Adatszerkezetek

14. hét – Fák: fogalmak, bináris fák, ábrázolás, bejárások

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Ismétlés

- Eddig **lineáris** adatszerkezetek: tömb, lista, verem, sor – minden elemnek legfeljebb **egy** rákövetkezője van
- **Rekurzió** (12. hét): a feladat visszavezetése kisebb példányokra
- **Verem** és **sor**: a bejárások segédeszközei

A mai óra kérdései

- Hogyan tárolunk **hierarchikus** adatokat?
- Mi a bináris fa, és hogyan ábrázolható a memóriában?
- Hogyan járjuk be egy fa összes csomópontját?

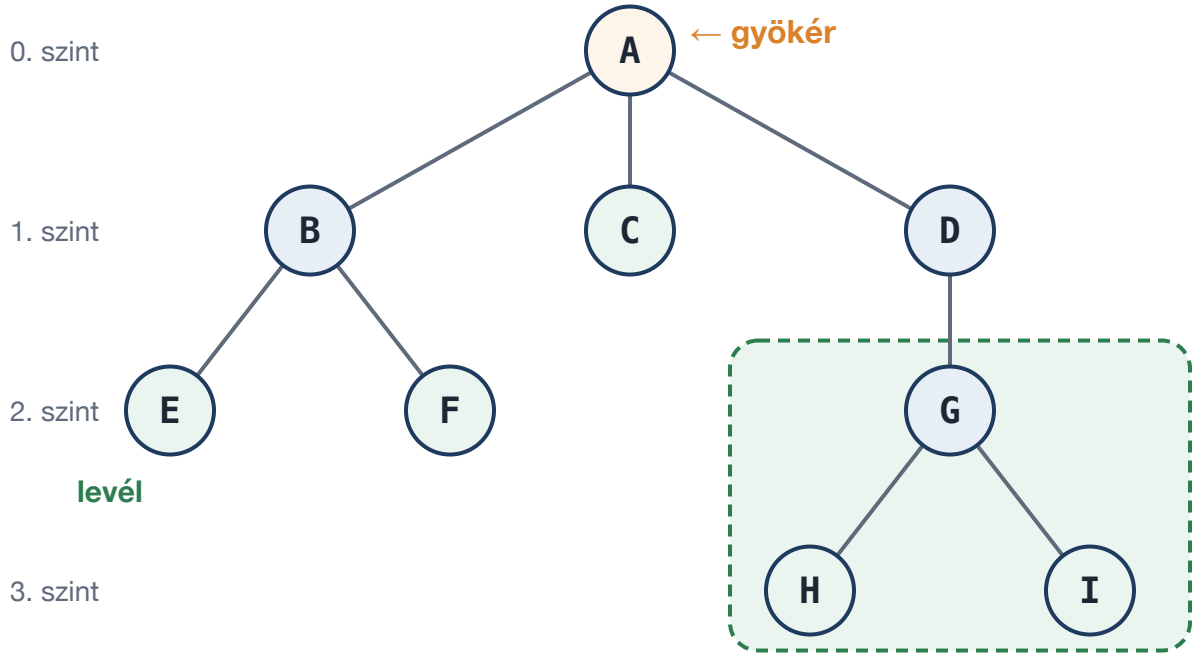
Hierarchikus adatok

Sok adat természetes módon **fa szerkezetű**:

- **Fájlrendszer:** mappák, almappák, fájlok
- **Szervezeti ábra:** egyetem → kar → intézet → tanszék
- **Családfa**
- **Weboldal** (HTML DOM): `<html>` → `<body>` → `<div>` → …
- **Kifejezések:** `(3 + 4) * (5 - 2)`
- **Döntések:** sakkprogram lépésfája, döntési fa a gépi tanulásban

> A fa **hierarchikus** adatszerkezet: minden elemnek **legfeljebb egy előzője** (szülője), de **több rákövetkezője** (gyereke) lehet (1. hét: osztályozás).

