

## DVOJIŠKA ISKALNA DREVESA

---

**Dvojiško iskalno drevo** (DID) je dvojiško drevo z naslednjo lastnostjo: za vsako vozlišče  $v$  velja, da so v levem poddrevesu vozlišča s ključi, ki so manjši, v desnem poddrevesu pa ključi, ki so večji od ključa v vozlišču  $v$ . Ta lastnost omogoča, da izpis ključev v vozliščih v načinu starš vmes (LSD) izpiše elemente urejene od najmanjšega do največjega.

Operacije definirane na dvojiških iskalnih drevesih:

- najdi(koren, ključ) ... v dvojiškem iskalnem drevesu s podanim korenem iščemo vozlišče z danim ključem
- obstaja(koren, ključ) ... vrne *true*, če v dvojiškem iskalnem drevesu s podanim korenem obstaja vozlišče z danim ključem
- vstavi(koren, vozlišče) ... v dvojiško iskalno drevo s podanim korenem vstavi novo vozlišče tako, da se ohrani lastnost dvojiškega iskalnega drevesa
- odstrani(koren, ključ) ... iz dvojiškega iskalnega drevesa s podanim korenem odstrani element z danim ključem, če obstaja
- minimum(vozlišče) ... vrne kazalec na vozlišče, ki vsebuje najmanjšo vrednost v drevesu s korenem *vozlišče*
- maximum(vozlišče) ... vrne kazalec na vozlišče, ki vsebuje največjo vrednost v drevesu s korenem *vozlišče*
- naslednik(vozlišče) ... vrne kazalec na vozlišče, ki vsebuje vrednost, ki je naslednja po vrsti glede na vrednost v vozlišču *vozlišče*

### ALGORITEM NAJDI(KOREN, X)

---

Algoritem v DID s korenem *koren* poišče vozlišče s ključem  $x$ , če obstaja, sicer vrne NULL.

Poglejmo najprej rekurzivno rešitev:

```
Algoritem najdiRekurzivno(koren, x)  
if (koren==NULL || koren->ključ == x)  
    vrni koren;  
else  
    if (koren->ključ > x)  
        najdiRekurzivno(koren->levi, x);  
    else  
        najdiRekurzivno(koren->desni, x);
```

In še nerekurzivna rešitev:

```
Algoritem najdi(koren, x)  
pomozni = koren;  
while (pomozni != NULL && pomozni->kljuc != x)  
    if (x < pomozni->kljuc)  
        pomozni = pomozni->levi;  
    else  
        pomozni = pomozni->desni;  
vrni pomozni
```

Časovna zahtevnost iskanja elementa je odvisna od globine dvojiškega iskalnega drevesa. V najslabšem primeru je drevo izrojeno in je časovna zahtevnost algoritma *najdi* linearna glede na število elementov.

## ALGORITEM VSTAVI(KOREN, VOZLIŠČE)

---

Algoritem v DID s korenem *koren* vstavi novo vozlišče *vozlišče*. Tudi vstavljanje v DID je linearne časovne zahtevnosti glede na število elementov v drevesu.

### **Algoritem vstavi(koren, vozlišče)**

```
pomožni = koren;
prejšnji = koren;

// najprej se premaknimo do vozlišča, za katerim vstavljamo
while pomožni!=NULL {
    // zapomnimo si starša od vozlišča, v katerega se bomo premaknili
    prejšnji = pomožni;

    if vozlišče->ključ < pomožni->ključ
        pomožni = pomožni->levi;
    else
        pomožni = pomožni->desni;
}

if koren==NULL
    koren = vozlišče;
else
    if vozlišče->ključ < prejšnji->ključ
        prejšnji->levi = vozlišče;
    else
        prejšnji->desni = vozlišče;
```

---

## ALGORITEM ODSTRANI(KOREN, VOZLIŠČE)

---

Pri odstranjevanju vozlišča iz DID moramo biti pozorni, da se ne pokvari lastnost DID. Postopek je naslednji:

- če je vozlišče list, ni težav z odstranjevanjem
- če ima vozlišče natanko enega otroka, je potrebno ustrezno samo povezati drevo s tem otrokom, preden izbrišemo vozlišče
- če ima vozlišče oba otroka, potem njegov *naslednik* (poglejte definicijo naslednika zgoraj) nima levega otroka. Zamenjaj ključe v vozlišču vozlišče in njegovem nasledniku in nato odstrani naslednika.

### Algoritem odstrani(koren, vozlišče)

```
if (vozlišče->levi == NULL || vozlišče->desni == NULL)
    pomožni = vozlišče;
else
    pomožni = naslednik(vozlišče);

if (pomožni->levi != NULL)
    otrok = pomožni->levi;
else
    otrok = pomožni->desni;

if (otrok != NULL)
    otrok->starš = pomožni->starš;

if (pomožni->starš) == NULL
    koren = otrok;
else
    if (pomožni == pomožni->starš->levi)
        pomožni->starš->levi = x;
    else
        pomožni->starš->desni = x;

if (pomožni != vozlišče)
    vozlišče->ključ = pomožni->ključ; // če imamo še kakšne ključe,
jih skopiramo

izbriši pomožni
```

---

## OPOMBE K IMPLEMENTACIJI

---

Pri implementaciji se ob sklicih na vozlišča prej prepričajte, da vozlišča sploh obstajajo. Zgoraj podani algoritmi so zgolj informativne narave, običajno jih je potrebno prilagoditi sami nalogi in implementaciji.