Adatszerkezetek

4. hét – Stringek: ábrázolás, kódolás, műveletek, mintakeresés

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Ismétlés

- A tömb **homogén, statikus, folytonos, közvetlen elérésű**
- Címszámítás → elérés **O(1)** · beszúrás/törlés a közepén **O(n)**
- **Dinamikus tömb:** kapacitásduplázás → hozzáfűzés amortizáltan O(1)

A mai óra kérdései

- Hogyan tárolja a gép a szöveget – és miért lehet `len("世界") == 6`?
- Miért lassú a `+` összefűzés egy ciklusban?
- Hogyan keresünk meg egy szót egy szövegben?

A string fogalma

A **string** (karakterlánc, szöveg) karakterek véges sorozata.

- Lényegében egy **karaktertömb** – plusz szövegspecifikus műveletek
- **Hossz:** a karakterek száma · **üres string:** `""` (hossza 0)
- **Részstring:** egymást követő karakterek egy szakasza (`"data"` része a `"database"` -nek)

Tulajdonság	String
Elemek	homogén (karakterek)
Tárolás	folytonos
Elérés	közvetlen (index alapján)
Módosíthatóság	Java, Go: **nem módosítható** (immutable)

Ábrázolási módok

C: nullával lezárt karaktertömb



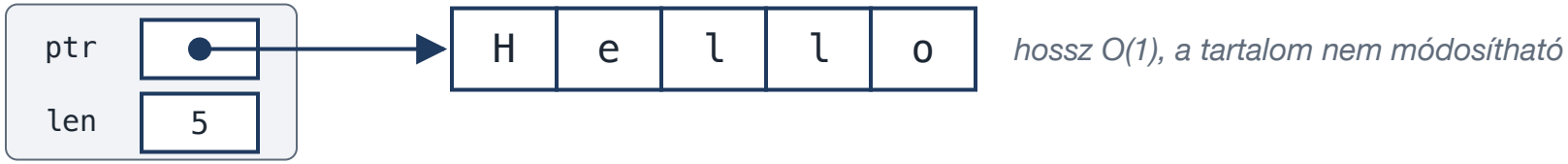
a hossz kiszámítása O(n): a \0-ig kell számolni

Pascal: hossz-előtag



a hossz O(1), de legfeljebb 255 karakter

Java / Go: leíró (hossz + mutató) + karaktertömb



hossz O(1), a tartalom nem módosítható

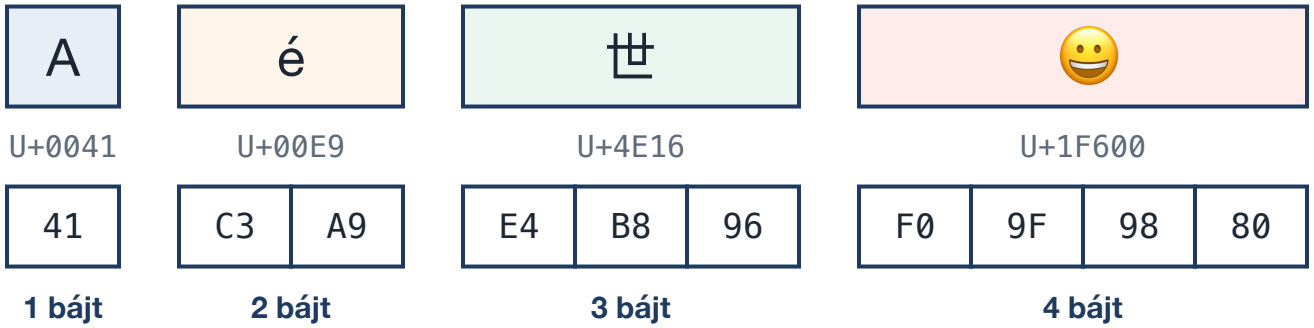
Karakterkódolás

ASCII, Unicode, UTF-8

- **ASCII** (1963): 7 bit, 128 karakter – angol ábécé, számjegyek, írásjelek
- **Unicode:** minden írásrendszer minden karaktere kap egy számot, a **kódpontot**
  `A` = U+0041 · `é` = U+00E9 · `世` = U+4E16 · `😀` = U+1F600
- A kódpontot bájtokká kell alakítani: **kódolás**

