elementa(2)

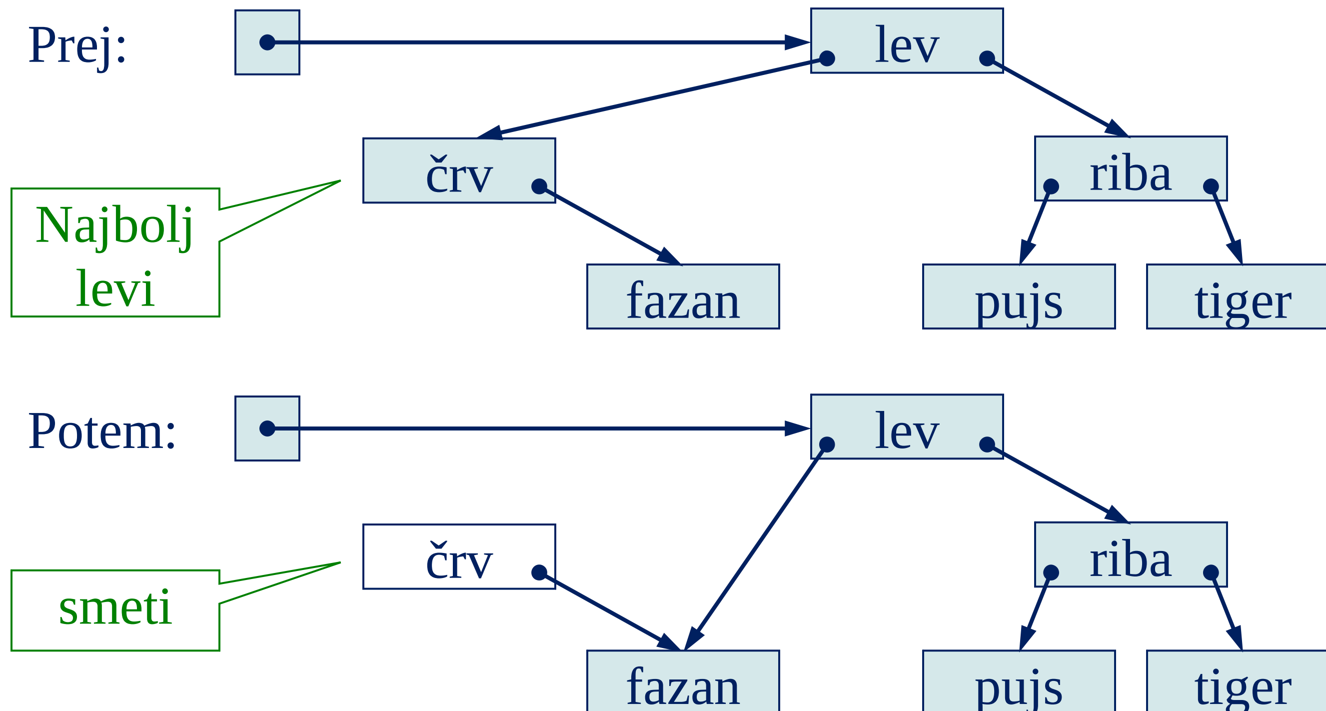
Primer 1 (Najvišji element nima levega sina):
Odstranimo najvišje vozlišče, ohranimo pa njegovo desno poddrevo. Primer:





Brisanje najbolj levega elementa (3)

Primer 2 (Najvišje vozlišče ima levega sina):
Povežemo starš najbolj levega vozlišča z desnim sinom najbolj levega vozlišča. Primer:



Brisanje najbolj levega elementa*(4)*

Algoritem brisanja najbolj levega elementa v nepraznem drevesu, ki ima najvišje vozlišče označeno s *top*:

1. If *top* has no left child:
   1.1. Terminate with *top*'s right child as answer.
2. If *top* has a left child:
   2.1. Set *parent* to *top*, and set *curr* to *top*'s left child.
   2.2. While node *curr* has a left child, repeat:
      2.2.1. Set *parent* to *curr*, and set *curr* to *curr*'s left child.
   2.3. Set *parent*'s left child to *curr*'s right child.
   2.4. Terminate with *top* as answer.

