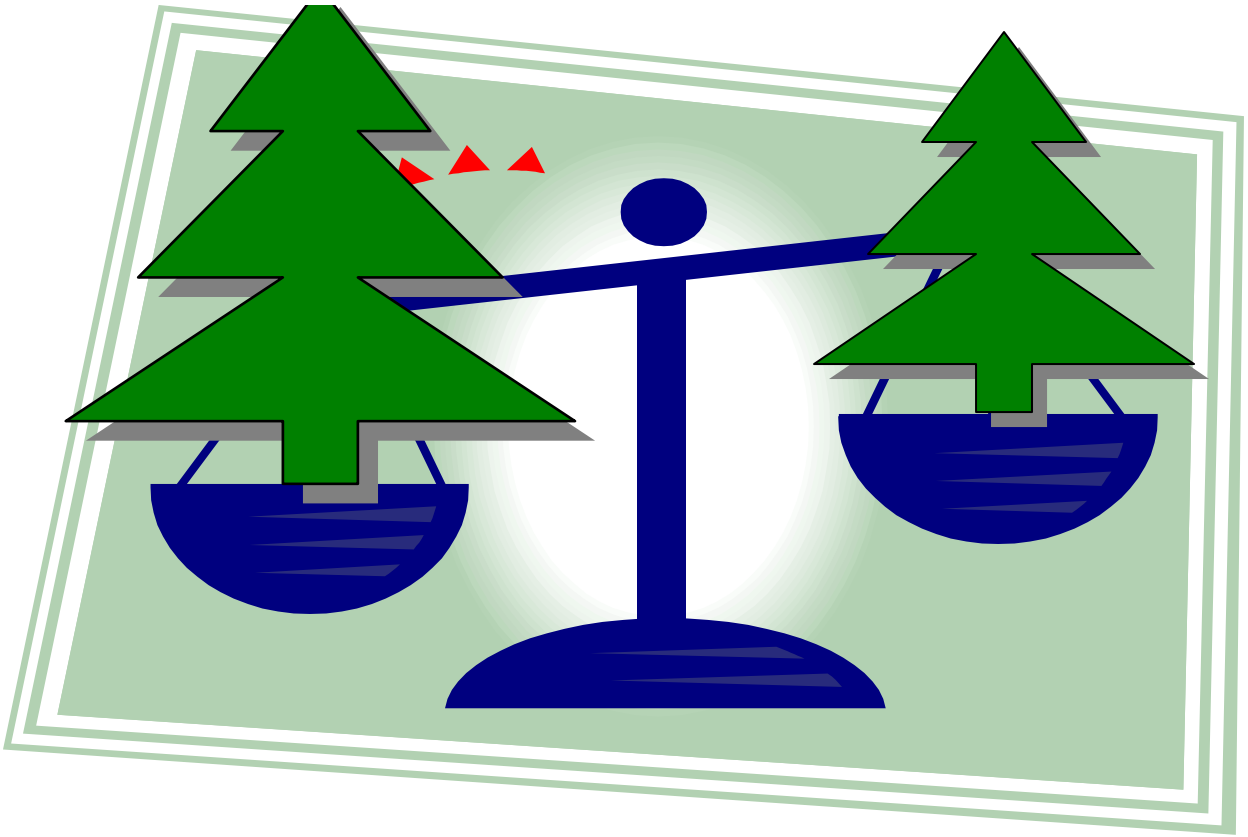
Datastructuren
Balanceren van bomen



Dr. D.P. Huijsmans

4e college 25 september 2013

Universiteit Leiden, LIACS

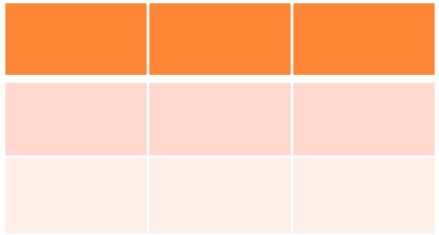
Balanceren en binary search

- Om O(logN) te kunnen zoeken moet de data gesorteerd zijn
- In een binary search tree (BST) InOrder doorlopen is de data gesorteerd toegankelijk
- De slechts denkbare BST realisatie na invoegen is een boom met allemaal lege linker of allemaal lege rechterknopen (backbone structuur)
- De complexiteit van zoeken bij een backbone BST is O(N)
- O(logN) <= Complexiteit zoeken BST <= O(N)
- Om in een BST in O(logN) te kunnen zoeken moet deze boom maximaal gebalanceerd zijn

Aantal BST's met zelfde inhoud

- Gesorteerd array met unieke gesorteerde waarden kent maar 1 verschijningsvorm:

- Hoeveel vormen kan de BST aannemen als deze 3 waarden in willekeurige volgorde binnenkomen?

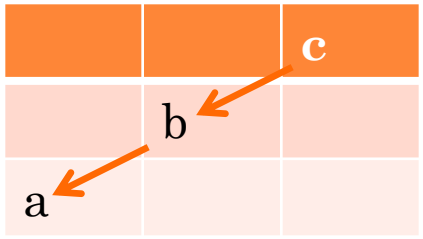


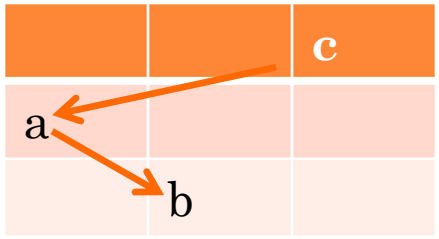
- Wat is de kans op een min-pad vorm?

