

Algoritmusok és adatszerkezetek I.

2. Előadás

Elemi adatszerkezetek és adattípusok:
a verem (Stack) és a sor (Queue) típus.
Láncolás, egyirányú listák

Tartalom

- Elemi adatszerkezetek és adattípusok
- Verem (stack) adattípus: LIFO
- Sor (queue) adatszerkezet: FIFO
- Lineáris adatszerkezetek
- Egyirányú listák (one-way or singly linked lists)
- Beszúró rendezés H1L listákra
- Összefésüléssel rendezés S1L-re

Elemi adatszerkezetek és adattípusok

- Adatszerkezet
 - adatok tárolásának és elrendezésének egy lehetséges módja
 - adatok elérése, módosítása, beszúrása, törlése
 - A megfelelő adatszerkezetek kiválasztása vagy megalkotása a programozási feladat megoldásának alapvető része
 - A programok hatékonysága nagymértékben függ az alkalmazott adatszerkezetektől
- Adattípus
 - Adatszerkezet + a rajta értelmezett műveletek
- Absztrakt adattípus (ADT) (Abstract Data Type)
 - Nem definiáljuk pontosan az adatszerkezetet, csak informálisan a műveleteket
 - Az ADT megvalósításának részei: (UML)
 - reprezentálása: adatszerkezet megadása (adattagok)
 - implementálása: műveleti kód (metódusok)

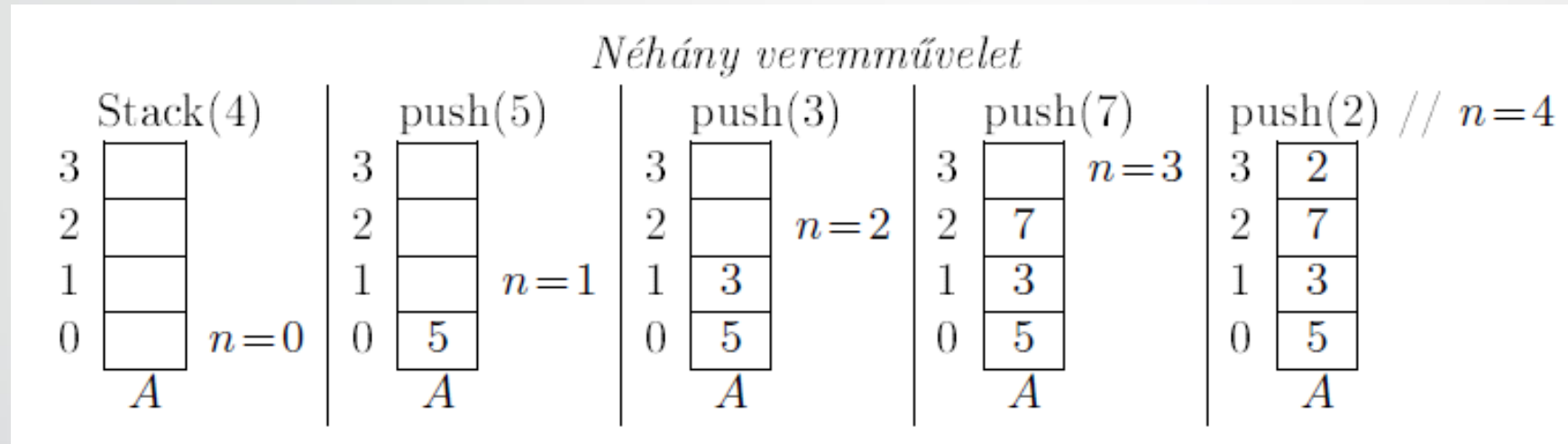
Verem (stack) adattípus: LIFO

- Az utoljára benne eltárolt, és még benne lévő adat érhető el, illetve törölhető

Stack

- $A : \mathcal{T}[]$ // \mathcal{T} is some known type ; $A.length$ is the physical
- constant $m0 : \mathbb{N}_+ := 16$ // size of the stack, its default is $m0$.
- $n : \mathbb{N}$ // $n \in 0..A.length$ is the actual size of the stack
- + $\text{Stack}(m : \mathbb{N}_+ := m0) \{ A := \text{new } \mathcal{T}[m] ; n := 0 \}$ // create empty stack
- + $\sim \text{Stack}() \{ \text{delete } A \}$
- + $\text{push}(x : \mathcal{T})$ // push x onto the top of the stack
- + $\text{pop}() : \mathcal{T}$ // remove and return the top element of the stack
- + $\text{top}() : \mathcal{T}$ // return the top element of the stack
- + $\text{isEmpty}() : \mathbb{B} \{ \text{return } n = 0 \}$
- + $\text{setEmpty}() \{ n := 0 \}$ // reinitialize the stack

