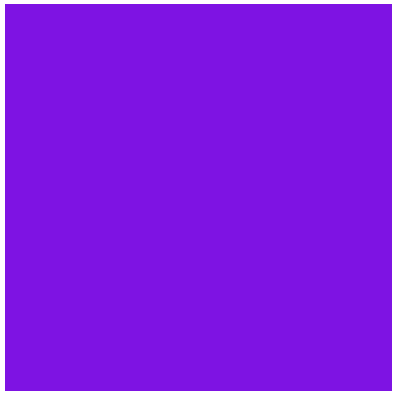


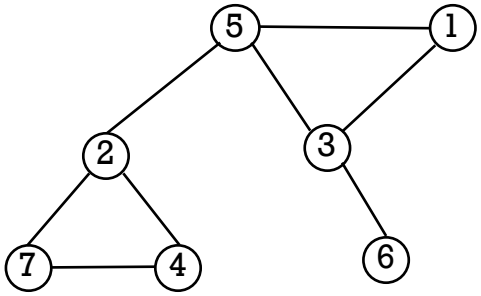
Bomen en Grafen

Les 3

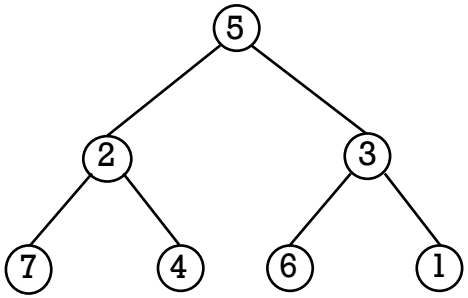
Datastructuren

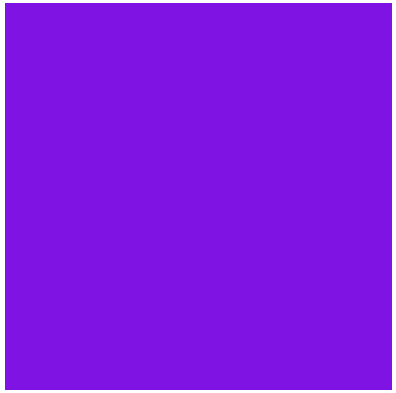
- Lineair: de elementen vormen een rij.

- Niet-lineair: de elementen vormen geen rij.
  - Graaf: lussen zijn toegelaten.



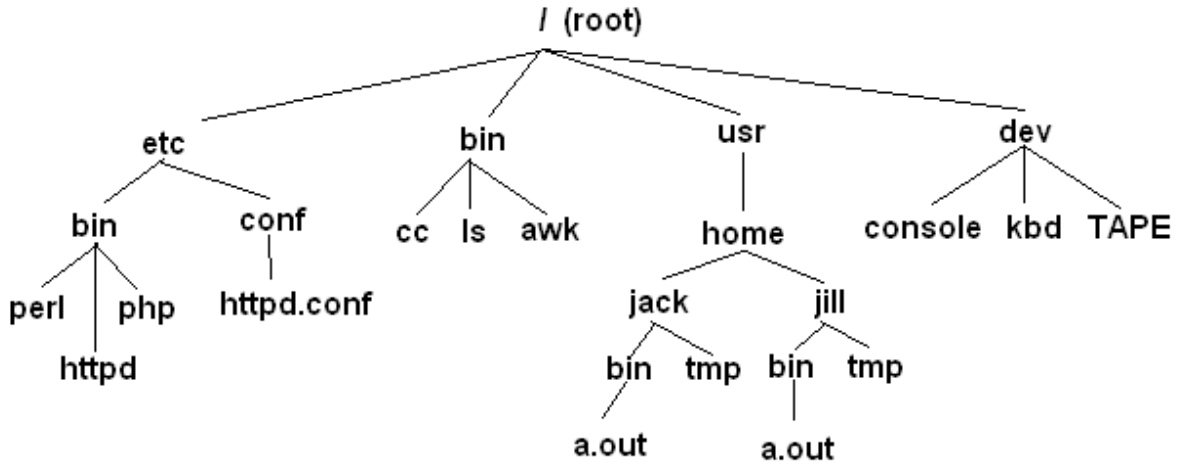
  - Boom: lussen zijn niet toegelaten.