Fogalmak



B szülője: A · A gyerekei: B, C, D

B, C, D testvérek

H ősei: G, D, A

A fokszáma: 3 · G fokszáma: 2

H mélysége: 3

a fa magassága: 3

levelek: C, E, F, H, I

Fogalmak – összefoglalva

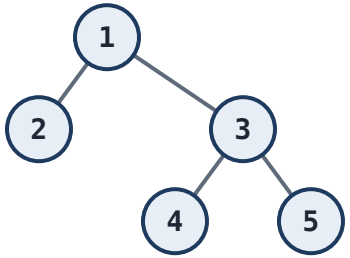
Fogalom	Jelentés
csomópont (node), **él** (edge)	az elemek, és a szülő–gyerek kapcsolatok
gyökér (root)	az egyetlen csomópont, amelynek nincs szülője
levél (leaf)	gyerek nélküli csomópont
szülő, gyerek, testvér	közvetlen kapcsolat · azonos szülő gyerekei
ős, leszármazott	a gyökér felé, illetve lefelé vezető úton lévők
részfa	egy csomópont és az összes leszármazottja
mélység / szint	a gyökértől vezető út hossza (a gyökéré 0)
magasság	a leghosszabb gyökér–levél út hossza
fokszám	a gyerekek száma · **erdő:** fák halmaza

Rekurzív definíció: a fa vagy **üres**, vagy egy **gyökérből** és annak **részfáiból** áll.

Bináris fák

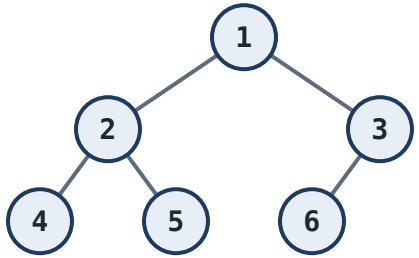
Bináris fa

Bináris fa: minden csomópontnak **legfeljebb két** gyereke van: **bal** és **jobb** gyerek.



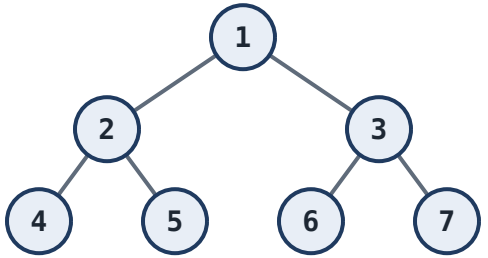
szigorú (teli)

0 vagy 2 gyerek



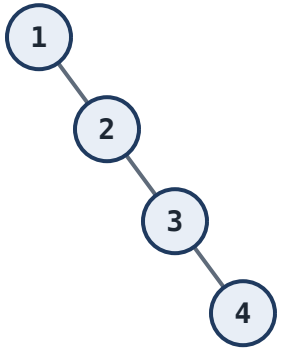
teljes

szintenként balról tele



tökéletes

minden szint tele



degenerált

valójában lista

A bináris fa tulajdonságai

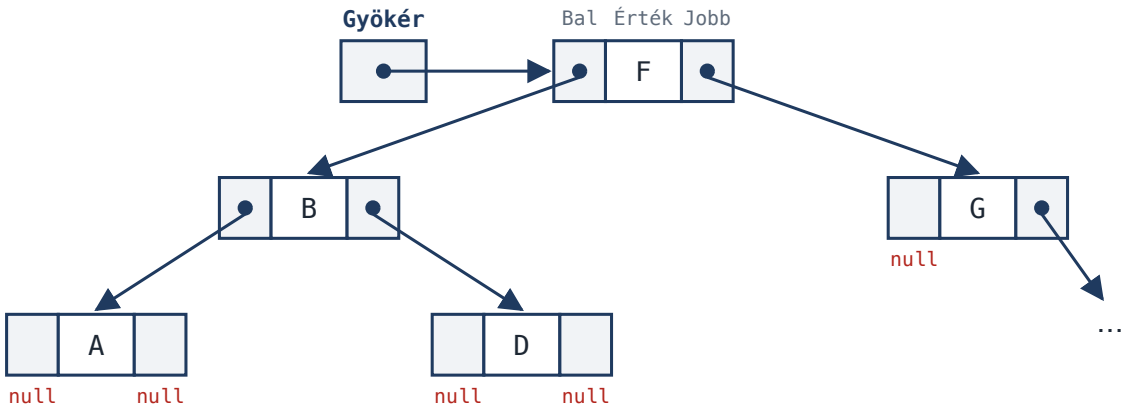
- Az **i**-edik szinten legfeljebb 2^i csomópont van
- Egy **h** magasságú bináris fában legfeljebb $2^{h+1} - 1$ csomópont van (tökéletes fa)
- Egy **n** csomópontú bináris fa magassága legalább $\lfloor \log_2 n \rfloor$, legfeljebb $n - 1$ (degenerált)