Primer 1

Primer 2

Implementacija metode v Javi

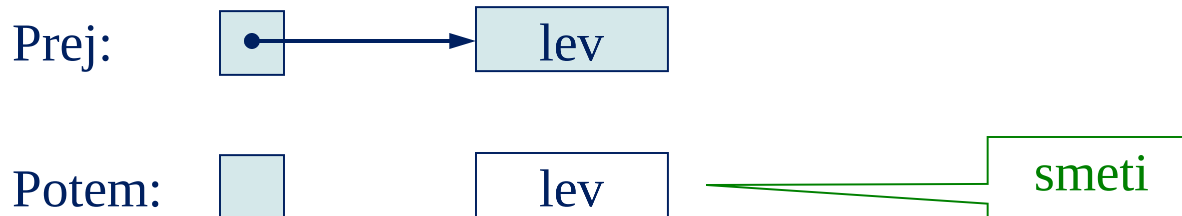
```
private BSTNode deleteLeftmost () {
    if (this.left == null)
        return this.right;
    else {
        BSTNode parent = this, curr = this.left;
        while (curr.left != null) {
            parent = curr;  curr = curr.left;
        }
        parent.left = curr.right;
        return this;
    }
}
```

Brisanje najvišjega elementa *(1)*

- Možni so 4 primeri:
  1) Najvišje vozlišče nima sinov.
  2) Najvišje vozlišče ima samo desnega sina.
  3) Najvišje vozlišče ima samo levega sina.
  4) Najvišje vozlišče ima levega in desnega sina.

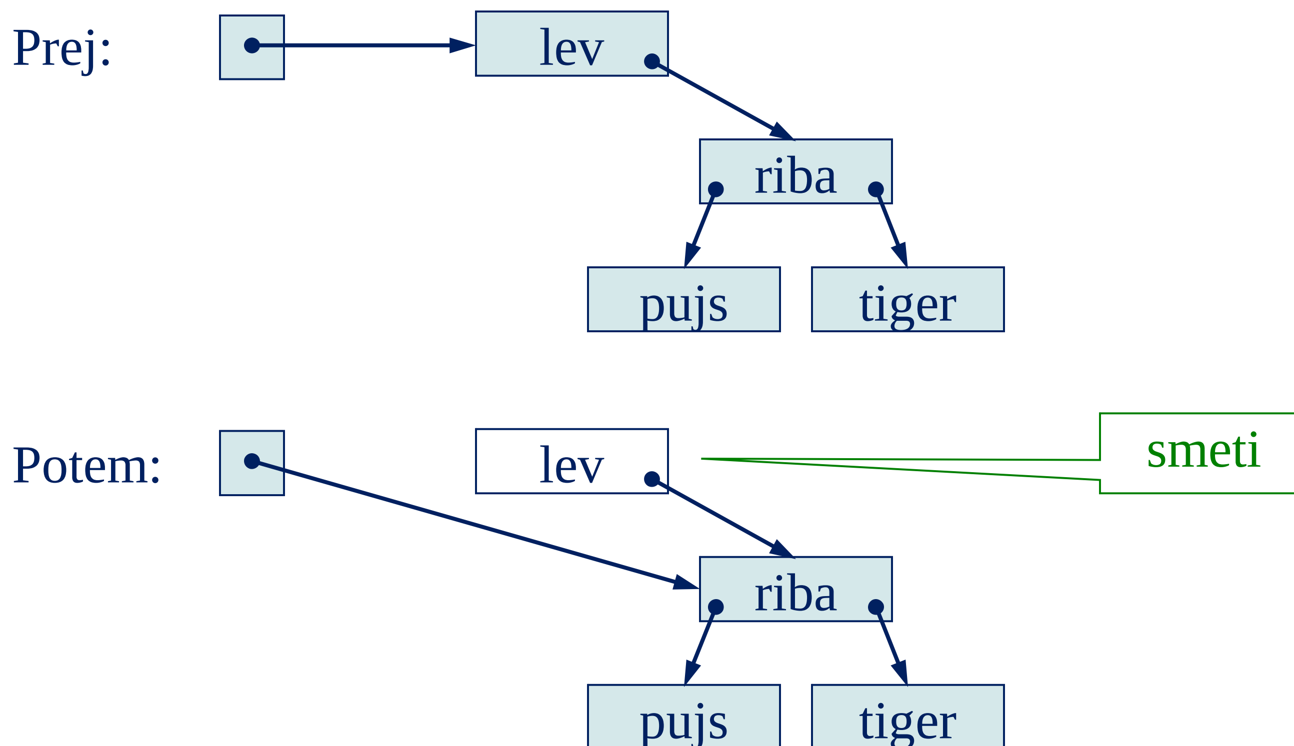
Brisanje najvišjega elementa (2)

