Programiranje I – RIN
Računalništvo I – MA

Tree

7th lecture

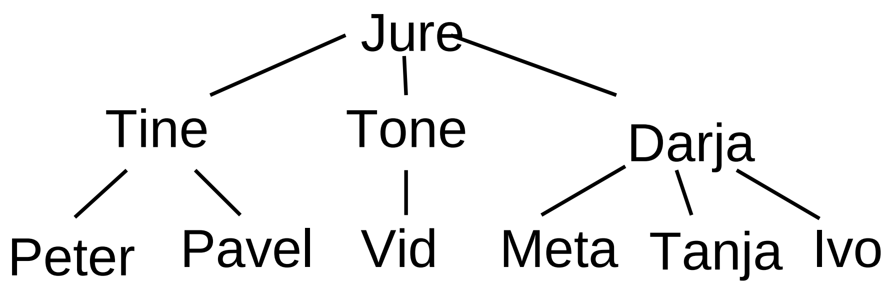
Overview

The tree data structure is non-linear;

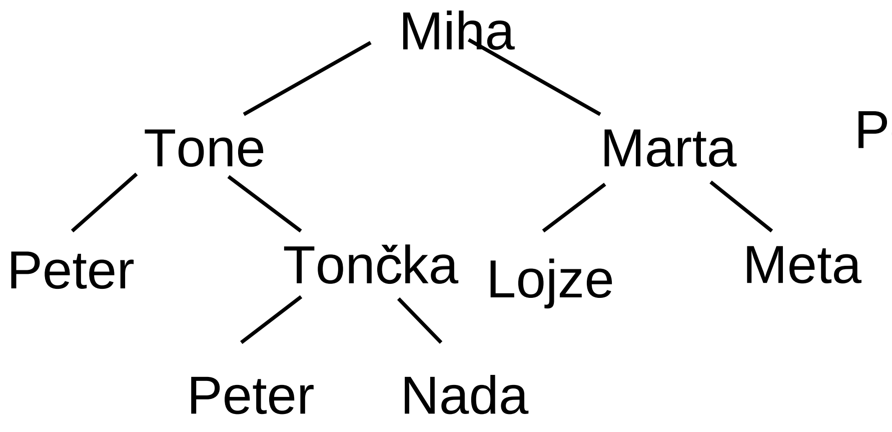
A node may have several successors, but at most one ancestor;

For example, two forms of family tree:

Descendants tree:

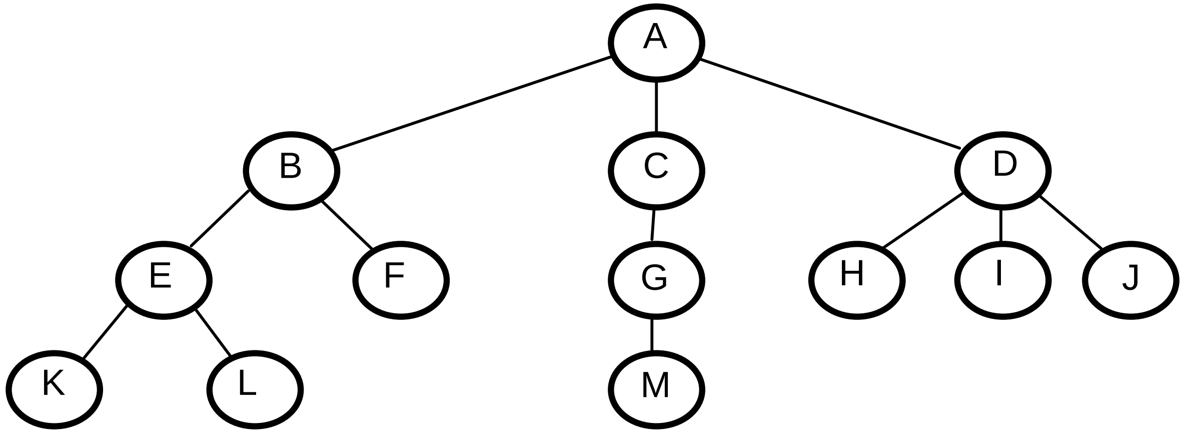


Ancestor tree:



Predstavitev dreves

- Običajni grafični zapis in predstava:



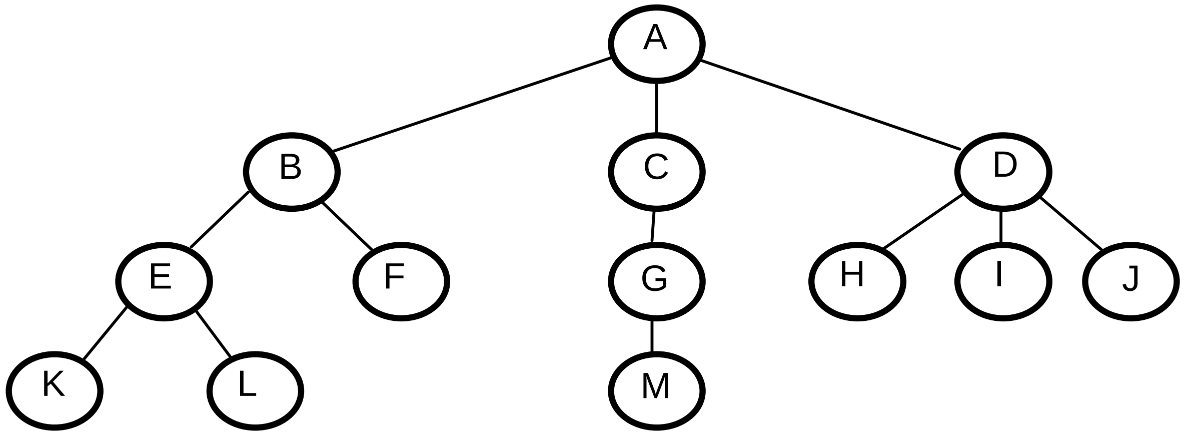
- Drevo stoji “na glavi” oz. rišemo njegove “korenine”

