

# Bináris fán alapuló keresés. Beszúrás, törlés és kiegyensúlyozás.

## Fák (kicsit különböző fogalmak)

- Nyílt fa (fa): összefüggő, körmentes, irányítatlan gráf (nincs kitüntetett csúcs, nincs gyökér)
- Erdő: nem összefüggő
- Tulajdonságai. Az alábbi állítások egyenértékűek:
  1.  $G=(E, V)$  egy nyílt fa
  2.  $G$  bármely két csúcsához egyértelműen tartozik egy őket összekötő egyszerű út
- $G$  összefüggő, de bármely élének elhagyása után már nem az
- $G$  összefüggő, és  $|E|=|V|-1$
- $G$  körmentes, és  $|E|=|V|-1$
- $G$  körmentes, de akár egyetlen éllel is bővítve, már nem az
- Bizonyítások:  $1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 6, 6 \rightarrow 1$

## Fák

- Gyökeres fák: egy pont kitüntetett
- $x$ -ben gyökerező fa
- Legyen  $x$  gyökér. Az  $x_w$  út elemeit  $w$  megelőzőjének nevezzük. Ha  $x$  megelőzője  $y$ -nak, akkor  $y$  rákövetkezője  $x$ -nek.
- Szülő, gyerek, testvér, foksám
- Fa szintjei, magassága
- Rendezett fa: ha minden csúcs gyerekei rendezettek (létezik, értelmezünk, használunk rendezési relációt, akár az elemek feletti rendezés, akár sorba állítás)
- $K$ -ágú teljes fák: melyekben nincsenek részben hiányzó gyerekek/a csúcsok fokszáma vagy  $0$  (levél) vagy  $K$ , ÉS a minden levél magassága egyforma  $H$ .
- Tétel:  $K$ -ágú teljes fák leveleinek a száma:  $K^H$ .
- Tétel:  $K$ -ágú teljes fák csúcsainak a száma:  $1+K+K^2+\dots+K^N = (K^{N+1}-1)/(K-1)$

## Bináris keresőfák

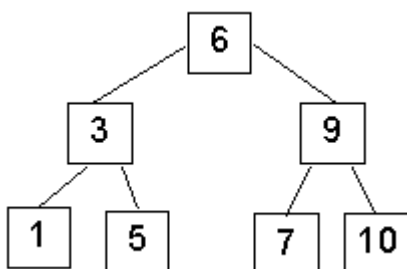
**Binárisnak** nevezünk egy fát, ha minden elemének legfeljebb két rákövetkezője van.

**Szigorú** értelemben vett **bináris** fáról akkor beszélünk, amikor minden elemnek 0 vagy 2 rákövetkező eleme van.

Bináris fáknál beszélhetünk **baloldali** és **jobboldali részfákról**.

- Adatszerkezet, amely a keresést felgyorsítja.
- Átlagosan:  $\Theta(\log(n))$ , legrosszabb esetben:  $\Theta(n)$
- Használat: elsőbbségi sorként, ill. szótárként
- Bináris keresőfa tulajdonság: minden  $v$  csúcsra:  $v.bal < v < v.jobb$ , ha létezik balfa, ill. jobbfa
- Ugyanahhoz a halmazhoz több ilyen fa is építhető

## **Bináris kereső fák**



Az adatalemek között definiálható egy rendezési sorrend

minden csúcsra igaz:

- ✚ Balra a nála kisebb elemek helyezkednek el
- ✚ Jobbra a nagyobb elemek helyezkednek el

A feltételt sokféleképpen kielégíthetjük, de a gyorsabb kezelés érdekében kiegyensúlyozott fákat használunk.

Műveletek bináris rendezési fákön:

## Beszúrás

### ✚ Levélelem beszúrása

- ✚ A levél kulcsa alapján megkeressük azt az egy leágazású csúcsot vagy levélelemet, amelyen levélelemként elhelyezkedne

- ✚ Allokálunk neki helyet, létrehozuk az elemet

- ✚ A megtalált csúcs megfelelő mutatóját beállítjuk

### ✚ Gyökérelem beszúrása: O

- ✚ Megnézem, hogy a beszúrandó elem kisebb vagy nagyobb a gyökéknél

- ✚ Ha a beszúrandó elem kisebb a gyökéknél

- ✚ Gyökér + jobboldali részfa jobbra

- ✚ Baloldali részfat két részre bontom  $B1 < O$ ;  $B2 > O$

- ✚ B1 beszúrása az új elemtől balra

- ✚ B2 beszúrása az eredeti gyökértől jobbra

- ✚ ha a beszúrandó elem nagyobb a gyökéknél ...

