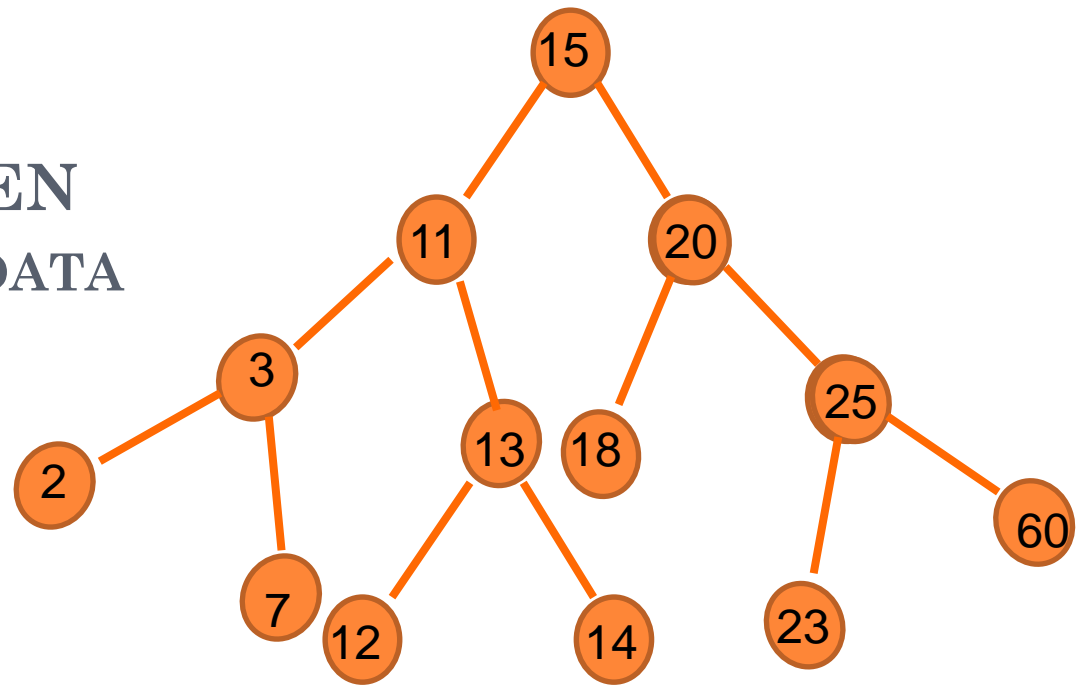


# DATASTRUCTUREN VOOR GESORTEERDE DATA



1

Dr. D.P. Huijsmans

18 sept 2013

Universiteit Leiden, LIACS

# ONDERWERPEN 18 SEPT

- - Tot nu toe rijen/lijsten met beperkte toegang
- - stack: bovenaan (LIFO)
- - cyclisch array of lijst: vooraan/achteraan (FIFO)
- - linked list implementatie DCLL
- (zie ADT uitwerkingen)
- - geheugengebruik beperken: array -> verbonden lijst
- - geordende lijsten: tussenin
- - nadeel sortering bij verbonden lijst:  $O(n)$  zoeken
- - binary search voordeel sortering -> BST:  $O(\log n)$

# MAGNEETBAND ANALOGON VOOR RECORDS (VASTE OF VARIABELE LENGTE)

- In plaats van een serie bytes kan een magneetband ook gedacht worden als een sequentie van records:
- - van vaste lengte (steeds m bytes/record lezen/schrijven)



- - van variabele lengte (steeds zoveel bytes lezen totdat EOR byte code bereikt is)



- -  $O(N)$  toegang (serieel voor/achteruit)

# OVERGANG MAGNEETBAND NAAR HARDE SCHIJF

- Magneetband: toegang serieel
- Harde schijf: opgedeeld in sectoren (typisch veelvoud van 512 bytes elk) die random-access toegankelijk zijn
- Overgang jaren (19)70

# ADT\_SINGLELINKEDRECORDSFILE

## RANDOMACCESS SCHIJFGEHEUGEN

- In plaats van byte voor byte gaat de I/O met vaste of variabele lengte records (EOR) die op het eind een pointer hebben naar het volgende record (NR);
- Laatste record in de rij bevat als next pointer een NR=EOF
- I/O functie die als bij een byte stream file met als uitwisselenheid het record (i.p.v. de byte) en beperking tot alleen m records voorwaarts kunnen “spoelen”

record	Bloknr nextrec
1	456
2	3
3	2299
4	16
5	EOF

# ADT\_RANDOMACCESSFILE

## RANDOMACCESSSCHIJFGEHEUGEN

- Vaste en ook variabele lengte records kunnen sneller benaderd worden bij een opzet waarin naast de afzonderlijke records een array met record-keys wordt bijgehouden; vanuit dit array kan elk record direkt benaderd worden via de pointer die bij de sleutel(key) opgeslagen wordt.
- Array van sleutels kan gesorteerd gehouden worden voor snel opzoeken bepaalde key.

key	1e blok record
3	4050
7	13
12	20134
18	5
76	6789

# UITBUITEN VOORDELEN GESORTEERDE LIJST DOORZOEKEN

- Opgeslagen data elementen oplopend op waarde gesorteerd (geïndexeerd)
- Maakt snel zoeken naar aanwezigheid en positie gezochte waarde mogelijk in array
- Methode binary search (herhaalde tweedeling):
  - Bekijk middelste waarde en vergelijk met gezochte
    - Als gelijk -> klaar return index
    - Als waarde < gezochte bekijk middelste linkerdeel
    - Als waarde > gezochte bekijk middelste rechterdeel
- Randvoorwaarden:
  - Rij even/oneven aantal midden?
  - stopcriterium

