Abstraktni podatkovni tipi in podatkovne strukture

Kaj bomo spoznali

- Podatkovne strukture.
- Abstraktne podatkovne tipe.

~~**Kaj NE bomo obravnavali**~~

Abstraktni podatkovni tipi

Podatkovni tipi: vrednosti, operacije in predstavitev podatkov.

Klasifikacija podatkovnih tipov, nekaj primerov:

- **Boolean**
- **integer**
- **Objekti** iz različnih razredov.

Vsak tak podatkovni tip označuje:

- Množica **vrednosti**
- **Predstavitev podatka** (enaka za vse te vrednosti)
- množica **operacij** (ki jih lahko izvedemo na teh vrednostih).

Abstraktni podatkovni tipi: le vrednosti in operacije

Abstraktni podatkovni tipi

Abstraktni podatkovni tip (**abstract data type** (**ADT)** je matematični model za določen razred podatkovnih struktur, ki imajo podobno obnašanje.

Abstraktni podatkovni tip za določene tipe podatkov, v enem ali več programskih jezikih, ki imajo podoben pomen.

Abstraktni podatkovni tip je definiran indirektno **le z operacijami**, ki jih lahko izvajamo na njemu, in z matematičnimi omejitvami na teh operacijah.

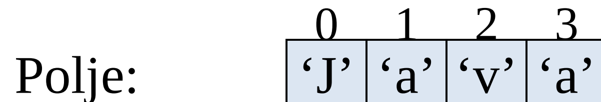
Uvod v podatkovne strukture

- Podatkovna struktura je sistematičen način organiziranja zbirke podatkov.
- **Statična** podatkovna struktura je tista, katere kapaciteto fiksiramo ob njeni tvorbi.
  Primer: tabela oziroma polje (array).
- **Dinamična** podatkovna struktura je tista, katere kapaciteta je spremenljiva. Kadarkoli jo lahko razširjamo ali krčimo.
  Primeri: povezan seznam (linked list), binarno drevo (binary tree).
- Za vsako podatkovno strukturo potrebujemo algoritme za vstavljanje, brisanje, iskanje itd.

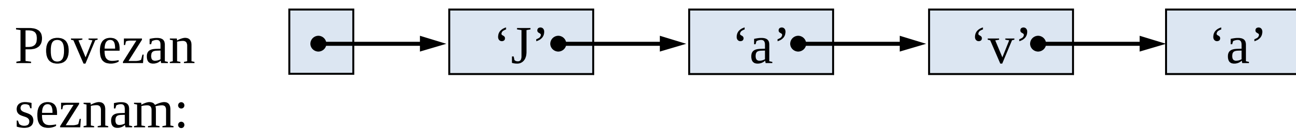
Primer: Predstavitev nizov

Možne podatkovne strukture za predstavitev niza "Java":

Polje:



Povezan seznam:

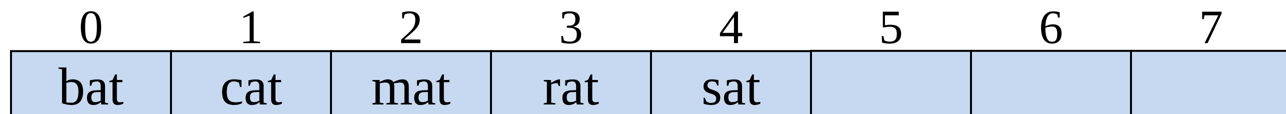


WEB-demo

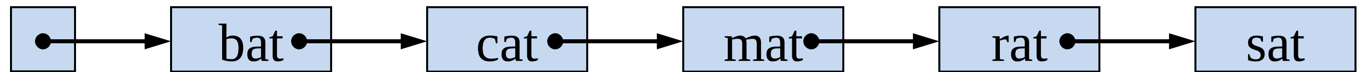
Primer: Predstavitev množic

Možne podatkovne strukture za predstavitev množice besed
{bat, cat, mat, rat, sat}:

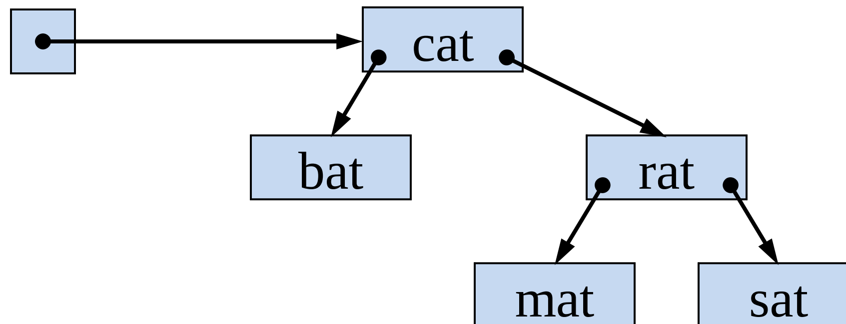
(Urejena) tabela:



(Urejen) povezan seznam



Binarno iskalno drevo:



Kolekcije

- Kolekcija je objekt, ki služi kot skladišče za druge objekte
- Kolekcija običajno nudi servise, kot so dodajanje,brisanje ali kakšno drugačno rokovanje z objekti, ki jih vsebuje
- Včasih lahko elemente neke kolekcije uredimo, včasih ne
- Kolekcije so včasih homogene, včasih pa heterogene

WEB

Abstrakcija

- Naše podatkovne strukture morajo biti abstrakcije
- To pomeni, da morajo skrivati nepotrebne podrobnosti
- Želimo ločiti vmesnik (interface) strukture od implementacije
- Tako lažje upravljamo s kompleksnostjo in spreminjamo implementacijo brez spremembe vmesnika

Kaj pomeni možnost spreminjanja implementacije brez spreminjanja vmesnika? Zakaj si to želimo?

Abstraktni podatkovni tipi

- Ko pišemo programsko kodo, ne razmišljamo, kako so nizi predstavljeni: preprosto deklariramo spremenljivke tipa `String` ter rokujemo z njimi s pomočjo operacij `String`.
- Podobno nam je vseeno, kako so predstavljene množice. Preprosto deklariramo spremenljivke tipa `Set` in rokujemo z njimi s pomočjo operacij `Set`.
- **Abstraktni podatkovni tip** je podatkovni tip, katerega predstavitev je programski kodi prikrita.
- Abstraktni podatkovni tipi lajšajo načrtovanje obsežnih programov.

Abstraktni podatkovni tipi (ADT)

- *Abstraktni podatkovni tipi* *(Abstract Data Types ,ADT)* oziroma podatkovne strukture ali kolekcije pomnijo podatke in omogočajo različne operacije za dostop do teh podatkov in njihovo spreminjanje.

- Množica operacij definira vmesnik (*interface*) na ADT

- Dokler ADT izpolnjuje obljube vmesnika, je vseeno, kako je ADT implementiran

- Objekti so primeren programski mehanizem za tvorbo ADT, saj so interne podrobnosti *enkapsulirane*

Abstraktni podatkovni tipi (2)

- Abstraktni podatkovni tip je matematična množica podatkov skupaj z definiranimi operacijami na takem tipu podatkov
- Primeri:
  - množica celih števil (do določene velikosti), z operacijami +, -, /, *, %

    *int oziroma integer so **implementacije** tega ADT na nivoju jezika (pascal, java, C ...)*
  - množica decimalnih števil (do določene velikosti), z operacijami +, -, /, *

  *real, double, float so **implementacije** tega ADT na nivoju jezika (pascal, java, C,..)*

Podatkovni tipi

- *int* ali *integer*, *float*, *double* ali *real* itd. sodijo k takoimenovanim "vgrajenim" podatkovnim tipom

- To pomeni, da jih ponuja sam programski jezik (pascal, c, java,..)

- Nove podatkovne tipe lahko tvori sam uporabnik s pomočjo polj, oštevilčenj, struktur, razredov (če imamo objektno programiranje) itd.

Podatkovne strukture

- Podatkovna struktura je abstraktni podatkovni tip, ki ga definira uporabnik
- Primeri:
  - **Kompleksna števila**: z operacijami+, -, /, *, *magnituda*, *kot*, itd.
  - **Sklad (stack)**: z operacijami *push, pop, peek, isempty*
  - **Vrsta (Queue)**: *enqueue, dequeue, isempty …*
  - **Binarna iskalna drevesa** : *insert, delete, search.*
  - **Kopica (Heap)**: *insert, min, delete-min*.

