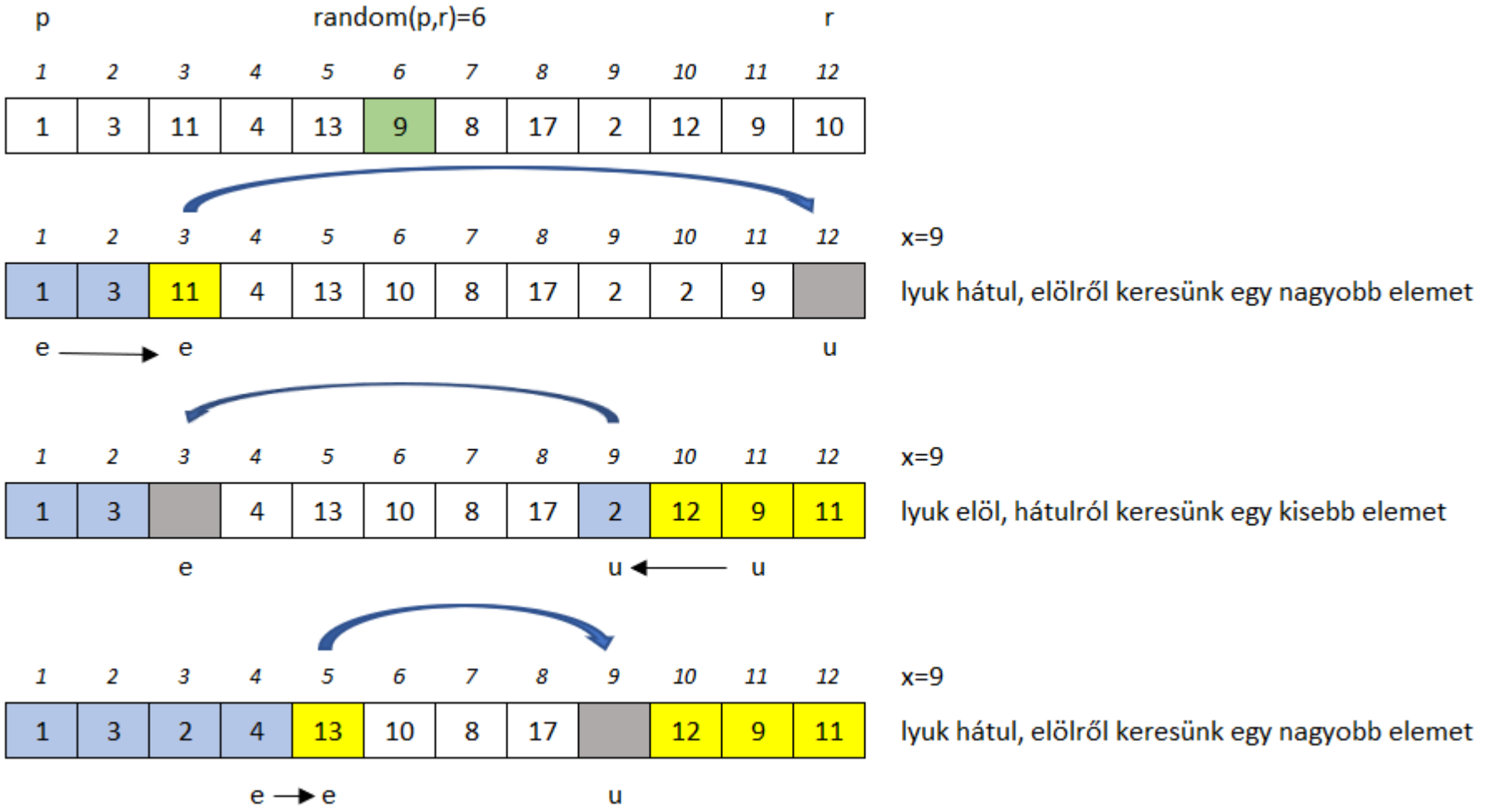
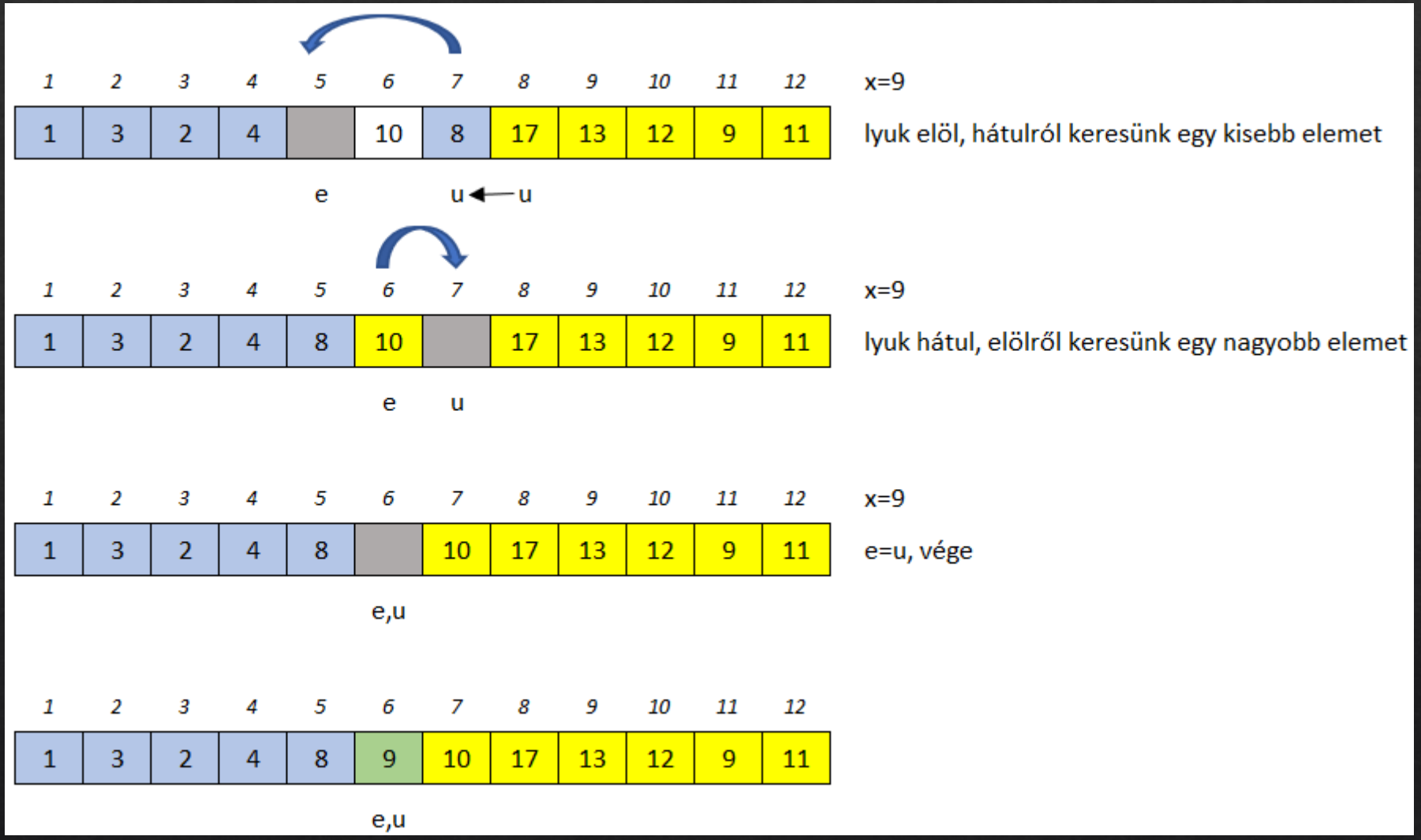
Algoritmusok és adatszerkezetek I.

4. gyakorlat

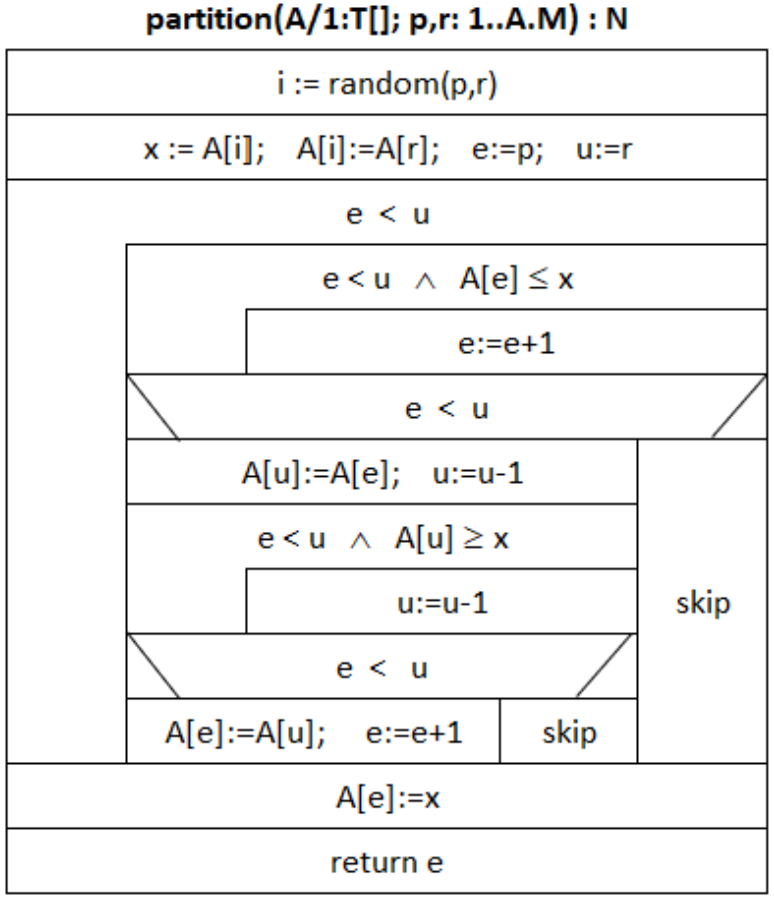
Quicksort particionálás



Quicksort particionálás



Quicksort particionálás algoritmus



"hátul" képzünk egy lyukat, e-elölről lépeget, u-hátulról

hátul van a lyuk (az u indexen), elölről keresünk a lyukba illő elemet

ha találtunk,

betesszük a lyukba, így elöl keletkezik a lyuk (az e indexen)

most hátulról indulva keresünk egy a lyukba illő elemet

ha találtunk,

betesszük a lyukba, így a lyuk megint hátul lesz (az u indexen)

végül e és u egyenlő lesz, így mindkettő a lyuk helyét jelenti

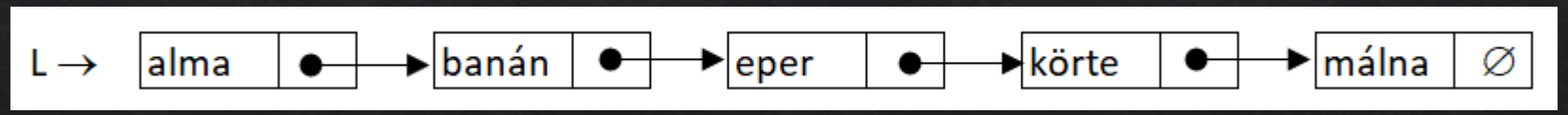
- Műveletigény: lineáris : O(r-p)
- Tengely elem mindegyik elemmel összehasonlítódik.
- Mozgatás: legjobb esetben konstans, legrosszabb esetben (r-p)