# BINAIR ZOEKEN NAAR “GEZOCHTE WAARDE” IN GESORTEERD ARRAY $A[1..N]$

- BS Algoritme:
- Initieel:
  - $Stap=N$ ;  $index=0$ ;  $Dir=+1$ ;
- Herhaal zolang  $Stap>1$ ; anders  $return(0)$ :
  - Halveren  $Stap$ (grootte) en aanpassen Index:
    - Functie  $half(stap)$ :
      - $Stap$  even  $\rightarrow Stap/2$ , anders  $(Stap+1)/2$
    - $Index=Index+Dir*Stap$ ;
  - Test = :
    - Als  $A[index]=$ “gezochte waarde”  $\rightarrow return(index)$
  - Richting volgende aanpassing index als nog niet gevonden:
    - $Dir=-1$ ; als  $A[Index]<$ “gezochte waarde”  $Dir=+1$



# GESORTEERD ARRAY ZOEK VOORBEELD

Zoeken naar  $A[\text{index}] = \text{"i"}$

a	b	c	d	e	f	g	h	i	j	k
					6					
								9		

i?	init	ronde1	ronde2
Stap	11	6	3
Index	0	6	9
Test		<	=
Dir	1	1	
Return	0	0	9

# GESORTEERD ARRAY ZOEK VOORBEELD

Zoeken naar  $A[\text{index}] = \text{"h"}$

a	b	c	d	e	f	g	h	i	j	k
					6					
								9		
						7				
							8			

h?	init	ronde1	ronde2	ronde3	ronde4
Stap	11	6	3	2	1
Index	0	6	9	7	8
Test		<	>	<	=
Dir	1	1	-1	1	
Return	0	0	0	0	8

# VERBORGEN VERONDERSTELLING

## BINARY SEARCH OP GESORTEERD ARRAY

- Wat gebeurt er als waarden meermalen voorkomen?
- Hoe zou je de datastructuur aanpassen zodat alle voorkomens van een bepaalde waarde gerapporteerd kunnen worden?

# BINAIR ZOEKALGORITME OP GESORTEERD ARRAY SLUITEND?

- Zelfs bij veronderstelling dat elke waarde in het array maar 1 maal voorkomt
- Zelf uitzoeken komende 5 minute (met BS algoritme op volgende slide):
- Ga na hoe algoritme uitpakt voor gesorteerde array's die maar 1,2 of 3 lang zijn;
- Kun je in die gevallen waar het mis loopt, het algoritme zo aanpassen dat het in alle gevallen correct werkt?

# BINAIR ZOEKEN NAAR “BEPAAALDE WAARDE” IN GESORTEERD ARRAY $A[1..N]$

- BS Algoritme met tekortkomingen:
- Initieel:
  - $Stap=N$ ;  $index=0$ ;  $Dir=+1$ ;
- Herhaal zolang  $Stap>1$ ; anders  $return(0)$ :
  - Halveren  $Stap$ (grootte) en aanpassen Index:
    - Functie  $half(stap)$ :
      - $Stap$  even  $\rightarrow$   $Stap/2$ , anders  $(Stap+1)/2$
    - $Index=Index+Dir*Stap$ ;
  - Test = :
    - Als  $A[index]=$ “bepaalde waarde”  $\rightarrow$   $return(index)$
  - Richting volgende aanpassing index als nog niet gevonden:
    - $Dir=-1$ ; als  $A[Index]<$ “bepaalde waarde”  $Dir=+1$

# GESORTEERD ARRAY N=3 VOORBEELD

a	b	c
	2	
1		3

?	init	ronde1	ronde2	ronde2	ronde3
Stap	3	2	1	1	stop
Index	0	2	1 of 3	1 of 3	
Test		<>	=	<>	
Dir	1	++1		--1	
Return	0	0	a of c	0	

# GESORTEERD ARRAY N=2 VOORBEELD

b	c
1	

a?	init	ronde1	ronde2
Stap	2	1	stop
Index	0	1	
Test		<	
Dir	1	-1	
Return	0	0	0

c?	init	ronde1	ronde2
Stap	2	1	stop
Index	0	1	
Test		>	
Dir	1	1	
Return	0	0	0

Element 2 kan niet bezocht worden

# GESORTEERD ARRAY N=1 VOORBEELD



b?	init	ronde1
Stap	1	stop
Index	0	
Test		
Dir	1	
Return	0	0

Element niet gevonden!

Stopcriterium  $\text{step} > 1$  faalt bij  $n=1$  of  $2$

Voor  $N=1$  kan stap initieel minstens 2 zaak oplossen, Maar voor  $N=2$  blijft 2e element buiten bereik

Vooraf test of  $A[N]$  voldoet zou dit oplossen, maar duur gezien  
Extra stap



# CORRECT

## BINARY SEARCH ALGORITME VOOR ELKE N

- In “programming pearls” staat een betere aanpak:
- $A[N]$  gesorteerd array  $N=[1..Max]$
- Gezocht index van  $A[i]=target$
- Initieel:  $low=1, high=Max$  (in C++:  $0$  en  $Max-1$ )
- Zolang  $low \leq high$ 
  - $mid=(low+high)/2$  (naar beneden afgerond)
  - Als  $A[mid]=target$  return( $mid$ )
  - Anders
    - Als  $A[mid]<target$   $low=mid+1$
    - Else  $high=mid-1$

Return(0)

# LAAT ONTDEKTE BUG IN BINARY SEARCH ALGORITME

- Pas laat is nog een mogelijke tekortkoming in dit voorbeeld algoritme ontdekt:
- De regel:  $mid = (low + high) / 2$  kan als  $mid$  een 16 of 32 bit Integer mogelijk leiden tot overflow bij de bewerking  $low + high$ ;
- Oplossing: een echt voor alle gevallen correct algoritme zou  $mid$  als volgt kunnen bepalen:
  - Of  $mid = low / 2 + high / 2$  of
  - Of  $mid = low + (high - low) / 2$  (nog efficiënter)

# CORRECT BINARY SEARCH ALGORITME VOOR ALLE N [1..MAX] VOORBEELD

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
							8							
											12			
													14	
												13		

m?	initieel	ronde1	ronde2	ronde3	ronde4
Mid		8	12	14	13
Test		<	<	>	=
Low	1	9	13	13	
High	15	15	15	13	
return	0	0	0	0	13

