Adatszerkezetek

3. hét – Tömbök: egy- és többdimenziós tömbök, tárolás, dinamikus tömb

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Ismétlés

- **O-jelölés:** a lépésszám nagyságrendje – O(1), O(log n), O(n), O(n^2)
- **Lineáris keresés:** O(n), bármilyen tömbön
- **Bináris keresés:** O(log n), rendezett tömbön, **közvetlen elérés** kell

A mai óra kérdései

- Hogyan tárolja a gép a tömböt, és miért O(1) az elérés?
- Mennyibe kerül egy elem beszúrása vagy törlése?
- Hogyan lehet „bővíteni” egy statikus tömböt?

A tömb fogalma

A **tömb** azonos típusú elemek véges sorozata, amelynek elemeit **indexszel** érjük el.

Tulajdonság	Tömb
Elemek típusa	**homogén**
Méret	**statikus** – létrehozáskor rögzül
Tárolás	**folytonos** – az elemek közvetlenül egymás után
Elérés	**közvetlen** – bármelyik elem O(1) idő alatt
Kapcsolat	**lineáris**

Más néven: **vektor** (egydimenziós), **mátrix** (kétdimenziós)

Tömbök Java és Go nyelven

Java

```java
int[] a = new int[5];          // 0-kkal
int[] p = {2, 3, 5, 7, 11};
p[0] = 1;
int n = p.length;              // 5

for (int x : p)
    System.out.print(x + " ");

int[] b = a;                   // UGYANAZ a tömb
int[] c = Arrays.copyOf(p, p.length);
```

- A tömb **objektum**, a változó referencia
- Alapérték: `0` , `0.0` , `false` , `null`

Go

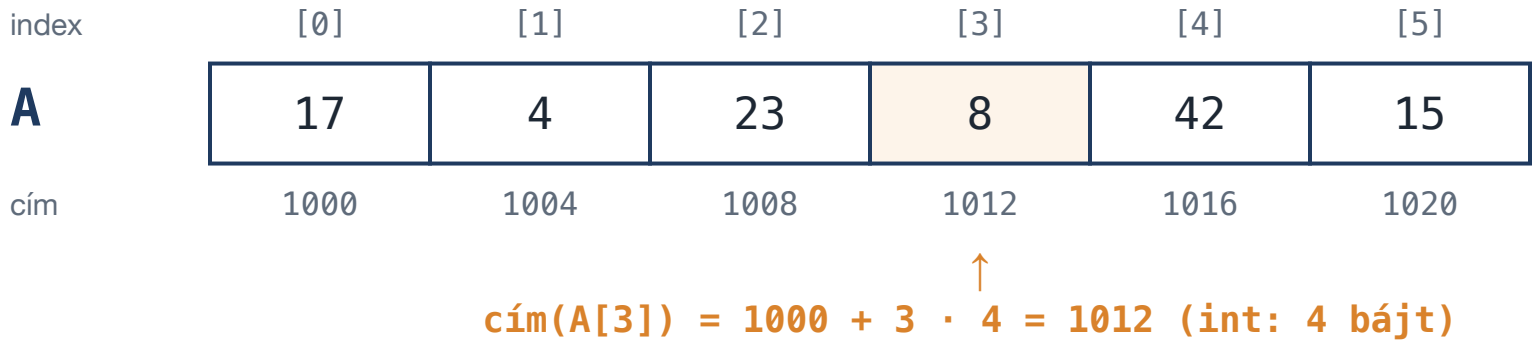
```go
var a [5]int                   // 0-kkal
p := [5]int{2, 3, 5, 7, 11}
p[0] = 1
n := len(p)                    // 5

for _, x := range p {
    fmt.Print(x, " ")
}

b := a                         // MÁSOLAT!
s := []int{2, 3, 5}            // szelet
```

- A méret a **típus része**: `[5]int` ≠ `[6]int`
- A tömb **érték** – a gyakorlatban szeletet (`[]int`) használunk

Tárolás a memóriában



Az **i**-edik elem címe **egyetlen számítással** megkapható:

> cím(A[i]) = kezdőcím + i · elemméret (0-tól indexelve)
> cím(A[i]) = kezdőcím + (i – 1) · elemméret (1-től indexelve)

Ezért **O(1)** az elérés – nem kell végigmenni az előző elemeken.