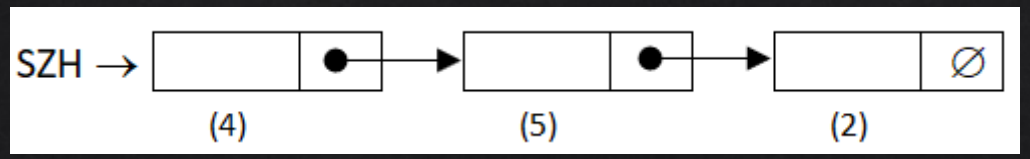
Láncolt listák

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

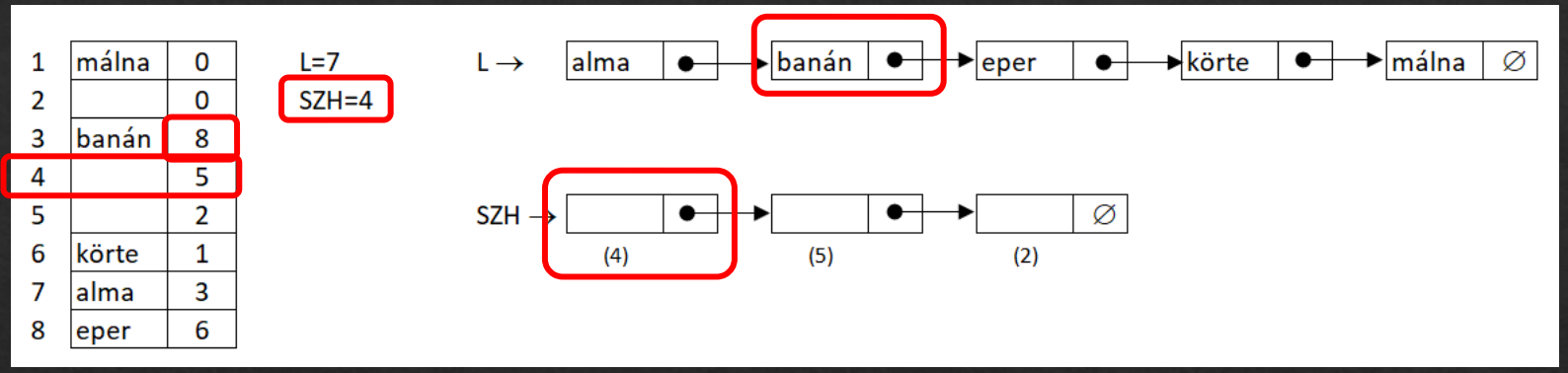


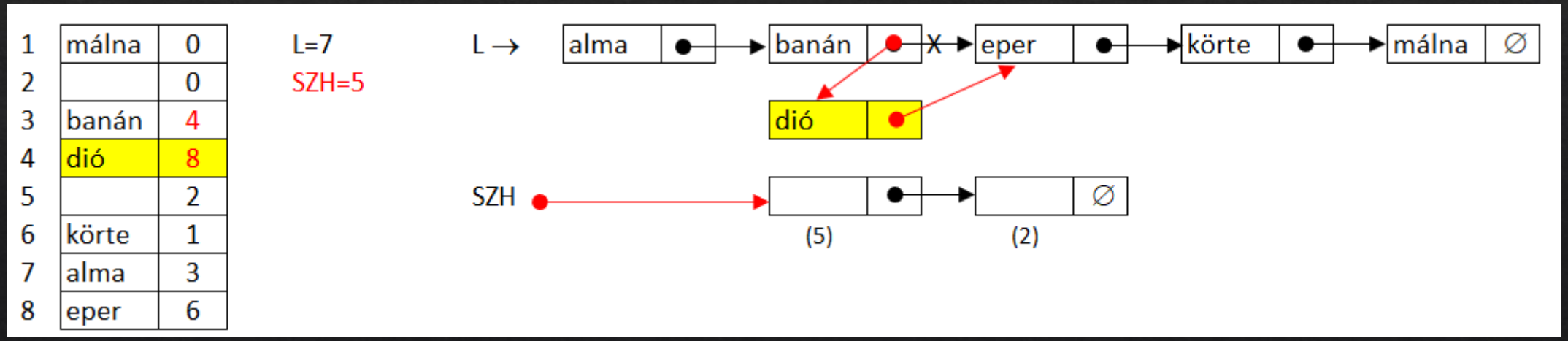


Szabad helyeket nyilvántartó lista

Beszúrás a listába

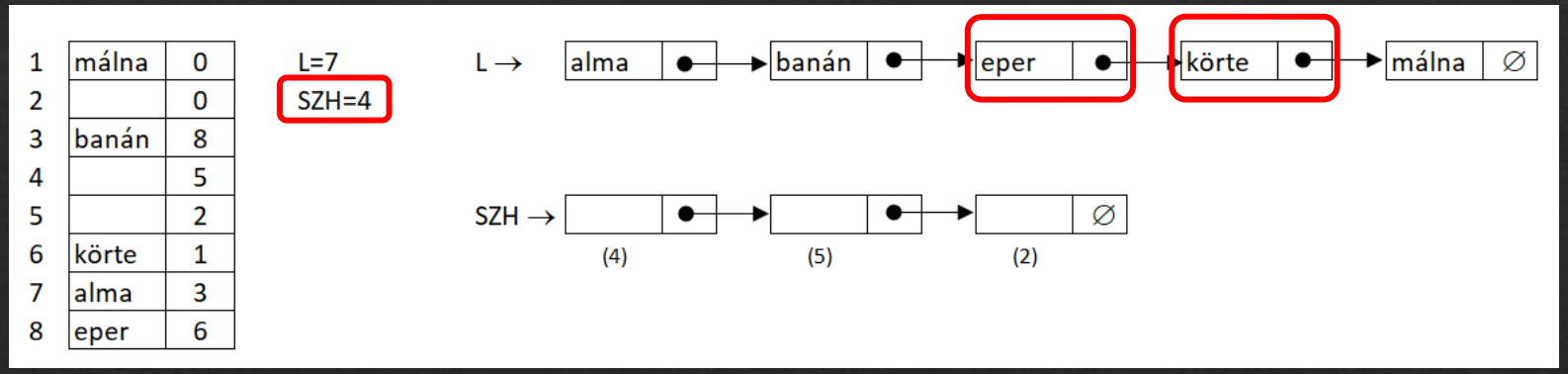
- Névsorba rendezetten szúrjuk be a listába a „dió”-t.

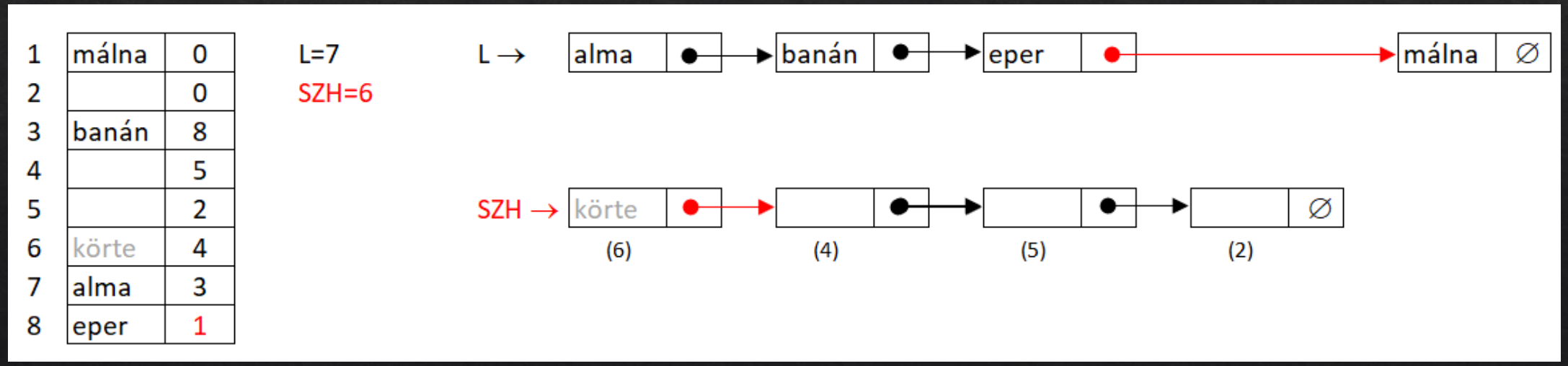




Törlés a listából

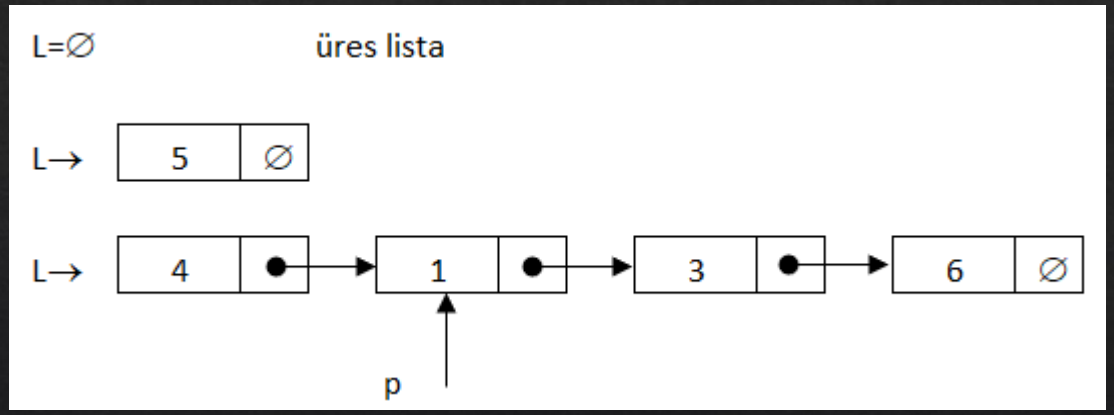
- Töröljük a „körte”-t a listából.





Egyirányú fejelem nélküli (egyszerű) listák: S1L

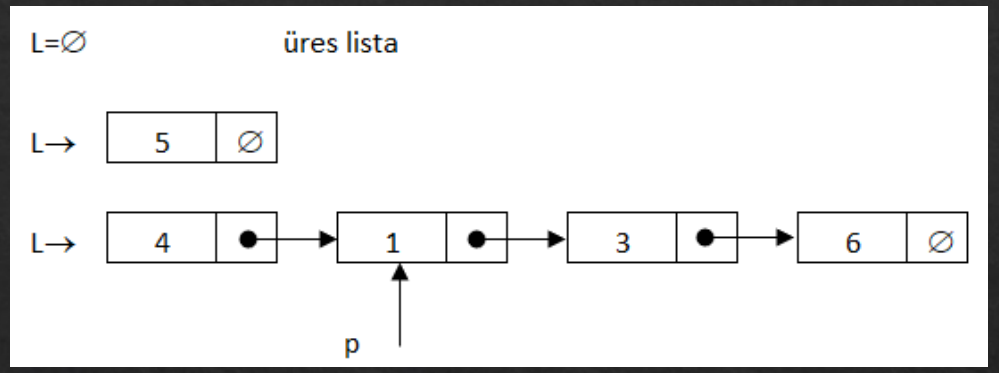
- A listaelemekben az adatokon kívül egy pointer található, amely a lista következő elemének címe (ha van utána elem), vagy nulla (NULL, NIL), ha nincs rákövetkező elem.
- A lista első elemére L (L1,L2) pointer mutat. L értéke nulla, ha üres a lista.
- A lista bejárásához bejáró pointereket (p,q,r,s) használunk.



E1
$+key : \mathcal{T}$ $\ldots$ // satellite data may come here $+next$: **E1***
$+\mathbf{E1}()\ \{\ next := \oslash\ \}$

Lista elemeinek elérése, feldolgozása

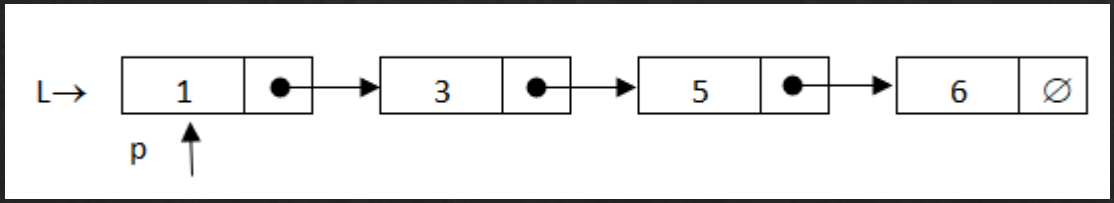
- p – legyen egy lista elemre mutató pointer
  p: E1*
- p key a listaelem kulcsa
  helyes még: (*p).key
- p next a következő elem címe, vagy nulla, ha nincs rákövetkező
  helyes még: (*p).next
- Új listaelem lefoglalása a szabad memóriából: p:= new E1
  p-ben lesz az új listaelem címe, a next pointer értéke nulla lesz (E1 konstruktor)
- Feleslegessé vált listaelem felszabadítása: delete p
  delete után a listaelem bármely részére való hivatkozás tilos!

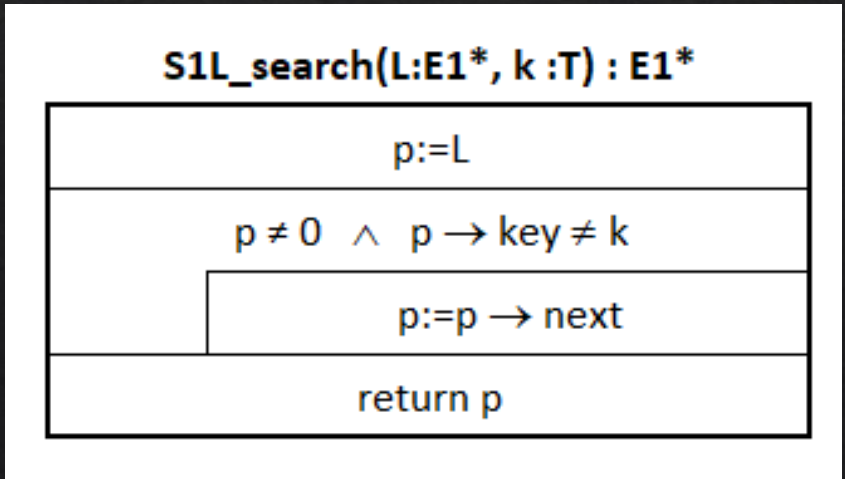


E1
$+key : \mathcal{T}$ $\ldots$ // satellite data may come here $+next$: **E1***
$+\mathbf{E1}()\ \{\ next := \oslash\ \}$

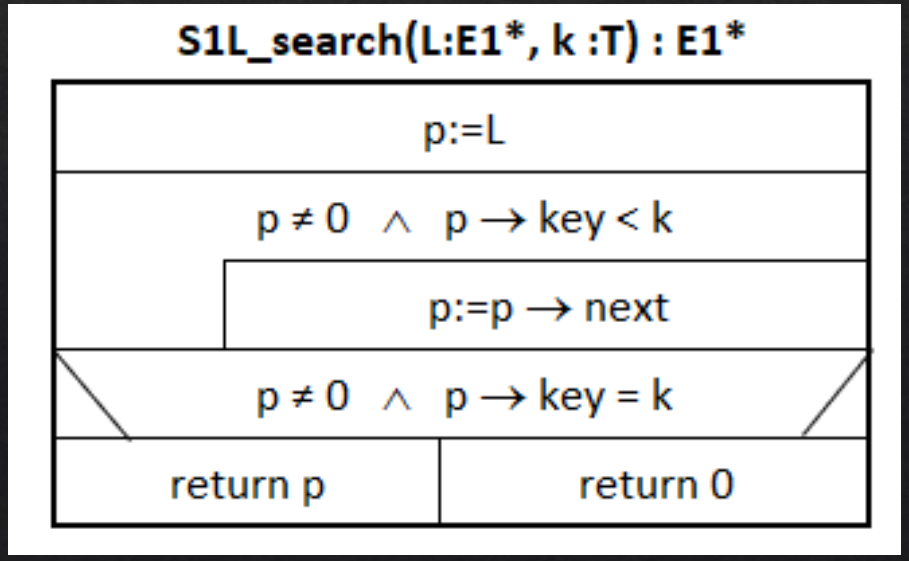
Alap műveletek: keresés

- L egy kulcs szerint szigorúan monoton növekvően rendezett S1L (egyirányú, fejelem nélküli lista). Keressük meg a lista egy adott kulcsú elemét. Ha megtaláltuk, adjuk vissza a listaelem címét, ha nem található meg, nullával térjünk vissza.