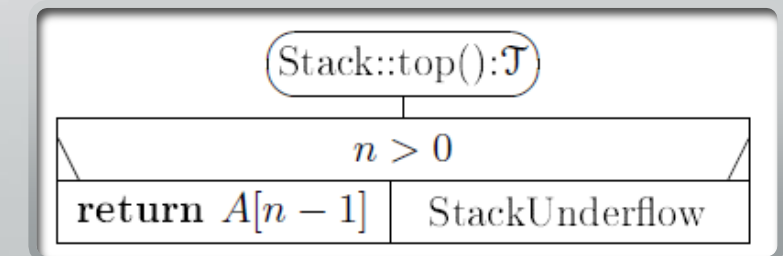
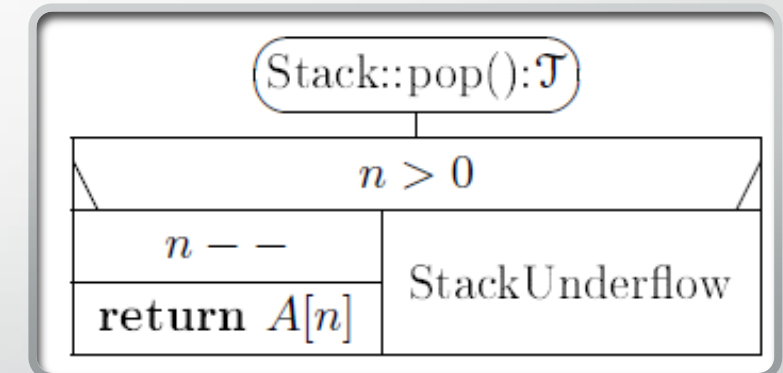
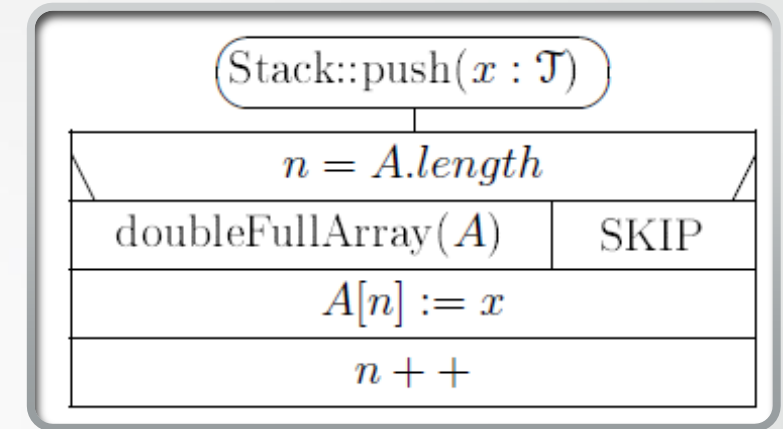
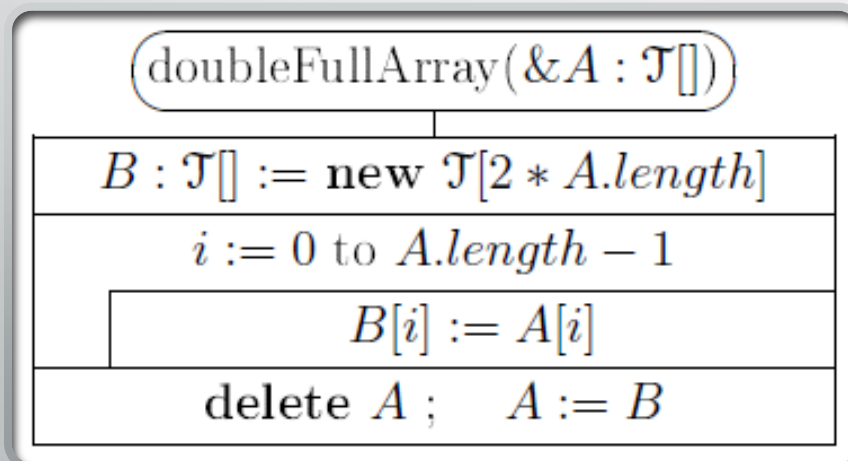
Verem példa