# BINARY SEARCH ZOEKEN BIJ EEN GESCHIKTE VERBONDEN STRUCTUUR

- Nadeel enkel- of dubbel-verbonden lijsten met gesorteerde rij elementen
- Geen binary search mogelijk
- Alternatieve opzet voor met pointers verbonden gesorteerde elementen
- Een twee-deel element met pointers

# BINAIRE ZOEK BOOM (KNOOP STRUCTUUR)

- Ouder knoop met maximaal 1 voorouder en maximaal 2 kinder knopen
- Als eenzijdig verbonden structuur:

Kind-links pointer	Ouder waarde	Kind-rechts pointer
--------------------	--------------	---------------------

Verbinding:  
edge  
link  
arc

- Als tweezijdig verbonden structuur:

	Kind-van pointer	
Kind-links pointer	Ouder waarde	Kind-rechts pointer

Ouder waarde:  
key:datarecord

- Leeg knoop element

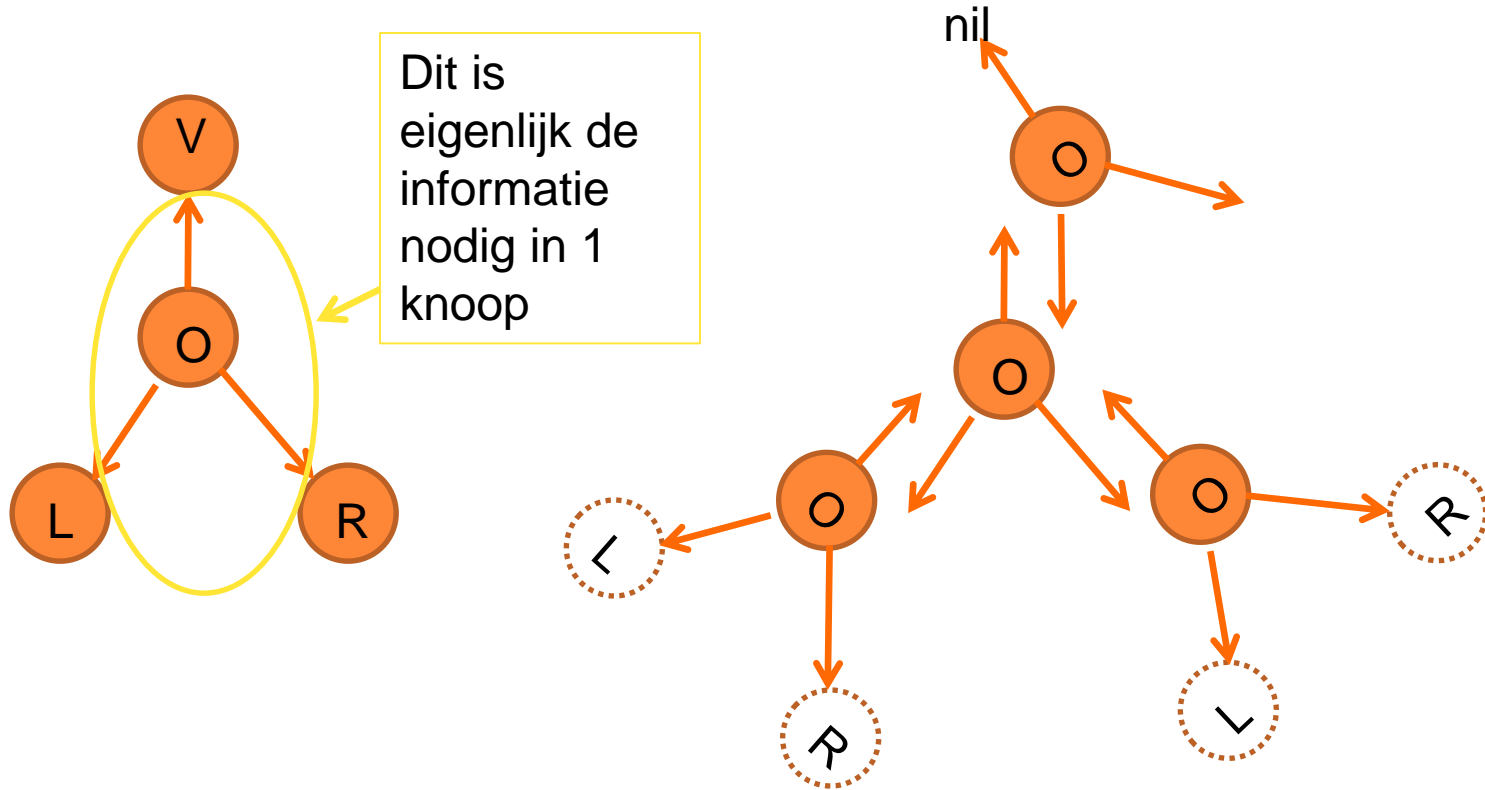
	nil pointer	
nil pointer	nil waarde	nil pointer

