Kahendpuu (Binary tree)

Programmeerimise süvendatud algkursus (ITI0140)

2014

Teemad

- Linked list
- Tree
- Binary tree
- Different types of binary trees

Terms (*mõisted*)

- Node (*sõlm*)
- Root (*juur*)
- Parent node (*vanem sõlm*)
- Child node (*laps-sõlm*)
- Sibling (*?*)
- Path (*tee*)

Embedded reference

- Attributes which refer to other objects are called **embedded references**
- Objects are said to be **linked** if an instance of one objects stores a reference to another object
- Special case of linkage is when an object contains a link to an object of the **same class**

Linked list

- Linked list is made up of **nodes**
- Each node contains a **reference to the next node** in the list
- In addition, each node contains a unit of data called the **cargo**

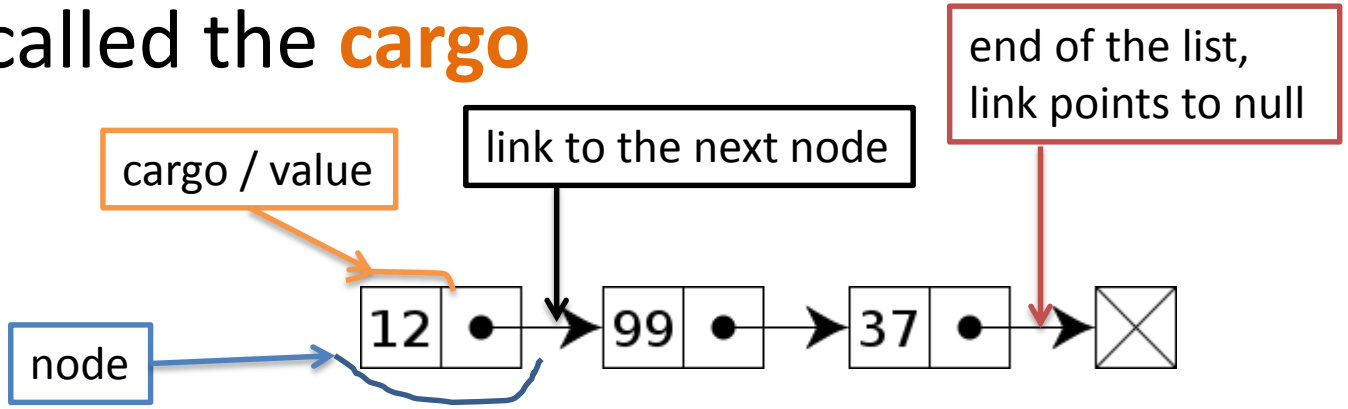


Image: http://en.wikipedia.org/wiki/File:Singly-linked-list.svg

Linked list

- Node:

```
class Node:
    def __init__(self, cargo=None, next=None):
        self.cargo = cargo
        self.next  = next

    def __str__(self):
        return str(self.cargo)
```

- Example:

```
node1 = Node(1)
node2 = Node(2)
node3 = Node(3)

node1.next = node2
node2.next = node3
```