- Gyakorlati alkalmazása
 - CALL STACK
 - Kifejezések postfix formára hozása
 - Postfix forma kiértékelés

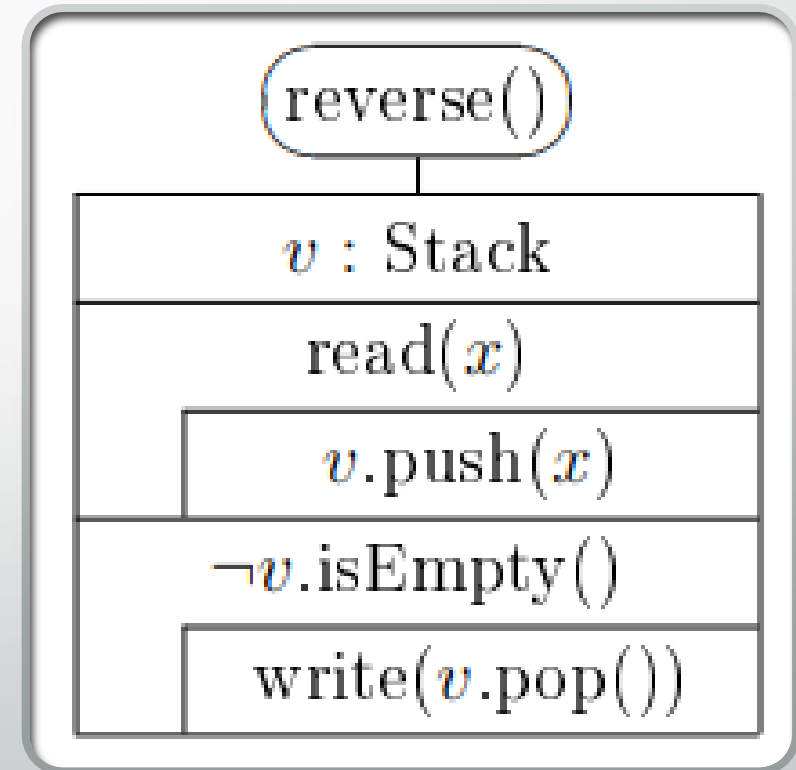
Verem műveletek

- Műveletidő: $\Theta(1)$
 - Nem tartalmaz ciklust / rekurziót
- Push művelet:
 - $mT(n) \in \Theta(1)$
 - $MT(n) \in \Theta(n)$
 - $AT(n) \in \Theta(1)$



Példa a verem használatára

- Az input adatok kiírása fordított sorrendben
- $\text{read}(\&x:\mathcal{T})$: \mathbb{B} függvény
 - a kurrens inputról olvas
 - sikeres (visszatérés: igaz)
 - ha nincs még vége az inputnak
 - Ilyenkor beolvassa x -be a következő input adatot
 - Sikertelen (visszatérés: hamis):
 - vége van az inputnak
 - Ekkor x értéke definiálatlan
- $\text{write}(x)$: a kurrens outputra írja x értékét



Sor (queue) adattípus: FIFO

Queue

- $Z : \mathcal{T}[]$ // \mathcal{T} is some known type ; $Z.length$ is the physical
 - constant $m0 : \mathbb{N}_+ := 16$ // length of the queue, its default is $m0$.
 - $n : \mathbb{N}$ // $n \in 0..Z.length$ is the actual length of the queue
 - $k : \mathbb{N}$ // $k \in 0..(Z.length - 1)$: the starting position of the queue in Z
-
- + Queue($m : \mathbb{N}_+ := m0$) { $Z := \mathbf{new} \mathcal{T}[m]$; $n := 0$; $k := 0$ }
// 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} { **return** n }
 - + isEmpty() : \mathbb{B} { **return** $n = 0$ }
 - + ~ Queue() { **delete** Z }
 - + setEmpty() { $n := 0$ } // reinitialize the queue

