Adatszerkezetek

10. hét – Strázsás lista, kétirányú lista, ciklikus lista

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Ismétlés

- **Egyszerű láncolt lista:** Fej → … → Nil; szekvenciális elérés
- Az elején és ismert elem **mögött**: O(1); a végén és keresés: O(n)
- **Strázsás keresés tömbben** (2. hét): a keresett elemet a végére tesszük → egy feltétel elég

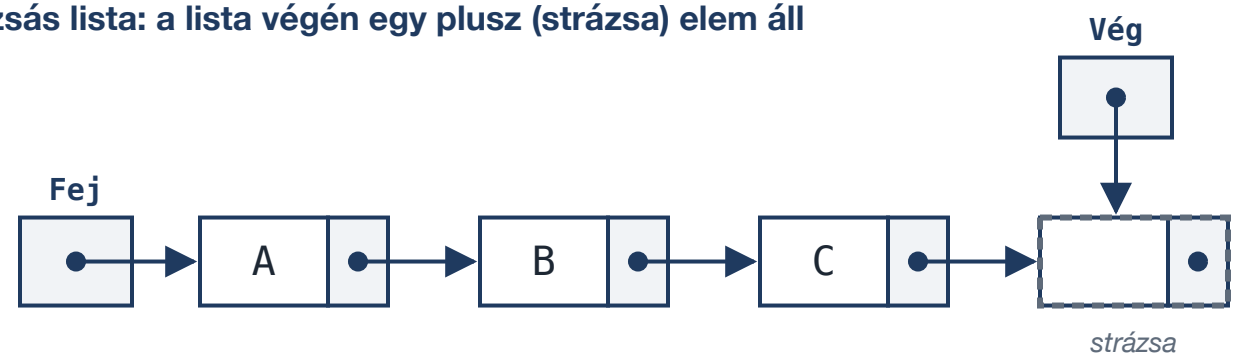
Az egyszerű lista gondjai

- Sok **speciális eset**: üres lista, első elem, utolsó elem
- A keresés ciklusának **két feltétele** van: `P ≠ Nil és Elem[P].Érték ≠ x`
- Egy elem **elé** nem tudunk beszúrni, és **magát** az elemet nem tudjuk törölni (csak a mögötte állót)

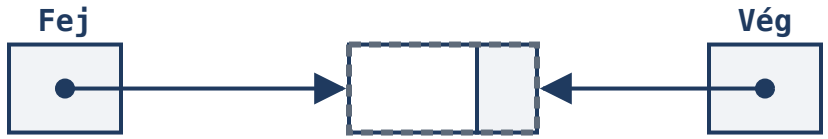
Strázsás lista

A strázsás lista felépítése

Strázsás lista: a lista végén egy plusz (strázsa) elem áll



Üres strázsás lista:



Fej = Vég (a tárigény NEM nulla!)

- A lista végén egy **strázsa** (őr) elem áll, amely nem tárol valódi adatot
- **Fej:** az első elemre, **Vég:** a strázsára mutat
- Üres lista: **Fej = Vég** – de **egy elem** (a strázsa) ekkor is foglal helyet!

A strázsás lista műveletei

Művelet	Ötlet	Lépésszám
Létrehozás	egy strázsa lefoglalása, Fej ← Vég ← strázsa	O(1)
Feldolgozás	Fej-től a Vég-ig (nem a Nil-ig!)	O(n)
Keresés	a keresett értéket a strázsába írjuk	O(n)
Beszúrás elé	másolás trükk	O(1)
Elem törlése	másolás trükk	O(1)
Kiürítés	Fej ← Vég (az elemek a szabad listára kerülnek)	O(1) – O(n)

Tulajdonság	**homogén · dinamikus · szétszórt · szekvenciális elérés**

Létrehozás és feldolgozás

Létrehozás

```
Eljárás Létrehoz(Fej, Vég)
  Ha Lefoglal(P) akkor
    Fej ← P
    Vég ← P
  különben
    HIBA("nincs szabad hely")
  Elágazás vége
Eljárás vége
```

