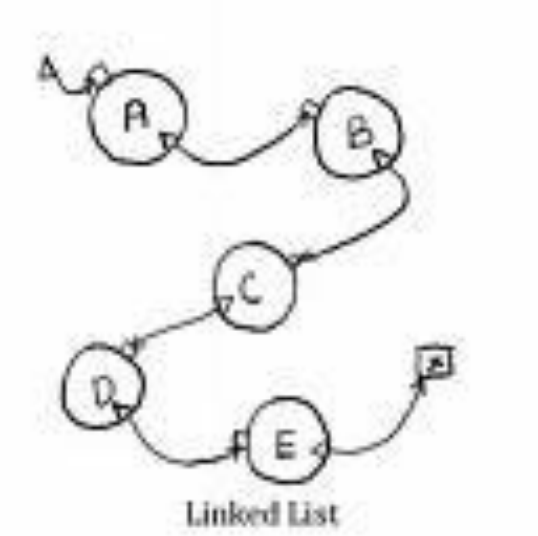


Linked List

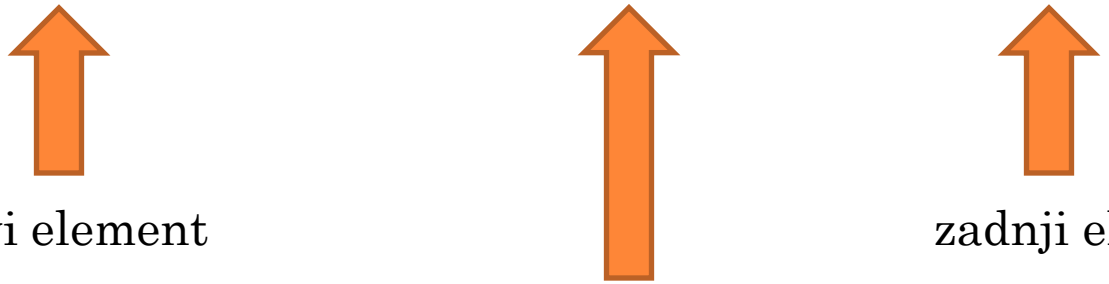
SEZNAM
(List)

SEZNAM

Seznam je zaporedje 0 ali več elementov, pri čemer velja:

- vrstni red elementov je pomemben
- elementi v seznamu se lahko ponavljajo

$$a_1, a_2, a_3, \ldots\ldots, a_i, \ldots\ldots\ldots, a_n$$



prvi element

indeksu i pravimo položaj elementa v seznamu

zadnji element

ADT LIST

- MAKENULL(L) – naredi prazen seznam L
- FIRST(L) – vrne položaj prvega elementa v seznamu
- LAST(L) – vrne položaj zadnjega elementa v seznamu
- NEXT(p, L) – vrne naslednji položaj položaja p
- PREVIOUS(p, L) – vrne predhodni položaj položaja p
- RETRIEVE(p, L) – vrne element a_p na položaju p
- INSERT(x, p, L) – vstavi element x na položaj p
- INSERT(x, L) – vstavi element x na poljuben položaj
- DELETE(p, L) – zbriše element a_p na položaju p
- EMPTY(L) – preveri, če je seznam prazen
- END(L) – vrne položaj, ki sledi zadnjemu elementu seznama
- OVEREND(p, L) – preveri, če je p = END(L)
- LOCATE(x, L) – poišče položaj elementa x v seznamu
- PRINTLIST(L) – po vrsti izpiše vse elemente seznama

ADT LIST

Implementacije ADT LIST:

- enosmerni seznam s kazalci
- dvosmerni seznam s kazalci
- s poljem
- z indeksnimi kazalci (kurzorji)

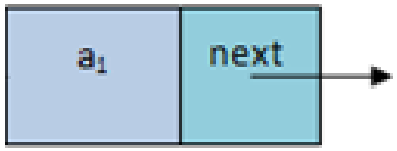
- Implementacije se razlikujejo po časovni in prostorski zahtevnosti posameznih operacij.

- Imajo poudarek na različnih lastnostih seznama in na različnih podmnožicah operacij.

ENOSMERNI SEZNAM S KAZALCI

Podatkovna struktura je podana z enim elementom seznama:

```
class ListLinkedNode {
   Object element;
   ListLinkedNode next;
   …
}
```

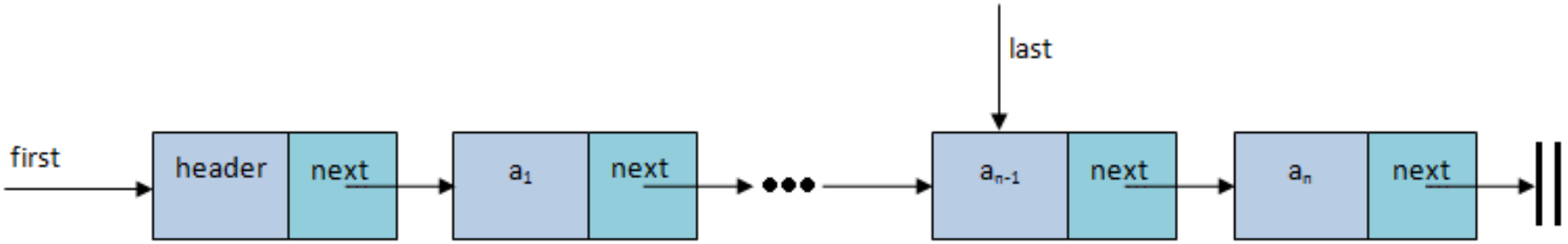


ter s samim seznamom elementov:

```
public class ListLinked {
   protected ListLinkedNode first, last;
   …
}
```

Seznam je definiran s položajem prvega in zadnjega elementa.

Enosmerni seznam s kazalci



Položaj elementa zamaknjen!

Zaradi učinkovite implementacije vstavljanja in brisanja je položaj elementa seznama podan s kazalcem na celico, ki vsebuje kazalec (next) na celico z elementom.

Enosmerni seznam s kazalci

Lastnosti enosmernega seznama:

- učinkovito vstavljanje elementa na znan položaj – **O(1)**,
- učinkovito brisanje elementa na znanem položaju – **O(1)** ,
  (razen v primeru brisanja zadnjega elementa, ko je O(n))
- seznam zaseda samo toliko pomnilnika, kolikor ga zares potrebuje,
- počasna operacija iskanja predhodnika v seznamu – **O(n)**,
- zaporedno premikanje po seznamu v eno smer.

