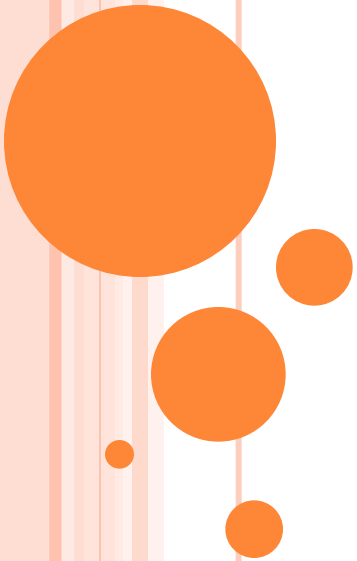


POPOLNOMA PORAVNANA DREVESA

2-3-drevesa (samo informativno)

B-drevesa

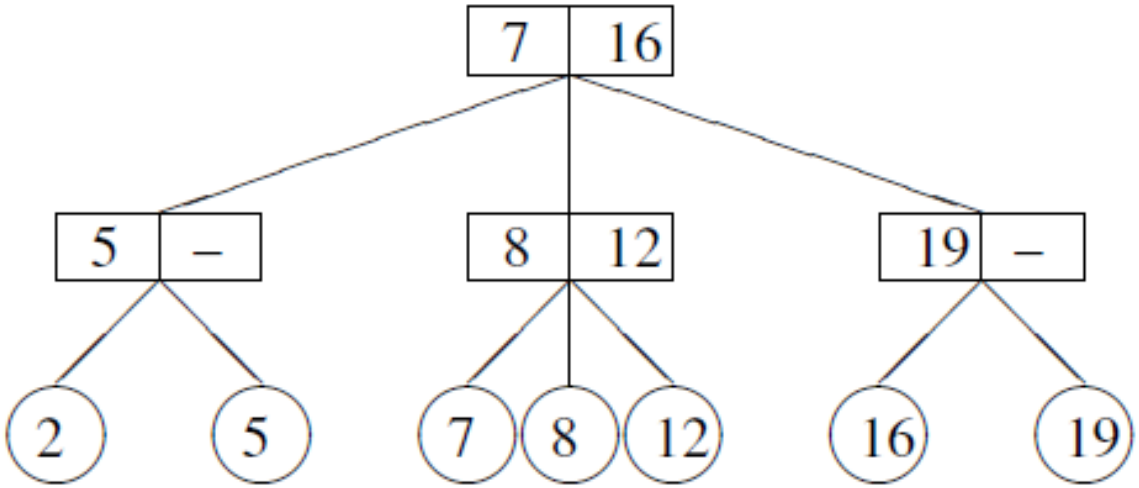
2-3 DREVESA

2-3 drevesa so

- popolnoma poravnana iskalna drevesa
- vsako vozlišče ima 2 ali 3 sinove in 1 ali 2 ključa
- 2 vrsti 2-3 dreves:
  1. Vsi elementi se nahajajo v listih: ključ v notranjem vozlišči je enak najmanjšemu ključu v desnem poddrevesu
  2. Elementi se nahajajo v notranjih vozliščih (listi so prazni): ključ je večji od vseh v levem in manjši od vseh v desnem poddrevesu – posplošitev BST na več ključev in poddreves (to so B-drevesa reda 3)
- vse operacije so reda $O(\log n)$

2-3 DREVESA

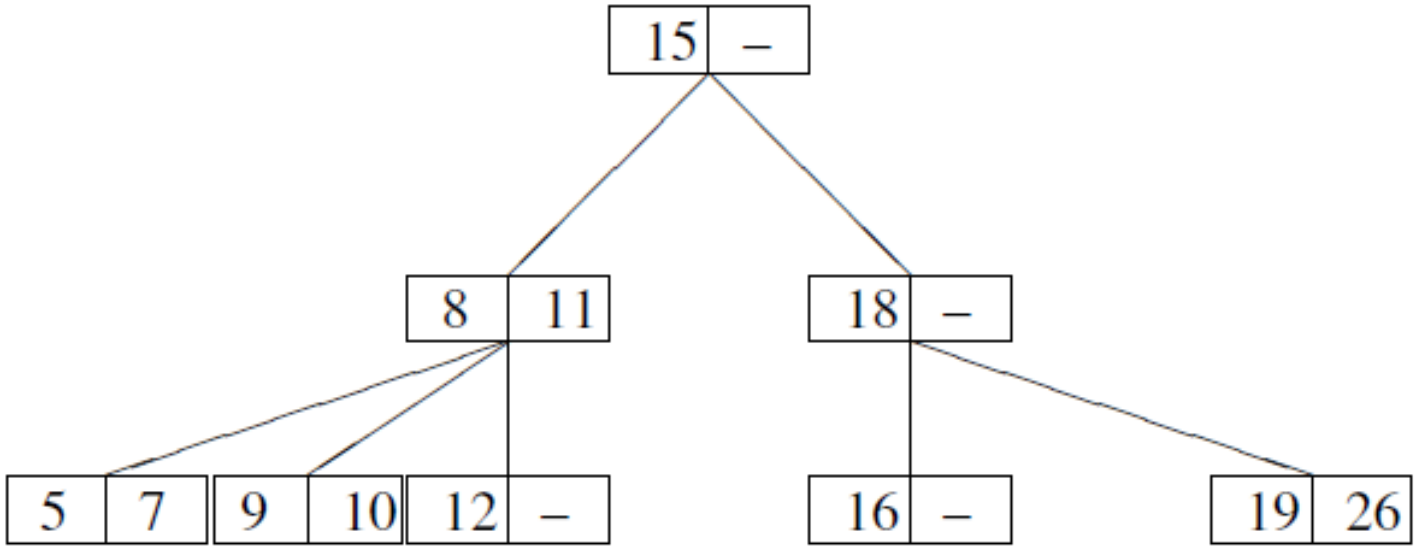
1. Vsi elementi se nahajajo v listih: ključ v notranjem vozlišči je enak najmanjšemu ključu v desnem poddrevesu

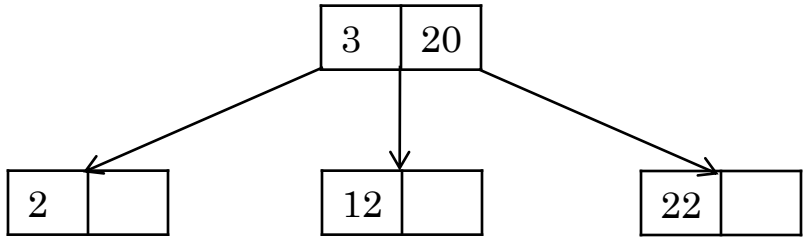


2-3 DREVESA

2. Elementi se nahajajo v notranjih vozliščih (listi so prazni): ključ je večji od vseh v levem in manjši od vseh v desnem poddrevesu – posplošitev BST na več ključev in poddreves, ki pa so VEDNO popolnoma poravnana!

(to so B-drevesa reda 3)





B-DREVO
(angl. *B-tree*)

