Algoritmusok és adatszerkezetek I.

5. gyakorlat

Emlékeztető: Únió

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

p := L1→next; q:=L2→next		
p ≠ L1 ∧ q ≠ L2		
p→key < q→key	p→key = q→key	p→key > q→key
p := p→next	s:=q	s:=q
	q:=q→next	q:=q→next
	L2→next:=q	L2→next:=q
	q→prev:=L2	q→prev:=L2
	delete s	s→prev:=p→prev
	p := p→next	s→next:=p
		s→prev→next:=s
		s→next→prev:=s
q ≠ L2		
e:=L2→next; u:=L2→prev		skip
e→prev:=L1→prev; L1→prev→next:=e		
L1→prev:=u; u→next:=L1		
L2→next := L2→prev := L2		

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.

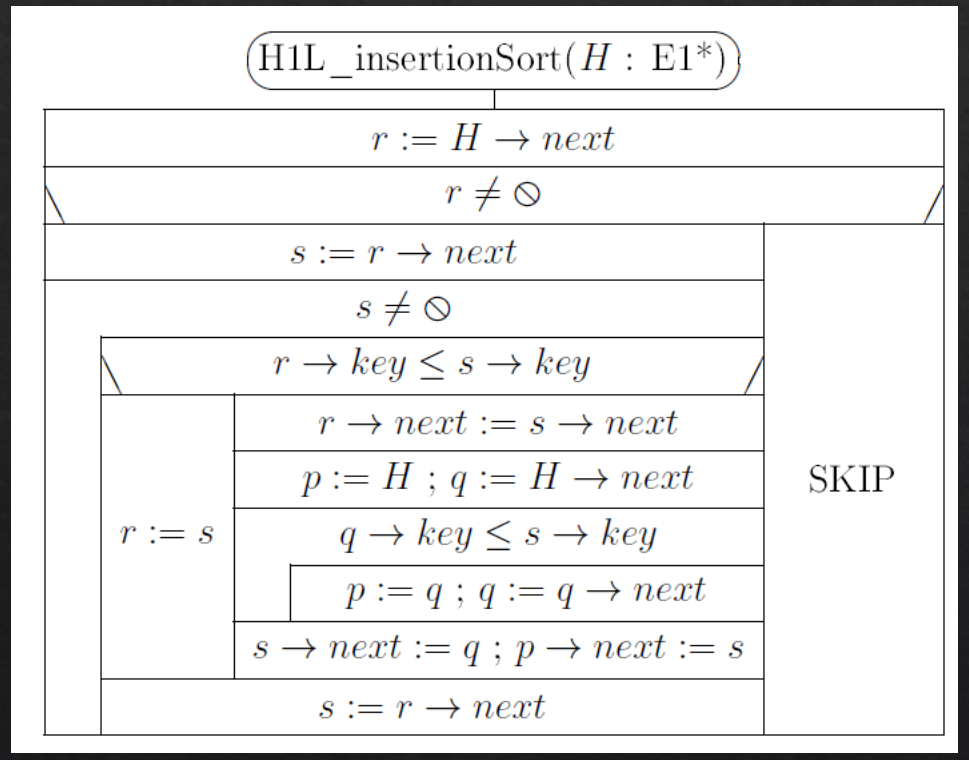
p →key > q →key esetén az előzőághoz hasonlóan s-ben lesz
az elem címe, q-val tovább lépünk, s-et kifűzzük,
majd L1-be a p elé befűzzük.

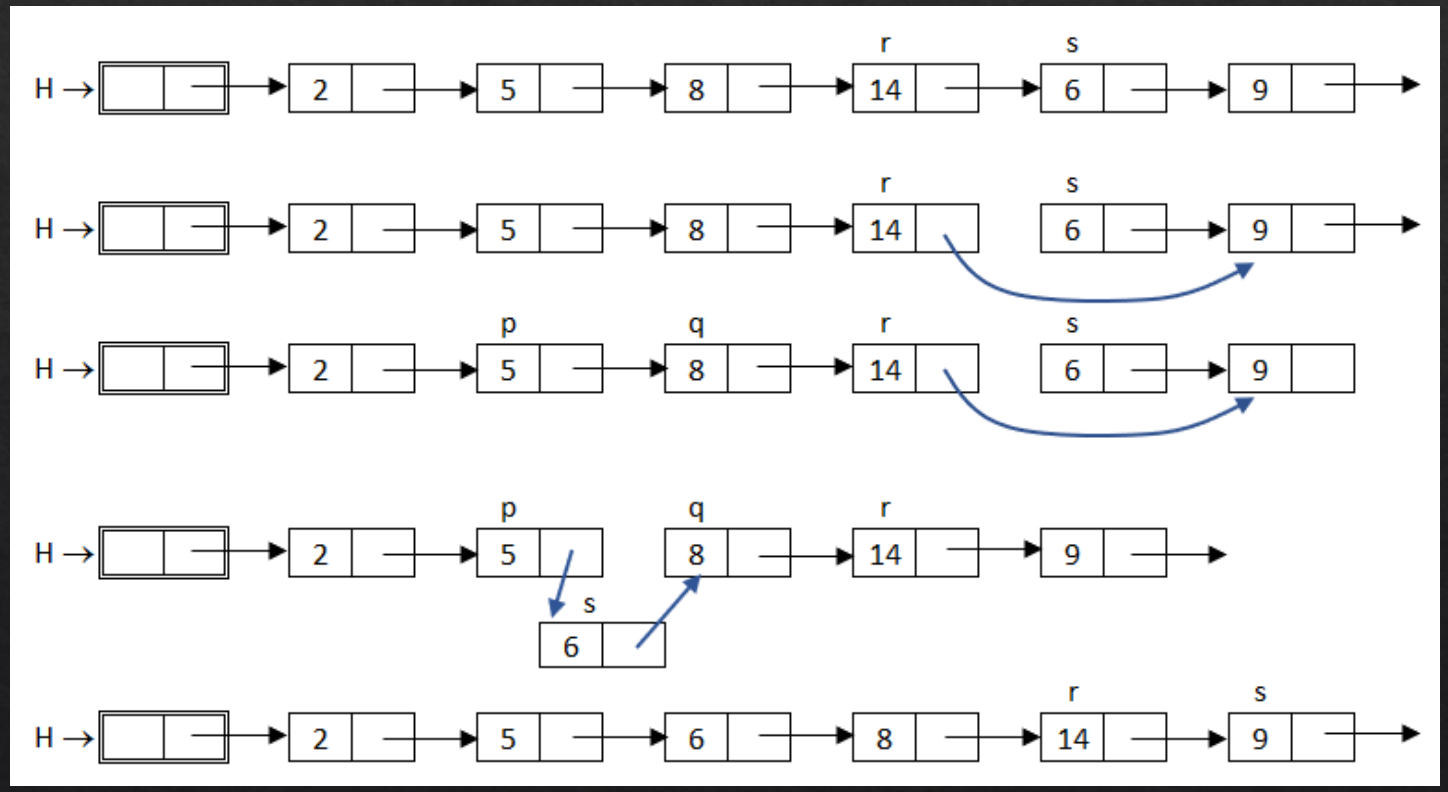
Ha L2 listában még vannak elemek, azokat még hozzá fűzzük
L1 végére. Ez konstans lépésben elvégezhető. Az e pointer
mutat az átfűzendő listadarab elejére, u pedig a végére (1.sor)
e..u listadarabot befűzzük L1 végére (2. és 3. sor)

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

Beszúró rendezés H1L-re

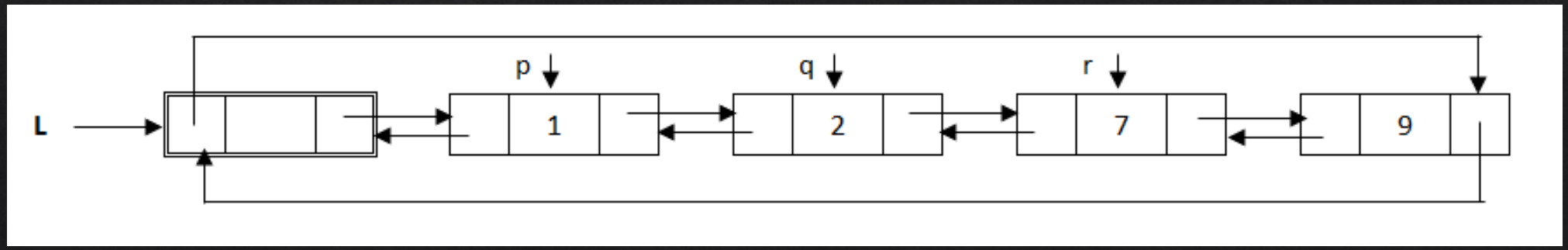
- r pointer mutatja a rendezett rész utolsó elemét.
- Ebbe szúrjuk be ha kell, az s című elemet.
- p és q közé illik az s