root-element

	nil pointer	
Kind-links pointer	key	Kind-rechts pointer

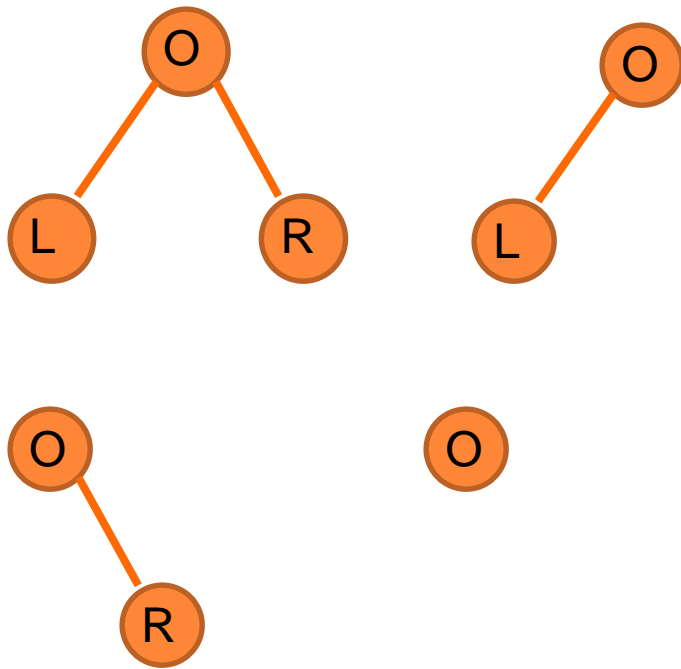
# BINARY SEARCH TREE

OUDER KNOOP MET 2 KINDER KNOPEN (L&R)  
EN 1 VOOROUDERKNOOP



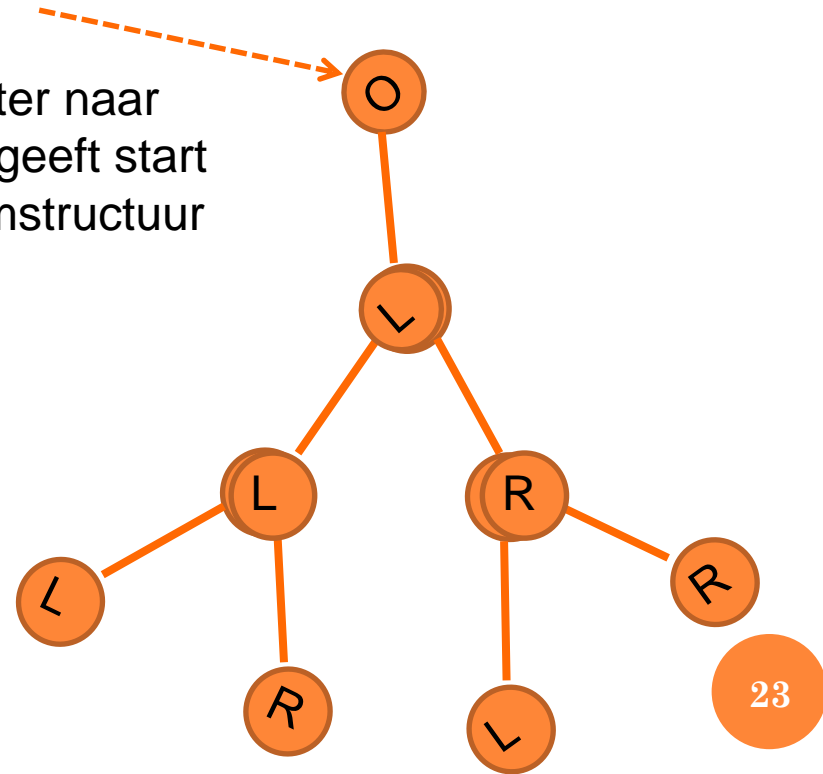
Door samen delen van gerichte pointers kan een ongerichte binaire boom structuur worden opgebouwd die zowel van boven naar beneden, van beneden naar boven en langs een willekeurig pad doorlopen kan worden

# VEREENVOUDIGDE GRAFISCHE VOORSTELLING KNOOP IN BST



Voorouder pointer niet apart nodig als  
adres voorouder op stack gezet wordt

Pointer naar  
root geeft start  
boomstructuur  
aan



Bouwstenen zonder weergave nil pointers;  
voorouder pointer en richting pointer

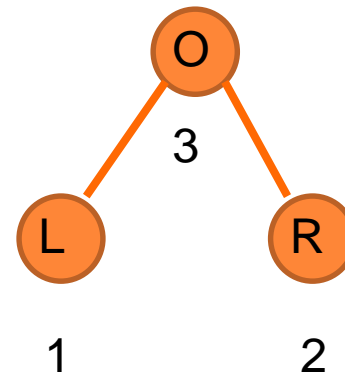
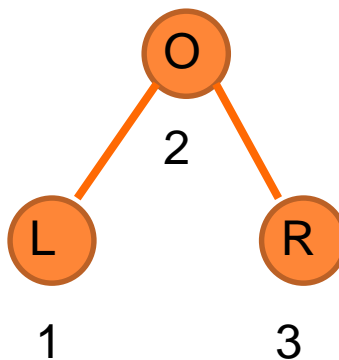
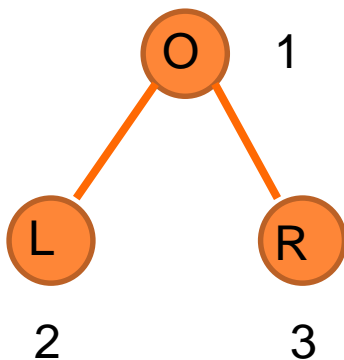
# PADEN IN EEN BOOM

- Knoop zonder voorouder is root (wortel)
- Knoop met als kinderen nil pointers is leaf (blad)
- Geordende Boom:
  - kindknopen in bepaalde volgorde (van L naar R b.v.)
- Simpel pad:
  - verbinding tussen root en leaf waarbij een verbinding maar 1 maal gebruikt wordt
- Lengte van het pad:
  - Aantal knopen langs pad
- Hoogte (height) van een Tree:
  - Aantal knopen in langste simpel pad
- Niveau (level) van een Tree:
  - Knopen op zelfde simpel pad afstand van wortel



# HANDIGE AFLOOPVOLGORDES BST BOMEN

- Paden waarin elke knoop precies 1 maal opgenomen wordt ook al wordt hij vaker gepasseerd:
- Ouders gaan voor; niveau voor niveau geordend (breadth first):
  - m.b.v. FIFO lijst
- Kinderen gaan voor (depth first) m.b.v. stack:
  - Preorder: OLR
  - Inorder:LOR voor gesorteerde lijst in BST
  - Postorder:LRO