Miért indul 0-tól az index?

- A 0-tól indexelés esetén az index az elem **eltolása** (offset) a tömb elejétől
- `A[0]` a kezdőcímen van, `A[3]` három elemmel arrébb
- C, Java, Go, Python: 0-tól · Pascal, Fortran, Lua, a matematika: 1-től

A pszeudokódban **1-től** indexelünk, a programokban **0-tól**. Átíráskor:
`Ciklus i ← 1-től n-ig` → `for (int i = 0; i < n; i++)`

Indexhatár-ellenőrzés

- Java: `ArrayIndexOutOfBoundsException` · Go: `panic: index out of range`
- C-ben nincs ellenőrzés: a program csendben **felülírja** a szomszédos memóriát – súlyos biztonsági hibák forrása

Műveletek költsége

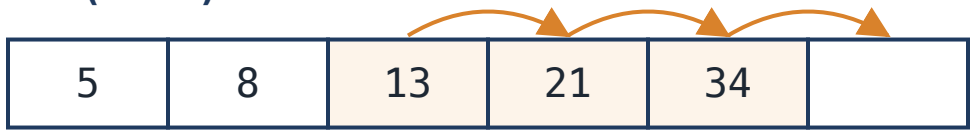
Művelet	Lépésszám	Miért?
i-edik elem olvasása / írása	**O(1)**	címszámítás
Keresés (rendezetlen)	O(n)	lineáris keresés
Keresés (rendezett)	O(log n)	bináris keresés
Beszúrás a végére (ha van hely)	O(1)	
Beszúrás az elejére / közepére	**O(n)**	az elemeket el kell tolni
Törlés az elejéről / közepéről	**O(n)**	az elemeket el kell tolni

A tömb **gyors olvasásra** és **lassú módosításra** – ha sokat szúrunk be és törlünk a közepén, a **láncolt lista** (9. hét) jobb választás lehet.

Beszúrás és törlés

Beszúrás a 2. indexre (x = 99)

előtte



utána



3 elem mozgatása jobbra

(hátulról kezdve!)

Törlés az 1. indexről

előtte



utána



3 elem mozgatása balra

(elölről kezdve)

Beszúrás és törlés – algoritmus

Beszúrás a k-adik helyre

```
Eljárás Beszúr(A, n, k, x)
  // feltétel: n < kapacitás
  Ciklus i ← n-től k-ig visszafelé
    A[i + 1] ← A[i]
  Ciklus vége
  A[k] ← x
  n ← n + 1
Eljárás vége
```

Törlés a k-adik helyről

```
Eljárás Töröl(A, n, k)
  Ciklus i ← k-tól n - 1-ig
    A[i] ← A[i + 1]
  Ciklus vége
  n ← n - 1
Eljárás vége
```

Java

```java
static int insert(int[] a, int n,
                  int pos, int x) {
    for (int i = n - 1; i >= pos; i--)
        a[i + 1] = a[i];
    a[pos] = x;
    return n + 1;
}
```

Beszúráskor **hátulról** kell kezdeni a mozgatást! Elölről haladva minden elemet felülírnánk az első eltolt értékkel.

Megfordítás és forgatás helyben

Megfordítás két indexszel – O(n), O(1) tár

```
Eljárás Megfordít(A, n)
  i ← 1 ; j ← n
  Ciklus amíg i < j
    csere(A[i], A[j])
    i ← i + 1 ; j ← j − 1
  Ciklus vége
Eljárás vége
```

`[1 2 3 4 5]` → `[5 4 3 2 1]`

Forgatás balra k hellyel – három megfordítással

`[1 2 | 3 4 5]`, k = 2

1. első k elem megfordítása: `[2 1 | 3 4 5]`
2. a többi megfordítása: `[2 1 | 5 4 3]`
3. az egész megfordítása: `[3 4 5 1 2]` ✓

Segédtömb nélkül, O(n) lépésben.