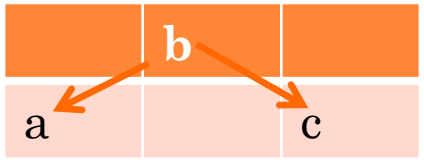
Aantal BST's met zelfde inhoud

- Gesorteerd array met unieke gesorteerde waarden kent maar 1 verschijningsvorm:

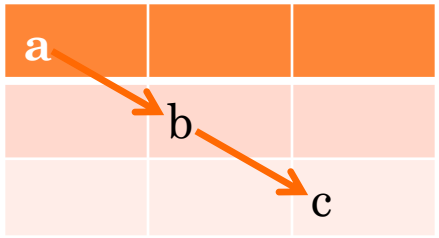
- BST met 3 gesorteerde knopen kent al 6 invoervolgordes met 5 BST vormen:



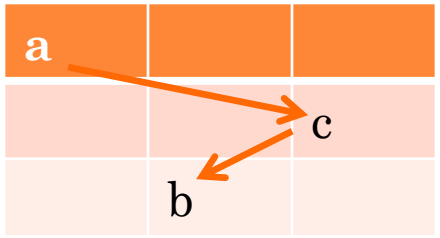




1 min pad vorm
Voor b,a,c en
Voor b,c,a
Kans 1/3



2 backbone vormen

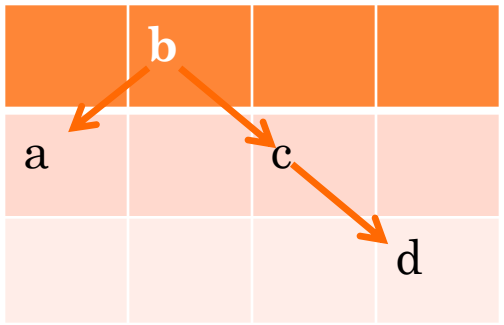


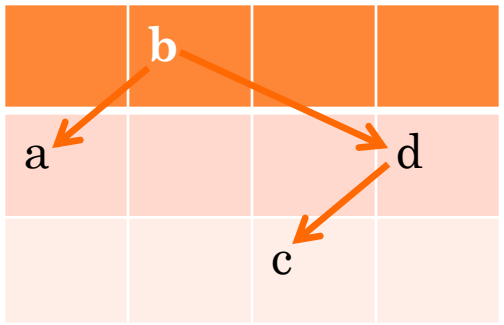
Aantal BST verschijningsvormen

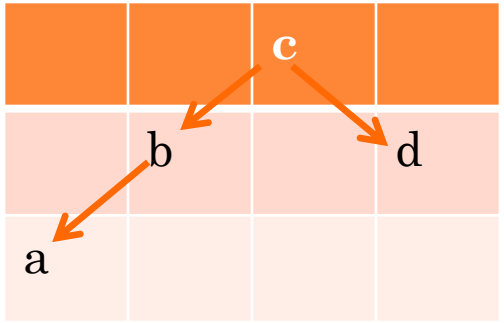
Knopen(N)	N!	BST's	Min paden
1	1	1	1
2	2	2	2
3	6	5	1
4	24	14	4
5	120	42	6
6	720	132	4

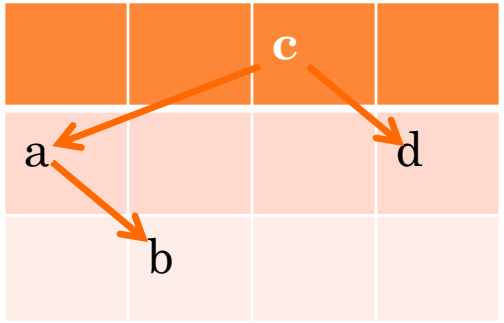
Waarom is er een groeiend verschil tussen het aantal invoegvolgordes (N!) en het aantal mogelijke BST's?

De 4 min pad
BST vormen van 4 knopen









Hoe creëer ik een min pad vorm voor mijn BST bij invoer?

- Bij toenemende N exponentiële toename aantal mogelijke BST's als waarden at random worden toegevoegd.

- Hoe te zorgen dat een min pad versie gevormd wordt en na toevoegingen en weglatingen ook blijft?
- Voor 1e inbreng zijn er methoden die dat bereiken en na wijzigingen methoden die:
  - Locaal BST evenwichtiger maken: AVL bomen
  - Globaal de BST omzetten naar een beste balans: DSW algoritme

BST OPBOUWEN VANUIT GESORTEERD ARRAY

- Een goed gebalanceerde BST kan makkelijk worden opgebouwd als alle waardes vooraf bekend zijn
- Zet de waardes in een array
- Sorteer het array
- Voeg elementen aan de BST toe door het gesorteerde array recursief met binaire 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