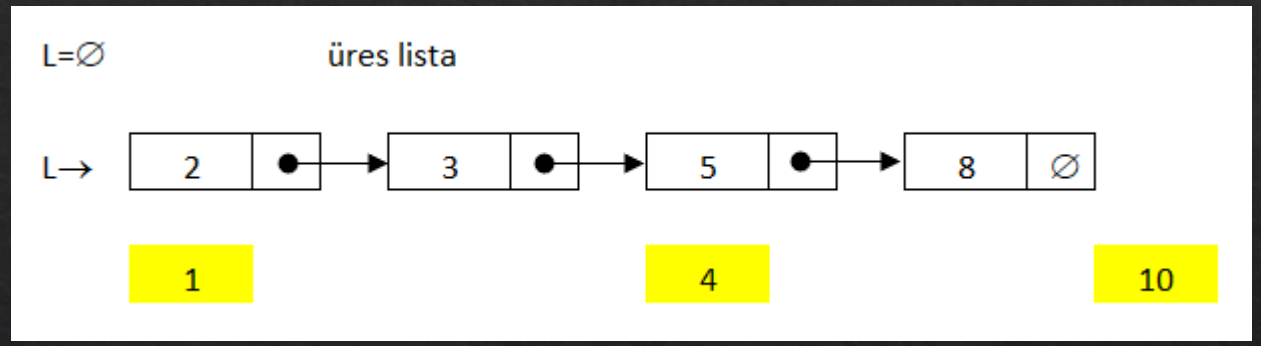


Használjuk ki, hogy rendezett a lista!



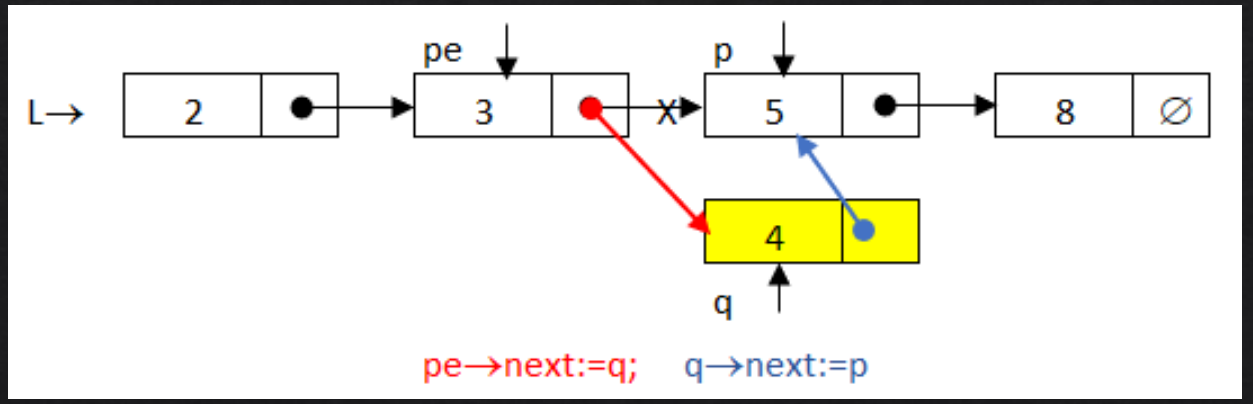
Alap műveletek: beszúrás

- L egy kulcs szerint szigorúan monoton növekvően rendezett S1L (egyirányú, fejelem nélküli lista). Szúrjunk be a listába egy k kulcsot a rendezettség szerinti helyére.
- Ha lenne már ilyen a kulcs a listában, ne történjen semmi.
- Négy esetre kell gondolni:
  - A lista belsejébe illik az elem,
  - A lista végére illik az elem,
  - A lista elejére illik az elem,
  - Üres a lista, most szúrjuk be az első elemet.



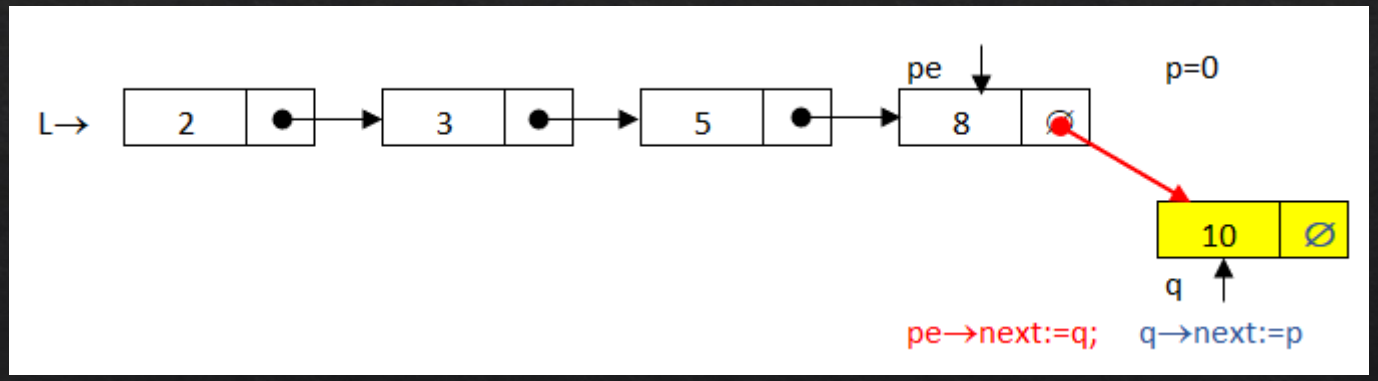
Beszúrás

- Két bejáró pointert fogunk használni: pe, p – a kettő közé kell befűzni az újelemet.
- Lista belsejébe történő befűzés:
  - p pointerrel megkeressük az első olyan elemet, amelyik nem kisebb, mint a beszúrandó kulcs, közben pe párhuzamosan halad p-vel, mindig a p előtti elem címét tartalmazza.
  - Ha a p megáll egy listaelemen, akkor a p című elem kulcsa nagyobbegyenlő, mint a beszúrandó kulcs.
  - Ellenőrizzük, hogy p key nem a beszúrandó kulcs. Ha nem, akkor pe és p közé befűzzük az új elemet. (Előtte lefoglalunk az új elem számára egy új listaelemet, és kitöltjük a kulccsal, és az egyéb szükséges adatokkal.)



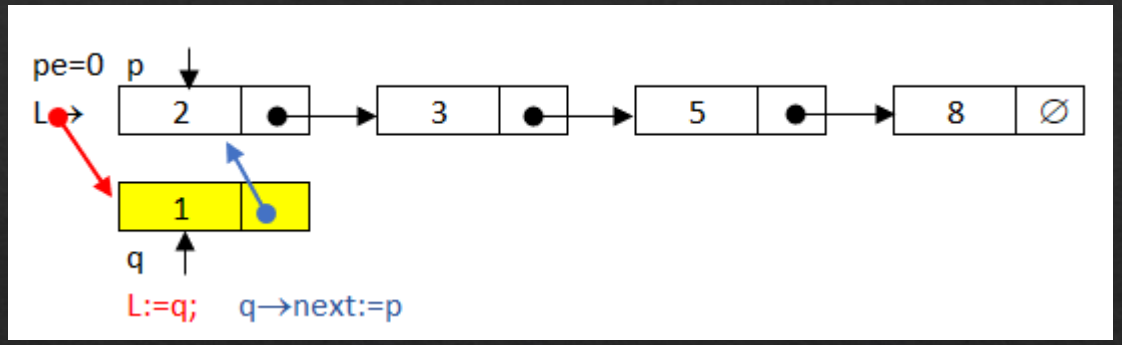
Beszúrás

- Két bejáró pointert fogunk használni: pe, p – a kettő közé kell befűzni az újelemet.
- Lista végére történő befűzés:
  - p pointer „leszalad” a listáról, mert a listaelemek mindegyike kisebb, mint a beszúrandó kulcs.
  - Mivel pe együtt haladt p-vel, a lista utolsó elemének címe pe-ben van, így segítségével az új listaelemet be tudjuk fűzni a lista végére.

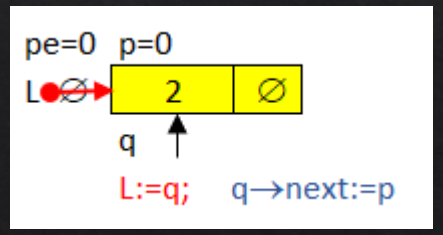


Beszúrás

- Ha a lista elejére kell befűzni az elemet, akkor L fog megváltozni (pe ilyenkor nulla):



- Üres lista esete: szintén L fog változni (pe, és p is nulla):



Beszúrás S1L-be

S1L_insert(&L: E1*, k: T)

	pe:=0; p:=L	
	p ≠ 0 ∧ p → key < k	
	pe:=p	
	p:=p → next	
	p ≠ 0 ∧ p → key = k	
skip	q := new E1	
skip	q → key := k	
skip	q → next:= p	
skip	pe = 0	
skip	L:=q	pe → next:= q

Fontos, hogy L referencia szerinti paraméter! Megváltozhat!

Bejáró pointerek indítása.

Első olyan elem megkeresése, amelyik nem kisebb.

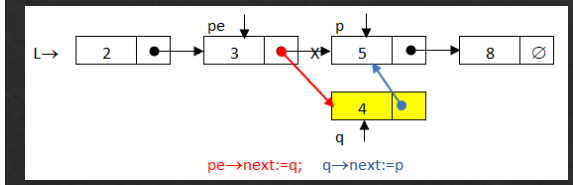
Ha van már k kulcsú elem, készen vagyunk…

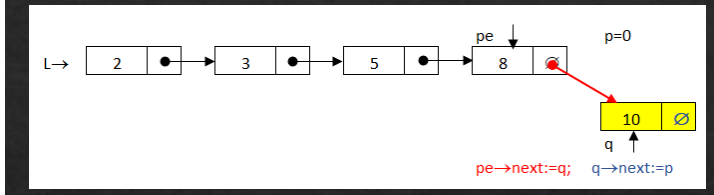
Új listaelem lefoglalása.

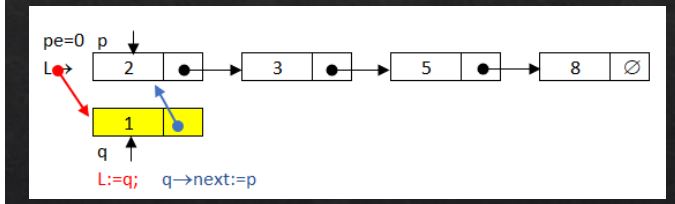
Kulcs beleírása az új elembe

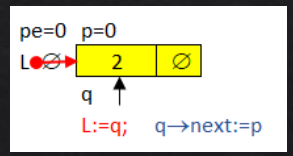
A következő elem p.

Attól függően, hogy volt-e előző elem, L vagy pe című elem next pointere változik.



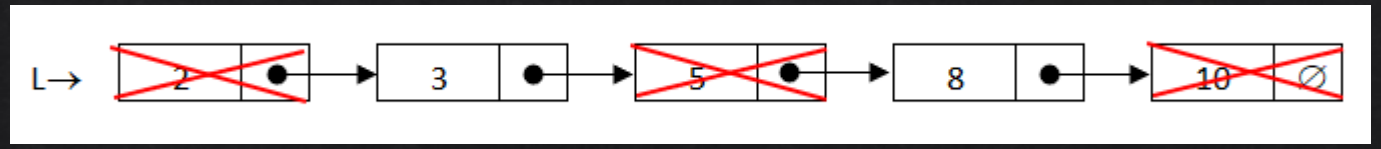




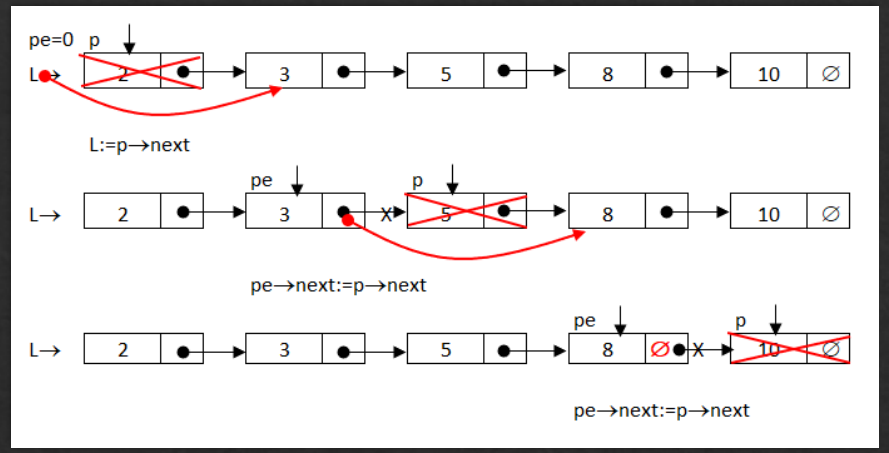


Alap műveletek: törlés

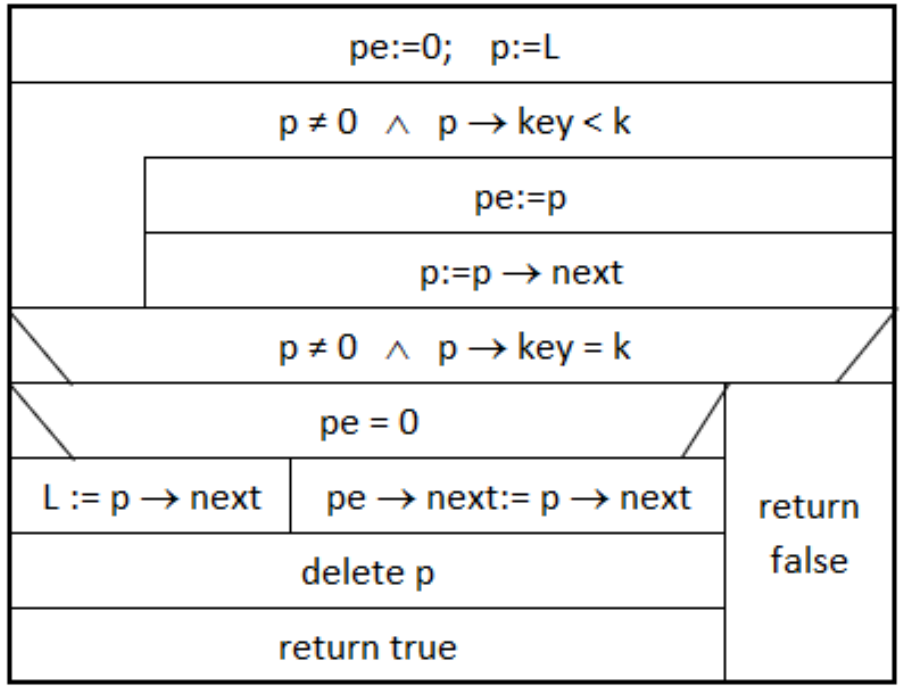
- L egy kulcs szerint szigorúan monoton növekvően rendezett S1L (egyirányú, fejelem nélküli lista). Keressük meg és töröljük a k kulcsú elemet, ha van.
- Ha a törlés sikeres, igazzal, ha sikertelen hamissal térjünk vissza.
- A kifűzéshez ismerni kell a listában a k kulcsú elemet megelőző elem címét.
- Ha van k kulcsú elem, akkor a következő esetek fordulhatnak elő:
  - Lista első elemét töröljük.
  - Lista belsejéből törlünk (előtte, és utána is van elem),
  - Lista legutolsó elemét töröljük (előtte van elem),



Törlés S1L-ből



S1L_delete(&L:E1*, k: T):B



Fontos, hogy L referencia szerinti paraméter! Megváltozhat!

