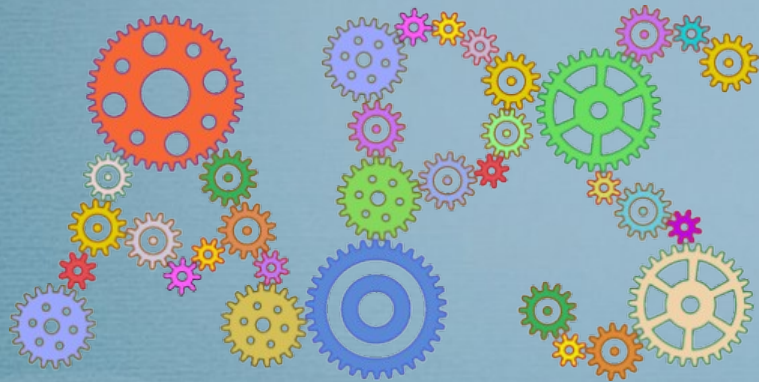


Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

**Povezani
seznami**



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, Date, ArrayList, polja (arrays), ...
- delijo stanje

```
Date a = new Date();  
Date b = a;  
b.setYear(2042);  
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]);
```

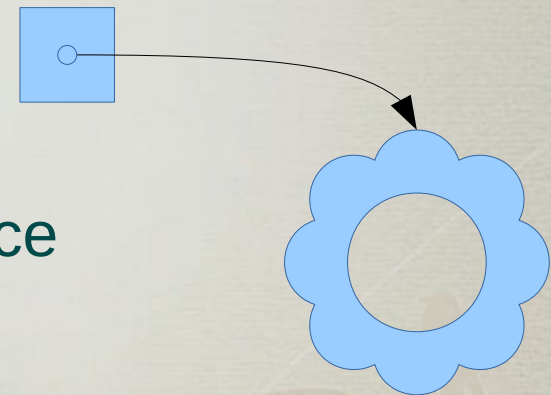
Kazalčne podatkovne strukture

- Kazalec (pointer)

- vrednost kazalca je pomnilniški **naslov**
 - pove, kje se objekt oz. vrednost nahaja
- **dereferenciranje**
 - pridobitev objekta, na katerega kaže kazalec
- računanje s kazalci (C, C++, ...)
 - računanje z naslov, zahteva previdnost
 - lahko uporabljamo tudi kazalce na kazalce

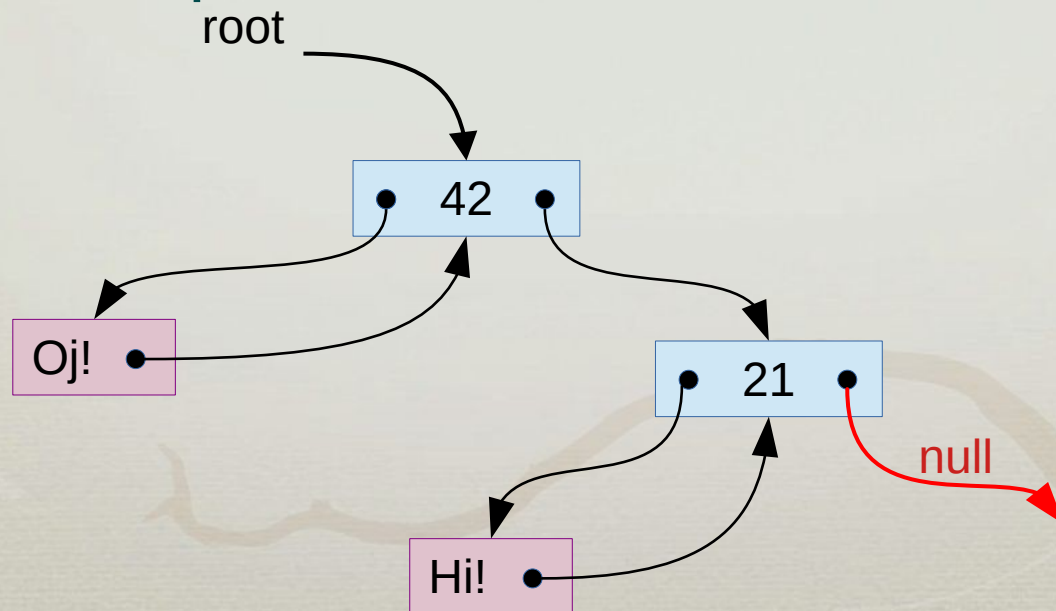
- Referenca (reference)

- prijazni kazalci, avtomatsko dereferenciranje
- računanje s kazalci ni možno (zelo omejeno)



