Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

Povezani seznami

Jurij Mihelič, UniLj, FRI

Kazalčne podatkovne strukture

- Primitivni tipi
  - Java: byte, short, int, long, float, double, boolean, char
  - ne delijo stanja

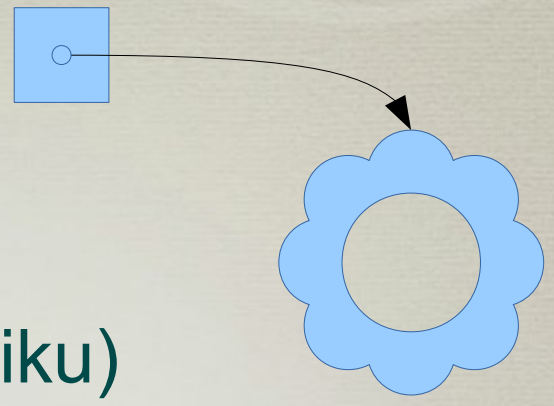
```
int a = 5;
int b = a;
b++;
System.out.println(a);
System.out.println(b);
```

- Referenčni tipi
  - Java: Integer, String, Data, ArrayList, polja (arrays), ...
  - delijo stanje

```
Date a = new Date();
Date b = a;
b.setYear(2042-1900);
System.out.println(a);
System.out.println(b);
```

```
int[] a = {1, 2, 3};
int[] b = a;
b[1] = 42;
System.out.println(a[1]);
System.out.println(b[1]);
```

Kazalčne podatkovne strukture

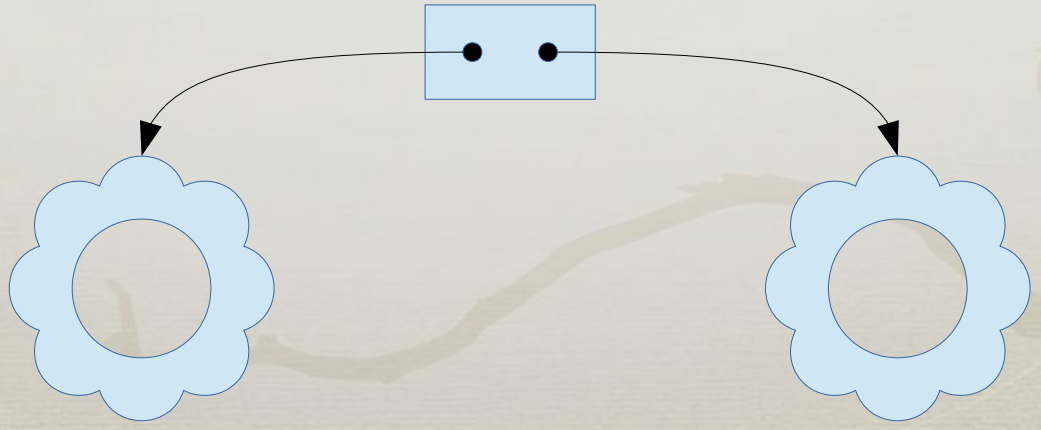


- Kazalec (pointer)
  - pove, kje se objekt nahaja
  - vrednost kazalca je naslov (v pomnilniku)
  - dostop do objekta je mogoč z dereferenciranjem
  - s kazalci je moč računati
  - lahko uporabljamo tudi kazalci na kazalce

- Referenca (reference) – prijazni kazalci
  - avtomatsko dereferenciranje
  - ni mogoče: računanje, reference na reference

Kazalčne podatkovne strukture

- Kazalčna podatkovna struktura
  - temelji na kazalcih (oz. referencah)
  - sestavljena podatkovna struktura
  - podstrukture so med seboj povezane s kazalci
  - dinamična podatkovna struktura
    - njena velikost se lahko preprosto spreminja



Kazalčne podatkovne strukture

- Vozlišče (node)
  - vsebuje **element** oz. podatek in enega ali več **kazalcev** na (druga) vozlišča
  - **vozlišča** so med seboj povezana preko **kazalcev**
    - različne oblike povezovanja

- Posebni kazalci
  - `null` … ne kaže nikamor
  - `first` … kaže na prvo vozlišče
  - `last` … kaže na zadnje vozlišče
  - itd.