B-DREVO

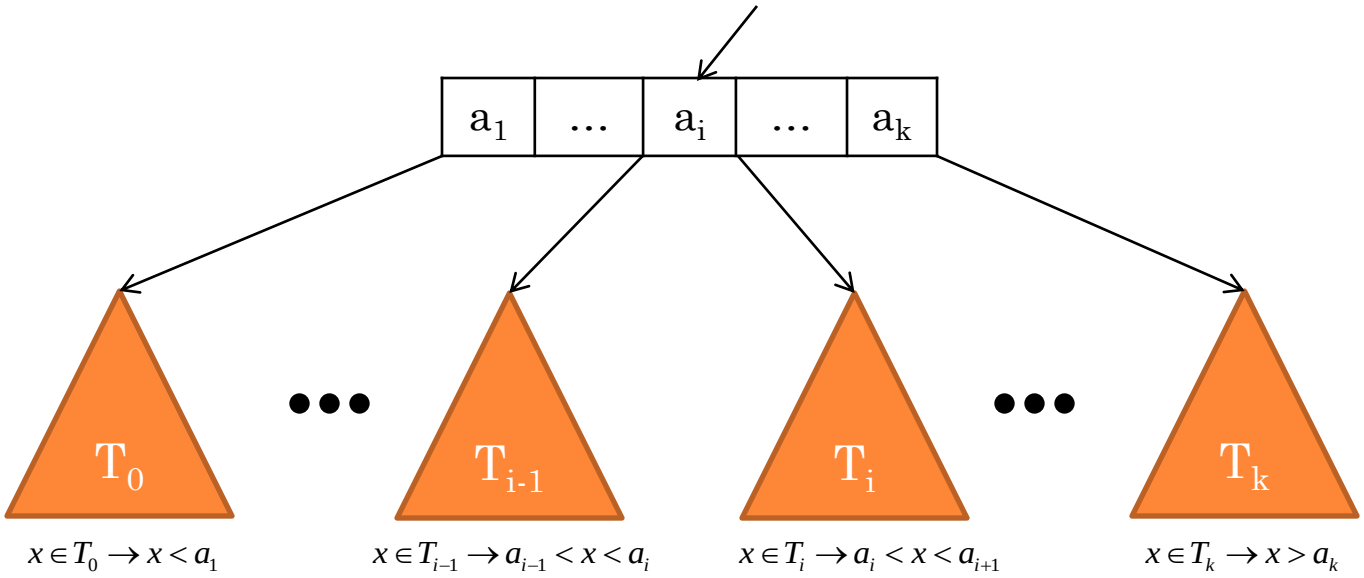
- B-drevo je popolnoma poravnano iskalno drevo – vsi listi so na istem nivoju

- vsako notranje vozlišče B-drevesa reda m ima lahko od $\lceil m/2 \rceil$ do m sinov ter en ključ manj kot ima sinov

- izjema je koren drevesa – v primeru, da je koren notranje vozlišče, ima lahko od 2 do m sinov in en ključ manj

- elementi so shranjeni v notranjih vozliščih
  (vsi listi so prazna poddrevesa)

B-DREVO



- posplošitev binarnih iskalnih dreves
- poravnano binarno iskalno drevo je B-drevo reda 2
- vsako B-drevo reda m, ki vsebuje n ≥ 1 elementov, ima višino največ:

$$h \leq \log_{\lceil m/2 \rceil}\left(\frac{n+1}{2}\right)+1$$

Zunanji pomnilnik

- velikost glavnega pomnilnika ni zadostna za hranjenje velikih baz podatkov – za ta namen potrebujemo zunanji pomnilnik (npr. trdi disk)

- zunanji pomnilnik je cenejši toda veliko počasnejši od glavnega pomnilnika

- zaradi relativne počasnosti želimo minimizirati število dostopov do zunanjega pomnilnika

- B-drevo je primer iskalnega drevesa, ki je namenjeno optimizaciji števila dostopov do zunanjega pomnilnika

B-DREVO

- 2-3 drevesa (2. vrste) so B-drevesa reda 3

- V praksi se uporablja m = 512, 1024 ali več
  → višina drevesa ≤ 5 = konstanta

- Prva dva (ali celo tri) novoje drevesa hranimo v hitrem pomnilniku → za dostop do enega izmed 100T podatkov potrebujemo samo 2 ali 3 branja iz diska

Implementacija B-drevesa

```
public class BTreeNode {
    int count ; // 0..m1−1 stevilo kljucev v vozliscu
    Comparable keys[] ; // array [0..m1−1]
    BTreeNode children[] ;// array [0..m1]: m1+1 moznih otrok
} // class BTreeNode
```

Operacije na B-drevesu

Vse operacije so garantirano *O(log n)*:

- **Iskanje** je posplošitev iskanja v BST: v vozlišču bodisi najdemo element ali pa ustrezno poddrevo.

- **Dodajanje**: element dodamo v list; če ni dovolj prostora, se list razbije na dva lista z ustreznim ključem, ki ga rekurzivno dodamo očetu.

- **Brisanje:** element zbrišemo tako, da ga nadomestmo z maksimalnim iz desnega (ali minimalnim iz levega) poddrevesa – torej element je vedno zbrisan v listu; če je premalo elementov v listu, ga združimo z bratom in ustreznim ključem, ki ga rekurzivno zbrišemo pri očetu.

Iskanje v B-drevesu

Iskanje elementa je posplošitev iskanja v navadnih binarnih iskalnih drevesih:

1. iskanje začnemo v vozlišču, ki je koren drevesa

2. iskani element zaporedno primerjamo z elementi v vozlišču, dokler:

   - ne naletimo na iskani element
   - ne naletimo na večji element in se iskanje rekurzivno nadaljuje v poddrevesu z istim indeksom
   - ne pregledamo zadnjega elementa in se iskanje rekurzivno nadaljuje v zadnjem poddrevesu

Časovna zahtevnost iskanja je $O(m \log n)$

Iskanje v B-drevesu

Namesto zaporednega primerjanja Elementa s ključi v vozlišču, ki vsebuje mnogo (npr. 1023) ključev, uporabimo **bisekcijo**:

```
Min = indeks prvega (najmanjšega) ključa - 1
Max = indeks zadnjega (največjega) ključa + 1;
ponavljaj
   če Max-Min = 1 potem poddrevo najdeno;
   sicer
   i = (Max-Min)/2;
   če Element < ključ(i) potem Max = i;
   sicer
   če Element > ključ(i) potem Min = i;
   sicer element najden; // ključ(i) == Element
dokler ne najdeš elementa ali ustreznega poddrevesa;
```

Namesto *O(m)* korakov bisekcija zahteva *O(log m)* korakov.
Časovna zahtevnost iskanja elementa v B-drevesu je $O(\log m \cdot \log n)$

Dodajanje v B-drevo

Element dodajamo v ustrezno notranje vozlišče največje globine:

1. če opazovano vozlišče vsebuje manj kot m-1 elementov, dodamo element na ustrezno mesto in končamo

2. če v opazovanem vozlišču ni prostora (ima že m-*1* elementov in z dodajanjem imamo m elementov), ga razbijemo na dve vozlišči:

   - določimo sredinski $\lceil m/2 \rceil$-ti element med m elementi
   - $\lceil m/2 \rceil$-*1* elementov, ki so manjši od sredinskega elementa, damo v novo levo vozlišče
   - $m - \lceil m/2 \rceil$ elementov, ki so večji od sredinskega elementa, damo v novo desno vozlišče
   - sredinski element rekurzivno dodamo očetu prejšnjega vozlišča (rekurzija se konča bodisi pri 1. točki, bodisi dobimo nov koren drevesa)

Časovna zahtevnost dodajanja je vedno $O(\log n)$

Primer (1/13)

Na začetku je B drevo reda 3 prazno. V drevo dodajte naslednje elemente: 2, 19, 3, 12, 18, 5 in 16.

Primer (2/13)

Dodamo 2.

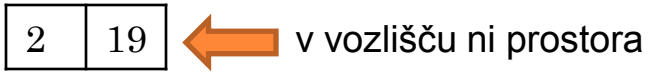
Primer (3/13)

Dodamo 19.

2	19

Primer (4/13)

Dodamo 3...

