4. Gyakorlat

Algoritmusok és adatszerkezetek 1.

Sor láncolt ábrázolása

Sor

Queue[2]
-first, last: E1* *//a sor első és utolsó elemére mutató pointerek* -size: $\mathbb{N}$
+ Queue() + add(x: $\mathbb{T}$) *// új elem hozzáadása a sor végére* + rem(): $\mathbb{T}$ *// a sor elején lévő elem eltávolítása* + first() : $\mathbb{T}$ *// a sor elején lévő elem lekérdezése* + length(): $\mathbb{N}$ + isEmpty(): $\mathbb{B}$ + ~Queue() + setEmpty()

Feladat

Adjuk meg a láncolt ábrázolású sor műveleteit a veremhez hasonló módon.

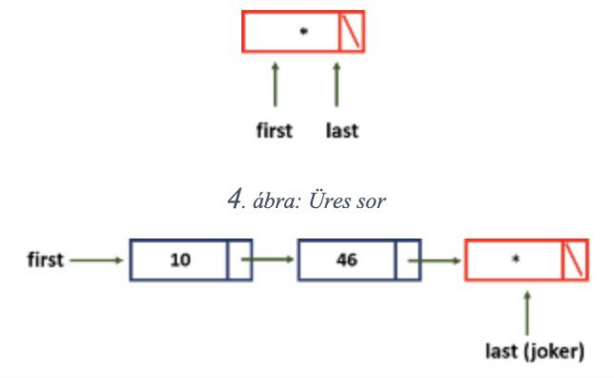
Legyen egy joker elemünk, ami egy “félkész” elem.

Ha üres a sor, csak ez az elem van a listában.

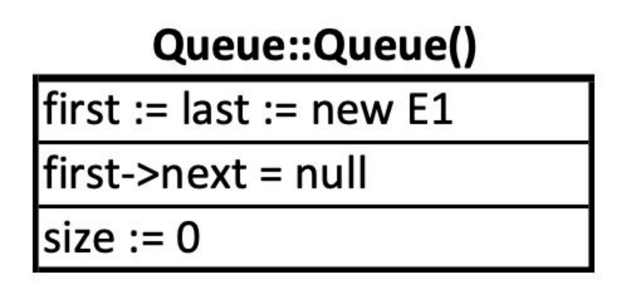
A first mutasson az első elemre a láncolt listában.

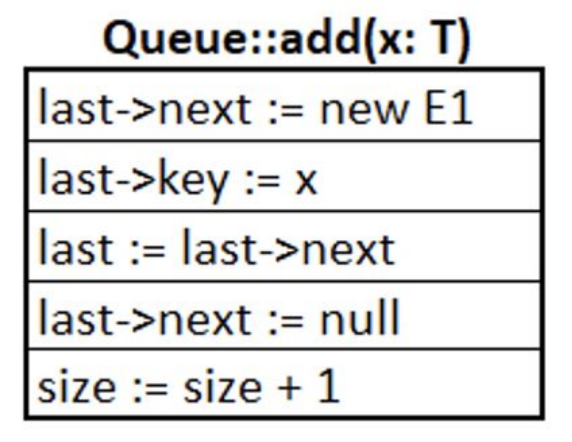
A last mutasson az utosó utáni elemre, ami a joker.

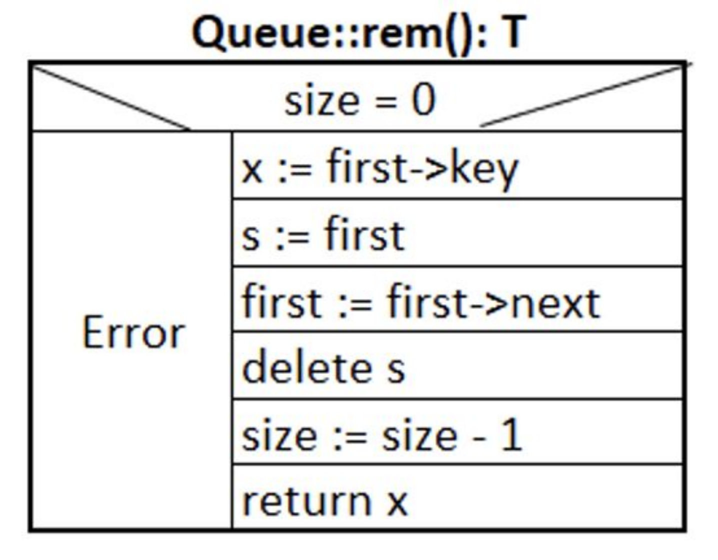
Beszúrásnál a joker elemet kitöltjük a beszúrt elemmel és létrehozzuk az új joker elemet.