Többdimenziós tömbök

Kétdimenziós tömb (mátrix)

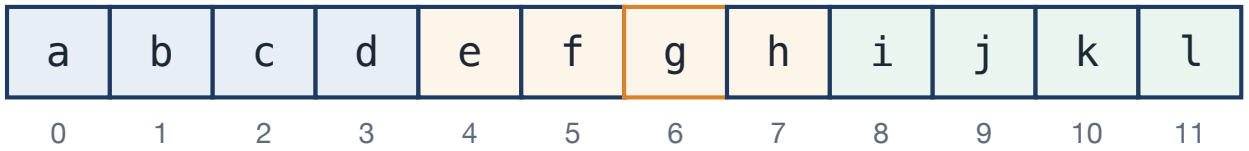
3 × 4-es mátrix

	0	1	2	3
0	a	b	c	d
1	e	f	g	h
2	i	j	k	l

`A[1][2] = g`

tárolás →

Sorfolytonos tárolás a memóriában




```
cím(A[i][j]) = kezdőcím + (i · oszlopszám + j) · elemméret
```

`A[1][2]: 1 · 4 + 2 = 6. elem`

- A memória **egydimenziós** – a mátrixot sorozattá kell „lapítani”
- **Sorfolytonos** tárolás (C, Java, Go): előbb az első sor, aztán a második …
- **Oszlopfolytonos** tárolás (Fortran, MATLAB): előbb az első oszlop …

Címszámítás: általános képlet

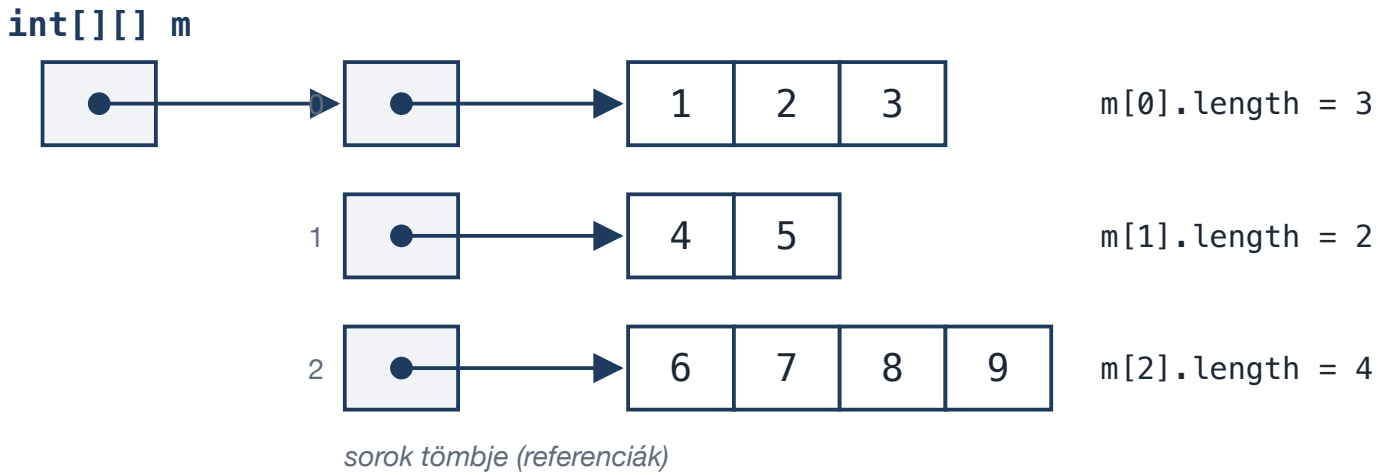
Egy **N** × **M**-es mátrix (N sor, M oszlop), elemméret **s**, kezdőcím **k**:

	0-tól indexelve	1-től indexelve
Sorfolytonos	$k + (i \cdot M + j) \cdot s$	$k + ((i - 1) \cdot M + (j - 1)) \cdot s$
Oszlopfolytonos	$k + (j \cdot N + i) \cdot s$	$k + ((j - 1) \cdot N + (i - 1)) \cdot s$

Példa: `int A[10][20]` (4 bájtos elemek), kezdőcím 2000, sorfolytonos, 0-tól indexelve.
cím(A[3][5]) = $2000 + (3 \cdot 20 + 5) \cdot 4 = 2000 + 260 =$ **2260**