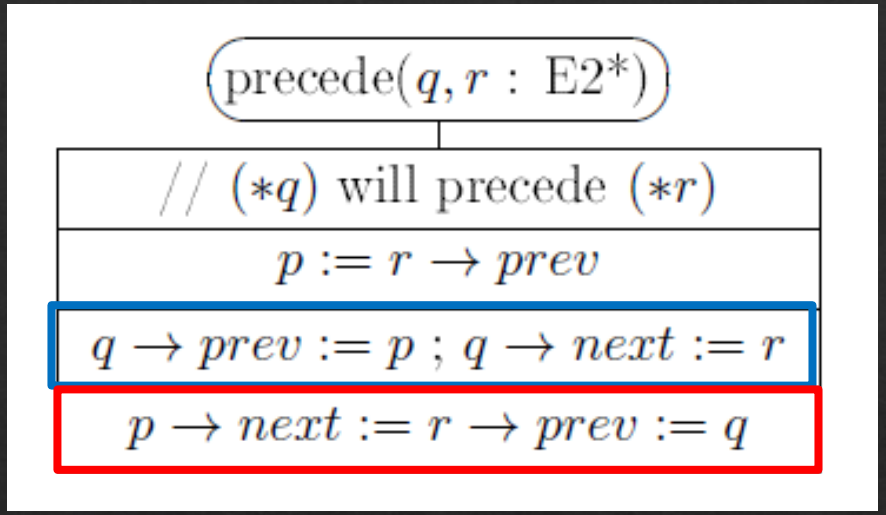


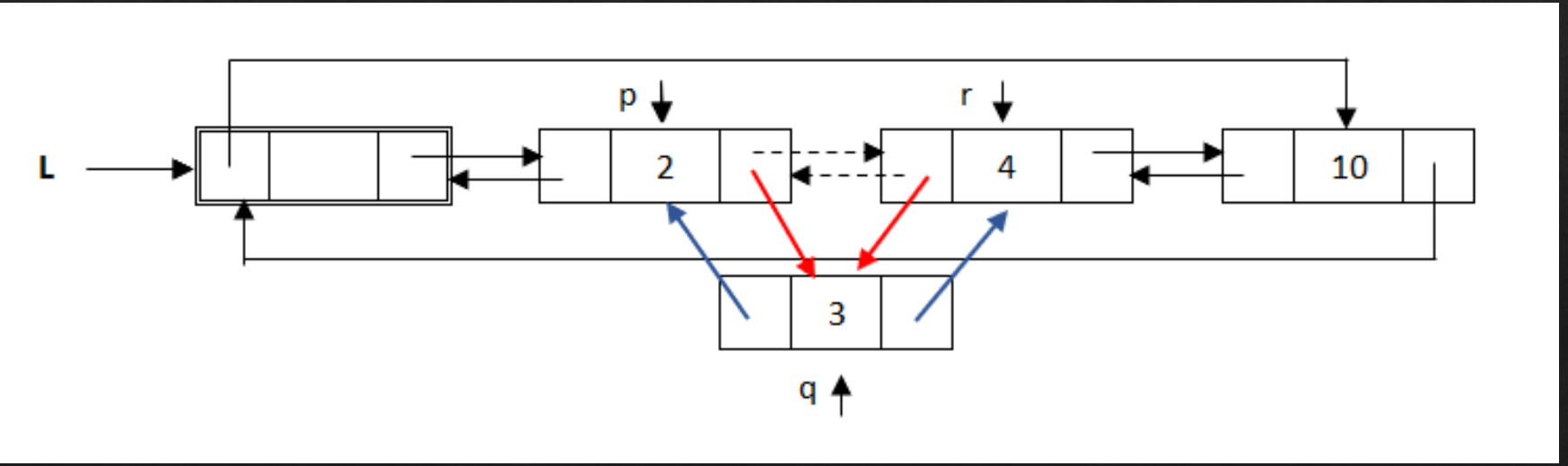
C2L listák alapvető műveletei

- Beszúrás adott elem elé:
  - precede(q,r : E2*)
- Beszúrás adott elem mögé:
  - follow(p,q : E2*)
- Adott elem kifűzése:
  - unlink (q : E2*)



precede(q,r : E2*)





follow(p,q : E2*)

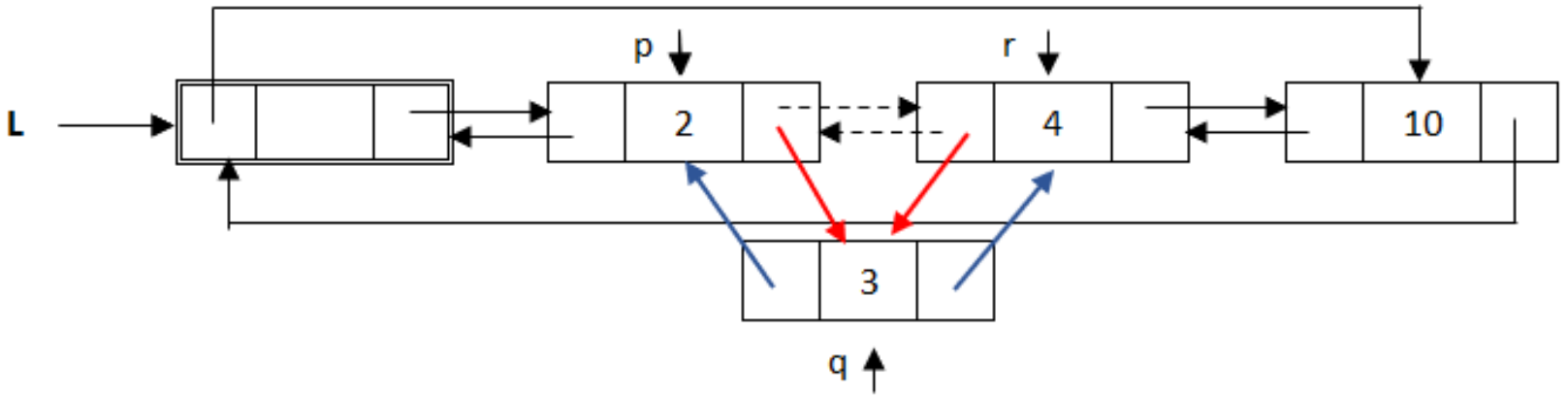
follow$(p, q :$ E2*)

// $(*q)$ will follow $(*p)$

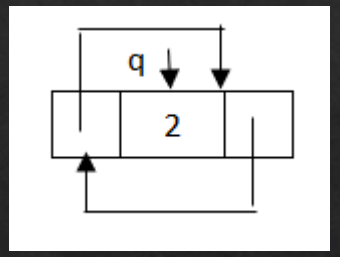
$r := p \rightarrow next$

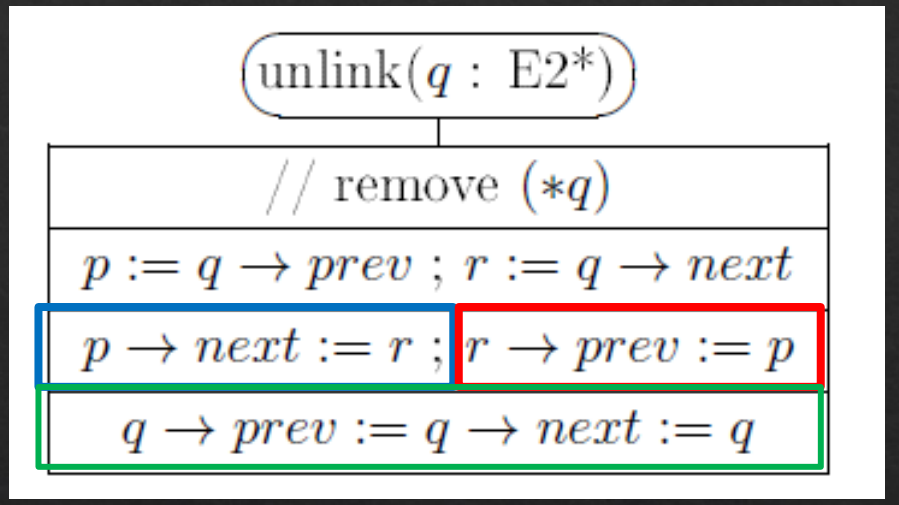
$q \rightarrow prev := p \; ; \; q \rightarrow next := r$