Linked list

- Adding node

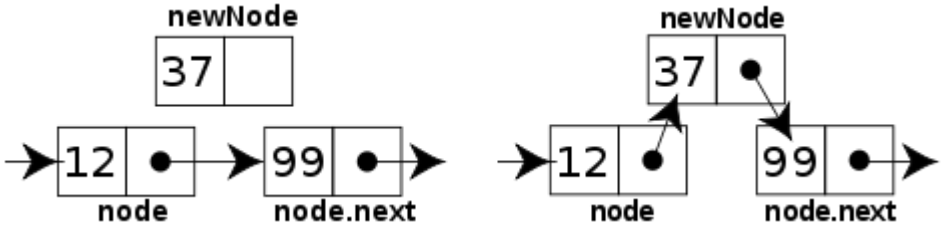


Image: http://en.wikipedia.org/wiki/File:CPT-LinkedLists-addingnode.svg

- Removing node

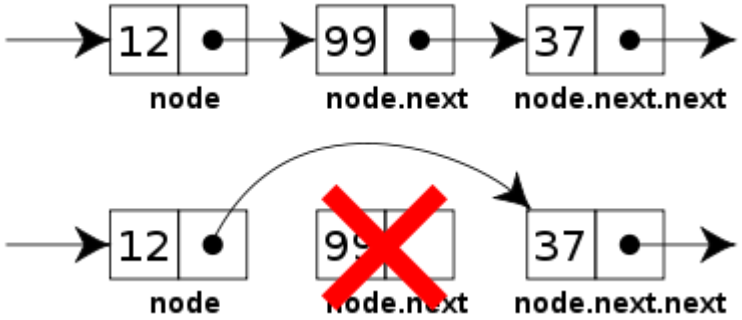


Image: http://en.wikipedia.org/wiki/File:CPT-LinkedLists-deletingnode.svg

Tree

- Like linked lists, trees are made up of nodes
- The top of the tree is called the **root**
- Other nodes are called **branches**
- Nodes with null reference are called **leaves**

- The top node is called **parent** and the nodes it refers to are its **children**
- Nodes with the same parent are called **siblings**
- All the nodes which are the same distance from the root comprise a **level** of the tree

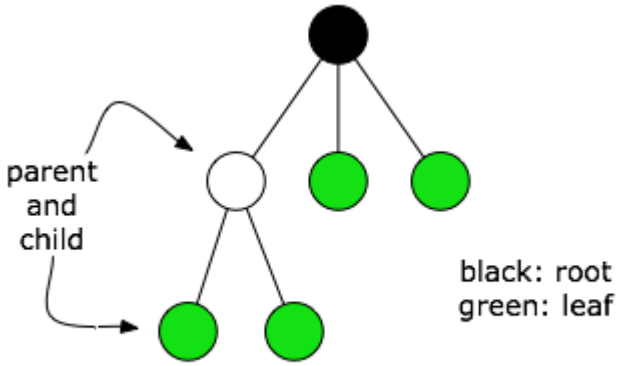
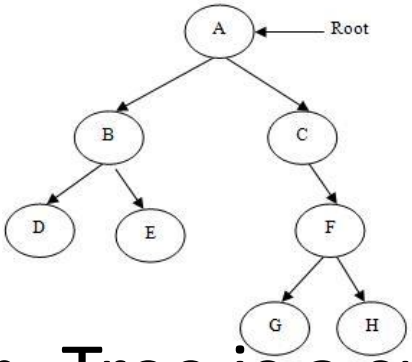
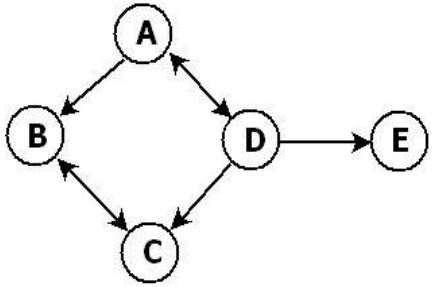


Figure: tree data structure

Image: http://xlinux.nist.gov/dads/HTML/tree.html



Tree *vs* graph



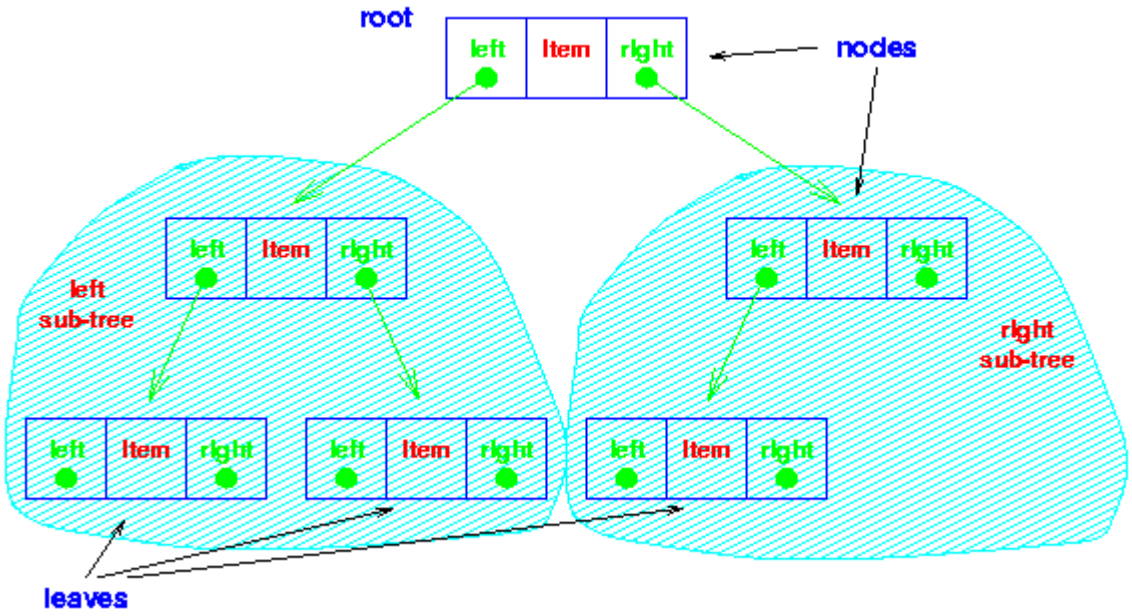
- Tree is a special form of graph, **minimally connected graph** having only one path between any two vertices
- Tree has **no loops**
- Tree has exactly **one root node**
- In tree, every **child** node has only **one parent**
- Tree has always **n-1 edges**