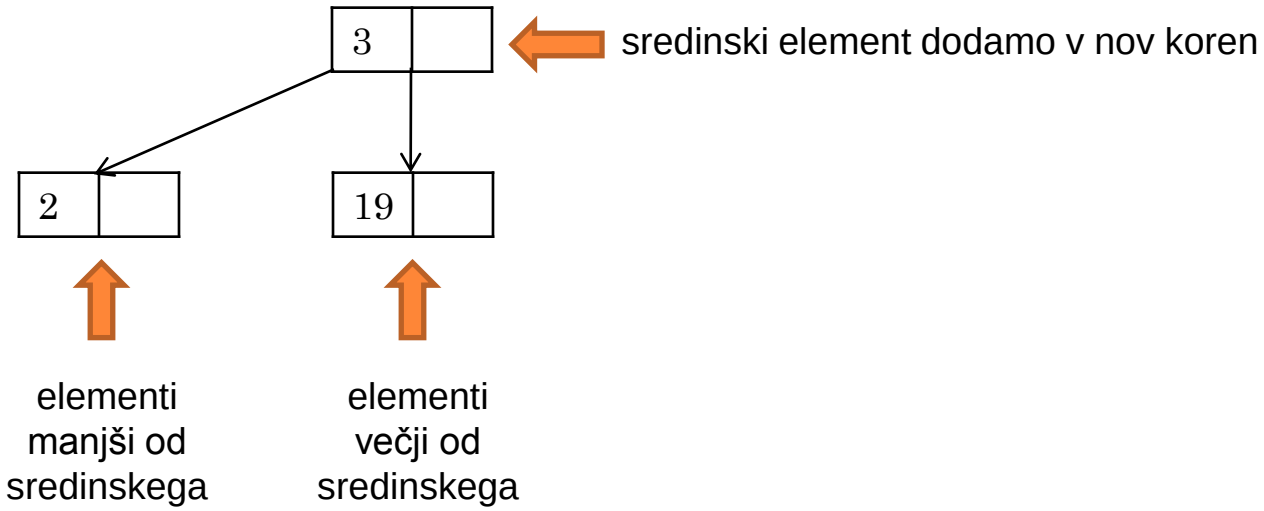


2, 3, 19

sredinski element

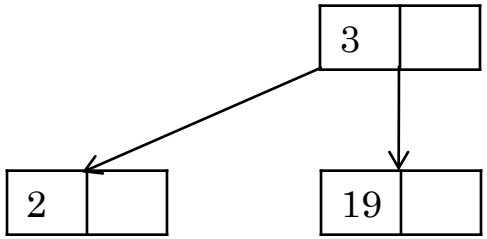
Primer (5/13)

Dodamo 3.



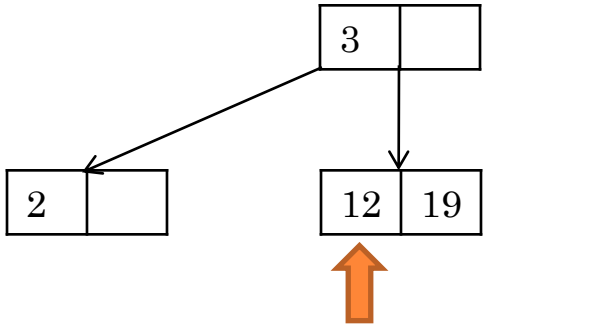
Primer (6/13)

Dodamo 12.



Primer (6/13)

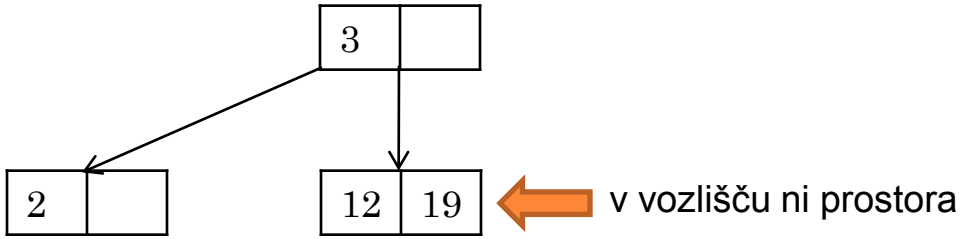
Dodamo 12.



element dodamo v ustrezno vozlišče največje globine

Primer (7/13)

Dodamo 18...

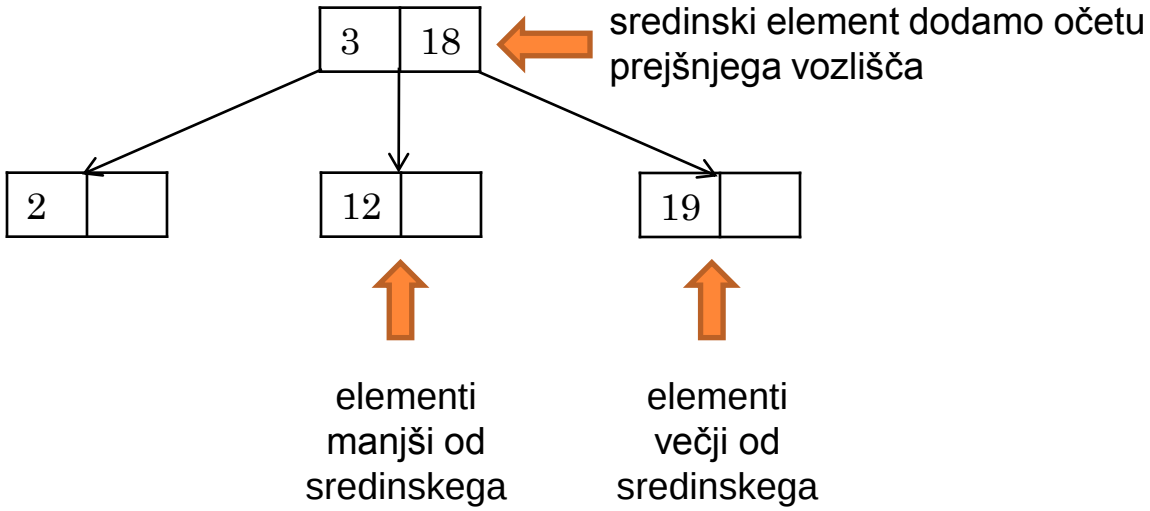


12, 18, 19

sredinski element

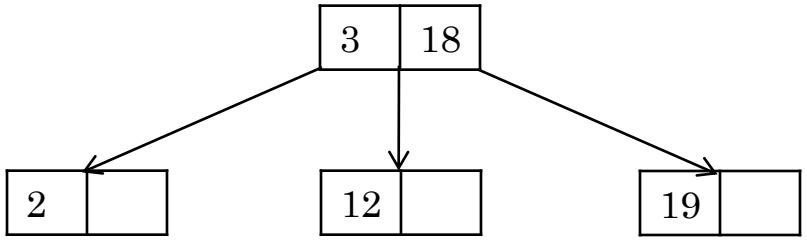
Primer (8/13)

Dodamo 18.



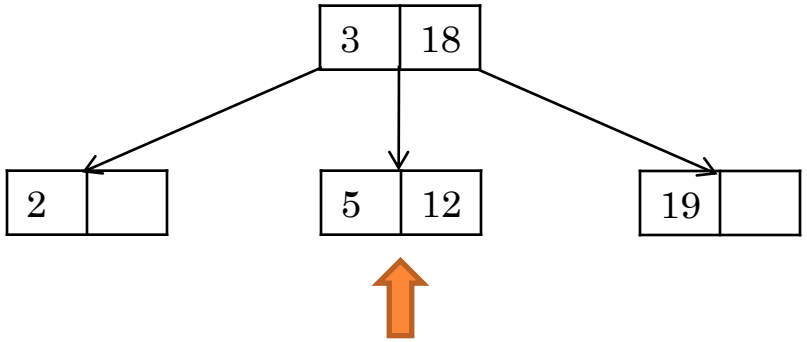
Primer (9/13)

Dodamo 5.



Primer (9/13)

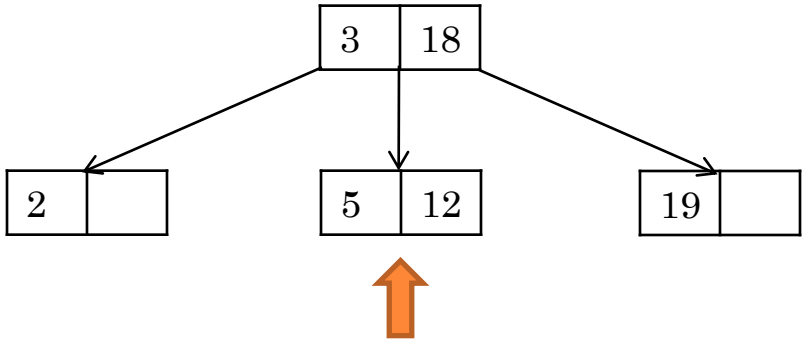
Dodamo 5.



element dodamo v ustrezno vozlišče največje globine

Primer (10/13)

Dodamo 16...