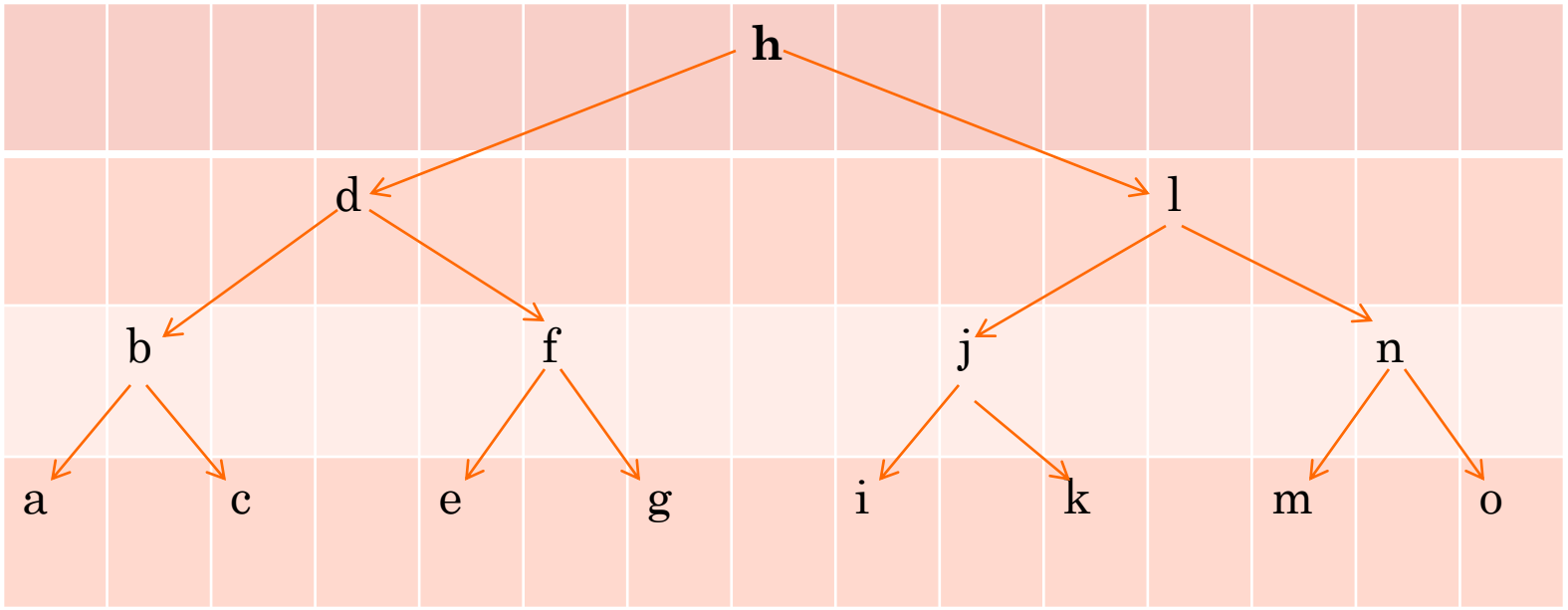
BST OPBOUWEN VANUIT GESORTEERD ARRAY-2

- Depth-first of niveau voor niveau toevoeging beide goed (maar 2 van de mogelijke gebalanceerde opbouw volgordes)

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

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

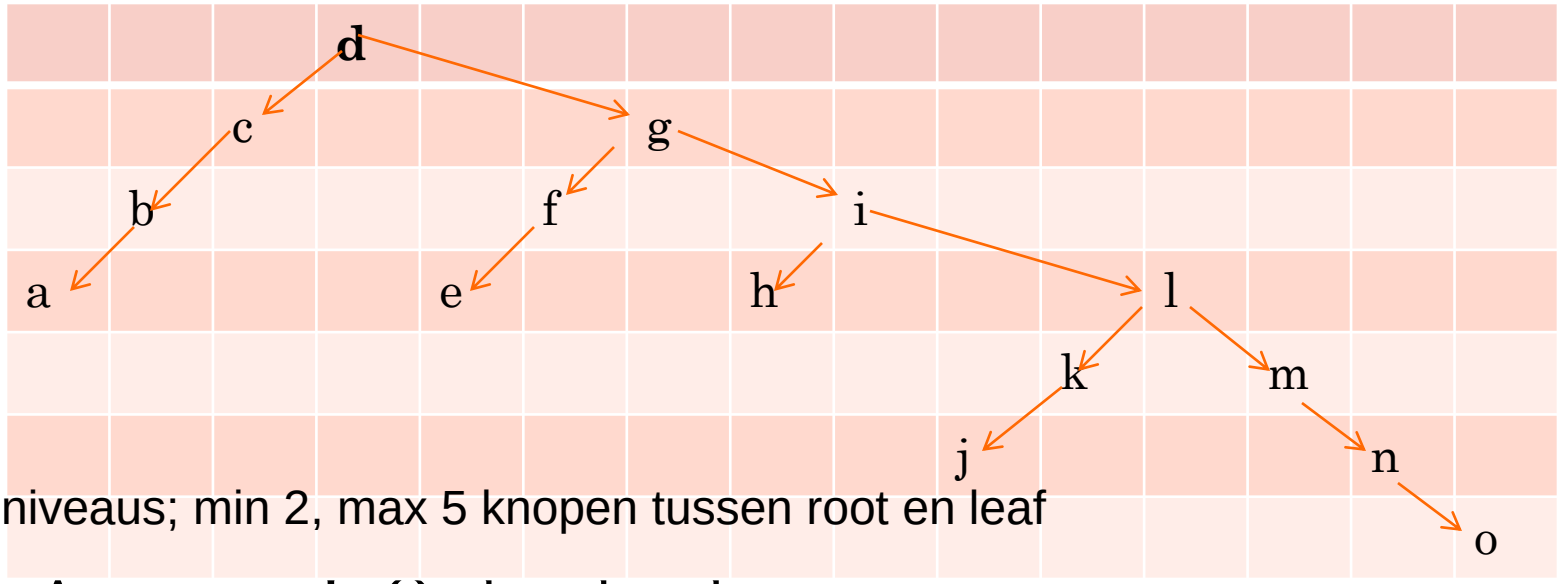
Resulterende BST



4 niveaus; max 2 knopen tussen root en leaf

Heropbouw slecht gebalanceerde boom via gesorteerd array

- Gesorteerd array kan worden opgebouwd door een bestaande BST (InOrder) te doorlopen en gepasseerde waardes in array weg te zetten.



7 niveaus; min 2, max 5 knopen tussen root en leaf

- Array na InOrder doorloop:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o

- Vanuit dit array BST gebalanceerd opbouwen:

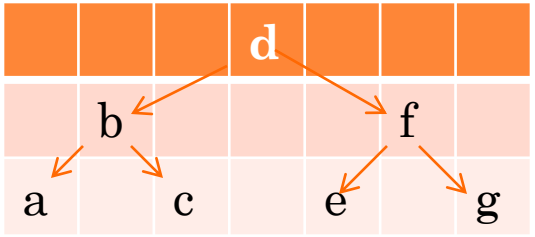
(Her)balanceren van BST

- Hoe goed is de balans binnen een boom?
- Hoe karakteriseren we een optimale balans?
- Hoe goed is de balans onder een knoop?
- Hoe kunnen we de balans lokaal verbeteren?
- Hoe wordt de balans verstoord t.g.v. insert/delete?
- Kunnen we deze aanpassingen niet automatisch in de ADT opnemen?