Dvosmerni seznam s kazalci

Vsaka celica seznama ima poleg elementa še dva kazalca:

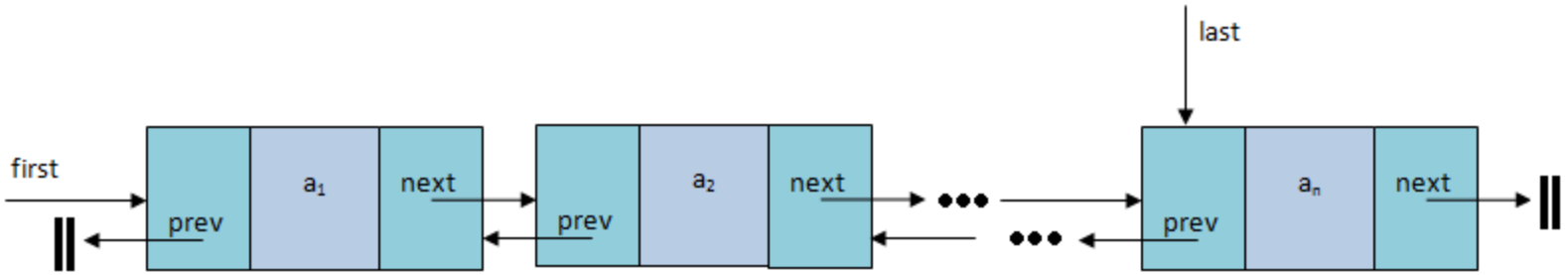
- kazalec na naslednji element
- kazalec na predhodni element

Omogoča učinkovito iskanje predhodnika v seznamu – **O(1)**

```
class ListTwoWayLinkedNode {
   Object element;
   ListTwoWayLinkedNode next, previous;
   …
}
```


```
public class ListTwoWayLinked {
   protected ListTwoWayLinkedNode first, last;
   …
}
```

Dvosmerni seznam s kazalci



V dvosmernem seznamu je položaj elementa podan s kazalcem na celico, ki ta element vsebuje.

DVOSMERNI SEZNAM S KAZALCI

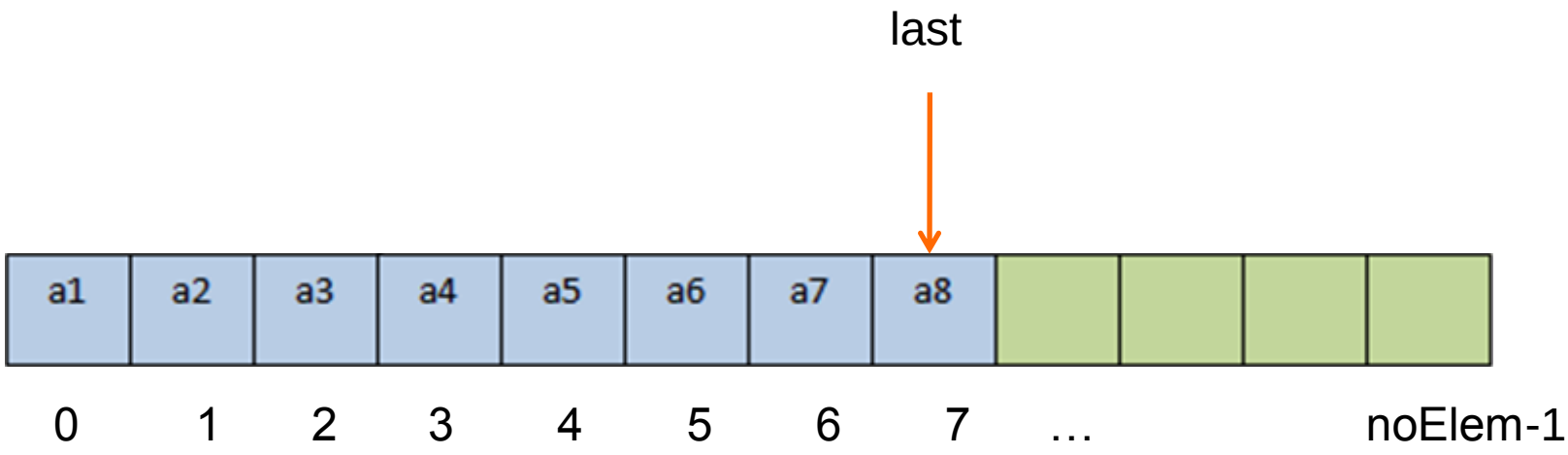
MAKENULL(L)	O(1)
FIRST(L)	O(1)
LAST(L)	O(1)
NEXT(p, L)	O(1)
PREVIOUS(p, L)	O(1)
RETRIEVE(p, L)	O(1)
INSERT(x, p, L)	O(1)
INSERT(x, L)	O(1)
DELETE(p, L)	O(1)
EMPTY(L)	O(1)
END(L)	O(1)
OVEREND(p, L)	O(1)
LOCATE(x, L)	O(n)
PRINTLIST(L)	O(n)

IMPLEMENTACIJA SEZNAMA S POLJEM

- uporablja podatkovno strukturo polje za shranjevanje elementov seznama
- položaj elementa v seznamu je podan z indeksom polja
- potrebujemo še indeks zadnjega elementa v seznamu

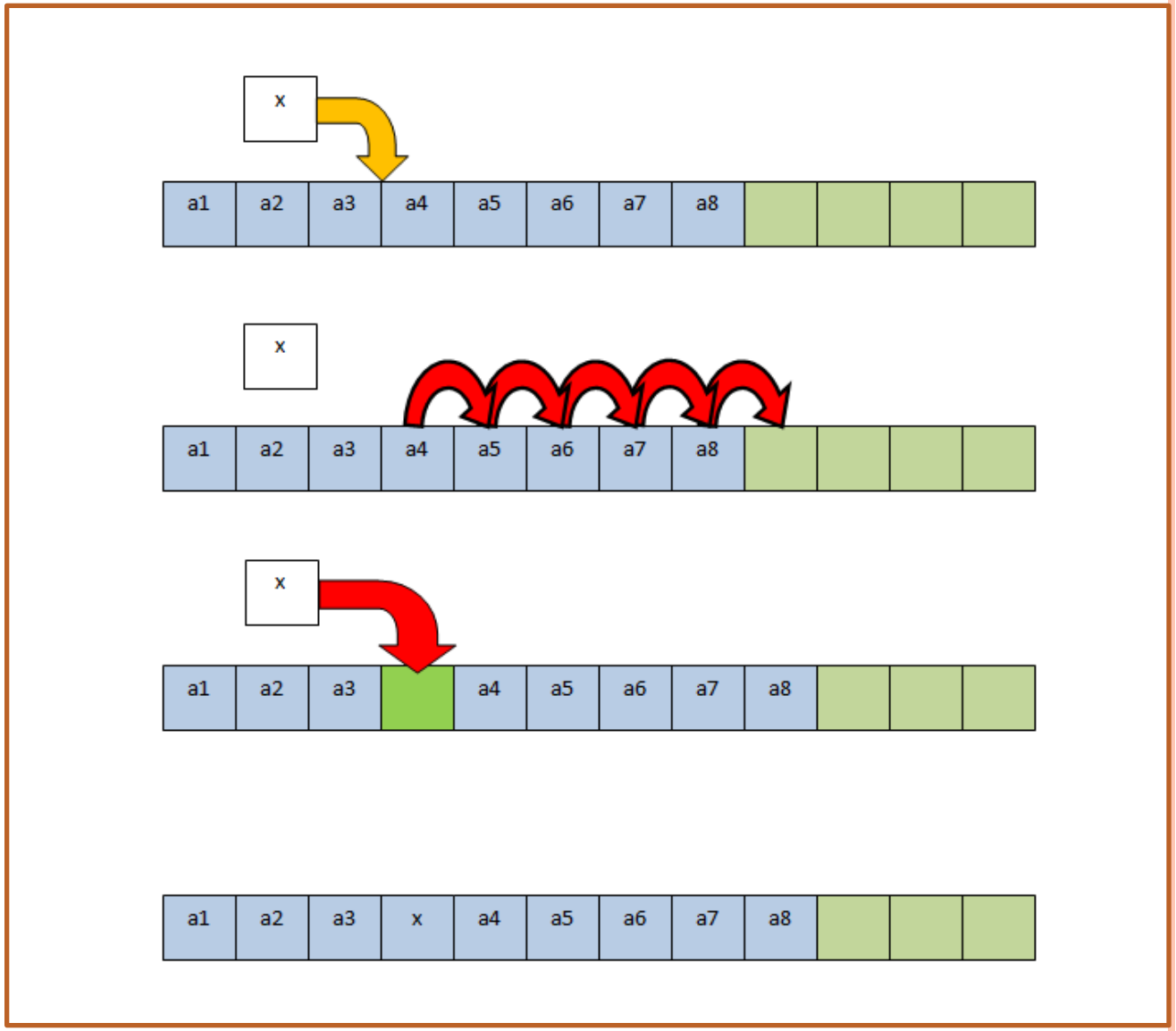
```
public class ListArray {
   private Object elements[];
   private int lastElement;
   …
   public ListArray(int noElem) {
        elements = new Object[noElem];
        …
   }
   …
}
```

