Abstraktni podatkovni tipi

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

- seznam (list)
- vrsta (queue)
- sklad (stack)
- preslikava (map)
- množica (set)

- DREVO (angl. *tree*) – zbirka vozlišč, hierarhično urejenih z relacijo oče-sin
- SLOVAR (angl. *dictionary*) – poseben primer *ADT množica*, ki omogoča samo vstavljanje, brisanje in iskanje elementa

Drevo (*tree*)

Drevo je zbirka hierarhično urejenih vozlišč (*nodes*).

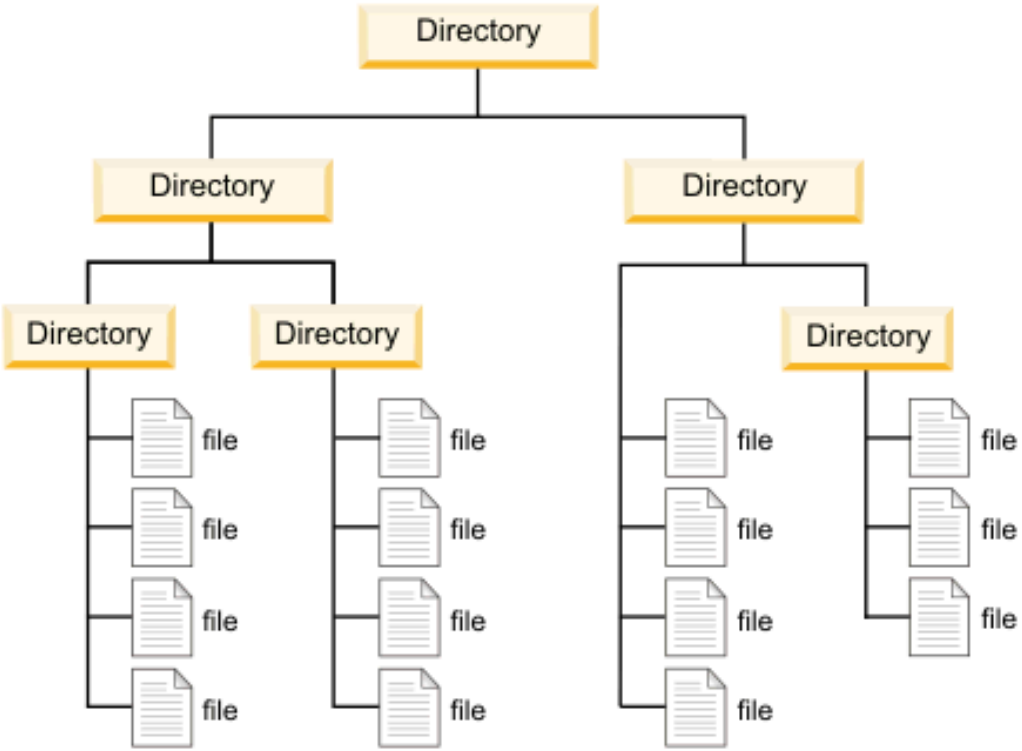
Vozlišča so urejena z relacijo oče-sin (*parent-child*).

Vsako vozlišče ima:

- oznako (vsebino) (*label*)
- enega očeta (razen korena (*root*) drevesa)
- nič ali več sinov (*children*)
- če obstaja sin, potem je to levi sin (izjema so binarna drevesa)