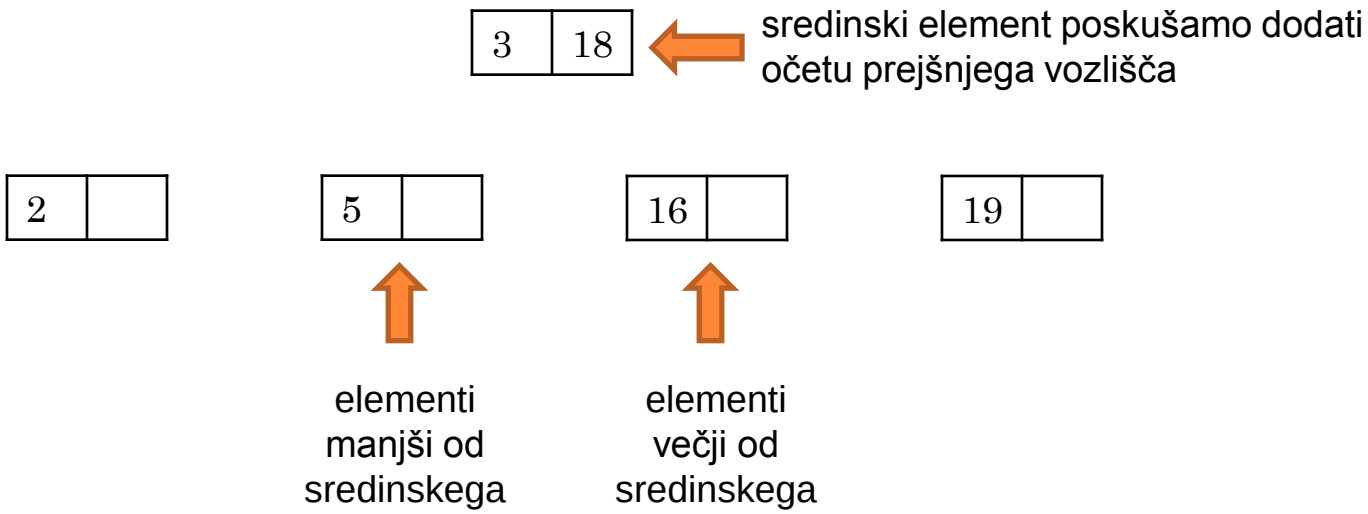
v vozlišču ni prostora

5, 12, 16

sredinski element

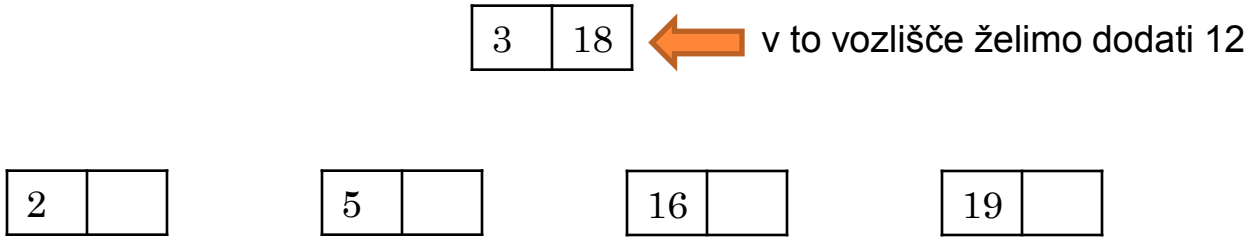
Primer (11/13)

Dodamo 16...



Primer (12/13)

Dodamo 16…

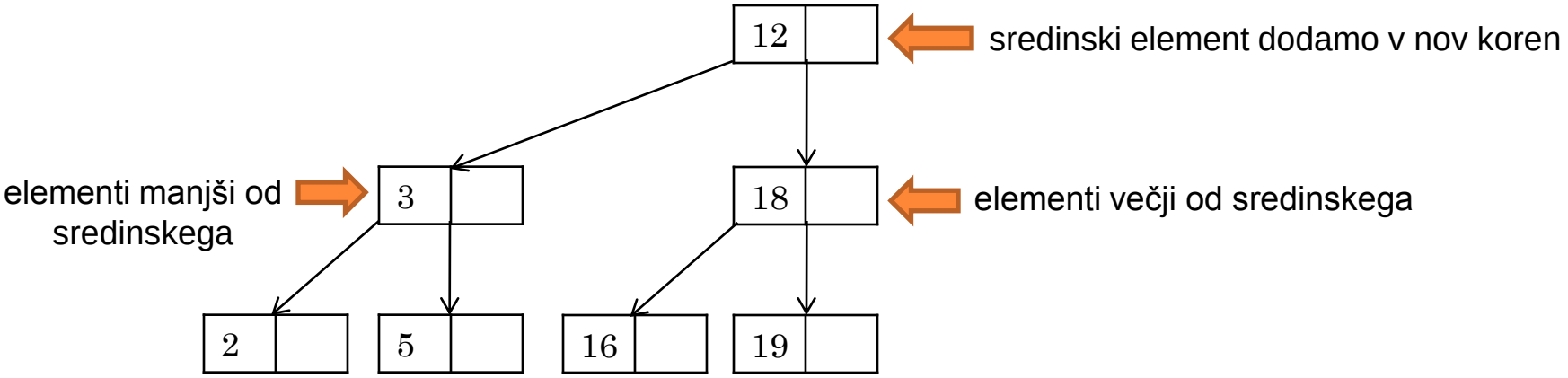


3, 12, 18

sredinski element

Primer (13/13)

Dodamo 16.



Brisanje iz B-drevesa

Vedno brišemo element iz vozlišča **na zadnjem nivoju**:

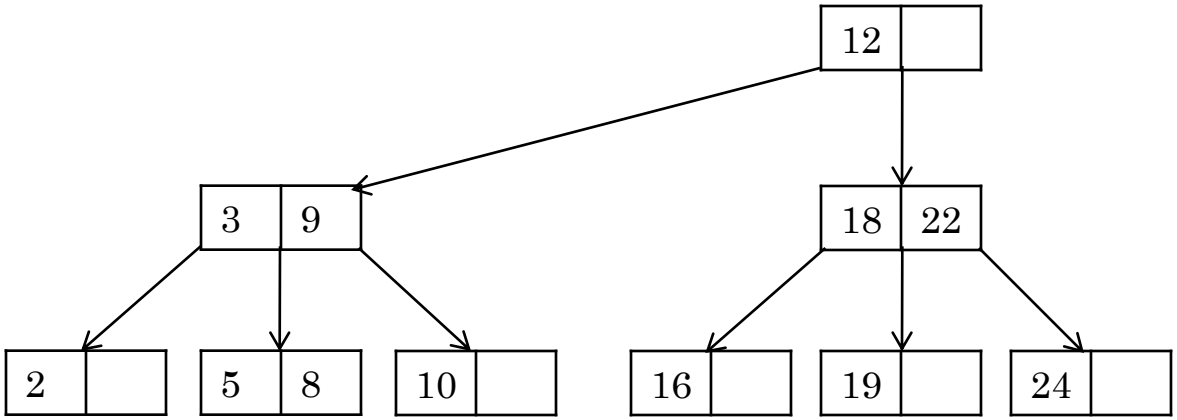
- pri brisanju elementa, ki ni na zadnjem nivoju, ga nadomestimo s predhodnikom (z največjim elementom ustreznega levega poddrevesa), ki ga dejansko zbrišemo

1. Če vozlišče, kjer smo element zbrisali, vsebuje dovoljeno število elementov (vsaj $\lceil m/2 \rceil$ - *1*; za koren zadošča *1*), je postopek **končan**.
2. Če vozlišče sedaj vsebuje premalo ($\lceil m/2 \rceil$ - *2*) elementov, potem:
   a) Eden izmed sosednih bratov (levi ali desni) vsebuje dovolj elementov, da si jih razdelita med seboj – v tem primeru se **od brata vzame** enega ali več elementov skupaj z ustreznimi poddrevesi in z zamenjavo ustreznega elementa pri očetu.
   b) Če nobeden od bratov nima dovolj velikega števila elementov, imamo dva brata (en $\lceil m/2 \rceil$ *-1* in drugi $\lceil m/2 \rceil$ *-2* elementov), ki ju skupaj z ustreznim elementom v očetu lahko **združimo** v eno vozlišče, ki ima $\lceil m/2 \rceil - 2 + \lceil m/2 \rceil - 1 + 1 \leq m - 1$ elementov.
   Postopek brisanja zatem **rekurzivno ponovimo pri očetu**.
   Rekurzija se konča bodisi pri 1. ali 2.(a) točki ali pa pri korenu brez elementov (v tem primeru koren zbrišemo in se višina drevesa zmanjša za 1).

