OO-datastructuren

Bachelor IT
Sven Mariën
(sven.marien01@ap.be)

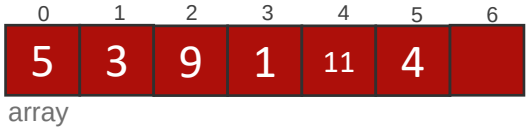
AP HOGESCHOOL ANTWERPEN

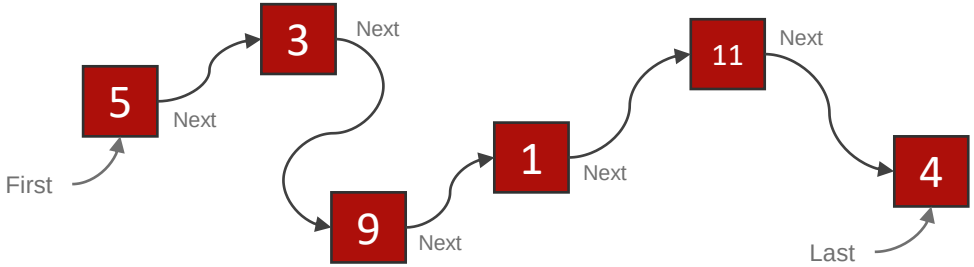
AP.BE

Even terugblikken

- Tot nu toe hebben we gewerkt met array's
  - = Aaneensluitend blok geheugen
  - Voordelen:
    - **Snelle lees / schrijf toegang** aan de hand van de **index**
  - Nadelen:
    - **Op voorhand geheugen reserveren**, wat misschien niet - volledig - gebruikt wordt & de **grootte ligt** dan ook **vast**.
    - Array toch groter maken => nieuwe array aanmaken en **alle elementen overkopiëren**
    - Een element tussenvoegen, ook dan moeten **alle volgende elementen 1 plaats worden opgeschoven**.

Alternatief: Linked list

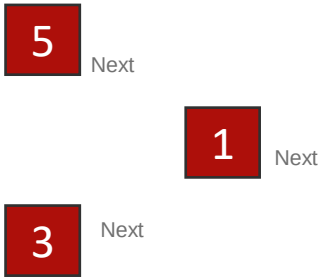




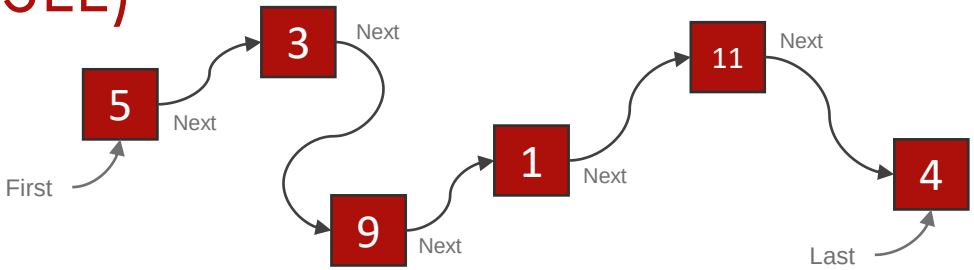
- Losse elementen her en der in het geheugen die naar elkaar verwijzen

Node

- Elk element in een “linked list” wordt een “**Node**” genoemd
- (in het nederlands spreekt men van een “gelinkte lijst” met “records”, knooppunten, schakels,…)
- Een Node heeft steeds
  - Een **Value** (een getal, een string, een persoon,..)
  - Een **verwijzing** (referentie) naar de **eerstvolgende** Node in de lijst.



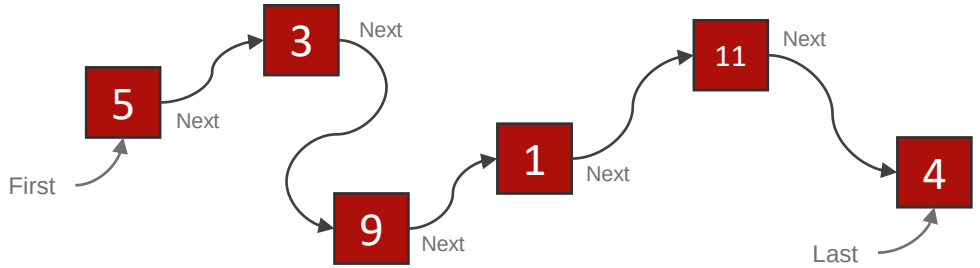
Singly Linked List (SLL)



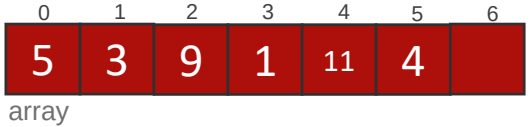
- Een node verwijst telkens naar de volgende node in de lijst, dit noemt een "**Singly linked list**"
- Er **moet** ook een **referentie** zijn naar de **1e node** ("First" of "Head")
- Soms houdt men ook een **referentie** bij naar de **laatste** node ("Last" of "Tail")
- In tegenstelling tot bij een array
  - zitten de nodes dus verspreid in het geheugen
  - de enige manier om een bepaalde node "te vinden", is **de lijst vanaf de "Head" 1 voor 1 aflopen**

Element opzoeken

- Zoek het element met een bepaalde “value” (bv. 1)
  - Beginnen bij “First”
  - Heeft deze node de gezochte value ?
  - Ga verder via Next tot de te zoeken value.
  - Of stop bij het einde van de lijst.