Primer 1 (Najvišje vozlišče nima sinov):
Poddrevo izpraznimo. Primer:



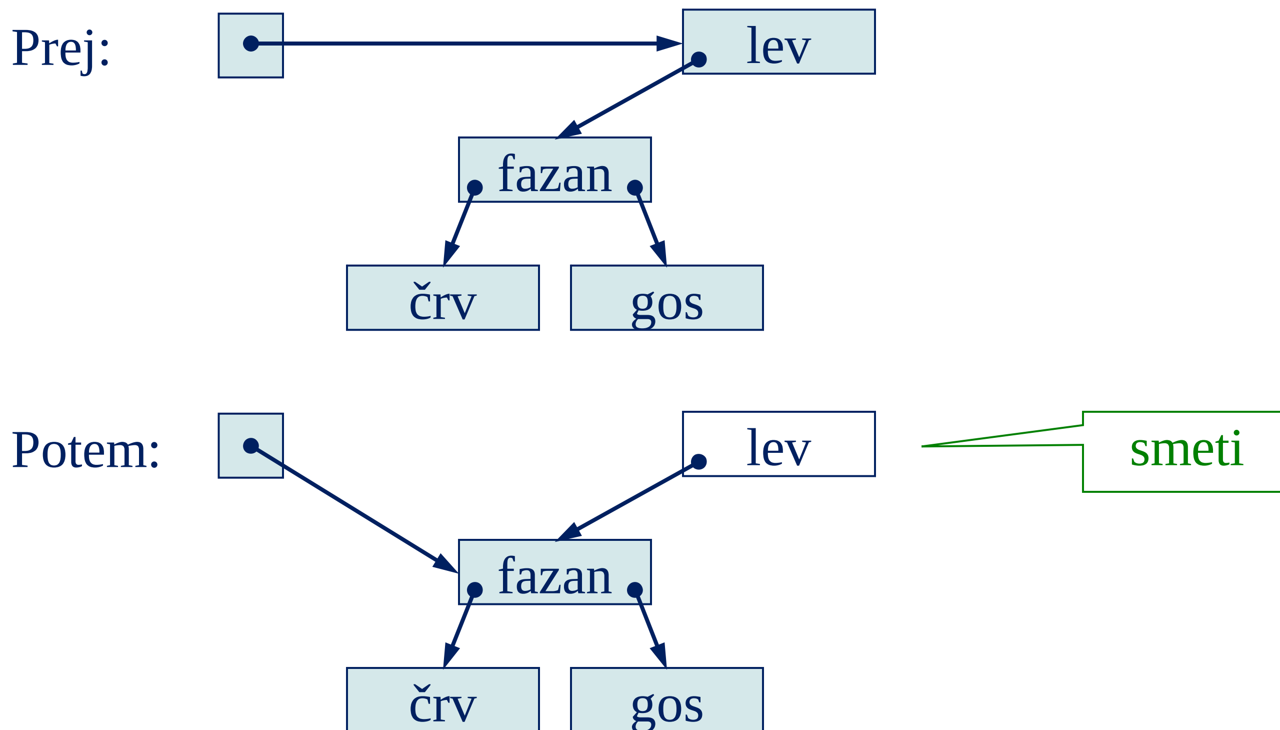
Brisanje najvišjega elementa (3)

Primer 2 (Najvišje vozlišče ima le desnega sina):
Najvišje vozlišče odstranimo, ohranimo pa njegovo desno poddrevo.
Primer.:



Brisanje najvišjega elementa *(4)*

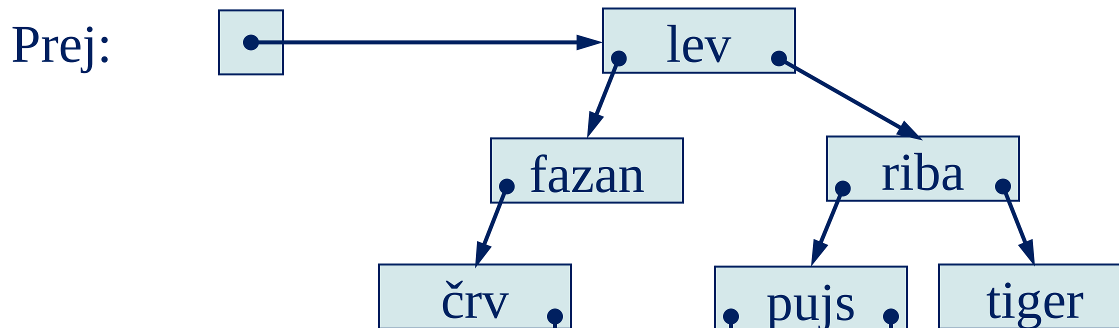
Primer 3 (Najvišje vozlišče ima le levega sina):
Odstranimo najvišje vozlišče, ohranimo poa njegovo levo poddrevo. Primer:

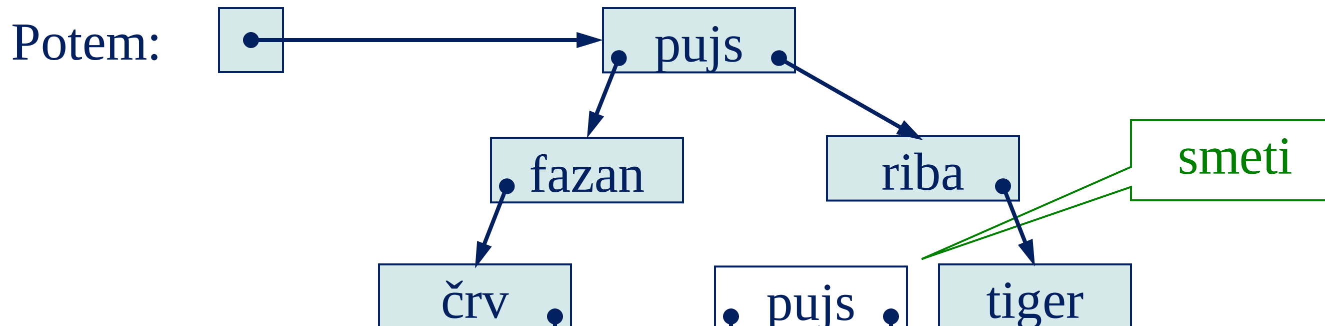


Brisanje najvišjega elementa *(5)*

Primer 4 (Najvišje vozlišče ima dva sinova):

Najbolj levi element desnega poddrevesa kopiramo v najvišje vozlišče. Nato zbrišemo najbolj levi element desnega poddrevesa. Primer.:





Brisanje najvišjega elementa *(6)*

- Algoritem:

Za brisanje najvišjega elementa v poddrevesu, katerega najvišje vozlišče je *top*:

1. If *top* has no left child:
   - 1.1. Terminate with *top*'s right child as answer.
2. If *top* has no right child:
   - 2.1. Terminate with *top*'s left child as answer.
3. If *top* has two children:
   - 3.1. Set *top*'s element to the leftmost element in *top*'s right subtree.
   - 3.2. Delete the leftmost element in *top*'s right subtree.
   - 3.3. Terminate with *top* as answer.

primera 1, 2

primera 1, 3

primer 4

Brisanje najvišjega elementa (7)

- Pomožen algoritem:

Določanje najbolj levega elementa v nepraznem poddrevesu, katerega najvišji element je *top*:

1. Set *curr* to *top*.
2. While *curr* has a left child, repeat:
   2.1. Set *curr* to *curr*'s left child.
3. Terminate with *curr*'s element as answer.