Comparison: http://freefeast.info/difference-between/difference-between-trees-and-graphs-trees-vs-graphs/

Binary tree

- Tree, where each node contains a reference to two other nodes (possibly None)
- References are referred to as the left and right sub-trees

- Binary tree consists of:
  - a root node
  - left and right sub-trees
- Both the sub-trees are themselves binary trees



Binary tree

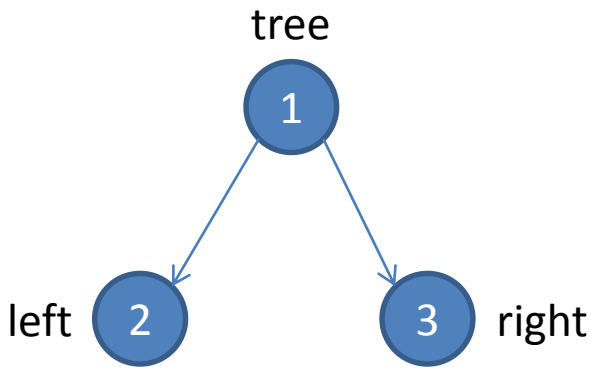
- Node:

```
class Tree:
    def __init__(self, cargo, left=None, right=None):
        self.cargo = cargo
        self.left  = left
        self.right = right

    def __str__(self):
        return str(self.cargo)
```

- Example:

```
left = Tree(2)
right = Tree(3)
tree = Tree(1, left, right)
```



Sorted binary tree
Binary search tree

- **Binary search tree** (BST), sometimes also called sorted or ordered binary tree:
  - the **left sub-tree** of a node contains only nodes with **values less than** node's key
  - the **right sub-tree** of a node contains only nodes with **values greater than** node's key
  - the left and right sub-trees have to be binary search trees
  - there must be **no duplicate nodes**

Searching BST

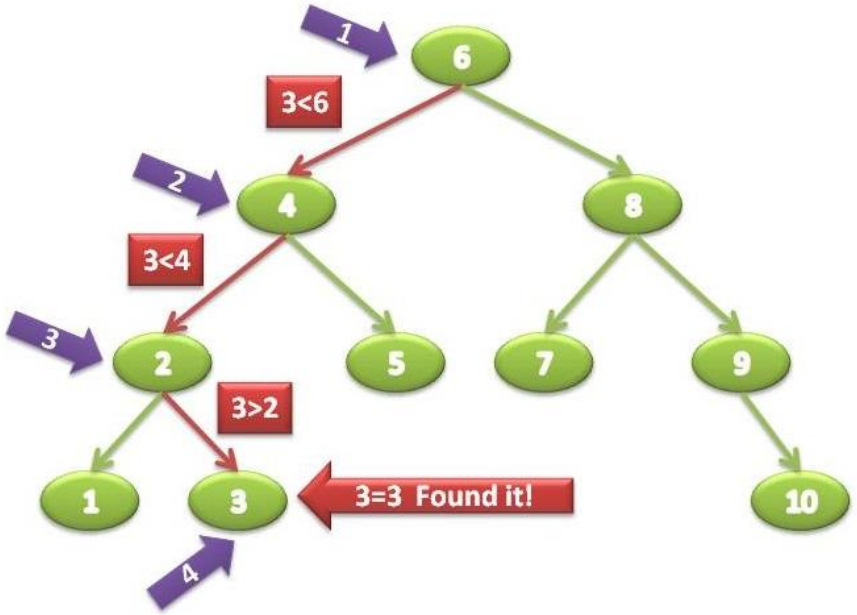


Image: http://encrypt3d.wordpress.com/2010/09/25/how-to-search-in-a-binary-search-tree/