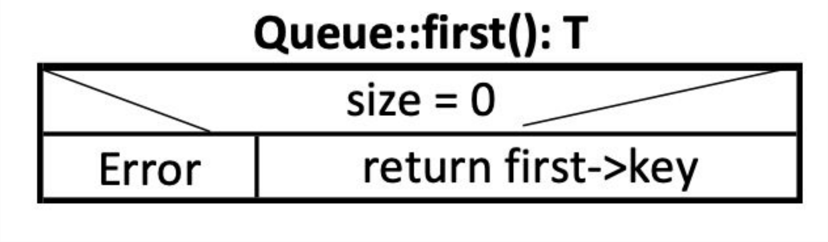
Struktogramok

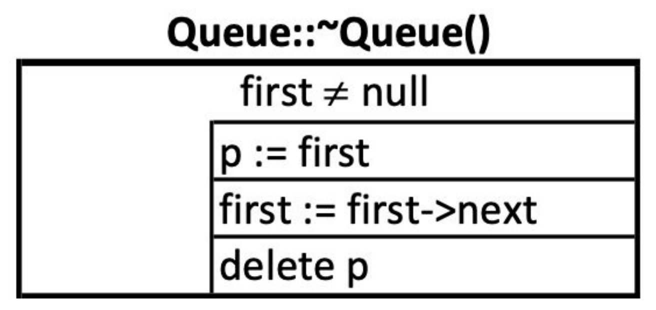


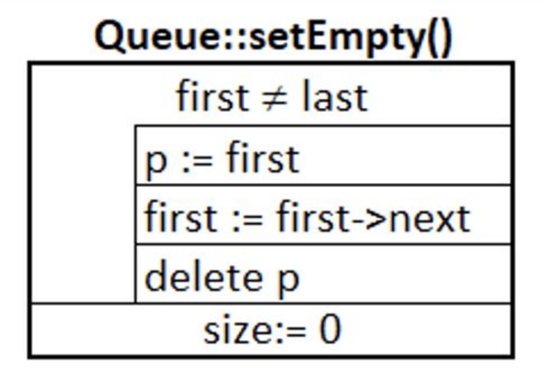




Struktogramok



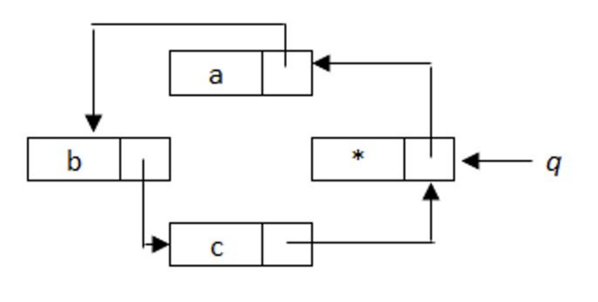




Feladat

Gondoljuk át, hogy hogyan lehetne egyetlen pointerrel ábrázolni a sort?

Ötlet: a joker elem next mutatója üres, azt nem használjuk a fenti megvalósításban Semmire. Használjunk ciklikus listát, a joker elem next mutatóját alkalmazzuk a first helyett!