Háromdimenziós tömb `A[i][j][l]`, méretek N × M × L: $k + ((i \cdot M + j) \cdot L + l) \cdot s$

2D tömbök Java-ban: tömbök tömbje



sorok tömbje (referenciák)

```
int[][] m = new int[3][4];          // 3 sor, mindegyik 4 hosszú
int[][] jagged = { {1, 2, 3}, {4, 5}, {6, 7, 8, 9} };
```

- Minden sor **külön objektum** → a sorok hossza eltérhet (**fogazott tömb**)
- `m.length` a sorok száma, `m[i].length` az **i**-edik sor hossza

2D tömbök Go-ban

Valódi 2D tömb – egy folytonos blokk

```
var grid [3][4]int
grid[1][2] = 7
```

- A méret fordításkor ismert
- Pontosan sorfolytonos tárolás

Szeletek szelete – mint Java-ban

```
m := make([][]int, rows)
for i := range m {
    m[i] = make([]int, cols)
}
```

- A méret futás közben adható meg
- A sorok hossza eltérhet

Mátrix bejárásakor a **sorfolytonos** sorrend (külső ciklus a sorokon) gyorsabb, mert jobban kihasználja a processzor gyorsítótárát (cache).

Mátrixműveletek

Főátló és mellékátló – O(n)

```
Ciklus i ← 1-től n-ig
  fő ← fő + A[i, i]
  mellék ← mellék + A[i, n - i + 1]
Ciklus vége
```

Transzponálás – O(n · m)

```
Ciklus i ← 1-től n-ig
  Ciklus j ← 1-től m-ig
    T[j, i] ← A[i, j]
  Ciklus vége
Ciklus vége
```

Mátrixszorzás – (n×k) · (k×m), **O(n · k · m)**

```
Ciklus i ← 1-től n-ig
  Ciklus j ← 1-től m-ig
    C[i, j] ← 0
    Ciklus x ← 1-től k-ig
      C[i, j] ← C[i, j] + A[i, x] · B[x, j]
    Ciklus vége
  Ciklus vége
Ciklus vége
```

Négyzetes mátrixokra **O(n³)** – három egymásba ágyazott ciklus.

Dinamikus tömb

A probléma: a tömb mérete rögzített

- Gyakran **nem tudjuk előre**, hány elemet kell tárolni (pl. beolvasott adatok)
- Túl kicsi tömb → nem fér el · túl nagy tömb → pazarlás

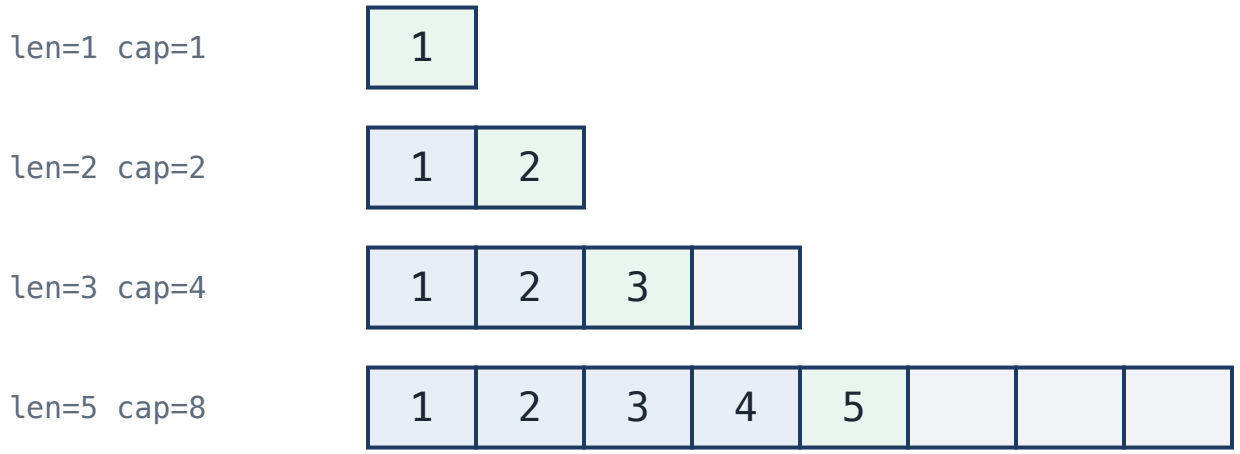
Megoldás: dinamikus tömb – egy tömb, amelyet szükség esetén **nagyobbra cserélünk**