Feldolgozás

```
Eljárás Feldolgoz(Fej, Vég)
  P ← Fej
  Ciklus amíg P ≠ Vég
    Művelet(Elem[P].Érték)
    P ← Elem[P].Köv
  Ciklus vége
Eljárás vége
```

Java

```java
class SentinelList {
    Node head, end;

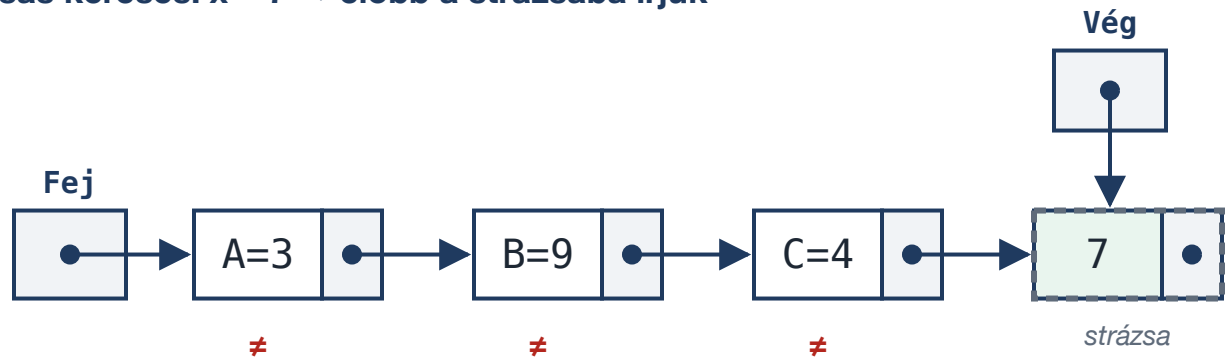
    SentinelList() {
        end = new Node();   // a strázsa
        head = end;
    }

    boolean isEmpty() { return head == end; }

    void print() {
        for (Node p = head; p != end; p = p.next)
            System.out.print(p.value + " ");
    }
}
```