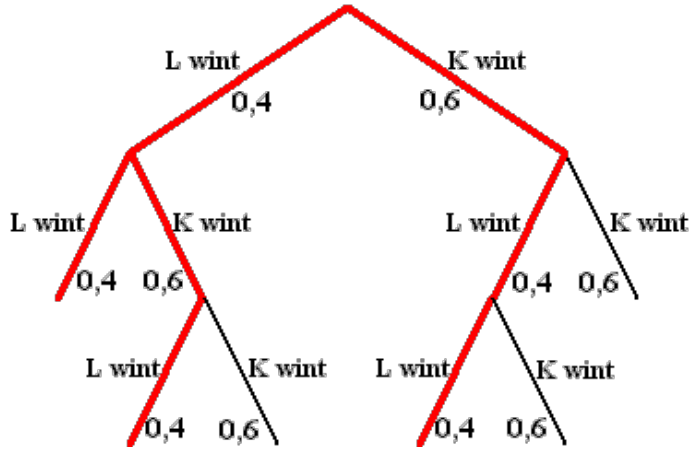


Bomen

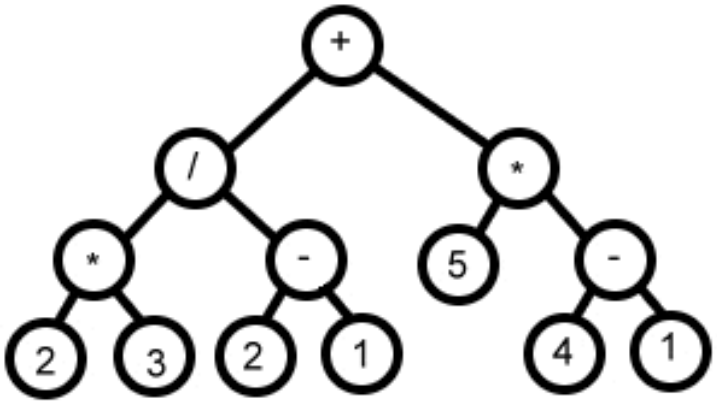
Bestandssystemen op computers



Kansbomen (statistiek)