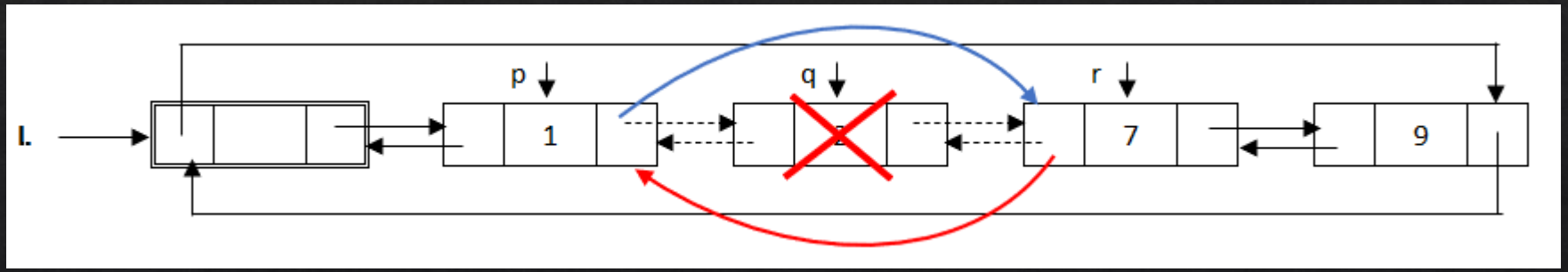
$p \rightarrow next := r \rightarrow prev := q$



unlink (q : E2*)

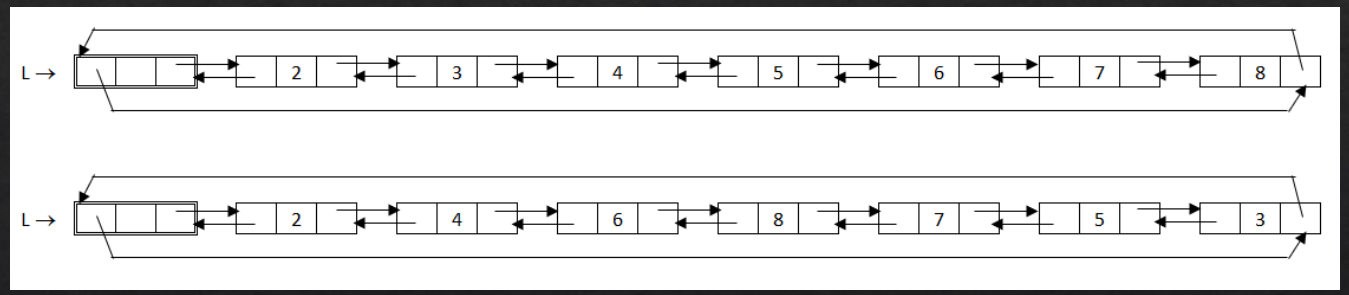

q
2


unlink(q : E2*)
// remove (*q)
p := q → prev ; r := q → next
p → next := r ;
r → prev := p
q → prev := q → next := q


I.
p
q
r
1
7
9

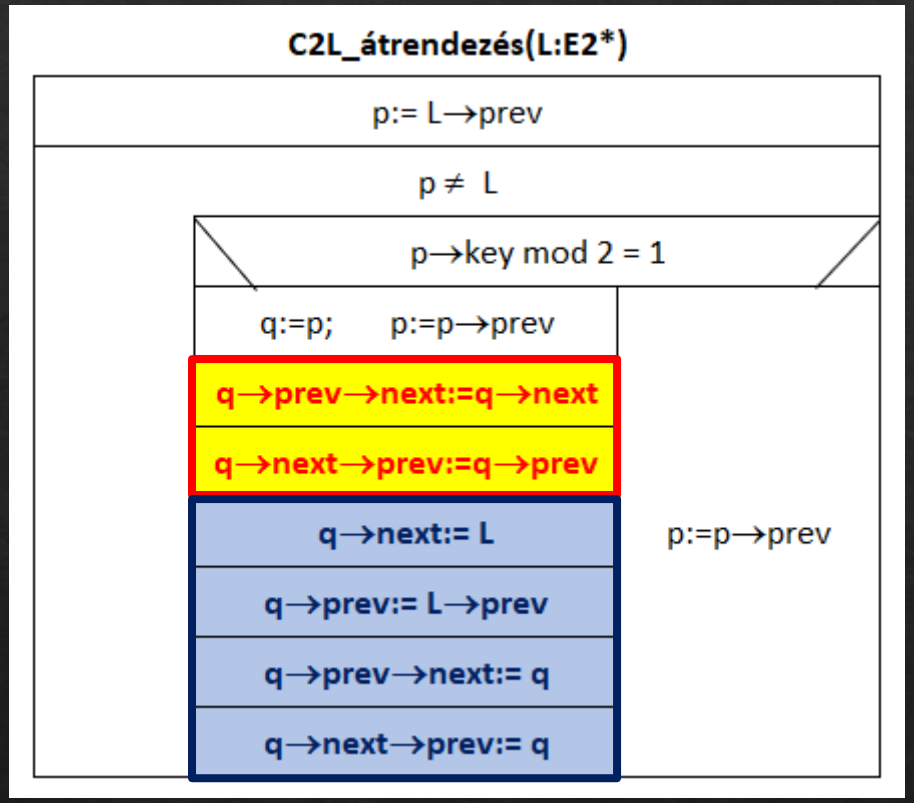
Példák a műveletek alkalmazására

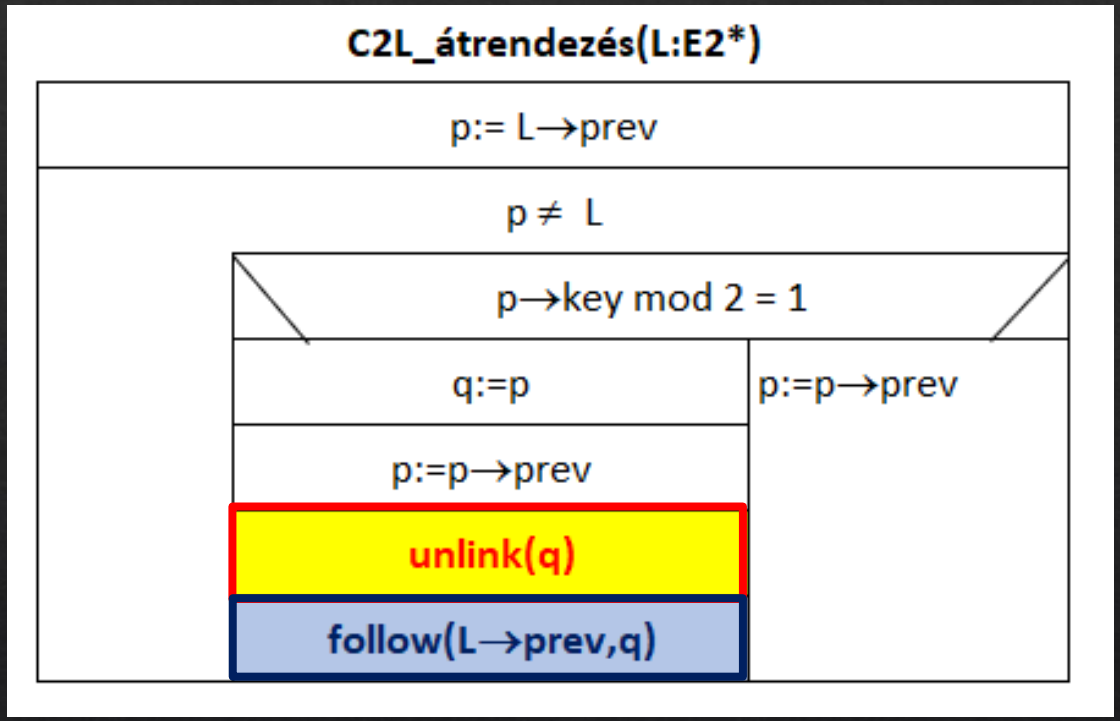
- Az előző gyakorlaton elkészítettük az alábbi feladat megoldását:
- Adott egy természetes számokat tartalmazó C2L lista. A lista szigorúan monoton növekvően rendezett.
- A lista egyszeri bejárásával rendezzük át az elemeit úgy, hogy a lista elején legyenek a páros számok növekvően, a végén pedig a páratlanok csökkenően.