Hoe goed is de balans binnen een boom?

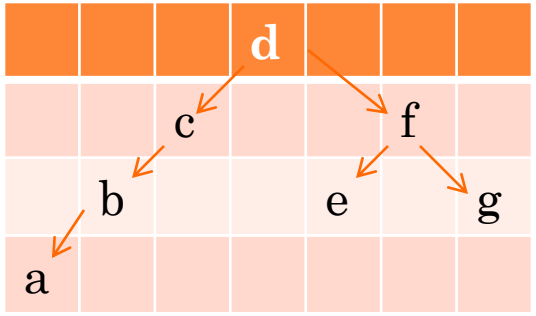
- Verschil in hoogte:
- Minimaal 0 alleen voor BST met 2^N-1 knopen
- Minimaal 1 voor BST's met ongelijk 2^N-1 knopen

Hoogte=3



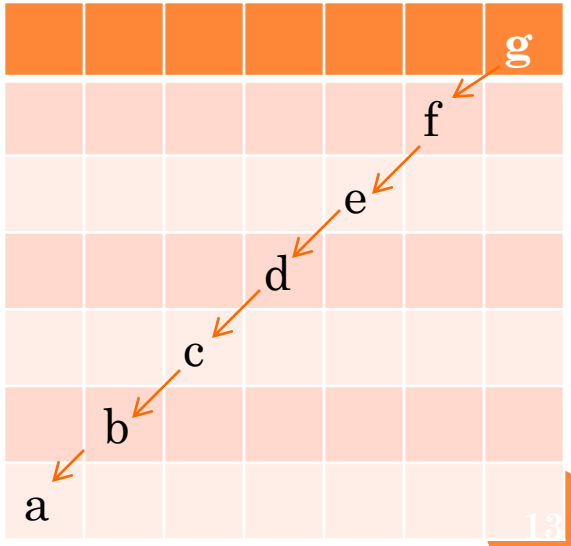
Bladeren op 1 niveau

Hoogte=4



Bladeren op 2 niveaus

Hoogte=7



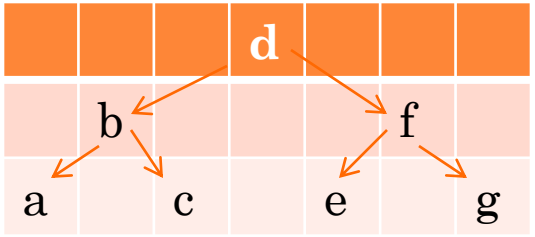
Blad op 1 niveau !

Kijken naar op hoeveel niveaus er bladeren zijn is niet genoeg!

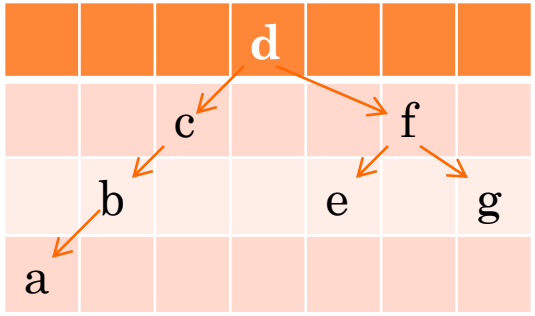
Hoe goed is de balans binnen een boom?

- Maximaal hoogteverschil voor subtrees van elke knoop:
- Minimaal 0 alleen voor BST met 2^N-1 knopen
- Minimaal 1 voor BST's met ongelijk 2^N-1 knopen
- Maximaal N-1 voor BST's met N knopen

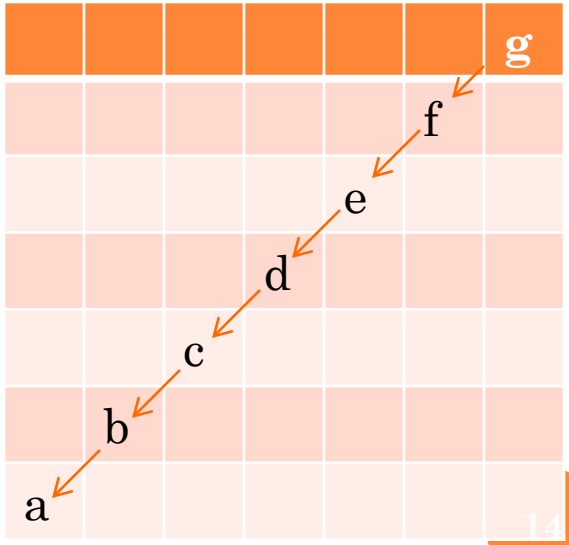
Max hoogteverschil=0



Max hoogteverschil=2



Max hoogteverschil=6

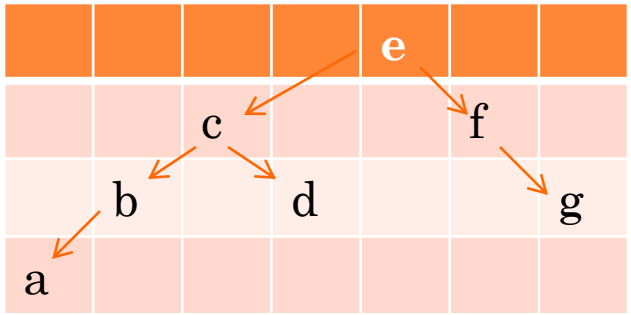


0<=Maximaal hoogteverschil<=N-1

(Hoogte) gebalanceerd (definitie)

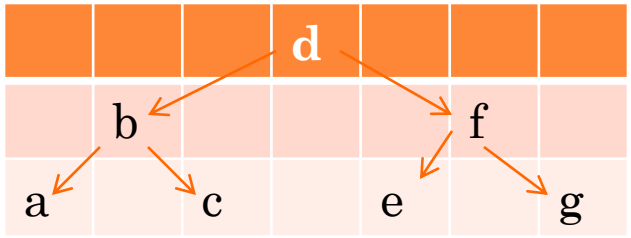
- Als: Maximaal hoogteverschil subtrees elke knoop 0/1
- Perfect gebalanceerd als tevens:
  - Alle bladeren op 1 of 2 niveaus
- Hoeft niet beste resultaat te zijn!!