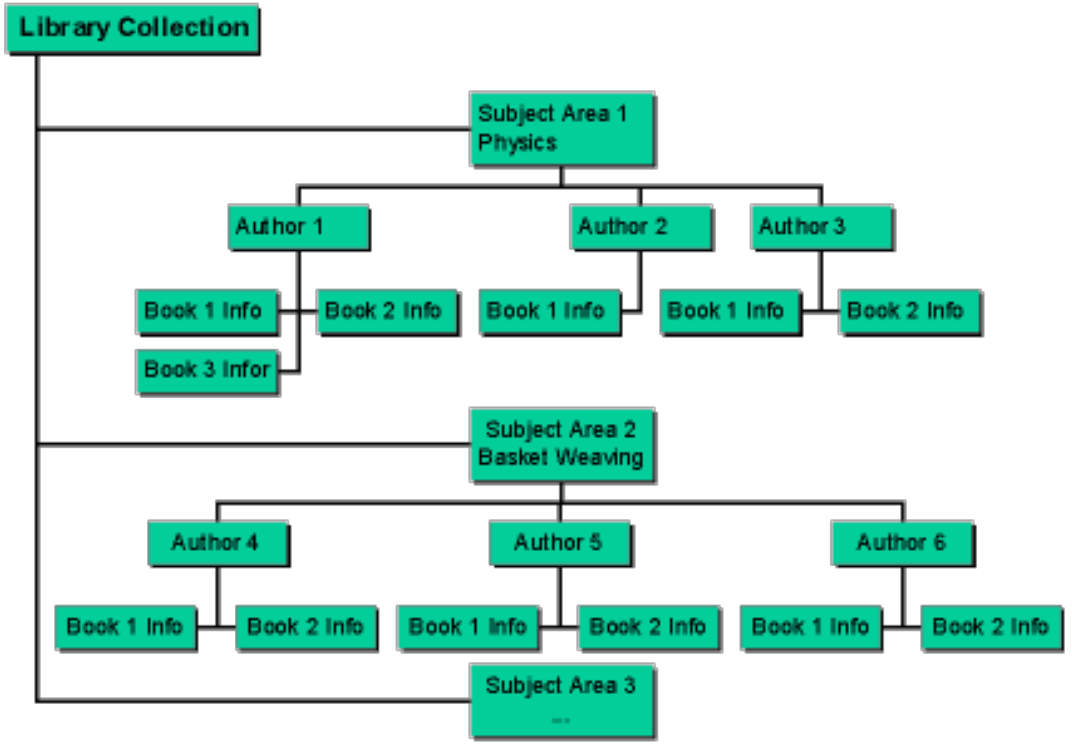


Expressiebomen

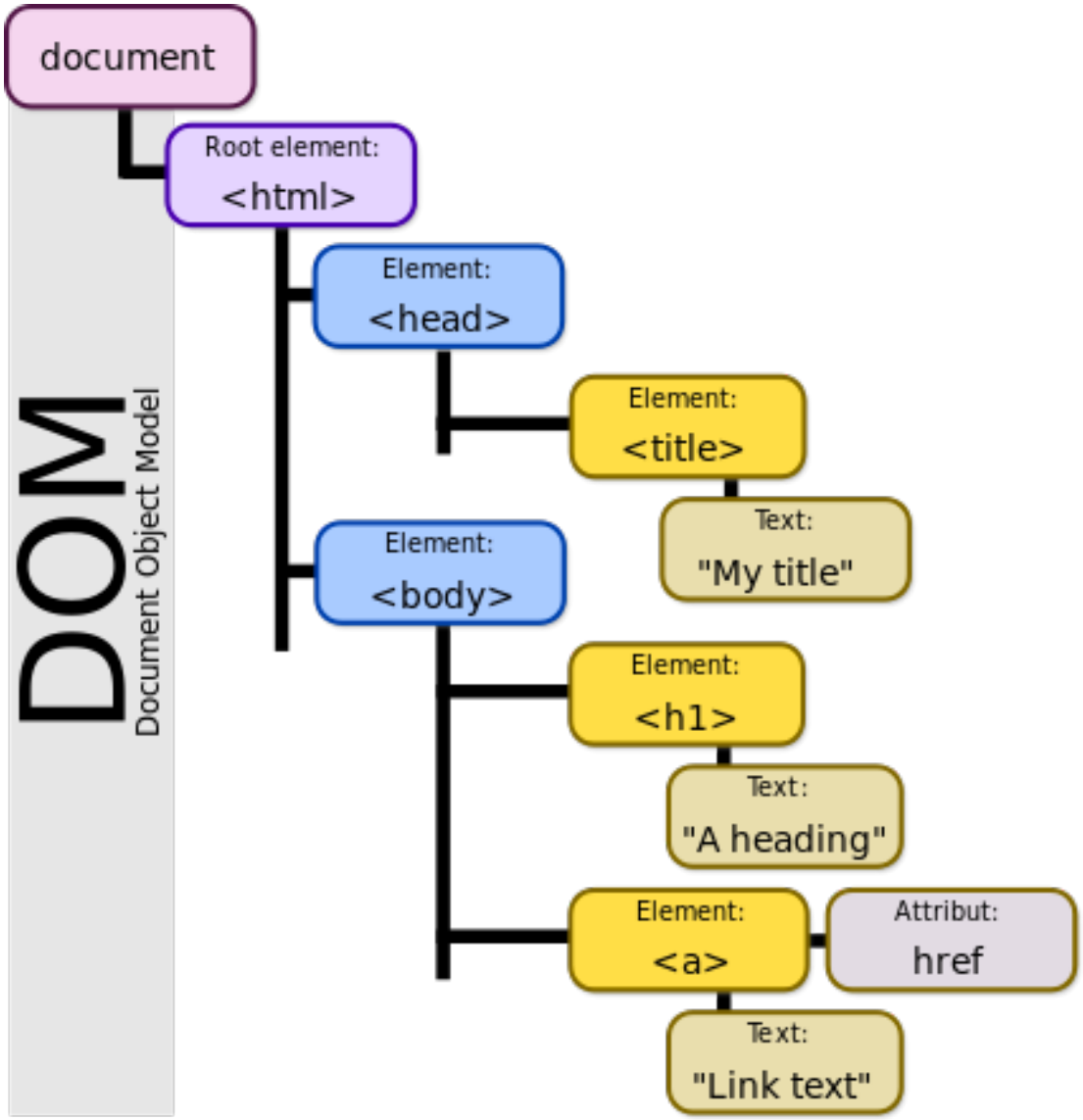


Expression tree for 2*3/(2-1)+5*(4-1)