Bejáró pointerek indítása.

Első olyan elem megkeresése, amelyik nem kisebb.

Ha megtaláltuk a keresett elemet,

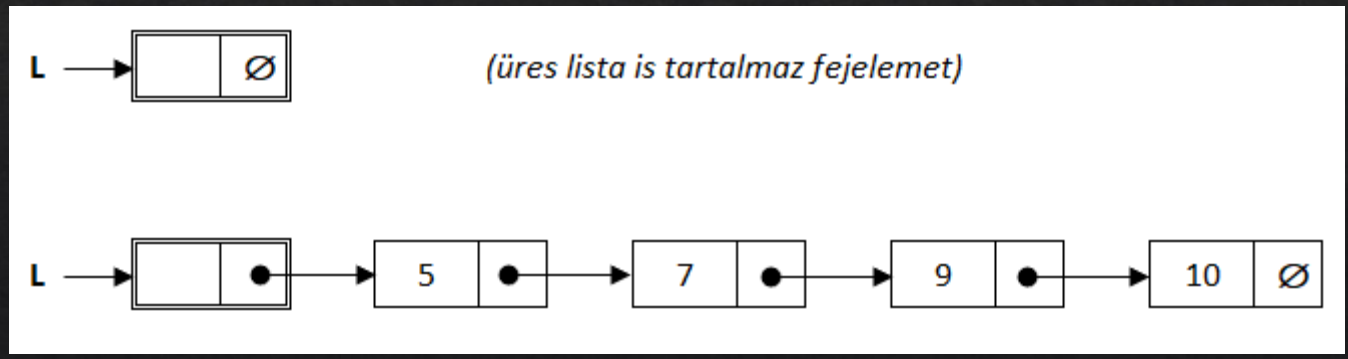
kiláncoljuk, attól függően, hogy a lista elején volt-e,

vagy valahol a lista belsejében.

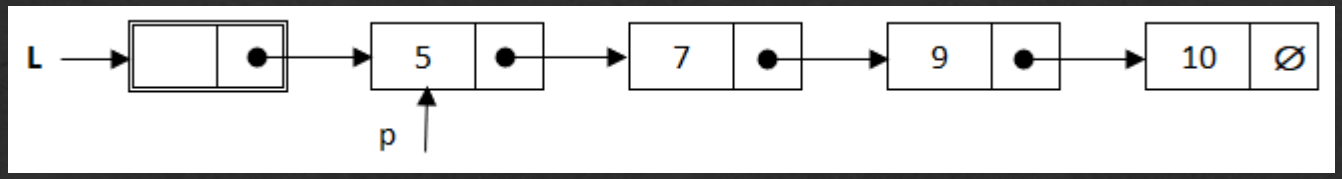
Kiláncolás után felszabadítjuk az elem helyét.

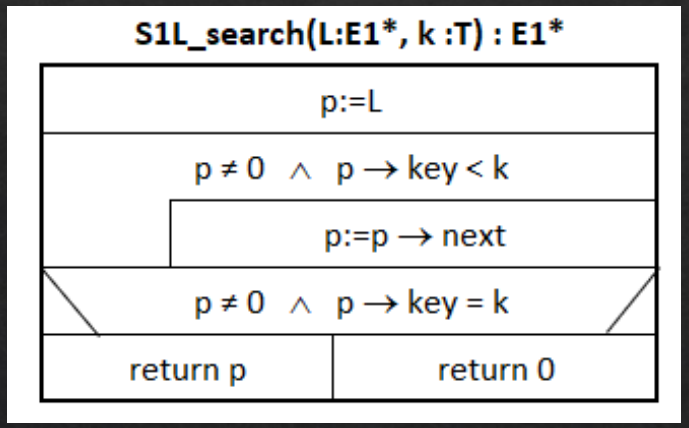
Fejelemes egyirányú listák (H1L)

- Az S1L listák kezelésénél az algoritmusnak külön kell gondoskodnia arról ha a lista elején történik a beszúrás/törlés.
- Beszúrásnál az üres lista külön estet jelentett.
- Ennek kiküszöbölésére egy fejlemet helyezhetünk a lista elejére.
- Így a listának mindig van egy eleme, a fejelem, soha nem lesz teljesen üres!
- A fejelemben nem tárolnak adatot.

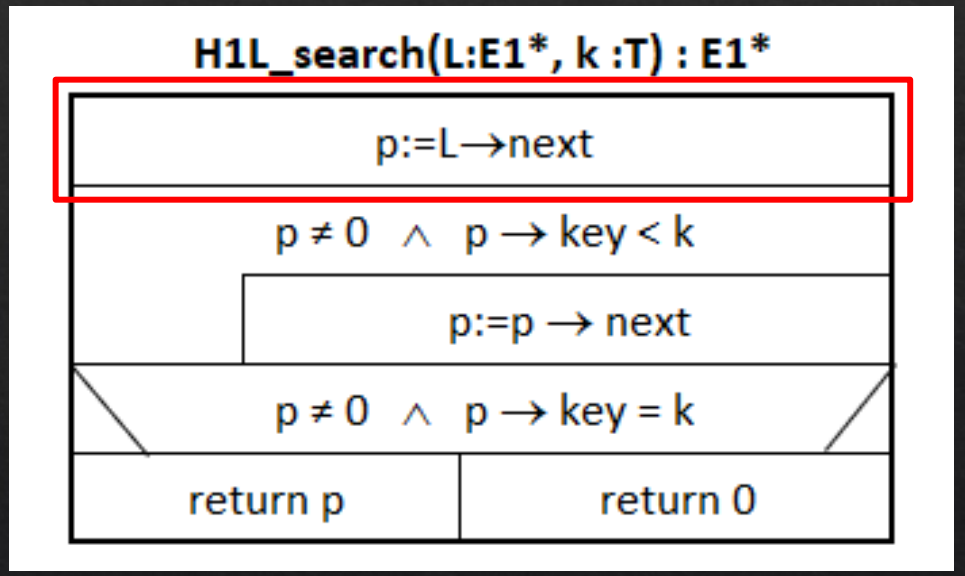


Keresés – H1L

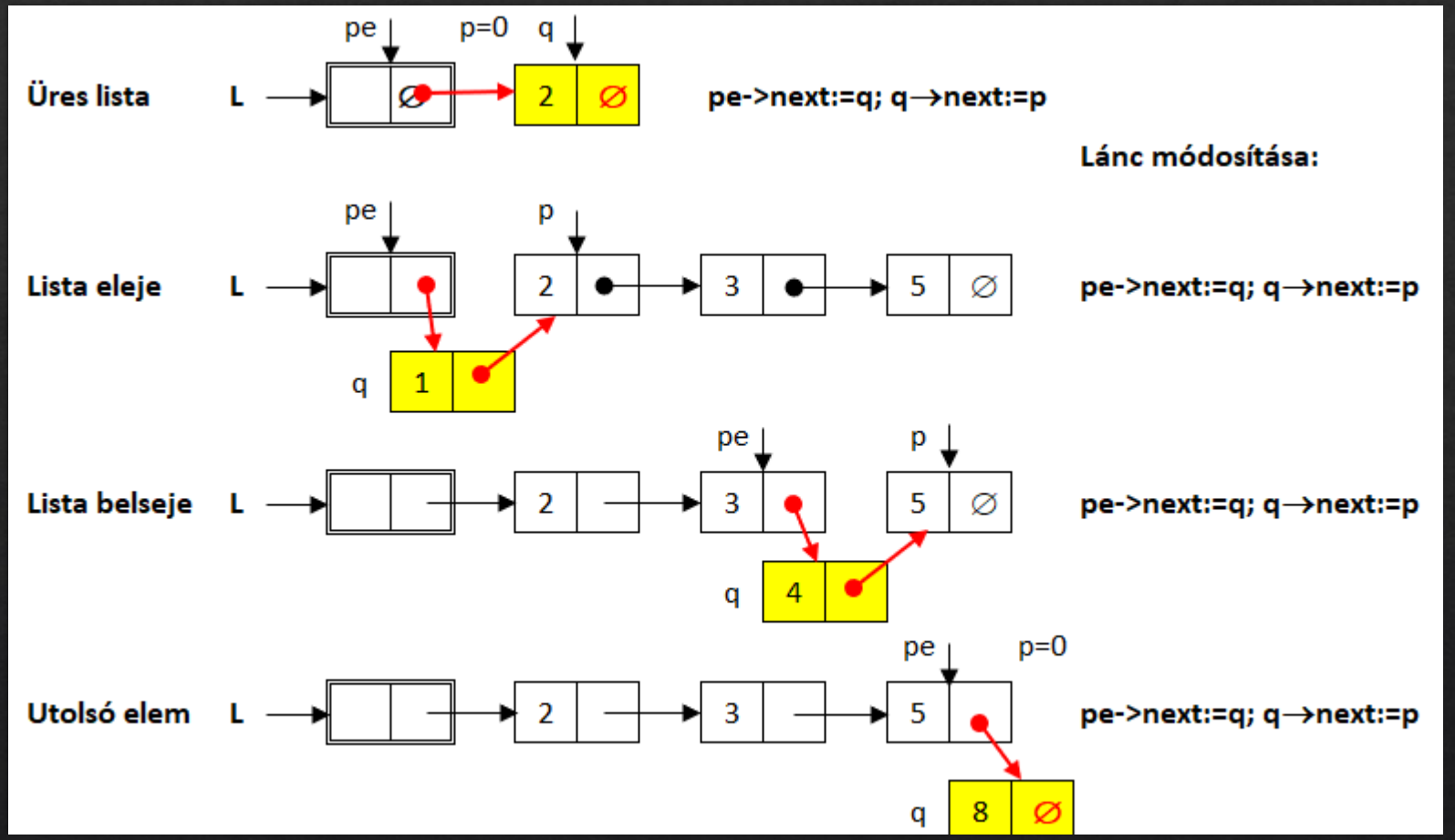




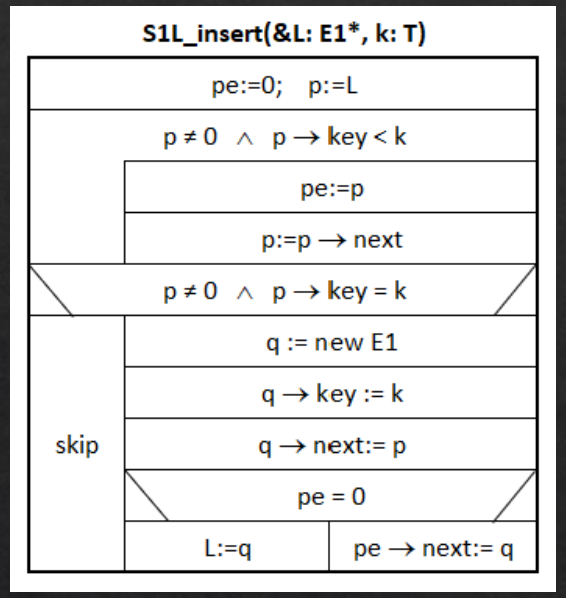
- Változás: az első elem címe: L next, onnan indul a keresés!