Sor példa

Egy sor néhány művelete

<p>Queue(4)</p> <p>0 1 2 3</p> <table><tr><td></td><td></td><td></td><td></td></tr></table> <p>k</p> <p>$n = 0$</p>					<p>add(5)</p> <p>0 1 2 3</p> <table><tr><td>5</td><td></td><td></td><td></td></tr></table> <p>k</p> <p>$n = 1$</p>	5				<p>add(3)</p> <p>0 1 2 3</p> <table><tr><td>5</td><td>3</td><td></td><td></td></tr></table> <p>k</p> <p>$n = 2$</p>	5	3			<p>rem() : 5</p> <p>0 1 2 3</p> <table><tr><td></td><td>3</td><td></td><td></td></tr></table> <p>k</p> <p>$n = 1$</p>		3		
5																			
5	3																		
	3																		
<p>rem() : 3</p> <p>0 1 2 3</p> <table><tr><td></td><td></td><td></td><td></td></tr></table> <p>k</p> <p>$n = 0$</p>					<p>add(7)</p> <p>0 1 2 3</p> <table><tr><td></td><td></td><td>7</td><td></td></tr></table> <p>k</p> <p>$n = 1$</p>			7		<p>add(2)</p> <p>0 1 2 3</p> <table><tr><td></td><td></td><td>7</td><td>2</td></tr></table> <p>k</p> <p>$n = 2$</p>			7	2	<p>add(4)</p> <p>0 1 2 3</p> <table><tr><td>4</td><td></td><td>7</td><td>2</td></tr></table> <p>k</p> <p>$n = 3$</p>	4		7	2
		7																	
		7	2																
4		7	2																

Sor műveletek

- Műveletidő: $\Theta(1)$
- Add művelet
 - $mT(n) \in \Theta(1)$
 - $MT(n) \in \Theta(n)$
 - $AT(n) \in \Theta(1)$
- `doubleFullQueueArray(Z, k)`
 - ha már tele van a tömb -> cserélje le nagyobbra, pontosan kétszer akkorára!

doubleFullQueueArray(&A:T[], n&k:N)

B:T[] := new T[2*A.length]

i := 0 to n-1

B[i] := A[(k+i) mod A.length]

delete A; A := B; k := 0

Queue::first() : T

$n > 0$

return Z[k]

QueueUnderflow

Queue::rem() : T

$n > 0$

$n --$

$i := k$

$k := (k + 1) \bmod Z.length$

return Z[i]

QueueUnderflow

Queue::add(x : T)

$n = Z.length$

doubleFullQueueArray(Z, k)

SKIP

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

$n ++$

Sor gyakorlati alkalmazása

- IO pufferek
- Ügyfélszolgálati programok (üzletek, bankok)
- Folyamat ütemezés
- Fák szintenkénti bejárása
- Szélességi keresés gráfokon
- Legrövidebb út keresése a legáltalánosabb esetben

Lineáris adatszerkezetek

Tömbök vs. Listák

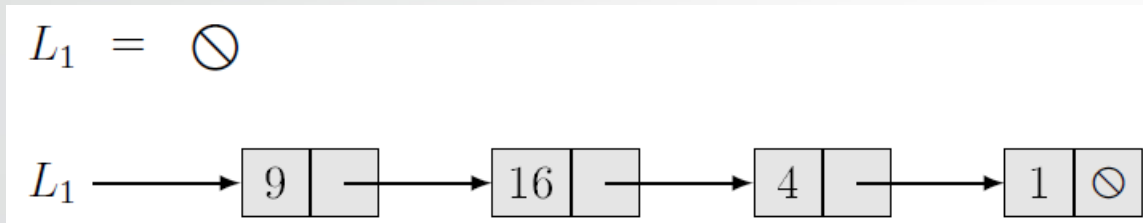
- $MT(\text{beszúrás / törlés adott pozícióra}): \Theta(n) \qquad \Theta(1)$
- $MT(\text{egy elemének elérése}): \qquad \Theta(1) \qquad \Theta(n)$

Láncolt listák fajtái

- Egyirányú – kétirányú
 - Mozgás: csak a lista elejétől vs vissza felé is (több memória, utasítás)
- Fejelemes- fejelem nélküli

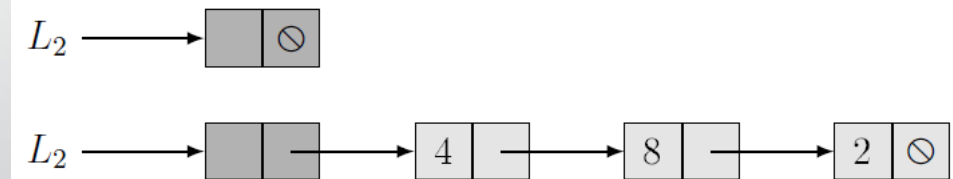
Egyirányú listák (one-way or singly linked lists)