Strázsás keresés

Strázsás keresés: x = 7 → előbb a strázsába írjuk

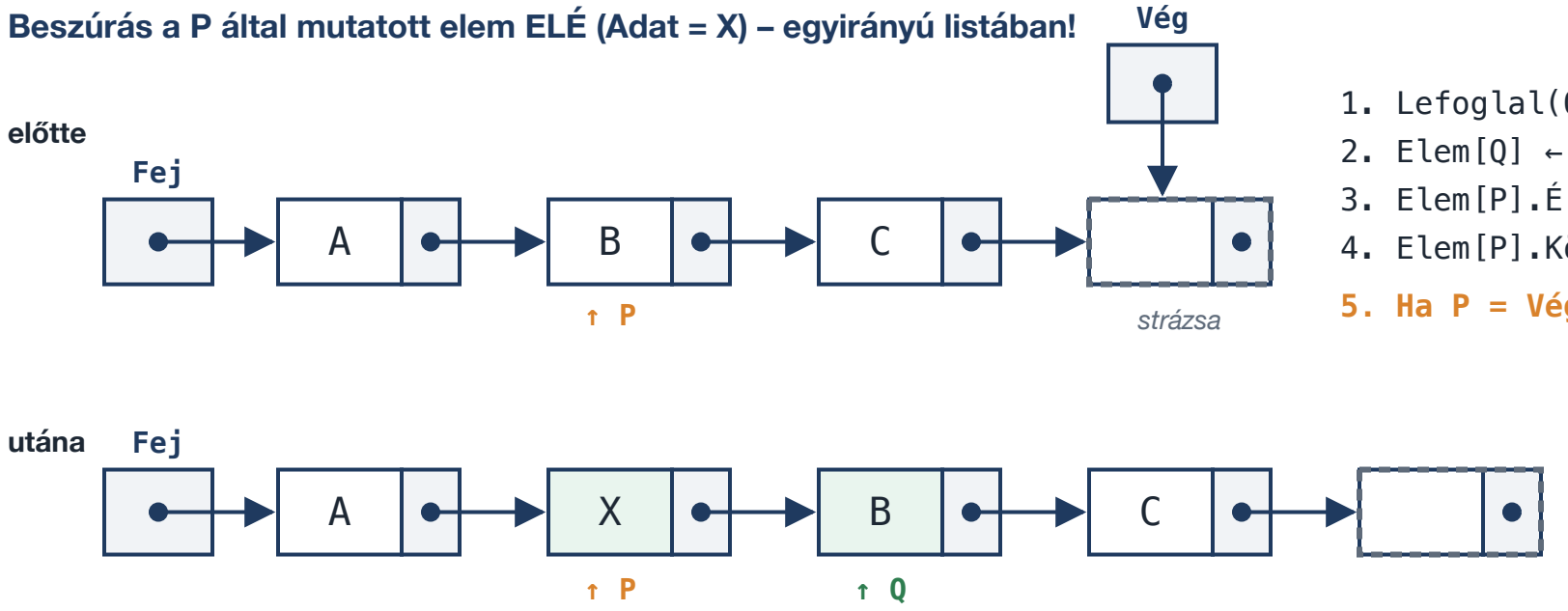


= → P = Vég: nincs benne

```
Függvény StrázsásKeres(Fej, Vég, x)
  Elem[Vég].Érték ← x                  // a strázsába írjuk
  P ← Fej
  Ciklus amíg Elem[P].Érték ≠ x        // egyetlen feltétel!
    P ← Elem[P].Köv
  Ciklus vége
  Ha P = Vég akkor Vissza: Nil  különben Vissza: P
Függvény vége
```

Beszúrás egy adott elem elé

Beszúrás a P által mutatott elem ELÉ (Adat = X) – egyirányú listában!




```
1. Lefoglal(Q)
2. Elem[Q] ← Elem[P] (másolat)
3. Elem[P].Érték ← Adat
4. Elem[P].Köv ← Q
5. Ha P = Vég akkor Vég ← Q
```

Trükk: P **elé** nem tudunk beszúrni (nem ismerjük az előzőt), de **mögé** igen. Ezért P **tartalmát** átmásoljuk az új elembe, és P-be írjuk az új adatot.

Beszúrás elé – pszeudokód és kód

Pszeudokód

```
Eljárás ElébeBeszúr(Fej, Vég, P, Adat)
  Ha Lefoglal(Q) akkor
    Elem[Q] ← Elem[P]       // Érték és Köv
    Elem[P].Érték ← Adat
    Elem[P].Köv ← Q
    Ha P = Vég akkor
      Vég ← Q               // Q az új strázsa
    Elágazás vége
  különben
    HIBA("nincs szabad hely")
  Elágazás vége
Eljárás vége
```

Beszúrás a végére = beszúrás a strázsa elé:

`ElébeBeszúr(Fej, Vég, Vég, Adat)` → **O(1)**!

Beszúrás az elejére = beszúrás a Fej elé

Java

```
void insertBefore(Node p, int x) {
    Node q = new Node();
    q.value = p.value;
    q.next = p.next;
    p.value = x;
    p.next = q;
    if (p == end) end = q;
}

void insertLast(int x)  { insertBefore(end, x); }
void insertFirst(int x) { insertBefore(head, x); }
```

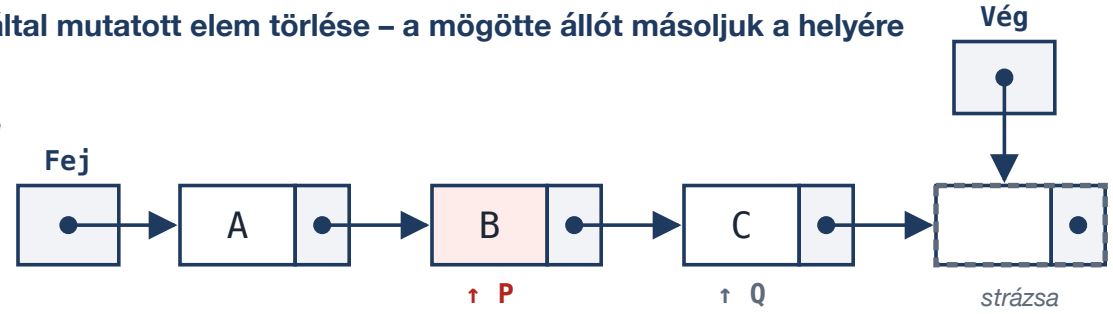
Go

```
func (l *SentinelList) InsertBefore(p *Node, x int) {
    q := &Node{Value: p.Value, Next: p.Next}
    p.Value = x
    p.Next = q
    if p == l.End {
        l.End = q
    }
}
```