Načrtovanje podatkovne strukture

- Specifikacija
  - Množice podatkov
  - Specifikacije operacij nad temi podatki
- Načrtovanje
  - Izgled organizacije podatkov
  - Algoritmi za operacije
- Namen načrtovanja: **hitre** operacije

DEMO1

DEMO2

Bistvene operacije

- Vsaka kolekcija ADT mora omogočati:
  - dodajanje elementa
  - brisanje elementa
  - Iskanje oziroma dostop do elementa
- In še druge možnosti
  - je kolekcija prazna?
  - Izprazni kolekcijo
  - Daj mi podmnožico kolekcije
  - in še in še…

To lahko implementiramo na različne načine, ki imajo različno ceno in prednosti

Vreče in množice

- Najbolj preprost ADT je **vreča** (bag)
  - stvari lahko dodajamo, odstranjujemo, dostopamo do njih
  - Stvari niso nič urejene
  - možni so duplikati
- **Množica** (Set)
  - Isto kot vreča, vendar duplikati elementov niso dovoljeni
  - unija, presek, razlika, podmnožica

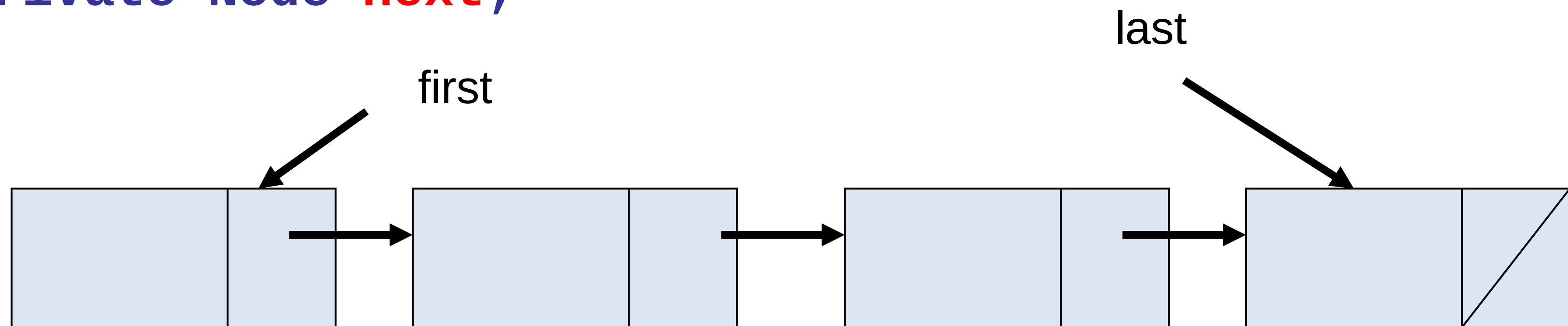
Povezani seznami (Linked Lists)

- Stvari imajo položaj v kolekciji
  - Naključen dostop ali ne?
- Polja (tabele)
  - Pomnjeno na zaporednih lokacijah v pomnilniku
- Povezani seznami(Linked Lists)

WEB

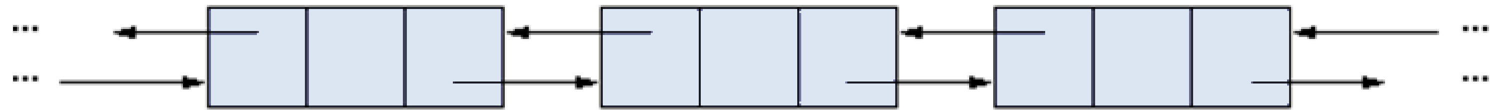
WEB

```
public class Node {
  private Object data;
  private Node next;
}
```

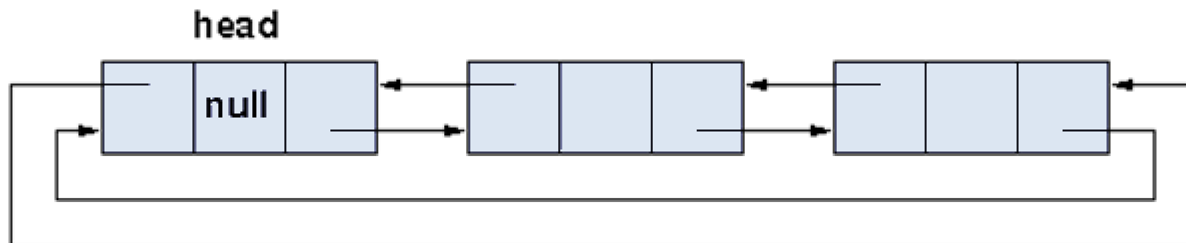


Dvojni povezani seznami

Dvojni povezani seznam



Dvojni povezani seznam z glavo in dvema elementoma



WEB

Povezani seznami: demo

Tvorba povezanih seznamov v C
Program prikazuje, kako tvorimo preprost linearen povezan seznam s pomočjo dinamične alokacije pomnilnika in kazalcev

Zakaj je izpis v obratnem vrstnem redu, kot smo podatke vnašali?
Kako bi naredili izpis v drugačnem vrstnem redu?

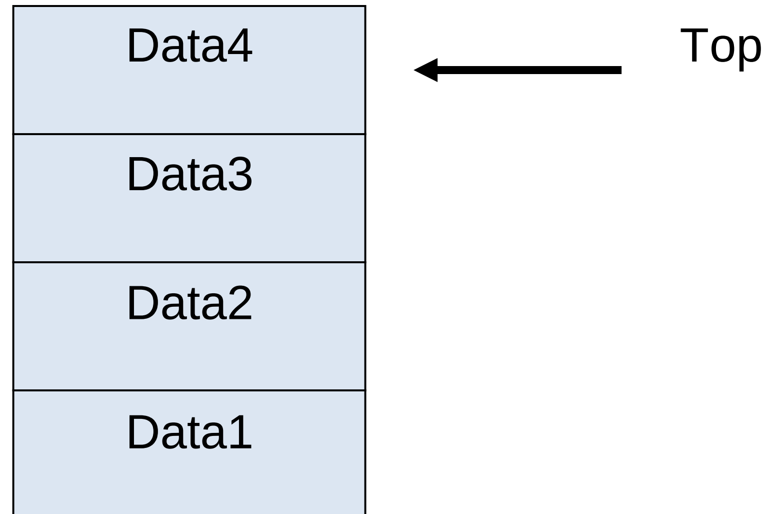
Tvorba povezanih seznamov v Javi
Program (SeznamFilmov) predstavlja interaktivno urejanje seznama filmov. Filme lahko dodajamo, iščemo, brišemo. Za vsak film imamo podan naslov in letnico. Podatke filma hranimo v svojem vozlu (Node)

Program ima didaktičen pomen (ni res, da v Javi ni kazalcev) in namenoma ne uporablja možnosti, ki jih nudijo kolekcije!

Sklad (Stack)

- Kolekcija z dostopom le do zadnjega vstavljenega elementa

- Last in first out
- insert/push
- remove/pop
- top
- make empty



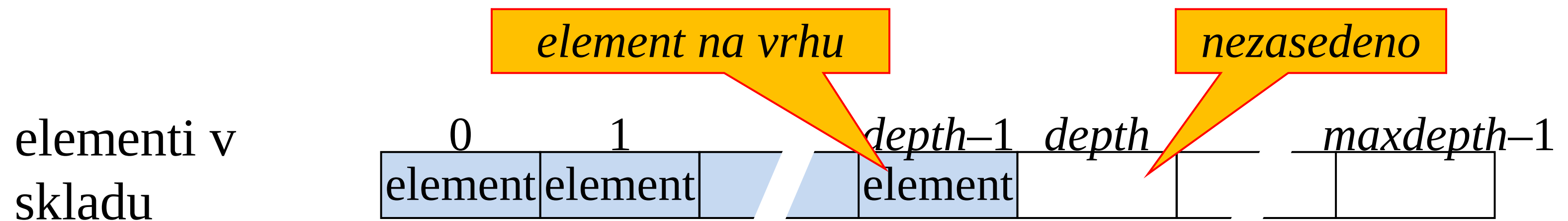
Uporaba:

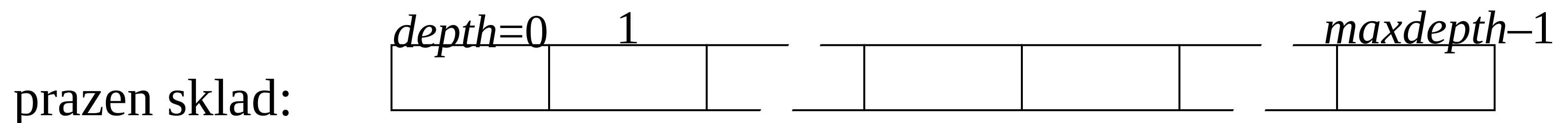
- Rekurzija,
- Obračanje podatkov
- Robotika.

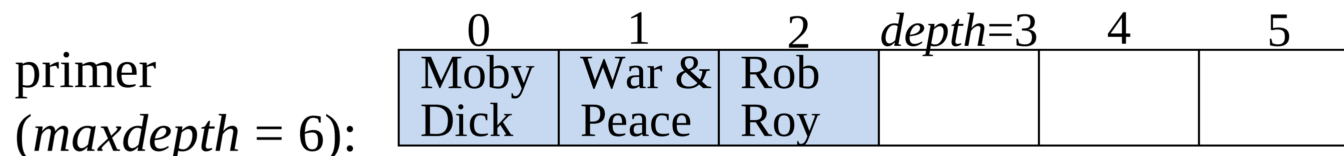
WEB

Implementacija sklada s pomočjo polja

- Predstavlja **omejen** sklad (depth ≤ *maxdepth*):
  - spremenljivka *depth* vsebuje trenutno globino
  - polje *elems* dolžine *maxdepth*, vsebuje elemente na skladu v *elems*[0… *depth*–1].







WEB

Sklad: demo

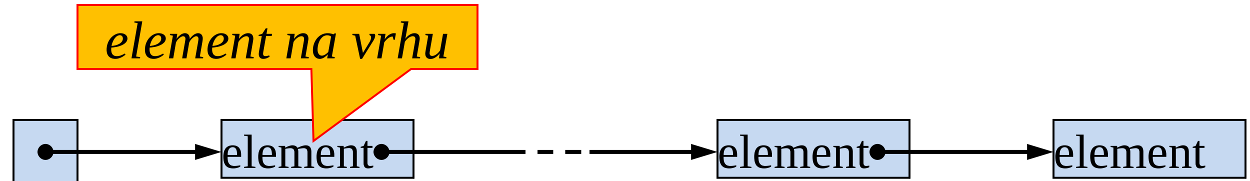
Primer implementacije sklada s poljem (stack.c)
Primer organizacije programa z dvema datotekama (projekt)
Primer uporabe "header" datoteke (stack.h)

Primer programa, ki tak sklad uporabi (stackTest.c)

Implementacija sklada z enojno povezanim seznamom

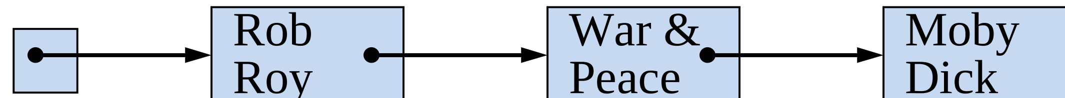
Predstavlja neomejen sklad.Prvi vozel vsebuje najvišji element.