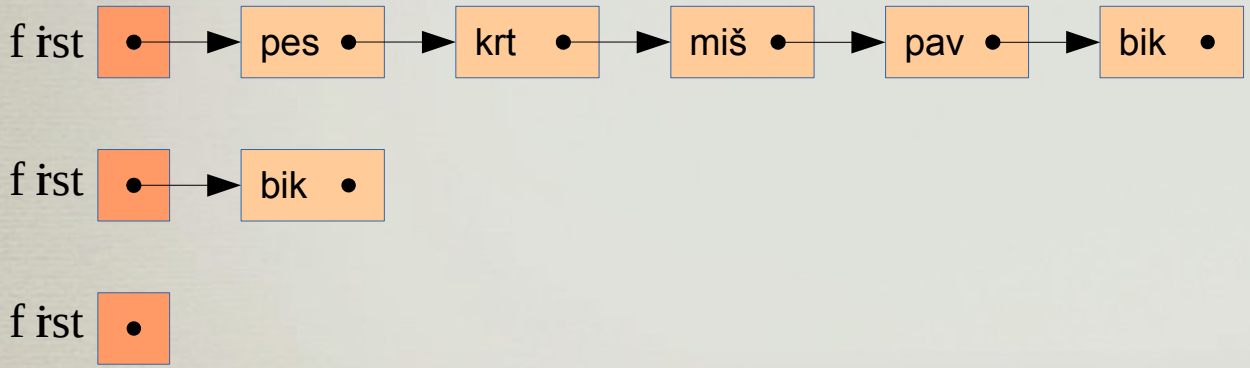
Enojno povezani seznam

- Vozlišče v EPS
  - vsebuje **element** in **kazalec** (povezavo) na naslednika

```
class Node is
    item: Object
    next: Node

    init Node(item, next) is
        this.item = item
        this.next = next
    end
end
```

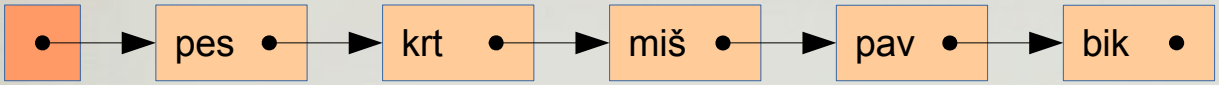
Enojno povezani seznam

- Vozlišča vsebujejo povezavo v eno smer
  - na naslednji element
  - kateri je prvi in kateri je zadnji?

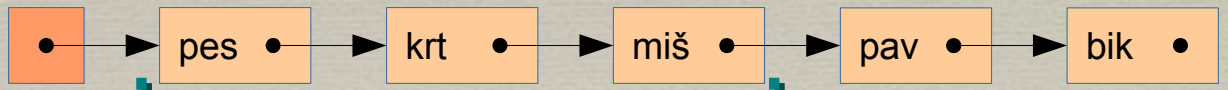


Enojno povezani seznam

- Ustvarjanje seznama
  - ustvarjanje vozlišč




```
first = Node("pes",
             Node("krt",
                  Node("miš",
                       Node("pav",
                            Node("bik", null)
                       )
                  )
             )
        )
```



Enojno povezani seznam

- Sprehodi po seznamu
  - prehod seznama

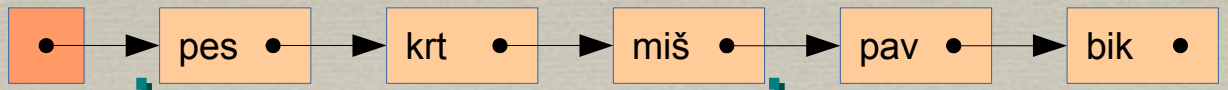
```
p = first
while p != null do
    println(p.item)
    p = p.next
```

  - iskanje vozlišča po vrednosti

```
p = first
while p != null and
      p.item != key do
    p = p.next
```

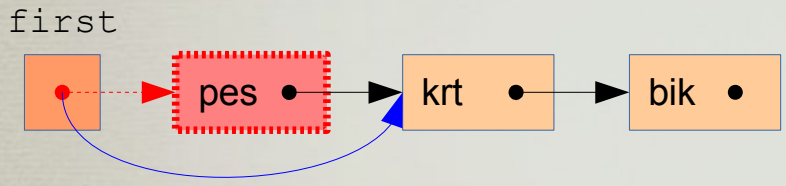
  - iskanje zadnjega vozlišča

```
p = first
while p.next != null do
    p = p.next
return p
```



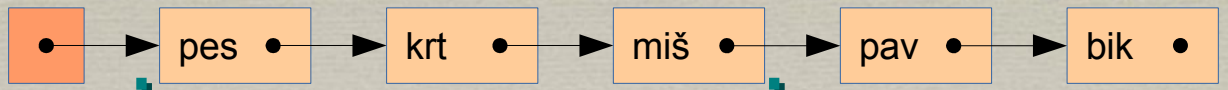
Enojno povezani seznam

- Odstranjevanje elementov
  - vseh
  - prvega
    - prazen seznam?