Kódolás	Egy karakter mérete	Ki használja?
UTF-8	1–4 bájt (ASCII = 1 bájt)	web, Linux, **Go**
UTF-16	2 vagy 4 bájt	**Java**, Windows, JavaScript
UTF-32	mindig 4 bájt	ritkán
GB 2312 / GBK / GB 18030	1–4 bájt	kínai szabványok

UTF-8: változó hosszúságú kódolás




```
Go: len("Aé世😀") = 10 bájt · 4 rune Java: "Aé世😀".length() = 5 char (UTF-16) · 4 kódpont
```

A „hány karakter?” kérdésre a válasz attól függ, mit számolunk: **bájtot**, **UTF-16 egységet** vagy **kódpontot**.

Karakterek Java-ban és Go-ban

Java – UTF-16

```java
String s = "Aé世😀";
s.length();                    // 5 (char)
s.codePointCount(0, s.length()); // 4
s.charAt(2);                   // '世'
s.getBytes(UTF_8).length;      // 10

// kódpontonként bejárni:
s.codePoints().forEach(cp -> ...);
```

- `char` = 16 bit → az emoji **két** `char` (surrogate pár)
- A kínai írásjegyek többsége **egy** `char`

Go – UTF-8

```go
s := "Aé世😀"
len(s)                         // 10 (bájt!)
utf8.RuneCountInString(s)      // 4
s[2]                           // 0xA9 — egy bájt!
[]rune(s)[2]                   // '世'

for i, r := range s {          // rune-onként
    fmt.Printf("%d %c\n", i, r)
}
```

- `s[i]` **bájtot** ad, a `range` **rune**-t
- Karakterenkénti munkához: `[]rune(s)`

Karakteraritmetika

Az angol betűk és a számjegyek kódjai **folytonosak**:

	Kód	Következmény
`'0'` … `'9'`	48 … 57	számjegy értéke: `c - '0'`
`'A'` … `'Z'`	65 … 90	betű sorszáma: `c - 'A'`
`'a'` … `'z'`	97 … 122	kisbetű → nagybetű: `c - 32`

```
int digit = '7' - '0';                    // 7
char upper = (char) ('g' - ('a' - 'A')); // 'G'
int[] freq = new int[26];
freq[c - 'a']++;                          // betűgyakoriság
```

A 26 elemű tömb a betűgyakoriság számlálásához egyszerű, de csak az angol ábécére működik. Általános esetben **hash táblát** használunk (11. hét).

Műveletek stringekkel

Nem módosítható (immutable) stringek

Java-ban és Go-ban egy string tartalma **létrehozás után nem változhat**.

```
String s = "hello";
s.toUpperCase();          // új stringet ad vissza, s nem változik!
s = s.toUpperCase();      // így kell
```

Előnyök:

- Biztonságosan megosztható (több változó, több szál)
- Kulcsként használható hash táblában
- Java: a **string pool** az azonos literálokat egyszer tárolja

Hátrány: minden „módosítás” új stringet hoz létre → **másolás**

Összefűzés ciklusban: O(n²)!

```java
// LASSÚ – O(n²)
String s = "";
for (int i = 0; i < n; i++)
    s += "x";   // új string + másolás
```

```java
// GYORS – O(n)
StringBuilder sb = new StringBuilder();
for (int i = 0; i < n; i++)
    sb.append("x");
String s = sb.toString();
```

Go: `strings.Builder` · `sb.WriteString("x")`

n	+=	Builder
10 000	10 ms	0,05 ms
40 000	46 ms	0,08 ms
160 000	624 ms	0,4 ms

(Go, mért értékek)

A `StringBuilder` belül **dinamikus tömb** (3. hét) – amortizáltan O(1) hozzáfűzés.