Terminology

- **Node**: element of a tree; it has some data and information about the subtrees.
- **Leaves (terminal nodes):** nodes without descendants.
- **Root**: the top node, node without a predecessor.
- **Internal node:** every node except leaves.
- **Offspring(or descendant):** root node of a subtree, v :: = sin (v);
- **Father (or predecessor):** the v:: = father (v);
- **Siblings:** root subtree of the same node, the node with the same father
- **Ancestors:** each node on the path from the root to a node;

Node

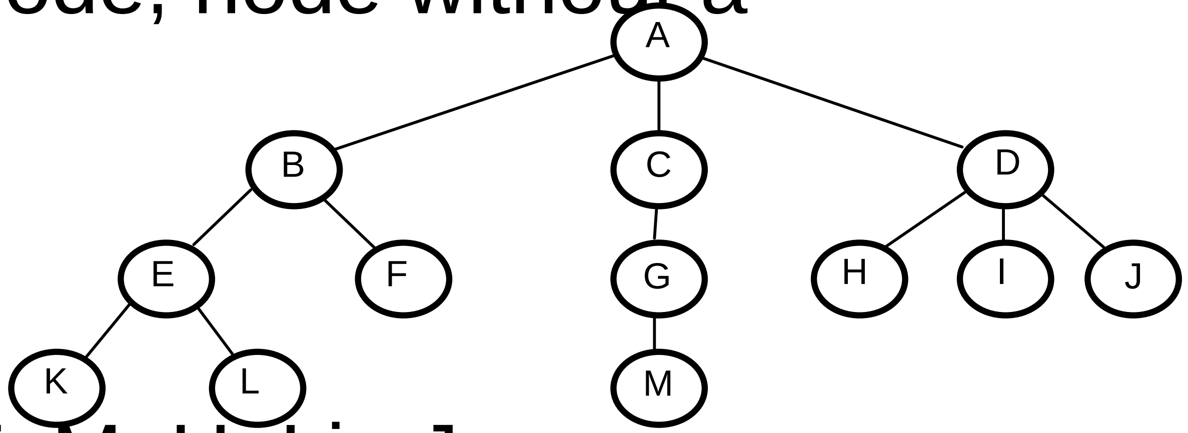
- All "circles". The nodes store data and maintain the structure.



- Nodes: A, B, C, D, ...

Leaves / root

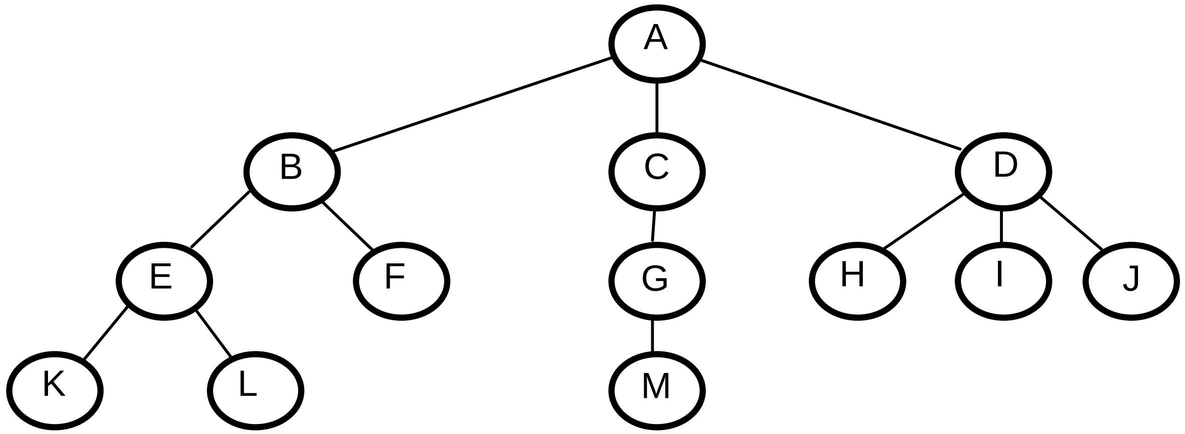
- **Leaves (terminal nodes):** nodes without descendants
- **Root**: the top node, node without a predecessor.



- Leaves: K, L, F, M, H, I in J
- Root: A

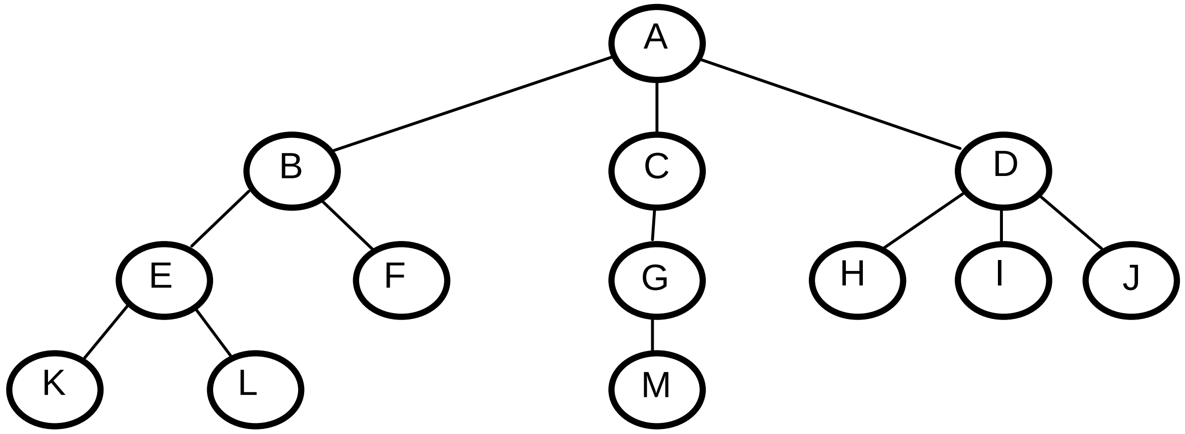
Offspring (sin) / father (oče)

- Node B is the son of node A and father of nodes E and F.
- Node A does not have a father (it is the root) and it has offsprings B, C and D.