Primeri dreves

Primer drevesne strukture:
datotečni sistem



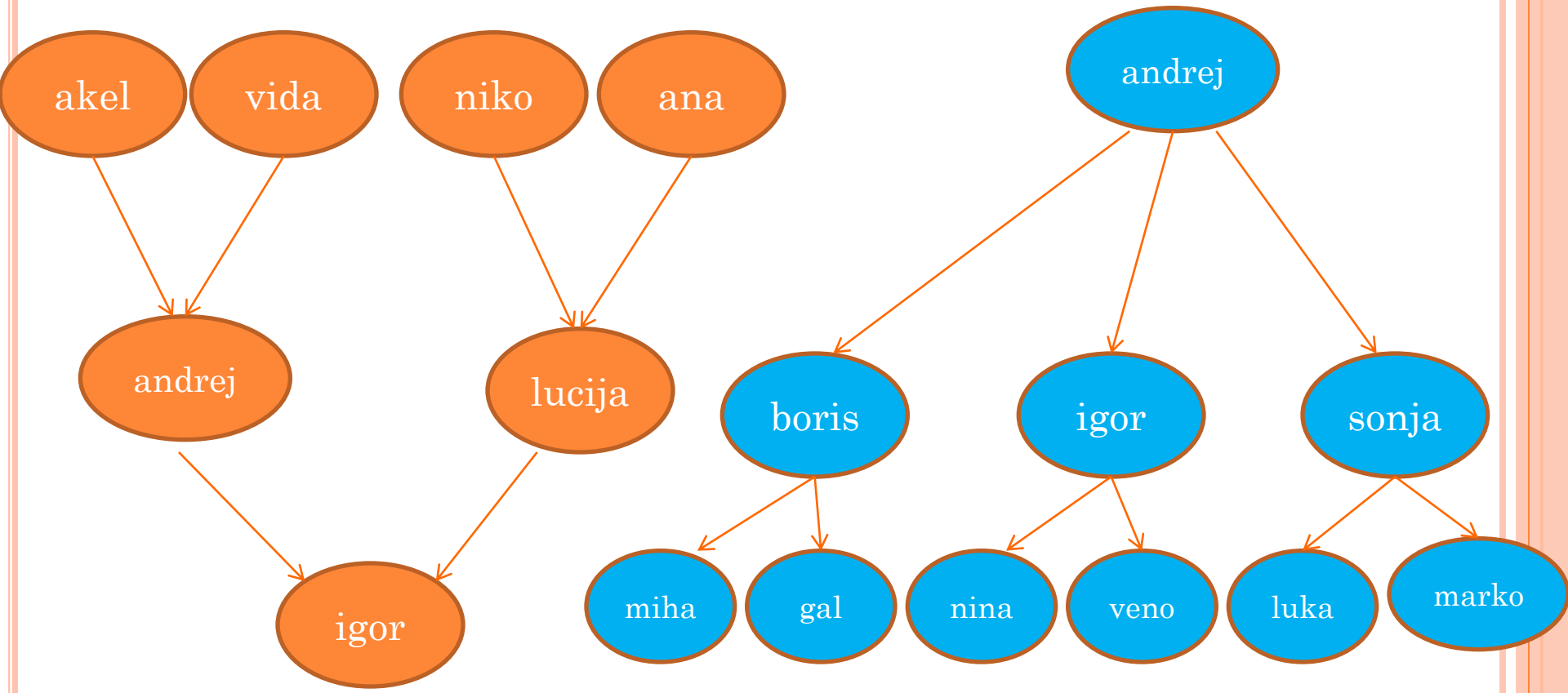
Primeri dreves

Primeri dreves

Družinsko drevo