IMPLEMENTACIJA SEZNAMA S POLJEM



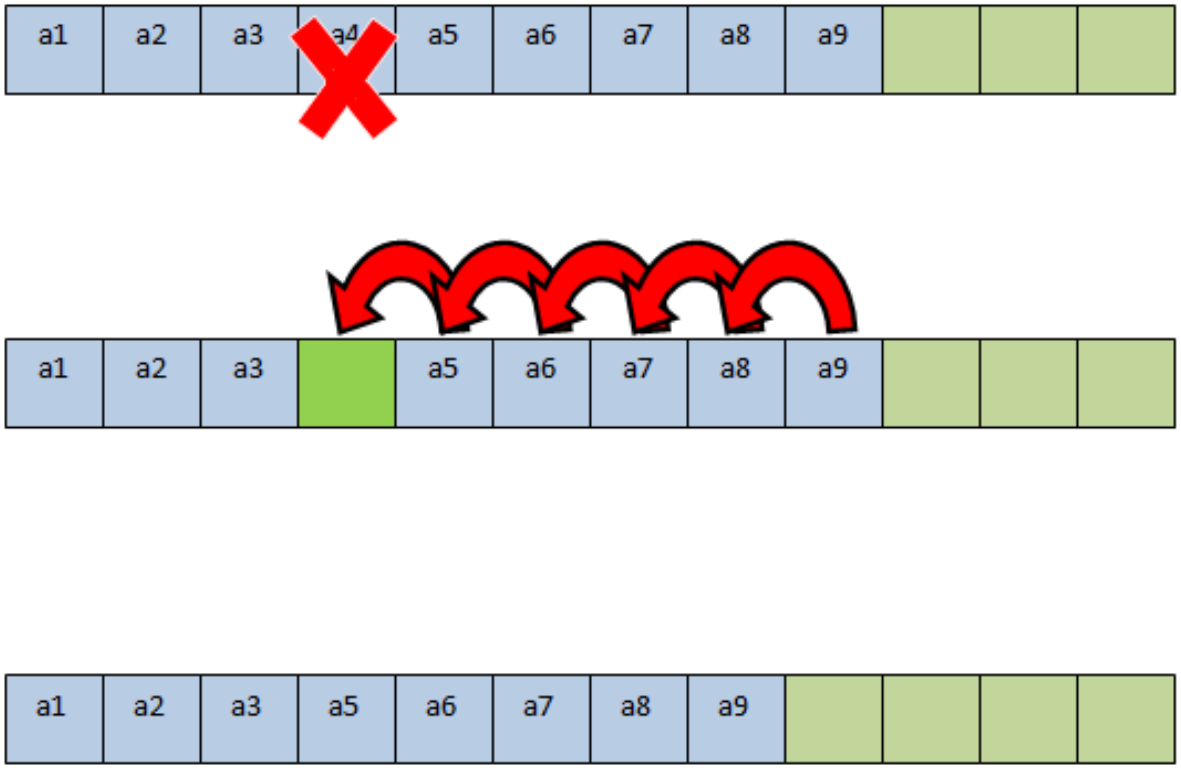
IMPLEMENTACIJA SEZNAMA S POLJEM

Vstavljanje



IMPLEMENTACIJA SEZNAMA S POLJEM

Brisanje



IMPLEMENTACIJA SEZNAMA S POLJEM

MAKENULL(L)	O(1)
FIRST(L)	O(1)
LAST(L)	O(1)
NEXT(p, L)	O(1)
PREVIOUS(p, L)	O(1)
RETRIEVE(p, L)	O(1)
INSERT(x, p, L)	O(n)
INSERT(x, L)	O(1)
DELETE(p, L)	O(n)
EMPTY(L)	O(1)
END(L)	O(1)
OVEREND(p, L)	O(1)
LOCATE(x, L)	O(n)
PRINTLIST(L)	O(n)

IMPLEMENTACIJA SEZNAMA S POLJEM

Prednosti:

- preprosta implementacija
- časovna zahtevnost mnogih operacij je **O(1),** vključno z dostopom do poljubnega elementa seznama

Slabosti:

- neučinkovito vstavljanje elementa na določen položaj - **O(n)**
- neučinkovito brisanje elementa – **O(n)**
- omejena dolžina seznama
- ves čas polje zaseda maksimalno količino pomnilnika

Primerna implementacija za aplikacije, kjer se seznam zelo malo spreminja.
Učinkovito, če sprostimo zahtevo po vrstnem redu elementov…

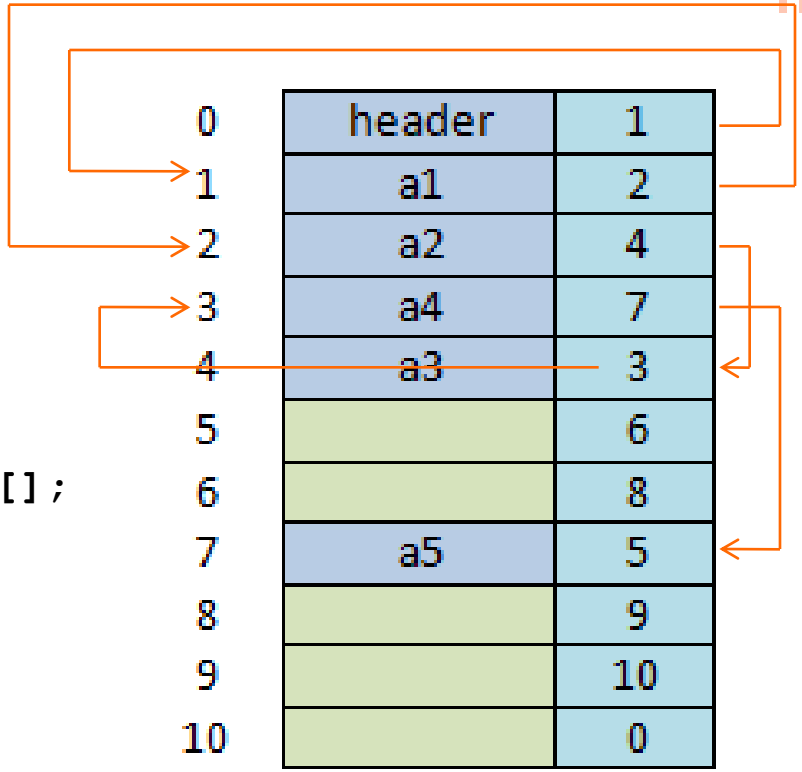
Seznam z indeksnimi kazalci

Uporaba v jezikih, ki ne omogočajo dinamičnih podatkovnih struktur.