- `size` (hossz): hány elem van benne ténylegesen
- `capacity` (kapacitás): mekkora a háttértömb

Java: `ArrayList` · Go: szelet + `append` · C++: `vector` · Python: `list`

A dinamikus tömb működése

append: ha betelt, kétszer akkora tömböt foglalunk, és átmásoljuk az elemeket



másolás: 1 + 2 + 4 + … < 2n

→ **egy append átlagosan O(1)**

(amortizált költség)

- Ha egyesével növelnénk: 1 + 2 + 3 + … + n = **O(n²)** másolás összesen
- **Duplázással:** 1 + 2 + 4 + … + n < 2n → összesen O(n), elemenként **O(1)**

Saját dinamikus tömb

Java

```java
class DynamicArray {
    private int[] data = new int[1];
    private int size = 0;

    void add(int x) {
        if (size == data.length) {
            int[] bigger =
                new int[2 * data.length];
            for (int i = 0; i < size; i++)
                bigger[i] = data[i];
            data = bigger;
        }
        data[size++] = x;
    }
    int get(int i) { return data[i]; }
}
```

Go

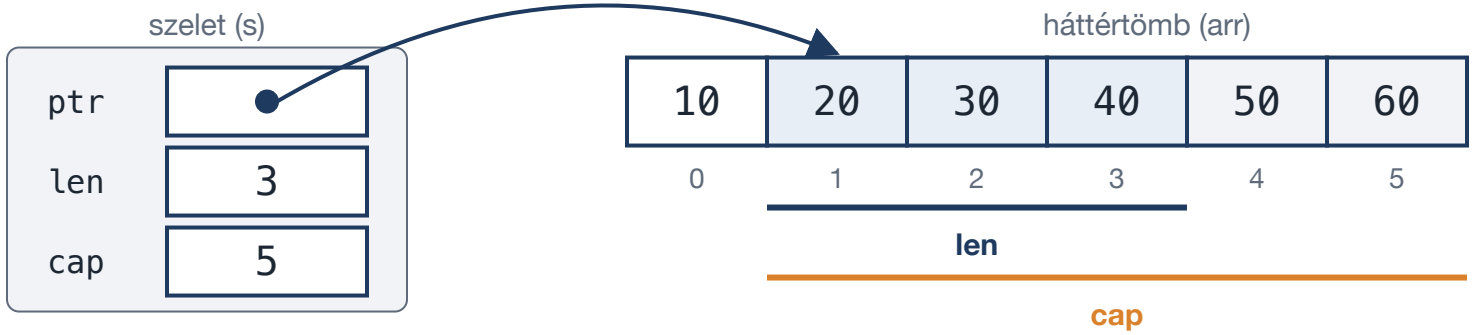
```go
type DynamicArray struct {
    data []int
    size int
}

func (d *DynamicArray) Add(x int) {
    if d.size == len(d.data) {
        bigger := make([]int, 2*len(d.data))
        copy(bigger, d.data[:d.size])
        d.data = bigger
    }
    d.data[d.size] = x
    d.size++
}
```

Mérés: 1 000 000 `add` → **1 048 575** másolás, elemenként **1,05**

Go szelet: hossz, kapacitás, háttértömb

```
s := arr[1:4] // len 3, cap 5
```




```
s[0] = 99 → arr[1] is 99 lesz! (közös háttértömb)
```

- A szelet három adatból áll: **mutató** a háttértömbre, **hossz** (`len`), **kapacitás** (`cap`)
- `append` csak akkor foglal új tömböt, ha a kapacitás elfogyott
- Ezért mindig így írjuk: `s = append(s, x)` – az eredményt vissza kell tenni!