Összehasonlítás

Lexikografikus (szótári) sorrend: karakterenként balról jobbra, az első eltérés dönt; ha az egyik a másik eleje, a rövidebb a kisebb.

```
Függvény Összehasonlít(S, T)          // −1, 0 vagy 1
  i ← 1
  Ciklus amíg i ≤ hossz(S) és i ≤ hossz(T) és S[i] = T[i]
    i ← i + 1
  Ciklus vége
  Ha i ≤ hossz(S) és i ≤ hossz(T) akkor
    Ha S[i] < T[i] akkor Vissza: −1 különben Vissza: 1
  különben ha hossz(S) = hossz(T) akkor Vissza: 0
  különben ha hossz(S) < hossz(T) akkor Vissza: −1
  különben Vissza: 1
Függvény vége
```

`"apple" < "apply"` · `"data" < "database"` · `"Zebra" < "apple"` (nagybetű kódja kisebb!)

Gyakori hiba Java-ban: `==` és `equals`

```
String a = "data";
String b = "data";                // ugyanaz az objektum (string pool)
String c = new String("data");    // új objektum

a == b          // true  – véletlenül!
a == c          // false – más objektum
a.equals(c)     // true  – a tartalom egyezik
```

Java-ban stringeket **mindig** `equals` **-szal** hasonlítunk össze. A `==` a **referenciákat** veti össze (1. hét).

Go-ban a `==` a **tartalmat** hasonlítja össze: `a == c` → `true`

Műveletek: Java és Go

Művelet	Java	Go	Lépésszám
hossz	`s.length()`	`len(s)` (bájt)	O(1)
i-edik karakter	`s.charAt(i)`	`s[i]` (bájt)	O(1)
részstring	`s.substring(a, b)`	`s[a:b]`	O(b–a) / O(1)*
összefűzés	`s + t`, `s.concat(t)`	`s + t`	O(n + m)
egyenlőség	`s.equals(t)`	`s == t`	O(n)
összehasonlítás	`s.compareTo(t)`	`s < t`, `strings.Compare`	O(n)
keresés	`s.indexOf(t)`	`strings.Index(s, t)`	O(n · m)**
darabolás	`s.split(",")`	`strings.Split(s, ",")`	O(n)
csere	`s.replace(a, b)`	`strings.ReplaceAll(s, a, b)`	O(n)
kis-/nagybetű	`toLowerCase()`, `toUpperCase()`	`strings.ToLower`, `ToUpper`	O(n)

* Go-ban a részstring nem másol, a közös bájtokra mutat. ** A könyvtárak ennél okosabb algoritmusokat is használnak.

Klasszikus feladat: palindrom

Palindrom: visszafelé olvasva ugyanaz – `"radar"`, `"上海自来水来自海上"`

Pszeudokód

```
Függvény Palindrom(S, n)
  i ← 1 ; j ← n
  Ciklus amíg i < j és S[i] = S[j]
    i ← i + 1
    j ← j − 1
  Ciklus vége
  Vissza: i ≥ j
Függvény vége
```

Két index halad egymás felé – **O(n)**, O(1) tárigény.

Go – Unicode-biztosan

```go
func isPalindrome(s string) bool {
    r := []rune(s)
    for i, j := 0, len(r)-1; i < j;
            i, j = i+1, j-1 {
        if r[i] != r[j] {
            return false
        }
    }
    return true
}
```

Go-ban bájtonként vizsgálva a kínai szöveg **nem** lenne palindrom!