Tulajdonság	Bináris fa
Elemek	homogén
Kapcsolat	**hierarchikus**
Méret	dinamikus
Tárolás	szétszórt (láncolt) vagy folytonos (tömb)
Elérés	**szekvenciális** (a gyökértől indulva, mutatókon át)

A magasság a fa műveleteinek **lépésszámát** határozza meg – ezért fontos, hogy a fa **kiegyensúlyozott** legyen (15. hét).

A bináris fa ábrázolása

Láncolt ábrázolás



Java

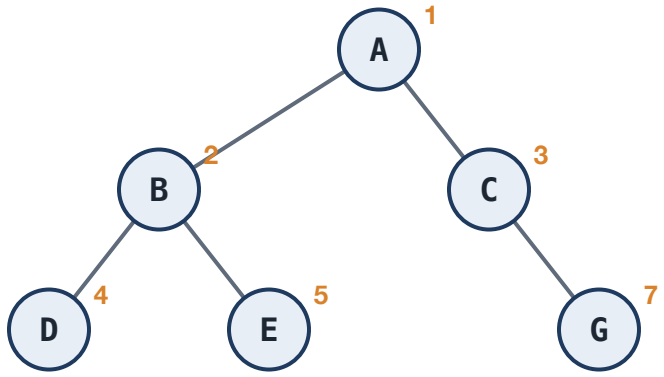
```
class Node {
    char value;
    Node left, right;
}
Node root;   // null: üres fa
```

Go

```
type Node struct {
    Value       rune
    Left, Right *Node
}
```

Pszeudokód: `Elem[P].Érték`, `Elem[P].Bal`, `Elem[P].Jobb`, a gyökérre a **Gyökér** mutató mutat.

Tömbös (folytonos) ábrázolás



A[1..7]

A	B	C	D	E		G
1	2	3	4	5	6	7

```
i bal gyereke: 2·i
i jobb gyereke: 2·i + 1
i szülője: i div 2
```

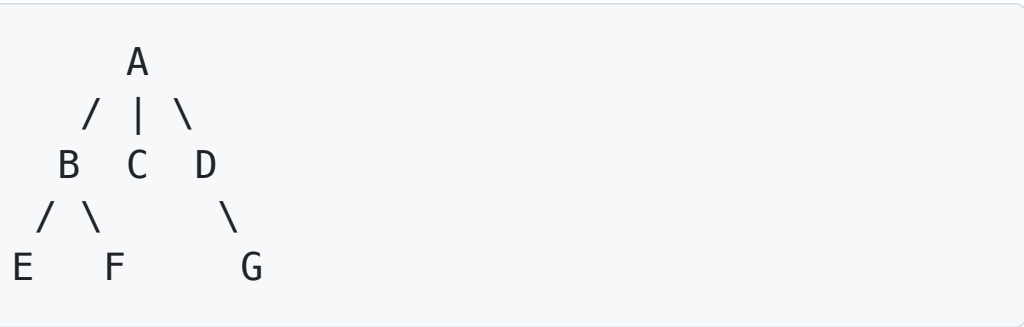
üres hely, ha a fa nem teljes (6. index)