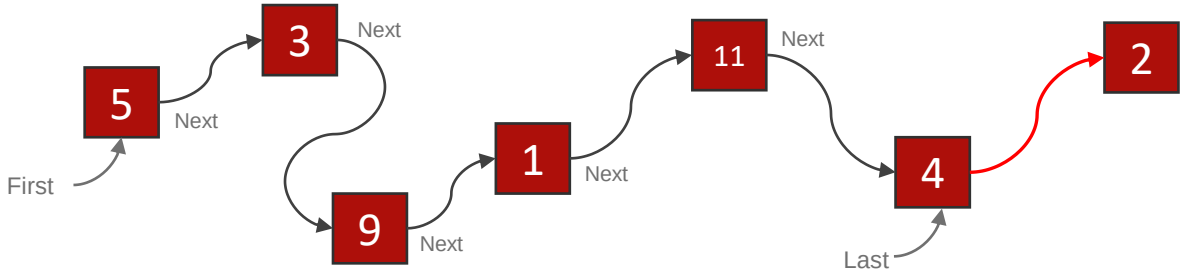


Vergelijk met een array..

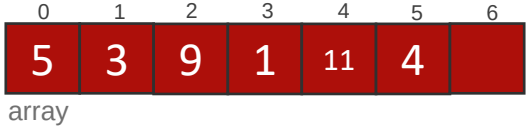


Element achteraan toevoegen

- Beginnen bij “Last”
- Maak een nieuwe Node aan (bv. met value = 2)
- Laat de “Last” node verwijzen naar de nieuwe Node
- Laat de “Last” pointer verwijzen naar de nieuwe Node

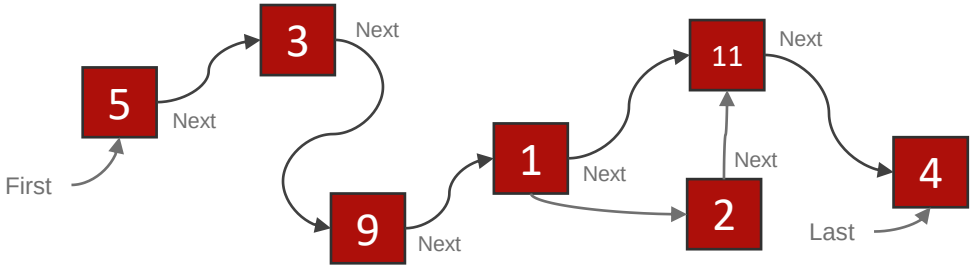


- Vergelijk met een array..

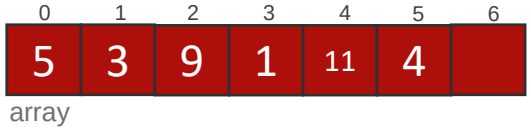


Element tussenvoegen

- **Na** een bepaalde node (bv. value = ‘1’)
  - Maak een nieuwe Node aan (met value = 2)
  - Laat de nieuwe node verwijzen naar de “Next” node
  - Laat de Node ‘1’ verwijzen naar de nieuwe Node

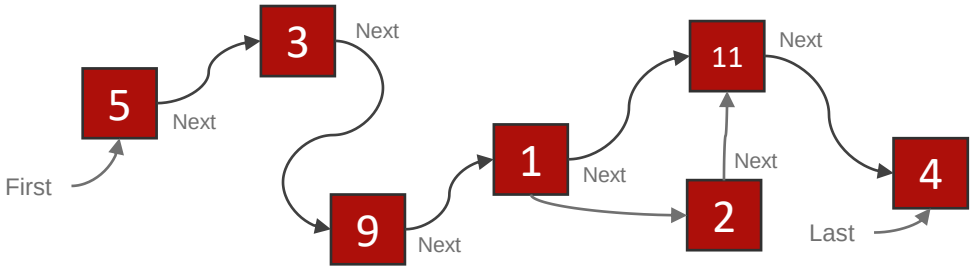


- Vergelijk met een array



Element tussenvoegen

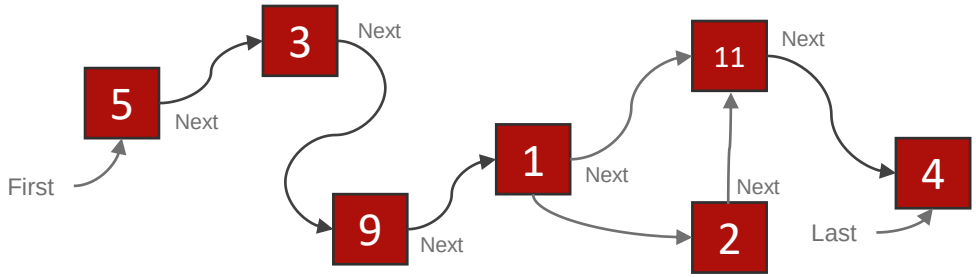
- **Voor** een bepaalde node (bv. value = ‘11’)
  - Maak een nieuwe Node aan (met value = 2)
  - Laat de nieuwe node verwijzen naar deze node (‘11’)
  - Laat de Node ervoor (‘1’) verwijzen naar de nieuwe Node