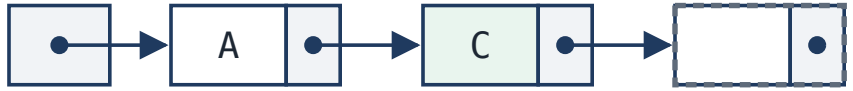
Egy adott elem törlése

A P által mutatott elem törlése – a mögötte állót másoljuk a helyére

előtte



utána



1. `Q ← Elem[P].Köv`
2. `Elem[P] ← Elem[Q]` (C a B helyére)
3. **`Ha Q = Vég akkor Vég ← P`**
4. `Felszabadít(Q)`

```
Eljárás Töröl(Fej, Vég, P)          // P ≠ Vég
  Q ← Elem[P].Köv
  Elem[P] ← Elem[Q]                 // a következő elem tartalma P-be
  Ha Q = Vég akkor Vég ← P          // P lett az új strázsa
  Felszabadít(Q)
Eljárás vége
```

Az **első** elem törlése sem speciális eset – a Fej nem változik!

Mit nyertünk a strázsával?

	Egyszerű lista	Strázsás lista
Üres lista	Fej = Nil	Fej = Vég (1 elem tárigény)
Keresés ciklusfeltétele	2 feltétel	**1 feltétel**
Beszúrás elé	nem megy (csak mögé)	**O(1)**
Adott elem törlése	nem megy (csak mögüle)	**O(1)**
Beszúrás a végére	O(n)	**O(1)**
Speciális esetek (első elem)	sok	**nincs**

A másolás trükk **megváltoztatja, melyik elem melyik értéket tárolja** – ha kívülről mutatók mutatnak az elemekre, azok „elcsúszhatnak”. Nagy rekordoknál a másolás is drága.

Rendezett strázsás lista

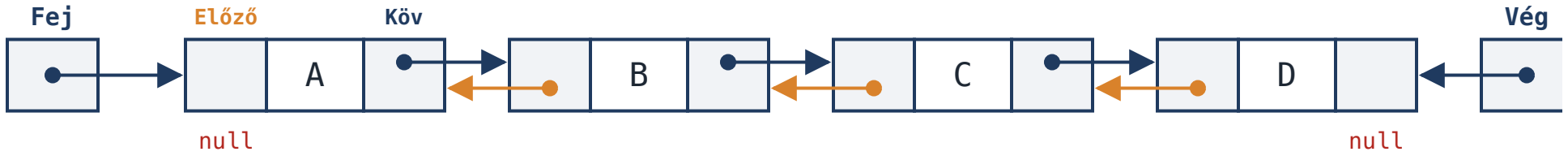
A strázsa a rendezett beszúrást is egyszerűsíti: a strázsába **végtelen nagy** értéket írunk, így a keresés biztosan megáll.

```
Eljárás RendezettBeszúr(Fej, Vég, Adat)
  Elem[Vég].Érték ← +∞
  P ← Fej
  Ciklus amíg Elem[P].Érték < Adat
    P ← Elem[P].Köv
  Ciklus vége
  ElébeBeszúr(Fej, Vég, P, Adat)
Eljárás vége
```

`5, 2, 8, 1, 9, 3` beszúrása után: `[1 2 3 5 8 9]` – üres listára, elejére, végére is ugyanaz a kód!