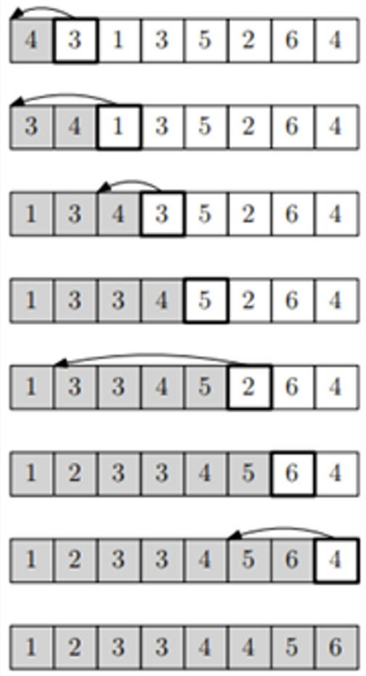
Beszúró rendezés

Beszúró rendezés algoritmusa

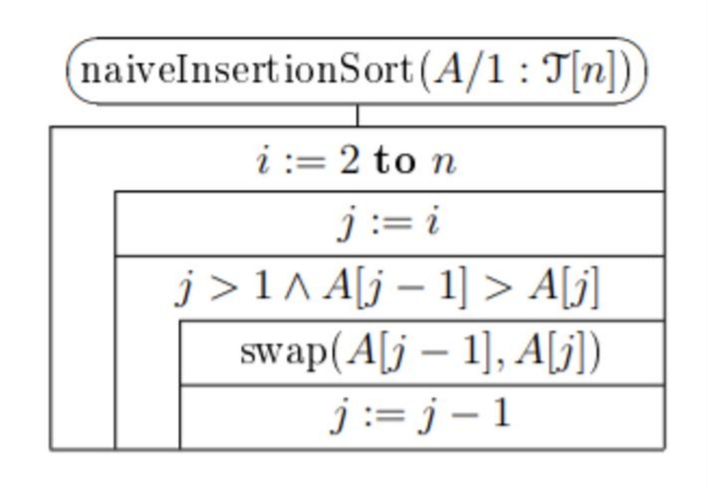
- Feladat: rendezzünk egy számokat tartalmazó A[n] tömböt nagyság szerint.
- Stratégia, tételezzük fel, hogy az i-edik lépésben a tömbünk 1..i része már rendezve van.
  Ekkor az i+1-edik elemnek keressük meg a helyét és szúrjuk be.

Hogyan:

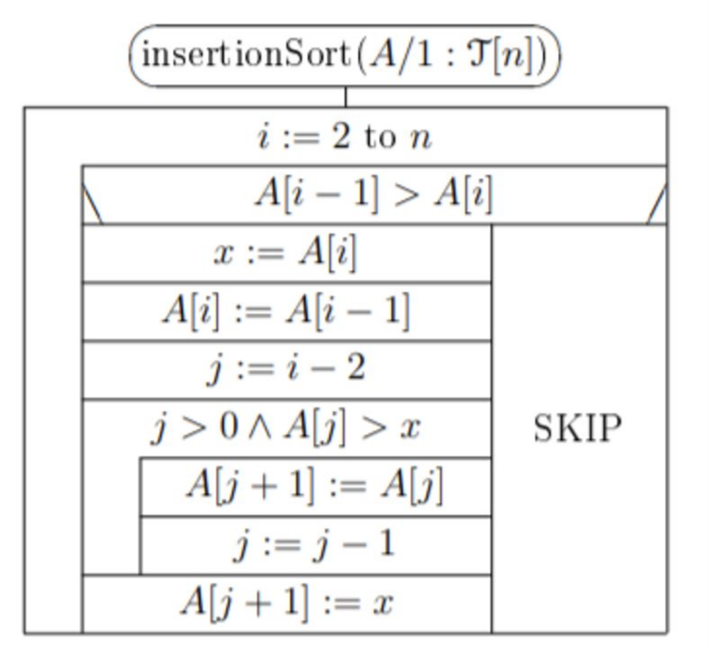
- Lehetőség 1: keressük meg a helyét, aztán shifteljük az összes elemet
- <u>Lehetőség 2: folyamatos cserékkel egyszerre keressük meg és visszük a helyére</u>