Siblings (Bratje), ancestors (predniki)

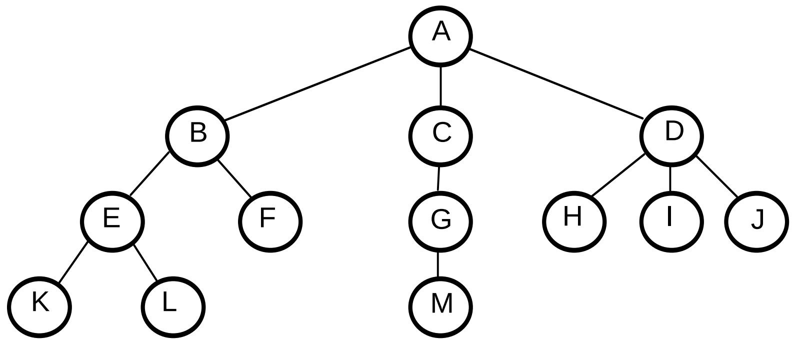
- Nodes B, C and D are siblings (their father is A)
- Predecessors of the node L are E, B and A.



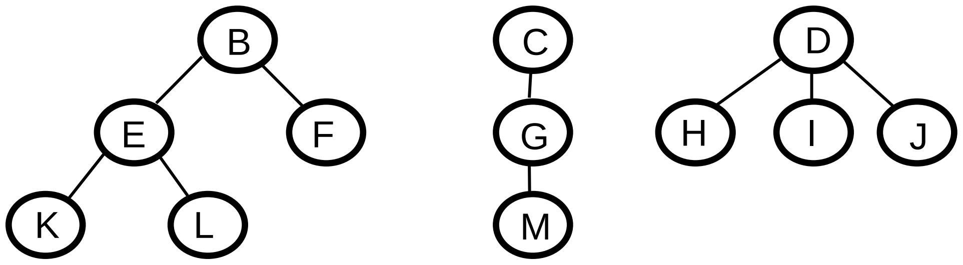
Concepts

- **Subtree:** the entire tree, starting from node.
- **Node degree:** given by the number of subtrees (the number of direct successors)**;**
- **Degree of the tree:** the highest degree of its nodes.
- **Level (depth) of a node:** root is on level (depth) 1. If a father is on depth n, the son is on n +1.
- **Height of the tree:** defined by the node with the highest level.
- **Forest:** a set of disjoint trees.
- **Ordered tree:** the order of subtree at each node is given and is important.

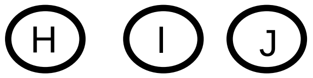
Subtree



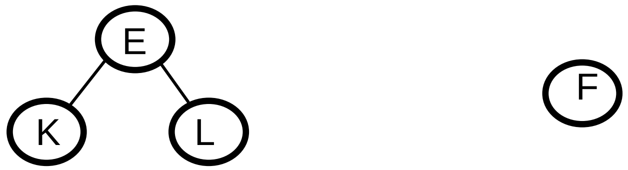
Subtrees of the node A



Subtrees of the node D

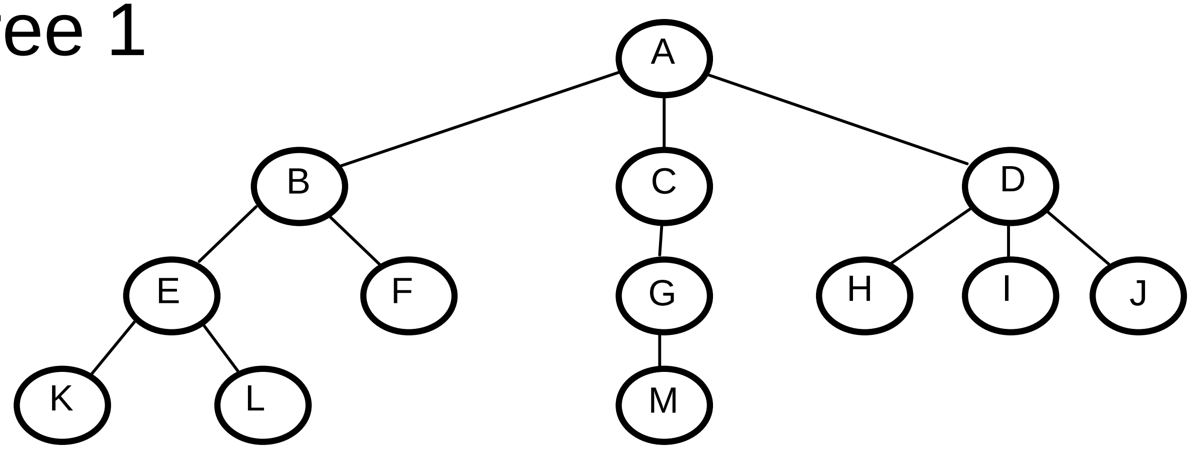


Subtrees of the node B



Degree of a node/ degree of the tree

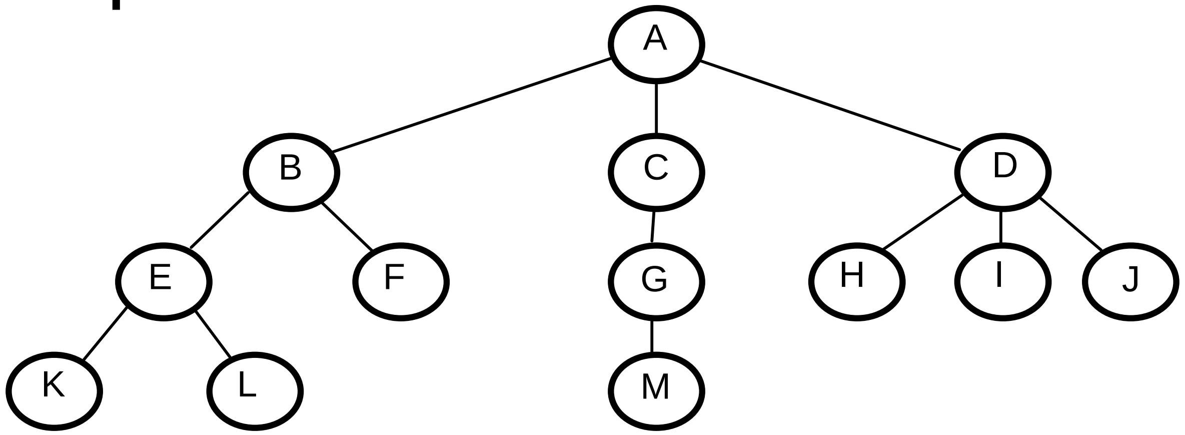
- Node A has degree 3
- Node K has degree 0 (all leaves have degree 0)
- Node C has degree 1