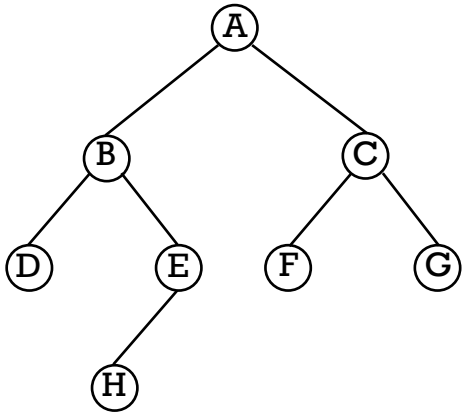
Hierarchical database model



Document Object Model



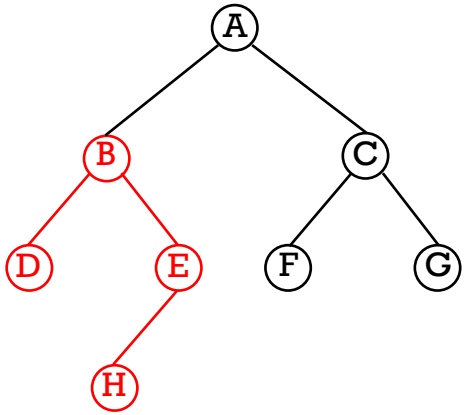
Terminologie



- A noemt men de wortel (root) van de boom.
- De knopen B en C zijn de kinderen (children) van de wortel, D en E zijn de kinderen van B, ...
- Omgekeerd is knoop C de ouder (parent) van F en G en is E de ouder van H.
- Een knoop zonder kinderen wordt een blad (leaf) of externe knoop genoemd. (D, F, G en H)
- Een knoop die minstens 1 kind heeft, wordt een interne knoop genoemd. (A, B, C en E)

Terminologie

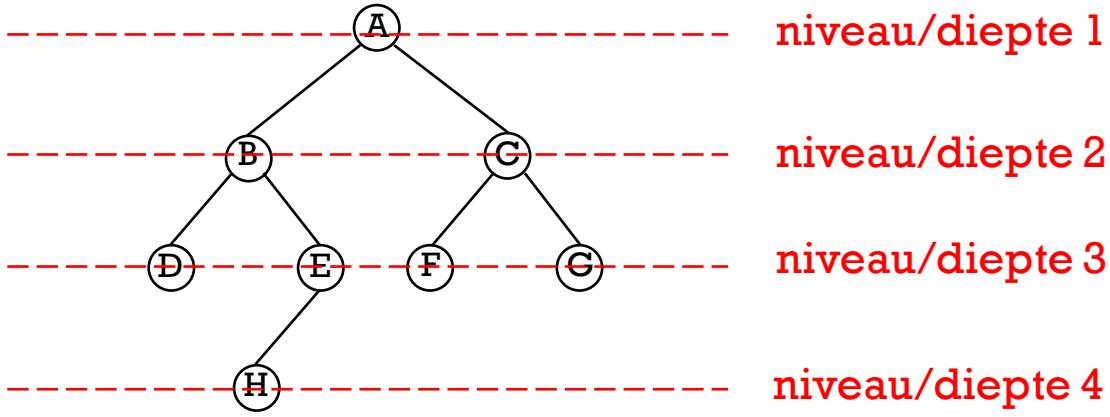
B en al zijn afstammelingen vormen een subboom waarvan B de wortel is.



- A noemt men de wortel (root) van de boom.
- De knopen B en C zijn de kinderen van de wortel, D en E zijn de kinderen van B, ...
- Omgekeerd is knoop C de ouder (parent) van F en G en is E de ouder van H.
- Een knoop zonder kinderen wordt een blad (leaf) of externe knoop genoemd. (D, F, G en H)
- Een knoop die minstens 1 kind heeft, wordt een interne knoop genoemd. (A, B, C en E)
- Elke knoop kan beschouwd worden als de wortel van een subboom.