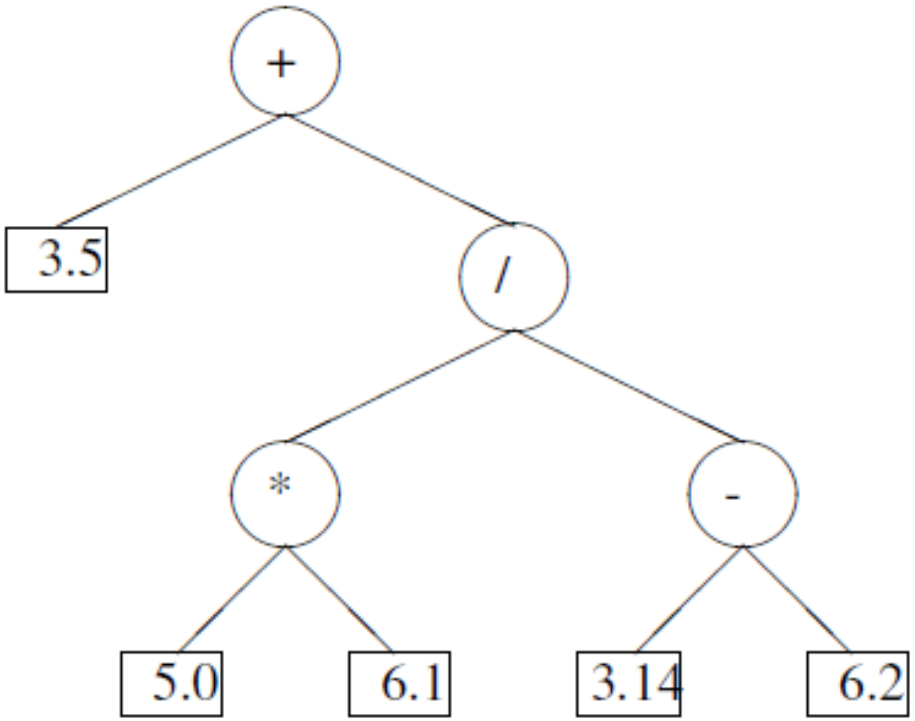
prednikov

naslednikov

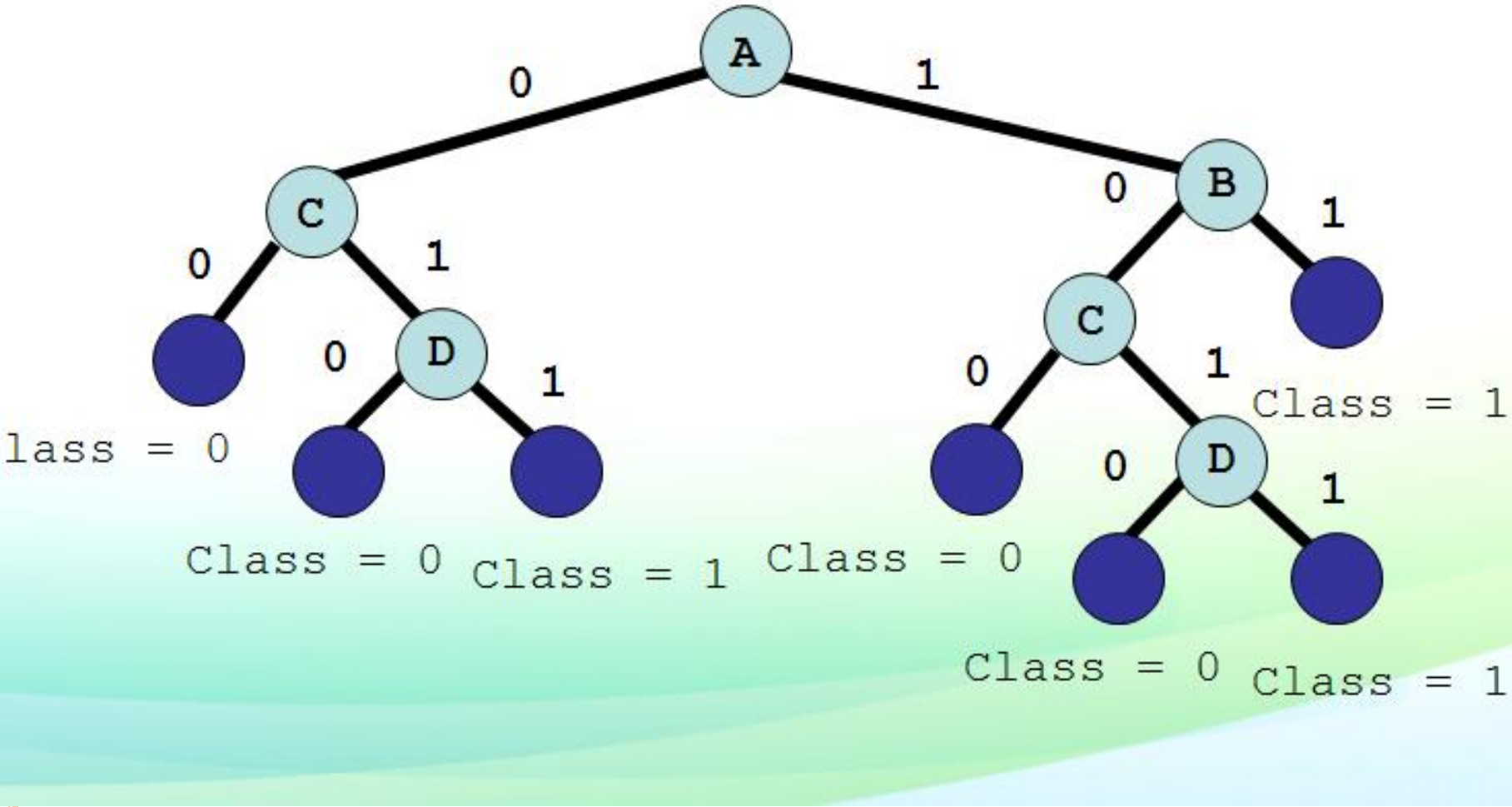


Izrazna drevesa

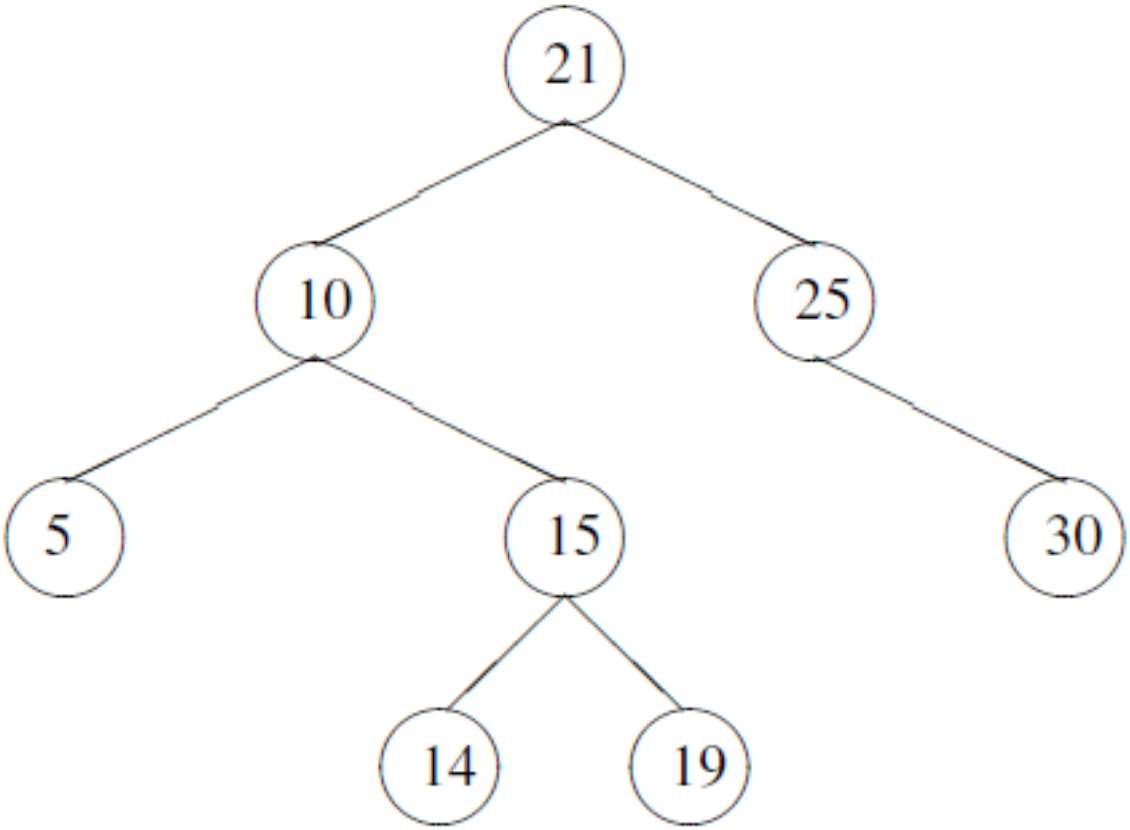