# BST

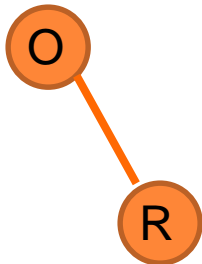
## TOEVOEGEN KNOOP MET WAARDE KEY

- Mogelijke beginsituaties na zoeken invoegpositie (currentknoop):
- Lege boom: creëer een (root)knoop met waarde key; pointers naar kinderen nil, rootpointer van nil -> (root)knoop.
- Moet in linker subtree van current knoop: creëer nieuwe knoop waarvan key de waarde wordt; linkerpointer currentknoop -> nieuwe knoop; nieuwe knoop met overname van linkerpointer current knoop en nil pointer rechter; currentknoop-> nieuwe knoop
- Moet in rechter subtree van currentknoop: analoog aan linker toevoeging

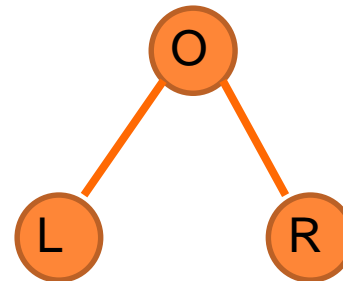
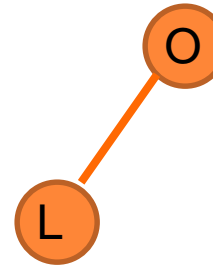
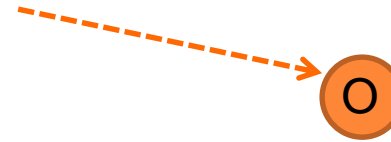
# BST: ROOT OF LEAF

## LINKS INVOEGEN BIJ CURRENT KNOOP

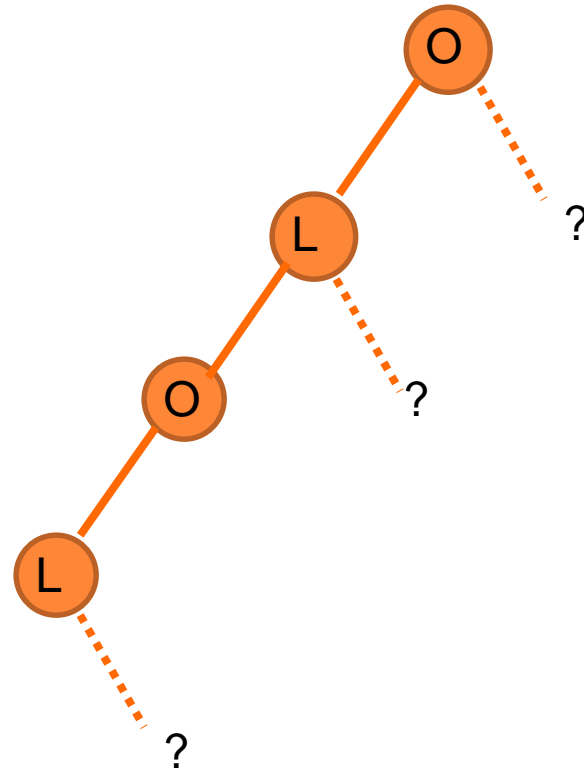
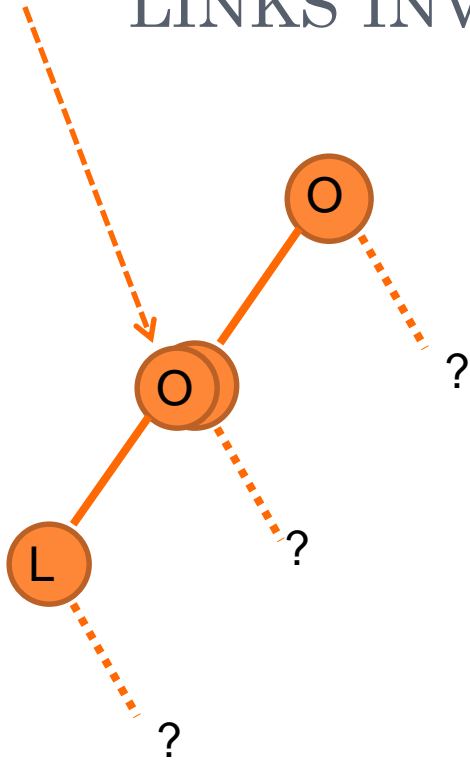
Voor:



Na:



# BST: NIET ROOT/LEAF INVOEGING LINKS INVOEGEN MET LEGE RECHTER

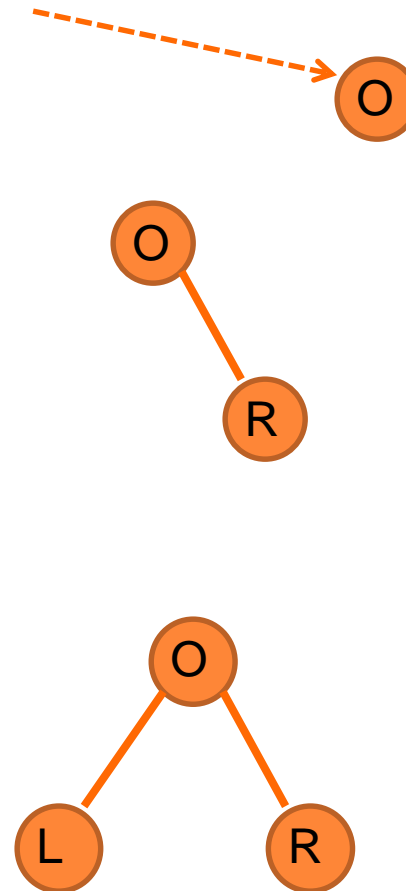


# BST: RECHTS INVOEGEN BIJ CURRENT KNOOP

Voor:



Na:



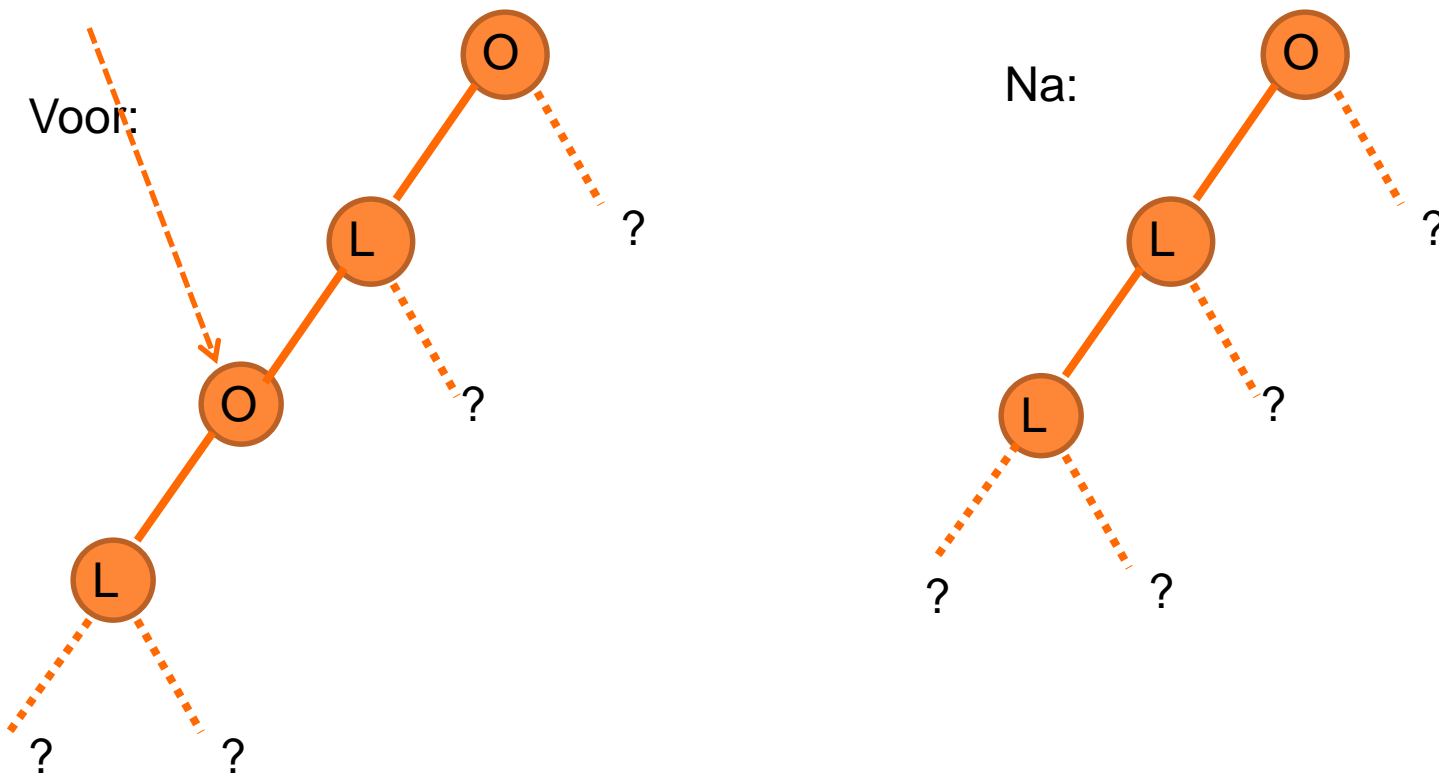
# BST:

## WEGHALEN WAARDE AAN BEGIN/EIND

- Weglaten van een waarde in een BST zou kunnen gebeuren door de opgeslagen waarde in een knoop op nil te zetten
- Als we ook een knoop weg willen halen uit de BST dan moeten we de volgende gevallen onderscheiden:
- BST is leeg -> niets doen
- BST alleen een root knoop, delete root knoop, zet BST pointer naar nil, size -> 0
- Knoop is leaf (nil pointers als kinderen): knoop daarboven die hier naar toe wees -> nil

# BST WEGHALEN WAARDE ONDERWEG

- Delete is alleen makkelijk als de weg te halen waarde in een knoop zit met alleen een linker of een rechter subtree:



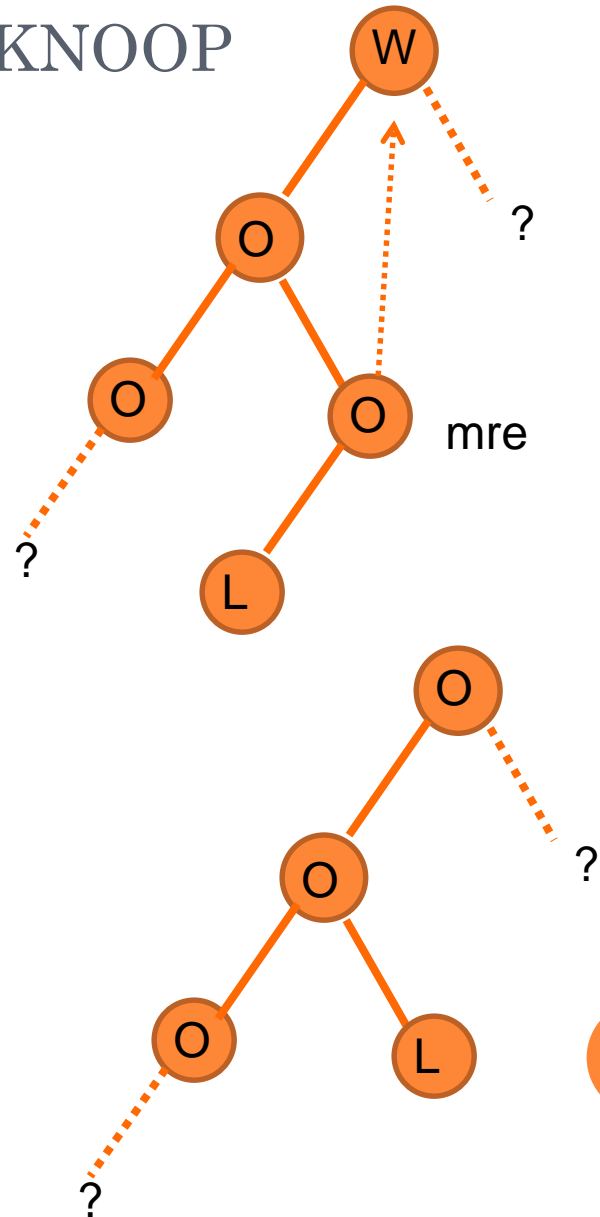
## BST: DELETE KNOOP MET 2 SUBTREES

- Als er onder een weg te halen knoop twee subtrees hangen kan maar 1 ervan makkelijk 1 nivo naar boven opschuiven; wat te doen met de elementen in de overblijvende subtree?
- Oplossing 1: verhuis waardes binnen boom
- Oplossing 2: loop de resterende subtree af en voeg elk van z'n knoopwaardes 1 voor 1 aan de BST toe
- Oplossing 3: merge de 2 resterende bomen



## BST: DELETE ROOT KNOOP

- Net zo lastig en vergelijkbaar is de situatie in een goed gevulde gebalanceerde boom waarin we de root knoop willen weghalen.
- Oplossing: kopieer meest rechtse element in linker subtree naar root en delete de knoop waarin mre oorspronkelijk stond

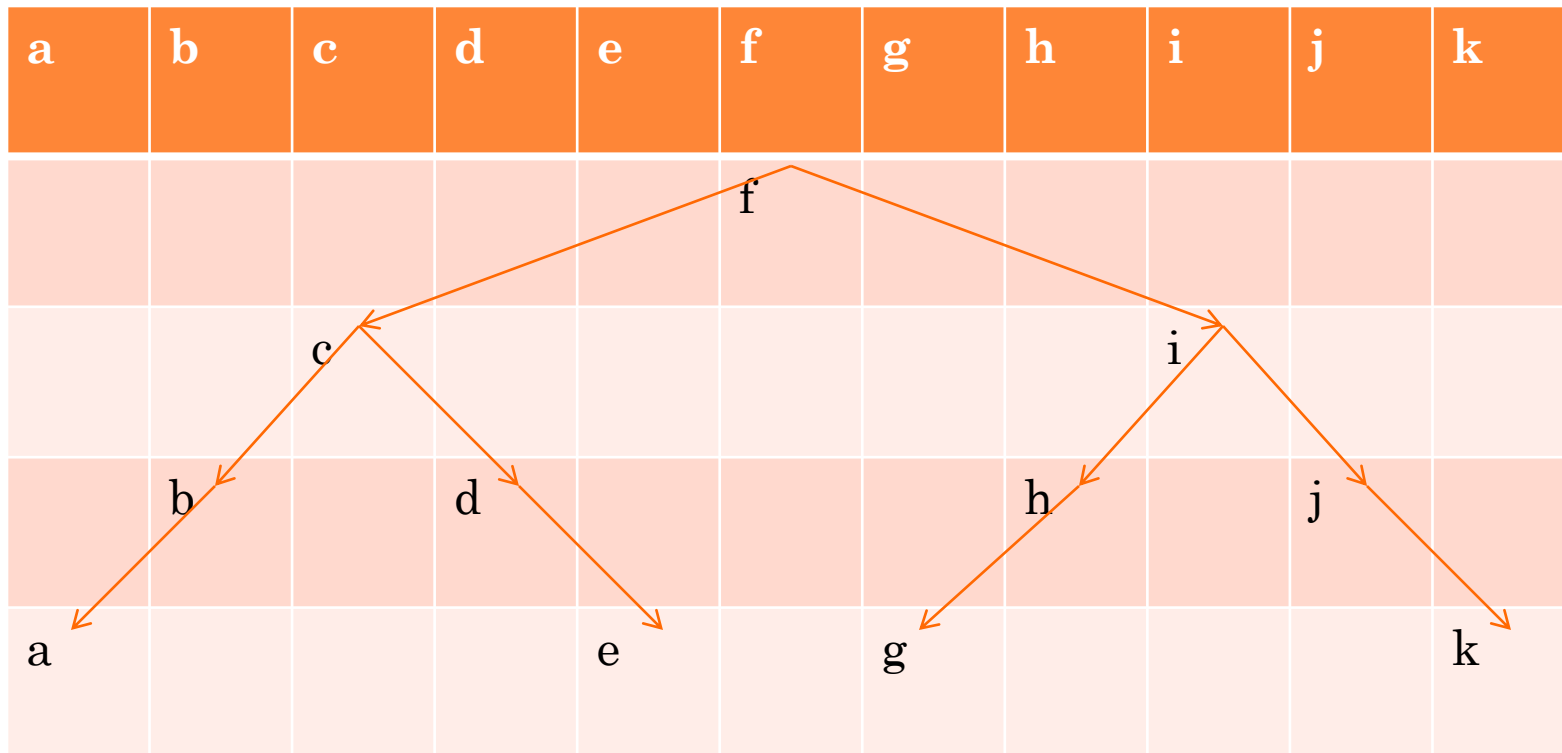


# MOGELIJKE BINAIRE BOOM OPSLAG VOORBEELD UITWERKEN

a	b	c	d	e	f	g	h	i	j	k

Hoe ziet de BST eruit bij de volgende  
Invoegvolgorde: f,c,i,b,d,h,j,a,e,g,k ?