- Térjünk vissza erre a feladatra, és alakítsuk át úgy a megoldásunkat, hogy az előbb bevezetett lista műveleteket használjuk.

Az algoritmus átalakítása





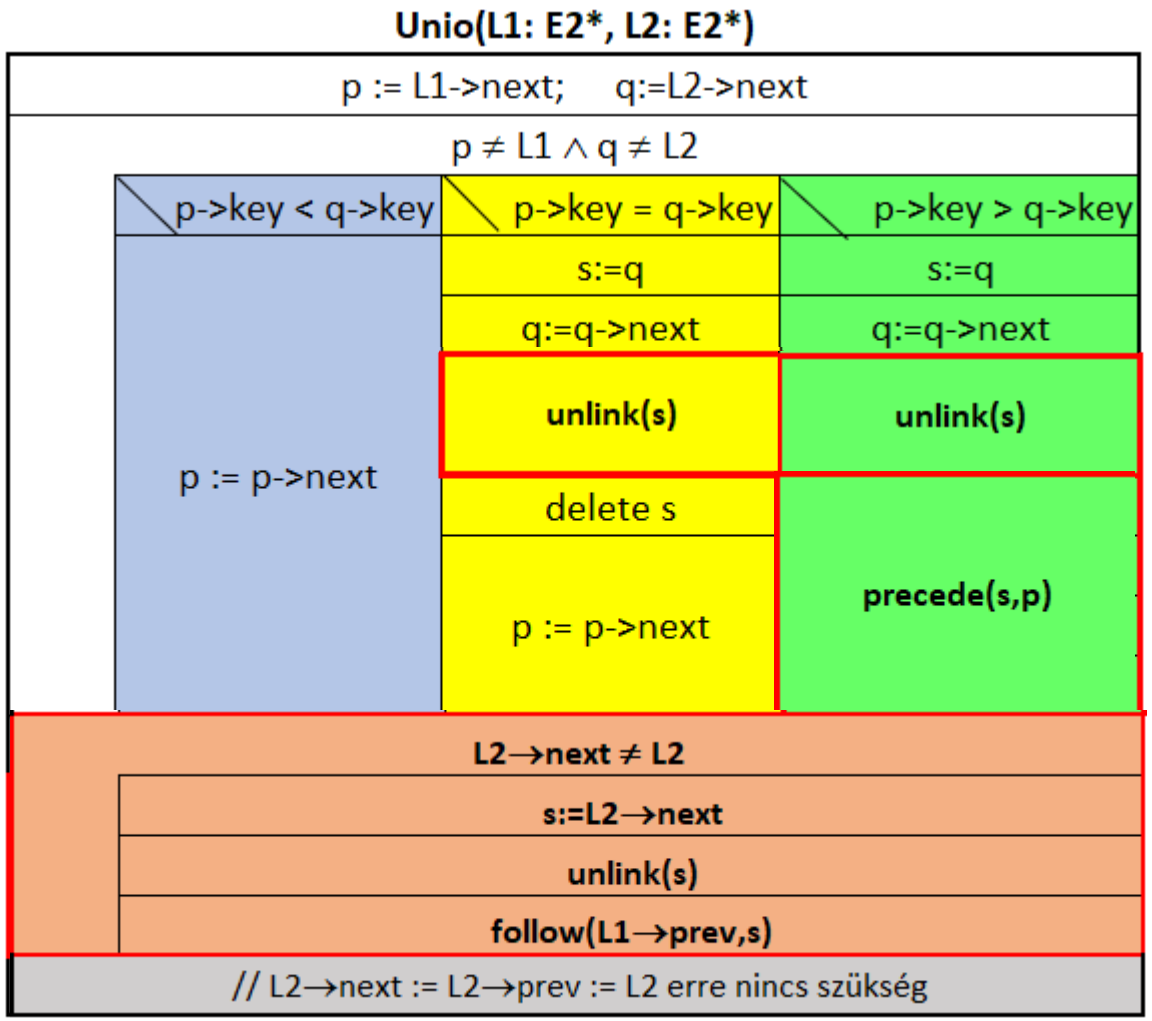
Alkalmazható a precede() is, hogyan?

precede(q,L)

Második példa

- Ezzel a feladattal is találkoztunk az előző órán (két rendezett C2L összefésülése):
- 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.
- Nézzük meg ennek is az átalakítását.

Megoldás átalakítása



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.

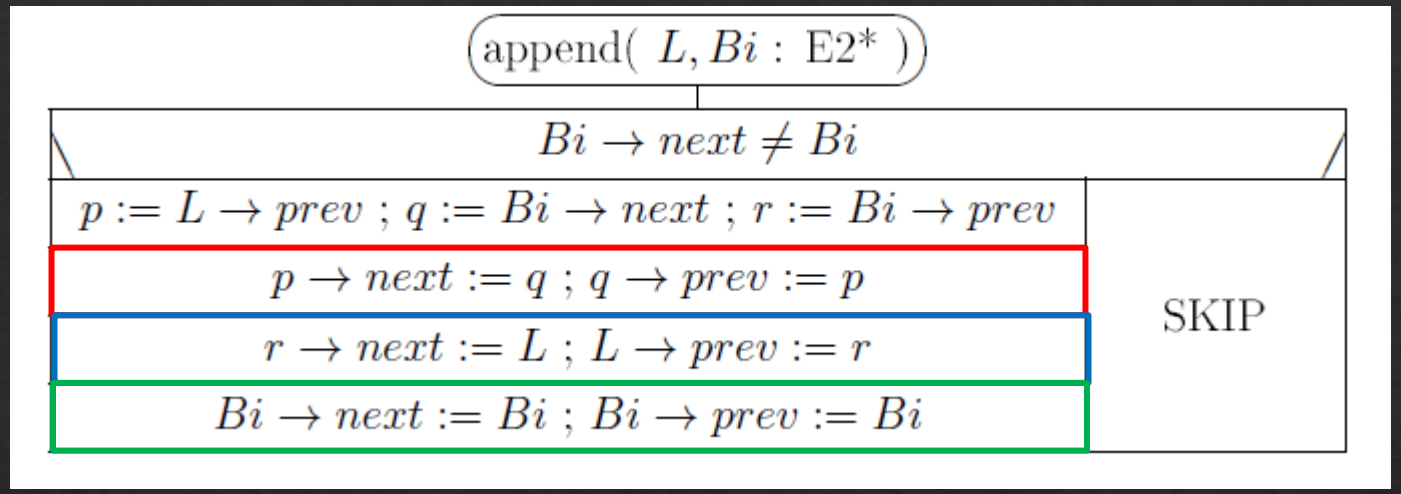
p-> key = q->key esetén s-ben megjegyezzük q címet,
q-val tovább lépünk,
kifűzzük s-et a listából, és felszabadítjuk,
végül p-vel is tovább lépünk.

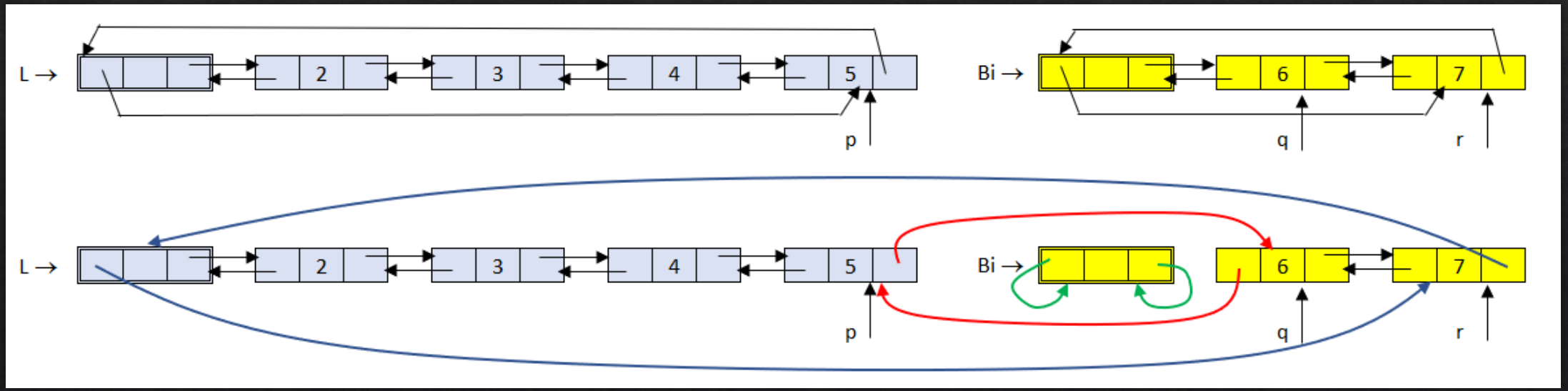
p->key > q->key esetén az előző ághoz hasonlóan s-ben lesz
az elem címe, q-val tovább lépünk, s-et kifűzzük,
majd L1-be a p elé befűzzük.