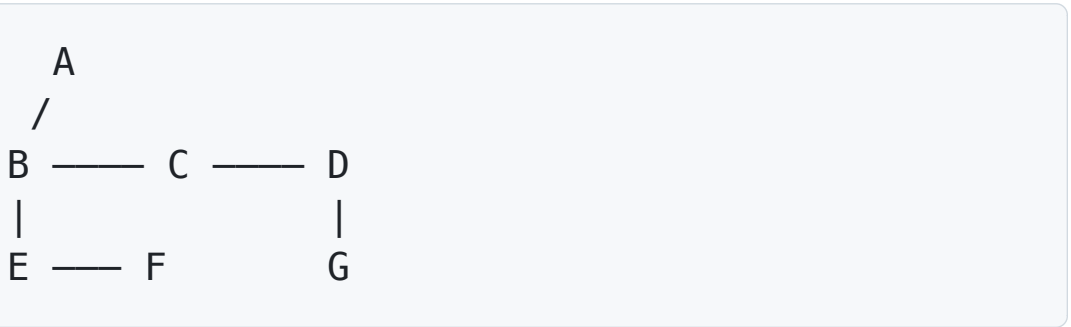
- Mutatók nélkül: a gyerekek és a szülő **indexe kiszámítható**
- **Teljes** fánál tökéletes (nincs üres hely) – ezt használja a **kupac** (15. hét)
- Degenerált fánál rengeteg üres hely: **h** magasságú lánchoz $2^{h+1} - 1$ elemű tömb kell!

Általános fa ábrázolása bináris fával

Ha egy csomópontnak **tetszőleges** számú gyereke lehet: **bal gyerek – jobb testvér** ábrázolás

- **Bal** mutató → az **első gyerek**
- **Jobb** mutató → a **következő testvér**



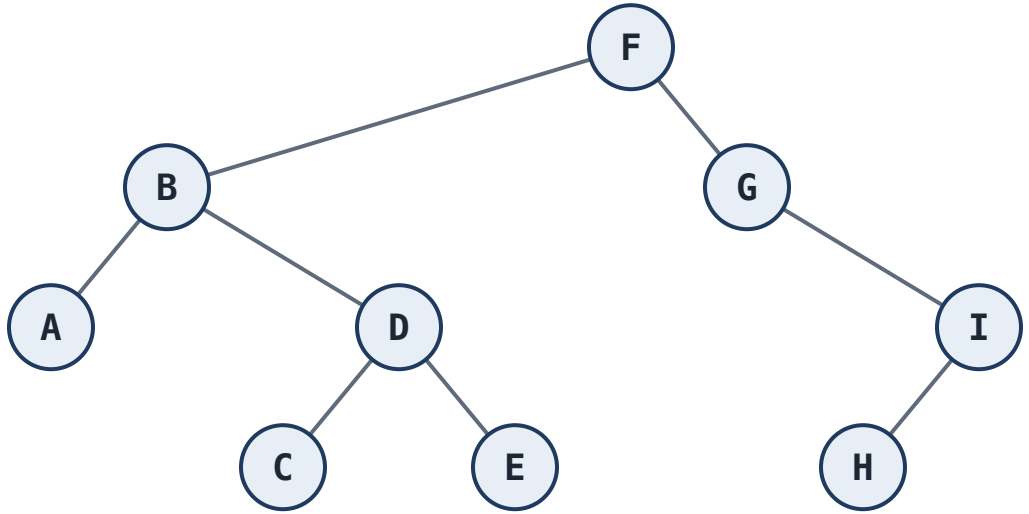


Így minden általános fa (és erdő) **bináris fával** tárolható – például egy fájlrendszer mappaszerkezete.

Bejárások

A fa bejárása

Bejárás: minden csomópont **pontosan egyszeri** feldolgozása, meghatározott sorrendben.



Preorder (gyökér, bal, jobb):
`F B A D C E G I H`

Inorder (bal, gyökér, jobb):
`A B C D E F G H I`

Postorder (bal, jobb, gyökér):
`A C E D B H I G F`

Szintenként (szélességi):
`F B G A D I C E H`

Mélységi bejárások – rekurzívan

Pszeudokód

```
Eljárás Preorder(P)
  Ha P ≠ Nil akkor
    Feldolgoz(Elem[P].Érték)
    Preorder(Elem[P].Bal)
    Preorder(Elem[P].Jobb)
  Elágazás vége
Eljárás vége

Eljárás Inorder(P)
  Ha P ≠ Nil akkor
    Inorder(Elem[P].Bal)
    Feldolgoz(Elem[P].Érték)
    Inorder(Elem[P].Jobb)
  Elágazás vége
Eljárás vége

Eljárás Postorder(P)
  Ha P ≠ Nil akkor
    Postorder(Elem[P].Bal)
    Postorder(Elem[P].Jobb)
    Feldolgoz(Elem[P].Érték)
  Elágazás vége
Eljárás vége
```