### ✚ Törlés

- ✚ **Levélelem törlése:** Az ősnének mutatóját NIL-re állítom

- ✚ **Egy leágazású csúcs törlése:** elődjének mutatóját rá állítjuk a leágazás csúcsára

### ✚ Két leágazású csúcs törlése

- ✚ Vagy a baloldali részfa legnagyobb elemét veszem, ami vagy levél, vagy 1 leágazású csúcs. A két elemet értékét felcserélem, és az eredeti értéket tartalmazó elemet a már definiált módon törlöm.

- ✚ Vagy a jobboldali részfa legkisebb elemét veszem, felcserélem az értékeket, és az eredeti értéket tartalmazó elemet törlöm.

- ✚ **Bejárás:** Inorder módon bejárható

- ✚ **Elem keresése:** A gyökérből kiindulva  $< >$  vizsgálatok szerint bal illetve jobb részfán haladva. Rekurzív módon viszonylag egyszerűen megoldható.

## Műveletek és megvalósítás

- Futásidő:  $O(h)$
- keres( $k$ :Kulcs):KulcsosRekord

```
    if k=tartalom.kulcs
    then return tartalom
    if k<tartalom.kulcs
    then return balfa.keres(k)
    else return jobbfa.keres(k)
```
- keres( $f$ :BinárisFa, $k$ :Kulcs):KulcsosRekord

```
    while f!=NULL és k!=f.tartalom.kulcs
    do if k<f.tartalom.kulcs
    then f=f.balfa
    else f=f.jobbfa
    return f
```

## **Fák bejárása**

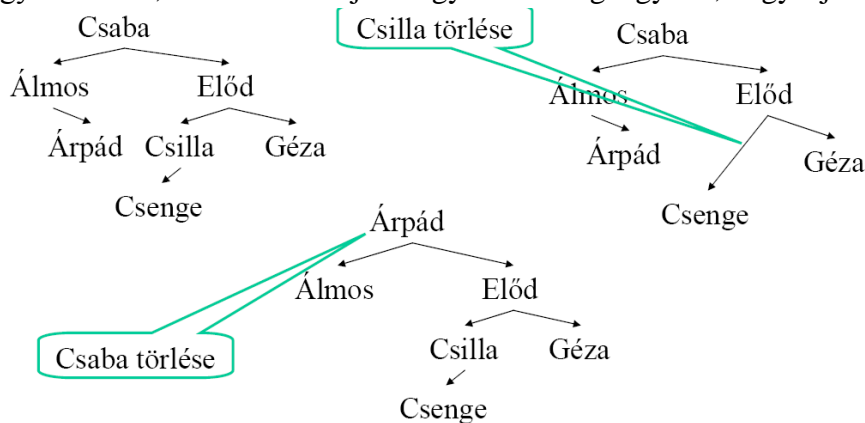
- InorderBejárás()  
if balfa!=NULL then balfa.InorderBejárás()  
print tartalom.kulcs  
if jobbfa!=NULL then jobbfa.InorderBejárás()
- PreorderBejárás()  
print tartalom.kulcs  
if balfa!=NULL then balfa.PreorderBejárás()  
if jobbfa!=NULL then jobbfa.PreorderBejárás()
- PosztorderBejárás()  
if balfa!=NULL then balfa.PosztorderBejárás()  
if jobbfa!=NULL then jobbfa.PosztorderBejárás()  
print tartalom.kulcs
- minimum():BinárisFa  
if balfa!=NULL then return balfa.minimum()  
else return Me
- maximum():BinárisFa  
if jobbfa!=NULL then return jobbfa.maximum()  
else return Me
- következő():BinárisFa  
if jobbfa!=NULL then return jobbfa.minimum()
- következő(k:Kulcs):BinárisFa  
ez = keres(k)  
if ez=NULL return NULL  
ez=ez.következő()  
if ez!=NULL return ez  
az = ez.ise  
while az!=NULL és ez=az.jobb  
ez=az; az=az.ise  
return az

## **Beszúrás**

- Addig keresünk a fában, amíg NULL-hez jutunk.
- Beszúr(r:KulcsosRekord)  
szülő=NULL, gyerek=Me  
while gyerek!=NULL  
do szülő=gyerek  
if r.kulcs=szülő.tartalom.kulcs  
then hiba „Már benne van!”,  
return  
if r.kulcs<szülő.tartalom.kulcs  
then gyerek=szülő.balfa  
else gyerek=szülő.jobbfa  
if r.kulcs<szülő.tartalom.kulcs  
then szülő.balfa=new BinárisFa(r)  
else szülő.jobbfa=new BinárisFa(r)