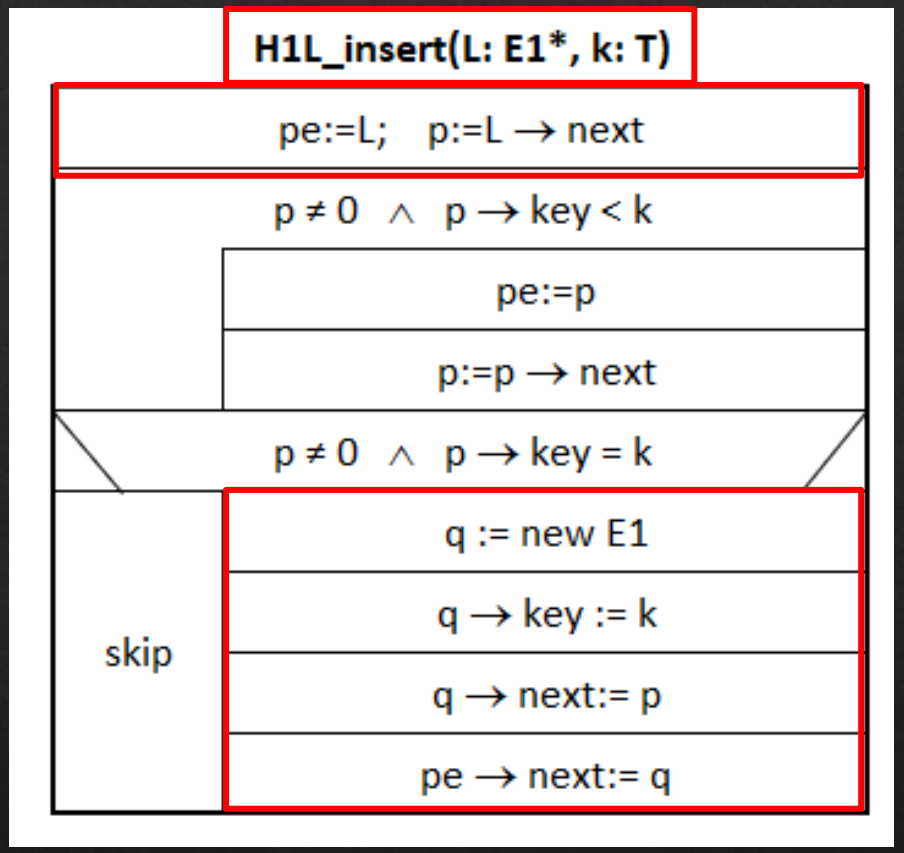


Beszúrás H1L



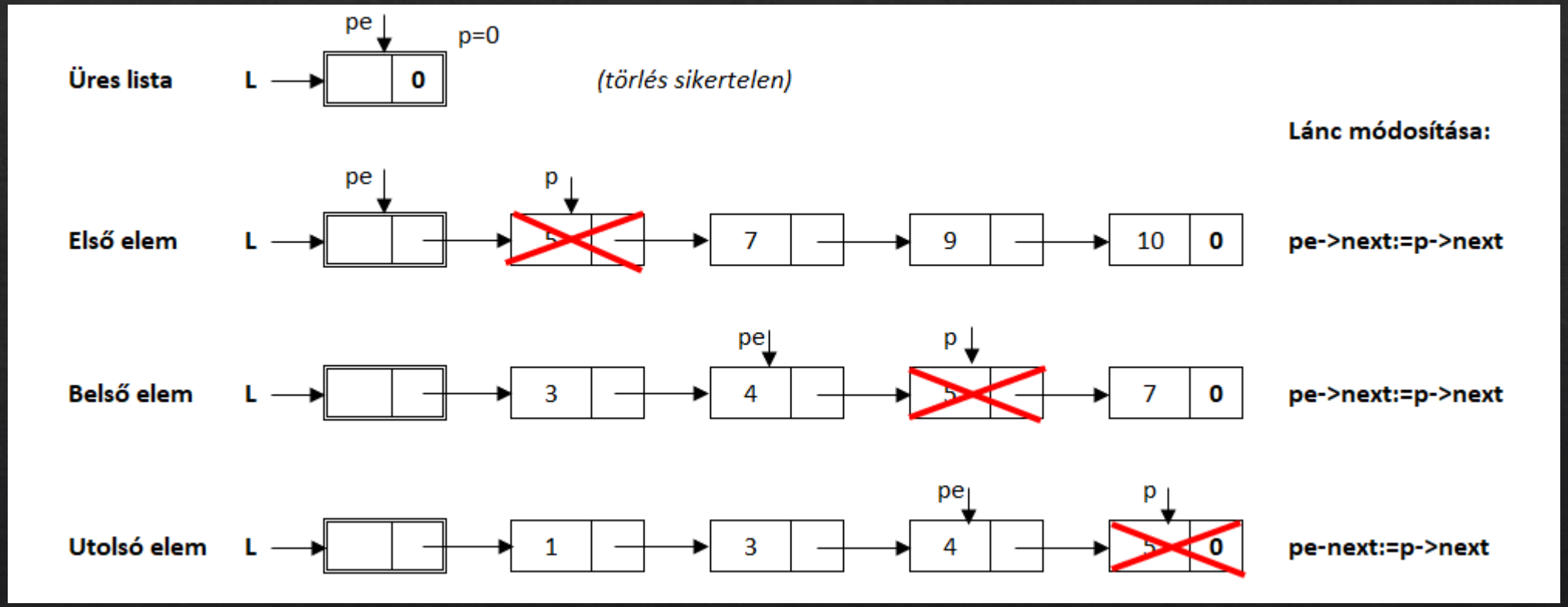
Beszúrás H1L



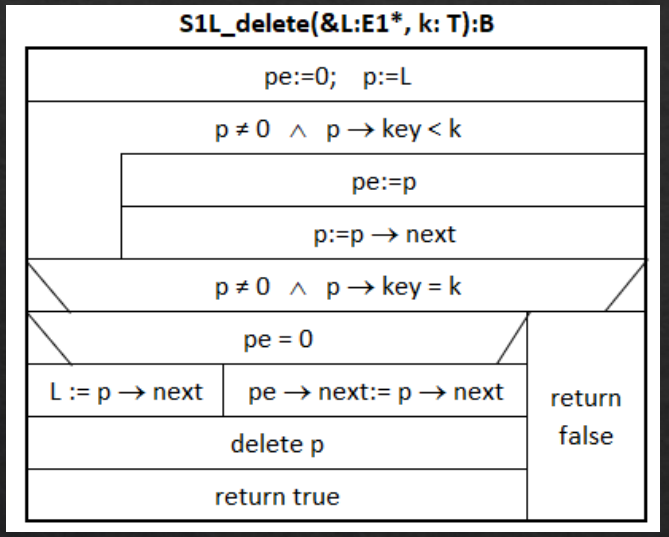


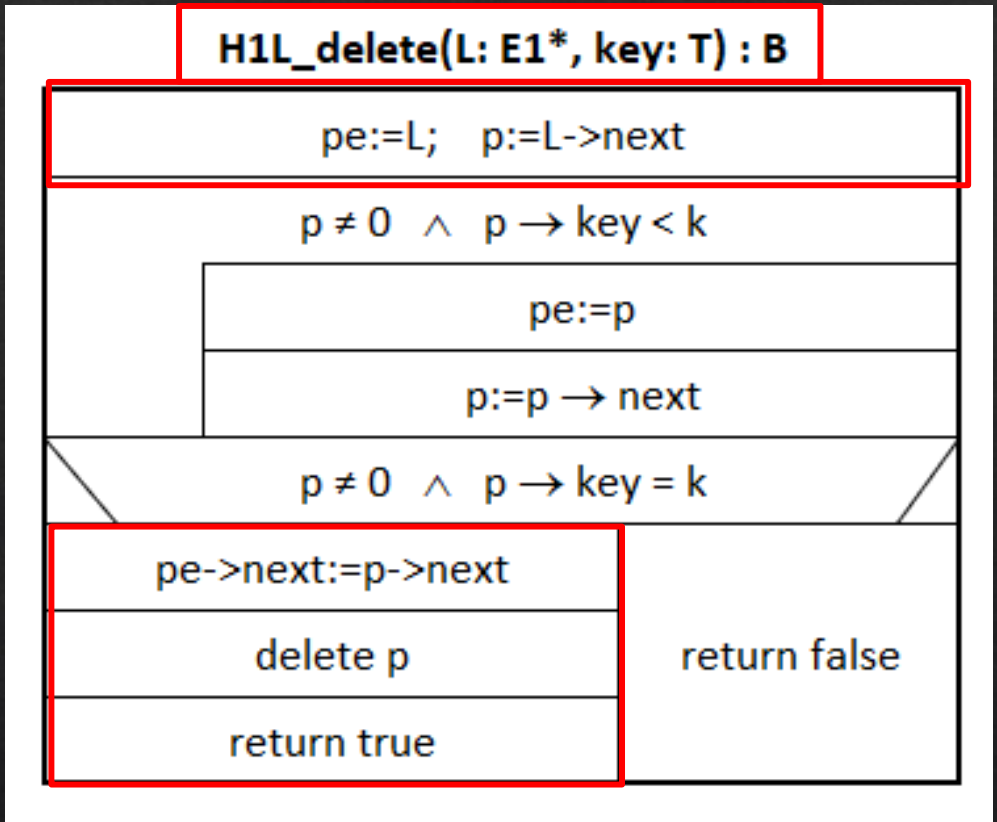
- Változások:
- L nem referencia paraméter
- Indulás: pe a fejelemről, p az első elemről (nulla is lehet).
- Befűzés esetén nem kell elágazás, mert pe soha nem nulla (a befűzendő elem előtt mindig van a listának eleme)

Törlés H1L


Üres lista
L
pe
p=0
0
(törlés sikertelen)
Lánc módosítása:
Első elem
L
pe
p
5
7
9
10
0
pe->next:=p->next
Belső elem
L
pe
p
3
4
5
7
0
pe->next:=p->next
Utolsó elem
L
pe
p
1
3
4
5
0
pe-next:=p->next

Törlés H1L





- Változások:
- L nem referencia paraméter
- Indulás: pe a fejelemről, p az első elemről (nulla is lehet).
- Törlés esetén mindig van előző eleme a listának

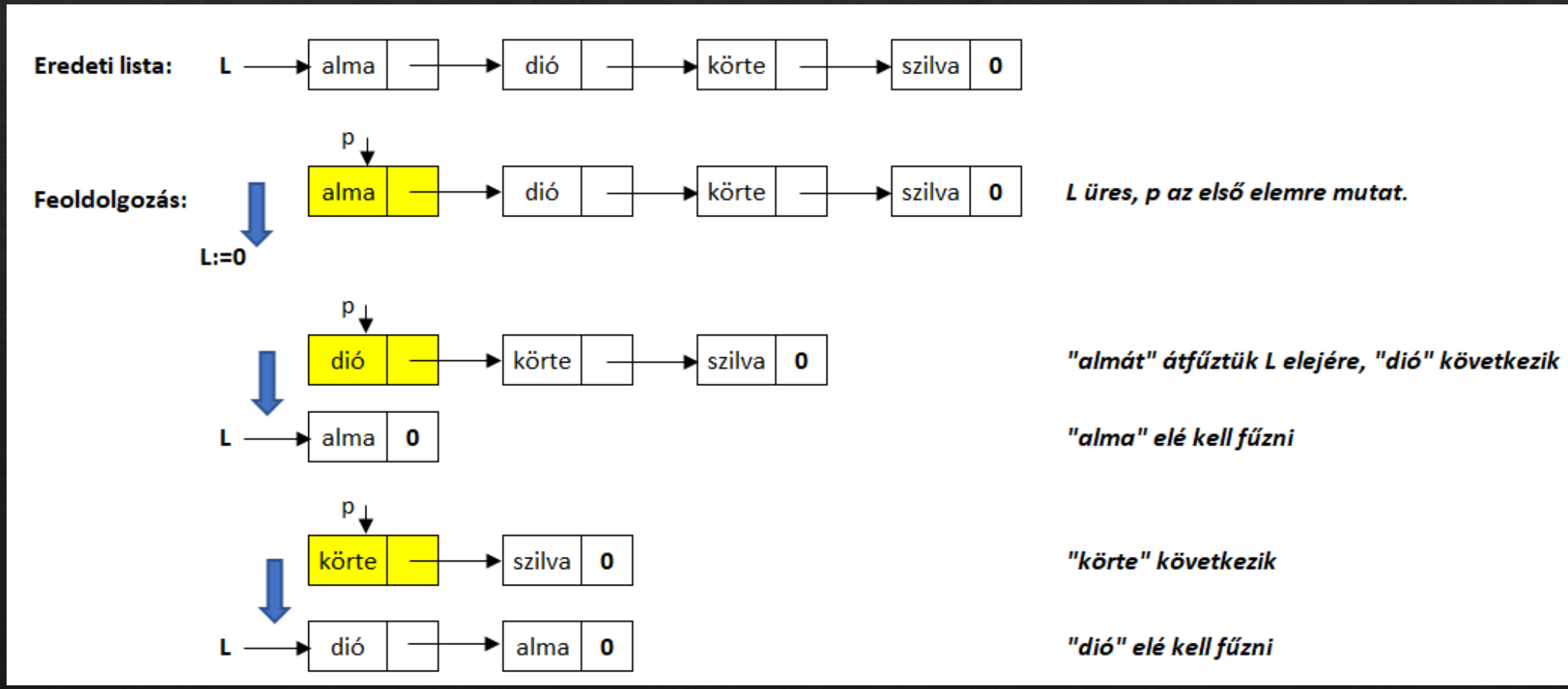
S1L lista megfordítása

- L legyen egy S1L lista. Fordítsuk meg az elemek sorrendjét! (Fűzzük össze a lista elemeit fordított sorrendben!)
- Például, ha L eredtileg a következő elemeket tartalmazta:
  L →alma→dió→körte→szilva

  akkor eredményül a következő listát kell kapnunk:
  L→szilva→körte→dió→alma

Ötlet:

- Az eredeti L listát elemenként „lebontjuk”, egy p pointer fog az első elemre mutatni.
- L listát üresre állítjuk. //L:=0
- Ha a mindenkori első elemet mindig az L lista elejére fűzzük, az elemek sorrendje meg fog fordulni!



Megoldás

Megfordít(&L:E1*)

```
p := L
L := 0
p ≠ 0
    q := p
    p := p->next
    q->next := L
    L := q
```

Fontos, hogy L címszerinti paraméter

Indulás: p az eredeti lista első elemére mutat

L listát üresre állítjuk

Amíg az eredeti lista el nem fogy:

q lesz az az elem, amelyet most átfűzünk L elejére,

mivel q beláncolásakor elveszítenénk a q utáni elemet, p-vel tovább kell lépni!

q beláncolásának lépései: q után L eddigi első eleme következzen,

L lista új első eleme legyen a q.

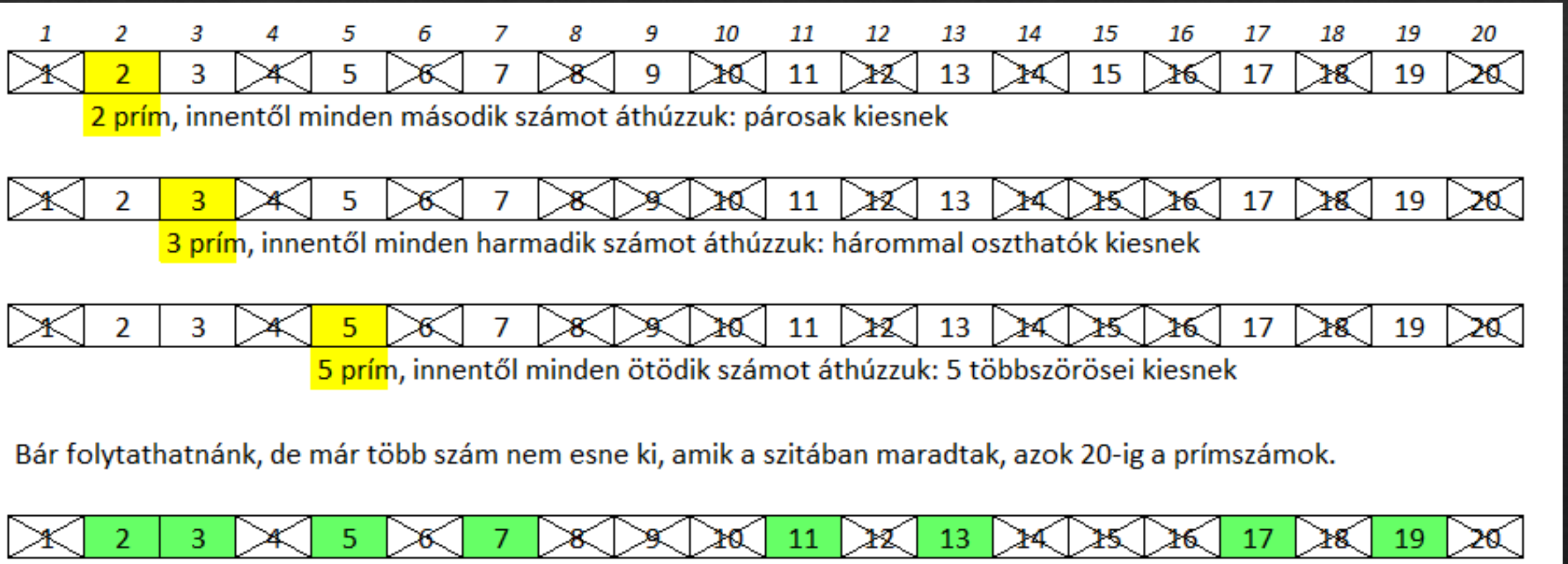
Vajon helyesen működik az algoritmus, ha L lista üres? Válasz: igen, gondolja meg miért?

