Recursive data structures

(list & tree)

8th lecture

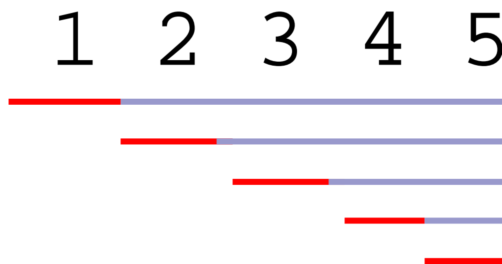
Recursive data structures

- *What is a recursive function?*
  - *A function that uses itself for calculation.*
- *What parts consists a recursive function of?*
  - *Stop condition, divide and conquer part.*
- *Similarly, in recursive data structures.*
- *Structure used for the storage of self data.*

© Branko Kavšek, Jernej Vičič

List as RDS

- List can be: **empty** ali **non-empty**
- If a list is non-empty, it consists of: **head** and **sublist** (that can be empty or non-empty …)
- Example:



List of integer numbers

- Operations on list:
  - Construction and destruction.
  - Add at the beginning or end.
  - Delete at the beginning or end.
  - Search for the first element (head) and rest of the list – tail.
  - Inquiries concerning the length of the list.

© Branko Kavšek, Jernej Vičič

List in JAVA

- Special list is ***empty list*** **`null`**
- Class of non-empty lists of integer numbers:

```
public class Seznam {
  private int  glava = 0;
  private seznam rep = null;

  …
} // Seznam
```

Construction

```
public Seznam(int elt, Seznam lst) {
  this(elt, lst);
} // Seznam
public Seznam(int elt) {
  this(elt, null);
} // Seznam
```

- Construction and destruction.
- Add at the beginning or end.
- Delete at the beginning or end.
- Search for the first element (head) and rest of the list – tail.
- Inquiries concerning the length of the list

Add at the begining or at the end

- Construction and destruction.
- Add at the beginning or end.
- Delete at the beginning or end.
- Search for the first element (head) and rest o the list – tail.
- Inquiries concerning the length of the list

© Branko Kavšek, Jernej Vičič

Class PovezanSeznam

```
public boolean prepend(int elt){
    Seznam tmp = new Seznam(elt);
    tmp.rep = sz;
    sz = tmp;
}
```

Poizvedovanja: glava, rep, dolžina, ...

```
public int  glava() { return glava; }
public Seznam rep() { return rep; }

public int dolzina() {
  if (rep == null) return 1;
  else return (1 + rep.dolzina());
} // dolzina
```

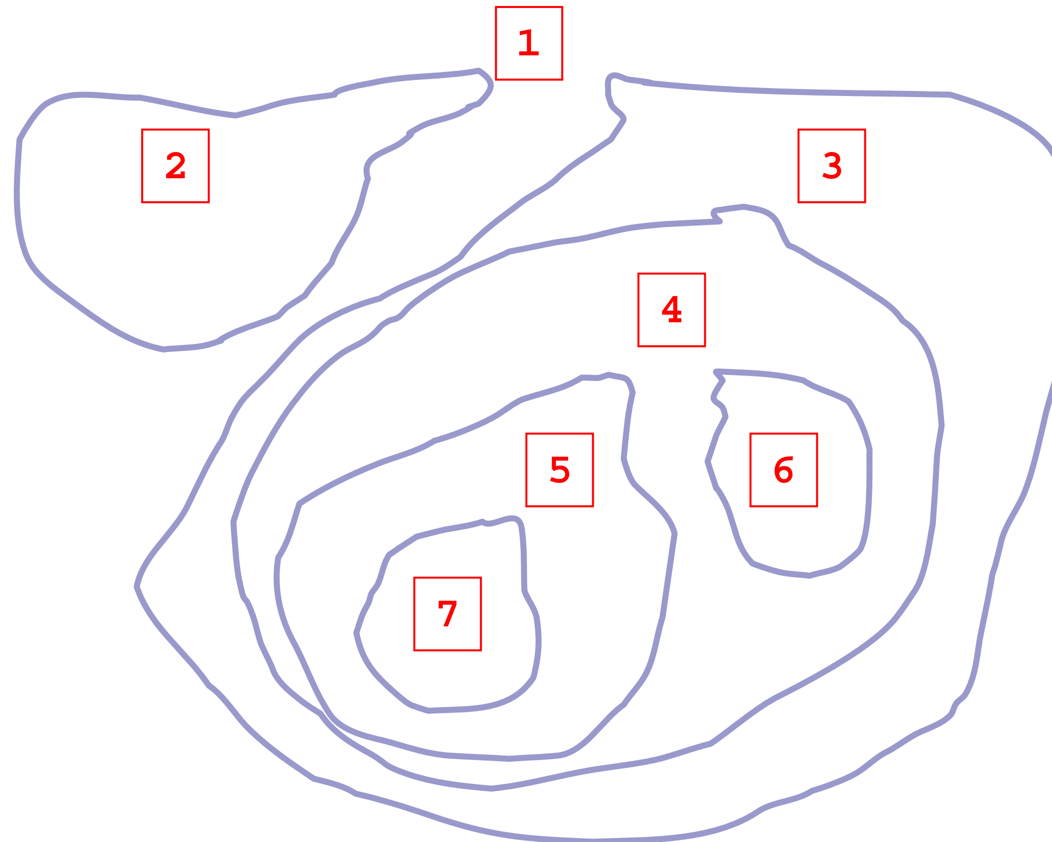
- Construction and destruction.
- Add at the beginning or end.
- Delete at the beginning or end.
- Search for the first element (head) and rest of the list – tail.
- Inquiries concerning the length of the list