Časovna zahtevnost brisanja je vedno *O(log n)*

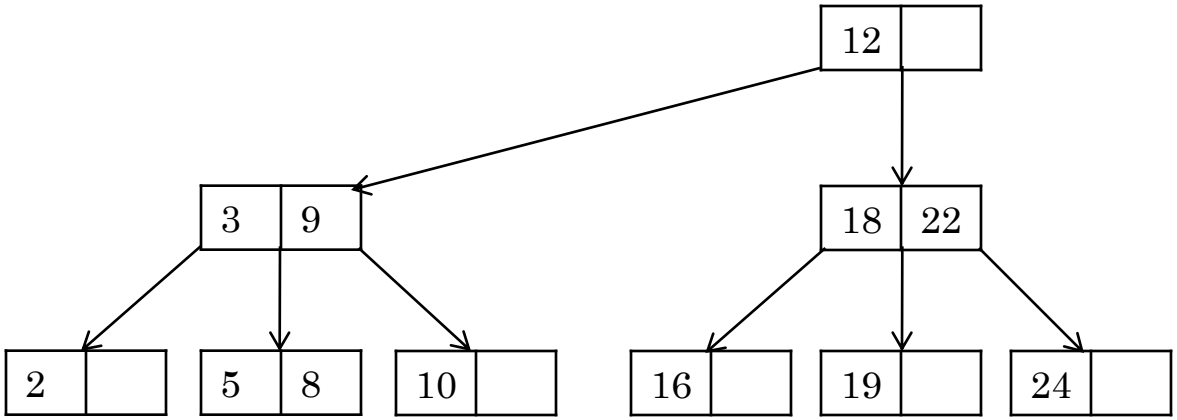
Primer (1/15)

Na sliki je podani B-drevo reda 3. Iz drevesa zbriši naslednje elemente: 5, 22, 24, 16, 18 in 2.



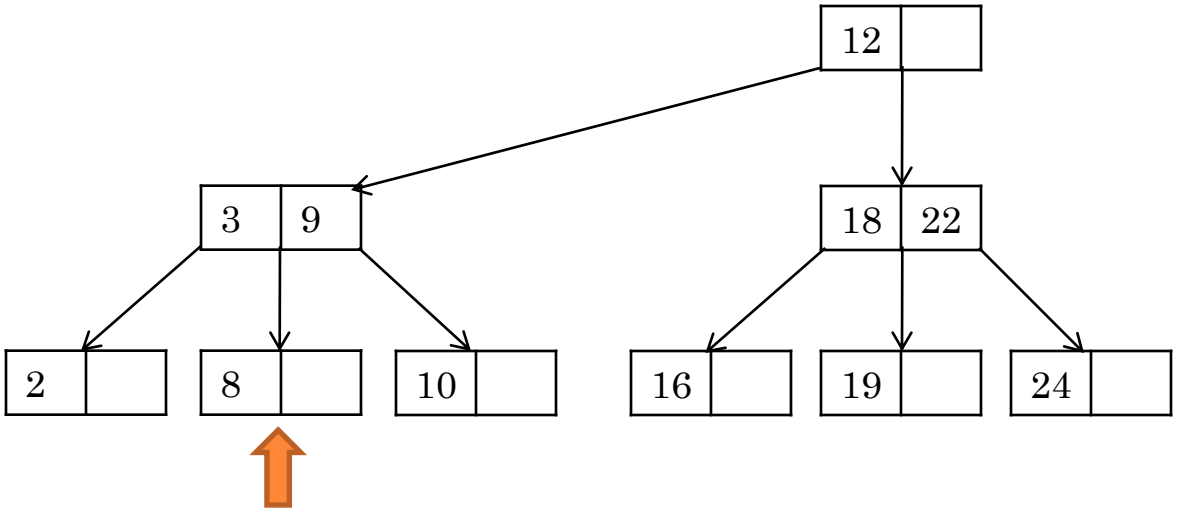
Primer (2/15)

Brišemo 5.



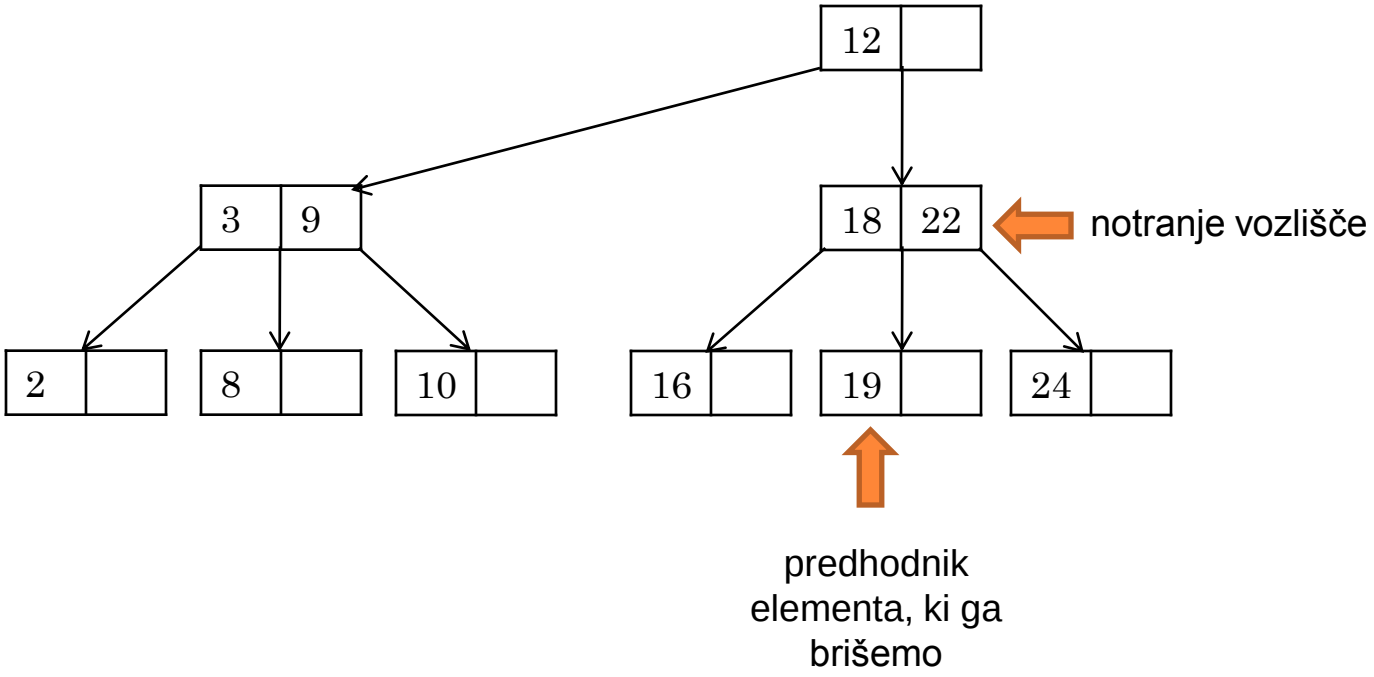
Primer (2/15)

Brišemo 5.



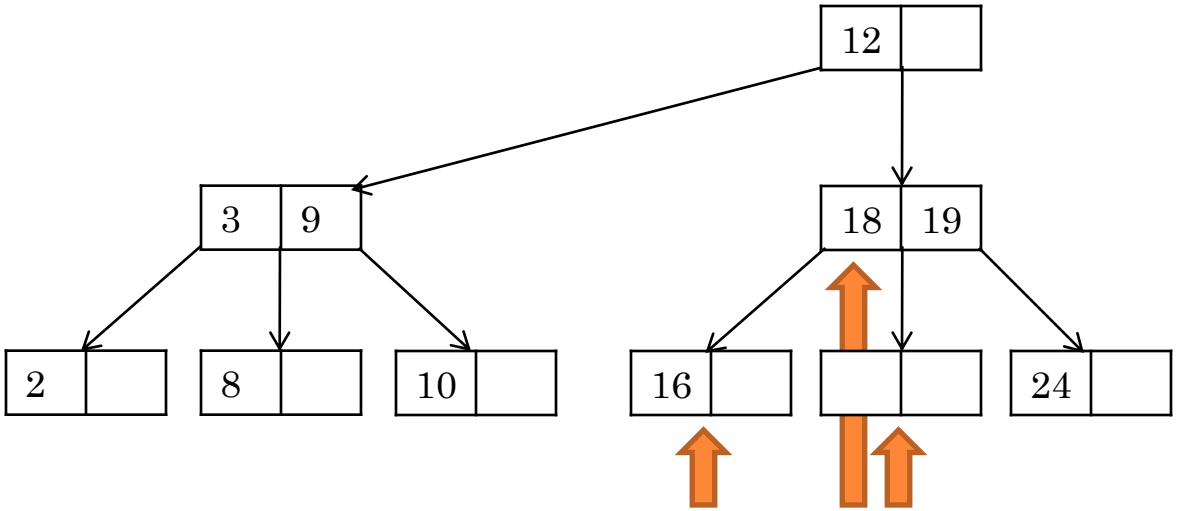
Primer (3/15)

Brišemo 22...