Java

```java
static void inorder(Node p) {
    if (p == null) return;
    inorder(p.left);
    System.out.print(p.value + " ");
    inorder(p.right);
}
```

Go

```go
func inorder(p *Node) {
    if p == nil {
        return
    }
    inorder(p.Left)
    fmt.Printf("%c ", p.Value)
    inorder(p.Right)
}
```

A három bejárás **csak abban különbözik**, hogy a gyökeret **mikor** dolgozzuk fel. Lépésszám: **O(n)**, tárigény: O(h).

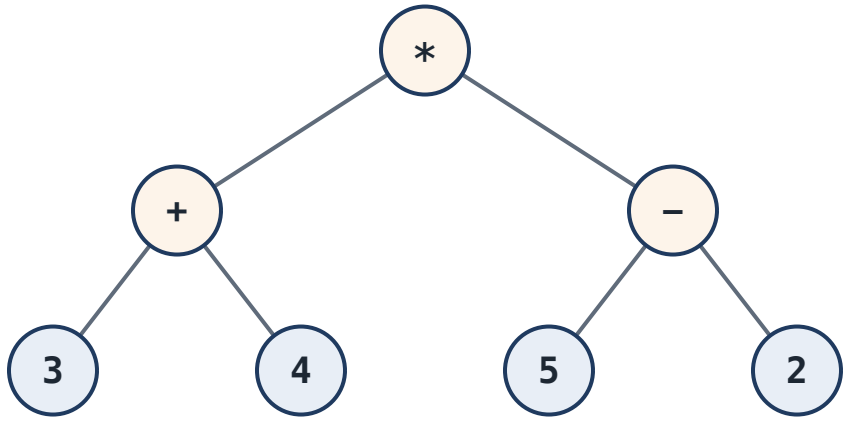
Szintenkénti (szélességi) bejárás

A gyökértől indulva **szintenként**, balról jobbra – **sorral** (8. hét):

```
Eljárás SzintenkéntBejár(Gyökér)
  Ha Gyökér = Nil akkor kilépés
  Sorba(Gyökér)
  Ciklus amíg nem Üres
    P ← Sorból
    Feldolgoz(Elem[P].Érték)
    Ha Elem[P].Bal ≠ Nil akkor Sorba(Elem[P].Bal)
    Ha Elem[P].Jobb ≠ Nil akkor Sorba(Elem[P].Jobb)
  Ciklus vége
Eljárás vége
```

Példa: **F B G A D I C E H** · Ha a sort **veremre** cseréljük (és előbb a jobb gyereket tesszük be), **preorder** bejárást kapunk!

Alkalmazás: kifejezésfa



inorder (zárójelekkel):
`(3 + 4) * (5 − 2)`

preorder (prefix):
`* + 3 4 − 5 2`

postorder (postfix, 7. hét):
`3 4 + 5 2 − *`

érték: 7 · 3 = 21

- A belső csomópontok **operátorok**, a levelek **operandusok**
- A **postorder** bejárás éppen a **postfix** alakot adja – a fa veremmel építhető fel a postfix alakból
- **Kiértékelés:** `Érték(P) = Érték(bal) op Érték(jobb)` – rekurzívan