Prím szita S1L listával

- Az Eratoszthenészi szita algoritmus mintájára készítsünk olyan S1L listát, mely 2-től n-ig a prímszámokat tartalmazza.
- Mi is a módszer?
  - Eratoszthenész Pentatlosz talált egy nagyon hatékony módszert egy nem túl nagy sokaságból a prímszámok kiválogatására.
  - Írjuk fel a listára a természetes számokat 1-től, addig ameddig a prímeket keressük. 1 nem prím, húzzuk ki. 2 az első prím. 2-től indulva minden második elemet húzzunk ki.
  - A megmaradt elemek közül a következő nem kihúzott a 3, megint prím, húzzuk ki innen indulva minden harmadik számot.
  - A következő ami nincs kihúzva: 5 prím, 5-től minden ötödik elemet húzzunk ki, a következő ami nincs kihúzva: 7 prím…
  - Folytassuk így amíg a felsorolt elemeken végig nem érünk. A szitában fennmaradt számok prímek.

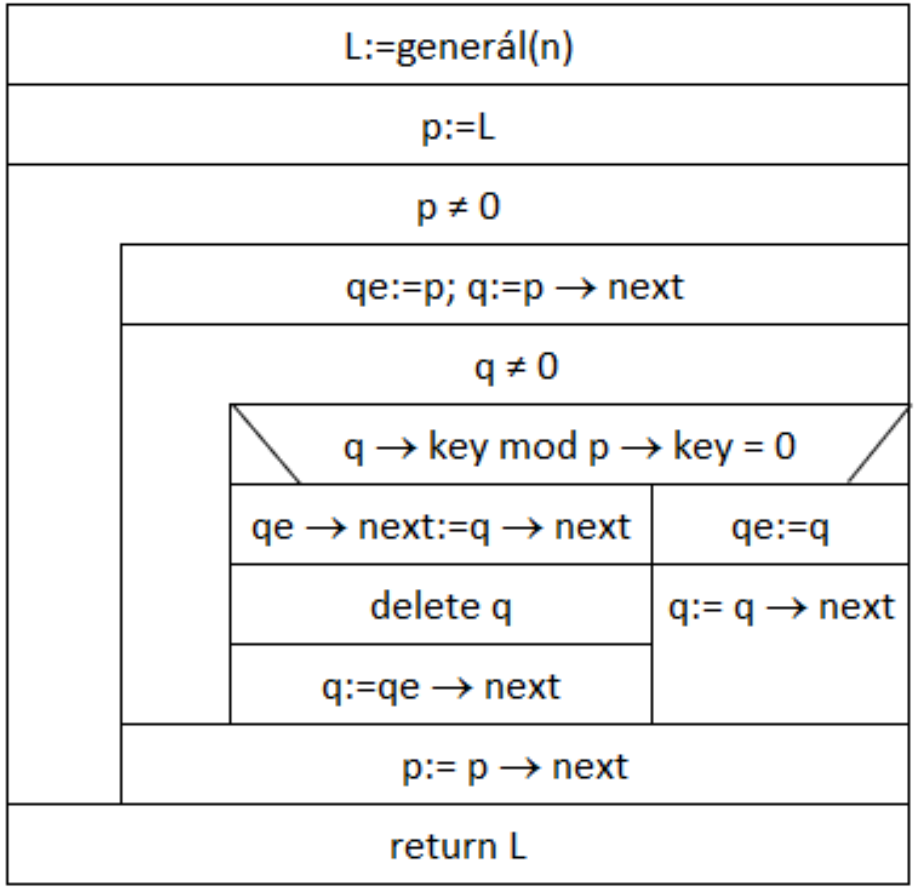
A módszer szemléltetése

Prímszámok előállítása Eratoszthenészi szitával 20-ig



Megoldás

Primszita(n:N) : E1*



Generálunk egy egyszerű listát 2..n természetes számokból.

p mindig a következő, még a szitában lévő prímszámra mutat (elsőként a 2-re). Amíg ez nem 0 …

q-val a p utáni elemeket fogjuk bejárni, és kifűzzük p → key többszöröseit

Ha p →key osztója q →key-nek, a q című elemet töröljük a láncból. Ehhez szükségünk van q elem előzőjére, ez lesz a qe pointerben.

ha p című elem többszöröseit töröltük, a következő elem megint prím lesz.

Megoldás folytatása, az induló listát előállító algoritmus

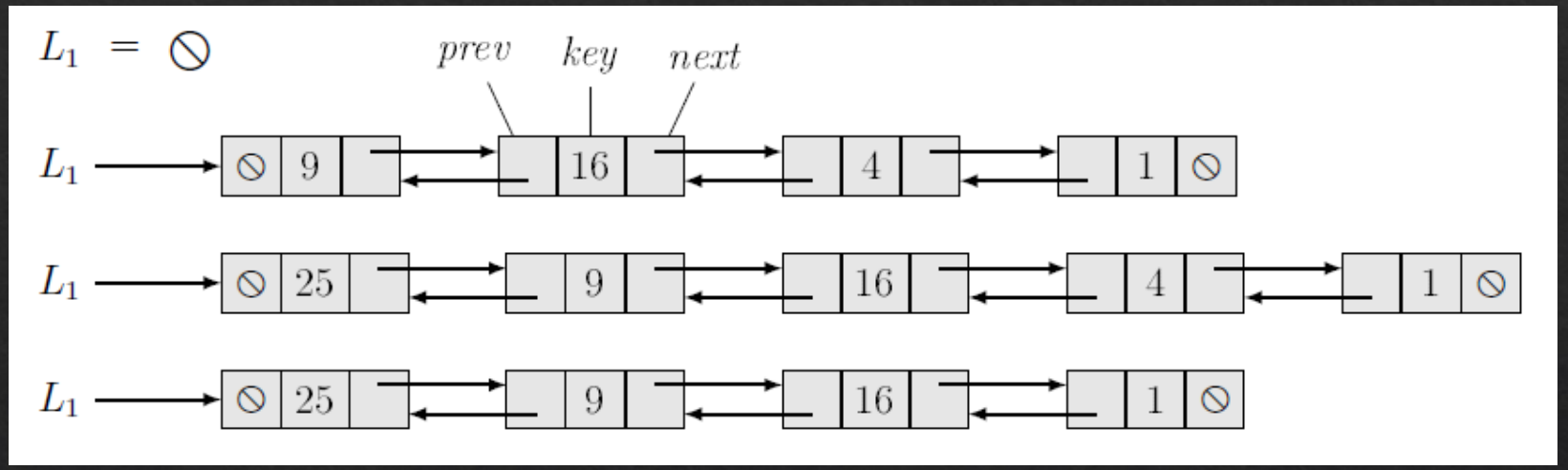
generál(n:N) : E1*

```
L:= 0
i = n downto 2
    p:= new E1
    p → key:= i
    p → next:=L
    L:=p
return L
```

Egyszerű lista feltöltése 2..n természetes számokkal. Trükk: a ciklust csökkenőleg futtatjuk, így fordított sorrendben generáljuk a számokat, tehát mindig a lista elejére kell befűzni az új elemet.

Kétirányú listák

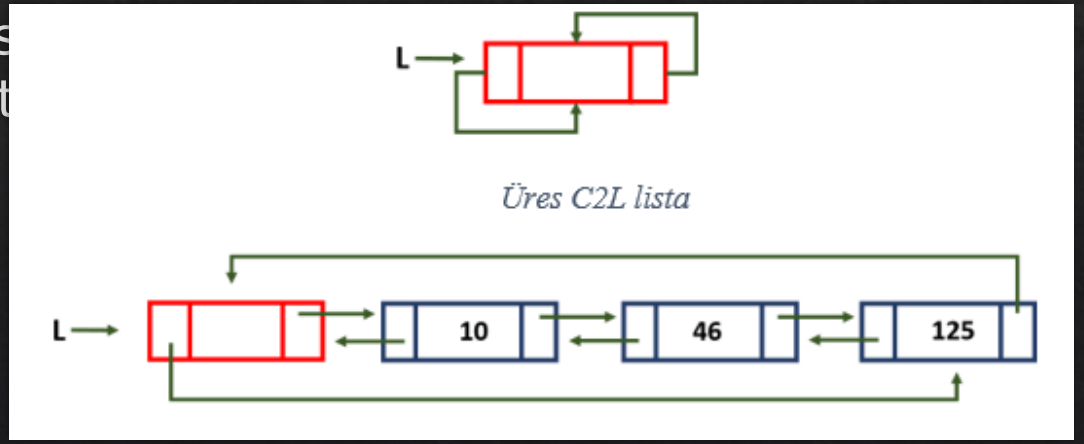
- Előnye: előre és visszafelé is haladhatunk a pointerek mentén.
- Egyszerű kétirányú listák (ábra a jegyzetből) S2L:



- Lista elejének, belsejének és végének kezelése különbözik, például befűzés esetén:
  - Egy belső elemnek bal és jobb szomszédja is van, a befűzés négy pointert érint,
  - Az első elemnek nincs bal szomszédja, az utolsónak nincs jobb szomszédja
- Üres listába történő befűzés is külön eset: a befűzött elemnek sem bal, sem jobb szomszédja sincs.

C2L lista, E2 elemtípus

- C2L alatt a fejelemes kétirányú ciklikus listát értjük. A C2L elemei két pointert tartalmaznak, az egyik (prev) az aktuális elemet megelőző, a másik (next) a következő elemre mutat.
- Fejelemet mindig tartalmaz a lista, így a műveletek átláthatóbbak, egyszerűbbek lesznek.
- A C2L ciklikus, azaz az utolsó elemének next pointere a fejelemre, a fejelem prev pointere pedig az utolsó elemre mutat. Így a lista elemei kétféle sorrendben is feldolgozhatók, például az ábrán látható lista kulcsai növekvő és csökkenő sorrendben is elérhetők. Bejáráskor az els el, azt, hogy körbe értünk a p=L vizsgálat
- Az üres C2L lista egy fejelemből áll, melynek mindkét pointere magára a fejelemre mutat.
- Az ilyen listák elemeinek ábrázolásához az E2 nevű típust használjuk.



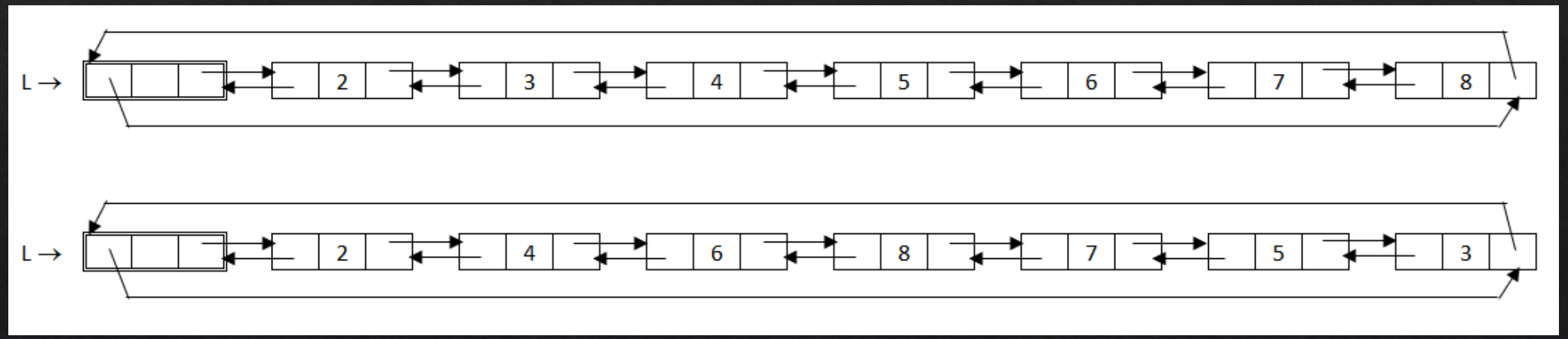
A jegyzetben definiált E2 elemtípus UML diagramja

E2
$+prev, next$: **E2*** // refer to the previous and next neighbour or be **this** $+key : \mathcal{T}$
+ E2() { $prev := next :=$ **this** }

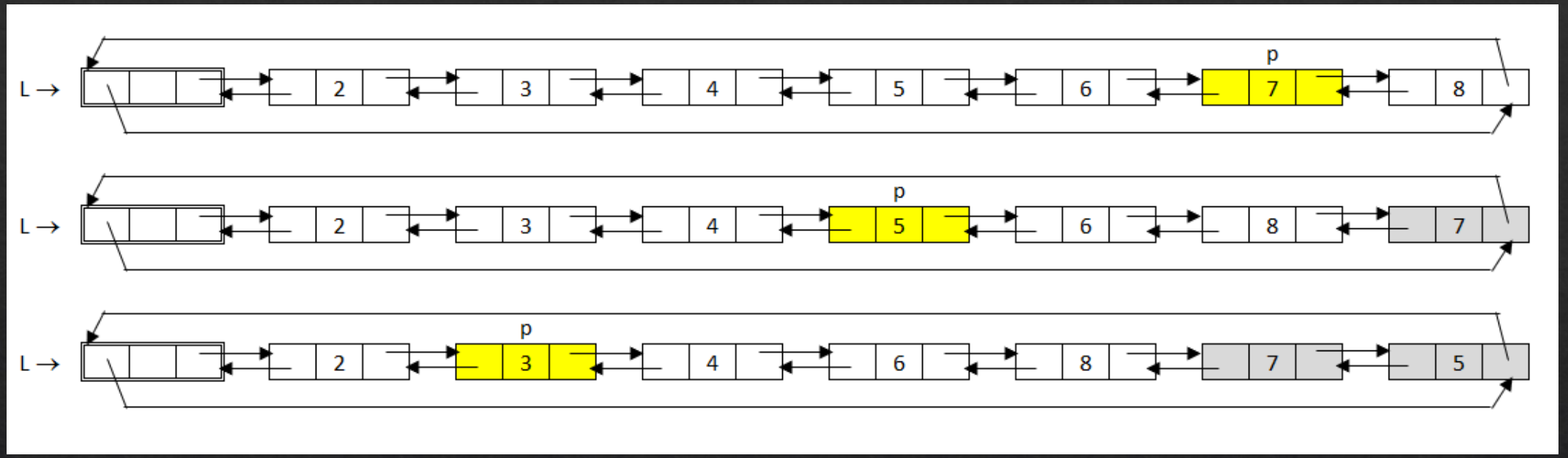
- Vegyük észre, hogy az elem konstruktora a pointereket úgy állítja be, hogy azok magára az elemre mutassanak! Ezt kihasználva egy új fejelemes C2L lista fejelemének létrehozása: L:= new E2 utasítással történhet!