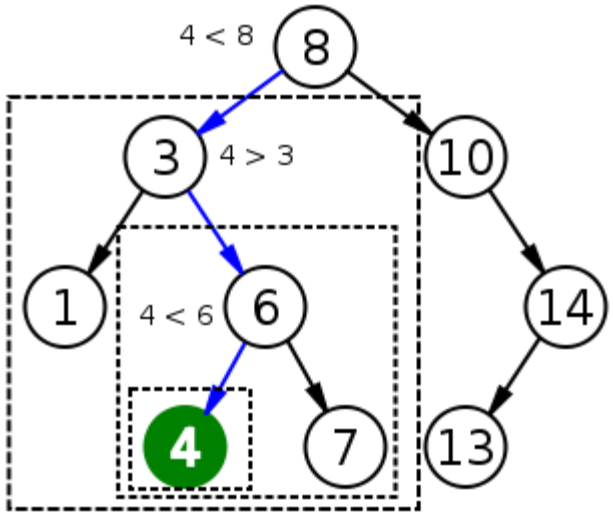
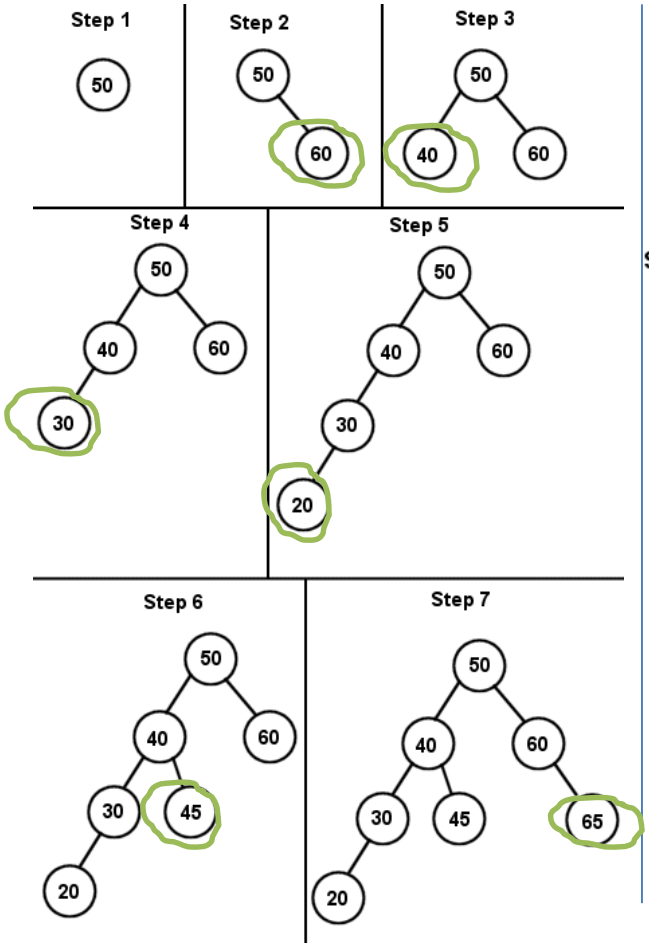
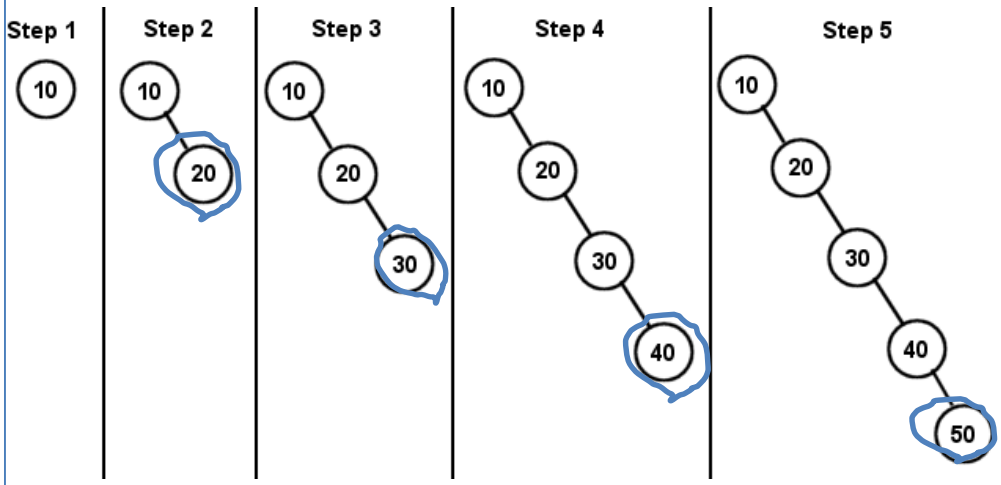