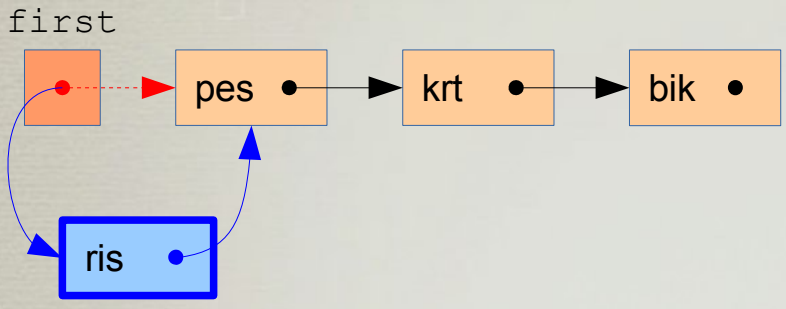
  - drugega
  - za podanim
  - zadnjega

```
// Brisanje vseh
first = null

// Brisanje prvega (n >= 1)
first = first.next

// Brisanje drugega (n >= 2)
first.next = first.next.next

// Brisanje za podanim p (p != last)
p.next = p.next.next
```



Enojno povezani seznam

- Dodajanje elementov
  - pred prvim



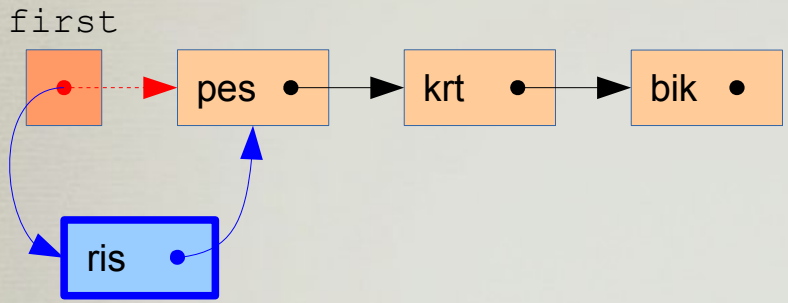
  - za prvim
  - za podanim
  - pred podanim

```
// Dodajanje pred prvim
first = Node(x, first)

// Dodajanje za prvim (n >= 1)
first.next = Node("lev", first.next)

// Dodajanje za podanim, p != null
p.next = Node("ris", p.next)

// Dodajanje pred podanim, p != null
// Trik: zamenjamo elementa (item)
p.next = Node(p.item, p.next)
p.item = "jak"
```

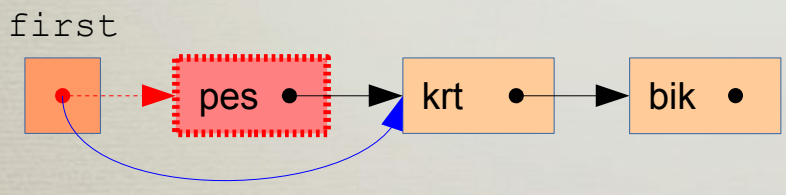
Enojno povezani seznam

- EPS kot sklad

push("ris")



pop("ris")

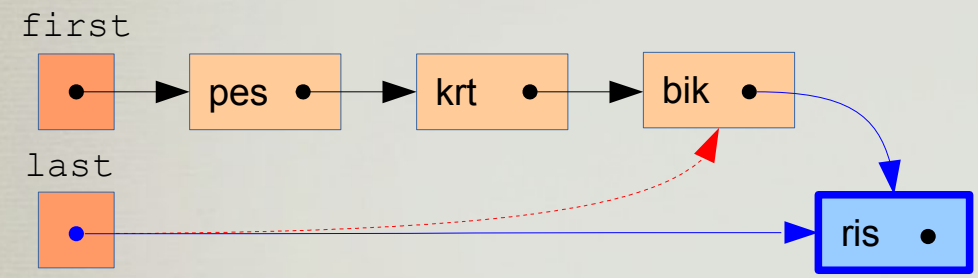



```
fun push(x) is
    first = Node(x, first)

fun pop() is
    if first == null then error
    x = first.item
    first = first.next
    return x
```