Mogelijk AVL resultaat



Max hoogteverschil subtrees=1
Bladeren op 2 niveaus

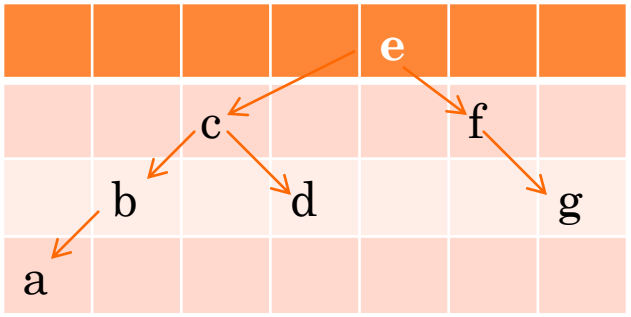
Zeker DSW resultaat



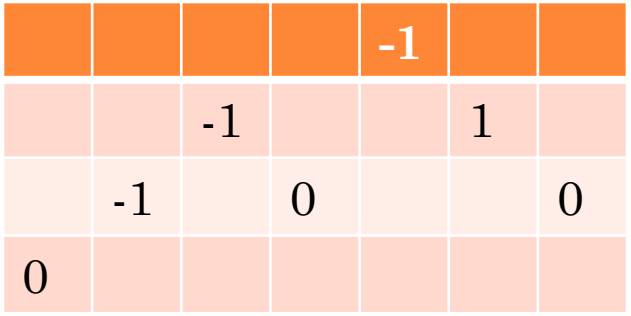
Max hoogteverschil subtrees =0
Bladeren op 1 niveau

Locale balans in AVL boom

- Balans factor in elke knoop:
- Hoogte_rechter_subtree – hoogte_linker_subtree



Balansfactoren:



Tree algoritmes voor het balanceren van BST's

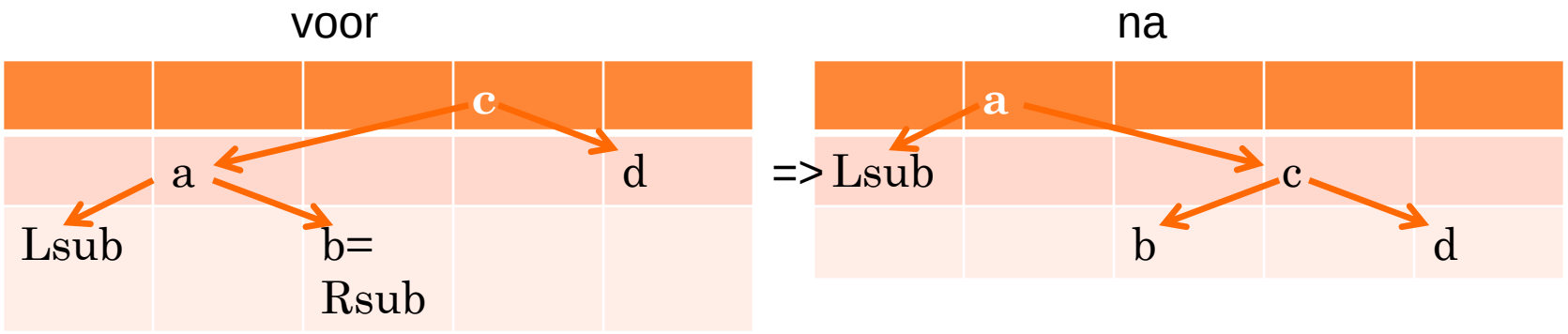
- Dure operatie om BST via array te balanceren O(N)
- Er is al vroeg gezocht naar wegen om een BST zonder tussenkomst van een array te balanceren op een O(1) of O(logN) manier
- We bekijken vandaag:
- - AVL trees
- - DSW trees

PAUZE

AVL TREES

- Adel'son, Velskii en Landis (1962)
- Eigenschap: BST waarbij maximaal verschil in hoogte van subtrees 1 is
- Hoe verminder je de hoogte in een subtree?
- Via een zogenaamde rotatie
- Een rotatie is een lokale wijziging van de BST waarbij de (InOrder) volgorde behouden blijft maar lokaal de hoogte 1 vermindert (L/R) en 1 vermeerdert (R/L); het verandert dus de balancering tussen de L en R subtrees onder een bepaalde knoop in de boom