- Tree has degree 3 (max degree of the nodes is 3) – triplet tree

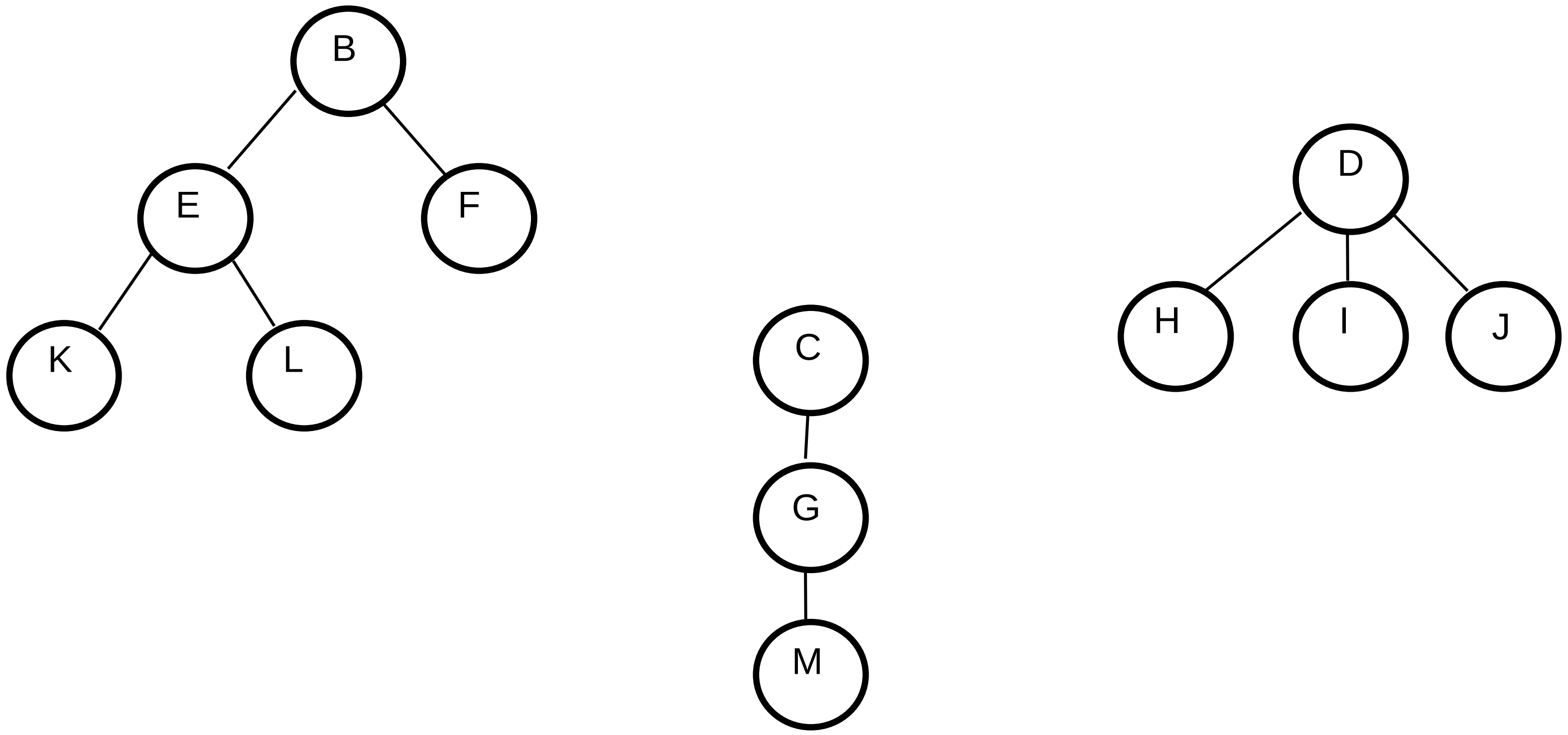
Level (depth) of a node / height of the tree

- Node A is at depth 1 (only root is at depth 1).
- Node K is at depth 4.
- Node C is at depth 2.



- The height of the tree is 4.

Forest



About trees

- Definitions:
  - http://en.wikipedia.org/wiki/Tree_%28data_structure%29

Types of trees

- Binary trees
  - Ancestor tree
  - A tree of an arithmetic expression
- 2-3 tree
- AVL tree
- B-tree
- Left-most tree
- ...

Binary tree

- Ordered tree.
- Degree of all nodes is at most two.
- Definition:
  - binary tree is either empty or it consists of special root node, which has a left and right subtree;
  - left and right subtree are again binary trees.

Binary tree

- Definition:
  - binary tree is either empty or it consists of special root node, which has a left and right subtree;
  - left and right subtree are again binary trees.

Data structure binary tree

```
structure BINARY TREE
  declare
      prepare: 0 → binary tree;
      compose: (binary tree, data, binary tree) → binary tree;
      return: binary tree → data;
      left subtree: binary tree → binary tree;
      right subtree: binary tree → binary tree;
      empty: binary tree → {true, false};
```