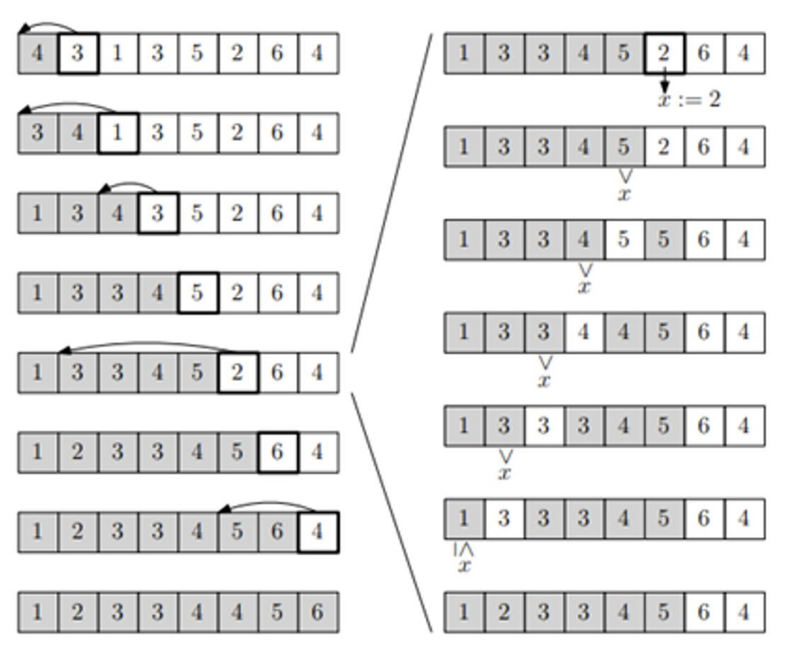


Naiv beszúró rendezés



Trükkös





Miért jobb ez? A swap(x,y) az 3 értékadás és egy segédváltozó, itt az üres hely megteremtésével ezt egyetlen értékadásra redukáltuk!

Beszúró rendezés tulajdonságai

Hatékonyság:

$mT(n) = n$, $MT(n)=n^2/2$, $AT(n)=n^2/4$

$mT(n) = \Theta(n)$, $MT(n)=AT(n)=\Theta(n^2)$

Tárigény: $O(1)$

Stabilitás*: A beszúró rend stabil, mert a rendezett szakaszba való beszúráskor az elem helyét jobbról balra keresi meg, a vele egyenlőket nem lépi át.

**: Egy rendező eljárást stabilnak nevezünk, ha nem változtatja meg az egyenlő kulcsú elemek egymáshoz viszonyított sorrendjét.*

Feladat

A beszúró rendezéssel rendezzük az alábbi tömböt.

A tömb 5. pozícióján található 1-es elem helyrevitelét lépésről lépésre mutassuk be.

`[8,4,10,6,1,9,3]`

[8,4,10,6,1,9,3]

[4,8,10,6,1,9,3]

[4,8,10,6,1,9,3]

[4,6,8,10,1,9,3]

[1,4,6,8,10,9,3]

[1,4,6,8,9,10,3]

[1,3,4,6,8,9,10]

[4,6,8,10,1,9,3] x=1

[4,6,8,10,10,9,3]

[4,6,8,8,10,9,3]

[4,6,6,8,10,9,3]

[4,4,6,8,10,9,3]

[1,4,6,8,10,9,3]

Megoldás

```
[8,4,10,6,1,9,3]
[4,8,10,6,1,9,3]
[4,8,10,6,1,9,3]
[4,6,8,10,1,9,3]
[1,4,6,8,10,9,3]
[1,4,6,8,9,10,3]
[1,3,4,6,8,9,10]
```

```
[4,6,8,10,1,9,3]
[4,6,8,10,10,9,3] x=1
[4,6,8,8,10,9,3]
[4,6,6,8,10,9,3]
[4,4,6,8,10,9,3]
[1,4,6,8,10,9,3]
```

MergeSort

MergeSort algoritmus

$\mathrm{mergeSort}(A : \mathcal{T}[n])$

$//$ sort $A[0..n)$
$\mathrm{ms}(A, 0, n)$

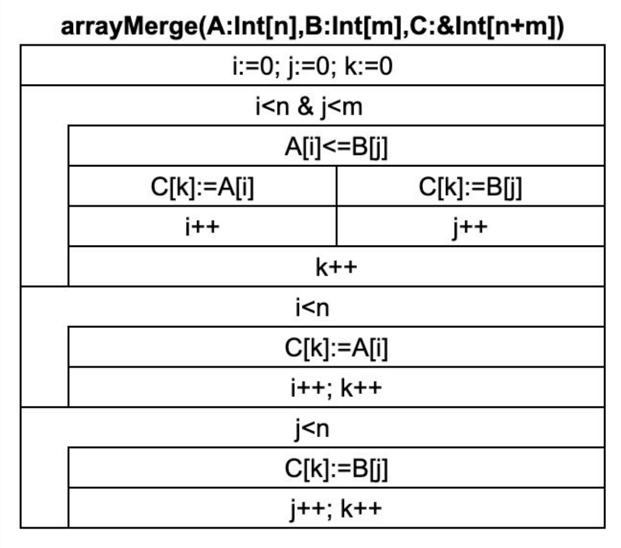
$\mathrm{ms}(A : \mathcal{T}[n]\ ;\ u, v : [0..n])$

$//$ sort $A[u..v)$	
$u < v - 1$	
$m := \left\lfloor \frac{u+v}{2} \right\rfloor$	SKIP
$\mathrm{ms}(A, u, m)$	
$\mathrm{ms}(A, m, v)$	
$\mathrm{merge}(A, u, m, v)$	

Két rendezett tömb összefésülése

Adott A[1..n] és B[1..m] rendezett tömbök és a C[1..n+m] tömb.