- Egyszerű egyirányú listák (S1L = Simple one-way List)

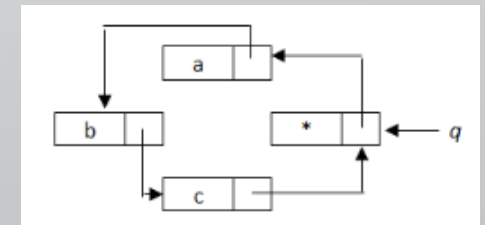


E1
$+key : \mathcal{T}$
\dots // satellite data may come here
$+next : \mathbf{E1}^*$
$+\mathbf{E1}() \{ next := \bigcirc \}$

- $\mathbf{E1}^*$ -pointer: E1 típusú objektum címét tartalmazhatja (vagy o)
 - $(*p).key$ / $p \rightarrow key$, $(*p).next$, $p \rightarrow next$
- Fejelemes egyirányú listák (H1L = Header node + one-way List)

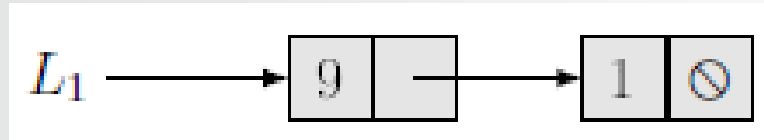



- Végelemes egyirányú listák
- Ciklikus egyirányú listák

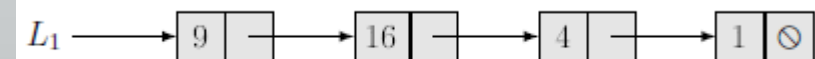


Egyszerű egyirányú listák (S1L)

- $L_1 \rightarrow \text{key}=9; L_1 \rightarrow \text{next} \rightarrow \text{key}=1; L_1 \rightarrow \text{next} \rightarrow \text{next} = \emptyset$



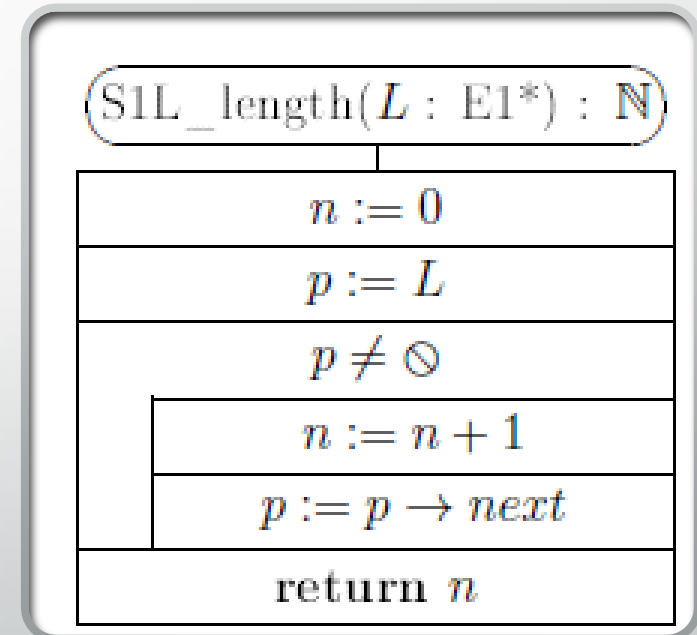
- $p = \emptyset$ 
 - $*p$ hibás
 - $p \rightarrow \text{next}; p \rightarrow \text{key}$ hibásak
 - futási hiba
- Ha p pointernek nem adunk értéket
 - $*p; p \rightarrow \text{next}; p \rightarrow \text{key}$ definiálatlanok



Példa: egyszerű egyirányú lista (S1L) hossza

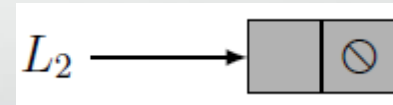
- $S1L_length(L:E1^*): \mathbb{N}$
 - L paraméter
 - absztrakt (azaz logikai) szinten: lista
 - $L:E1^*$: a függvény által elvégzett számítás helyességének *előfeltétele*
 - konkrét (azaz fizikai) szinten: memóriacím, ami a listát azonosítja
- Műveletigény:
 - n : az L lista hossza
 - a ciklus n iterációt végez

➤ $T_{S1L_length}(n) \in \Theta(n)$



Fejelemes listák (H1L)