Double recursive list

- Double recursive list can be (as a simple list): **empty** or **non-empty**
- If a list is non-empty, it consists of: **head** and two **sublists** (that can be empty or non-empty …)

© Branko Kavšek, Jernej Vičič

Double recursive list



Double recursive list

- Double recursive list is called double (binary) tree.
- Head in the tree is called root and sublists are called left subtree and right subtree.
- There are also k-recursive lists that are similarly called k-trees
  - these trees have (up) to k subtrees
  - and (up to) k-1 elements at the root

Example of a binary tree

- Special case in **empty tree** `null`
- Class of non-empty trees of integer numbers:

```
public class Drevo {
  private int glava = 0;
  private tree levo = null;
  private tree desno = null;
  …
} // Drevo
```


© Branko Kavšek, Jernej Vičič

Construction

```
public Drevo(int elt, Drevo novoLevo, Drevo novoDesno){
  this(elt, novoLevo, novoDesno);
} // Drevo

public Drevo(int elt) {
  this(elt, null, null);
} // Drevo
```

Add

- Add at the begining (prepend)

```
public Drevo prepend(int elt) {
  Drevo novoDrevo = new Drevo(elt, this, null);
  return novoDrevo;
} // prepend
```

- Add at the end (append)

```
public Drevo append(int elt) {
  if (right == null) right= new Drevo(elt);
  else               right= right.append(elt);
  return this;
} // append
```

Queries

- Query: root, subtree

```
public int  value()          { return root;  }
public Drevo levoPodDrevo()  { return left;  }
public Drevo desnoPodDrevo() { return right; }
```

- Query: height

```
public int visina() {
  int levaVisina;
  int desnaVisina;
  if (levo == null) levaVisina = 0;
  else              levaVisina = levo.visina();
  if (desno == null) desnaVisina = 0;
  else               desnaVisina = desno.visina ();
  return (1 + max(levaVisina, desnaVisina));
} // visina
```


© Branko Kavšek, Jernej Vičič

Queries

- Query: search

```
public boolean najdi(int elt) {
  boolean nasel;
  if (glava == elt) return true;
  if (levo == null) nasel = false;
  else nasel = levo.search(elt);
  if(!nasel)
    if (desno != null) nasel = desno.najdi(elt);
  return nasel;
} // najdi
```


© Branko Kavšek, Jernej Vičič

Traversals and prints

- Prvi (*preorder*)

```
public void preorder(void) {
  System.out.println(glava);  // PRE - pred
  if (levo != null) levo.preorder();
  if (desno != null) desno.preorder();
} // preorder
```

- Vmesni (*inorder*)

```
public void inorder(void) {
  if (levo != null) levo.inorder();
 System.out.println(glava); // IN - vmes
  if (desno != null) desno.inorder();
} // inorder
```

- Zadnji (*postorder*) – task nr. 5

Binary search tree

```
public class Drevo {
      int glava;
      Drevo levi, desni;
//metode
}
```

Binary search tree - insert

```
boolean vstavi(int elt){
  if(glava > elt){
    if(levi == null){
      levi = new Drevo();
      levi.glava = elt;
      return(true);
    }
    else{return(levi.vstavi(elt));}
  }
  else{
    if(desni == null){
      desni = new Drevo();
      desni.glava = elt;
      return(true);
    }
    else{return(desni.vstavi(elt));}
  }
}
```

Binary search tree - search

```
boolean isci(int elt){
    if(glava == elt){return(true);}
    else{
      if(glava > elt){
        if(levi == null){
          return(false);
        }
        else{return(levi.isci(elt));}
      }
      else{
        if(desni == null){
          return(false);
        }
        else{
          return(desni.isci(elt));
        }
      }
    }
}
```

Binary search tree - delete

```
boolean brisi(int elt){
  if(glava > elt){
   if(levi == null){return(false);}
   else{
    if(levi.glava == elt){
     //prevec za prvi letnik
     return(true);
    }
    else{return(levi.brisi(elt));}
  }
  else{
   if(desni == null){
    return(false);
   }
   else{
    if(desni.glava == elt){
     //prevec za prvi letnik
     return(true);
    }
    else
     return(desni.brisi(elt));
   }
  }
}
```

At the end

- We have seen:
  - Recursive data structures: *lists, trees*
  - Simple to use: All thet the structure needs to know is:
    - ***About itself*** *and*
    - about its immediate surroundings (successor subtree, …).

Summary

- Recursive data structures
- List
  - construction,
  - add (at the beginning / end)
  - query
  - delete (the beginning / end)
  - List as a queue
  - empty lists
- Double recursive list (tree)
  - Traversals and prints in Trees

© Branko Kavšek, Jernej Vičič