Kazalčne podatkovne strukture

- Kazalčna podatkovna struktura
 - temelji na kazalcih ali referencah
 - **vozlišče** (node)
 - predstavlja podstrukturo
 - me seboj so podstrukture povezane s kazalci
 - dinamična podatkovna struktura



Enojno povezani seznam

- Vozlišče v EPS
 - vsebuje **element** in
 - **povezavo** na naslednika

```
// z referenco
class Node is
    item: Object
    next: Node

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

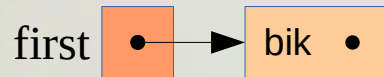
```
// Java (referenca)
class Node {
    Object item;
    Node next;
}
```

```
// C (kazalec)
struct Node {
    Object item;
    Node* next;
}
```

```
Rust (ovojni tipi)
struct Node<T> {
    item: T,
    next: Option<Box<Node<T>>>,
}
```

Enojno povezani seznam

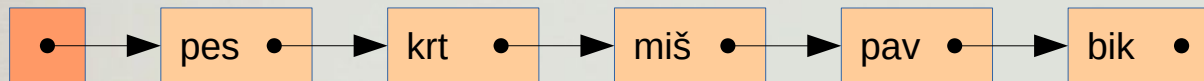
- Vozlišča vsebujejo povezavo v eno smer
 - na naslednji element
 - kateri je prvi in kateri je zadnji?



Enojno povezani seznam

- Ustvarjanje seznama
 - ustvarjanje vozlišč

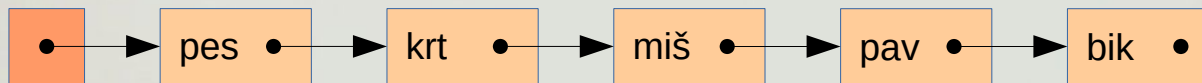
```
first = Node("bik", null);  
first = Node("pav", first);  
first = Node("miš", first);  