Ha L2 listában még vannak elemek, azokat még hozzá fűzzük
L1 végére. A definiált műveletekkel csak egyesével
tudjuk végrehajtani.
Ciklusunk lépésszáma O(m), ha L2 lista elemszáma m.

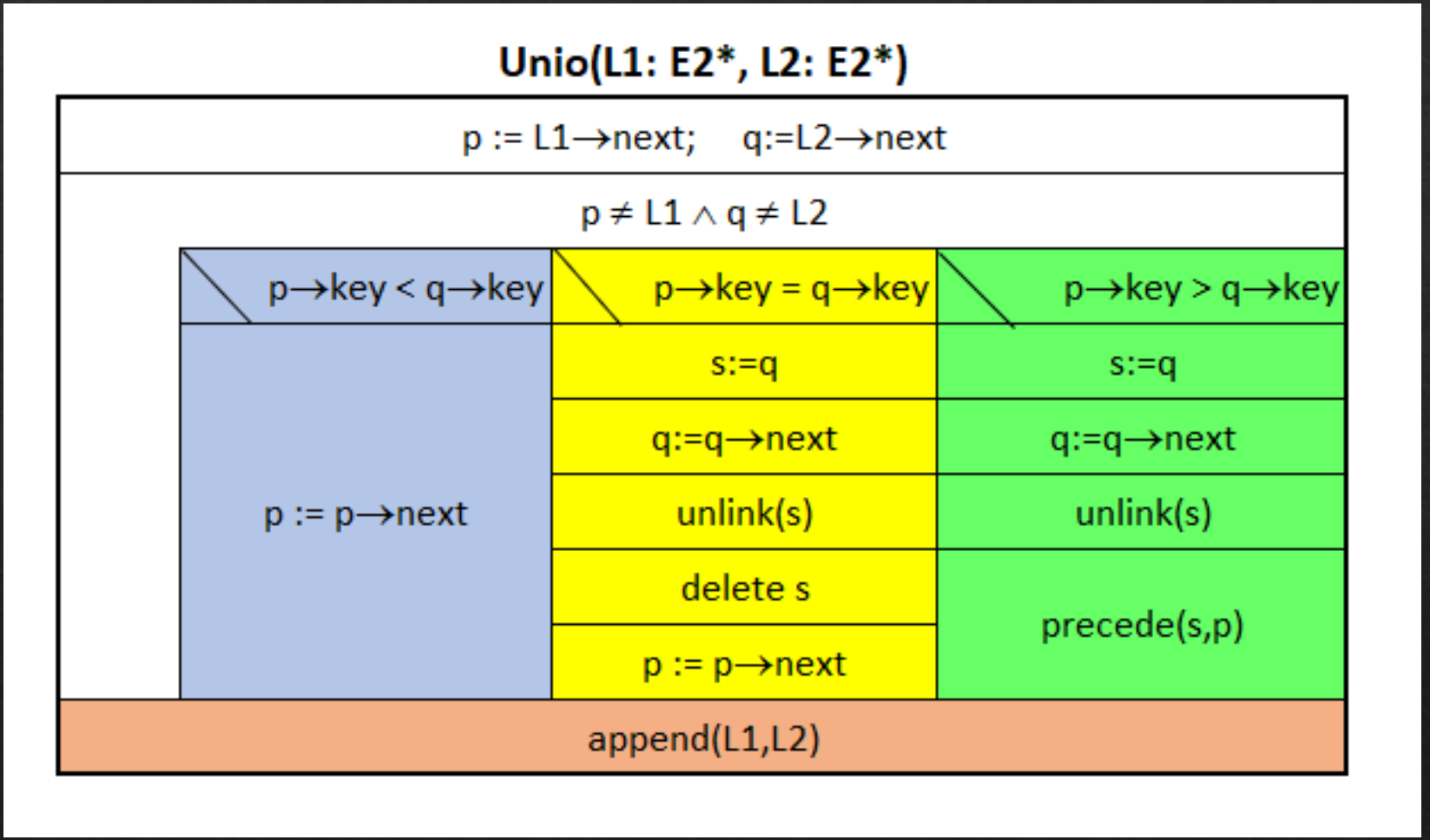
append művelet

(jegyzet, radix rendezés listára, 116. oldal)



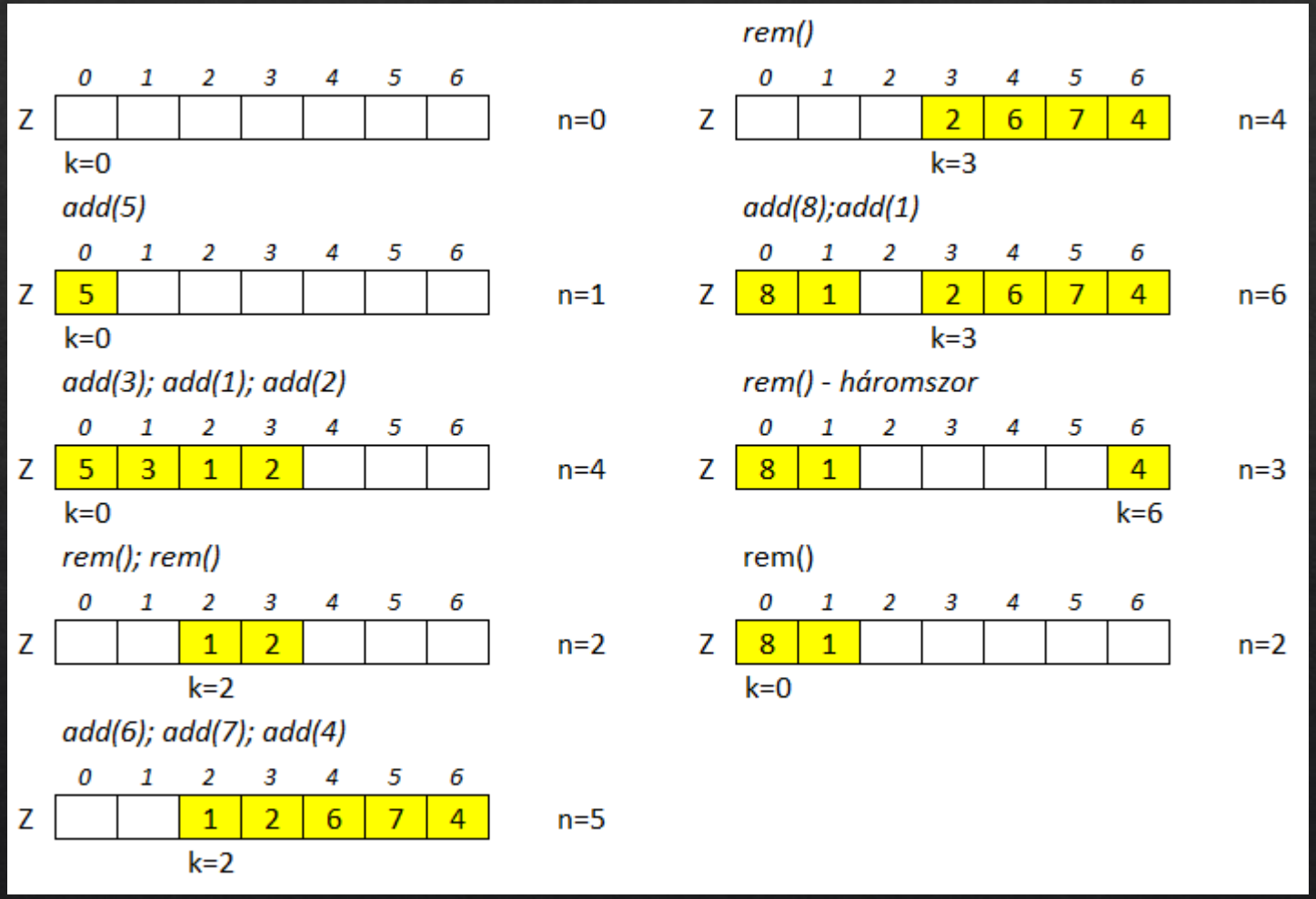


Unió végleges változat



Sor (Queue)