izraz 3.5 + 5.0*6.1/(3.14-6.2)



Odločitvena drevesa



Iskalna drevesa



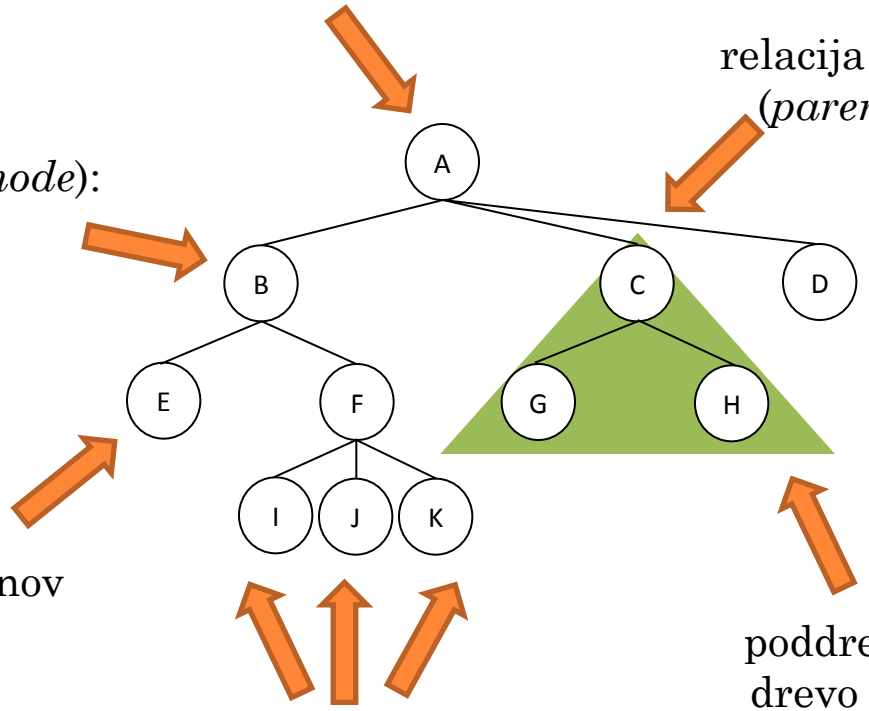
DREVO

Terminologija:

koren (*root*): vozlišče brez očeta

relacija oče-sin (*parent-child*)

notranje vozlišče (*internal node*): vozlišče z vsaj enim sinom



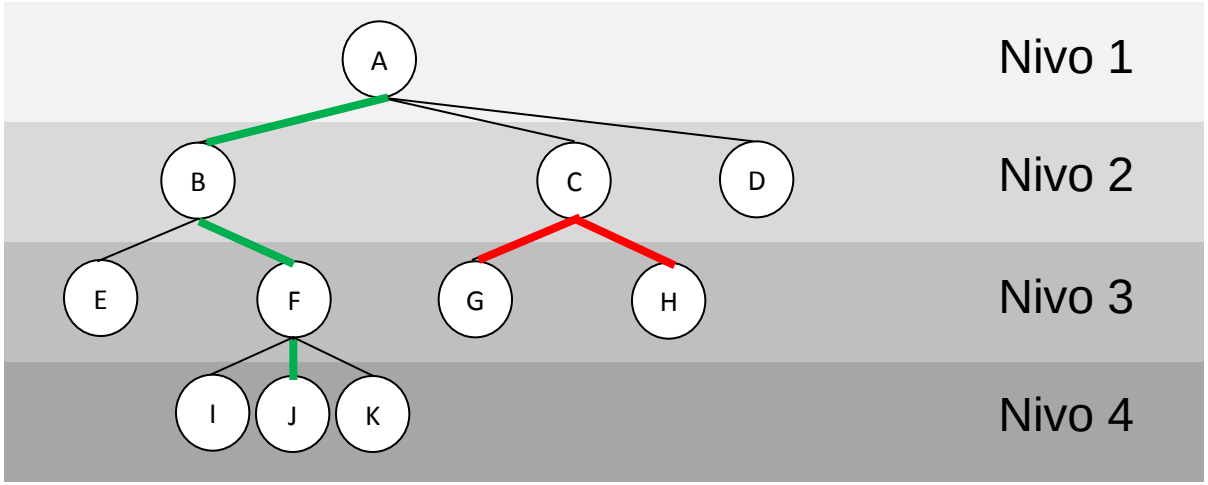
list (*leaf*): vozlišče brez sinov

poddrevo (*subtree*): drevo s korenom v danem vozlišču

bratje (*siblings*): sinovi istega očeta

DREVO

Terminologija:



stopnja (*degree*) **vozlišča**: število sinov vozlišča (npr. stopnja vozlišča C je 2)

stopnja drevesa: največje št. sinov nekega vozlišča (npr. stopnja drevesa na sliki je 3)

pot (*path*): zaporedje $n_1,n_2,\ldots,n_k$, pri čemer velja za vsak $1 \leq i < k$, da je n_i oče n_{i+1} (npr. A, B, F, J)

($j > i$): n_i je prednik (*ancestor*) vozlišč n_j, oz. n_j so nasledniki (*descendants*) vozlišč n_i
Dolžina poti (*path length*) je enaka številu vozlišč na poti.

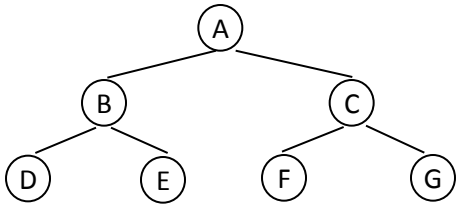
nivo (*level*) **vozlišča**: dolžina poti od korena do vozlišča (npr. nivo vozlišča E je 3)

višina (*height*) **drevesa**: dolžina najdaljše poti od korena do lista (npr. za zgornje drevo je 4)

DREVO

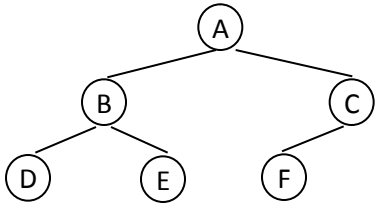
Terminologija:

poravnano ali uravnovešeno (*balanced*) drevo:

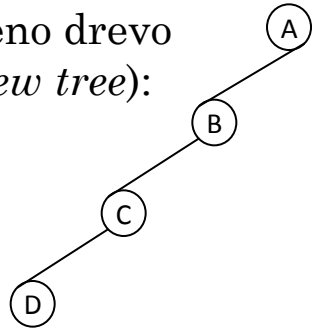


Stopnja = k → višina = $\log_k n$

levo poravnano drevo:



izrojeno drevo (*skew tree*):



Stopnja = 1, višina = n

PRIMER: VIŠINA BINARNEGA DREVESA

Rekurzivna definicija binarnega drevesa:

1. Prazno binarno drevo je brez vozlišč.
2. Binarno drevo ima koren in dve binarni poddrevesi.

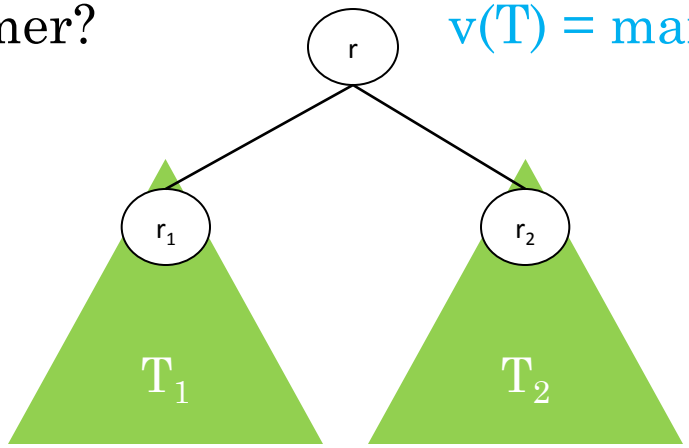
Kaj je rekurzijska spremenljivka? (T = koren drevesa)

Kaj mi pomaga rešitev manjšega problema?

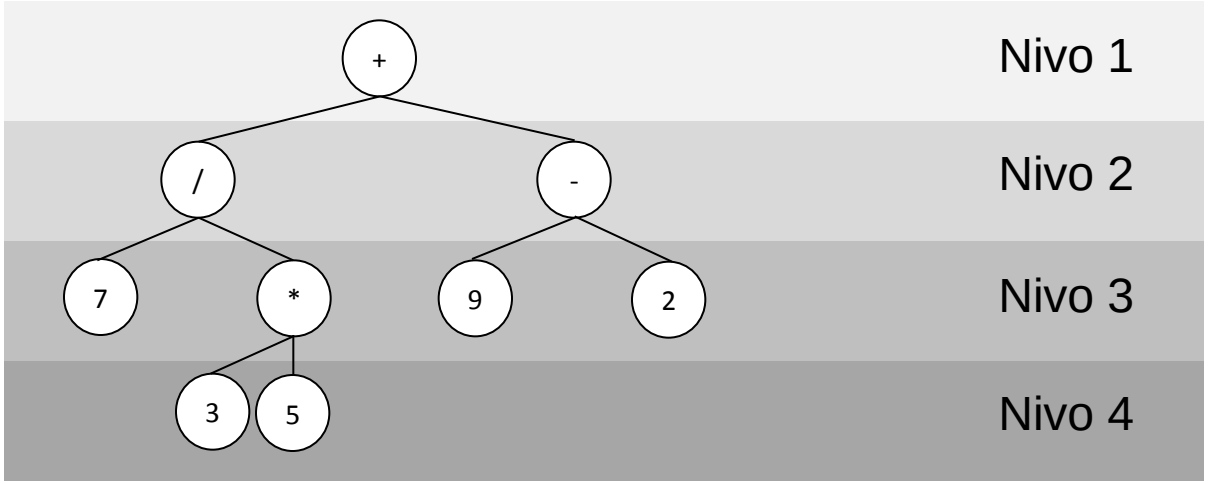
Iz višin obeh poddreves lahko izračunam višino drevesa.

Kaj je robni pogoj? (T = null, v(null) = 0)

Kaj je splošni primer? v(T) = max(v(T1),v(T2)) + 1



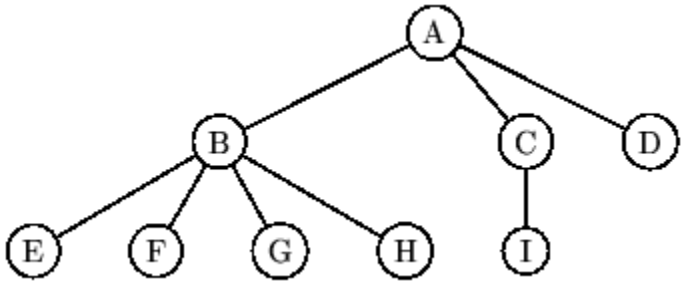
Primer: Izračun izraza



Izrazno drevo: v notranjih vozliščih operacije, v listih števila

7/(3*5)+(9-2)

Kaj je rekurzijska spremenljivka? (T = koren drevesa)
Kaj mi pomaga rešitev manjšega problema?
Iz vrednosti obeh poddreves lahko izračunam vrednost drevesa.
Kaj je robni pogoj? (T = Število, v(Število) = Število)
Kaj je splošni primer? v(operator) = v(T1) operator v(T2)