Primer (4/15)

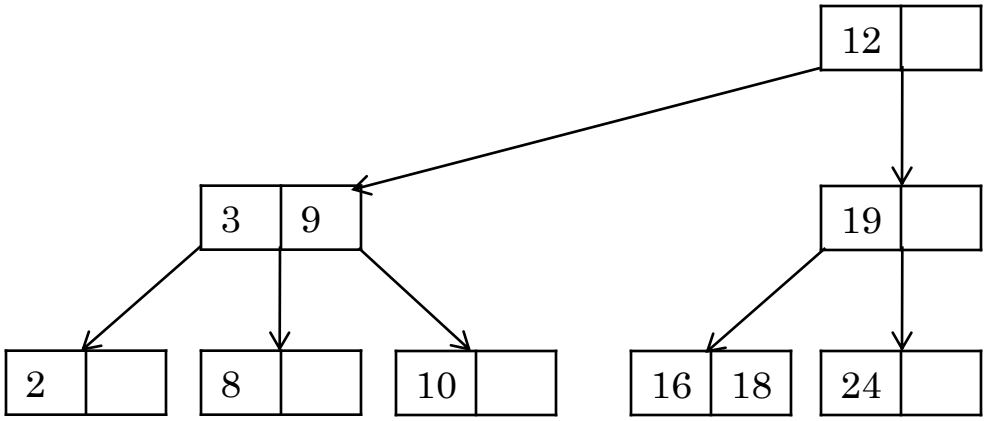
Brišemo 22...



prazno vozlišče združimo z levim bratom, skupaj z ustreznim elementom od očeta

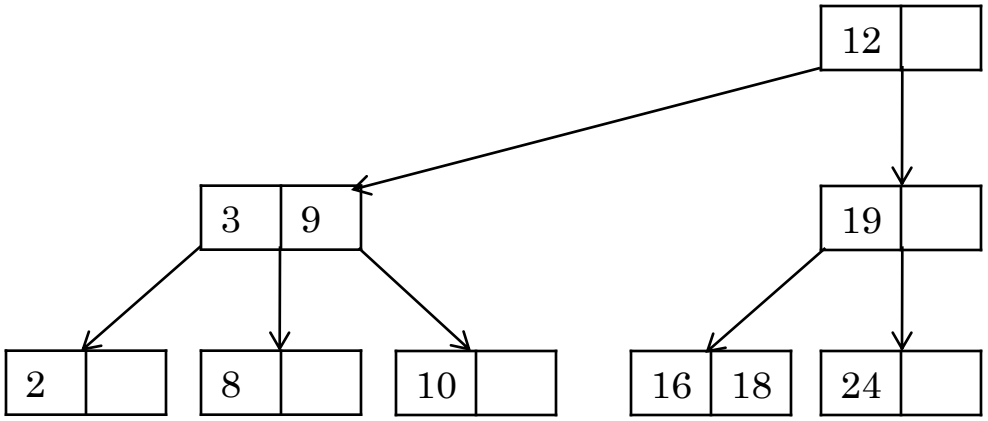
Primer (5/15)

Brišemo 22.



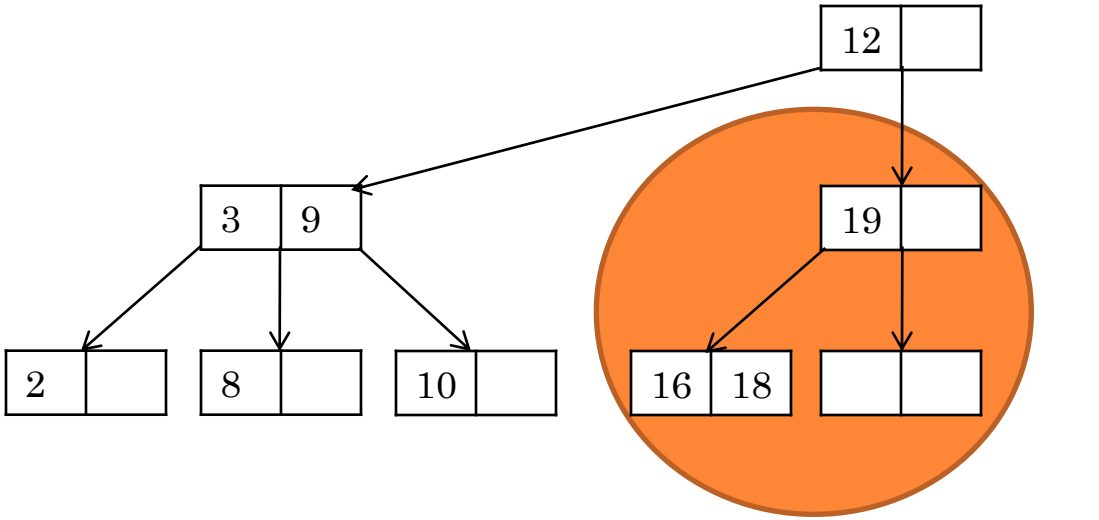
Primer (6/15)

Brišemo 24.



Primer (6/15)

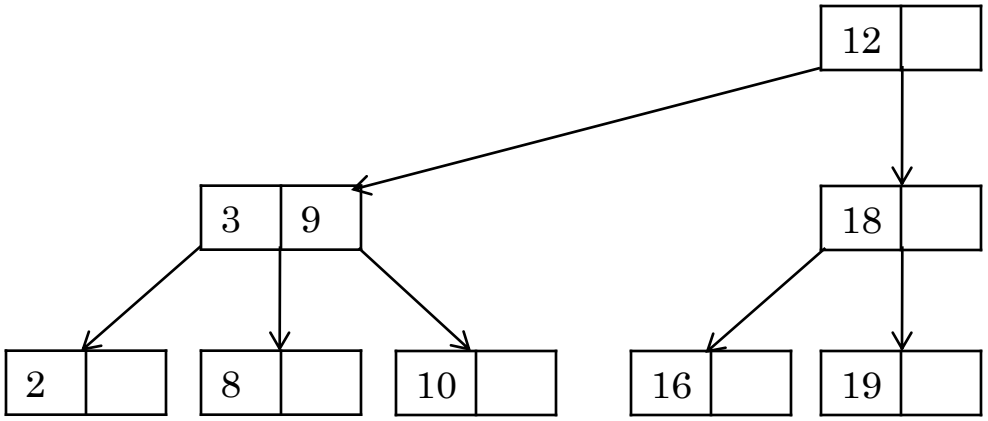
Brišemo 24…



porazdelimo elemente z levim bratom skupaj z zamenjavo ustreznega elementa pri očetu

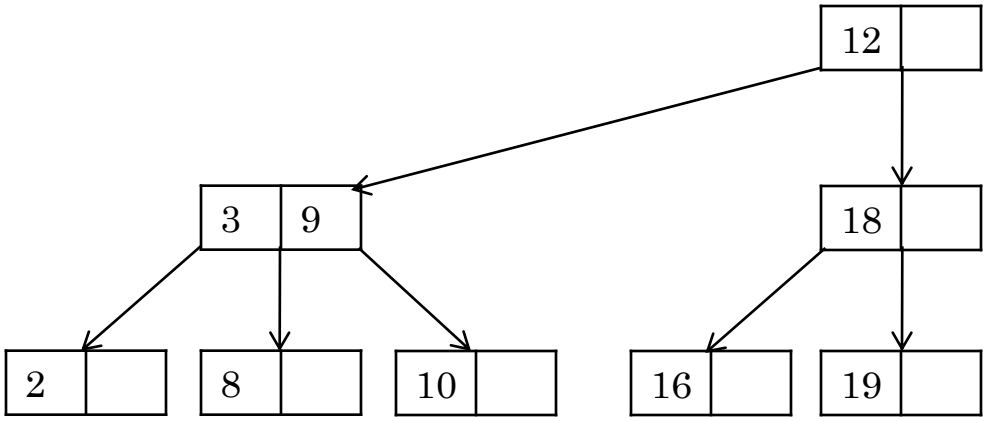
Primer (7/15)

Brišemo 24.