Data structure binary tree

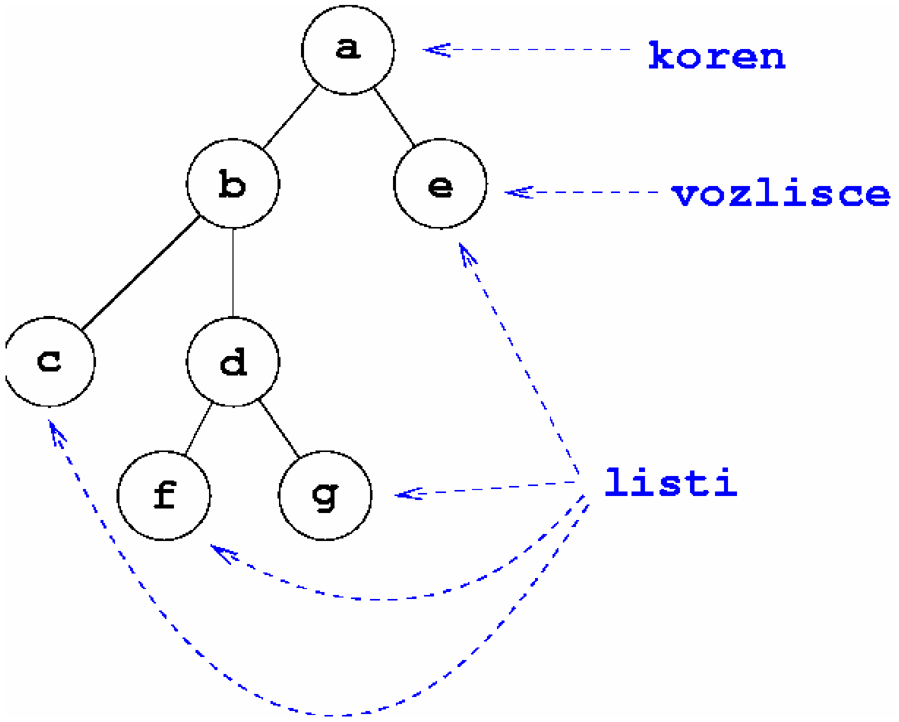
where

left subtree (*prepare*) ::= ERROR;
left subtree(*compose*(l,k,d)) ::= l;
right *subtree*(*prepare*) ::= ERROR;
right subtree(*compose*(l,k,d)) ::= d;
empty(*prepare*) ::= true;
empty(*compose*(l,k,d)) ::= false;
return(*prepare*) ::= ERROR;
return(*compose*(l,k,d)) ::= k;

end;

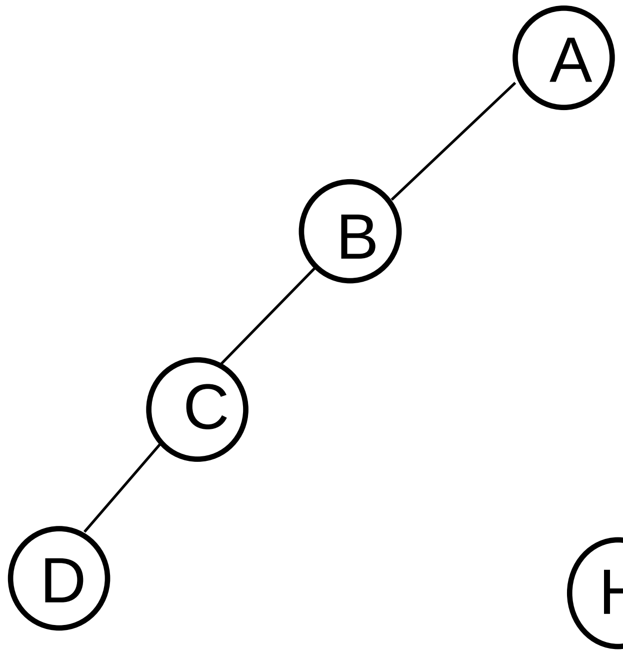
Tasks

- Mirror copy
- Count leaves
- Average tree

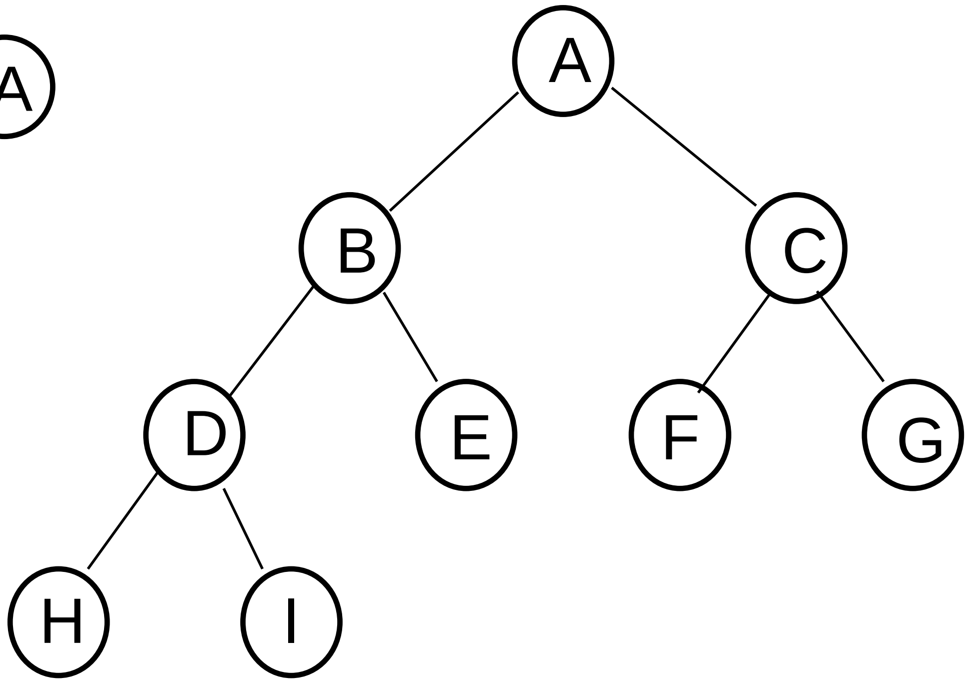


Special forms of binary trees

Degenerated binary tree

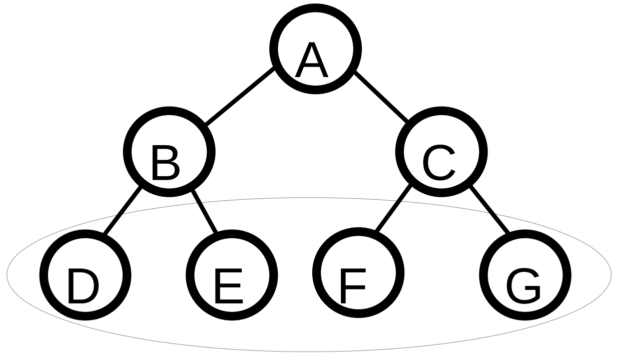


Left-most binary tree



Binary tree properties

Maximum number of leaves at level (depth)i:



example.: 2^{i-1}

Level = 3

Nimber of nodes = 2 $^{3-1}$

Binary tree properties

Maximum number of nodes in a tree of height k:

$$n_{\max} = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

Smalles height of a tree with n nodes:

$$k = \log_2(n+1)$$

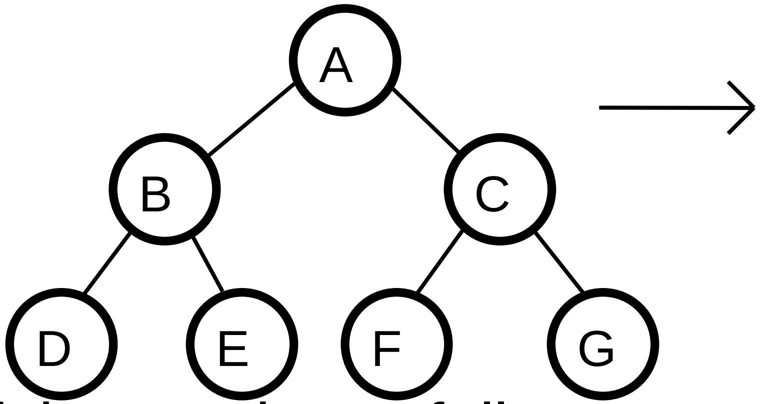
Predstavitev dvojiških dreves

- array
  - Forget about index.
  - Sort items by levels.
  - At the same level from left to right.
  - Example:
    - left offspring(i) = 2i, if i has a left offspring, else undef
    - right offspring(i) = 2i + 1, if i has a right offspring, else undef

- Using pointers
  - (data, pointer to father)
  - (data, pointer to left and right offspring)
  - (data, pointer to father, pointer to left and right offspring)
  - Depends on usage

Static presentation of a tree

With linear array: favorable, if the tree is full or left aligned HOW DO WE MAINTAIN THE STRUCTURE? Elements are entered into the table starting with the root node by levels; blanks are specially marked;



Nodes in the array:

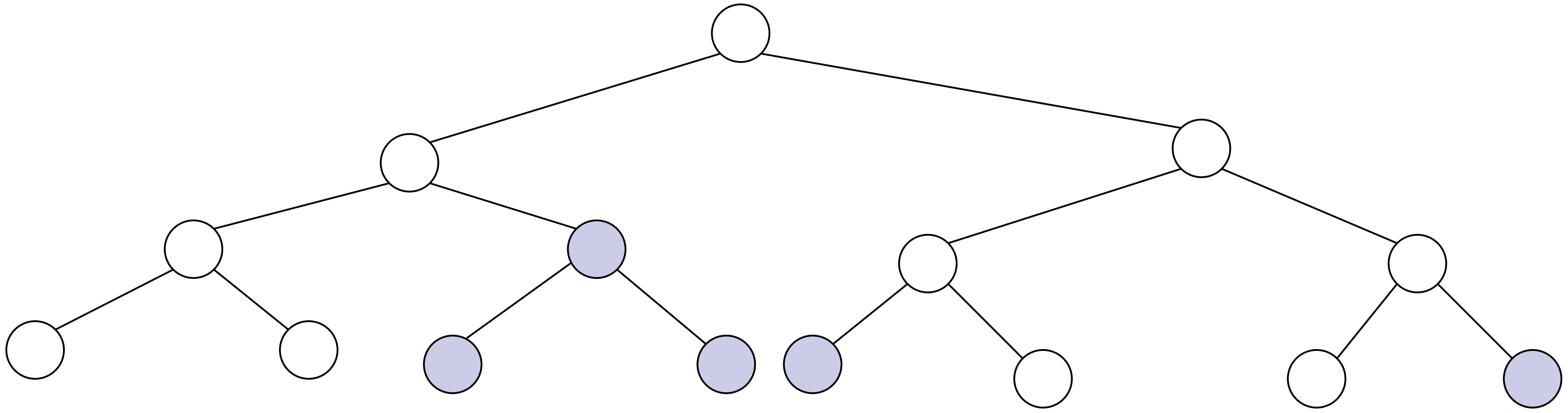
A	B	C	D	E	F	G

Rules:
Father(i):=i % 2 (or none)
Left_offspring:=father*2 (or none)
Right_offspring:=father*2+1(or none)

If the tree is not full, we need for each node to release vacant empty space into a vector to keep indexing .

Presentation of binary trees

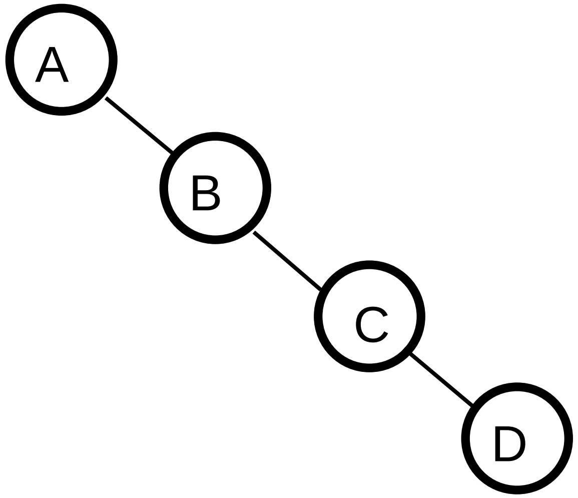
```
int[] drevo = new int[100];
```



```
d[1] := 10; d[2] := 12; d[3] := 31; d[4] := 4;
d[5] := 0; d[6] := 51; d[7] := 16; d[8] := 71;
d[9] := 8; d[10] := 0; d[11] := 0; d[12] := 0;
d[13] := 9; d[14] := 10; d[15] := 0
```

Tree

Bad case: degenerated tree



Presentation of the vector in static presentation:

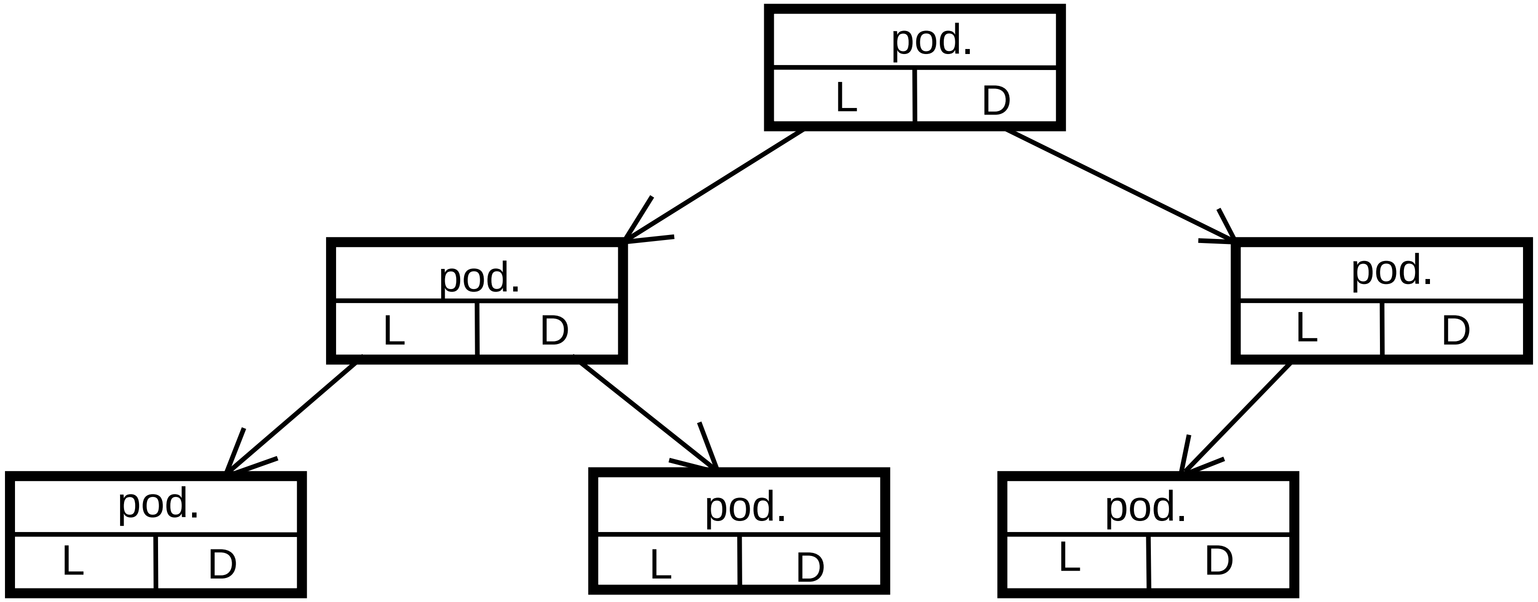
A	-	B	-	-	-	C	-	-	-	-	-	-	D

In poorly occupied or even degenerated tree we deal with great loss of space. "Holes" that are high in a tree, cause significant loss because all their successors are missing at lower levels (two positions at the level of the hole, four on the next, etc.).

Presentation with pointers (kazalci, referencami)

- Ussually:
  - Pointer to left offspring
  - Data
  - Pointer to right offspring

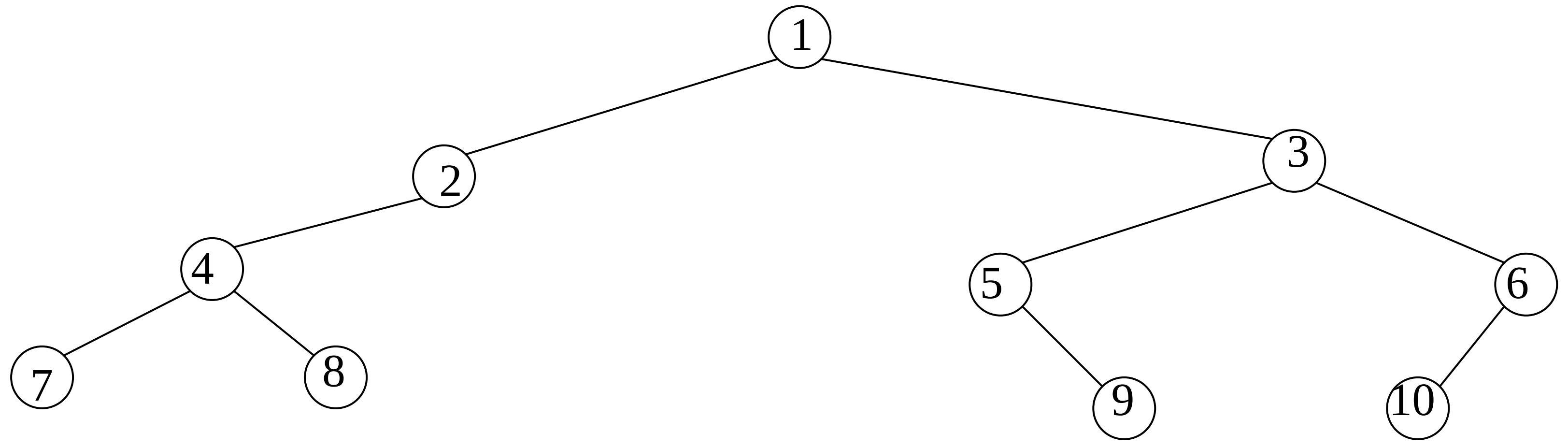
Dinamic tree presentation



Binary tree presentation

- Visit nodes in a specific order:
  - in depth
  - by levels
  - the leaves, ...
- The most important:
  - **Preorder** (premi): visit the father, preorder(left subtree), preorder(right subtree)
  - **Inorder** (vmesni): inorder (left subtree), visiti father, inorder(right subtree)
  - **Postorder**(obratni): postorder(leftt subtree), postorder(right subtree), visit father.