ADT DREVO
(angl. *tree*)

ADT TREE

Operacije definirane za ADT TREE:

- PARENT(n, T) – vrne očeta vozlišča *n* v drevesu *T*
- LEFTMOST_CHILD(n, T) – vrne najbolj levega sina vozlišča *n*
- RIGHT_SIBLING(n, T) – vrne desnega brata vozlišča *n*
- LABEL(n, T) – vrne oznako vozlišča *n*
- ROOT(T) – vrne koren drevesa *T*
- MAKENULL(T) – naredi prazno drevo *T*
- CREATE(r, v, T1,…, Ti) – generira drevo s korenom *r* z oznako *v* ter s stopnjo *i* s poddrevesi *T1*, …, *Ti*

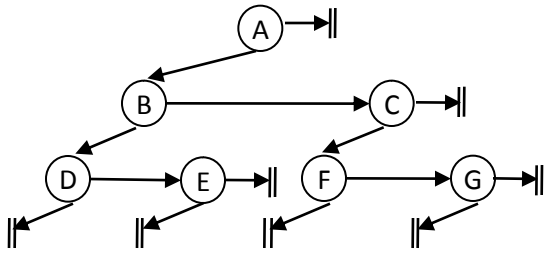
ADT TREE

V javi lahko definiramo abstraktni razred:

```
public abstract class Tree {
    public abstract TreeNode parent(TreeNode n) ;
    public abstract TreeNode leftmostChild(TreeNode n) ;
    public abstract TreeNode rightSibling(TreeNode n) ;
    public abstract void create(Object v, ListLinked subTrees) ;
    public abstract TreeNode root() ;
    public abstract void makenull() ;
    public Object label(TreeNode n) {
       return n.element ;
    }
} // class Tree
```

PRIMER: VIŠINA DREVESA

- RIGHT_SIBLING in LEFTMOST_CHILD omogočata dostop do levega otroka in desnega brata
- Rekurzija gre lahko v ti dve smeri
- Robni primer: prazno poddrevo

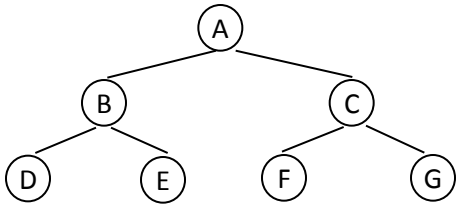



```
public int height(TreeNode n) {
   if (n==null)
      return 0 ;
   else
         return Math.max(height(leftmostChild(n))+1,
                         height(rightSibling(n))) ;
} // height
```

Če sta operaciji O(1),
potem računanje višine O(n)

Obhod drevesa

Obhod (traversal) drevesa lahko opravimo na več načinov:



- **premi (*preorder*)**: izpišemo oznako korena pred oznakami poddreves (za drevo na sliki dobimo izpis a, b, d, e, c, f, g)
- **obratni (*postorder*):** izpišemo najprej oznake vozlišč vseh poddreves in zatem oznako korena (za drevo na sliki dobimo izpis d, e, b, f, g, c, a)
- **vmesni (*inorder*):** izpišemo najprej oznake vozlišč najbolj levega poddrevesa, zatem oznako korena in zatem oznake vozlišč vseh ostalih poddreves korena (za drevo na sliki dobimo izpis d, b, e, a, f, c, g)
- **nivojski**: izpišemo najprej vsa vozlišča na 1. nivoju, zatem na 2. nivoju in tako naprej (za drevo na sliki dobimo izpis a, b, c, d, e, f, g)

PREMI OBHOD

```
public void preorder(TreeNode r) {
   if (r != null) {
      // najprej izpisemo oznako korena
      r.writeLabel() ; System.out.print(",␣") ;
      TreeNode n = leftmostChild(r) ;
      while (n != null) {
         preorder(n) ;
         n = rightSibling(n) ;
      } // while
   } // if
} // preorder
```