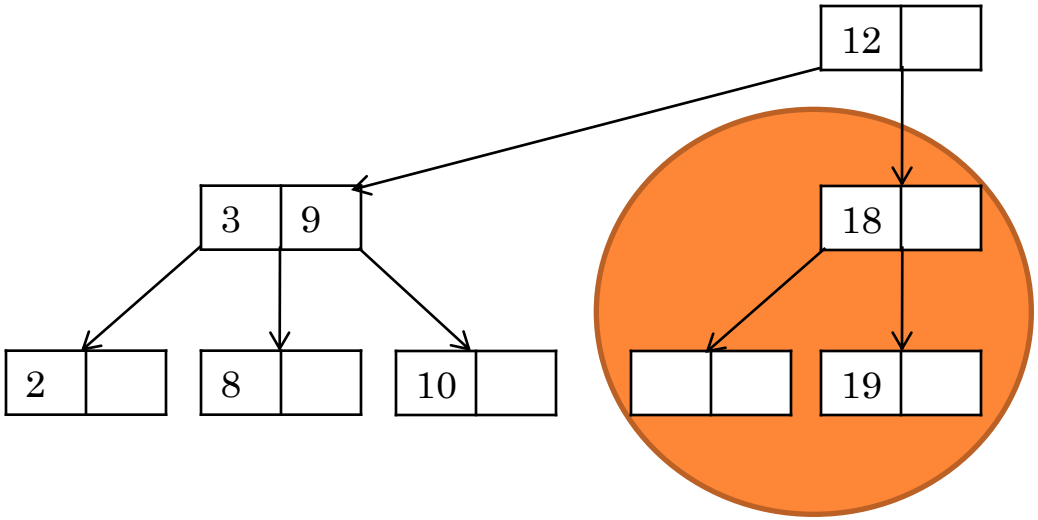
Primer (8/15)

Brišemo 16.



Primer (8/15)

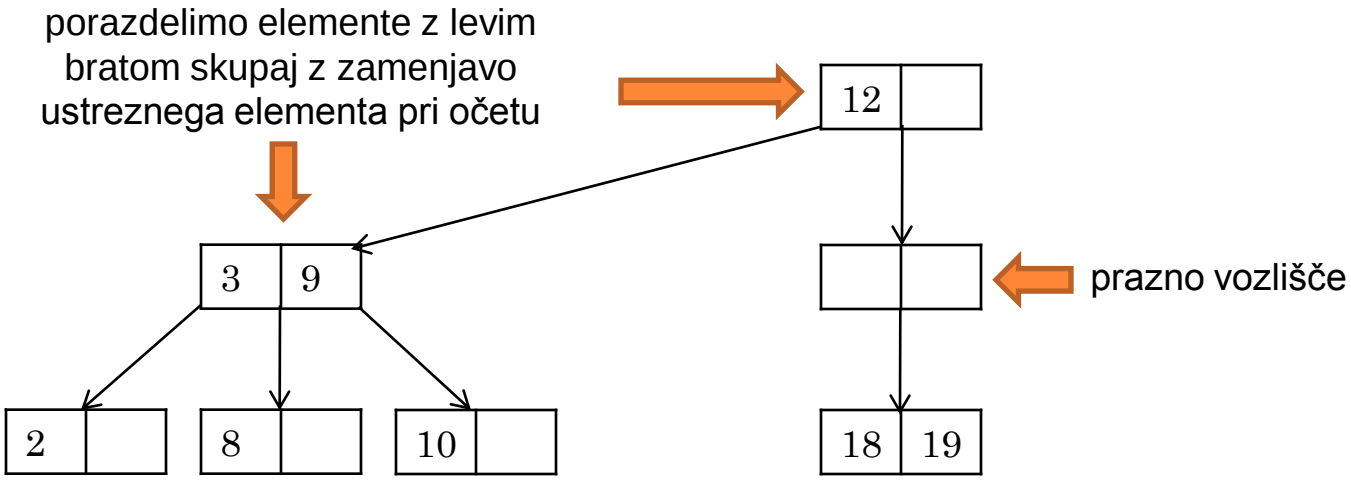
Brišemo 16...



prazno vozlišče združimo z desnim bratom, skupaj z ustreznim elementom od očeta

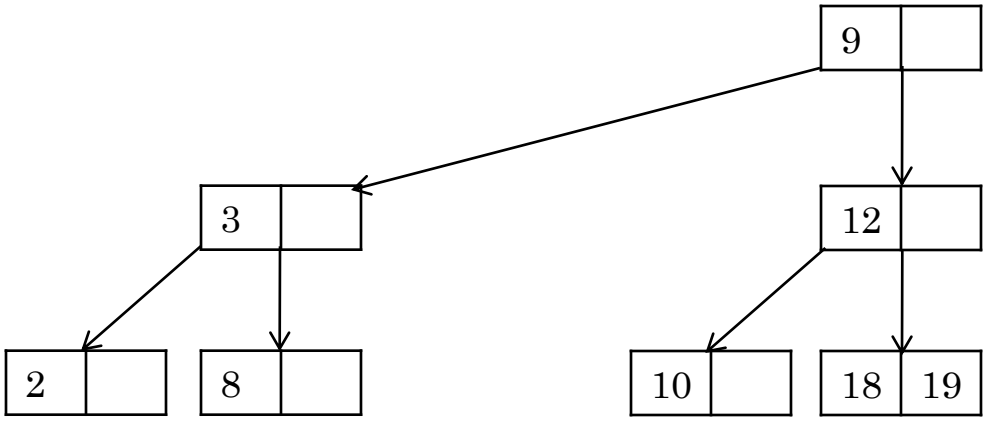
Primer (9/15)

Brišemo 16...



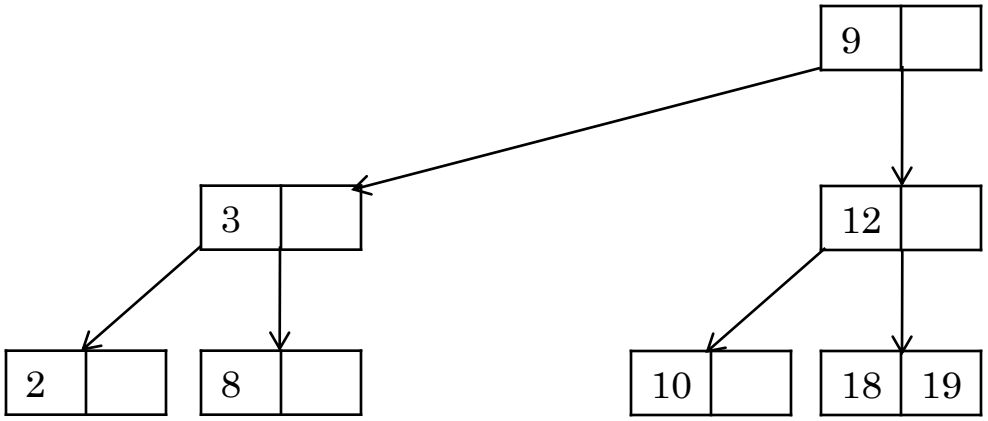
Primer (10/15)

Brišemo 16.