Elementi v skladu:



Prazen sklad:

Primer:



Sklad v Javi

```
import java.util.*;
public class StackDemo{
  public static void main(String[] args) {
  Stack stack=new Stack();
  stack.push(new Integer(10));
  stack.push("a");
  System.out.println(“Vsebina sklada je " + stack);
  System.out.println(“Velikost sklada je " + stack.size());
  System.out.println(“Iz sklada vzamemo " + stack.pop());
  System.out.println(“Iz sklada vzamemo  " + stack.pop());
  //System.out.println(“Iz sklada vzamemo " + stack.pop());
  System.out.println(“vsebina sklada je " + stack);
  System.out.println(“Velikost sklada je " + stack.size());
  }}
```

Kaj, če to odkomentiram?

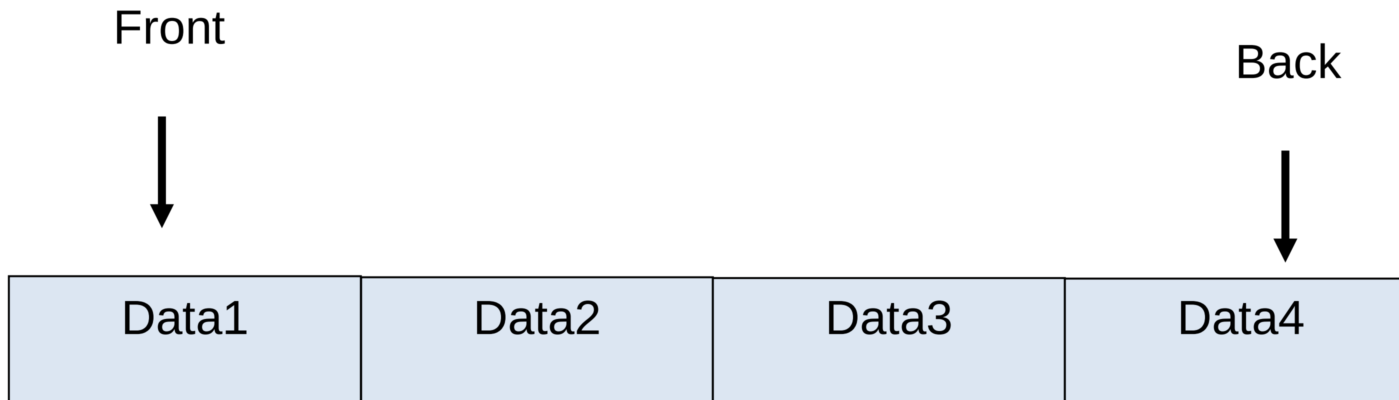
Demo

Vrste (Queues)

- Kolekcija z dostopom do elementa, ki je v njej najdlje
- "Last in last out" ali "first in first out"
- enqueue, dequeue, front
- priority queues and deque

Uporaba vrst v računalništvu

- Print server
  - vzdržuje vrsto poslov, čakajočih tiskanje.
- Disk driver
  - Vzdržuje vrsto zahtevkov dostopa do diska.
- Scheduler (razvrščevalnik v operacijskem sistemu)
  - vzdržuje vrsto procesov, ki čakajo na rezino računalniškega časa.

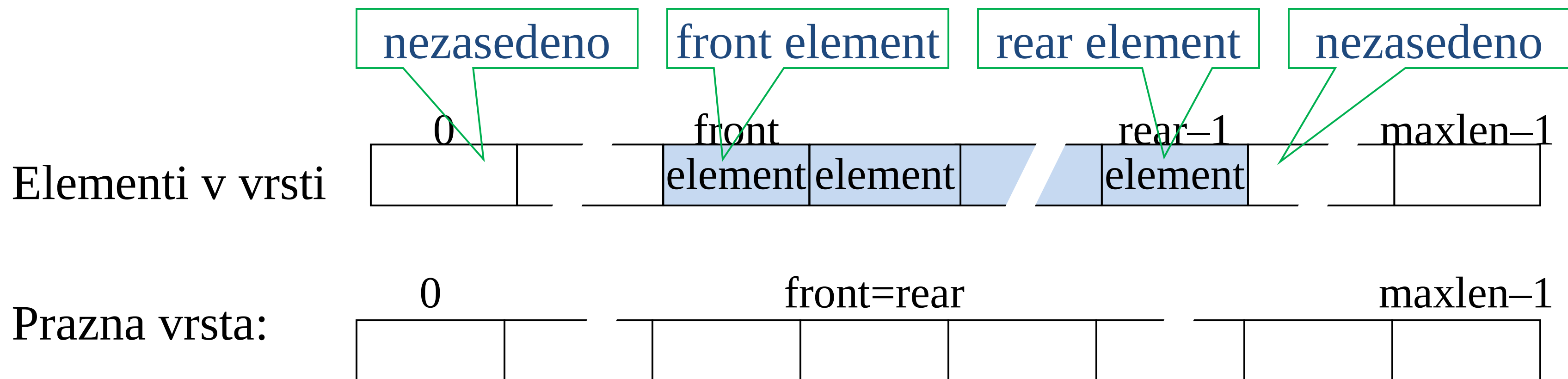


ADT vrste: možen API v Javi

```
public interface Queue {
    // Each Queue object is a queue whose elements are objects.
    //////////////// Accessors ///////////////
    public boolean isEmpty ();
    // Return true if and only if this queue is empty.
    public int size ();
    // Return this queue's length.
    public Object getFirst ();
    // Return the element at the front of this queue.
    //////////////// Transformers ///////////////
    public void clear ();
    // Make this queue empty.
    public void addLast (Object elem);
    // Add elem as the rear element of this queue.
    public Object removeFirst ();
    // Remove and return the front element of this queue.
}
```

Implementacija vrst z uporabo polj

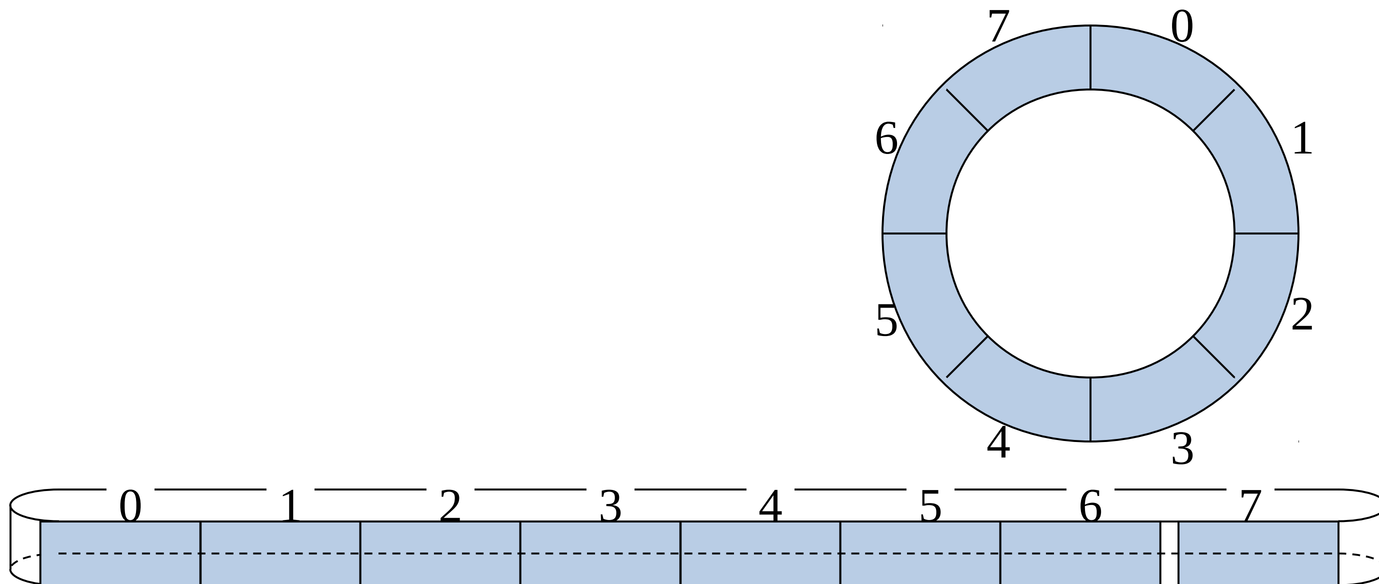
- Imamo **omejeno** vrsto (length ≤ *maxlen*) :
  - spremenljivka *length* vsebuje trenutno dolžino
  - spremenljivki *front* in *rear (prvi in zadnji)*
  - polje *elems* dolžine *maxlen*, vsebuje elemente, uvrščene v *elems*[*front*…*rear*–1]:



WEB WEB

Ciklična polja

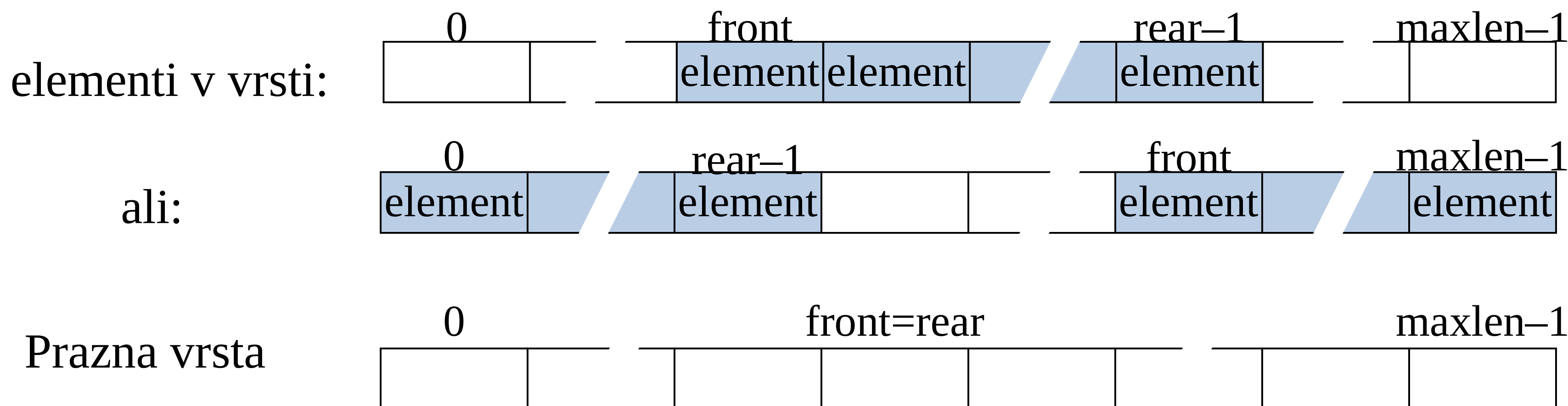
- Pri cikličnem polju dolžine *n* ima vsak element svojega naslednika in predhodnika.
- Posebni primeri:
  - Naslednik *a*[*n*–1] je *a*[0]
  - predhodnik *a*[0] je *a*[*n*–1].
- Kako lahko gledamo na ciklično polje (dolžine 8)



WEB

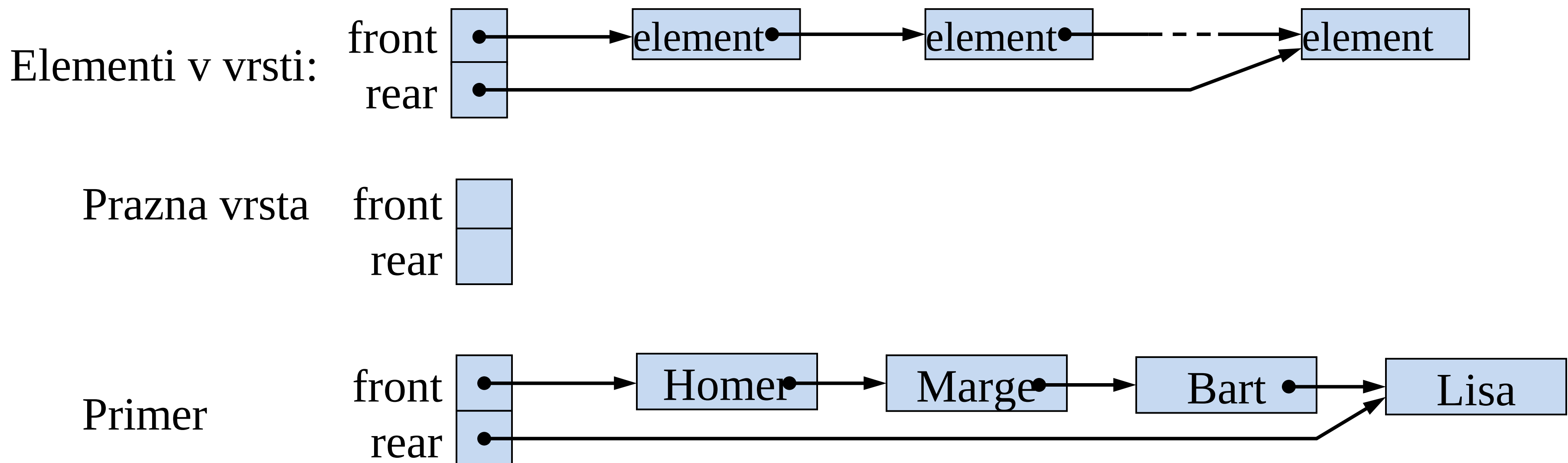
Implementacija vrst s cikličnim poljem

- Imamo **omejeno** polje (length ≤ *maxlen*):
  - spremenljivka *length* pove trenutno dolžino
  - spremenljivki *front* in *rear*
  - ciklično polje *elems* velikosti *maxlen*, ki vsebuje uvrščene elemente **ali** v *elems*[*front*…*rear*–1] **ali** v *elems*[*front*…*maxlen*–1] in *elems*[0…*rear*–1].



Implementacija vrste z enojno povezanim seznamom

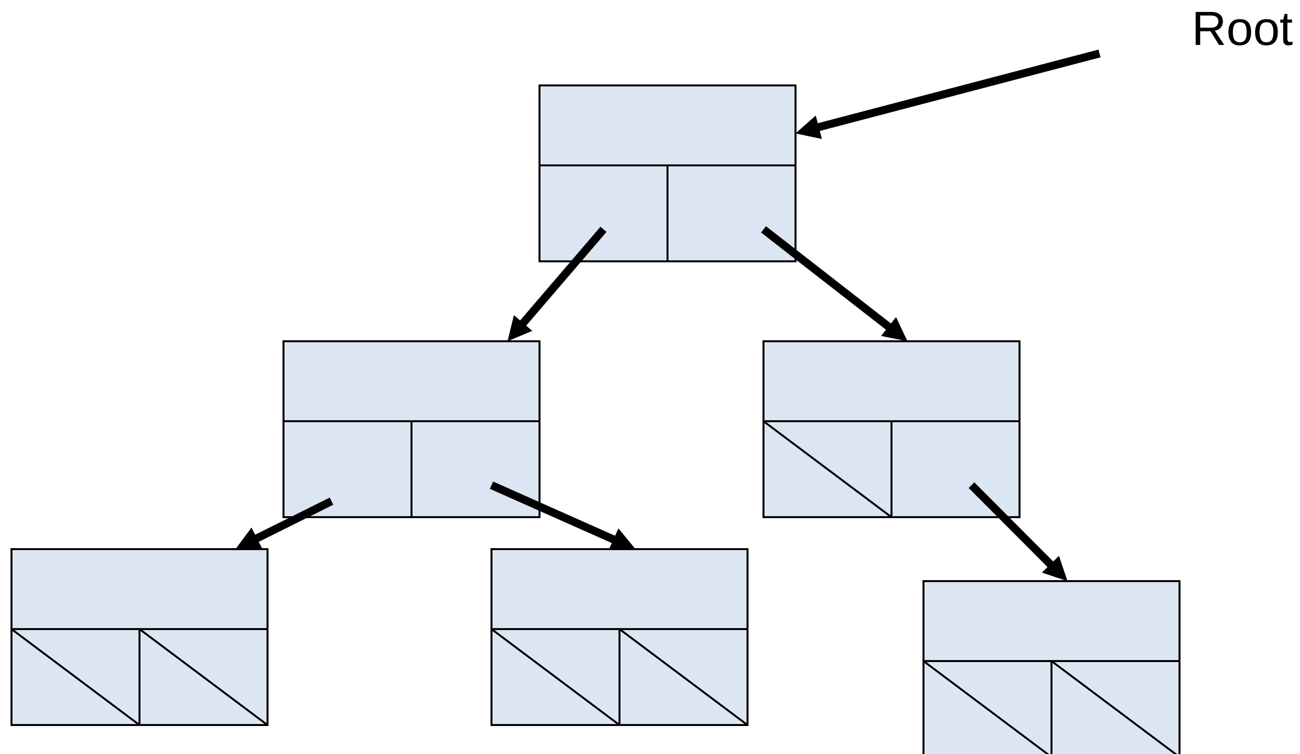
- Imamo neomejeno vrsto:
  - Prvi vozel vsebuje prvi element (front element), glava vsebuje vezi na prvi (front) in zadnji (rear) vozel.
  - spremenljivka *length* ni nujna.