C2L bejárás, módosítás egy feladaton keresztül

- Oldjuk meg a következő feladatot:
- Adott egy természetes számokat tartalmazó C2L lista. A lista szigorúan monoton növekvően rendezett.
- A lista egyszeri bejárásával rendezzük át az elemeit úgy, hogy a lista elején legyenek a páros számok növekvően, a végén pedig a páratlanok csökkenően.



Ötlet:



- Fordított irányban járjuk be a listát. A bejáró pointer p lesz.
- Ha a p elem kulcsa páratlan, az elemet kifűzzük, és átláncoljuk a lista végére (a fejelem elé).
- A kiláncoláshoz egy q segéd pointert fogunk használni, p-vel pedig a kiláncolás előtt tovább lépünk.

Megoldás

C2L_átrendezés(L:E2*)

p:= L→prev

p ≠ L

p→key mod 2 = 1

q:=p; p:=p→prev	
q→prev→next:=q→next	
q→next→prev:=q→prev	
q→next:= L	p:=p→prev
q→prev:= L→prev	
q→prev→next:= q	
q→next→prev:= q	

A kékkel és zölddel jelölt lépések illusztrációját a következő dián nézheti meg.

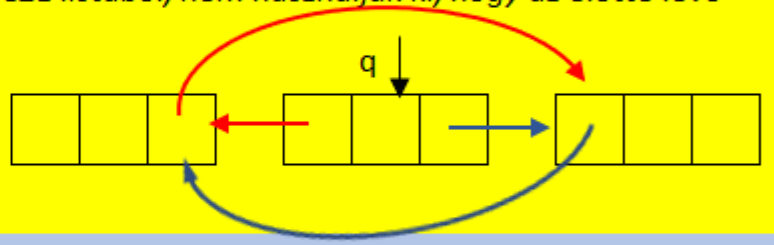
P-vel a L utolsó elemére állunk.

Amíg nem a fejelemre mutat p,

Ha p páros, akkor megyünk tovább visszafelé, a prev pointer segítségével,

ha p páratlan, q-ban megjegyezzük a címét, p-vel azonnal tovább lépünk

A q című elem kifűzése C2L listából, nem használjuk ki, hogy az előtte lévő elemet ismerjük:



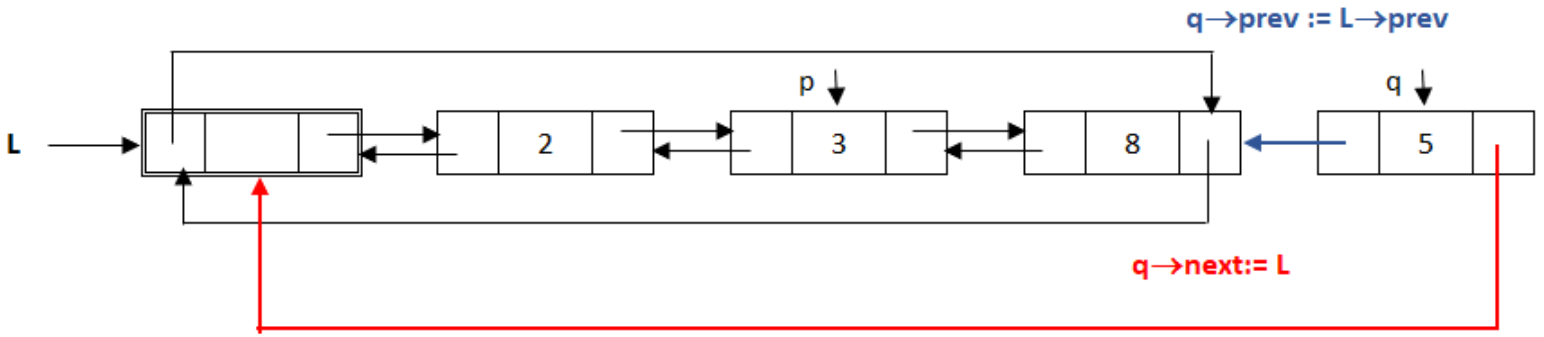
A q című elem befűzése a fejelem elé, elsőként q pointereit állítjuk be: q után a fejelem jön, az utolsó elem (L →prev) lesz a q előzője.

Majd q két szomszédjának pointereit állítjuk be: a q előtti elem rákövetkezője q, továbbá a q utáni elemnek az előzője szintén q. FONTOS, hogy előtte q →prev és q →next már be legyen állítva!

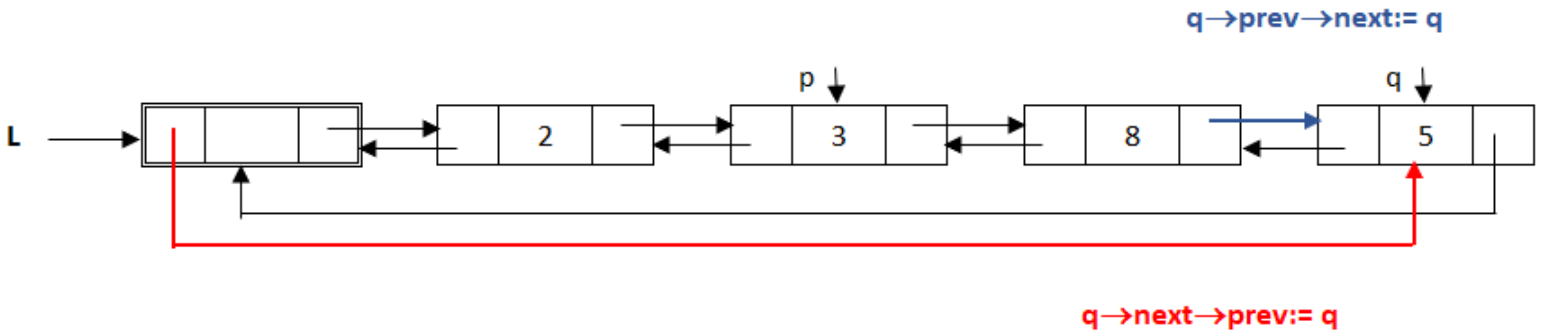
A befűző lépés szemléltetése

q című elem befűzése a lista végére (a fejelem elé)

Elsőként q két pointerét rállítjuk a két szomszédjára.



Ezután már könnyű a két szomszéd megfelelő pointereit q-ra állítani.



Mi történik, ha a lista csupa páratlan elemből áll?

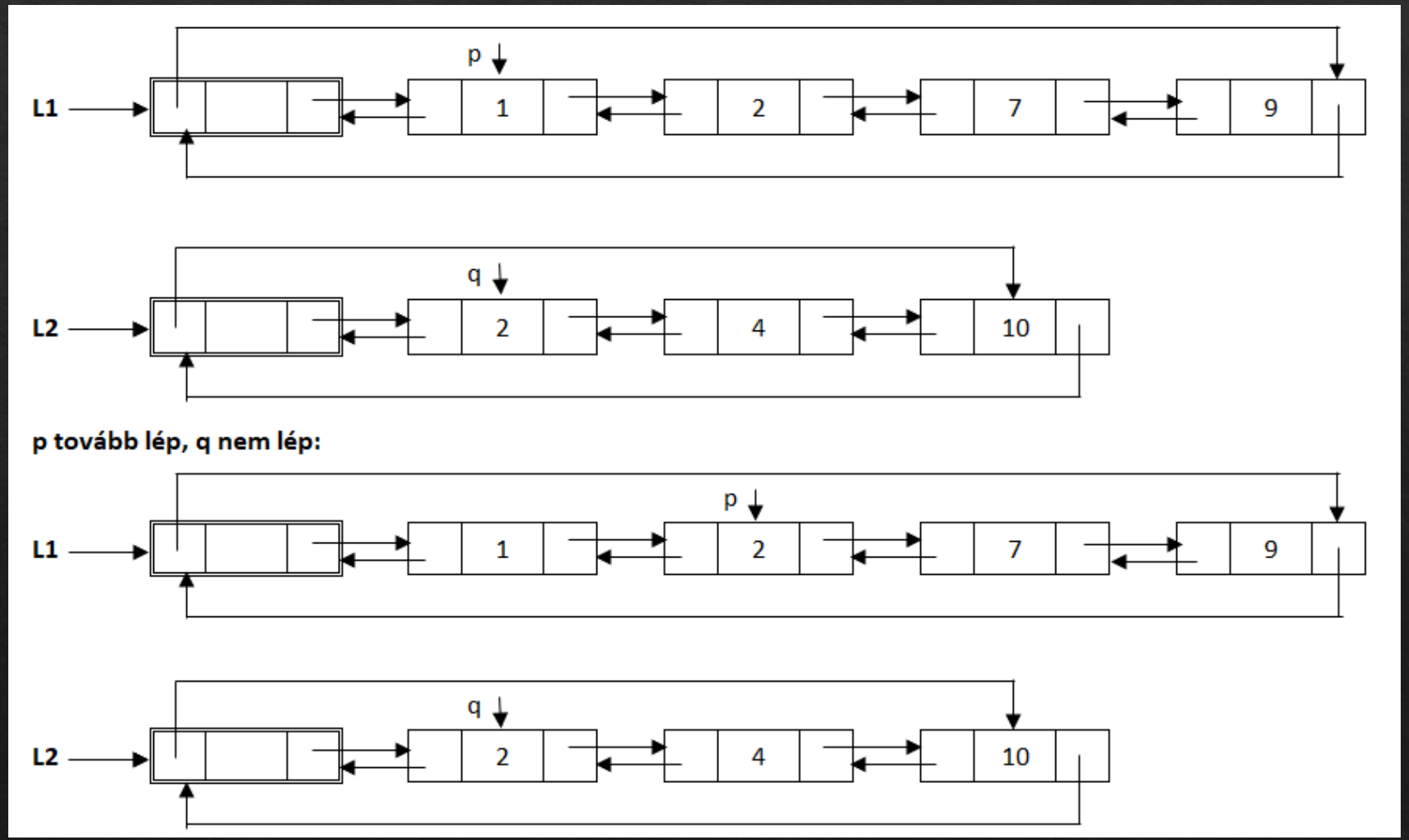
A lista sorrendje megfordul.

Két rendezett C2L lista összefésülése

- Igen gyakoriak azok a feladatok, melyek két rendezett sorozat összefésülésén alapulnak (például egy raktárban tárolt készlet naprakészítése a beszállítás / kiszállítás alapján). Ilyenkor a rendezettség nagyban növeli az algoritmus hatékonyságát. Ha a sorozatok listába vannak fűzve, akkor a pointerek állításával gyorsan és hatékonyan elvégezhető a két lista összefésülése.
- Az összefésüléses feladatok általában fejelemes listákkal kapcsolatosak, a lista típusa H1L vagy C2L, de később az előadáson fogják tanulni az összefésülő rendezést (merge sort) listára, amely viszont egy S1L listát fog rendezni összefésüléssel.
- Oldjuk most meg a következő feladatot:
  L1 és L2 egy-egy szigorúan monoton növekvően rendezett C2L lista fejelemére mutat. Mivel a kulcsok egyediek, a listát halmaznak tekinthetjük. Állítsuk elő L1-ben a két halmaz unióját, úgy hogy a szükséges elemeket L2-ből átfűzzük L1-be, a többit felszabadítjuk. Így az L2 lista az algoritmus végére kiürül.