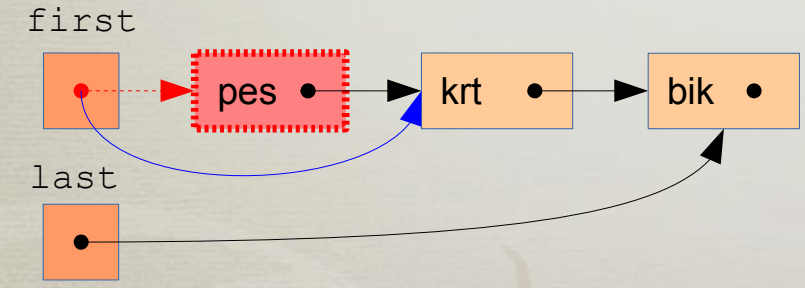
Enojno povezani seznam

- EPS kot vrsta
  - dodamo atribut `last`

`enqueue("ris")`



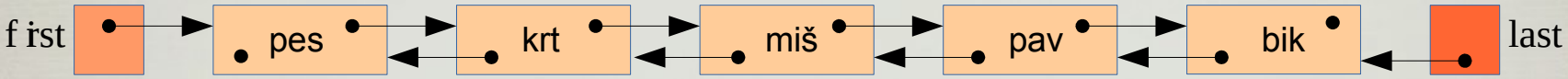
`dequeue()`




```
fun enqueue(x) is
    node = Node(x, null)
    if first == null then
        first = node
    else
        last.next = node
    last = node

fun dequeue() is
    if first == null then
        error
    x = first.item
    first = first.next
    if first == null then
        last = null
    return x
```

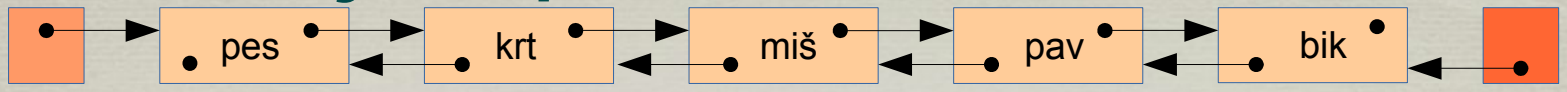
Dvojno povezani seznam

- Vsako vozlišče ima dve povezavi:
  - naprej oz. naslednji
  - nazaj oz. prejšnji




```
class Node is
    item: Object
    prev: Node
    next: Node

    init Node(item, prev, next) is
        this.item = item
        this.prev = prev
        this.next = next
    end
end
```

Dvojno povezani seznam



- Operacije
  - brisanje spredaj in zadaj

```
// Brisanje prvega
first = first.next
first.prev = null
```

```
// Brisanje zadnjega
last = last.prev
last.next = null
```

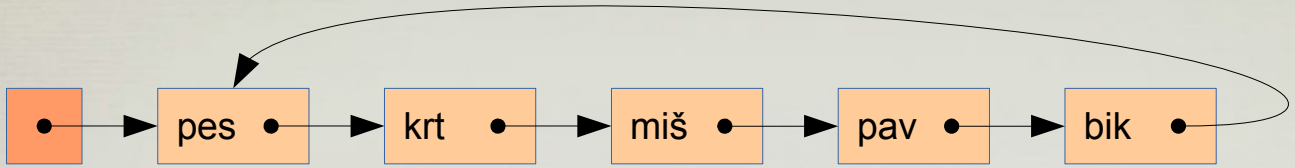
  - dodajanje spredaj in zadaj

```
// Dodajanje spredaj (n >= 1)
first = Node("noj", null, first)
first.next.prev = first
```

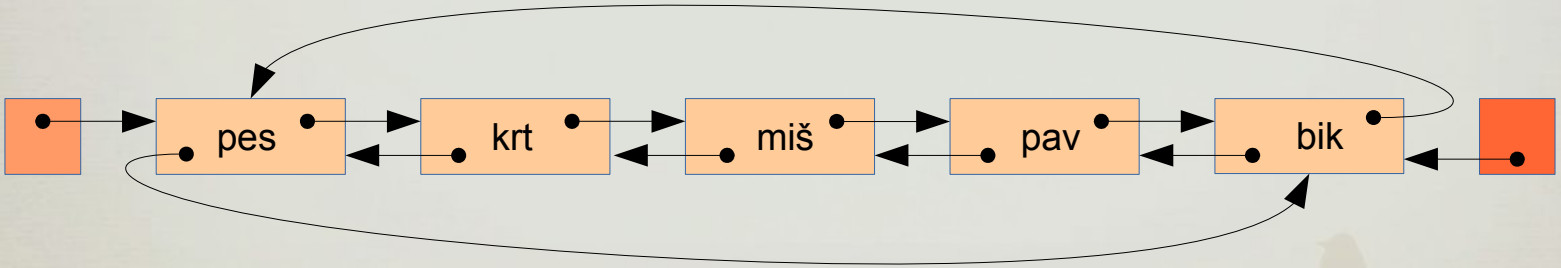
```
// Dodajanje zadaj
last = Node("noj", last, null)
last.prev.next = last
```