Drevo (Tree)

Podobno povezanim seznamom

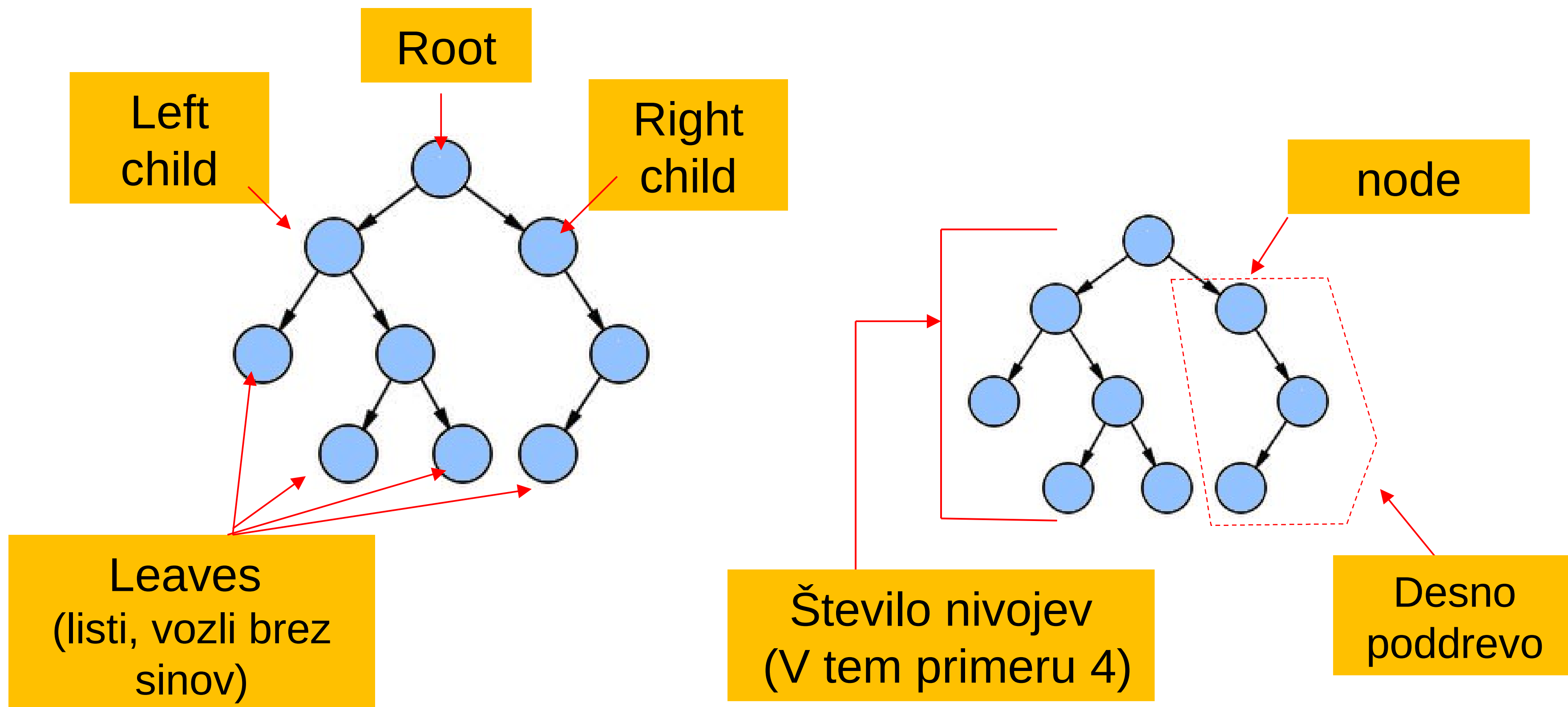
```
public class TreeNode{
   private Object data;
   private TreeNode left;
   private TreeNode right;
}
```



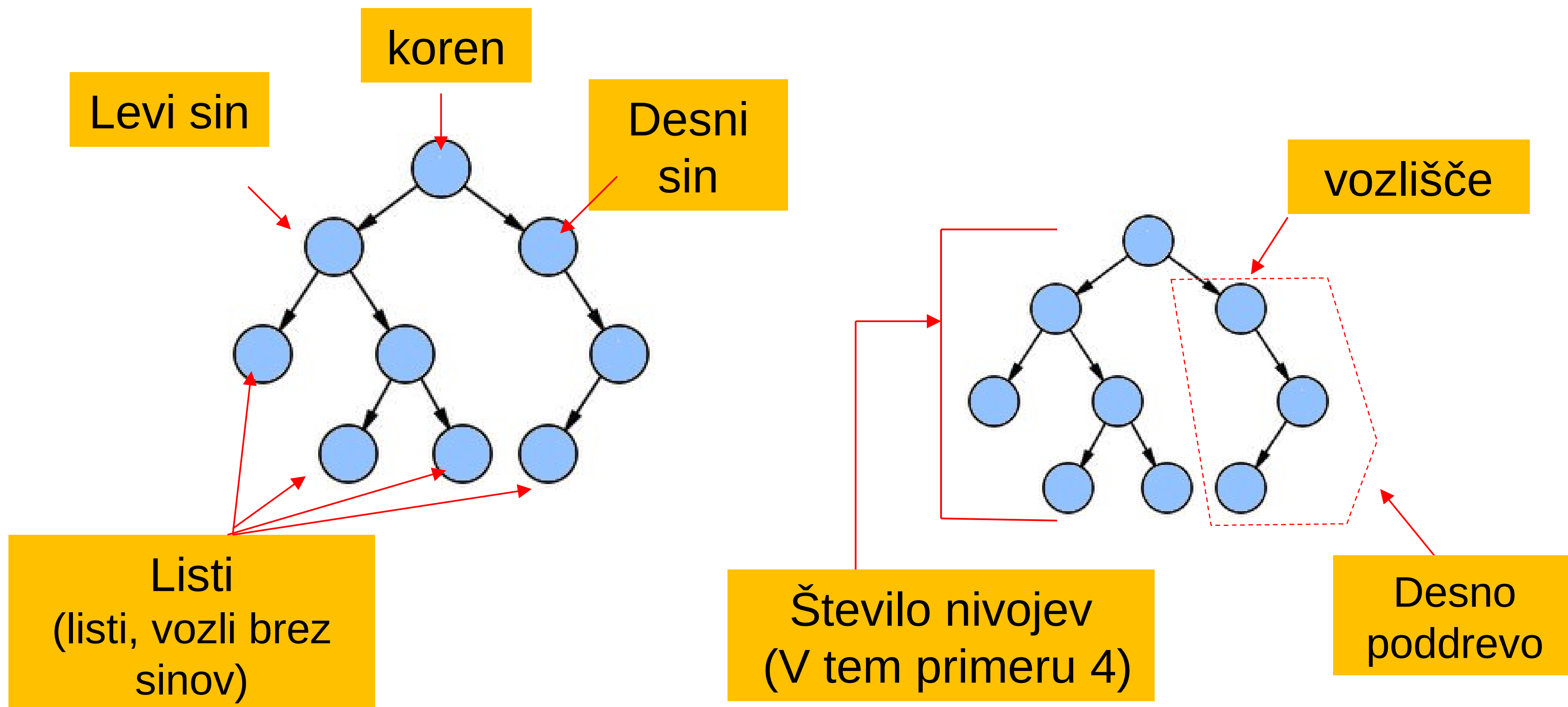
WEB

Demo

Pojmi pri binarnih drevesih



Pojmi pri binarnih drevesih



Lastnosti binarnih dreves

- Že plitko drevo lahko vsebuje veliko vozlišč. Primer: drevo z 20 nivoji lahko vsebuje $2^{20} - 1$ (cca 1.000.000) vozlišč.
- V vsakem vozliščcu se moramo odločiti, kako napredovati (levo ali desno).
- Pot do dna je relativno kratka v primerjavi s celotnim številom vozlišč.

Razred TreeNode

- Predstavlja vozlišče binarnega drevesa
- Za razliko od vozlišča pri povezanem seznamu imamo tu namesto kazalca *“next*” kazalca *“left*” in *“right”*

```
public class TreeNode
{
    private Object value;
    private TreeNode left;
    private TreeNode right;
    ...
```

Pomni referenco na levega sina

Pomni referenco na desnega sina

Drevesa in rekurzija

- Drevesna struktura je rekurzivna po svoji naravi — leva in desna poddrevesa so manjša drevesa:

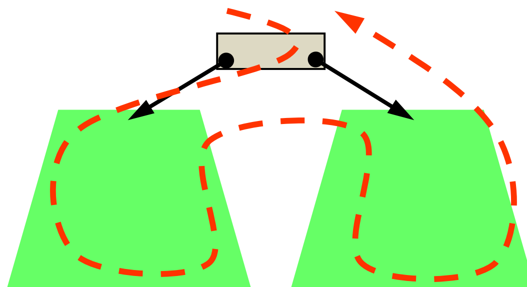
```
private void traverse (TreeNode root)
{
   // Base case: root == null,
   // the tree is empty -- do nothing
   if (root != null)  // Recursive case
   {
      process (root.getValue ());
      traverse (root.getLeft ());
      traverse (root.getRight ());
   }
}
```

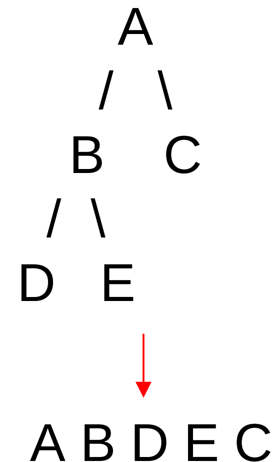
Rekurzivno prehajanje po drevesu

Prehod po drevesu

- **Premi prehod** (preorder): izpišemo oznako korena pred oznakami levega in desnega poddrevesa.

```
preorder(node)
    print node.value
    if node.left ≠ null then preorder(node.left)
    if node.right ≠ null then preorder(node.right)
```



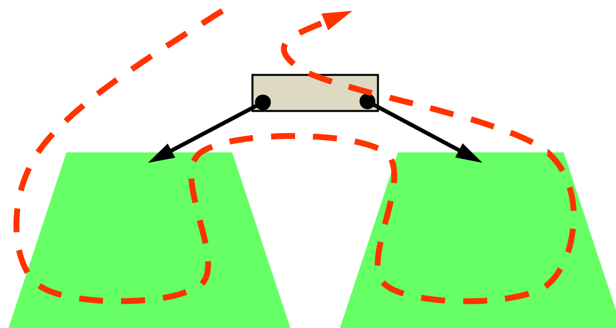


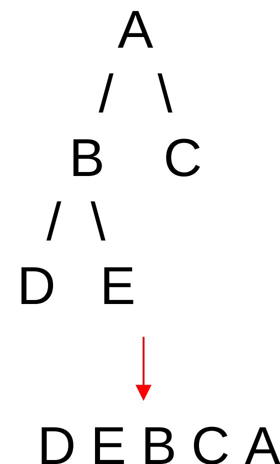
Prehod je smiseln na primer pri iskanju dane vrednosti v drevesu