Kétirányú lista

A kétirányú lista felépítése



minden elem két mutatót tárol: az előzőre és a következőre

- Minden elem tárolja az **előző** és a **következő** elem címét is
- **Fej** és **Vég** mutató: a lista mindkét irányban bejárható
- Tárigény: elemenként **két** mutató

Kétirányú lista – műveletek

P elem törlése – O(1)

```
Eljárás Töröl(Fej, Vég, P)
  Ha Elem[P].Előző ≠ Nil akkor
    Elem[Elem[P].Előző].Köv ← Elem[P].Köv
  különben
    Fej ← Elem[P].Köv
  Elágazás vége
  Ha Elem[P].Köv ≠ Nil akkor
    Elem[Elem[P].Köv].Előző ← Elem[P].Előző
  különben
    Vég ← Elem[P].Előző
  Elágazás vége
  Felszabadít(P)
Eljárás vége
```

Java

```java
void remove(Node p) {
    if (p.prev != null) p.prev.next = p.next;
    else head = p.next;
    if (p.next != null) p.next.prev = p.prev;
    else tail = p.prev;
}

void insertLast(int x) {          // O(1)
    Node n = new Node(x);
    n.prev = tail;
    if (tail != null) tail.next = n;
    else head = n;
    tail = n;
}
```

Nincs szükség másolásra – **mindkét szomszédot ismerjük**.

Beszúrás elé/mögé, törlés a végéről: mind **O(1)**.

Kétirányú lista a gyakorlatban

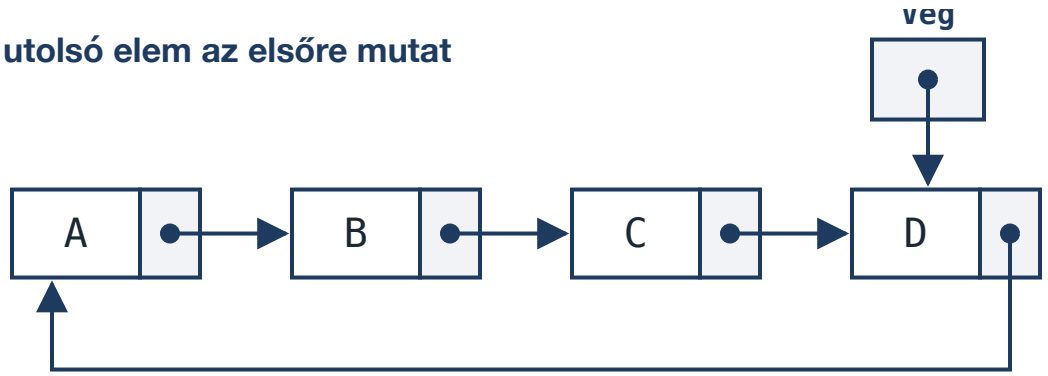
- Java `LinkedList`: kétirányú lista – `addFirst`, `addLast`, `removeFirst`, `removeLast` mind O(1)
- Go `container/list`: kétirányú, strázsás, ciklikus lista
- **Alkalmazások:**
  - böngésző előzmények (vissza **és** előre)
  - zenelejátszó: előző / következő szám
  - szövegszerkesztő: a kurzor mozgatása sorok között
  - **LRU gyorsítótár:** a legrégebben használt elem gyors törlése (hash táblával kombinálva)

> **Kétirányú strázsás lista:** egyetlen strázsa, amely egyszerre az első előtti és az utolsó utáni elem – így **minden** művelet speciális eset nélkül írható (a Go `container/list` is így működik).

Ciklikus lista

Ciklikus (körkörös) lista

Ciklikus lista: az utolsó elem az elsőre mutat




```
első = Elem[Vég].Köv
```

nincs Nil a láncban!

- Az utolsó elem **nem Nil**-re, hanem az **első** elemre mutat
- Elég egyetlen mutató az **utolsó** elemre: az első = `Elem[Vég].Köv` → beszúrás az elejére **és** a végére is O(1)
- Bejárás: az elsőtől addig, amíg **vissza nem érünk** (hátultesztelő ciklus!)