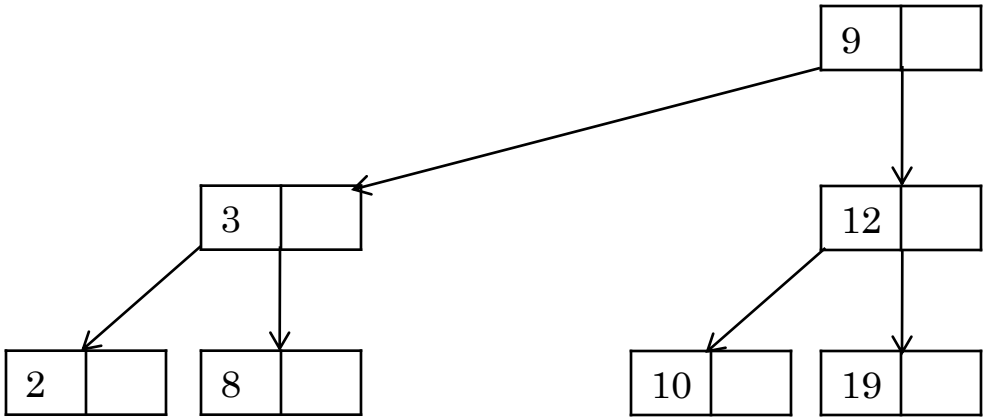
Primer (11/15)

Brišemo 18.



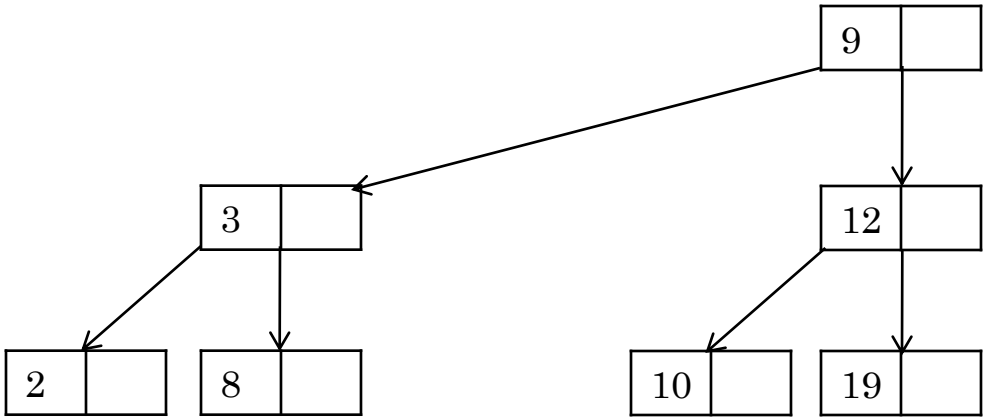
Primer (11/15)

Brišemo 18.



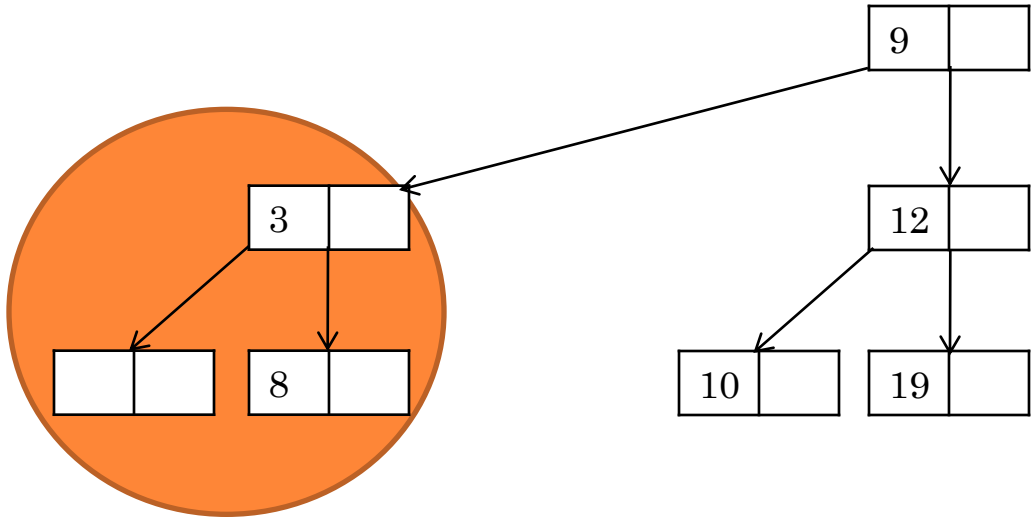
Primer (12/15)

Brišemo 2.



Primer (12/15)

Brišemo 2...

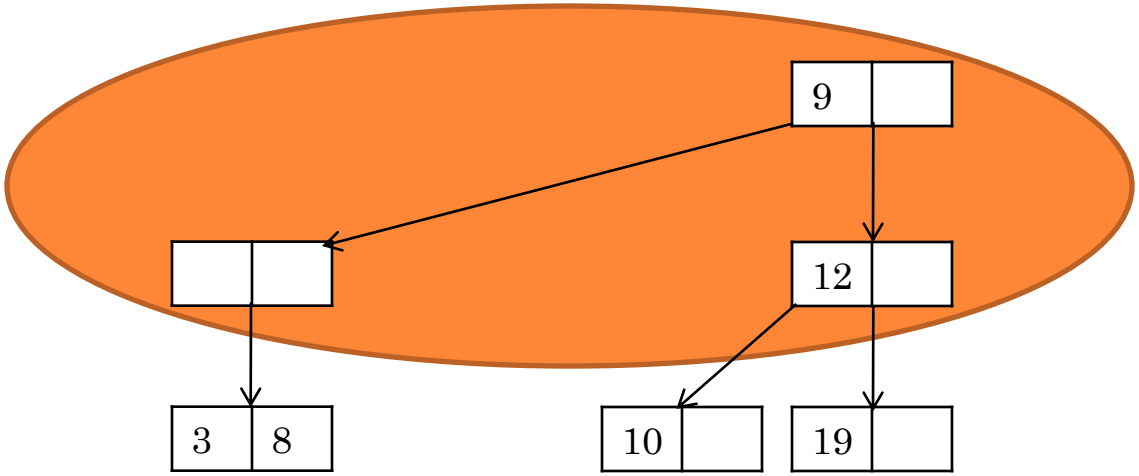


prazno vozlišče združimo z desnim bratom, skupaj z ustreznim elementom od očeta

Primer (13/15)

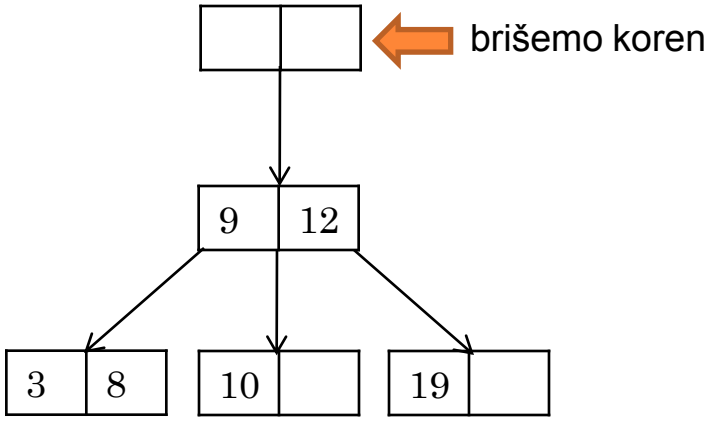
Brišemo 2...

prazno vozlišče združimo z desnim bratom, skupaj z ustreznim elementom od očeta



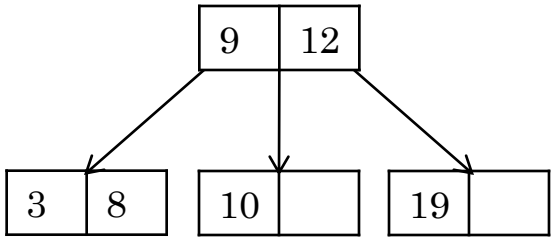
Primer (14/15)

Brišemo 2...



Primer (15/15)

Brišemo 2.



Povzetek

- B-drevesa so popolnoma poravnana iskalna drevesa.
- Vse operacije na B-drevesih so reda $O(\log n)$
- m je konstanten in zato preiskovanje elementov znotraj vozlišča izvedemo v konstantnem času, ki je lahko $O(m)$ ali pa z bijekcijo $O(\log m)$
- B-drevesa visokega reda se uporabljajo za shranjevanje velikih baz podatkov na zunanjem pomnilniku.