- FIFO (First In First Out)
- Műveletek futási ideje: $\Theta(1)$
- Sor ábrázolása tömbben:
  - A sor „körbe-körbe” jár.
  - A tömböt itt célszerű nullától indexelni.
  - Ha sor első elemének indexét (k), és a sor elemszámát (n) ismerjük, akkor ki is tudunk venni, és be is tudunk tenni konstans időben.
  - Első elem (ha a sor nem üres): Z[k]
  - Első szabad hely (ha a sor nincs tele): Z[k+n mod m]



Queue

$-Z : \mathcal{T}[]$ // $\mathcal{T}$ is some known type
$-n : \mathbb{N}$ // $n \in 0..Z.length$ is the actual length of the queue
$-k : \mathbb{N}$ // $k \in 0..(Z.length - 1)$ is the starting position of the queue in array Z

$+$ Queue$(m : \mathbb{N})\{\ Z := \textbf{new}\ \mathcal{T}[m]\ ;\ n := 0\ ;\ k := 0\ \}$ // create an empty queue
$+$ add$(x : \mathcal{T})$ // join x to the end of the queue
$+$ rem$() : \mathcal{T}$ // remove and return the first element of the queue
$+$ first$() : \mathcal{T}$ // return the first element of the queue
$+$ length$() : \mathbb{N}$ $\{\textbf{return}\ n\}$
$+$ isFull$() : \mathbb{B}$ $\{\textbf{return}\ n = Z.length\}$
$+$ isEmpty$() : \mathbb{B}$ $\{\textbf{return}\ n = 0\}$
$+ \sim$ Queue() { **delete** Z }
$+$ setEmpty() $\{n := 0\}$ // reinitialize the queue

Sor UML ábra

Queue::add$(x : \mathcal{T})$

$n < Z.length$	
$Z[(k+n) \bmod Z.length] := x$	fullQueueError
$n{+}{+}$	

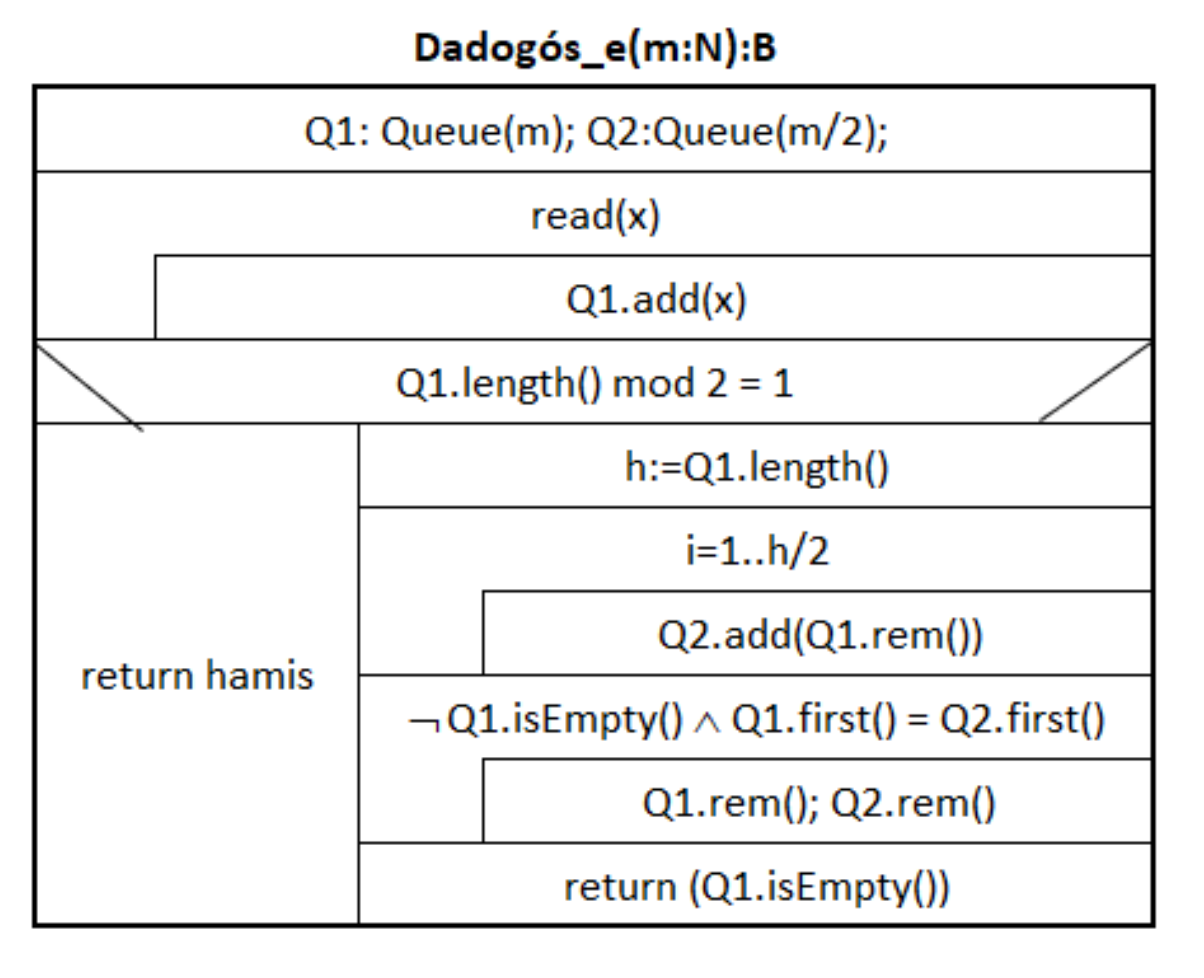
Queue::rem$() : \mathcal{T}$

$n > 0$	
$n{-}{-}$	emptyQueueError
$i := k$	
$k := (k+1) \bmod Z.length$	
return $Z[i]$	

Egy példa sor használatára

- Bemenetről érkezik egy szöveg.
- Két sor segítségével döntsük el, hogy „dadogós-e” a szöveg!
- Példa dadogós szövegre:
  - aa
  - almaalma
  - <> (üres szöveg is dadogósnak tekinthető)
- Ötlet:
  - Olvassuk be a szöveget egy sorba.
  - Ha páratlan hosszú, nem lehet dadogós.
  - Ha páros, másoljuk át a felét egy másik sorba, és hasonlítsuk össze a két sor tartalmát.

Megoldás



- Létrehozzuk a sorokat.
- Beolvassuk a teljes szöveget Q1 sorba.
- Ha páratlan hosszú, nem lehet dadogós.
- Q1 sor „felét” átmásoljuk Q2-be.
- Összehasonlítjuk a két sor tartalmát: ha a két első elem egyezik, minden rendben van, kivesszük őket, és folytatjuk.
- Ha üres lett Q1 (Q2 is üres ilyenkor), akkor végig egyezett a két sor, dadogós a szöveg.

Sor (Queue) láncolt ábrázolása

- Queue típus műveletei:
- rem() : T
- add(x : T)
- first() : T
- length() : N
- isEmpty() : B
- setEmpty()

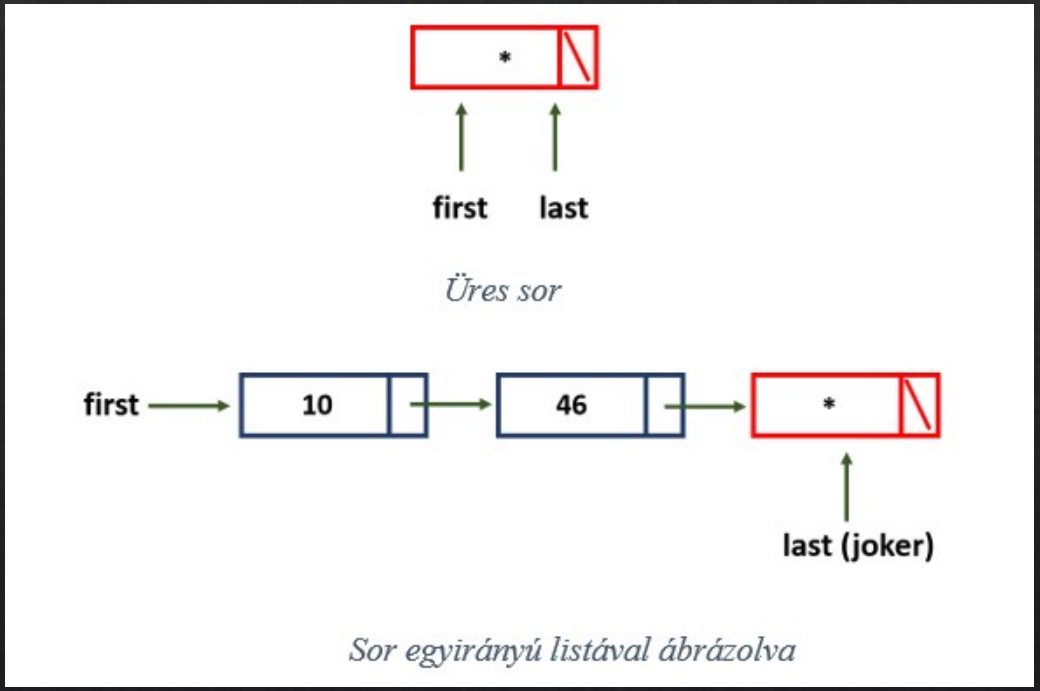
Műveletigény tömbös ábrázolás esetén: Θ (1) (valamennyi művelet esetén)