Vsaka celica polja je sestavljena iz elementa in iz indeksa naslednjega elementa.

```
class ListCursorNode {
   Object element;
   int next;
   …
}

public class ListCursor {
   private ListCursorNode cells[];
   private int last;
   …
}
```



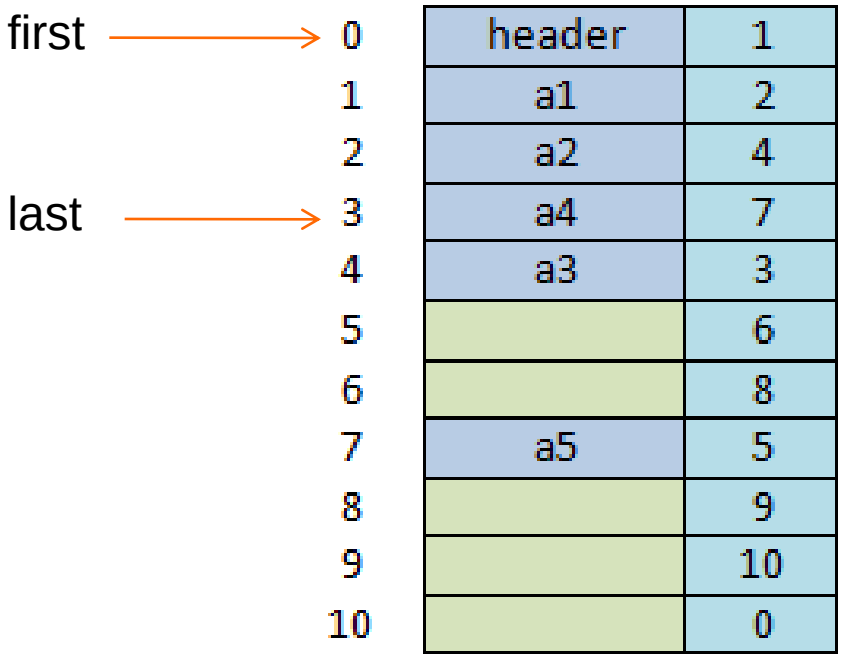
Seznam z indeksnimi kazalci

Definicije operacij so podobne operacijam s kazalci, le da programer sam skrbi za dodeljevanje in "čiščenje" pomnilnika.

Položaj je ZAMAKNJEN!

Dva seznama:

- prvi seznam vsebuje elemente dejanskega seznama
- drugi seznam povezuje vse prazne celice v polju



Seznam z indeksnimi kazalci

MAKENULL(L)	O(cells.length)
FIRST(L)	O(1)
LAST(L)	O(1)
NEXT(p, L)	O(1)
PREVIOUS(p, L)	O(n)
RETRIEVE(p, L)	O(1)
INSERT(x, p, L)	O(1)
INSERT(x, L)	O(1)
DELETE(p, L)	O(1) … O(n)
EMPTY(L)	O(1)
END(L)	O(1)
OVEREND(p, L)	O(1)
LOCATE(x, L)	O(n)
PRINTLIST(L)	O(n)

Seznam z indeksnimi kazalci

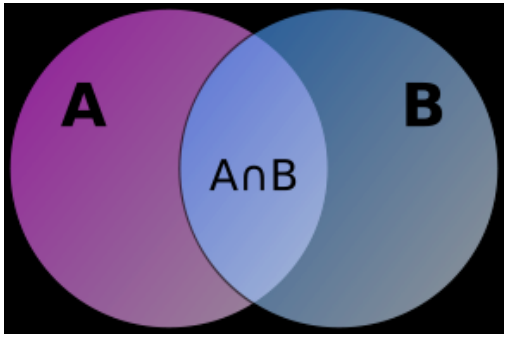
Lastnosti enosmernega seznama z indeksnimi kazalci podobne kot pri enosmernem seznamu s kazalci

- učinkovito vstavljanje elementa na znan položaj – **O(1)**,
- učinkovito brisanje elementa na znanem položaju – **O(1)** , (razen v primeru brisanja zadnjega elementa, ko je O(n))
- počasna operacija iskanja predhodnika v seznamu – **O(n)**,
- zaporedno premikanje po seznamu v eno smer.

Slabosti:

- programer mora sam skrbeti za dodeljevanje in sproščanje pomnilnika,
- seznam ves čas zaseda maksimalno količino pomnilnika,
- maksimalna velikost seznama mora biti podana vnaprej,
- počasna inicializacija – **O(n)**

Lahko implementiramo tudi *dvosmeren seznam* z indeksnimi kazalci.



MNOŽICA
(*set*)

ADT SET

Seznam je zaporedje 0 ali več elementov, pri čemer velja:

- vrstni red elementov je pomemben
- elementi v seznamu se lahko ponavljajo

- MNOŽICA (angl. *set*) – zbirka elementov, kjer vrstni red ni pomemben in elementi se ne ponavljajo

Implementacija je lahko izpeljana iz enosmernega seznama s kazalci:

- ohranimo kazalec last, čeprav ga v principu ne potrebujemo
- zaradi kazalca last je brisanje zadnjega elementa neučinkovito – O(n)
- ohranimo pojem položaja elementa, čeprav vrstni red ni pomemben
- pri dodajanju elementa moramo paziti, da elemente ne podvajamo – O(n)

ADT SET

- MAKENULL(S) – naredi prazno množico S
- FIRST(S) – vrne položaj prvega elementa v množici S
- NEXT(p, S) – vrne naslednji položaj položaja p
- RETRIEVE(p, S) – vrne element a_p na položaju p
- INSERT(x, S) – vstavi element x v množico S brez podvajanja
- DELETE(p, S) – zbriše element a_p na položaju p
- EMPTY(S) – preveri, če je množica prazna
- OVEREND(p, S) – preveri, če je p položaj, ki sledi zadnjemu elementu množice
- LOCATE(x, S) – poišče položaj elementa x v množici S
- PRINTSET(S) – po vrsti izpiše vse elemente množice S

- UNION(S1,S) – v množico S doda (brez podvajanja) vse elemente iz množice S1
- INTERSECTION(S1, S) – iz množice S zbriše vse elemente, ki se ne nahajajo tudi v S1