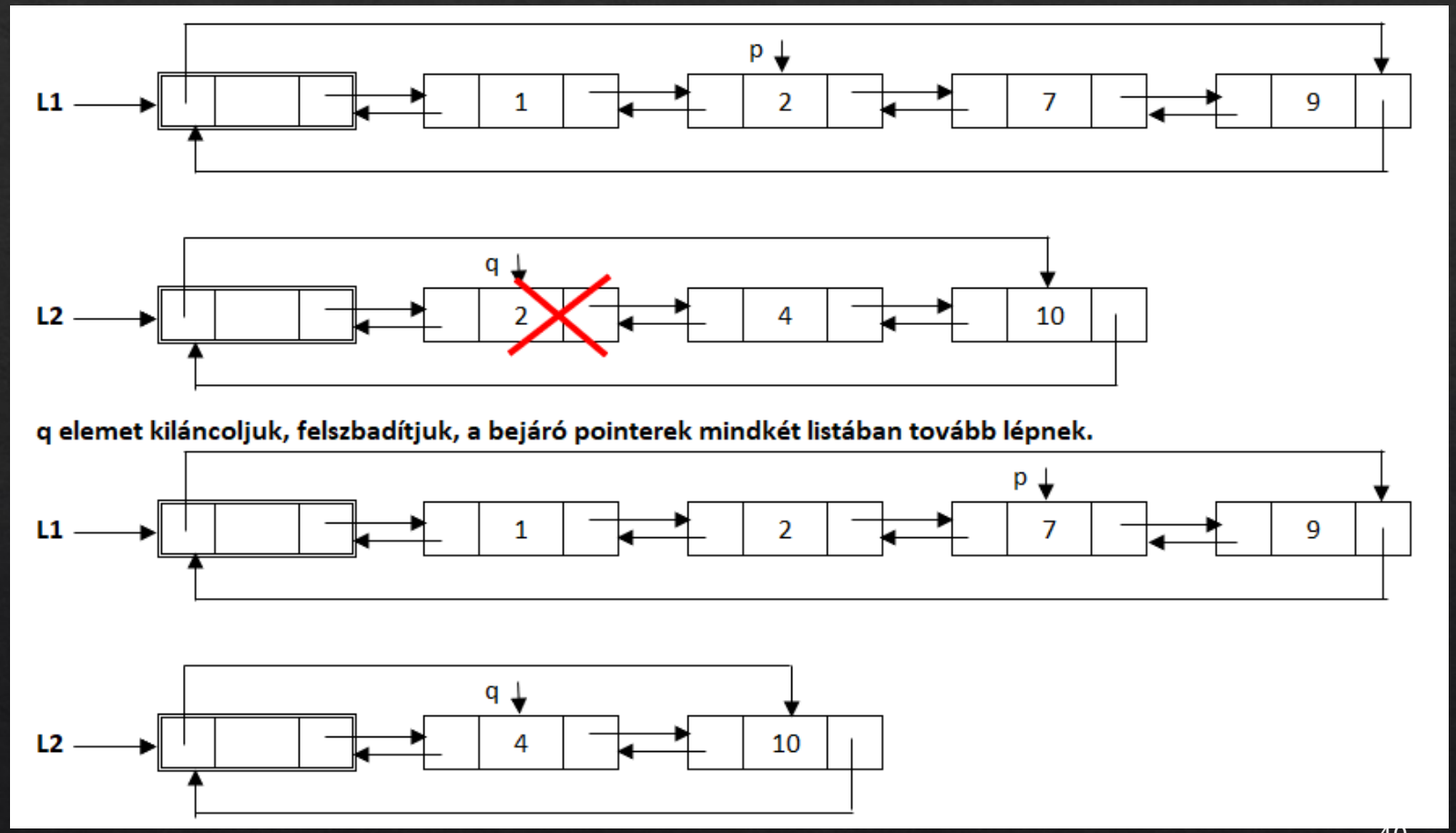
Ötlet: egy-egy bejáró pointerrel lépegetünk a listában, a pointerek jelölik ki az aktuálisan feldolgozandó elemet, három eset lehetséges

(1) L1 lista aktuális elemében lévő kulcs kisebb, mint az L2 lista aktuális elemének kulcsa, ekkor L1 lista bejáró pointere tovább lép, L2 lista bejárója nem lép.



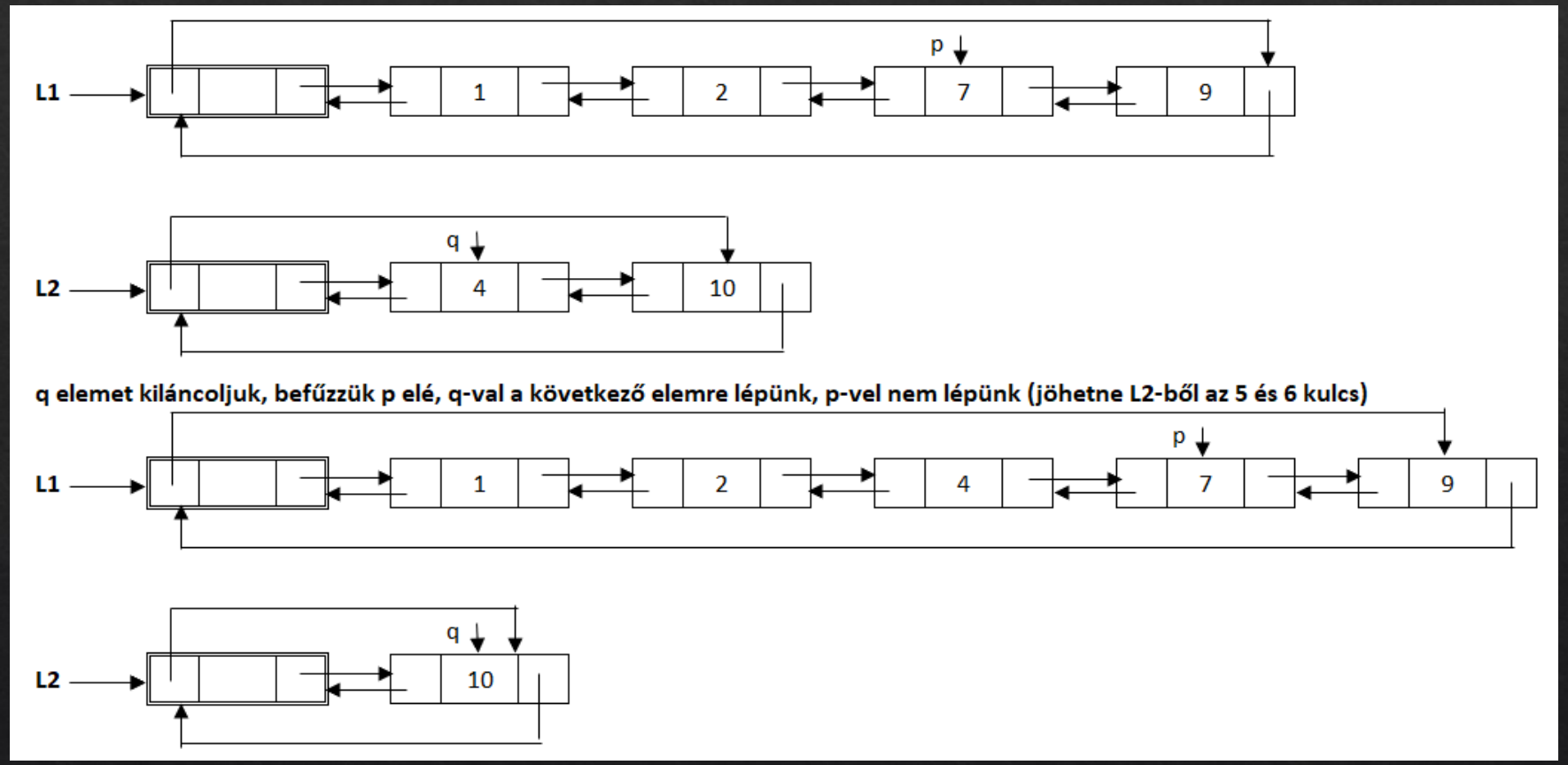
Ötlet: egy-egy bejáró pointerrel lépegetünk a listában, a pointerek jelölik ki az aktuálisan feldolgozandó elemet, három eset lehetséges

(2) L1 lista aktuális elemében lévő kulcs egyenlő az L2 lista aktuális elemében lévő kulccsal. Ekkor az L2 listában lévő elemet ki kell láncolni, fel kell szabadítani, a bejáró pointerek mindkét listában tovább lépnek.



Ötlet: egy-egy bejáró pointerrel lépegetünk a listában, a pointerek jelölik ki az aktuálisan feldolgozandó elemet, három eset lehetséges

(3) L1 lista aktuális elemében lévő kulcs nagyobb, mint az L2 lista aktuális elemében lévő kulcs. Ekkor az L2 listában lévő elemet ki kell láncolni, át kell fűzni L1 lista aktuális eleme elé, L2 listában tovább lépünk, L1 listában nem.

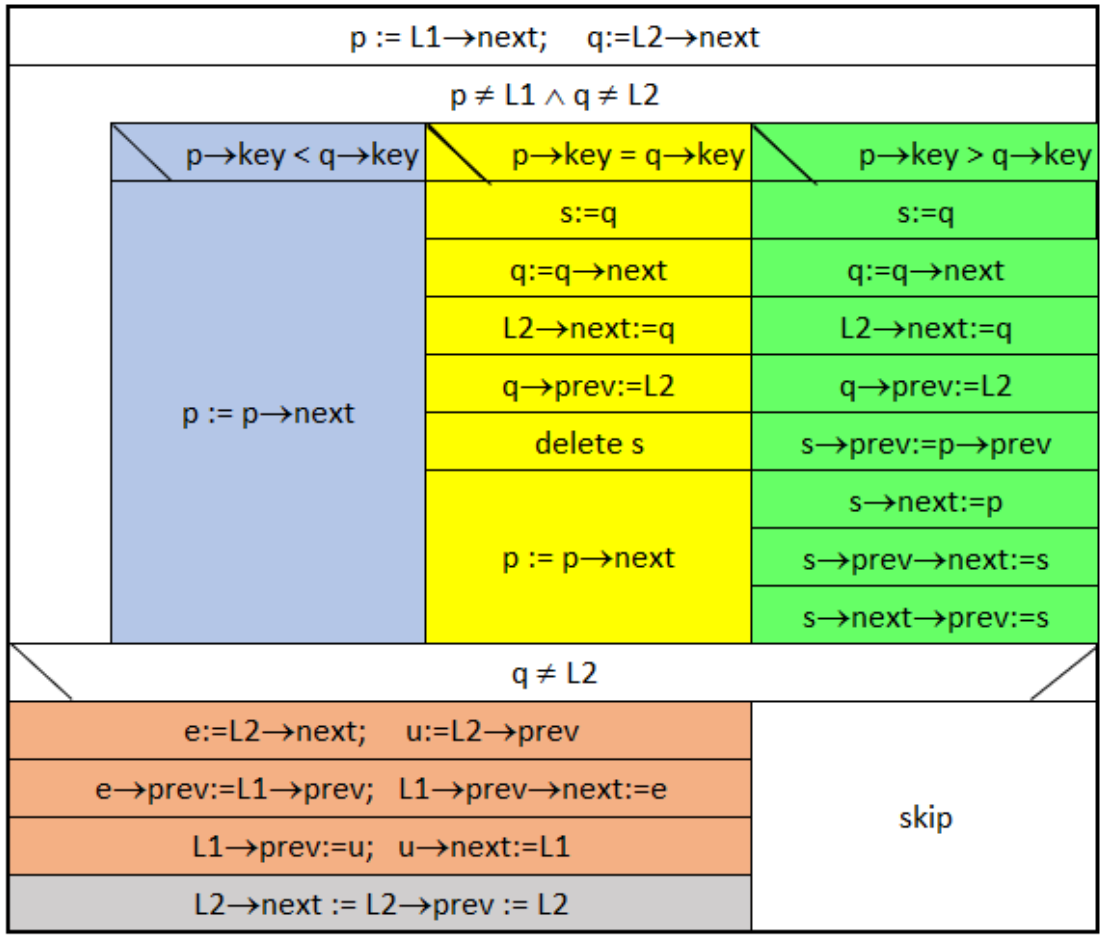


Ha valamelyik listán körbe érünk...

- Ha p=L1 akkor L1, ha q=L2, akkor L2 listán körbe értünk. (Mivel egyenlő kulcsok esetén mindkét bejáróval lépünk, az is előfordulhat, hogy egyszerre érünk a listák végére.)
- Ha L2 listán értünk körbe, akkor kész vagyunk, ha viszont L2 listában még vannak elemek, azokat be kell fűzni L1 végére. Ezt megtehetjük egyesével, de hatékonyabb, ha a maradék lánc-részt egyszerre, konstans lépésben fűzzük át.
- Megjegyzés:
  Észrevehető, hogy valójában q bejáró pointerre nem is lenne szükség, mert az algoritmus során, a ciklus indulásakor q=L2 next mindig teljesül. De a rövidebb írásmód, és érthetőbb megfogalmazás miatt q-t használunk L2 aktuális elemének címzéséhez.

Megoldás

Unio(L1: E2*, L2: E2*)



p pointerrel L1, q pointerrel L2 listán megyünk végig,
ha valamelyik pointerrel körbe értünk, akkor a ciklusból kilépünk.

a háromféle eset feldolgozása:

p →key < q →key esetben L1 listában tovább lépünk.

p →key = q →key esetén s-ben megjegyezzük q címet,
q-val tovább lépünk,
kifűzzük s-et a listából, és felszabadítjuk,
végül p-vel is tovább lépünk.

p →key > q →key esetén az előzőághoz hasonlóan s-ben lesz
az elem címe, q-val tovább lépünk, s-et kifűzzük,
majd L1-be a p elé befűzzük.

Ha L2 listában még vannak elemek, azokat még hozzá fűzzük
L1 végére. Ez konstans lépésben elvégezhető. Az e pointer
mutat az átfűzendő listadarab elejére, u pedig a végére (1.sor)
e..u listadarabot befűzzük L1 végére (2. és 3. sor)

Végül L2 lista fejelemének pointereit önmagára állítjuk, mivel
az utolsó elemek átcsatolása után üressé vált.

Műveletigény

- Vizsgáljuk meg az összefésülő algoritmusunk műveletigényét! Legyen L1 lista hossza: n, L2 lista hossza m.
- Mit mondhatunk mT(n,m) illetve MT(n,m) műveletigényről?
- Az algoritmus az egyik listán mindenképp végig iterál, tehát:
  mT(n,m) = Θ(n) abban az esetben, ha L1 minden eleme kisebb, mint L2 első eleme. Miért? Indokoljuk az állítást!
  mT(n,m) = Θ(m) abban az esetben, ha L2 minden eleme kisebb, mint L1 első eleme. Miért? Indokoljuk az állítást!
- MT(n,m) = Θ(n+m) abban az esetben, ha nincsenek azonos elemek, és L1utolsó előtti eleme kisebb, a legutolsó pedig nagyobb, mint L2 legutolsó eleme. Indokoljuk meg, miért?

Szorgalmi házi feladatok

- L egy egész számokat tartalmazó egyirányú, fejelemes lista (H1L).
  - A lista rendezetlen.
  - Készítsen algoritmust, mely a lista páratlan elemeit kifűzi és visszaadja egy S1L listába fűzve.
- Legyenek L1 és L2 szigorúan monoton növekedő C2L listák. Készítse el azt az algoritmust, mely L3 C2L típusú szigorúan monoton növekedő listában előállítja L1 és L2 listák metszetét. L1 és L2 listák ne változzanak meg. L3 nem létezik, az algoritmus hozza létre!