# MOGELIJKE BINAIRE BOOM OPSLAG VOORBEELD



Gebalanceerde BST:  
Invoegvolgorde: f,c,i,b,d,h,j,a,e,g,k  
Zoeken hierin bijna  $O(\log N)$

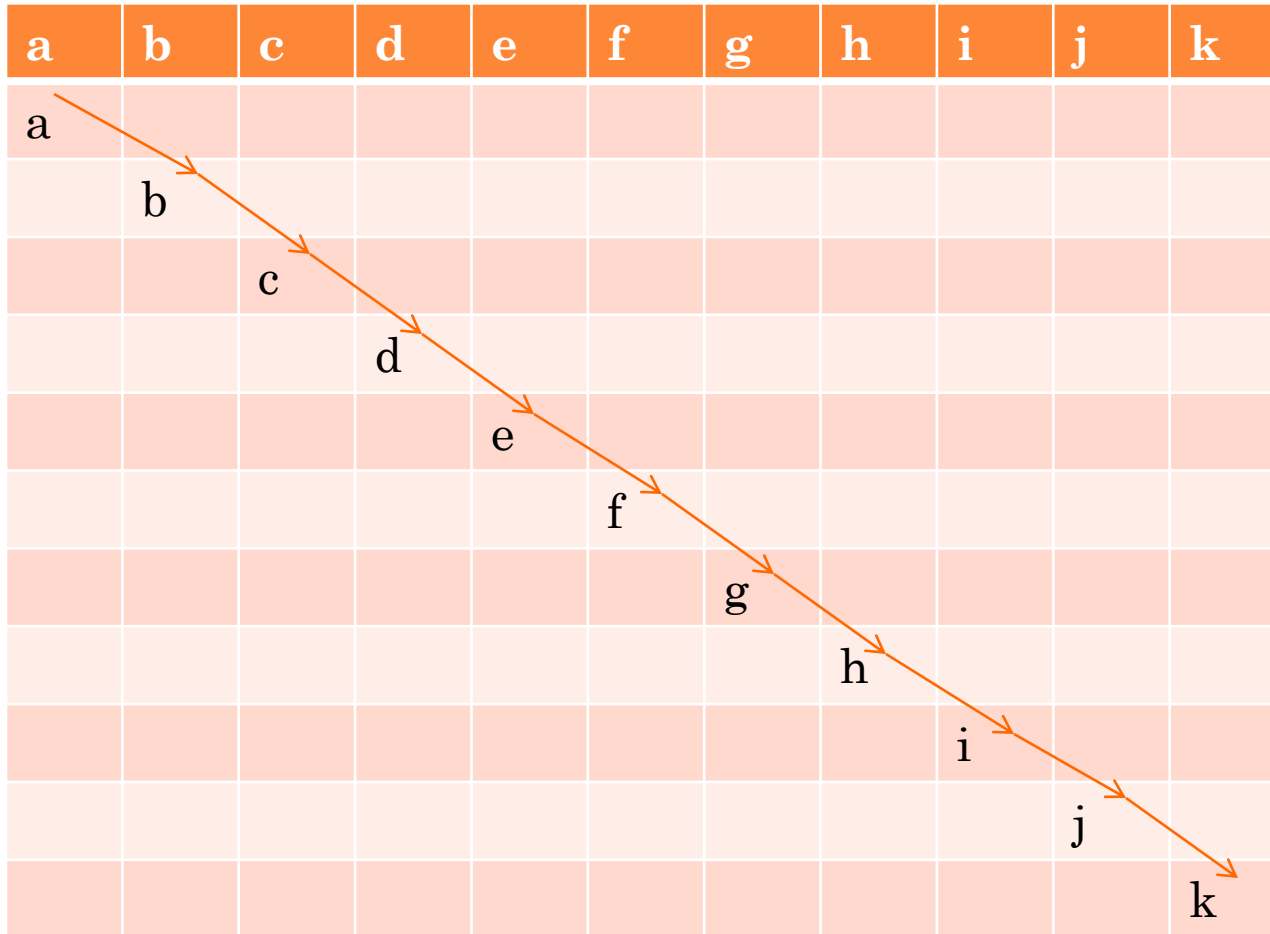
# MOGELIJKE BINAIRE BOOM OPSLAG VOORBEELD UITWERKEN

a	b	c	d	e	f	g	h	i	j	k

Hoe ziet de BST eruit bij de volgende  
Invoegvolgorde: a,b,c,d,e,f,g,h,i,j,k ?

# ONGEBALANCEERDE OPSLAG IN BST

VERGELIJKBAAR MET GESORTEERDE ENKEL VERBONDEN LIJST



Dit resultaat heet ook wel “backbone structuur” (komt terug!)

Invoegvolgorde: a,b,c,d,e,f,g,h,i,j,k

Zoeken hierin  $O(N)$

# AANTAL TWEEDELINGEN EN AANTAL KNOPEN IN VOLLE BST

- Wortel (root):  $n=1$  knoop (niveau  $h=1$ )
- 1 tweedeling:  $n=3$  knopen (niveau  $h=2$ ):  $1+2$
- 2 tweedelingen:  $n=7$  knopen (niveau  $h=3$ ):  $1+2+4$
- .....
- $h-1$  tweedelingen:  $n=2^h-1$  knopen (niveau  $h=\log_2(n+1)$ )

# GESORTEERD EN GEBALANCEERD HOUDEN

- Zaak is algoritmes voor BST zo te ontwerpen dat:
- Afloopvolgorde en sortering samenvallen
- De boom zo gebalanceerd mogelijk blijft:
  - Bij opbouw van de BST
  - Bij wijzigingen op de BST

# BST OPBOUWEN

## VANUIT GESORTEERD ARRAY

- Een goed gebalanceerde BST kan makkelijk worden opgebouwd als de waardes vooraf bekend zijn
- Zet de waardes in een array
- Sorteert het array
- Voeg elementen aan de BST toe door het gesorteerde array recursief met binair zoekadressering te doorlopen

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
							1							
			2								9			
	3				6				10				13	
4		5		7		8		11		12		14		15



# DROZDEK OVER BST

- Binary Search Tree
- H6.1, H6.2, H6.3