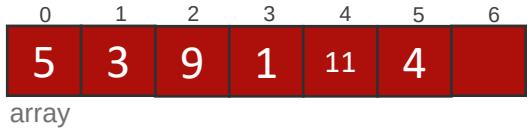
Is er een verschil in time complexity met de vorige ?

Element verwijderen

- Een node verwijderen (bv. met value = ‘2’)
  - Laat de node ervoor verwijzen naar de “Next” node
  - De node ‘2’ kan verwijderd worden (gebeurt automatisch in .NET)

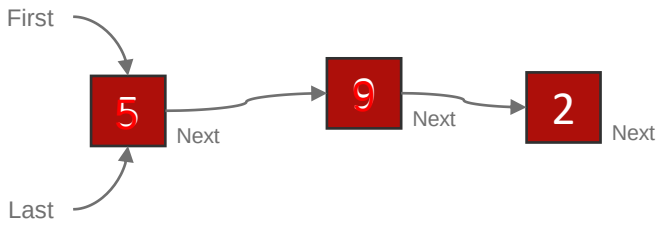


- Vergelijk met een array:



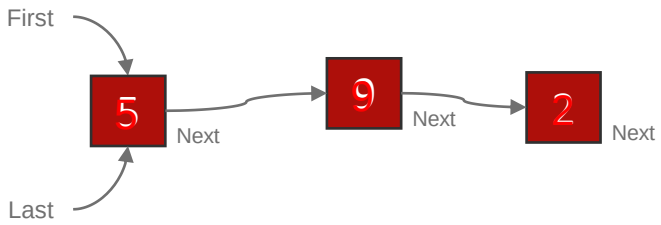
“Queue” op basis van een “Singly Linked List”

- Queue
  - Is een FIFO structuur (wachtrij)
  - heeft 2 methodes (Enqueue, Dequeue)
- Bv.
  - Lege queue
  - Enqueue(5)
  - Enqueue(9)
  - Enqueue(2)
  - int a = Dequeue()
  - Int b = Dequeue()
  - …



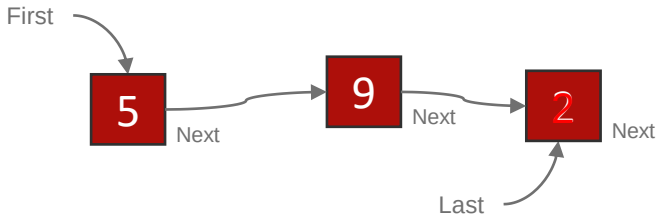
“Stack” op basis van een “Singly Linked list”

- Stack
  - Is een LIFO structuur (stapel)
  - Heeft 3 methodes (push, pop, peek)
- Bv.
  - Lege stack
  - Push (5)
  - Push (9)
  - Push (2)
  - int a = Pop()
  - int b = Pop()
  - Int c = Peek()



Had je dit opgemerkt ?

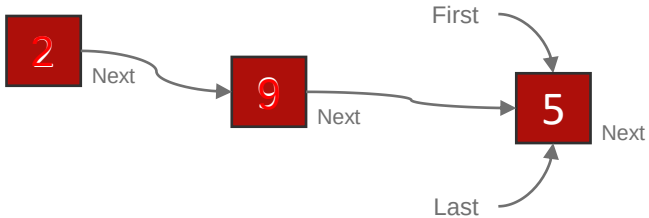
- Als ik van een stack een Pop wil implementeren..



- Hoe weet naar **welke node** mijn Last variabele nadien moet verwijzen …?
- We moeten de volledige lijst doorlopen tot we aan de node komen die zich voor de “Last node” bevindt (maw. die dus verwijst naar de last node).

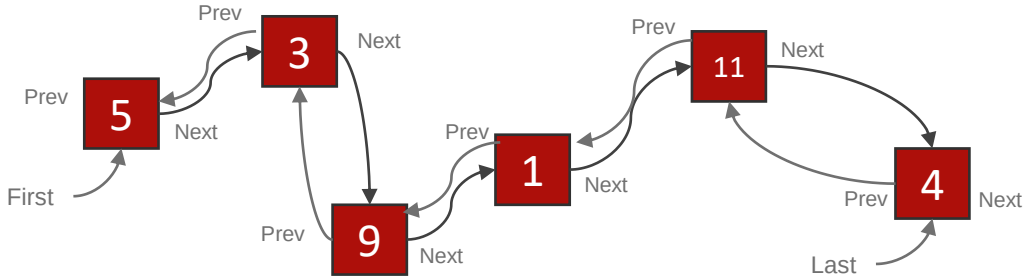
Kan dit beter ?