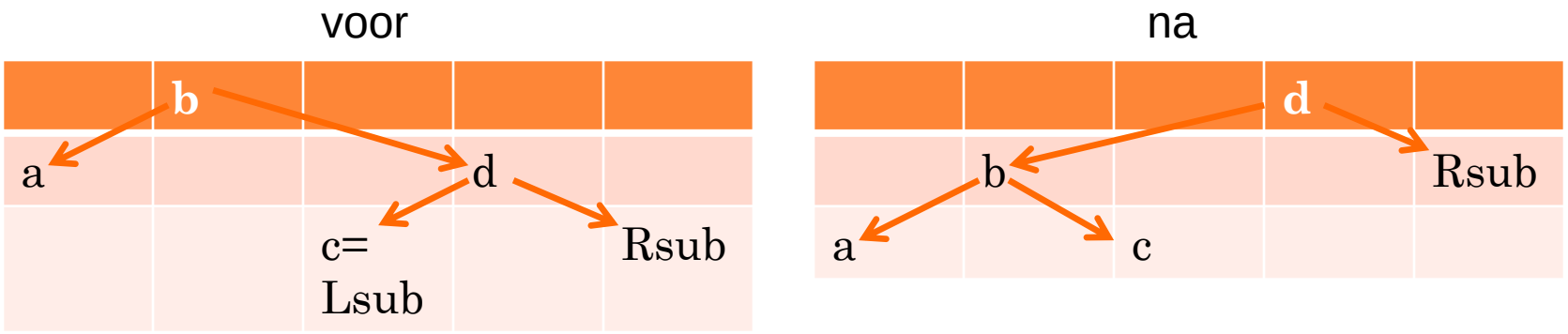
Rechts Rotatie



a en de linker subtree van a komt 1 niveau hoger;
rechter subtree van a blijft op gelijke hoogte als linker subtree van c;
c en rechter subtree c komen 1 niveau lager terecht

InOrder BST volgorde behouden

Links Rotatie

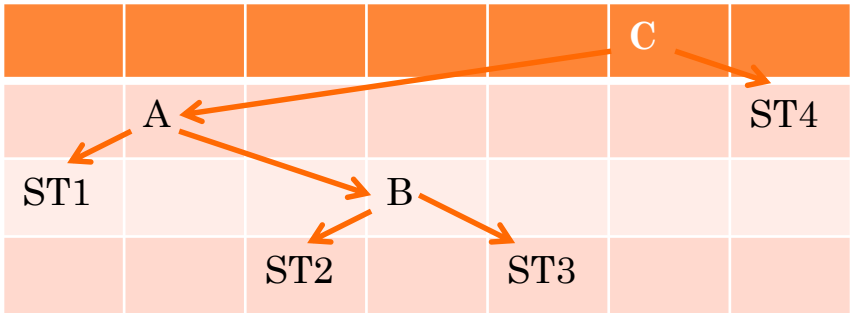


d en de rechter subtree van d komen 1 niveau hoger;
linker subtree van d blijft op gelijke hoogte als rechter subtree van b;
b en linker subtree a komen 1 niveau lager terecht

InOrder BST volgorde behouden

Welke rotaties nodig?