Ciklično povezani seznam

- Ciklični EPS

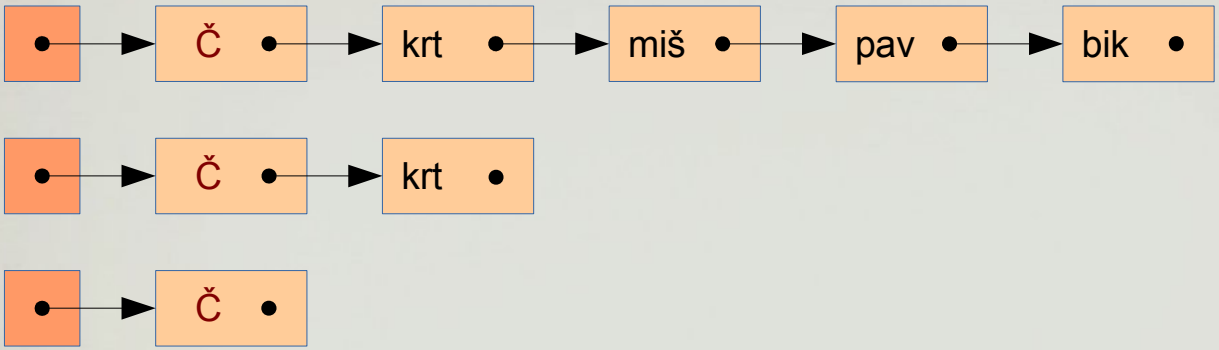


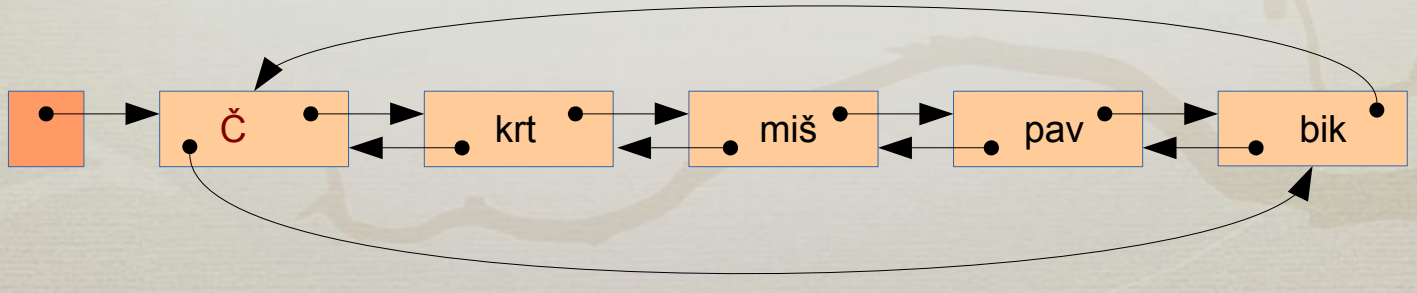
- Ciklični DPS



Povezani seznam s čuvajem

- Čuvaj (*sentinel*) *praznega seznama*
  - poseben element, ki je vedno prisoten





Seznam v polju

- Motivacija
  - za izvedbo seznama ne želimo uporabiti kazalcev in dinamične alokacije posameznih vozlišč
    - kazalci so 32/64 bitni, zasedajo preveč prostora
    - alokacija je „prepočasna“ ali pa ni na voljo
  - namesto tega želimo polje z vnaprej dano kapaciteto
    - kazalci so lahko „krajši“, npr. 8 bitni
    - za alokacijo poskrbimo sami

Seznam v polju

- Enojno povezani seznam
  - polje `next` vsebuje indekse naslednikov
  - polje `item` vsebuje elemente
  - atribut `first`