- Uiteraard !
- Bv.
  - Lege stack
  - Push (5)
  - Push (9)
  - Push (2)
  - int a = Pop()
  - int b = Pop()
- Of …. er is ook nog een ander alternatief



Doubly Linked List (DLL)

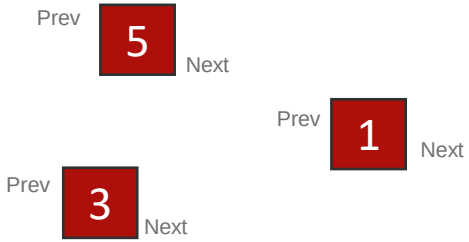
- Bij een doubly linked list verwijst elke node zowel naar
  - de **volgende node** (Next)
  - de **vorige node** (Previous)



  - Dit betekent dat we deze lijst in **2 richtingen** kunnen doorlopen !
  - Of, als we een bepaalde node hebben dat we nu zowel de volgende als de voorgaande node kennen.

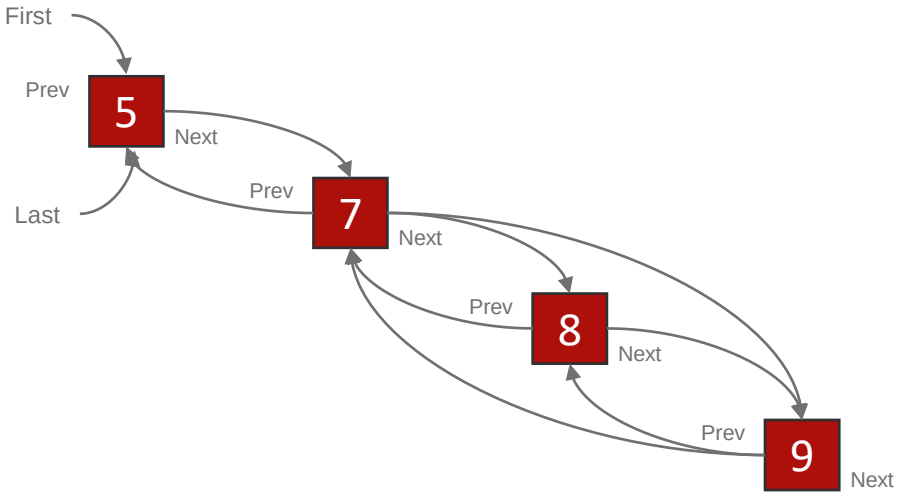
Node

- Een Node in een “doubly linked list” heeft steeds
  - Een **value** (een getal, een string, een persoon,..)
  - Een **referentie** naar de **eerstvolgende** Node in de lijst (Next).
  - Een **referentie** naar de **eerstvoorgaande** Node in de lijst (Prev).



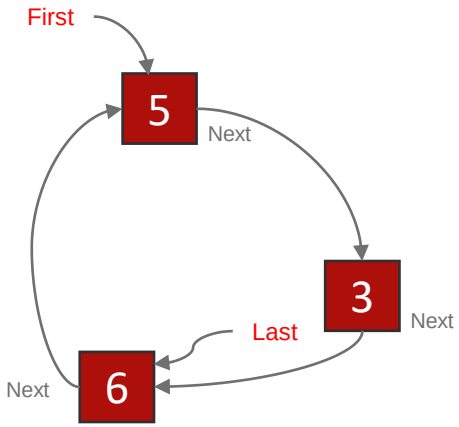
Operaties met een DLL

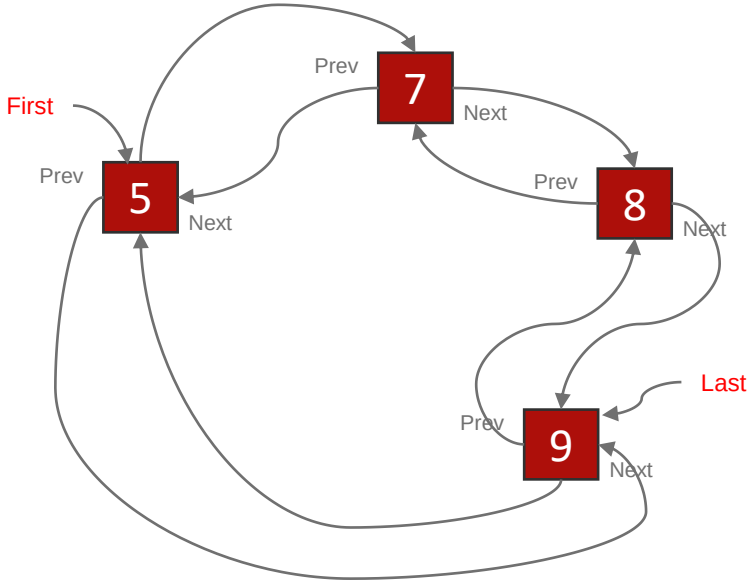
- Nodes toevoegen, invoegen of verwijderen