ADT SET

n – moč množice S
m – moč množice S1

- UNION(S1,S) – v množico S doda (brez podvajanja) vse elemente iz množice S1

Eno dodajanje elementa v S brez podvajanja: O(n)
m dodajanj elementa v S brez podvajanja: m X O(n) = O(mn)

- INTERSECTION(S1, S) – iz množice S zbriše vse elemente, ki se ne nahajajo tudi v S1

Eno brisanje elementa: element poiščemo O(n) plus zbrišemo O(1)…O(n)
m brisanj elementa: m krat iskanje plus min(m,n) krat brisanje:
m X O(n) + min(m,n) X [O(1) … O(n)] = O(mn) …O((m+n)n)

ADT SET

MAKENULL(S)	O(1)
FIRST(S)	O(1)
NEXT(p, S)	O(1)
RETRIEVE(p, S)	O(1)
INSERT(x, S)	O(n)
DELETE(p, S)	O(1) … O(n)
EMPTY(S)	O(1)
OVEREND(p, S)	O(1)
LOCATE(x, S)	O(n)
PRINTSET(S)	O(n)
UNION(S1,S)	O(mn)
INTERSECTION(S1,S)	O(mn) … O((m+n)n)

Implementacija množic

Množico lahko implementiramo

- s povezanim seznamom:
  - INSERT(x, S) ima časovno zahtevnost reda O(n)
  - LOCATE(x, S) ima časovno zahtevnost reda O(n)
  - UNION(S1, S) in INTERSECTION(S1, S): O(mn).
  - Neučinkovite operacije, ki so vezane na urejenost elementov.
- z zgoščeno tabelo:
  - učinkoviti operaciji INSERT(x, S) in LOCATE(x, S): O(1).
  - UNION(S1, S) ima časovno zahtevnost O(m).
  - INTERSECTION(S1, S) ima časovno zahtevnost O(n).
  - Neučinkovite operacije, ki so vezane na urejenost elementov.

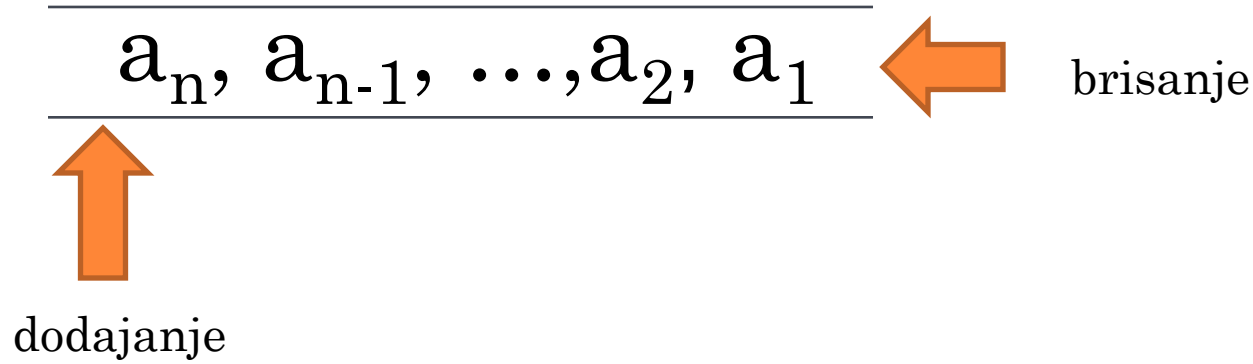
  Časovne zahtevnosti veljajo pod pogojem, da zgoščevalna funkcija enakomerno razprši elemente.
- z iskalnim drevesom:
  - Učinkovita INSERT(x, S) in LOCATE(x, S): O(log n)
  - UNION(S1, S) ima časovno zahtevnost O(m log n)
  - INTERSECTION(S1, S): O(m log n)
  - Hitre operacije, ki so vezane na urejenost elementov.

Vrsta
(Queue)

ABSTRAKTNI PODATKOVNI TIP VRSTA

Vrsta (queue) je zbirka elementov, kjer elemente vedno:

- dodajamo na konec vrste
- brišemo na začetku vrste



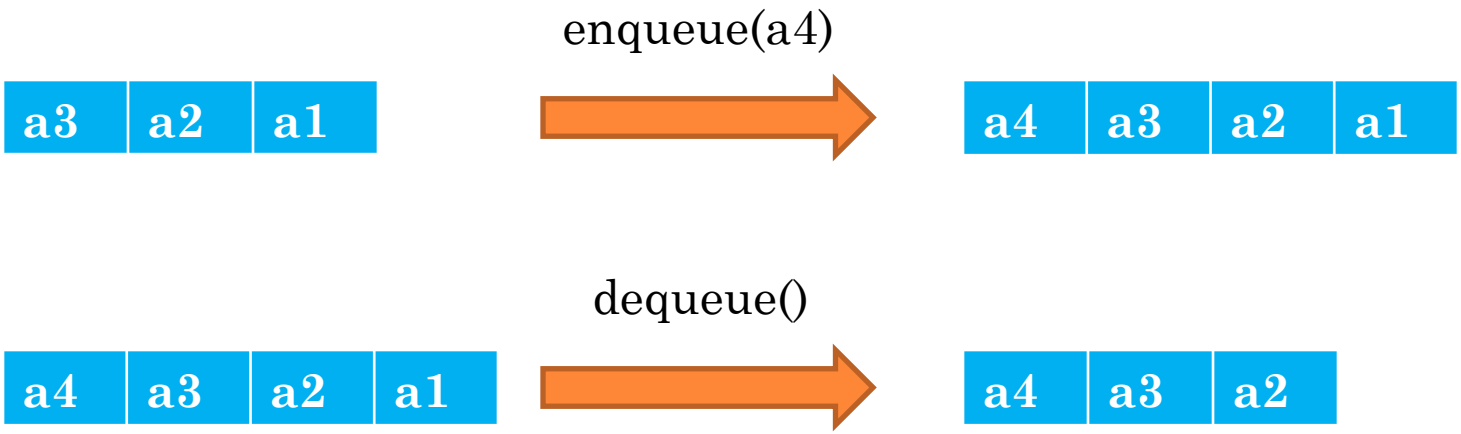
Vrsti pravimo tudi FIFO (first-in-first-out).