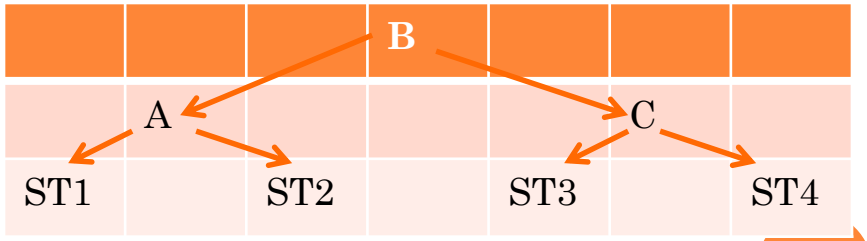
start



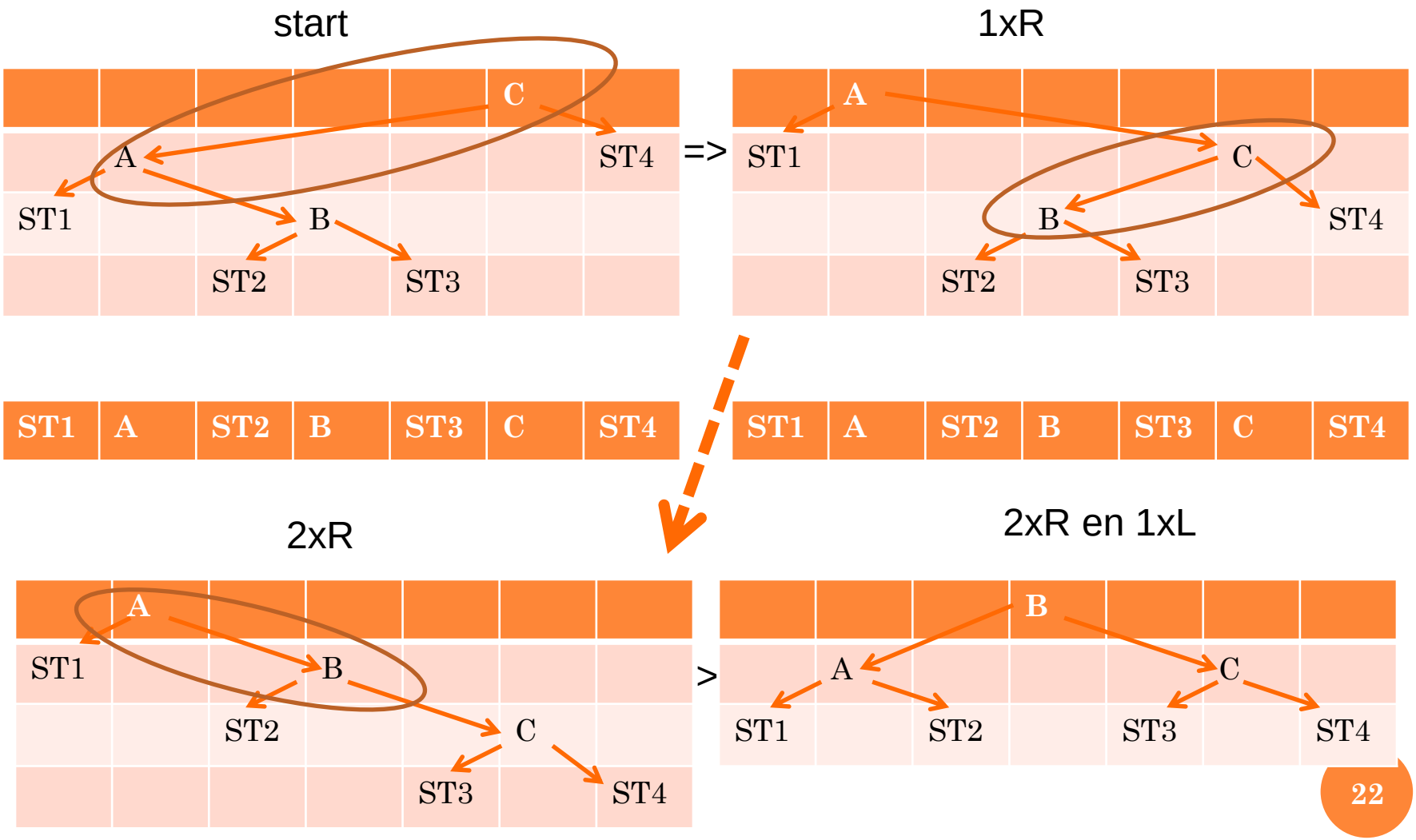
ST1	A	ST2	B	ST3	C	ST4

ST1	A	ST2	B	ST3	C	ST4

2xR en 1xL



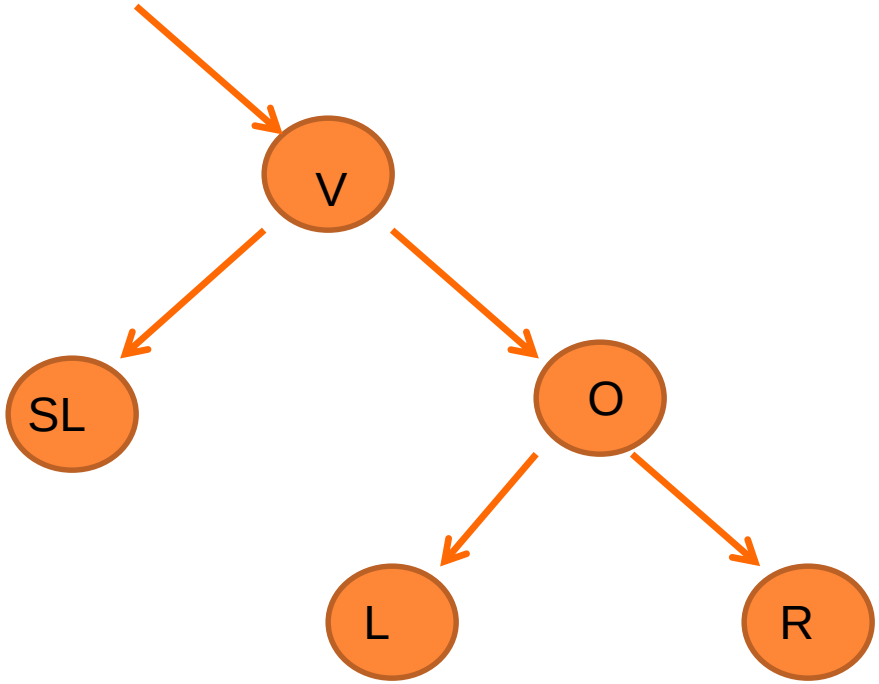
Dubbele Rechts Rotatie en 1 x Links



DSW BALANCERING VAN EEN BST

- Day (1976) + Stout & Warren (1986)
- Maximaal gebalanceerd via louter rotaties
- Stap 1: roteren naar backbone structuur via rechts rotaties (tot alle linker subtrees leeg) in reverse Inorder knoop volgorde (ROL)
- Stap 2: roteren naar maximaal gebalanceerd m.b.v. linksrotaties beginnend op het laagste nivo (M=(Nback-1)/2) steeds 2 opvolgende in backbone; herhalen met M=(Nback-1)-M zolang M>0