## Törlés

- ha nincs gyereke, akkor töröljük a rámutatót
- ha egy gyereke van, akkor a rámutató a gyerekre mutat
- ha két gyereke van, akkor kicseréljük vagy a balfa legnagyobb, vagy a jobbfá legkisebb elemével



```
törlés(k:Kulcs)
csúcs=keres(k)
if csúcs.bal=NULL vagy csúcs.jobb=NULL then
vág=csúcs
else vág=elızı(csúcs)
if vág.bal=NULL then
fia=vág.jobb
else fia=vág.bal
if fia!=NULL
then fia.őse=vág.őse
if fia=vág.bal
then vág.őse.bal=fia
else vág.őse.jobb=fia
if csúcs!=vág then
csúcs.tartalom=vág.tartalom
```

## Keresőfák egyenlő kulcsokkal

A Beszúr eljárás módosítása: a fa tulajdonságot  $\leq$  kell módosítani...

1. stratégia: minden elemhez vegyünk fel egy láncolt listát az egyenlő kulcsú elemek tárolására.
2. stratégia: az új, megegyező kulcsú elemet hol a bal, hol a jobb részében tároljuk váltakozva
3. stratégia: a bal és jobb részét véletlenszerűen változtatjuk

(a balfa legjobboldalsó/legnagyobb elemeit, ill. a jobbfá legbaloldalsó/legkisebb elemeit építjük láncszerűen tovább)

## Radix fák (kódfák)

**Def:** a jelsorozat lexikografikusan kisebb b-nél, ha 1. a első b-étől különböző eleme kisebb. 2. Ha a előtagja b-nek. (abc szerinti rendezés)  
A jelsorozatok tárolása nem szükséges

## Véletlen építésű keresőfák

Adott kulcskészlethez többféle keresőfa is felépíthető (a beszúrás sorrendjétől függően).

**Def:** Építsünk keresőfát  $n$  db. elemből. Ha mindegyik sorrend (permutáció) egyformán valószínű, akkor véletlen építésű keresőfáról beszélünk.

**Def:** Kiegyenlített a fa, ha a levelek magassága közel megegyező

Tétel: Egy  $n$  különböző kulcsot tartalmazó véletlen építésű keresőfa átlagos magassága  $O(\lg(n))$

Biz nélkül...

→ vagyis a kiegyenlített fák létrejötte sokkal valószínűbb, mint különböző degenerációiké (pl. láncoké)

### **Piros-fekete fák**

Beszúrás, ill. törléskor a fa elveszítheti az egyensúlyát.

A piros-fekete fák: az egyensúly megtartását biztosítják. +1 bit információ (szín) → a legrövidebb és leghosszabb út legfeljebb 2-szeres arányú lehet

Fatulajdonságok:

- minden csúcs vagy fekete, vagy piros
- minden levél fekete
- minden piros csúcs mindkét utódja fekete
- bármely két azonos csúcsból a levélig jutó útban ugyanannyi fekete csúcs van

### **AVL(Adelson-Velskij-Landis) kiegyensúlyozó algoritmus**

AVL-fa alatt egy ön-kiegyensúlyozó bináris keresőfát értünk. Ez volt az első ilyen adatszerkezet. Egy AVL-fában bármely csúcspont két részfájának magassága közti különbség legfeljebb egy. Kiegyensúlyozott-magasságúnak is hívják az ilyen struktúrákat. A keresés, beillesztés és törlés egyaránt  $O(\log n)$  nagyságrendű időt vesz igénybe, még a legrosszabb esetben is. Beillesztés és törlés esetén szükséges lehet forgatások alkalmazásával újra kiegyensúlyozni a fát.

**Tétel:** Ha  $S$  egy  $n$  csúcsú AVL fa, akkor a Beszúr művelet után legfeljebb egy forgatással helyreállítható az AVL tulajdonság → a beszúrás költsége ezután is  $O(\log n)$  marad.

Miért? Minden csúcsban tároljuk az itt gyökerező részfa magasságát. Beszúrás után az új elemhez vivő út bejárásával megkapjuk az első olyan elemet, ahol az AVL tulajdonság először megsérül → itt végzünk forgatást

**Tétel:** Ha  $S$  egy  $n$  csúcsú AVL fa, akkor a Töröl művelet után legfeljebb  $1.44 \cdot \log(n)$  forgatással helyreállítható az AVL tulajdonság

Biz nélkül...