Fésüljük össze C-ben az A és B tömböket!



Merge segédalgoritmus

A tulajdonképpeni összefésülést a merge(A, u, m, v) eljárás első explicit ciklusa végzi el, az utolsó ciklussal kiegészítve.

A Z[0..d) segédtömbre azért van szükség, hogy az összefésülés során az output ne írja felül az inputot.

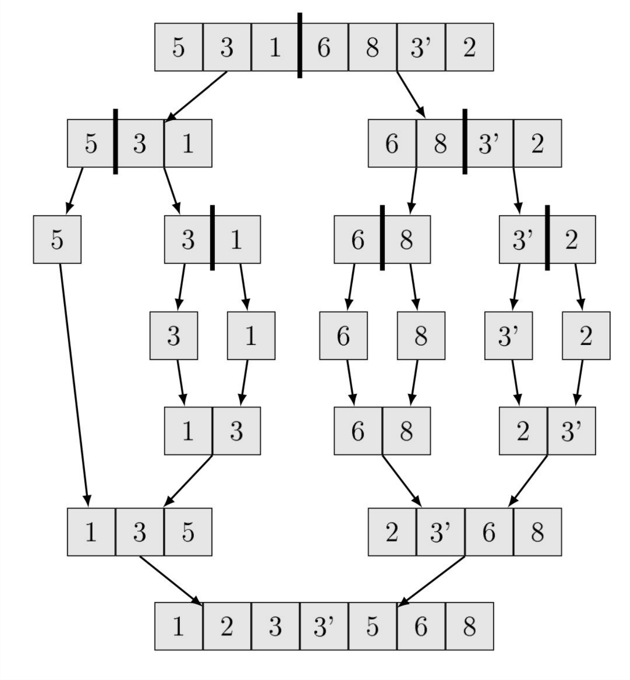
A triviális megoldás mindkét résztömböt átmásolná, de elég a bal oldalit, mert az összefésülés első explicit ciklusa során végig igaz lesz, hogy k < i.

A végén miért csak a Z tömb végét kell átmásolni? Mert ha az A[] tömb vége már ott van!

$\mathrm{merge}(A : \mathcal{T}[n] \;;\; u, m, v : [0..n])$

- // sorted merge of $A[u..m)$ and $A[m..v)$ into $A[u..v)$
- $d := m - u$ // d is the length of $A[u..m)$.
- $Z : \mathcal{T}[d] \;;\; Z[0..d) := A[u..m)$
- // sorted merge of $Z[0..d)$ and $A[m..v)$ into $A[u..v)$
- $k := u$ // copy into $A[k]$
- $j := 0 \;;\; i := m$ // from $Z[j]$ or $A[i]$
- $i < v \wedge j < d$
  - $A[i] < Z[j]$
    - $A[k] := A[i]$; $i := i + 1$
    - $A[k] := Z[j]$; $j := j + 1$
  - $k := k + 1$
- $j < d$
  - $A[k] := Z[j]$
  - $k := k + 1 \;;\; j := j + 1$

Szemléltetés



Feladat

Rendezzük a MergeSort algoritmussal az alábbi 15 elemű tömböt:

```
[15,34,12,11,16,21,22,10,46,17,18,40,22,23,31]
```

Lejátszás

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	34	12	11	16	21	22	10	46	17	18	40	22	23	31
15	34	12	11	16	21	22	10	46	17	18	40	22	23	31
15	34	12	11	16	21	22	10	46	17	18	40	22	23	31
15	34	12	11	16	21	22	10	46	17	18	40	22	23	31
15	34	12	11	16	21	22	10	46	17	18	40	22	23	31
15	12	34	11	16	21	22	10	46	17	18	22	40	23	31
12	15	34	11	16	21	22	10	17	18	46	22	23	31	40
11	12	15	16	21	22	34	10	17	18	22	23	31	40	46
10	11	12	15	16	17	18	21	22	22	23	31	34	40	46

Láncolt listák

Ismétlés: Pointerek koncepciója

- Tömbben megvalósított láncolt lista
- A tömb elemei két mezőből állnak, egy szöveg és egy egész értékből.
- Az egész érték egy tömbindex, mely egy láncba fűzi a tömb elemeit.