ADT QUEUE

Operacije, definirane za ADT QUEUE:

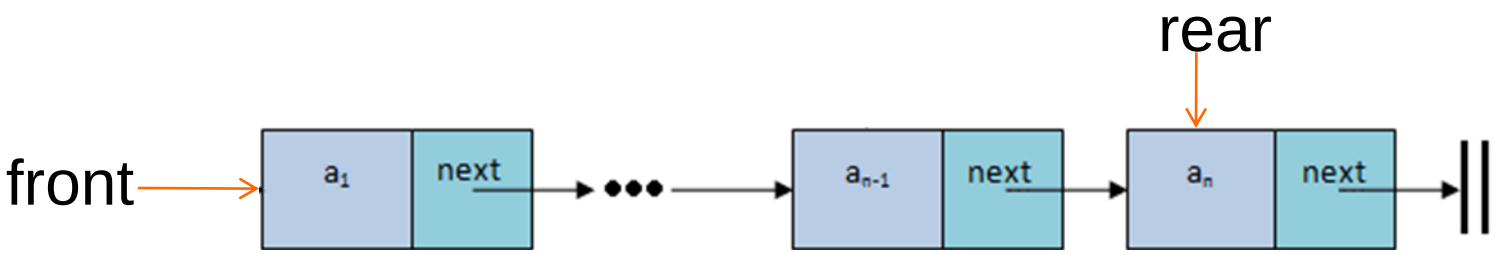
- MAKENULL(Q) – naredi prazno vrsto
- EMPTY(Q) – ali je vrsta prazna
- FRONT(Q) – vrne prvi element v vrsti
- ENQUEUE(x, Q)- vstavi element x na konec vrste
- DEQUEUE(Q) – zbriše prvi element iz vrste

ABSTRAKTNI PODATKOVNI TIP VRSTA



ADT QUEUE

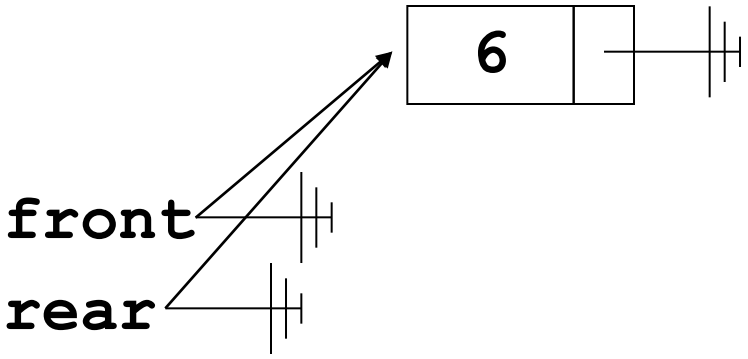
Vrsto lahko učinkovito implementiramo z enosmernim seznamom s kazalci. Položaj NI zamaknjen.



Brišemo na začetku seznama: O(1)
Dodajamo na koncu seznama: O(1)

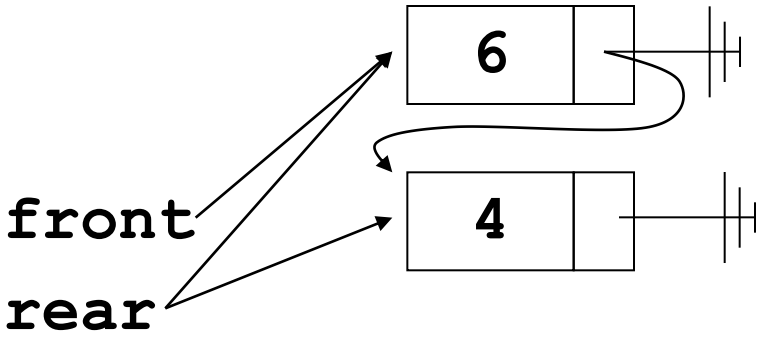
MAKENULL(Q)	O(1)
EMPTY(Q)	O(1)
FRONT(Q)	O(1)
ENQUEUE(x, Q)	O(1)
DEQUEUE(Q)	O(1)

PRIMER



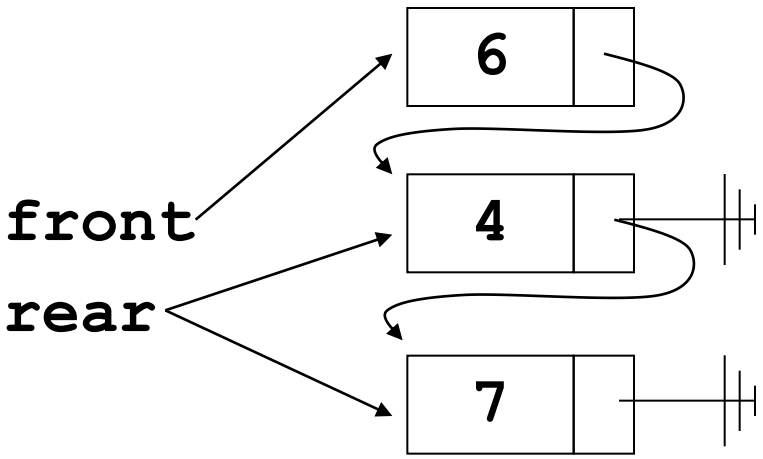

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
```

PRIMER



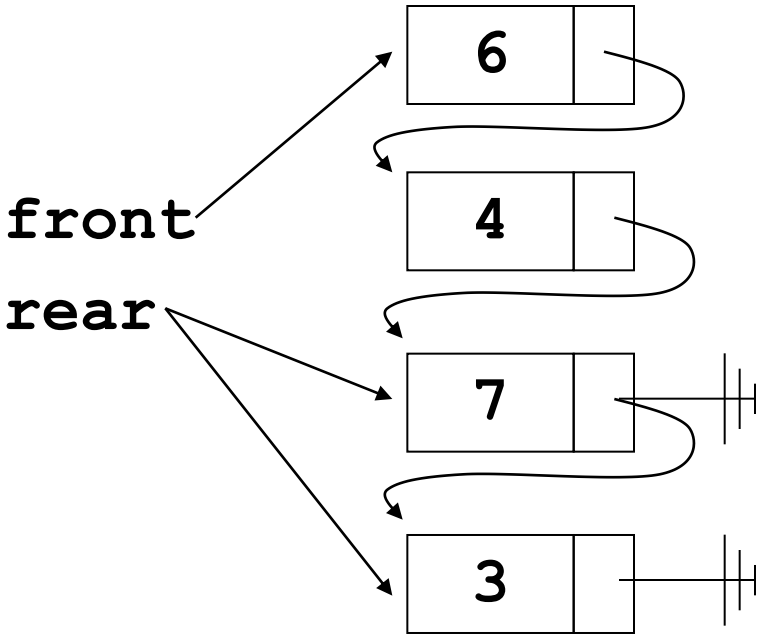

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
```