Circulaire SLL of DLL

- Er bestaan ook circulaire versies van beide types





Time complexity

Operatie	Array	Singly Linked List	Doubly Linked List
Opzoeken “by index”	O(1)	O(n)	O(n)
Opzoeken ”by value”	O(n) (*)	O(n)	O(n)
Element achteraan toevoegen	O(1)	O(1)	O(1)
Element vooraan invoegen	O(n)	O(1)	O(1)
Element invoegen na een bepaalde index/node	O(n)	O(1)	O(1)
Element invoegen voor een bepaalde index/node	O(n)	O(n)	O(1)
Element verwijderen	O(n)	O(n)	O(1)

(*) we zien later bij “search” algoritmes dat een veel betere complexity haalbaar is onder bepaalde omstandigheden

“Memory usage”

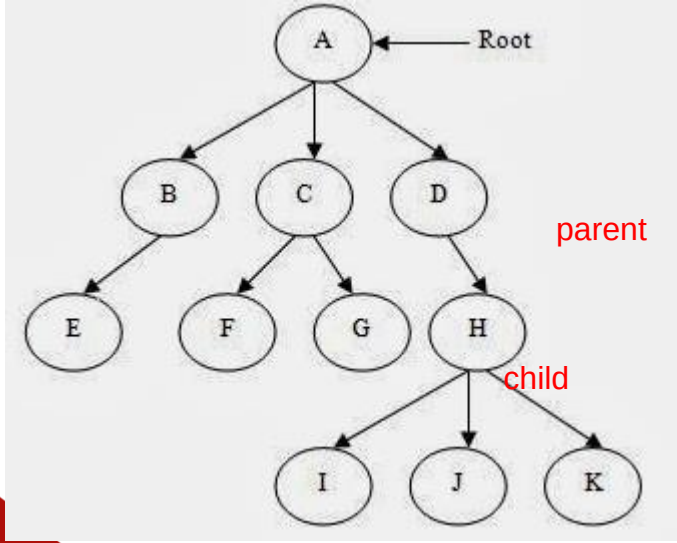
	Array	Singly Linked List	Doubly Linked List
Lege lijst	Vast aantal elementen op voorhand te reserveren. Zolang de array niet volzet is wordt er geheugen gereserveerd dat niet wordt gebruikt.	Geen reservatie op voorhand nodig. Geheugen wordt gebruikt naarmate er nodes bijkomen of verdwijnen.	
Wat “kost” 1 element	Enkel geheugen nodig om 1 “value” in te kunnen bewaren per element. (dus bv. 4 bytes voor een int,..)	Elke node heeft geheugen nodig voor de “value” + referentie naar de next node	Elke node heeft geheugen nodig voor de”value” + 2 referenties (next en previous)
Element toevoegen of invoegen	Geen extra geheugen nodig, tenzij de array volzet is. Dan moet er een nieuwe array worden aangemaakt en moet alle data worden over gekopieerd.	Extra geheugen nodig telkens voor 1 “node”	
Element verwijderen	Er komt geen geheugen vrij, tenzij de array zou verkleind worden als deze bv. achteraan leeg is, maar dat betekent ook weer de data over kopiëren.	Geheugen van 1 node wordt telkens terug vrijgegeven	

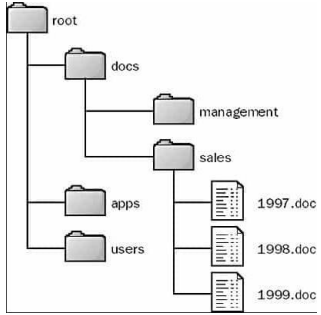
Wanneer / wat gebruiken

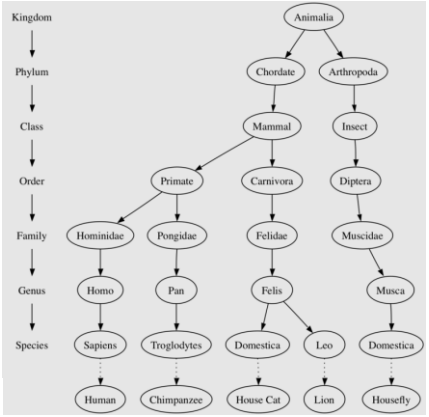
Wat	Array	Singly Linked List	Doubly Linked List
Queue	✓	✓	✓
Stack	✓	✓	✓
Bubble sort	✓	✓	✓
Selection sort	✓	✓	✓
Insertion sort	✓	✓	✓
Quick sort	✓	✓	✓
Merge sort	✓	✓	✓