Mintakeresés

A mintakeresés feladata

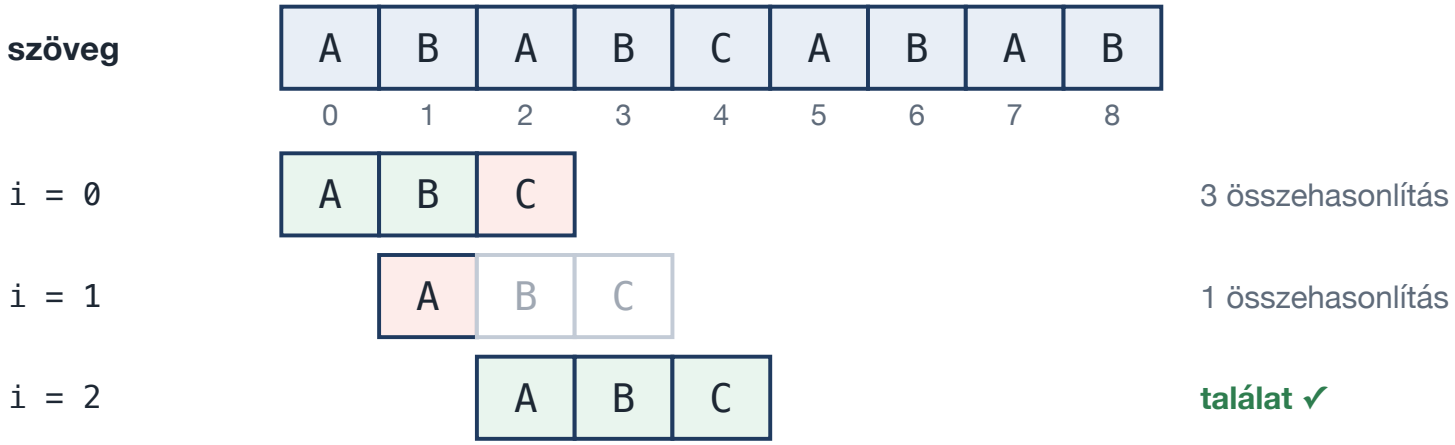
Adott: egy **n** hosszú **szöveg** (S) és egy **m** hosszú **minta** (P), $m \leq n$.
Feladat: hol fordul elő P először S-ben?

Alkalmazások: szövegszerkesztő „keresés”, `grep`, víruskeresők, **DNS-szekvenciák** elemzése, plágiumkeresés

Naiv (brute force) algoritmus:

- Próbáljuk ki a minta minden lehetséges kezdőpozícióját: $i = 1, 2, \ldots, n - m + 1$
- Minden pozícióban karakterenként hasonlítsuk össze a mintát a szöveggel
- Első eltérésnél lépjünk a következő pozícióra

Naiv mintakeresés – példa



Összesen **7** karakter-összehasonlítás

Naiv mintakeresés – algoritmus

Pszeudokód

```
Függvény NaivKeresés(S, n, P, m)
  Ciklus i ← 1-től n − m + 1-ig
    j ← 1
    Ciklus amíg j ≤ m és S[i + j − 1] = P[j]
      j ← j + 1
    Ciklus vége
    Ha j > m akkor Vissza: i
  Ciklus vége
  Vissza: 0
Függvény vége
```

Java

```java
static int naiveSearch(String s, String p) {
    int n = s.length(), m = p.length();
    for (int i = 0; i <= n - m; i++) {
        int j = 0;
        while (j < m &&
               s.charAt(i + j) == p.charAt(j))
            j++;
        if (j == m) return i;
    }
    return -1;
}
```

A naiv algoritmus hatékonysága

- **Legjobb eset:** a minta az elején van → O(m)
- **Legrosszabb eset:** minden pozícióban majdnem végig egyezik
  S = `aaaa…aab` (n = 10 001), P = `aa…ab` (m = 101)
  → **1 000 001** összehasonlítás ≈ n · m → **O(n · m)**
- Természetes szövegben az eltérés általában az első 1–2 karakternél kiderül → a gyakorlatban gyors