first = Node("krt", first);  
first = Node("pes", first);
```



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

Enojno povezani seznam

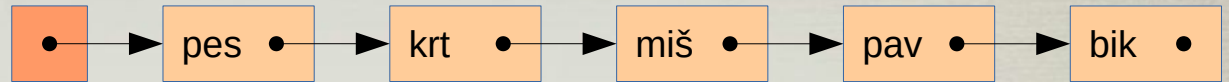
- Sprehodi po seznamu
 - prehod seznama
 - iskanje vozlišča po vrednosti
 - iskanje zadnjega vozlišča



Enojno povezani seznam

- Odstranjevanje elementov

- vseh



- prvega

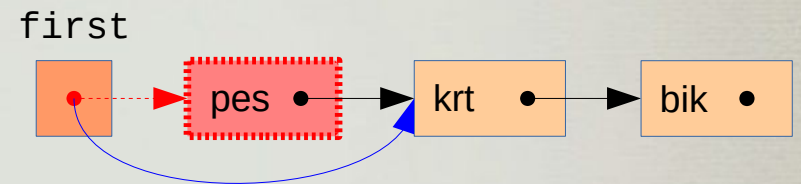


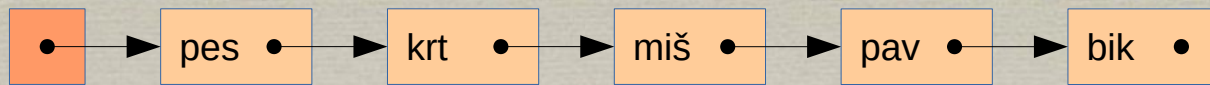
- prazen seznam?

- drugega

- za podanim

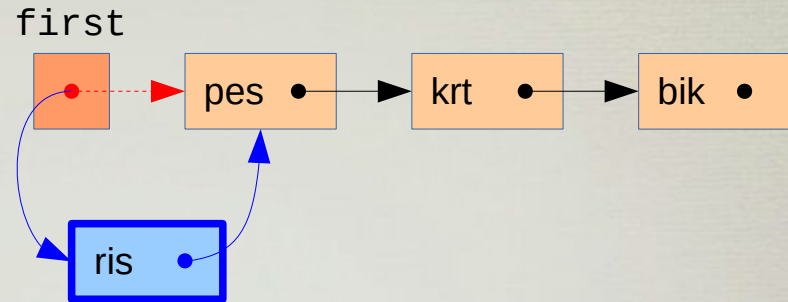
- zadnjega





Enojno povezani seznam

- Dodajanje elementov
 - pred prvim
 - za prvim
 - za podanim
 - pred podanim

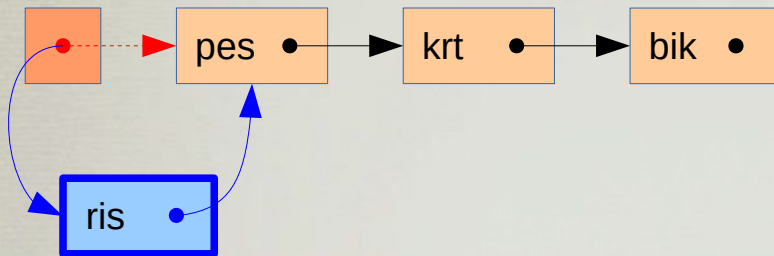


Enojno povezani seznam

- EPS kot sklad

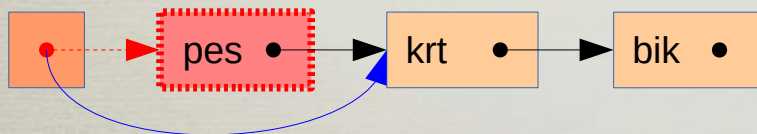
push("ris")

first



pop("ris")

first



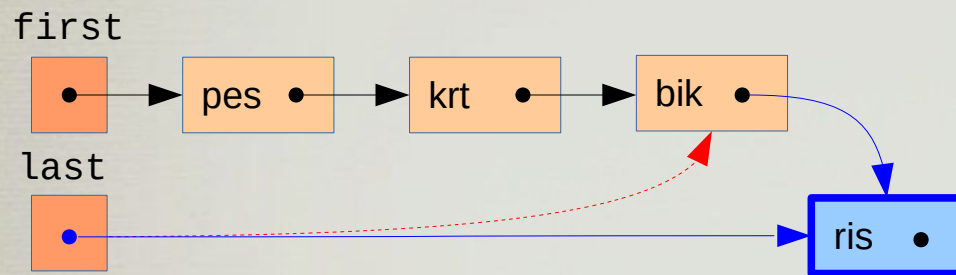
```
fun push(x) is  
    first = Node(x, first)
```

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

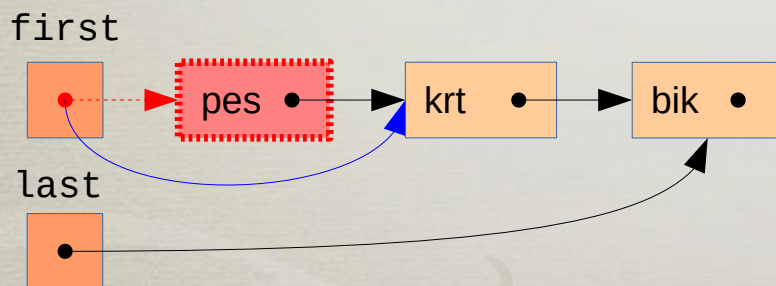
Enojno povezani seznam

- EPS kot vrsta
 - dodamo atribut last

enqueue("ris")



dequeue()

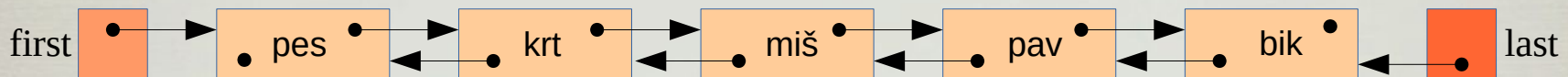