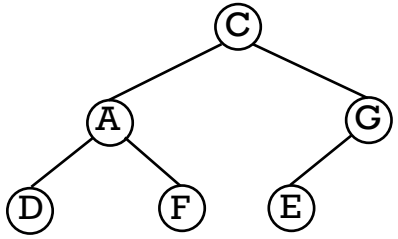
Terminologie

niet-complete binaire boom

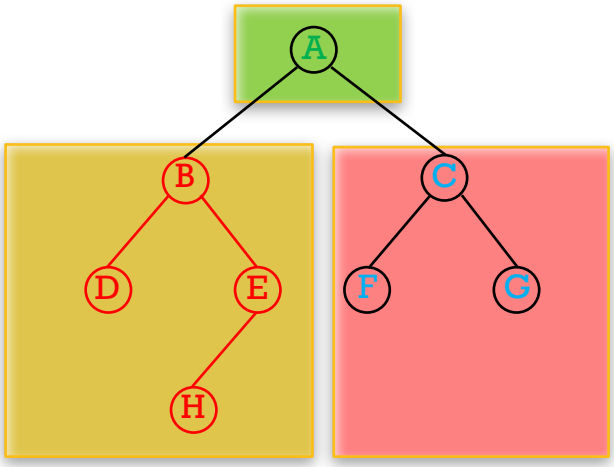


- Een binaire boom is een boom waarvan elke knoop ten hoogste twee kinderen heeft.
- Knoop B is het linkerkind (left child) van A en C is het rechterkind (right child) van A.
- De (maximale) diepte van de boom is het maximaal aantal knopen van een pad van de wortel tot een blad. Bovenstaande boom heeft een (maximale) diepte gelijk aan 4.
- Een binaire boom wordt compleet genoemd als al zijn niveaus behalve eventueel de laatste volledig gevuld zijn en alle knopen op het laatste niveau aan de linkerzijde zijn.

complete binaire boom met diepte 3



Implementatie Binaire Boom

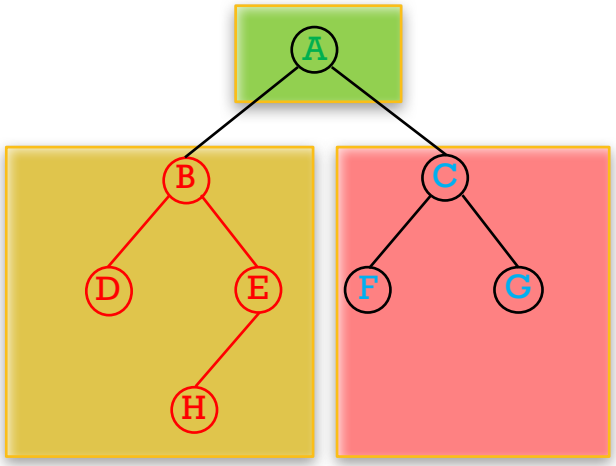


- Een binaire boom bestaat dus uit:
  - Wortel met data
  - Linkerdeelboom
  - Rechterdeelboom

Dit is een recursieve datastructuur (boom is gedefinieerd in functie van zichzelf)

Implementatie van een binaire boom

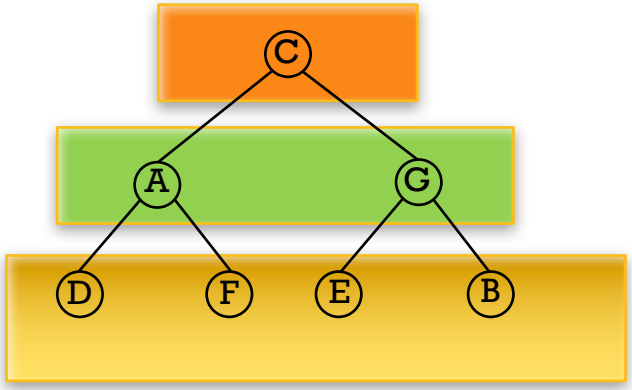

```
public class BinaryTree<E> {
	private E data;
	private BinaryTree<E> leftTree, rightTree;

	public BinaryTree(E data){
		this(data,null,null);
	}

	public BinaryTree(E data, BinaryTree<E> leftTree, BinaryTree<E> rightTree){
		this.data= data;
		this.leftTree= leftTree;
		this.rightTree= rightTree;
	}
}
```