- Mindig tartalmazznak egy fejelemet
 - A fejelemére mutató pointer azonosítja
 - Key mező: definiálatlan
 - Next pointere: a H1L-nek megfelelő S1L-t azonosítja
- Üres H1L:
 - van fejeleme, aminek a next pointere \emptyset
 - ($L_2 \rightarrow key$: definiálatlan; $L_2 \rightarrow next = \emptyset$)
- $H1L_length(H : E1^*) : \mathbb{N}$
 - $T_{H1L_length}(n) \in \Theta(n)$



$H1L_length(H : E1^*) : \mathbb{N}$
return $S1L_length(H \rightarrow next)$

Egyirányú listák további műveletei

- Fejelemes egyirányú listák (H1L)
 - alpműveletek segítségével minden összetett listamódosító művelet megadható
- Egyszerű egyirányú listák (S1L)
 - beszúrás a lista legelejére
 - első elem kifűzésére

// Insert $*q$ at the

// front of list L .

$q \rightarrow next := L$

$L := q$

// Unlink the first element of list L .

$q := L$

$L := q \rightarrow next$

$[q \rightarrow next := \emptyset]$

Egyirányú listák alapl műveletei

- Egyirányú listák kezelése
 - Listaelemek megfelelő átláncolása
 - Kerüljük a felesleges adatmozgatást
 - Járulékos adatok
- Listába szúrás
 - *p eleme után fűzi a *q objektumot
 - végrehajtása előtt *q nincs listába fűzve
 - $T \in \Theta(1)$
- Kifűzés listából
 - EF: *p és *q valamely egyirányú lista egymás utáni elemei,
 - $p \rightarrow next = q \neq \emptyset$
 - $T \in \Theta(1)$.
 - átfűzésnél a $q \rightarrow next := \emptyset$ elhagyható

// Let *p be followed by *q.

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

$p \rightarrow next := q$

// Provided that *p is followed

// by *q, unlink *q.

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

$[q \rightarrow next := \emptyset]$

S₁L és H₁L összehasonlítása

S₁L

- Sok rövid lista
 - Szignifikáns a tárigények különbsége
 - Fejelemek allokálása és deallokálása a futási időt jelentősen megnövelheti
 - (pl. hasító táblák, összefésüléses rendezés)
- Egy (rész)feladatban a listát mindig csak a legelején, azaz veremszerűen kell módosítani
- A lista első eleme biztosan a helyén marad

H₁L

- Kevesebb esetszétválasztás
 - Mindig valami után kell beszúrni
 - Mindig valami mögül kell kifűzni.
- Eggyel több objektumot tartalmaz
 - Megnöveli a program tárigényét
- Őrszem: fejelem, végelem
- Végelem: pl. sorok

$\text{cut}(L : E1^* ; n : \mathbb{N}) : E1^*$

- L S1L-t kettévágása
 - Az első n elemét hagyja L -ben
 - Visszaadja a lista levágott maradékát azonosító pointert
- A listaelemek sorrendjét megtartja