Prehod po drevesu (2)

- **Obratni prehod** (postorder): Najprej izpišemo oznake levega in desnega poddrevesa, nato pa oznako korena

```
postorder(node)
    if node.left ≠ null then postorder(node.left)
    if node.right ≠ null then postorder(node.right)
    print node.value
```



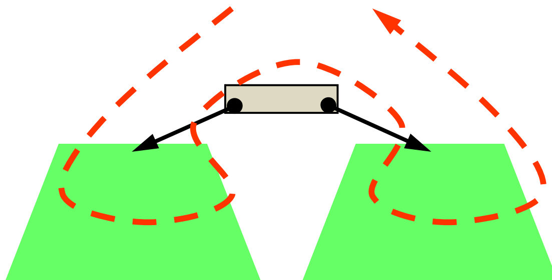


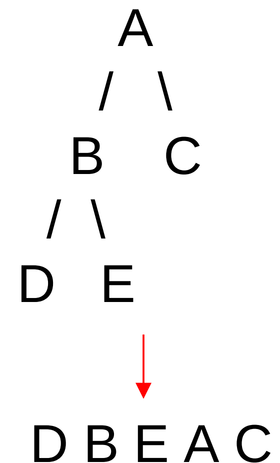
Prehod je smiseln na primer pri iskanju dane vrednosti v drevesu

Prehod po drevesu (3)

- **Vmesni prehod** (inorder): najprej gremo po levem poddrevesu, nato obdelamo koren (root) in nato še desno poddrevo.

```
inorder(node)
    if node.left ≠ null then inorder(node.left)
    print node.value
    if node.right ≠ null then inorder(node.right)
```





Štetje vozlišč binarnega drevesa

```
public int countNodes (TreeNode root)
{
    if  (root == null)
        return 0;
    else
        return 1  + countNodes (root . getLeft ()) +
                        countNodes (root . getRight ());
}
```

Osnovni primer

(za root)

Brez rekurzije bi bilo to težko

Kopiranje drevesa

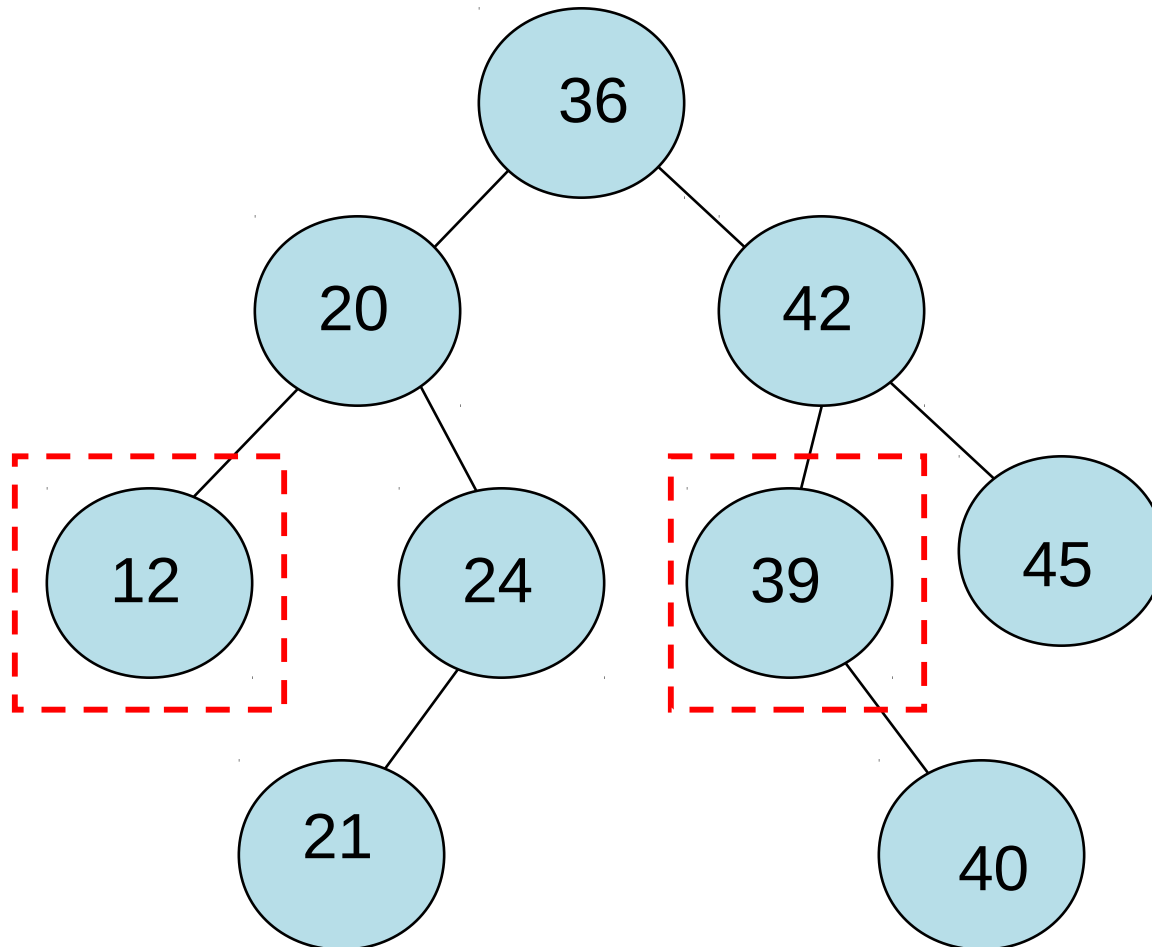
```
//  vrne referenco na novo drevo,
//  ki je “plitka” kopija, drevesa, ki začenja pri root
//   kopirana so le vozlišča, ne pa objekti (vrednosti)
// zato tako originalna kot kopirana vozlišča naslavljajo iste
objekte

public TreeNode copy (TreeNode root)
{
    if  (root == null)
       return null;
    else
       return new TreeNode (root . getValue (),
                                              copy (root . getLeft ()),
                                                  copy (root . getRight ()));
}
```

Razlaga plitkega in globokega kopiranja

Brisanje vozlišč - primeri

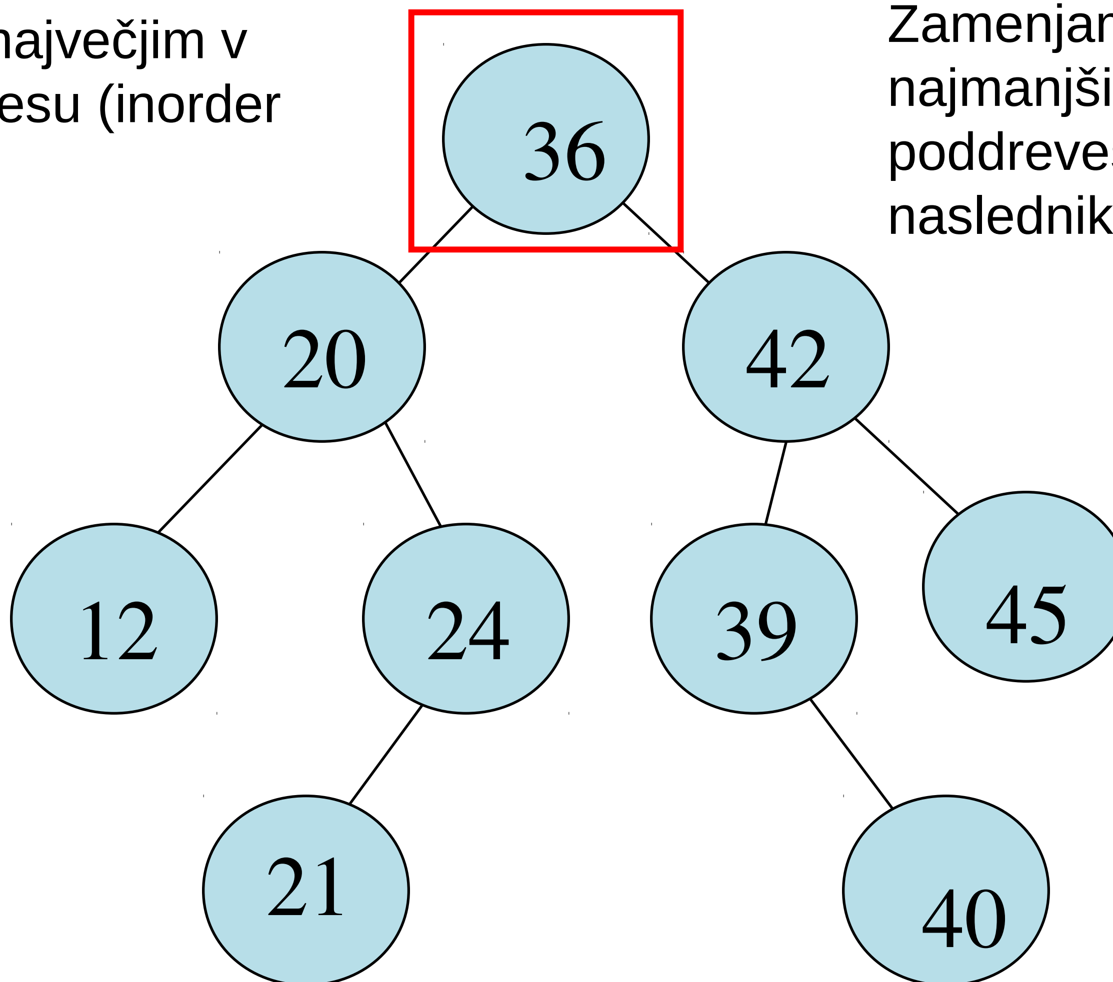
- Brisanje lista
  - Kazalce na tako vozlišče (v staršu) moramo spremeniti na NULL
- Brisanje vozlišča z enim poddrevesom
  - Kazalec na tako vozlišče (v starševskem vozlišču) moramo spremeniti na to (neprazno) poddrevo
- Brisanje vozlišča z dvema nepraznima poddrevesoma