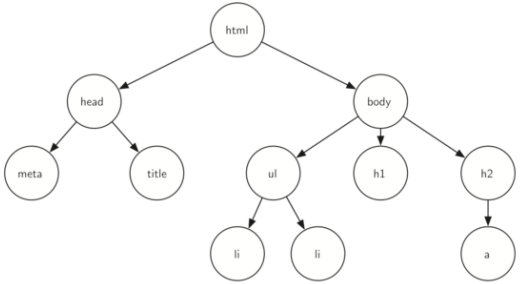
Boomstructuur

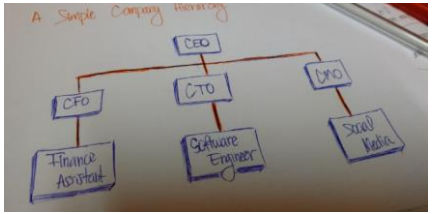
- Een boomstructuur is een hiërarchische (dus niet-lineaire) datastructuur bestaande uit “nodes”





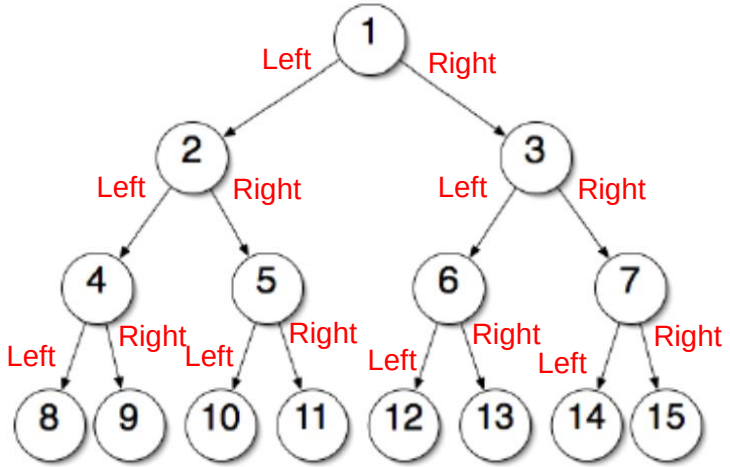






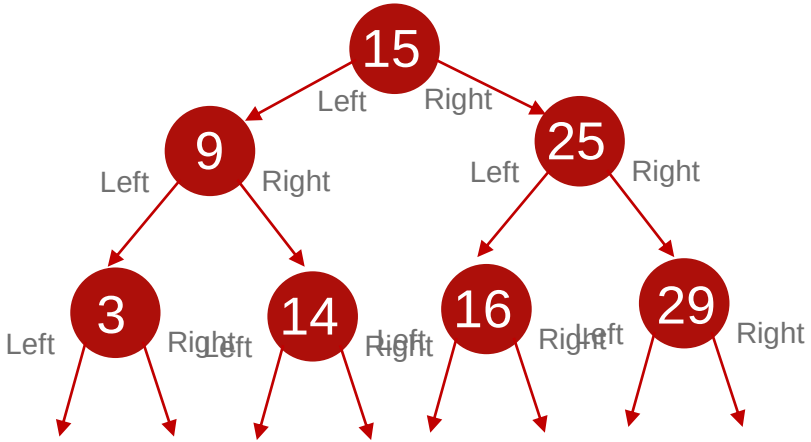
Binary tree (Binaire boom)

- Bij een binaire boom zijn heeft een “parent” maximum 2 “children”
- Men spreekt dan over de
  - “Left child”
  - “Right child”



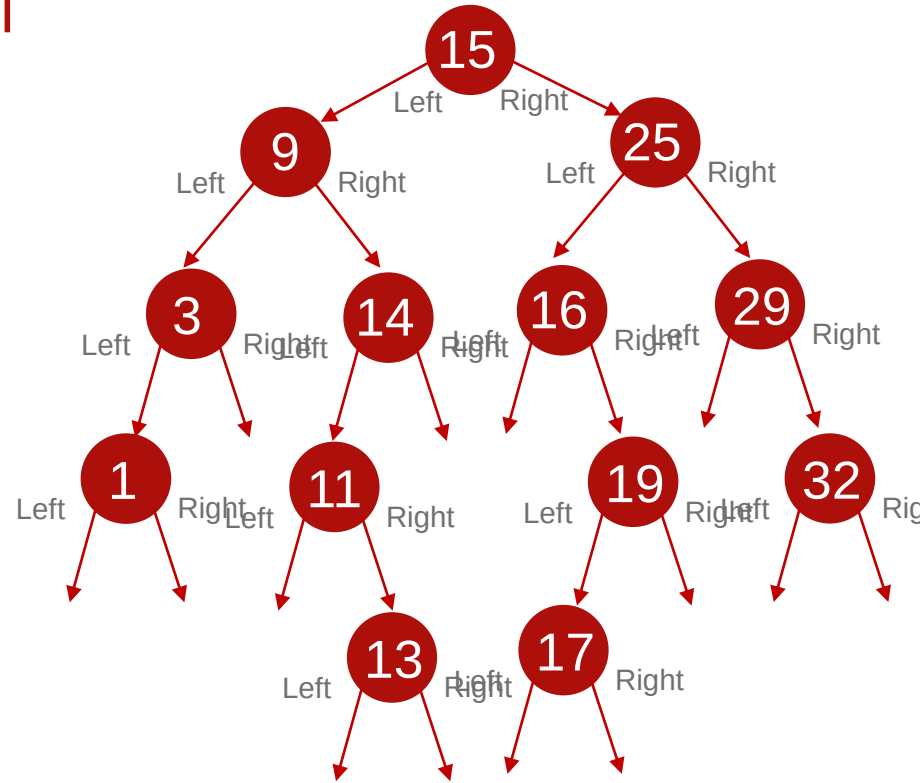
“Sorted binary tree” of “Binary search tree” (BST)