Image: http://commons.wikimedia.org/wiki/File:Binary_search_tree_search_4.svg

Inserting to BST



Worst case:



Images: http://archive.gamedev.net/archive/reference/articles/article1433.html

Balanced binary tree

- A **balanced binary tree** is commonly defined as a binary tree in which the depth of the left and right sub-trees of every node differ by 1 or less
- Balanced binary tree has a predictable depth
- Lookup, insertion, deletion completed in O(log *n*) time for *n* node tree

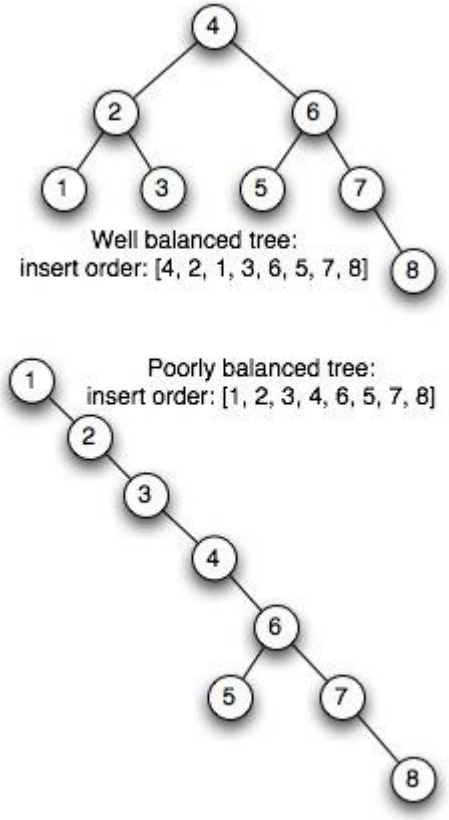


Image: http://scienceblogs.com/goodmath/2007/01/01/balanced-binary-trees-in-haske/

Additional info

- http://en.wikipedia.org/wiki/Binary_tree
- http://archive.gamedev.net/archive/reference/articles/article1433.html
- http://algs4.cs.princeton.edu/32bst/
- http://encrypt3d.wordpress.com/category/data-structures/binary-tree/
- http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree

Ülesanne

- Lugeda sisse sõnad failis.
- Paigutada sõnad kahendpuusse
- Iga *node* peaks omama järgmiseid parameetreid:
  - sõna
  - sõnade arv (algselt 1, kui sama sõna sisestatakse, siis suurendatakse)
  - viide vasakule alampuule (kus sõnad on tähestikus eespool)
  - viide paremale alampuule (kus sõnad on tähestikus tagapool)
- Realiseerida funktsioonid:
  - **maxDepth()** - tagastab maksimaalse sügavusega elemendi sügavuse
  - **minValue()** - tagastab kõige väiksema väärtusega elemendi (kõige "vasakpoolsem" element)
  - **printTree(Node)** - prindib puu elemendid välja rekursiivselt. Tulemus peaks olema sorteeritud sõnastik.
  - **search(word)** - tagastab node'i, mille otsitav sõna sisaldub, kauguse juurelemendist. Lisaks tagastab sõnade arvu (mitu korda seda sõna tekstis esines). Kui sõna ei leidu, tuleks vastav teade anda.
- Boonusülesanne:
  - Tasakaalustada puu (kas jooksvalt või lõpus)