Brisanje najvišjega elementa (8)

Implementacija metode v Javi:

```
public BSTNode deleteTopmost () {
if (this.left == null)
    return this.right;
else if (this.right == null)
    return this.left;
else {  // to vozlisce ima dva sinova
    this.element = this.right.getLeftmost();
    this.right = this.right.deleteLeftmost();
    return this;
}
}
```

Pomožna metoda

```
private Comparable getLeftmost () {
    BSTNode curr = this;
    while (curr.left != null)
        curr = curr.left;
    return curr.element;
}
```

Brisanje danega elementa (algoritem)

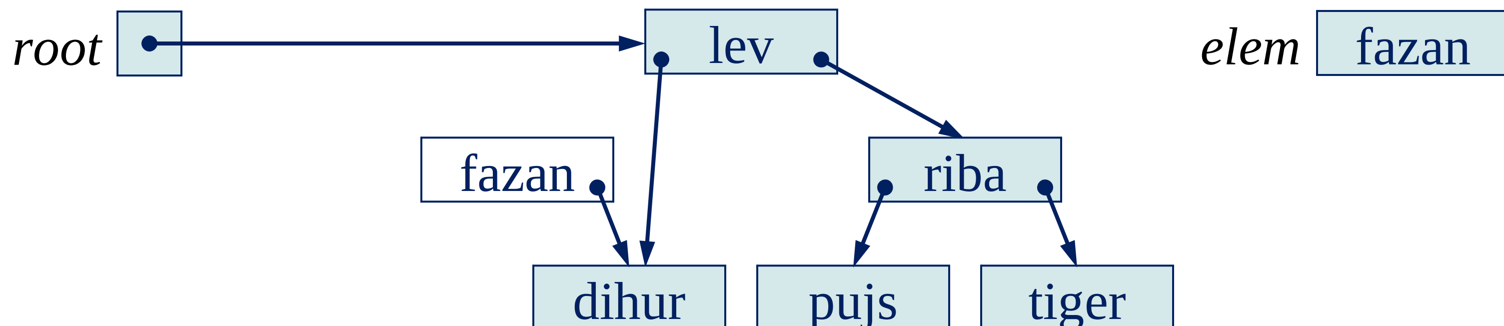
Brišemo element *elem*

1. Set *parent* to null, and set *curr* to the BST's root node.
2. Repeat:
   2.1. If *curr* is null:
      2.1.1. Terminate.
   2.2. Otherwise, if *elem* is equal to *curr*'s element:
      2.2.1. Delete the topmost element in the subtree whose topmost node is *curr*, and let *del* be a link to the resulting subtree.
      2.2.2. Replace the link to *curr* by *del*.
      2.2.3. Terminate.
   2.3. Otherwise, if *elem* is less than *curr*'s element:
      2.3.1. Set *parent* to *curr*, and set *curr* to *curr*'s left child.
   2.4. Otherwise, if *elem* is greater than *curr*'s element:

Brisanje danega elementa (animacija)

To delete the element *elem* in a BST:

1. Set *parent* to null, and set *curr* to the BST's root node.
2. Repeat:
   2.1. If *curr* is null, terminate.
   2.2. Otherwise, if *elem* is equal to *curr*'s element:
      2.2.1. Delete the topmost element in the subtree whose topmost node is *curr*, and let *del* be a link to the resulting subtree.
      2.2.2. Replace the link to *curr* by *del*.
      2.2.3. Terminate.
   2.3. Otherwise, …