- Bij een “sorted binary tree” is **ten allen tijde**:
  - de **waarde** van de “**left child**” **kleiner** dan die van de “**parent** node”
  - De **waarde** van de “**right child**” **groter** dan die van de “**parent** node”



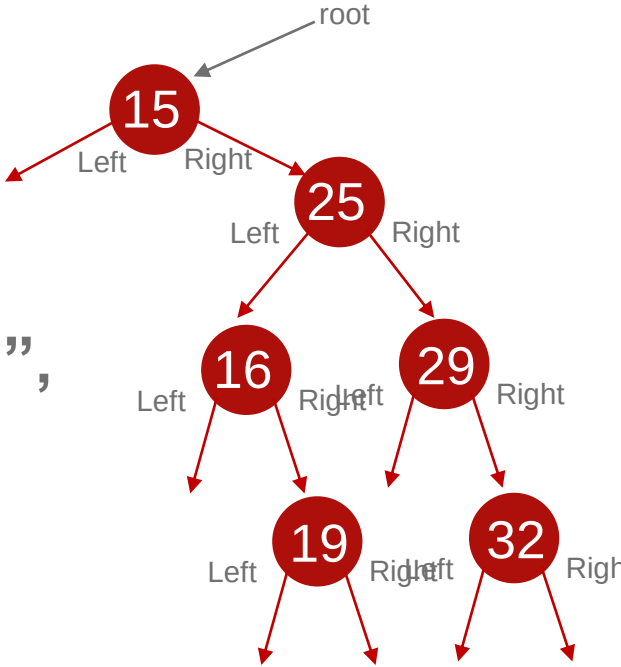
Opbouwen van een BST

- Insert (15) : => root node
- Insert (9) : 9 < 15 => left child
- Insert (25) : 25 > 15 => right child
- Insert (3)
- Insert (16)
- Insert (1)
- Insert (29)
- ….



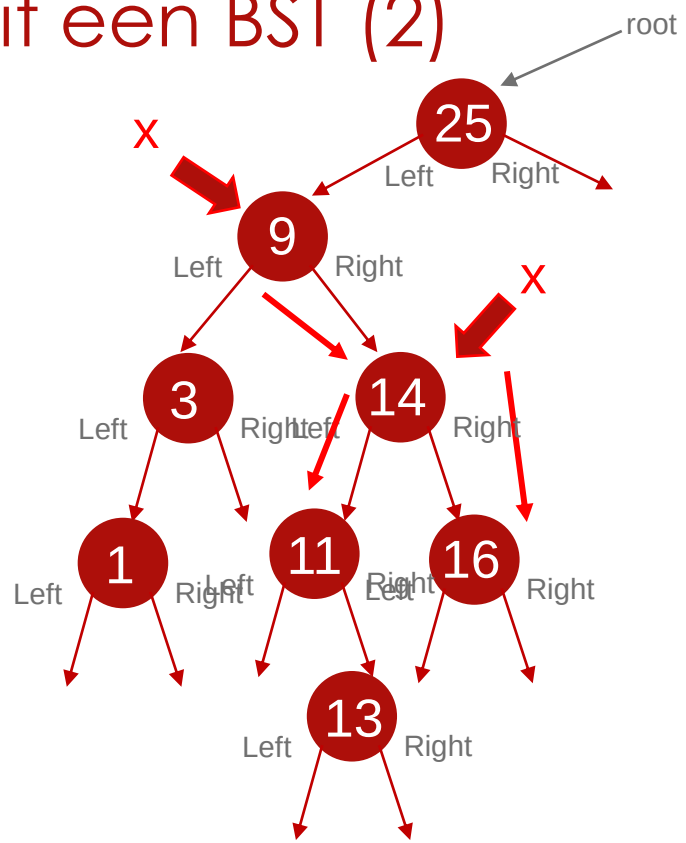
Verwijderen van elementen uit een BST

- Indien de te verwijderen node
  - Een “parent” is **zonder “children”**, dan kan de node gewoon verwijderd worden.
  - Een “parent” is **met slechts “1 child”,** dan kan de “parent“ gewoon verwijderd worden