Obratni obhod

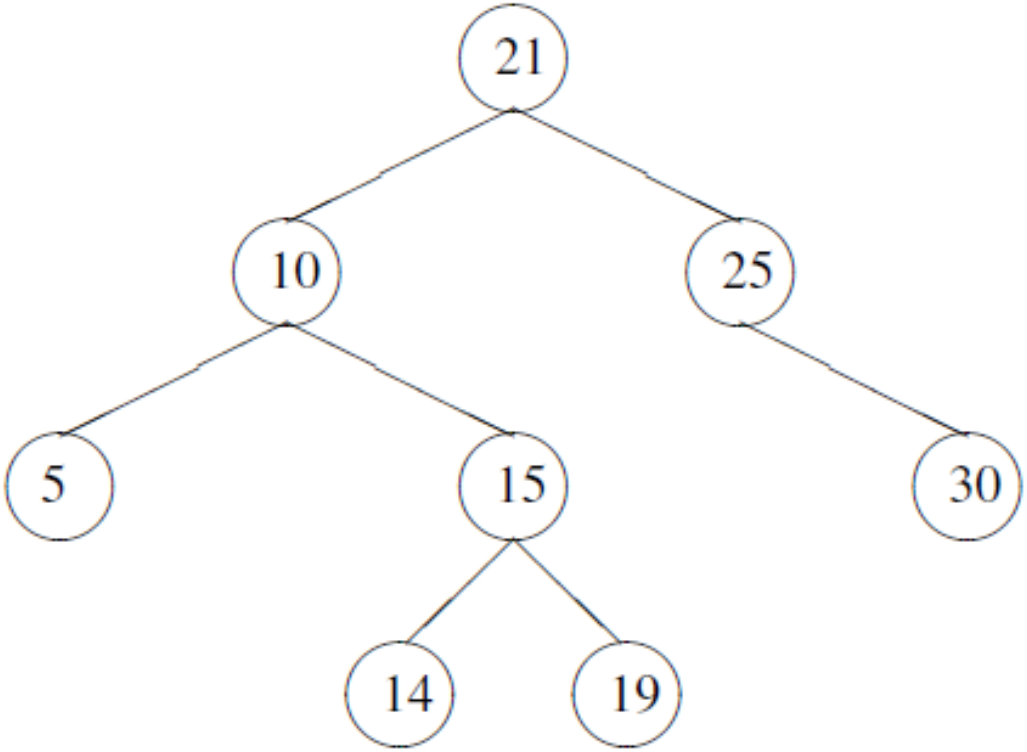
```
public void postorder(TreeNode r) {
   if (r != null) {
      TreeNode n = leftmostChild(r) ;
      while (n != null) {
         postorder(n) ;
         n = rightSibling(n) ;
      } // while
      // nazadnje izpisemo oznako korena
      r.writeLabel() ; System.out.print(",␣") ;
   } // if
} // postorder
```

VMESNI OBHOD

```
public void inorder(TreeNode r) {
   if (r != null) {
      TreeNode n = leftmostChild(r) ;
      inorder(n) ;
      // za levim poddrevesom izpisemo oznako korena
      r.writeLabel() ; System.out.print(",␣") ;
      n = rightSibling(n) ;
      while (n != null) {
         inorder(n) ;
         n = rightSibling(n) ;
      } // while
   } // if
} // inorder
```

Vmesni obhod

Vmesni obhod je najbolj zanimiv za binarna iskalna drevesa



Obhod: 5, 10, 14, 15, 19, 21, 25, 30

Obhod po nivojih

Po nivojih izpisujemo tako, da
povečujemo nivo
dokler se še kaj izpiše.

```
public void printByLevel() {
   int l = 1 ;
   while (printLevel(l,root())) {
      l++ ;
      System.out.println() ;
   }
} // printByLevel
```

Rekurzija teče po številu nivoja in po korenu poddrevesa.

Obhod po nivojih

```
private boolean printLevel(int l, TreeNode r) {
    if (r==null)
       return false ;
    else if (l==1) {
       r.writeLabel(); System.out.print(",␣") ;
       return true ;
     }
     else {
       TreeNode n = leftmostChild(r) ;
       boolean existsLevel = false ;
       while (n !=null) {
          // izpis nivoja je mozen v vsaj enem poddrevesu
          existsLevel |= printLevel(l−1, n) ;
          n = rightSibling(n) ;
       }
       return existsLevel ;
    }
} // printLevel
```

IZZIV: PO NIVOJIH LINEARNO

- Namig: shrani nivo v pomnilniku
- Kateri ADT?

ADT TREE: IMPLEMENTACIJA

- PARENT(n, T) – vrne očeta vozlišča *n* v drevesu *T*
- LEFTMOST_CHILD(n, T) – vrne najbolj levega sina vozlišča *n*
- RIGHT_SIBLING(n, T) – vrne desnega brata vozlišča *n*
- LABEL(n, T) – vrne oznako vozlišča *n*
- ROOT(T) – vrne koren drevesa *T*
- MAKENULL(T) – naredi prazno drevo *T*
- CREATE(r, v, T1,…, Ti) – generira drevo s korenom *r* z oznako *v* ter s stopnjo *i* s poddrevesi *T1*, …, *Ti*

IMPLEMENTACIJA:

- *s poljem*
- *s kazalci*