- $T_{\text{cut}}(n) \in \Theta(n)$

$\text{cut}(L : E1^* ; n : \mathbb{N}) : E1^*$

$p := L$

$n > 1$

$n := n - 1$

$p := p \rightarrow next$

$q := p \rightarrow next$

$p \rightarrow next := \emptyset$

return q

H1L_read() : E1*

- Feladata

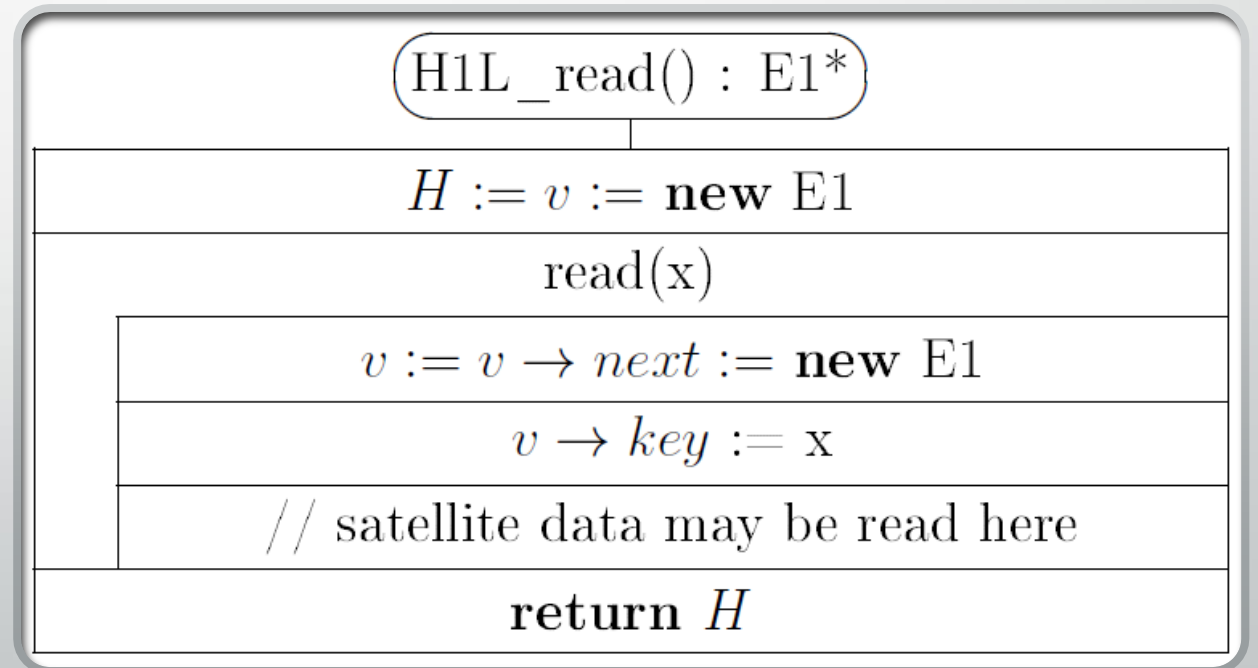
- beolvas egy adatsort a kurrens inputról
- bemenet sorrendje szerint egy H1L-t épít belőlük
- visszaadja a fejeleme címét

- $\text{read}(\&x: \mathcal{T}): \mathbb{B}$

- következő adat beolvasása x-be
- igaz: ha a beolvasás előtt még volt adat a bemeneten
- hamis: különben (x definiálatlan marad)