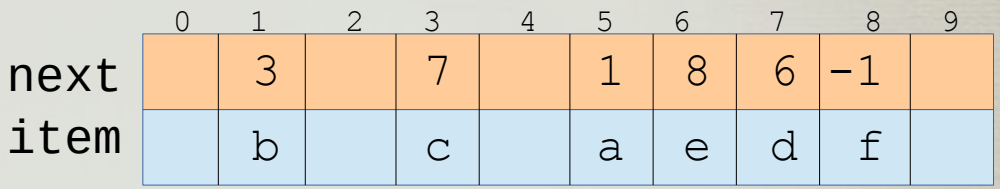
`first = 5`

	0	1	2	3	4	5	6	7	8	9
next		3		7		1	8	6	-1	
item		b		c		a	e	d	f	

Seznam v polju

- Enojno povezani seznam
  - operaciji `push(x)` in `pop()`

`first = 5`

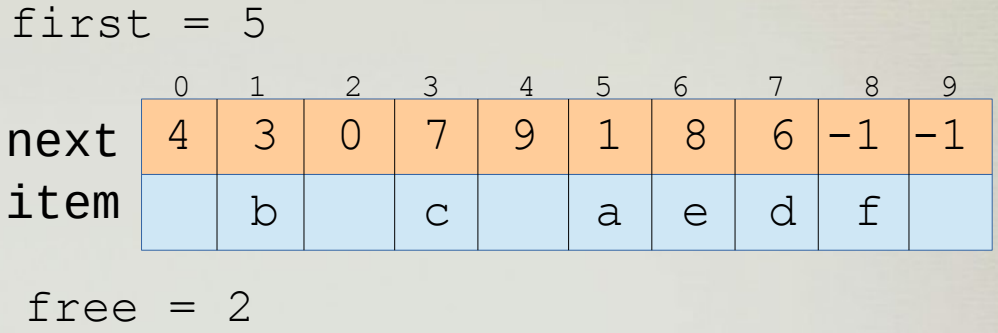



```
fun push(x) is
    i = allocate()
    item[i] = x
    next[i] = first
    first = i
```

```
fun pop() is
    // check if empty
    x = item[first]
    i = first
    first = next[first]
    free(i)
    return x
```

Seznam v polju

- Prosta mesta
  - atribut `free`
  - `allocate()`
  - `free(i)`




```
fun allocate() is
    if free == -1 then error
    i = free
    free = next[free]
    return i
```

```
fun init is
    first = -1
    free = 0
    for i = 0 to next.size-2 do
        next[i] = i+1
    next[next.size-1] = -1
```

```
fun free(i) is
    next[i] = free
    free = i
```

Persistenca

- Hranjenje zgodovine sprememb
  - enojno povezani seznam
  - operaciji `push(list, first)` in `pop(list)` vrneta nov seznam (oz. kazalec na prvi element seznama)

```
fun push(l, x) is
    return Node(x, l)

fun pop(l) is
    return l.next
```

Implicitne in eksplicitne PS

- Implicitna podatkovna struktura
  - porabi *zelo malo* dodatnega prostora poleg prostora za shranjene podatke
  - odnosi med elementi so shranjeni implicitno
    - z vrstnim redom elementov v pomnilniku
    - npr. zaporedje je podano z vrstnim redom v polju
- Eksplicitna podatkovna struktura
  - potrebuje dodatni prostor za hranjenje odnosov
  - odnosi med elementi so shranjeni eksplicitno
    - pogosta uporaba kazalcev
    - npr. zaporedje je podano z relacijo *naslednik* (kazalec)

Povzetek

operacija		EPS	DPS
enqueue(x)	sklad, vrsta, dvrsta	$O(1)$	$O(1)$
dequeue(), pop()	sklad, vrsta, dvrsta	$O(1)$	$O(1)$
enqueueFront(x), push(x)	sklad, vrsta, dvrsta	$O(1)$	$O(1)$
dequeueBack()	sklad, vrsta, dvrsta	$O(n)$	$O(1)$
get(i)	zaporedje	$O(n)$	$O(n)$
set(i, x)	zaporedje	$O(n)$	$O(n)$
find(x)	zaporedje	$O(n)$	$O(n)$
insert(i, x)	zaporedje	$O(n)$	$O(n)$
delete(i)	zaporedje	$O(n)$	$O(n)$
remove(x)	vreča, množica	$O(n)$	$O(n)$
add(x) – vreča	vreča, množica	$O(1)$	$O(1)$
addUnique(x) – množica	vreča, množica	$O(n)$	$O(n)$