Rekurzív műveletek fákon

Minta: **„kezeld a gyökeret, és hívd meg önmagad a két részfára”**

```
Függvény Elemszám(P)
  Ha P = Nil akkor Vissza: 0
  Vissza: 1 + Elemszám(Elem[P].Bal)
            + Elemszám(Elem[P].Jobb)
Függvény vége

Függvény Magasság(P)
  Ha P = Nil akkor Vissza: -1
  Vissza: 1 + max(Magasság(Elem[P].Bal),
                  Magasság(Elem[P].Jobb))
Függvény vége
```

```
Függvény Levelek(P)
  Ha P = Nil akkor Vissza: 0
  Ha Elem[P].Bal = Nil és Elem[P].Jobb = Nil
    akkor Vissza: 1
  Vissza: Levelek(Elem[P].Bal)
        + Levelek(Elem[P].Jobb)
Függvény vége
```

További példák: összeg, maximum, keresés (O(n)), tükrözés, másolás, két fa összehasonlítása

A példafára: elemszám 9, levelek 4 (A, C, E, H), magasság 3.

Összefoglalás

- **Fa:** hierarchikus adatszerkezet; gyökér, levél, szülő, gyerek, részfa, mélység, magasság
- **Bináris fa:** legfeljebb két gyerek; szigorú, teljes, tökéletes, degenerált
- **Láncolt ábrázolás:** Bal és Jobb mutató · **tömbös:** gyerekek 2i és 2i + 1, szülő i div 2
- **Bejárások:** preorder (G-B-J), inorder (B-G-J), postorder (B-J-G) – rekurzívan, O(n); szintenként – sorral
- A fákon a műveletek **természetes módon rekurzívak**

Következő hét: bináris keresőfa – keresés, beszúrás, törlés; kiegyensúlyozottság; kupac.

Ellenőrző kérdések

1. Igaz vagy hamis: a bináris fa hierarchikus adatszerkezet.
2. Add meg a bejárási sorrendeket: gyökér 1; 1 bal gyereke 2, jobb gyereke 3; 2 gyerekei 4 és 5; 3 jobb gyereke 6.
3. A bináris fa elemeit egy A tömbben tároljuk (A[1] a gyökér, az **i**-edik elem gyerekei A[2i] és A[2i + 1]). A = [K, F, R, B, H, –, T]. Rajzold le a fát, és add meg az inorder bejárást!
4. Hány csomópontja lehet legfeljebb egy 4 magasságú bináris fának?
5. Melyik bejárás adja egy kifejezésfából a postfix alakot?
6. Mi lesz `Magasság(P)` értéke egyetlen csomópontból álló fára?

Gyakorlat

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Bináris fa	`BinaryTree.java`	`h14/binarytree`	négyféle bejárás, elemszám, levelek, magasság, oldalra döntött kiírás
Tömbös fa	`ArrayTree.java`	`h14/arraytree`	indexképletek, bejárás indexekkel
Kifejezésfa	`ExpressionTree.java`	`h14/expressiontree`	felépítés postfixből, kiértékelés, infix/prefix
Bejárás veremmel	`IterativeTraversal.java`	`h14/iterative`	preorder és inorder rekurzió nélkül

✎ Feladatok

1. **Összeg és maximum:** számítsd ki egy egész számokat tároló bináris fa elemeinek összegét és legnagyobb elemét!
2. **Egygyerekes csomópontok:** hány csomópontnak van pontosan egy gyereke?
3. **Szigorú fa:** döntsd el, hogy minden csomópontnak 0 vagy 2 gyereke van-e!
4. **k-adik szint:** írd ki a **k**-adik szinten lévő elemeket!
5. **Utak:** írd ki az összes gyökér–levél utat!
6. **Azonos fák:** döntsd el, hogy két bináris fa szerkezetre és értékekre is azonos-e!

Megoldások: `Exercises14.java`, `h14/exercises`

✎ Házi feladat

1. Írj függvényt, amely egy láncolt ábrázolású bináris fát **tömbös** ábrázolásúvá alakít (és fordítva)! Mekkora tömb kell?
2. Építsd fel a bináris fát a **preorder** és az **inorder** bejárásából! (Pl. preorder: F B A D C E G I H, inorder: A B C D E F G H I)
3. Írd ki a fát **szintenként, soronként** (minden szint új sorba kerüljön)!
4. Írj programot, amely egy mappa tartalmát **fa** formában írja ki (mint a `tree` parancs)! (Java: `File.listFiles()`, Go: `os.ReadDir`)