Ha az add műveletnél megengedjük, hogy a sort
ábrázoló tömb mérete meg tudjon növekedni, annak műveletigénye mindig épp az aktuális hossz,
de mivel ez ritkán fordul elő, átlagosan az add ekkor
is konstans lépésszámúnak tekinthető.

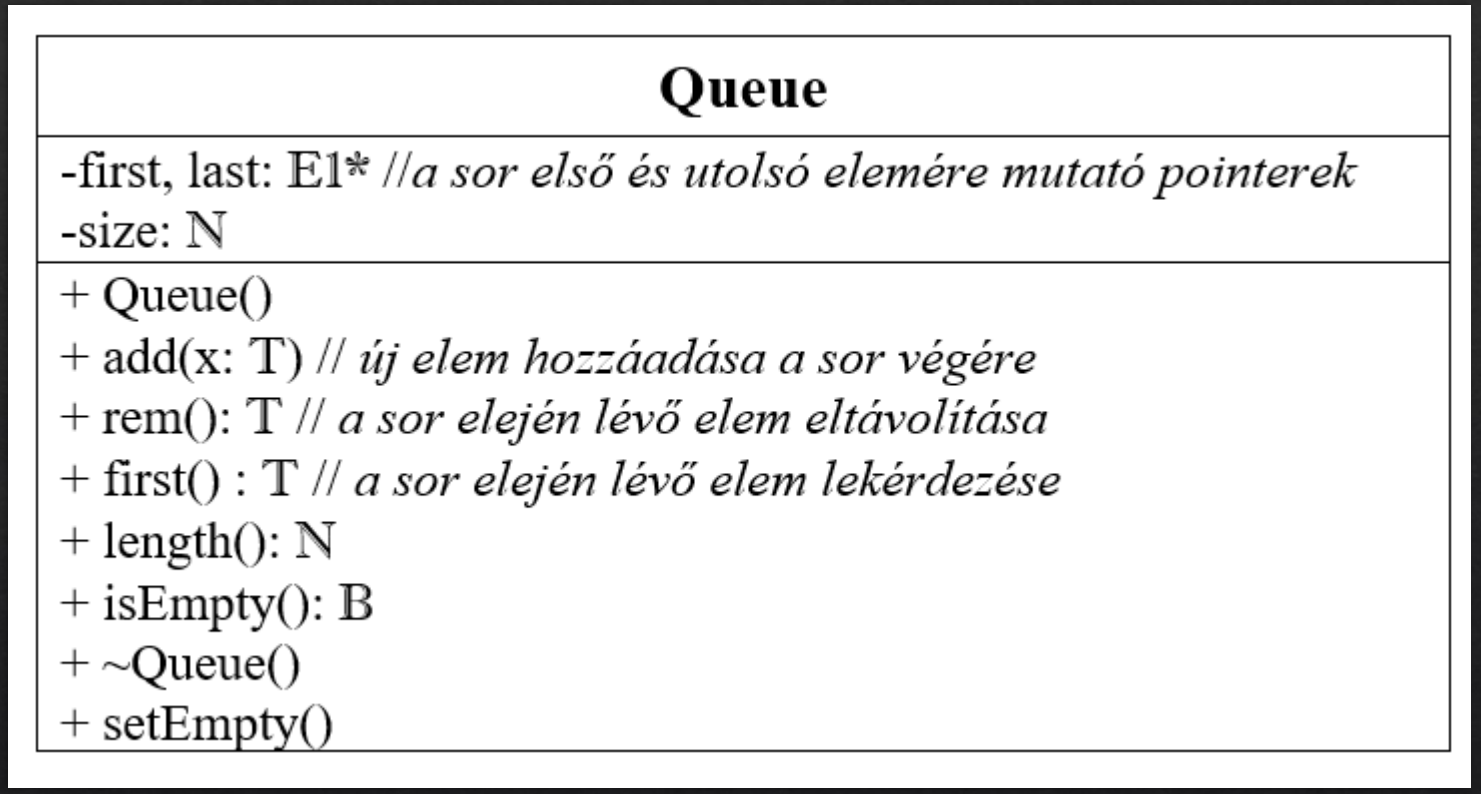
Ezt listás ábrázolás esetén is biztosítani kell!

Első módszer

- Ötlet: A sort egyegyirányú lista ábrázolja, két pointerrel.
- A lista mindig tartalmaz egy üres (***joker***) lista elemet, ami a sor végén fog elhelyezkedni.
- Új elem hozzáadásakor a beszúrandó kulcsot a joker elemben tároljuk el, majd készítünk egy új üres joker elemet, amit a lista végére fűzünk.
- A joker elem címét tárolja a last pointer. A lista segítségével elméletileg korlátlan hosszúságú sort hozhatunk létre (amíg a new művelet sikeresen le tud futni).
- A joker elem hatékonysági célokat szolgál, nem lesz sohasem üres a lista.



Az UML diagram



A műveletek struktogramjai

Queue::Queue()

first := last := new E1
first->next := 0
size := 0

A konstruktor létrehozza a joker elemet.

Queue::add(x: T)

last->next := new E1
last->key := x
last := last->next
last->next := 0
size := size + 1

Queue::setEmpty()

first ≠ last	
	p := first
	first := first->next
	delete p
size:= 0	

Queue::rem(): T

size = 0	
Error	x := first->key
	s := first
	first := first->next
	delete s
	size := size - 1
	return x

Queue::first(): T

size = 0	
Error	return first->key

Queue::isEmpty(): B

return size = 0

Queue::length(): N

return size

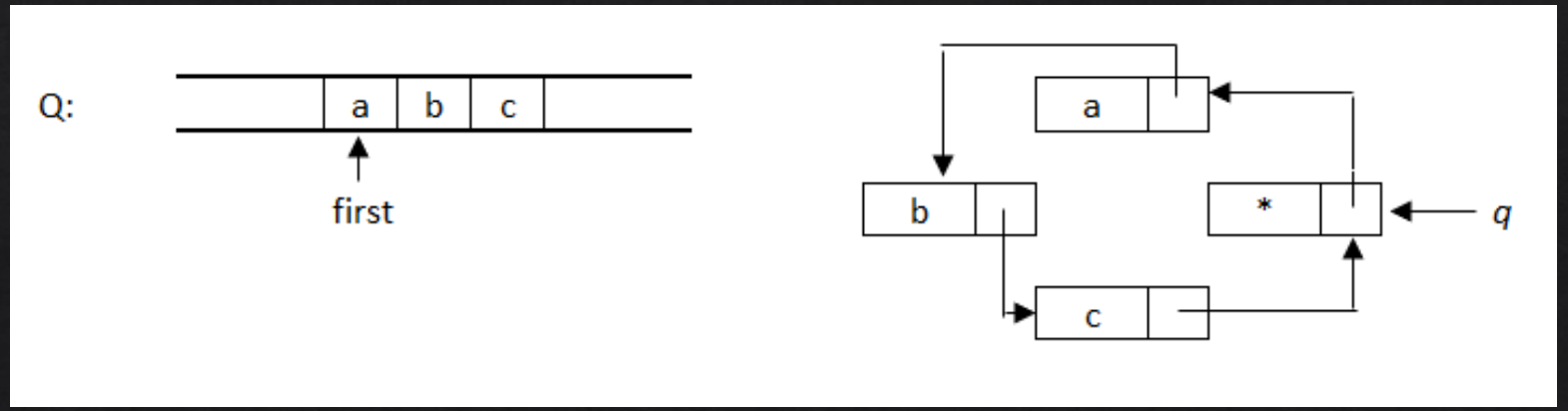
Queue::~Queue()