PRIMER



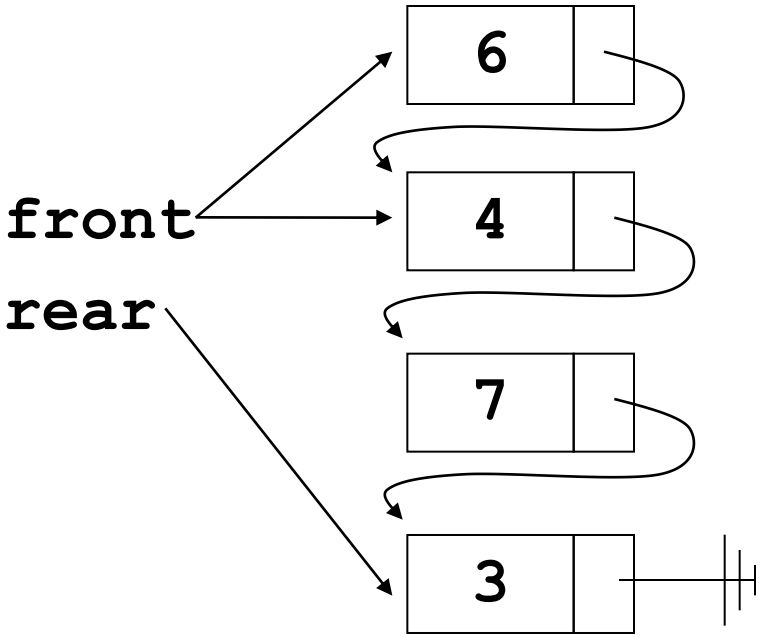

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
```

PRIMER




```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
```

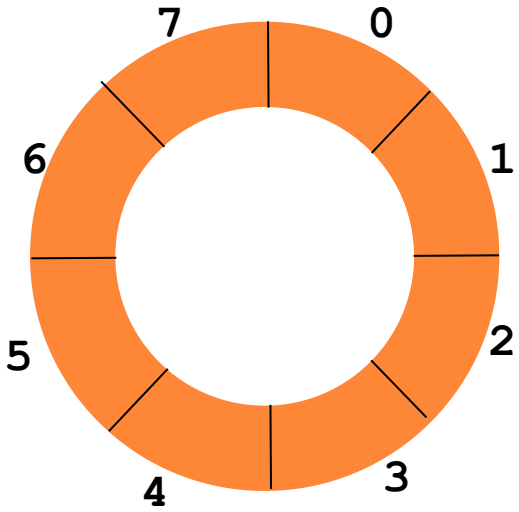
PRIMER




```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.dequeue();
```

ADT QUEUE

Implementacija s t.i. krožnim poljem (circular array): vrsta se krožno premika po polju.



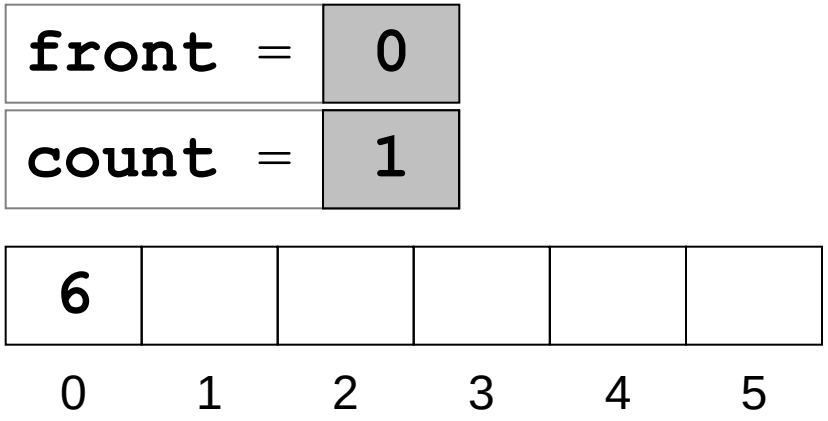
Pomagamo si z operatorjem ostanka pri deljenju:

```
1%5 = 1, 2%5 = 2, 5%5 = 0, 8%5 = 3
```

Premik indeksov front in rear (oziroma števca count):

```
front = (front + 1) % items.length;
rear =(rear + 1) % items.length;
rear =(front + count - 1) % items.length
```

Primer




```
//Java Code
Queue q = new Queue();
q.enqueue(6);
```

dodaj 6 pod indeks: (front + count) % items.length

PRIMER

front =	0
count =	5

6	4	7	3	8	
0	1	2	3	4	5

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
```

PRIMER

front =	2
count =	4

6	4	7	3	8	9
0	1	2	3	4	5

front = (0 + 1) % 6 = 1

front = (1 + 1) % 6 = 2

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
q.dequeue();//front = 1
q.dequeue();//front = 2
q.enqueue(9);
```

PRIMER

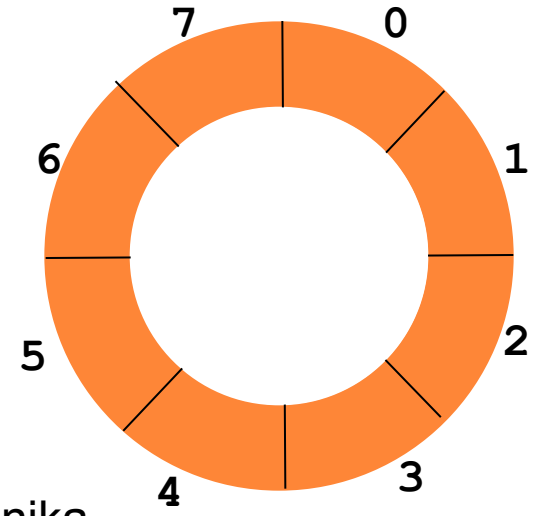
front =	2
count =	5

5		7	3	8	9
0	1	2	3	4	5

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
q.dequeue();//front = 1
q.dequeue();//front = 2
q.enqueue(9);
q.enqueue(5);
```

dodaj 5 pod: (front + count) % 6 = (2 + 4) % 6 = 0

ADT QUEUE



Slabosti:
- velikost vrste je navzgor omejena
- ves čas zaseda maksimalno pomnilnika

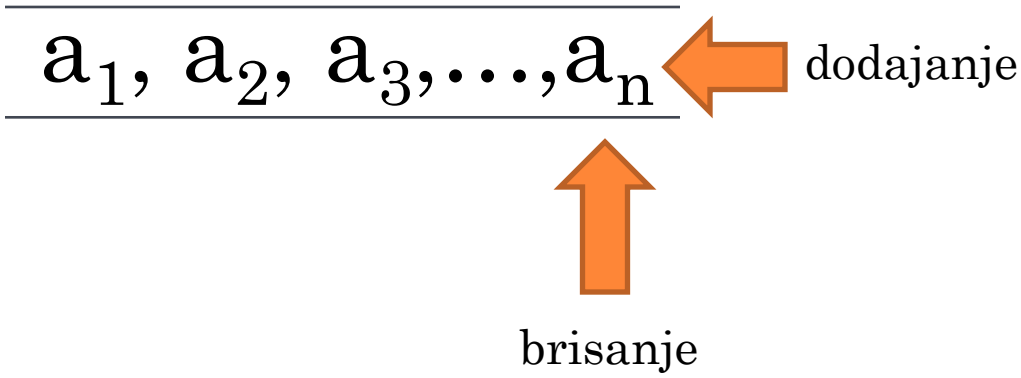
MAKENULL(Q)	O(1)
EMPTY(Q)	O(1)
FRONT(Q)	O(1)
ENQUEUE(x, Q)	O(1)
DEQUEUE(Q)	O(1)

Sklad
(Stack)

ABSTRAKTNI PODATKOVNI TIP SKLAD

Sklad (stack) je zbirka elementov, kjer elemente vedno:

- dodajamo na vrh sklada
- brišemo z vrha sklada



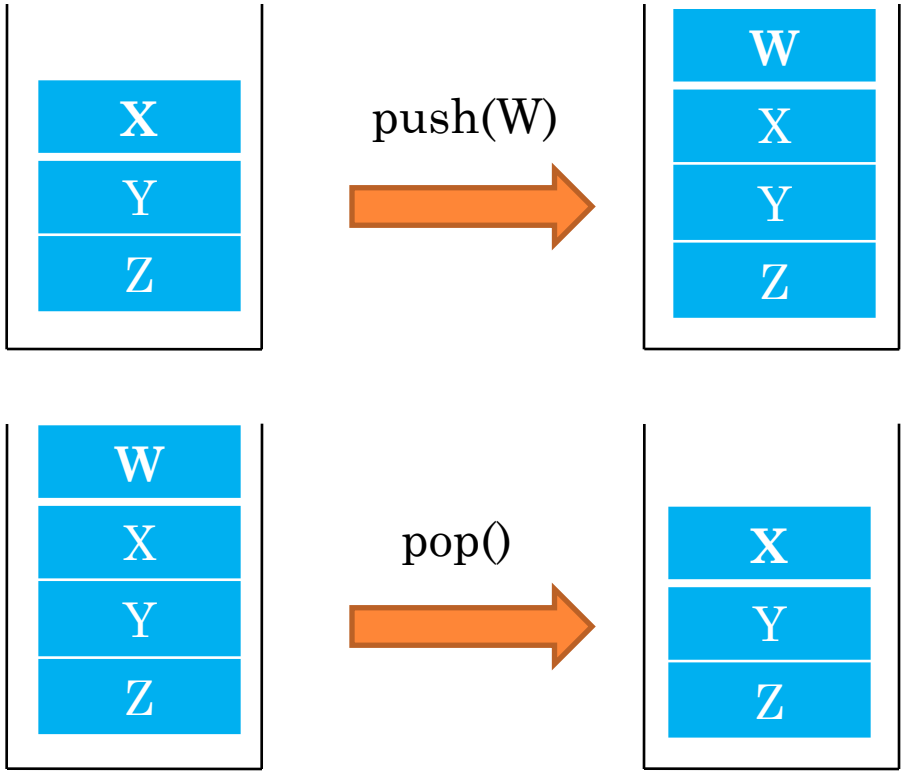
Skladu pravimo tudi LIFO (last-in-first-out).

ADT STACK

Operacije, definirane za ADT STACK:

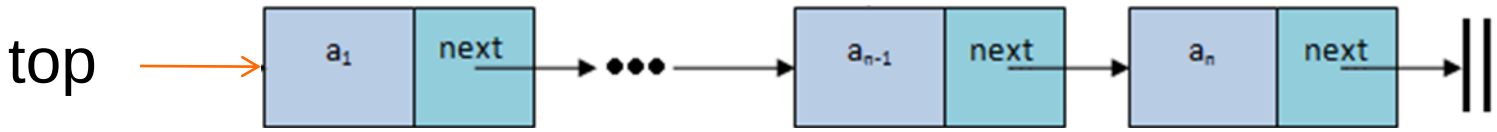
- MAKENULL(S) – naredi prazen sklad
- EMPTY(S) – ali je sklad prazen
- TOP(S) – vrne vrhnji element sklada
- PUSH(x, S)- vstavi element x na vrh sklada
- POP(S) – zbriše vrhnji element sklada

ADT STACK



ADT STACK

Sklad lahko učinkovito implementiramo z enosmernim seznamom s kazalci. Položaj NI zamaknjen, vrh je začetek.



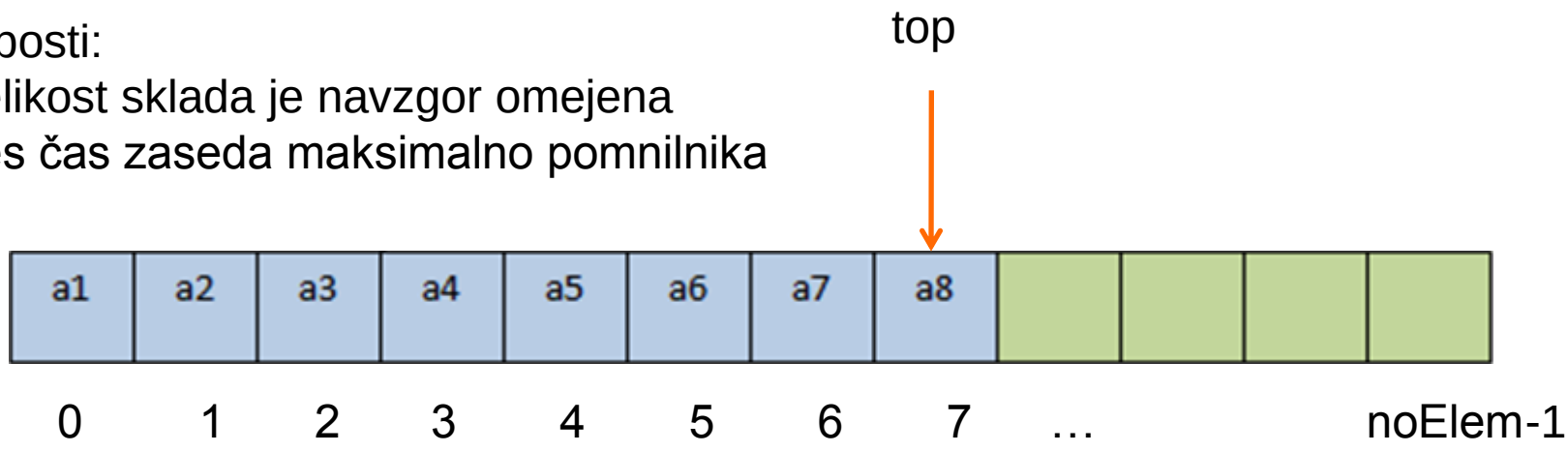
Brišemo na začetku seznama: O(1)
Dodajamo na začetku seznama: O(1)

MAKENULL(S)	O(1)
EMPTY(S)	O(1)
TOP(S)	O(1)
PUSH(x, S)	O(1)
POP(Q)	O(1)

IMPLEMENTACIJA SKLADA S POLJEM

Dodajamo in brišemo na koncu (vrhu) sklada – vedno O(1).

Slabosti:
- velikost sklada je navzgor omejena
- ves čas zaseda maksimalno pomnilnika



MAKENULL(S)	O(1)
EMPTY(S)	O(1)
TOP(S)	O(1)
PUSH(x, S)	O(1)
POP(Q)	O(1)

Abstraktni podatkovni tipi

Osnovni abstraktni podatkovni tipi, ki jih potrebujemo za razvoj algoritmov so:

- **seznam (list)** – zbirka elementov, ki se lahko *ponavljajo*; vrstni red elementov je *pomemben*
- **množica (set)** – zbirka elementov, kjer vrstni red *ni pomemben*; elementi se *ne ponavljajo*
- **vrsta (queue)** – zbirka, kjer elemente vedno dodajamo na konec vrste in jih vedno brišemo na začetku vrste
- **sklad (stack)** – zbirka, kjer se elementi dodajajo in brišejo vedno na vrhu sklada
- **preslikava (map)** – vsakemu elementu *d* iz domene priredi ustrezen element *r* iz zaloge vrednosti