Implementacija brisanja danega elementa v Javi

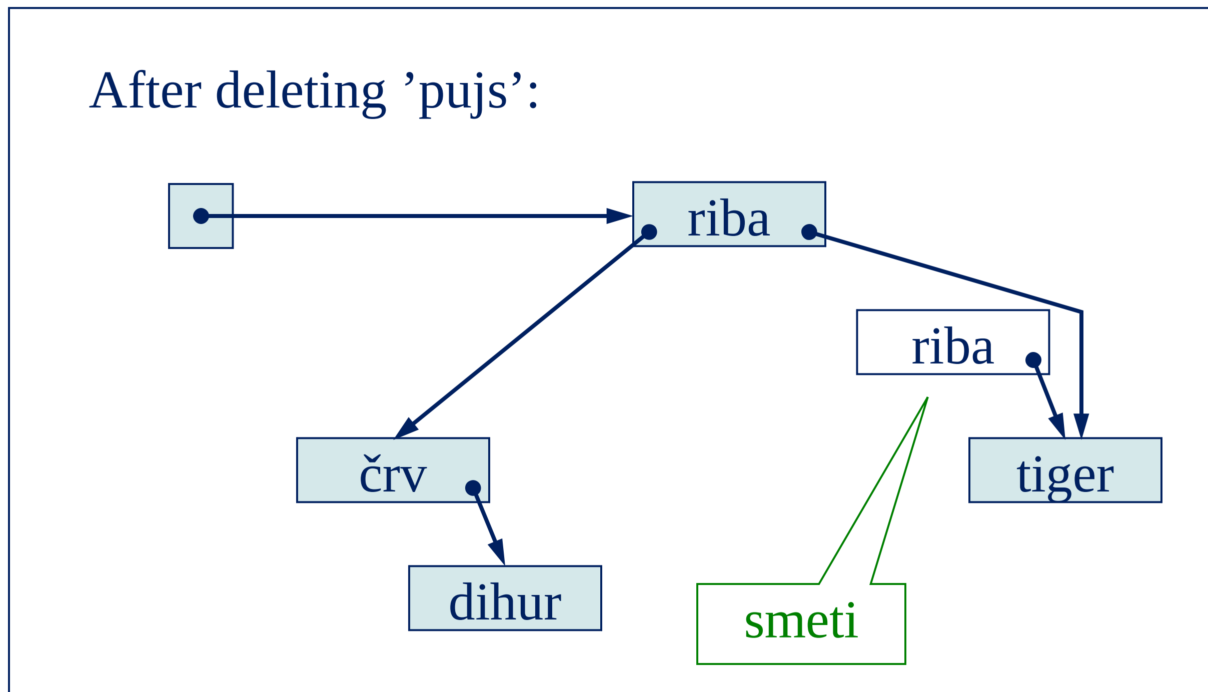
```
public void delete (Comparable elem) {
    int direction = 0;
    BSTNode parent = null, curr = root;
    for (;;) {
        if (curr == null)  return;
        direction = elem.compareTo(curr.element);
        if (direction == 0) {
            BSTNode del = curr.deleteTopmost();
            if (curr == root)  root = del;
            else if (curr == parent.left)
                parent.left = del;
            else  parent.right = del;
            return;
        }
        parent = curr;
        if (direction < 0)
            curr = parent.left;
        else  // direction > 0
            curr = parent.right;
    }
}
```

Brisanje BST v praksi

- Ali je BST uravnovešeno ali neuravnovešeno, je odvisno od vrstnega red vnašanj in brisanj.
- Brisanja lahko uravnovešeno drevo spremene v neuravnovešeno in obratno.

Primer: zaporedna brisanja

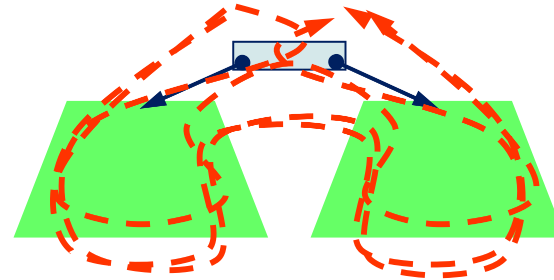
Animacija (brisanje ‘lev, ’fazan’, ’pujs’):



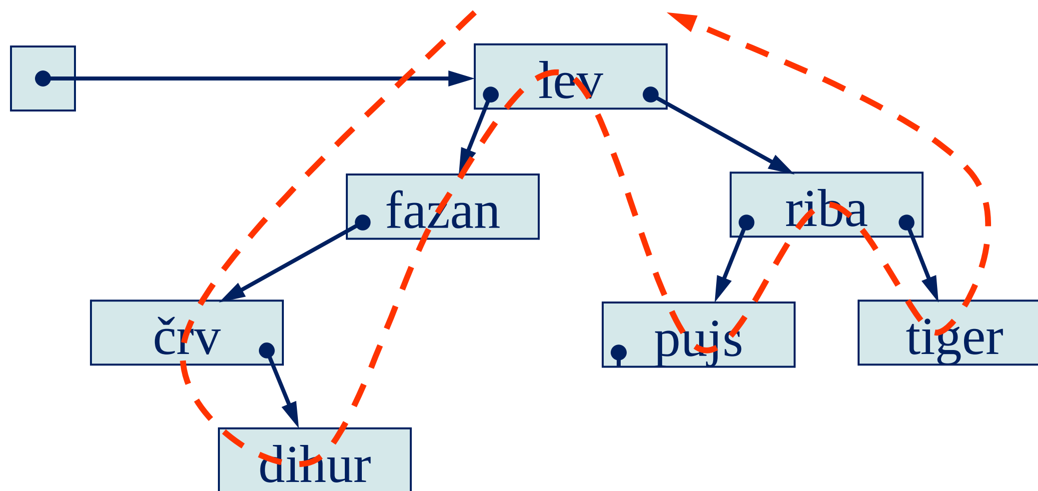
Prehod po binarnem drevesu

- **Prehod po binarnem drevesu**: Vsa vozlišča (elemente) drevesa obiščemo v določenem vrstnem redu.