Hatékonyabb algoritmusok (csak említés szintjén)

- **Knuth–Morris–Pratt (KMP):** O(n + m) – a minta szerkezetéből tanul, nem lép vissza a szövegben
- **Boyer–Moore:** jobbról balra hasonlít, nagyokat ugrik – a gyakorlatban nagyon gyors
- **Rabin–Karp:** hash értékekkel hasonlít (11. hét)

Összefoglalás

- A string **karakterek sorozata**, folytonosan tárolva, közvetlen eléréssel
- Kódolás: **Unicode** kódpont → **UTF-8** (Go, web) vagy **UTF-16** (Java)
- A „hossz” lehet bájt, UTF-16 egység vagy kódpont – Go: `len(s)` bájtot ad!
- Java és Go stringjei **nem módosíthatók** → ciklusban `StringBuilder` / `strings.Builder`
- Java-ban `equals`, nem `==`!
- **Naiv mintakeresés:** O(n · m) legrosszabb esetben; jobb: KMP, Boyer–Moore

Következő hét: rendezések I. – buborék-, beszúrásos és kiválasztásos rendezés.

Ellenőrző kérdések

1. Mennyi Go-ban `len("中国")` értéke, és mennyi Java-ban `"中国".length()`?
2. Igaz vagy hamis: Java-ban az `s.toUpperCase()` hívás után `s` nagybetűs lesz.
3. Miért $O(n^2)$ a következő kód? `for (...) s += "x";`
4. Melyik a kisebb lexikografikusan: `"data"` vagy `"Data"`? Miért?
5. Hány karakter-összehasonlítást végez a naiv algoritmus, ha `S = "AAAAA"` és `P = "AAB"`?
6. Mit ír ki Java-ban: `new String("go") == "go"`?

Gyakorlat

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Alapok	`StringBasics.java`	`h04/stringbasics`	műveletek, `==` vs `equals`, összehasonlítás
Kódolás	`Encoding.java`	`h04/encoding`	bájt, `char`, kódpont, UTF-8 bájtok
Builder	`Builder.java`	`h04/builder`	`+=` vs `StringBuilder` időmérés
Mintakeresés	`PatternSearch.java`	`h04/patternsearch`	naiv keresés, összes előfordulás, legrosszabb eset

✎ Feladatok

1. **Palindrom:** döntsd el, hogy egy mondat palindrom-e (kis- és nagybetű, írásjelek figyelmen kívül hagyásával)! (`"A man, a plan, a canal: Panama"` ✓)
2. **Szavak száma:** hány szó van egy szövegben, ha a szavakat egy vagy több szóköz választja el?
3. **Betűgyakoriság:** számold meg az angol betűk előfordulását egy 26 elemű tömbbel!
4. **Caesar-kód:** told el a betűket **k** hellyel (`"Hello" → "Khoor"` , k = 3), és dekódold vissza!
5. **Anagramma:** ugyanazokból a betűkből áll-e két szó? (`listen` – `silent`)
6. **Tömörítés:** `"aaabbcdddd" → "a3b2c1d4"`
7. **Szavak megfordítása:** `"I love data structures" → "structures data love I"`

Megoldások: `Exercises04.java` , `h04/exercises`

✎ Házi feladat

1. Írd meg a lexikografikus összehasonlítást **könyvtári függvény nélkül** (`-1` , `0` , `1` visszatérési értékkel)!
2. Írj függvényt, amely visszaadja egy minta **összes** előfordulásának kezdőindexét – átfedő előfordulásokkal együtt! (`"aaaa"` -ban `"aa"` : 0, 1, 2)
3. Írj programot, amely kiírja egy szöveg **leggyakoribb** szavát! (Tipp: Java `HashMap<String,Integer>` , Go `map[string]int` .)
4. Go-ban: írd meg egy string megfordítását úgy, hogy a kínai karakterek és az emojik is helyesen jelenjenek meg!