first ≠ 0	
	p := first
	first := first->next
	delete p

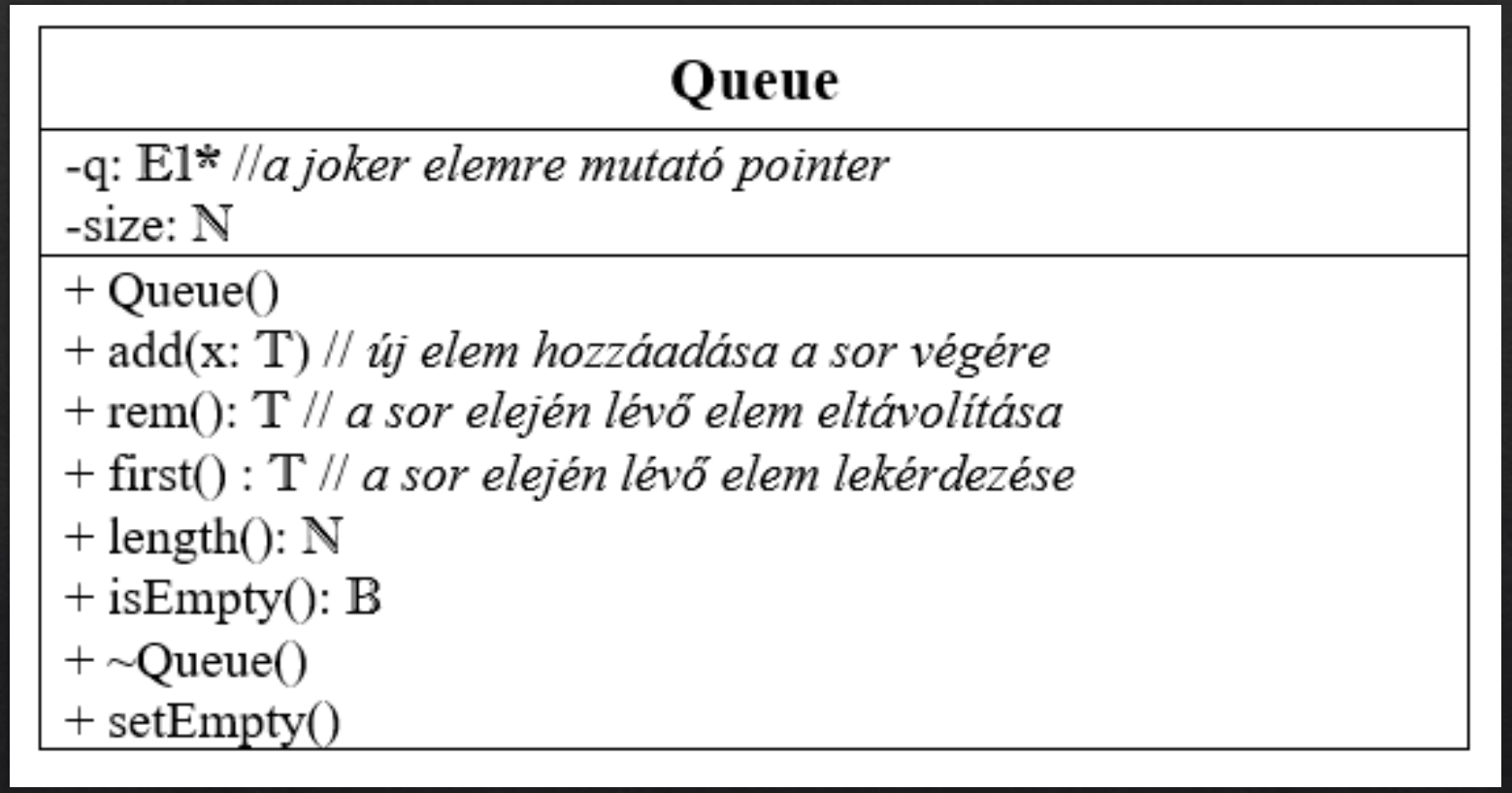
Műveletigény a setEmpty() és a destruktort kivéve : $\Theta(1)$ (azok a lista hosszával arányosak)

Második módszer

- Nagyon érdekes ötleten alapul. Egy egyirányú, ciklikus lista ábrázolja a sort. Itt is van egy joker elem, ami a sor „végén” található. A joker elem mutat az első elemre.
- Egyetlen pointerre van szükség (az ábrán ez q), aminek a joker elemre kell mutatnia.
- A rem() művelet esetén az első elem így azonnal elérhető a q segítségével és kifűzhető.
- Az add() művelet esetén a joker elembe kerül sorba berakni kívánt érték, majd létrehozunk egy új joker elemet, befűzzük a láncba, és q-val erre az új elemre mutatunk.



Az UML diagram



Az érdekesebb műveletek struktogramjai

Queue::Queue()

q := new E1
q->next := q
size := 0

Queue::rem(): T

size = 0	
ERROR	r := q->next->next
	x := q->next->key
	delete q->next
	q->next := r
	size:= size-1
	return x

Queue::add(x: T)

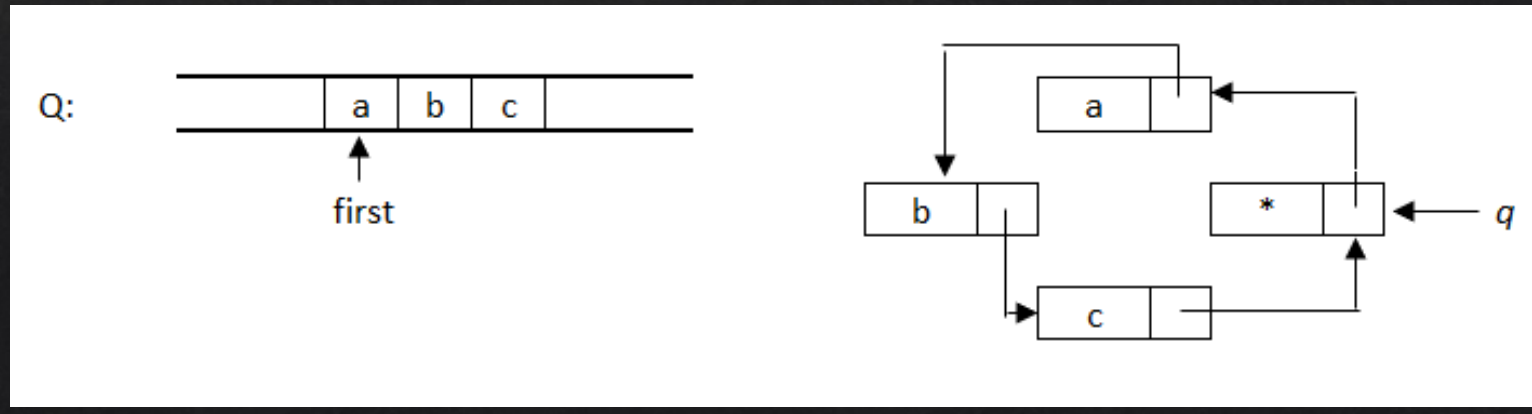
r := q->next
q->key := x
q->next := new E1
q := q->next
q->next := r
size:= size+1

Queue::setEmpty()

p := q->next	
p ≠ q	
	r := p
	p := p->next
	delete r
q->next := q	
size:= 0	

Queue::~Queue()

p := q->next		
	p ≠ q	
		r := p
		p := p->next
		delete r
delete q		



szorgalmi házi feladat

- Készítse el a verem típus láncolt ábrázolását. Egyirányú egyszerű listával (S1L) kell a vermet ábrázolnia. Adjon UML diagramot, és adja meg a verem műveleteinek struktogramját.
- Hogyan valósítható meg a maximum kiválasztásos rendezés H1L listára? Készítse el a következő algoritmust: először keressük meg a lista maximális elemét, majd fűzzük ki a listából, és egy segéd (S1L) listába fűzzük be. Ezután keressük meg a második legnagyobb elemét és fűzzük a segéd lista elejére. Folytassuk ezt, amíg egy elem marad az eredeti listában. Ekkor a segéd listát fűzzük a megmaradt elem mögé. Mit mondhatunk MT(n) és mT(n) futási időkről, ha n a lista elemszáma?