Java `ArrayList`

```java
ArrayList<Integer> list = new ArrayList<>();
list.add(5);          // végére – amortizáltan O(1)
list.add(0, 3);       // beszúrás az elejére – O(n)
list.get(1);          // olvasás – O(1)
list.set(1, 9);       // írás – O(1)
list.remove(0);       // törlés index szerint – O(n)
list.size();          // elemszám
```

- Csak **objektumot** tárol: `int` helyett `Integer` (automatikus be- és kicsomagolás)
- Belül egy `Object[]` tömb, amit 1,5-szeresére növel, ha betelik

Összefoglalás

- A tömb **homogén, statikus, folytonos, közvetlen elérésű** adatszerkezet
- Címszámítás: `kezdőcím + index · elemméret` → elérés **O(1)**
- Beszúrás és törlés a közepén **O(n)** – elemeket kell mozgatni
- Mátrix: **sorfolytonos** tárolás, cím = k + (i · M + j) · s
- Java: 2D tömb = tömbök tömbje · Go: `[N][M]T` vagy `[][]T`
- **Dinamikus tömb:** kapacitásduplázás → hozzáfűzés amortizáltan **O(1)**

Következő hét: stringek – karakterek sorozata, kódolás, műveletek, mintakeresés.

Ellenőrző kérdések

1. Igaz vagy hamis: a tömb elemei szétszórtan helyezkednek el a memóriában.
2. Mekkora a lépésszáma egy **n** elemű tömb első elemének törlésének?
3. Egy `int A[5][8]` mátrix sorfolytonosan van tárolva a 100-as címtől, 4 bájtos elemekkel, 0-tól indexelve. Mi az `A[2][3]` elem címe?
4. Miért kell beszúráskor hátulról kezdeni az elemek mozgatását?
5. Miért duplázzuk a dinamikus tömb kapacitását, és miért nem növeljük eggyel?
6. Mit ír ki? `s := []int{1,2,3}; t := s; t[0] = 9; fmt.Println(s[0])`

Gyakorlat

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Alapok	`ArrayBasics.java`	`h03/arraybasics`	létrehozás, alapértékek, indexhatár, másolás
Műveletek	`ArrayOps.java`	`h03/arrayops`	beszúrás, törlés, megfordítás, forgatás
Mátrix	`Matrix.java`	`h03/matrix`	átlók, transzponálás, szorzás, fogazott tömb
Dinamikus tömb	`DynamicArray.java`	`h03/dynarray`	kapacitásduplázás, másolások száma, `append`

A `dynarray` Go programban figyeld meg, hogyan nő a beépített `append` kapacitása, és mi történik a közös háttértömbbel!

✎ Feladatok

1. **Átlag:** számítsd ki egy egészekből álló tömb elemeinek átlagát!
2. **Tartalmazás:** döntsd el, hogy a tömb tartalmaz-e egy adott értéket!
3. **Második legnagyobb:** keresd meg a második legnagyobb (különböző) elemet **egyetlen** bejárással!
4. **Ismétlődések törlése:** egy **rendezett** tömbből töröld helyben az ismétlődő elemeket, és add vissza az új hosszt! (`[1,1,2,3,3,3,5] → [1,2,3,5]`)
5. **Összefésülés:** fésülj össze két rendezett tömböt egy rendezett tömbbé O(n + m) lépésben!
6. **Sor- és oszlopösszegek:** számítsd ki egy mátrix minden sorának és oszlopának összegét!
7. **Spirál:** írd ki egy mátrix elemeit spirál sorrendben! (`1 2 3 6 9 8 7 4 5`)

Megoldások: `Exercises03.java` , `h03/exercises`

✎ Házi feladat

1. Egészítsd ki a saját `DynamicArray` osztályt `insert(i, x)` és `remove(i)` műveletekkel! Mi a lépésszámuk?
2. Írd meg a `remove` műveletet úgy, hogy ha a tömb **negyedéig** ürül, a kapacitás feleződjön! Miért nem már félig ürült állapotban felezünk?
3. Írj programot, amely egy **n** × **n**-es mátrixot **helyben** 90 fokkal elforgat az óramutató járásával megegyezően! (Tipp: transzponálás + a sorok megfordítása.)
4. Számold ki: `double T[50][40]` (8 bájtos elemek) oszlopfolytonosan, 1-től indexelve, kezdőcím 5000. Mi a `T[12][7]` címe?