Binary tree presentation



PREMI (preorder): 1, 2, 4, 7, 8, 3, 5, 9, 6, 10

VMESNI (inorder): 7, 4, 8, 2, 1, 5, 9, 3, 10, 6

OBRATNI (postorder): 7, 8, 4, 2, 9, 5, 10, 6, 3, 1

Binary tree - demo

- http://www.cs.usask.ca/resources/tutorials/csconcepts/1998_6/bintree/2-2.html
- http://www.educa.fmf.uni-lj.si/www375/2000/Algoritmi/Java-examples/Tree.html

Preorder(premi vrstni red)

- Same idea as in print
- recursion

Reconstruction of a tree

- Two full visits are enough:
- preorder: 1, 2, 4, 7, 8, 3, 5, 9, 6, 10
- inorder: 7, 4, 8, 2, 1, 5, 9, 3, 10, 6
- root: 1
- Left subtree V: 7, 4, 8, 2
- left P: 2, 4, 7, 8
- right subtree V: 5, 9, 3, 10, 6
- right P: 3, 5, 9, 6, 10

Iskalno dvojiško drevo

- ni podvojenih elementov
- def: *dvojiško drevo, za katerega velja, da so vsi elementi v levem poddrevesu (ki je tudi iskalno drevo) manjši od korena in v desnem poddrevesu večji od korena. Tudi desno poddrevo je iskalno dv. drevo*
- Iskanje podatkov je lahko dokaj hitro: odvisno od učinkovitosti predstavitve

Iskanje v IDD

- http://www.cs.queensu.ca/home/jstewart/applets/bst/bst-searching.html

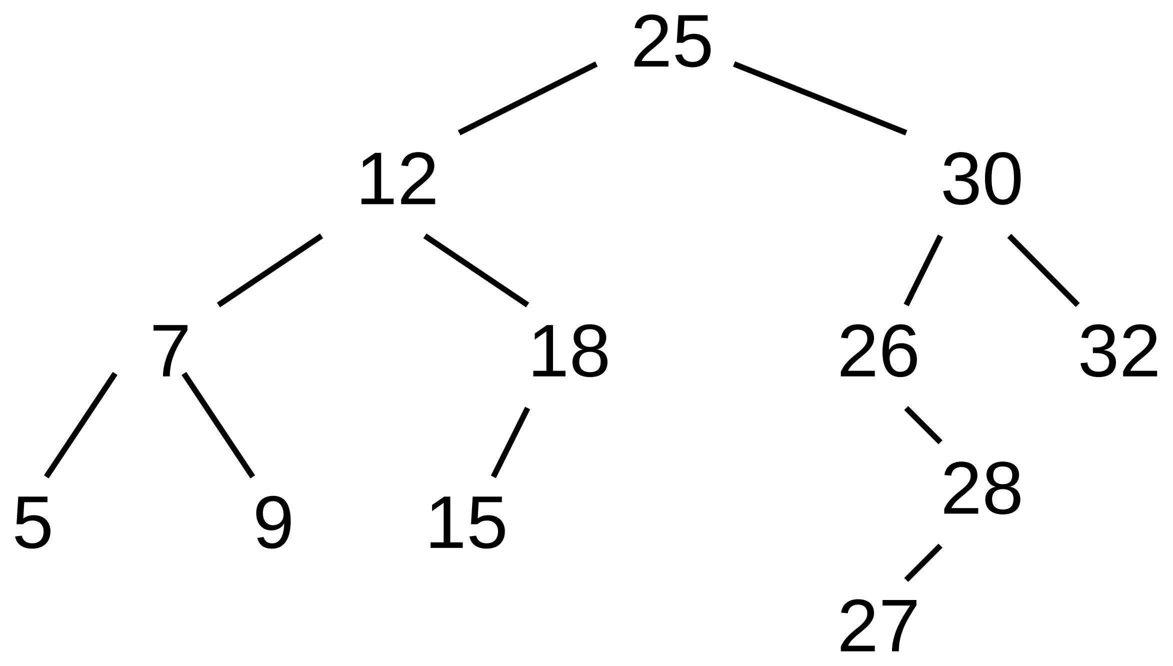
Search tree

- contains data arranged in by one key;
- each node: each element in the set of data in the left subtree is not larger than and each element in the set of data in the right subtree in not smaller than the data in the root
- **binary search tree or binary dictionary**

Search tree

- Binary dictionary:
  - description of the process

Example:

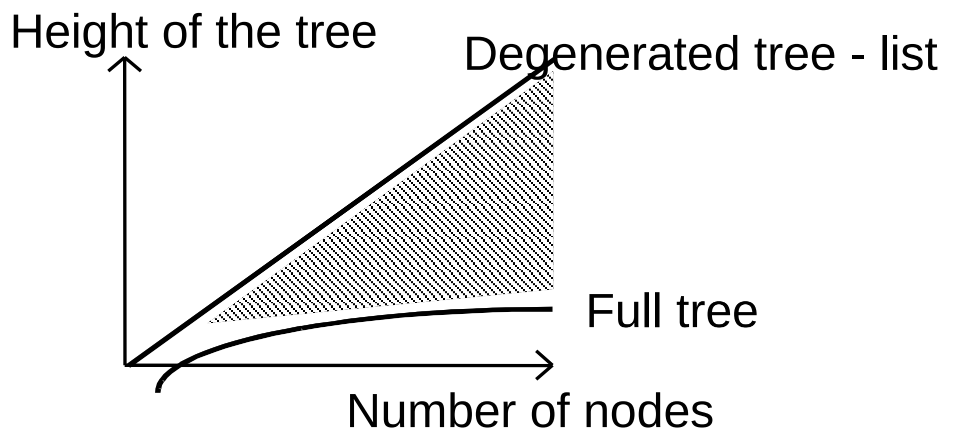


Tree

Vzdrževanje urejenih podatkov v seznamu in v iskalnem drevesu

- število primerjav in premikov, potrebnih, da pregledamo iskalno drevo, je enako višini drevesa; rast višine drevesa s številom podatkov je počasna, posebej za velike n;
- višina polnega drevesa z n podatki je proporcionalna log(n);
- če v iskalno drevo vstavljamo že urejeno zaporedje podatkov, dobimo izrojeno drevo - seznam;
- višina izrojenega drevesa z n vozlišči je n.

Comparison



- The expected height of search tree is in the dashed area. We want to be close to the bottom curve.
- Advantage of maintaining data in the tree increases with more data. Twice as much data increases its height (full tree) by 1 (number of necessary comparisons and movements).

Example:

- Search data in list with 10^6 elements: worst case 10^6 comparisons and moves.
- Search data in full binary tree with 10^6 elements: worst case:

$$\log_2(10^6) = \frac{\log_{10}(10^6)}{\log_{10}(2)} = \frac{6}{0.303} \cong 20$$

comparisons.

- Search in a tree with $2*10^6$ data needs only one more comparison.

Insert

- Same as search