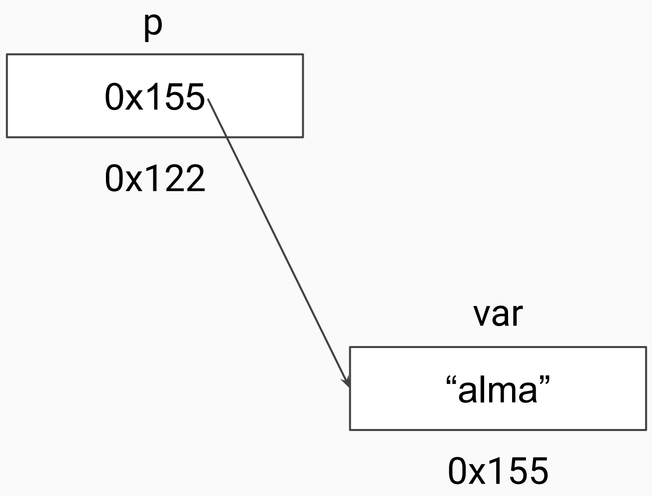
1	málna	0
2		0
3	banán	8
4		5
5		2
6	körte	1
7	alma	3
8	eper	6

L=7

SZH=4

Ismétlés: Pointerek / mutatók

- Ez is egy típus
- Egy változó, amelyben egy memóriacím van
  - p: a változó amely pointer típusú
  - *p vagy p-> a p által mutatott változó
    - Esetlet p^ régi jegyzetben
  - Tudunk adattagokra hivatkozni
    - p->key
  - Tudunk a változóval műveletet végezni
    - (*p) = körte
  - Ha üres, akkor nullpointer (NULL esetleg nil)
- A pointer típusos

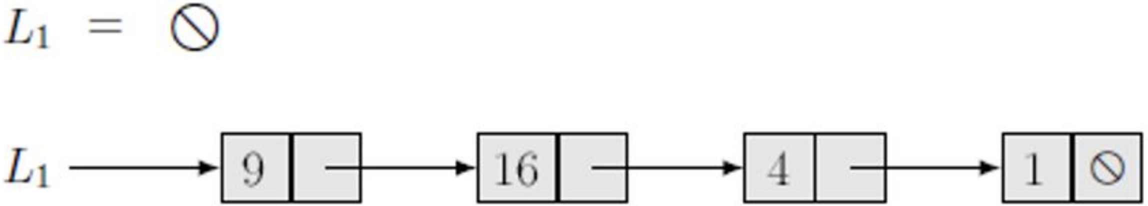


Láncolt listák

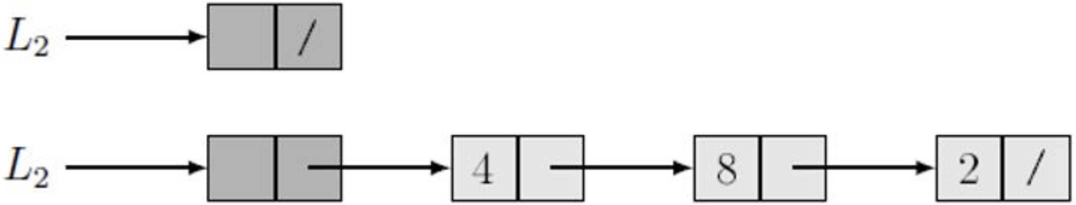
- Egy vagy két irányúak lehetnek.
- Előny: a rendezett beszúrás/törlés nem igényel elemmozgatást. Persze a beszúrás/törlés helyének megtalálása rendezett esetben O(n).
- Hátrány: nem indexelhető konstans műveletszámmal, csak O(n)-nel!

Egyirányú lista

- Listaelem: E1 osztály {key: Típus, next: E1*}
    - Tehát az E1 egy kulcsból és egy E1 típusú elemre mutató pointer
    - Megengedjük a p->key := X típusú értékadásokat, tehát ez inkább egy rekord, mint egy osztály
- Egyszerű, egyirányú láncolt lista (S1L): a legegyszerűbb forma, az első elemre egy pointer mutat. Ha még nincs eleme a listának, ez a pointer 0 (Null, Nil) értékű.
- Fejelemes egyirányú láncolt lista (H1L): Gyakori trükk, hogy egy valódi adatot nem tároló elemet helyezünk el a lista elejére. Célja: a lista elején (vagy az üres listával) végzett műveletek megkönnyítése



3. ábra: egyszerű láncolt lista



4. ábra Fejelemes egyirányú láncolt lista