Lahki primeri



“Težek” primer

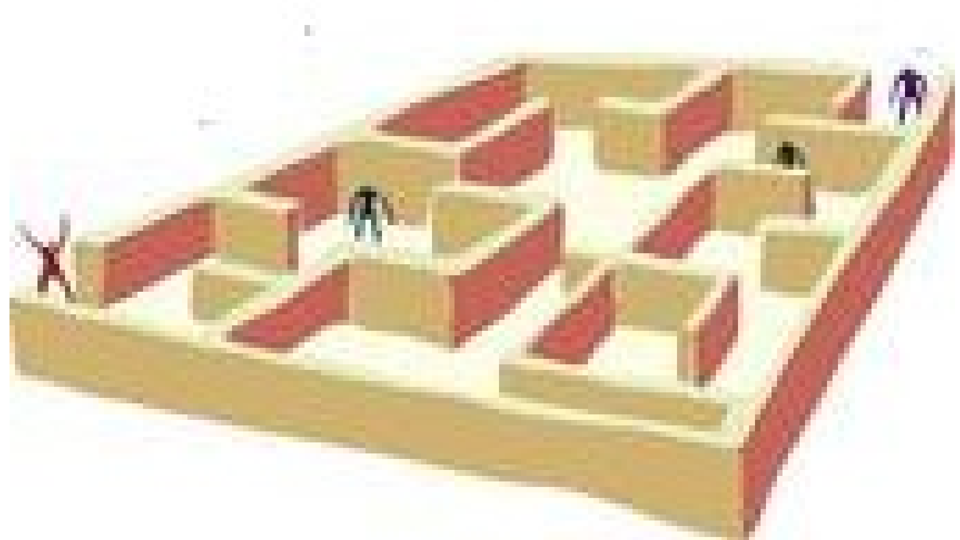
Zamenjamo z največjim v levem poddrevesu (inorder predhodnik)



Zamenjamo z najmanjšim v desnem poddrevesu (inorder naslednik)

Nekatere uporabe dreves

- Iskanje podatkov
- Prednostne vrste
- Odločitveni sistemi
- Hierarhije
- Grafika, igre



Binarno drevo: demo

Program prikazuje izgradnjo binarnega drevesa v jeziku C.
Uporablja dinamično alikacijo pomnilnika, kazalce in rekurzijo.

V drevo vstavljamo naključne vrednosti
Na koncu z rekurzijo izpišemo urejeno vsebino drevesa.

Spremeni program tako, da bo izpisal vsebino drevesa v padajočem vrstnem redu!

Primer v Javi:
Program prebere vhodno tekstovno datoteko in besede uvrsti v binarno drevo. Besede nato urejeno po abecedi izpiše v izhodno datoteko. Imeni obeh datotek moramo navesti na začetku.

Program iz didaktičnih razlogov ne uporablja javanskih kolekcij.

Drugi tipi dreves

- Binarna iskalna drevesa
  - urejene vrednosti
- Kopica (Heap)
  - Urejena z drugačnim algoritmom
- AVL in rdeče črna drevesa
  - binarna iskalna drevesa, ki ostanejo uravnovešena
- Poševna drevesa
- B drevesa

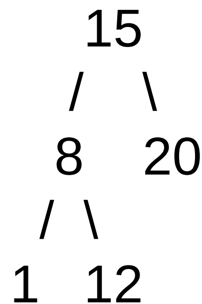
WEB

Binarna iskalna drevesa

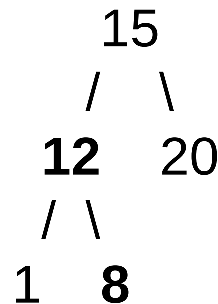
- Binarna iskalna drevesa vsebujejo primerljive objekte.
- Pri vsakem vozlišču so vse vrednosti v levem poddrevesu **manjše** od vrednosti v vozlišču. Vse vrednosti v desnem poddrevesu pa so **večje** od vrednosti v vozlišču.

- Zato je vsako poddrevo spet binarno iskalno drevo (BST, Binary Search Tree)

Je BST

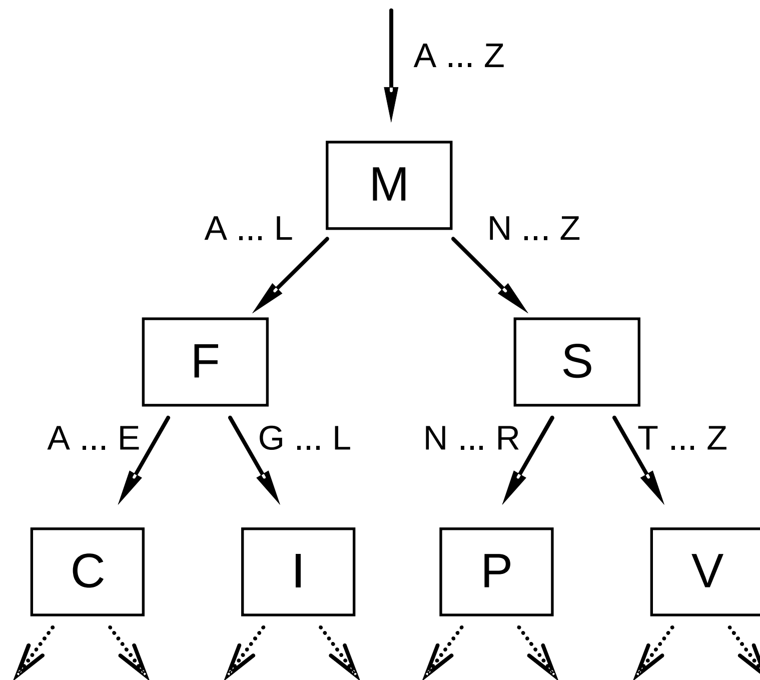


Je binarno, , **NI** pa BST



Še o binarnih iskalnih drevesih

Binarna iskalna drevesa združujejo dobre lastnosti urejenih polj za hitro iskanje in povezanih seznamov za brisanje in dodajanje vrednosti.



Dodajanje vozla

```
private TreeNode add (TreeNode root, Object value) {
    if (node == null)
        node = new TreeNode(value);
   else {
       int  diff = ((Comparable<Object>)value).compareTo
                                                  (root.getValue());
       if (diff < 0)
            root.setLeft (add (root.getLeft(), value));
       else // if (diff > 0)
           root.setRight (add (root.getRight(), value));
   }
   return node;
}
```

*Če vrednost v drevesu že obstaja****, ne*** *dodamo novega vozlišča*

Binarna drevesa in rekurzivni “contains”

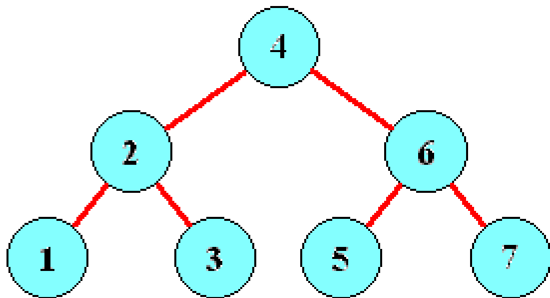
```
// root refers to a BST; the nodes hold Strings

private boolean contains (TreeNode root, String target) {
   if (root == null)
      return false;

   int diff = target.compareTo ((String) root .getValue ());

   if (diff == 0)
        return true;
   else if (diff < 0)
       return contains (root .getLeft (), target);
   else  // if (diff > 0)
       return contains (root .getRight (), target);
}
```

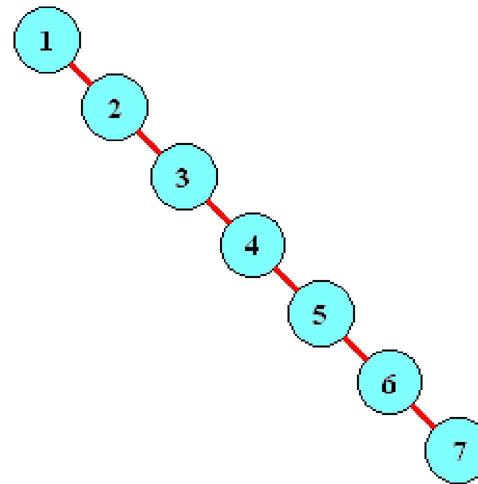
Binarna drevesa in iterativni “contains”

```
private boolean contains (TreeNode root, String target)
{
  TreeNode node = root;

  while ( node != null )
  {
    int diff = target.compareTo (node.getValue ());

    if (diff == 0)
       return true;
    else if (diff < 0)
       node = node.getLeft ();
    else // if diff > 0
       node = node.getRight ();
  }
  return false;
}
```

Algoritmi in binarna drevesa

- Če je drevo uravnovešeno, potrebujejo metode *add*, *remove*, in *contains* $O(\log n)$ časa, pri čemer je *n* število vozlišč v drevesu.
- Če vozlišča dodajamo naključno, kahko binarno iskalno drevo degenerira v približno linearno obliko.