- Premi prehod,
- Obratni prehod
- Vmesni prehod



Primer vmesnega prehoda



Pri vmesnem prehodu obiskujemo elemente v naraščajočem zaporedju.

Algoritem vmesnega prehoda

To traverse, in in-order, the subtree whose topmost node is *top*:

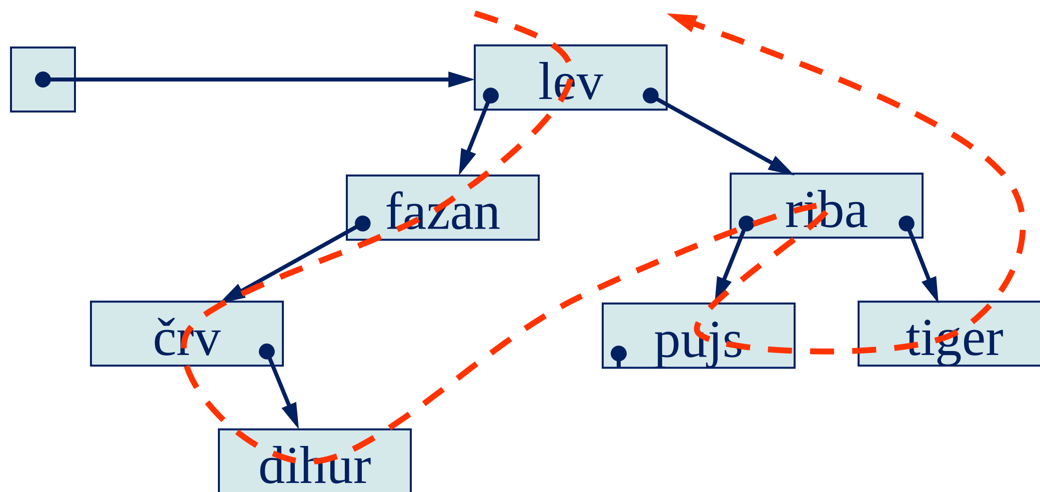
1. If *top* is not null:
   1.1. Traverse, in in-order, *top*'s left subtree.
   1.2. Visit *top*.
   1.3. Traverse, in in-order, *top*'s right subtree.
2. Terminate.

Primer: izpis elementov v premem prehodu

```
public static void printInOrder (BSTNode top) {
// Print, in ascending order, all the elements in the BST subtree
// whose topmost node is top.
    if (top != null) {
        printInOrder(top.left);
        System.out.println(top.element);
        printInOrder(top.right);
    }
}
```

Obiščemo top (Izpis elementa).

Primer premega prehoda

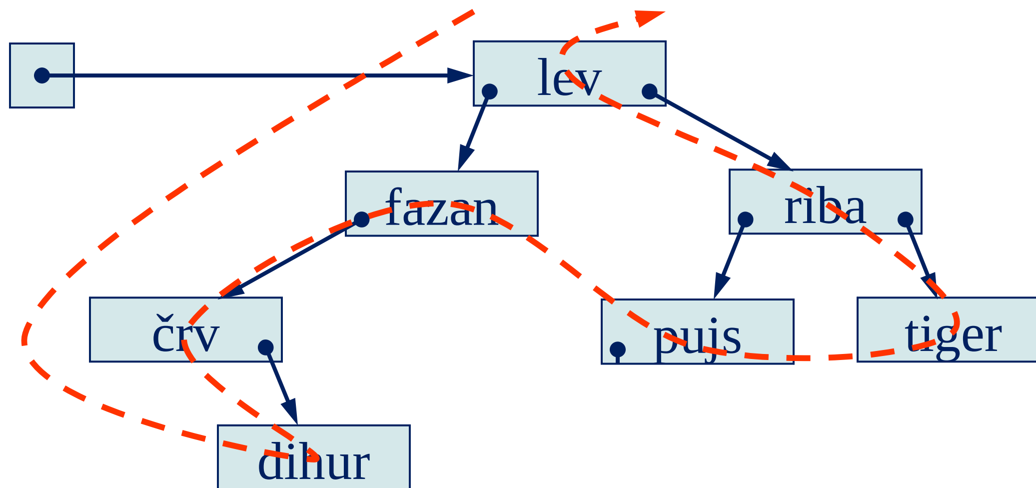


Algoritem premega prehoda

// najvišje vozlišče poddrevesa je *top*:

1. If *top* is not null:
    1.1. Visit *top*.
    1.2. Traverse, in pre-order, *top*'s left subtree.
    1.3. Traverse, in pre-order, *top*'s right subtree.
2. Terminate.

Perimer obratnega prehoda



Algoritem obratnega prehoda

// najvišje vozlišče poddrevesa je *top*:

1. If *top* is not null:
   1.1. Traverse, in post-order, *top*’s left subtree.
   1.2. Traverse, in post-order, *top*’s right subtree.
   1.3. Visit *top*.
2. Terminate.

Časovna kompleksnost algoritmov

Operacija	**Algoritem**	**Časovna kompleksnost**	
`contains`	BST iskanje	$O(\log n)$ $O(n)$	best worst
`add`	BST vnašanje	$O(\log n)$ $O(n)$	best worst
`remove`	BST brisanje	$O(\log n)$ $O(n)$	best worst

Več o binarnih drevesih

- Sekcija 1. Struktura binarnega drevesa – hiter uvod v binarna drevesa in kodo operacij na njih
- Sekcija 2. Problemi z binarnimi drevesi – praktični problemi v naraščajočem vrstnem redu po težavnosti
- Sekcija 3. C rešitve – koda rešitev problemov za programerje v C in C++
- Sekcija 4. Java verzija – koda rešitev problemov za programerje v Javi

Uravnotežena drevesa

Faktor uravnovešenosti (**balance factor)** vozla je višina njegovega levega poddrevesa minus višina desnega poddrevesa.

Vozel s faktorjem uravnovešenosti enakim 1, 0, ali −1 je uravnovešen. Vozel z drugačnim faktorjem uravnovešenosti je neuravnovešen in terja ponovno uravnovešenje drevesa. Faktor uravnovešenosti je lahko shranjen v vsak vozel ali pa se računa iz višine poddreves

AVL drevesa — Wiki — Wiki-SLO — WEB — DEMO

Rdeče črna drevesa — Wiki — Wiki-SLO — WEB — DEMO

Nekaj zanimivih povezav

Dober tutorial o AVL drevesih:
http://facultyfp.salisbury.edu/despickler/personal/Resources/AdvancedDataStructures/Handouts/AVL_TREES.pdf

Simulacija različnih binarnih dreves, vizualizacija kode:
http://groups.engin.umd.umich.edu/CIS/course.des/cis350/treetool/

Lepa, kvalitetna animacija AVL dreves:
http://www.qmatica.com/DataStructures/Trees/AVL/AVLTree.html

MatrixPro (orodje za učenje algoritmov in struktur)
http://www.cse.hut.fi/en/research/SVG/MatrixPro/

DEMO

B drevesa

B-drevesa sta uvedla Bayer in McCreight leta 1971, medtem ko sta delala v Boeing Research Labs. Toda avtorja nista nikoli navedla izvor črke B.
Ta črka bi lahko stala kot začetnica naslednjih besed: "balanced", "broad", "bushy", "Boeing" ali celo bolj pravično po avtorju "Bayer-trees".
B-drevesa so uravnotežena iskalna drevesa za delo na diskih ali drugih zunanjih pomnilniških napravah.
So podobna rdeče-črnim drevesom, toda bolje minimizirajo diskovne vhodno-izhodne operacije.

Tutorial: http://www.nauk.si/materials/4676/out/#state=1

Tutorial: http://www.bluerwhite.org/btree/

Applet: http://slady.net/java/bt/view.php
Applet: http://ats.oka.nu/b-tree/b-tree.manual.html

Primer kode v C: http://www.indiastudychannel.com/resources/13022-C-Program-insertion-deletion-B-tree.aspx

Primer kode v Javi:
http://www.koders.com/java/fid0049BA25AB502309753D2558C4661215CDC8A3A7.aspx