Maken van een binaire boom

- Maak bomen overeenkomstig de bladeren (externe knopen)
- Maak deelbomen met als root de interne knopen van onder naar boven uitgezonderd de wortel
- Maak boom met root

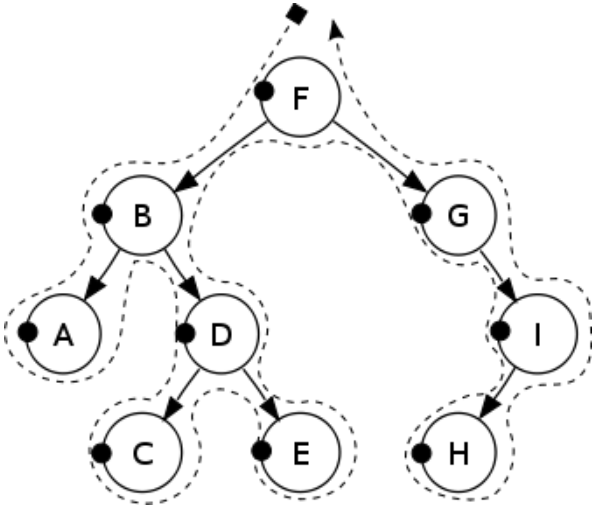
Voorbeeld:



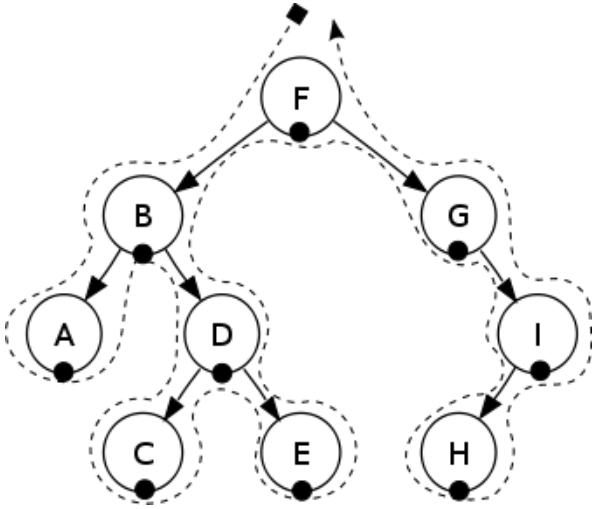

```
// bladeren
BinaryTree<String>  nodeD = new BinaryTree("D");
BinaryTree<String>  nodeF = new BinaryTree("F");
BinaryTree<String>  nodeE = new BinaryTree("E");
BinaryTree<String>  nodeB = new BinaryTree("B");
// nodeA heeft links nodeD en rechts nodeF
BinaryTree<String>  nodeA = new BinaryTree("A",nodeD,nodeF);
BinaryTree<String>  nodeG = new BinaryTree("G",nodeE,nodeB);

BinaryTree<String> boom = new BinaryTree("C",nodeA,nodeG);
```

Wandelen door een boom

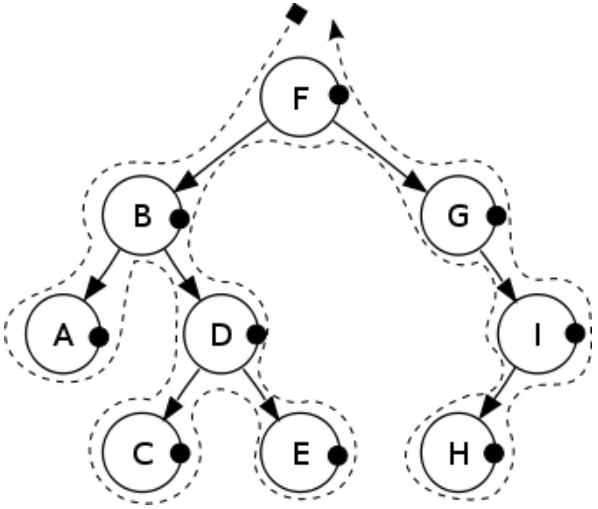
Wandelen door een boom



Pre-order
F, B, A, D, C, E, G, I, H.