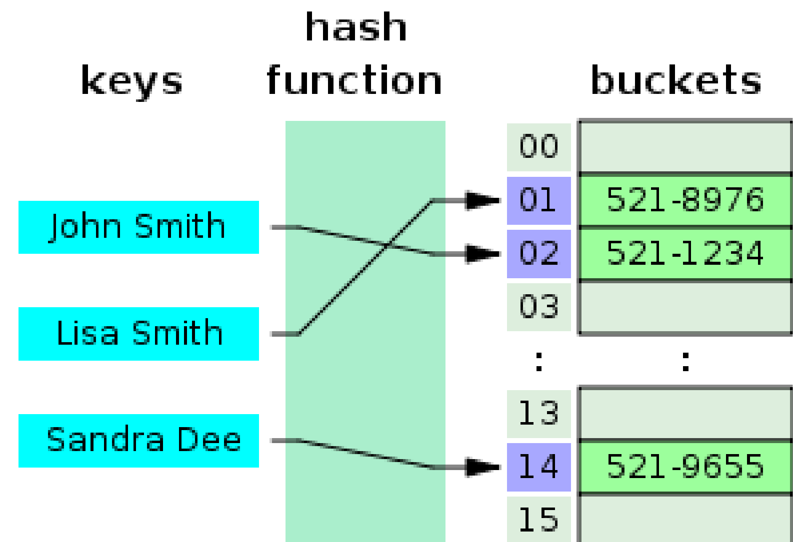




- Z bolj sofisticiranimi algoritmi vzdržujemo uravnovešenost drevesa.

Razpršene tabele (Hash tables)

- Vzamemo ključ in uporabimo funkcijo
- f(key) = hash value
- pomnimo podatke ali objekte na osnovi "hash" vrednosti
- Urejanje O(N), dostop O(1) če je hash funkcija popolna in imamo dovolj pomnilnika za tabelo
- Kaj pa, če pride do kolizije? Odgovor

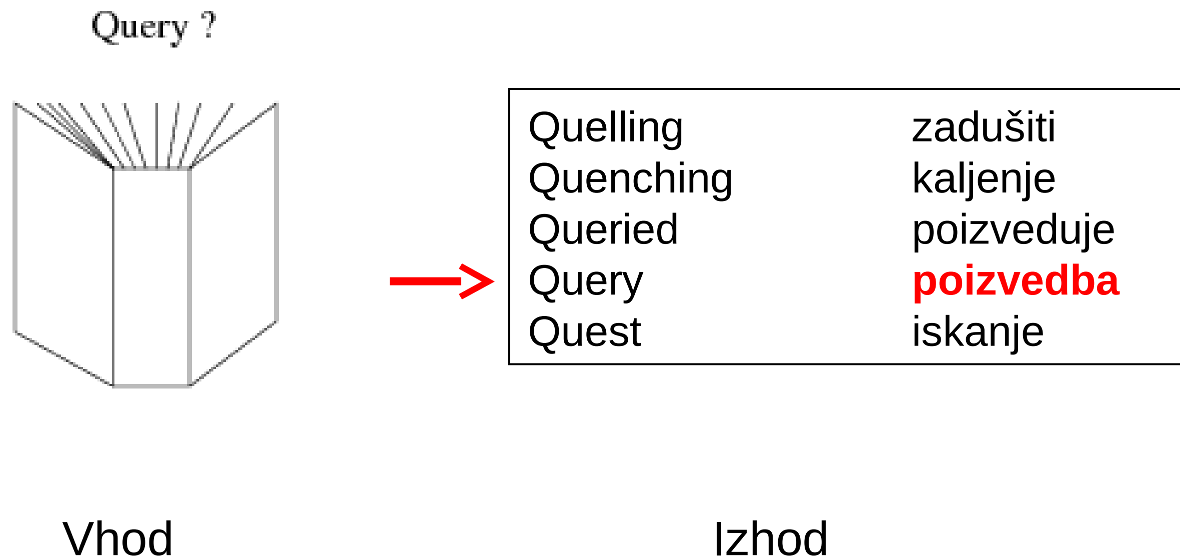


WEB

WEB

Drugi abstraktni podatkovni tipi (ADT)

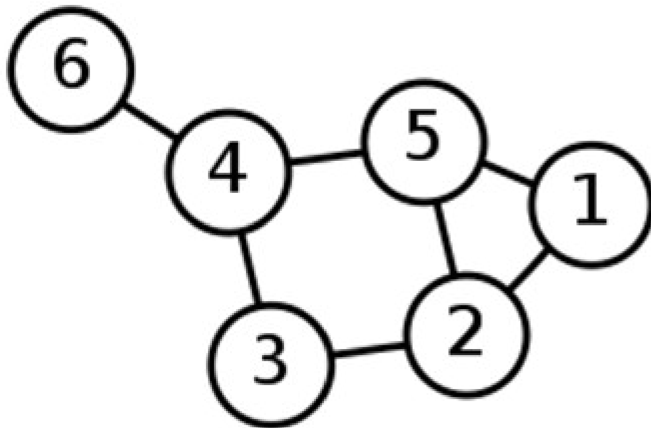
- "Map"
  - Tudi slovar
  - kolekcije elementov s ključem in asociirano vrednostjo
  - podobne hash tabelam, ki so pogosto uporabljene za implementacijo slovarjev



Drugi abstraktni podatkovni tipi (ADT)

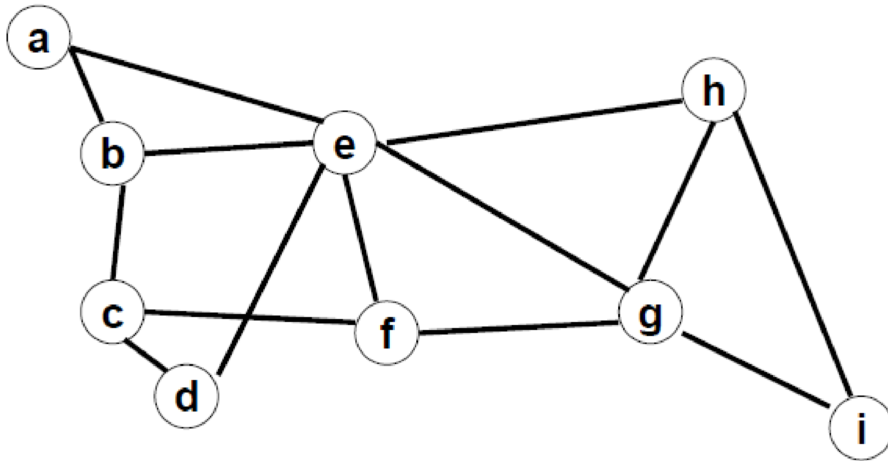
- Grafi
  - Vozlišča z neomejenimi povezavami med vozlišči

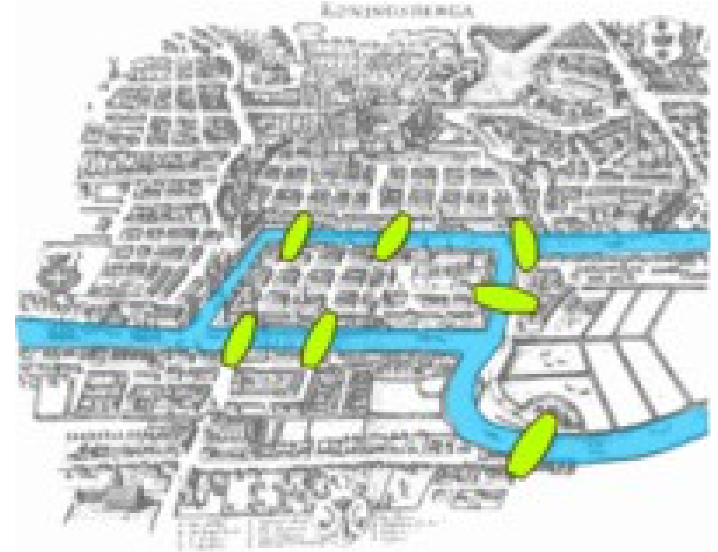
WEB



- Redko posejani vektorji in matrike

Grafi - uporaba



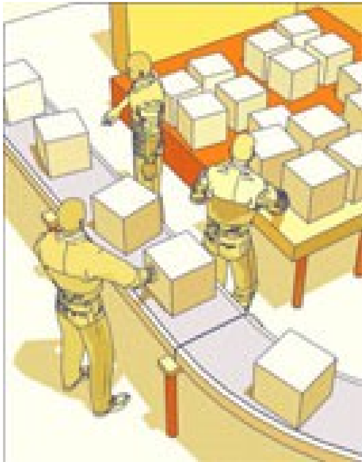


- Simulacije
- Omrežja: telekom, transportna, hidravlična, električna,
- Predstavitev matrik
- Predstavitev sistemov

Vaja

Kakšno strukturo bi uporabil:

- Za simulacijo potnikov, ko vstopajo/izstopajo iz avtobusa
- Za simulacijo tekočega traku
- Za prikaz letalskih povezav letalske družbe
- Za simulacijo knjig, ki jih moramo preurediti na knižnji polici
- Za simulacijo cestnih povezav v Ljubljani



Implementacija podatkovne strukture

- Predstavitev podatkov z vgrajenimi podatkovnimi tipi danega programskega jezika (kot na primer int, double, char, string, array, struct, razredi, kazalci itd.)
- Jezikovna implementacija (koda) algoritmov za operacije

- Če implementiramo podatkovno strukturo v neobjektno usmerjenem jeziku, kot je C, so predstavitev podatkov in operacije ločene.
- Pri objektno usmerjenih jezikih (kot sta Java in C++) so tako podatkovna struktura kot operacije združene skupaj v takoimenovane **objekte**
- Podatkovnemu tipu takih objektov pravimo **razredi** (classes).
- Razredi so "načrti", objekti so **instance** (primerki).