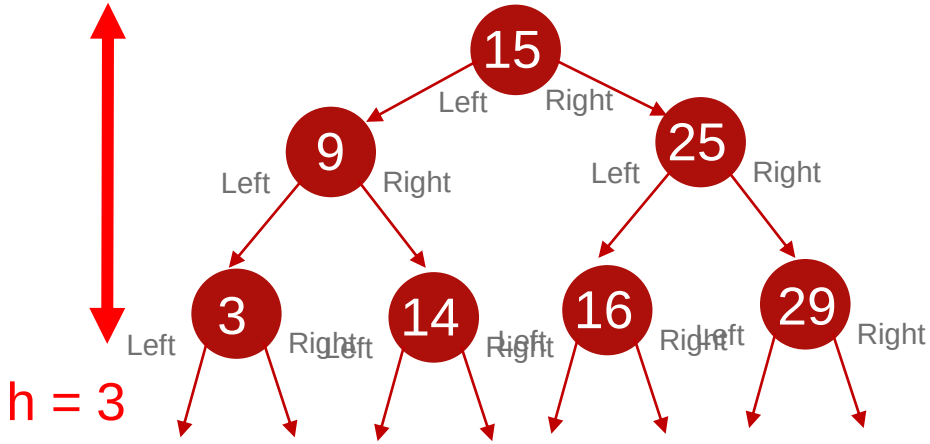


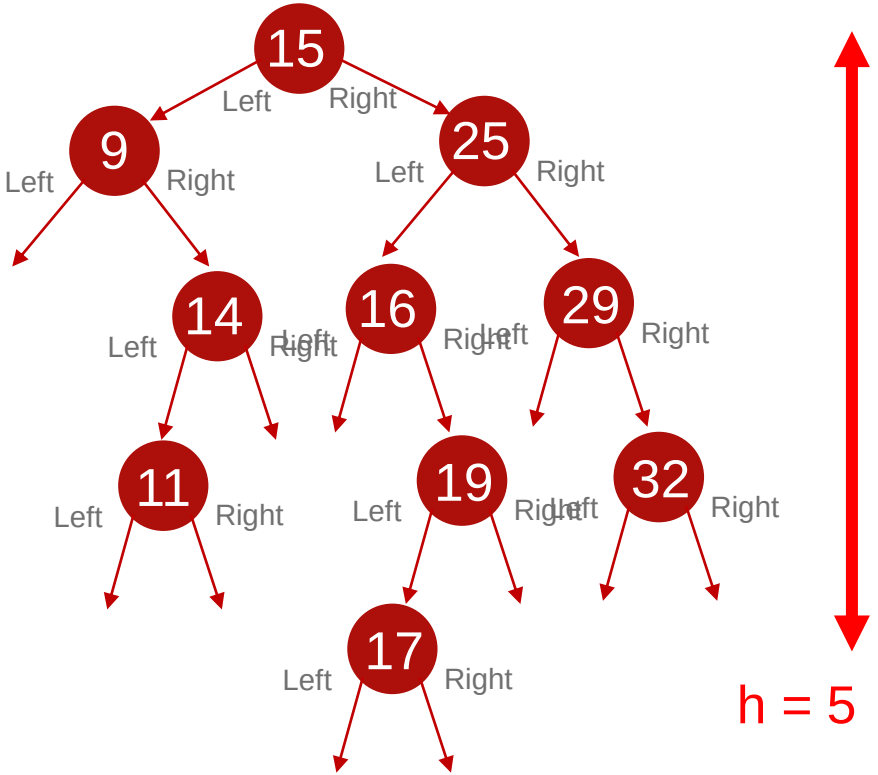
Verwijderen van elementen uit een BST (2)

- Indien de te verwijderen node een “parent” is **met 2 “children”**
  - Dan gaan we eerst op zoek naar het **kleinste** element bij de “**right children”** van die node**.**
  - Dit kleinste element zal de node vervangen.



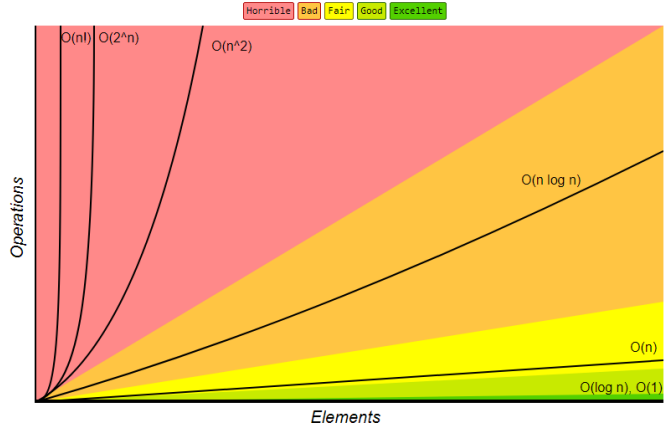
Balanced vs. Unbalanced BST





Time complexity BST

- Zoeken, toevoegen of verwijderen.
- Worst case: O(n)
  - Totaal ongebalanceerde BST
  - De BST is herleid tot een “linked list”
- Average case: O(log n) of ook O(h)
  - Bij een gebalanceerde BST.



N	Max. aantal compares (h)
7	3
15	4
512	9
1.000.000	20

AP HOGESCHOOL
ANTWERPEN