```
fun enqueue(x) is
  node = Node(x, null)
  if first == null then
    first = node
  else
    last.next = node
    last = node
```

```
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
```

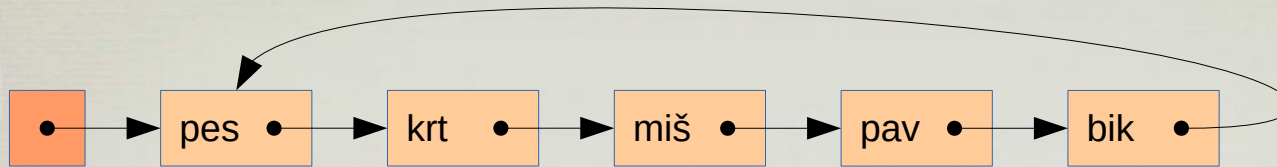
- dodajanje spredaj in zadaj

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

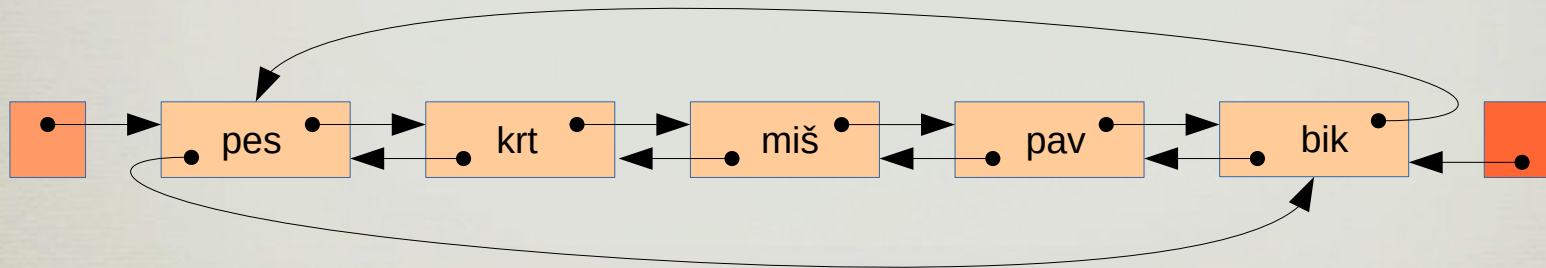
```
// Dodajanje zadaj  
last = Node("noj", last, null)  
last.prev.next = last
```

Ciklično povezani seznam

- Ciklični EPS

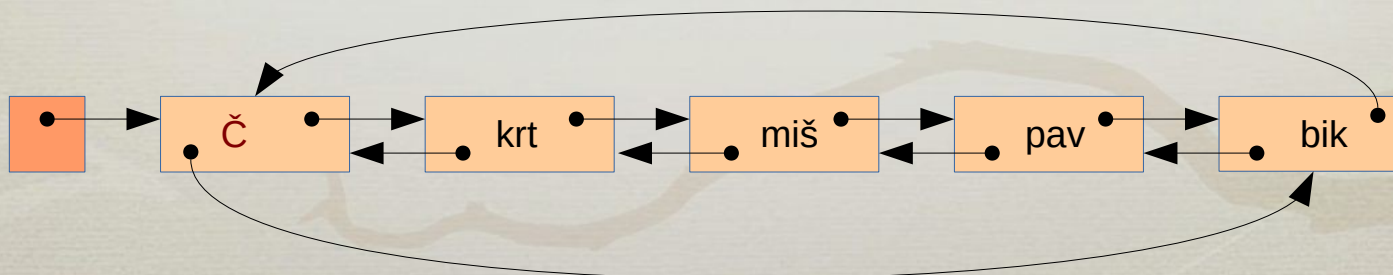
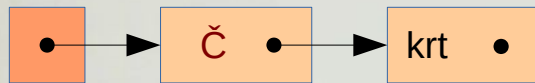
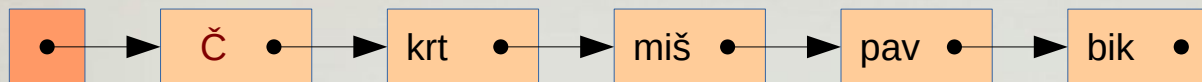


- Ciklični DPS



Povezani seznam s čuvajem

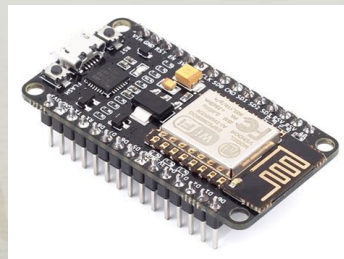
- Čuvaj (*sentinel*) praznega seznama
 - poseben element, ki je vedno prisoten



Povezani seznam v polju

- Motivacija

- če je dinamična alokacija vozlišč težavna
 - prepočasna alokacija, 32/64 bitni kazalci so preveliki
 - izvajalno okolje ne podpira kopice
- uporabimo statična polja
 - namesto *kazalcev* uporabljamo *indekse*
 - za hranjenje elementov, za hranjenje »kazalcev«
 - preprosto alokacijo implementiramo sami



Povezani 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	

*Kako dodajamo
nove elemente?*



Povezani seznam v polju

- EPS: dodajanje in odvzemanje
 - operaciji `push(x)` in `pop()`

`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	

```
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
```

Povezani seznam v polju

- EPS: alokacija in proste celice
 - atribut free
 - allocate()
 - free(i)

first = 5

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

free = 2

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

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

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()		$O(1)$	$O(1)$
enqueueFront(x), push(x)		$O(1)$	$O(1)$
dequeueBack()		$O(n)$	$O(1)$
get(i)	zaporedje	$O(n)$	$O(n)$
set(i, x)		$O(n)$	$O(n)$
find(x)		$O(n)$	$O(n)$
insert(i, x)		$O(n)$	$O(n)$
delete(i)		$O(n)$	$O(n)$
remove(x)	vreča množica	$O(n)$	$O(n)$
add(x) – vreča		$O(1)$	$O(1)$
addUnique(x) – množica		$O(n)$	$O(n)$