Alkalmazások: round-robin ütemezés, körbejáró lejátszási lista, Josephus-probléma, többjátékos körökre osztott játék

Az ismert listák összehasonlítása

	Egyszerű	Strázsás	Kétirányú	Ciklikus
Mutatók elemenként	1	1	**2**	1
Üres lista tárigénye	0 elem	**1 elem**	0 elem	0 elem
Beszúrás az elejére	O(1)	O(1)	O(1)	O(1)
Beszúrás a végére	O(n)	**O(1)**	**O(1)**	**O(1)**
Törlés a végéről	O(n)	O(n)	**O(1)**	O(n)
Adott elem törlése	O(n)*	**O(1)**	**O(1)**	O(n)*
Visszafelé bejárás	✗	✗	✓	✗

* az előző elemet meg kell keresni

Összefoglalás

- **Strázsás lista:** Fej + Vég, a Vég egy üres strázsa elemre mutat; üres lista: Fej = Vég
  - strázsás keresés **egy** feltétellel · beszúrás elé és törlés **másolás trükkel** O(1) · nincs speciális eset
- **Kétirányú lista:** Előző + Köv mutató; mindkét irányban bejárható; törlés O(1)
- **Ciklikus lista:** az utolsó az elsőre mutat; egy Vég mutató elég
- A választás a **leggyakoribb műveletektől** és a **tárigénytől** függ

Következő hét: listára épülő verem és sor, valamint a hash tábla.

Ellenőrző kérdések

1. Igaz vagy hamis: az üres strázsás lista tárigénye nulla, hiszen nem tárolunk benne elemet.
2. Igaz vagy hamis: a strázsás lista és az ugyanazt a sorozatot tároló kétirányú lista tárigénye mindig azonos, hiszen mindkettő két mutatóval (Fej és Vég) adható meg.
3. Írd meg pszeudokódban a strázsás keresést! A listát a Fej és Vég mutatók adják meg.
4. Rajzold le a `[3, 7]` strázsás listát, majd az állapotot a `7` elé történő beszúrás (Adat = 5) után!
5. Miért kell hátultesztelő ciklus a ciklikus lista bejárásához?
6. Melyik listát választanád, ha gyakran kell a lista **végéről** törölni?

Gyakorlat

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Strázsás lista	`SentinelList.java`	`h10/sentinellist`	létrehozás, strázsás keresés, elé beszúrás, törlés, rendezett beszúrás
Kétirányú lista	`DoublyLinkedList.java`	`h10/doublylist`	előre/visszafelé bejárás, O(1) törlés, beépített lista
Ciklikus lista	`CircularList.java`	`h10/circularlist`	bejárás, Josephus-probléma

A `SentinelList` a régi beadandó strázsás lista részének (inicializálás, feldolgozás, keresés, bővítés, törlés) teljes megoldása.

✎ Feladatok

1. **Maximum:** keresd meg a strázsás lista legnagyobb elemét!
2. **Számlálás strázsával:** hányszor fordul elő **x**? A belső ciklusnak csak egy feltétele legyen!
3. **Páros elemek törlése:** töröld a strázsás lista összes páros elemét a másolás trükkel!
4. **Palindrom:** kétirányú listán, két mutatóval a két végről!
5. **Ciklikus lista mérete:** számold meg a ciklikus lista elemeit!

Megoldások: `Exercises10.java`, `h10/exercises`

✎ Házi feladat

1. Valósítsd meg a **kétirányú strázsás** listát egyetlen strázsával (az első előtti és az utolsó utáni elem ugyanaz)! Írd meg a beszúrás elé, beszúrás mögé és a törlés műveletet – egyetlen `if` nélkül!
2. Fésülj össze két rendezett strázsás listát egy harmadik rendezett strázsás listába!
3. Valósíts meg egy **LRU gyorsítótárat** kétirányú listával: a legutóbb használt elem kerüljön az elejére; ha betelt, a végéről töröljünk!
4. Mérd meg: hány elemnél lesz a strázsás keresés észrevehetően gyorsabb az egyszerű keresésnél?