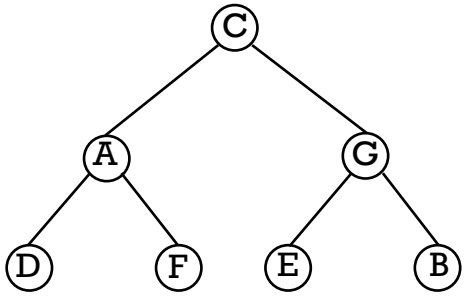
In-order
A, B, C, D, E, F, G, H, I.



Post-order
A, C, E, D, B, H, I, G, F.

https://en.wikipedia.org/wiki/Tree_traversal

Wandelen door een boom



Een boomwandeling verwijst naar een systematische manier om een boom te doorlopen zodanig dat elke knoop juist 1 keer wordt bezocht.

- Depth-first
  - Pre-order
    1) Bezoek de knoop
    2) Wandel door de linkersubboom
    3) Wandel door de rechtersubboom

    C A D F G E B
  - In-order
    1) Wandel door de linkersubboom
    2) Bezoek de knoop
    3) Wandel door de rechtersubboom

    D A F C E G B
  - Post-order
    1) Wandel door de linkersubboom
    2) Wandel door de rechtersubboom
    3) Bezoek de knoop

    D F A E B G C

Implementatie pre-order printen

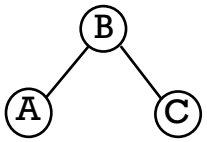
Recursief : implementeer in de klasse BinaryTree

```
public void printPreorder(){
                    System.out.print(this.data + " ");
                    if (this.leftTree != null) this.leftTree.printPreorder();
                    if (this.rightTree != null) this.rightTree.printPreorder();
        }
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output:

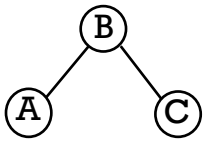
printPreorder root = B_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B

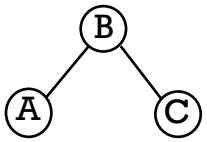
printPreorder root = B_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B

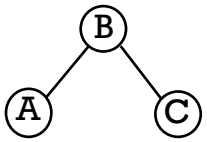
printPreorder node = B

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B

printPreorder root = B_node

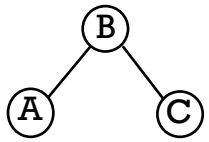
printPreorder root = A_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A

printPreorder root = B_node

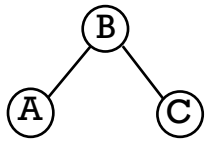
printPreorder root = A_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A

printPreorder root = B_node

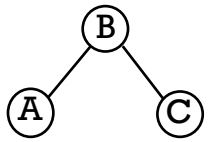
printPreorder root = A_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A

printPreorder root = B_node

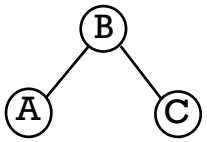
printPreorder root = A_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A

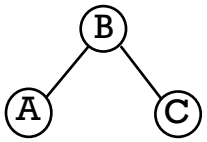
printPreorder root = B_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A

printPreorder root = B_node

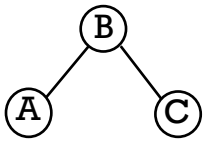
printPreorder root = C_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A C

printPreorder root = B_node

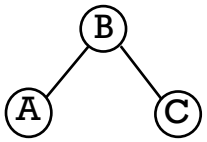
printPreorder root = C_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A C

printPreorder root = B_node

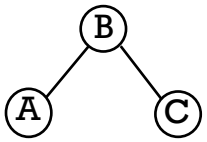
printPreorder root = C_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A C

printPreorder root = B_node

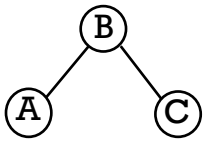
printPreorder root = C_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```

Implementatie pre-order printen

Recursief

Voorbeeld:



Output: B A C

printPreorder root = B_node

```
public void printPreorder(){
System.out.print(this.data  + " ");
if (this.leftTree != null) this.leftTree.printPreorder();
if (this.rightTree != null) this.rightTree.printPreorder();
}
```


Vragen?