- Műveletigény

- $T_{H1L_read}(n) \in \Theta(n)$



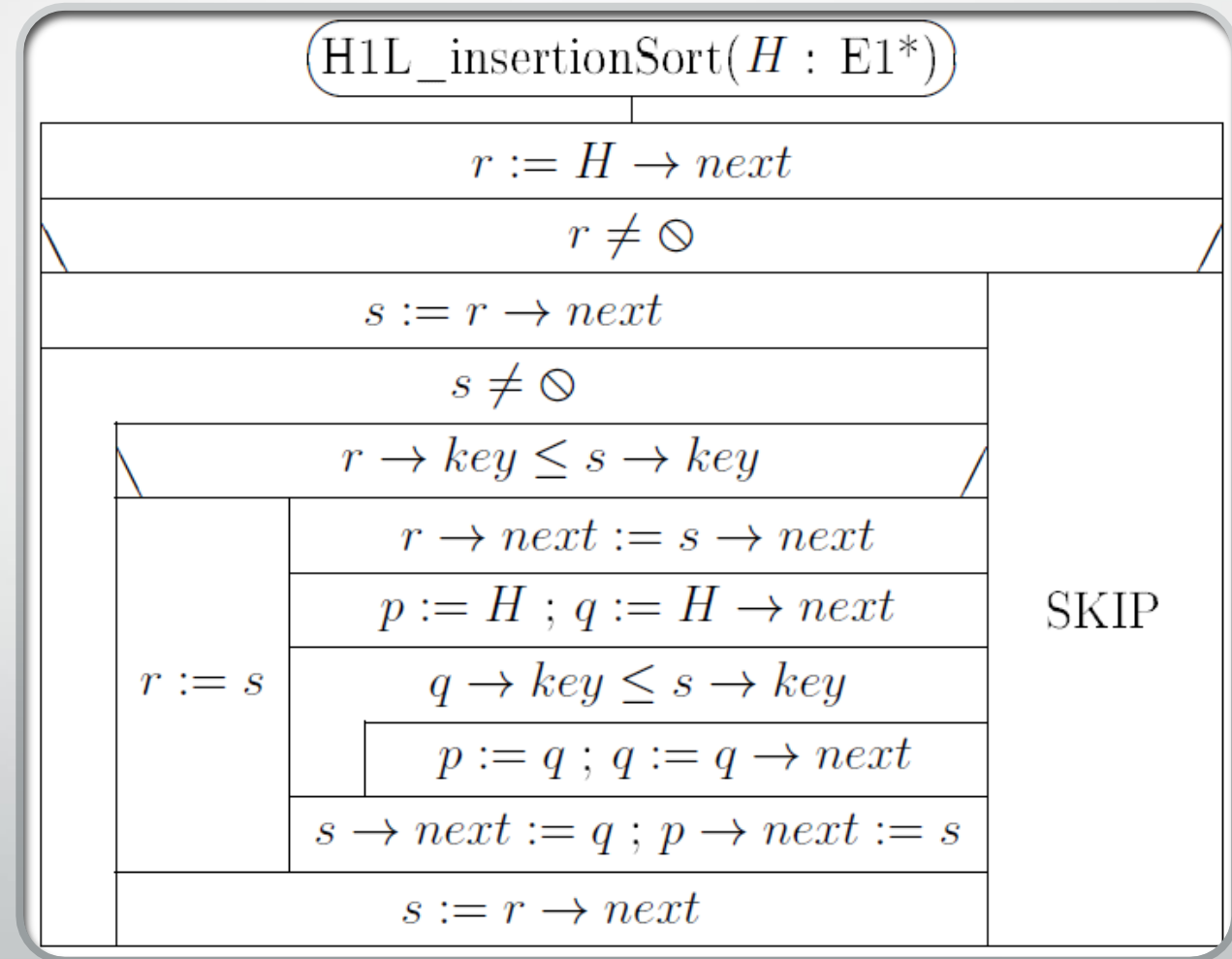
Dinamikus memóriagazdálkodás

- objektumok dinamikus létrehozása: `new T`
 - T típusú objektumot hoz létre, és visszaadja a címét
 - helyfoglalás a memóriában
- `p` mutató
 - a `p := new T` utasítás végrehajtása előtt is létezik
 - azt a mutató deklarációjának kiértékelése hozza létre
- objektumok dinamikus törlése: `delete p`
 - p mutató által hivatkozott objektumot törli
 - memória felszabadítása
- `delete p` végrehajtása után is létezik
 - egészen az őt (automatikusan) deklaráló eljárás vagy függvény végrehajtásának befejezéséig
- Az absztrakt programokban az objektumokat dinamikusan létrehozó (`new`) és törlő (`delete`) utasítások műveletigényeit konstans értékeknek, azaz $\Theta(1)$ -nek vesszük,
 - Valójában nem tudjuk, mennyi.
 - A lehető legkevesebbet használjuk a `new` és a `delete` utasításokat.

Beszúró rendezés H1L listákra

$$mT_{IS}(n) \in \Theta(n)$$

$$AT_{IS}(n), MT_{IS}(n) \in \Theta(n^2)$$




Összefésüléses rendezés S1L-re

$\text{mergeSort}(\&L : E1^*)$
// L is an S1L.
$n := \text{S1L_length}(L)$
$\text{ms}(L, n)$

$\text{ms}(\&L : E1^* ; n : \mathbb{N})$
$n > 1$
$n1 := \lfloor \frac{n}{2} \rfloor$
$L2 := \text{cut}(L, n1)$
$\text{ms}(L, n1)$
$\text{ms}(L2, n - n1)$
$L := \text{merge}(L, L2)$
SKIP

$\text{merge}(L1, L2 : E1^*) : E1^*$
$L1 \rightarrow \text{key} \leq L2 \rightarrow \text{key}$
$L := t := L1$
$L := t := L2$
$L1 := L1 \rightarrow \text{next}$
$L2 := L2 \rightarrow \text{next}$
$L1 \neq \emptyset \wedge L2 \neq \emptyset$
$L1 \rightarrow \text{key} \leq L2 \rightarrow \text{key}$
$t := t \rightarrow \text{next} := L1$
$t := t \rightarrow \text{next} := L2$
$L1 := L1 \rightarrow \text{next}$
$L2 := L2 \rightarrow \text{next}$
$L1 \neq \emptyset$
$t \rightarrow \text{next} := L1$
$t \rightarrow \text{next} := L2$
return L

$$mT_{MS}(n), MT_{MS}(n) \in \Theta(n \log n)$$



Köszönöm a figyelmet!

Pusztai Kinga

A bemutató Ásványi Tibor: Algoritmusok és adatszerkezetek I.
Előadásjegyzete alapján készült.