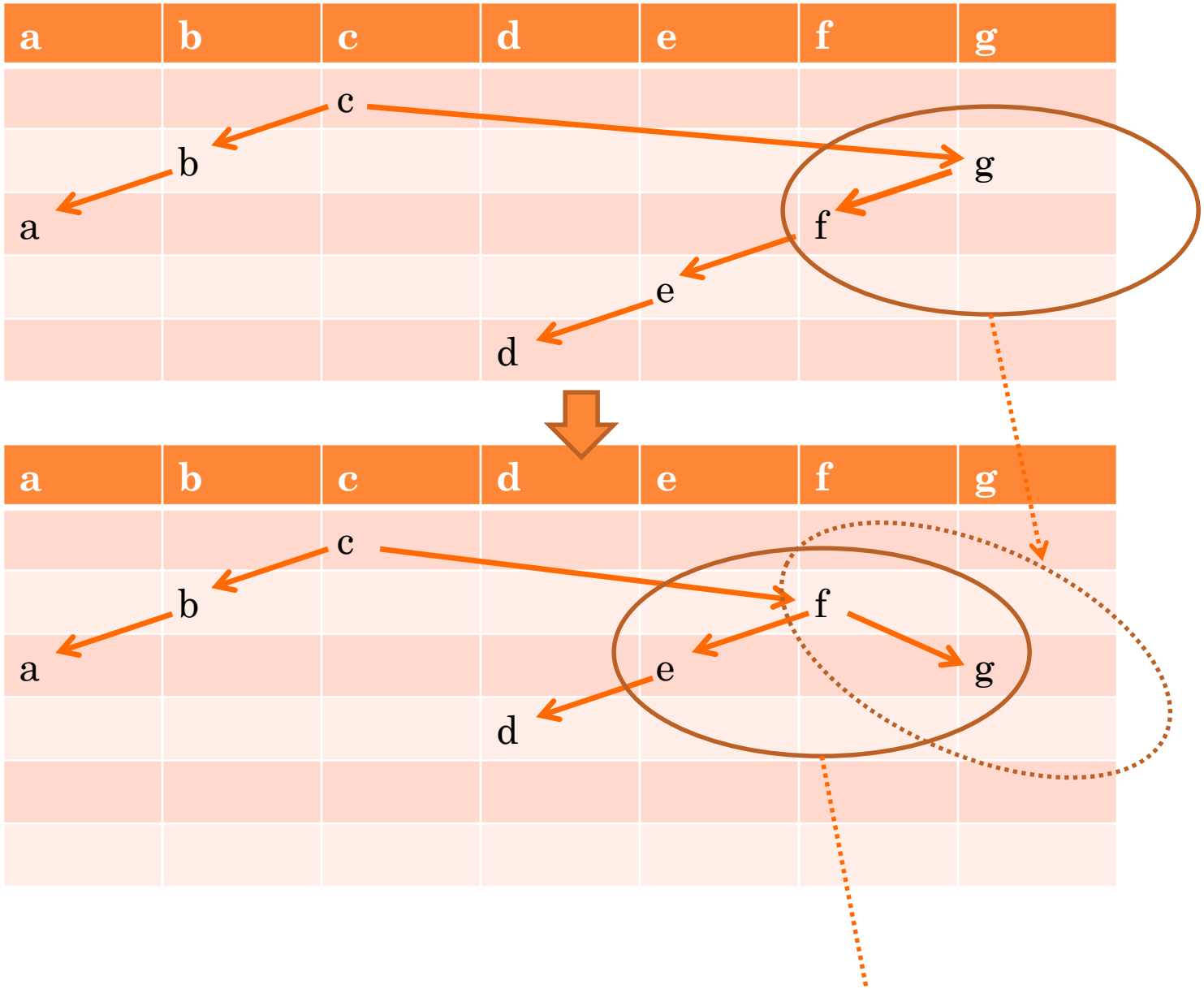
Reverse inOrder pad in BST



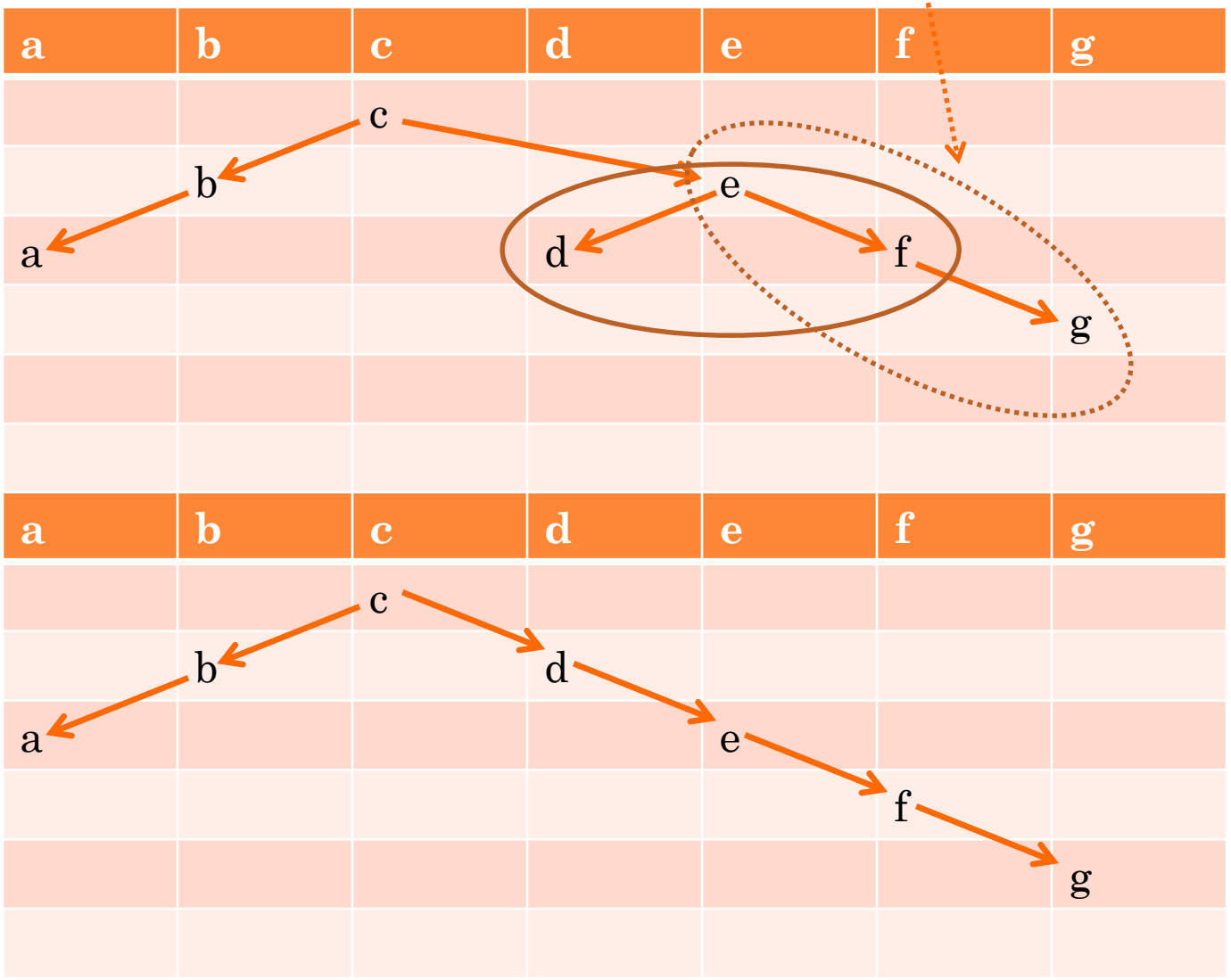
InOrder:…SL,V,L,O,R

Reverse InOrder: R,O,L,V,SL,….

DSW VOORBEELD OP 7 KNOPEN BST



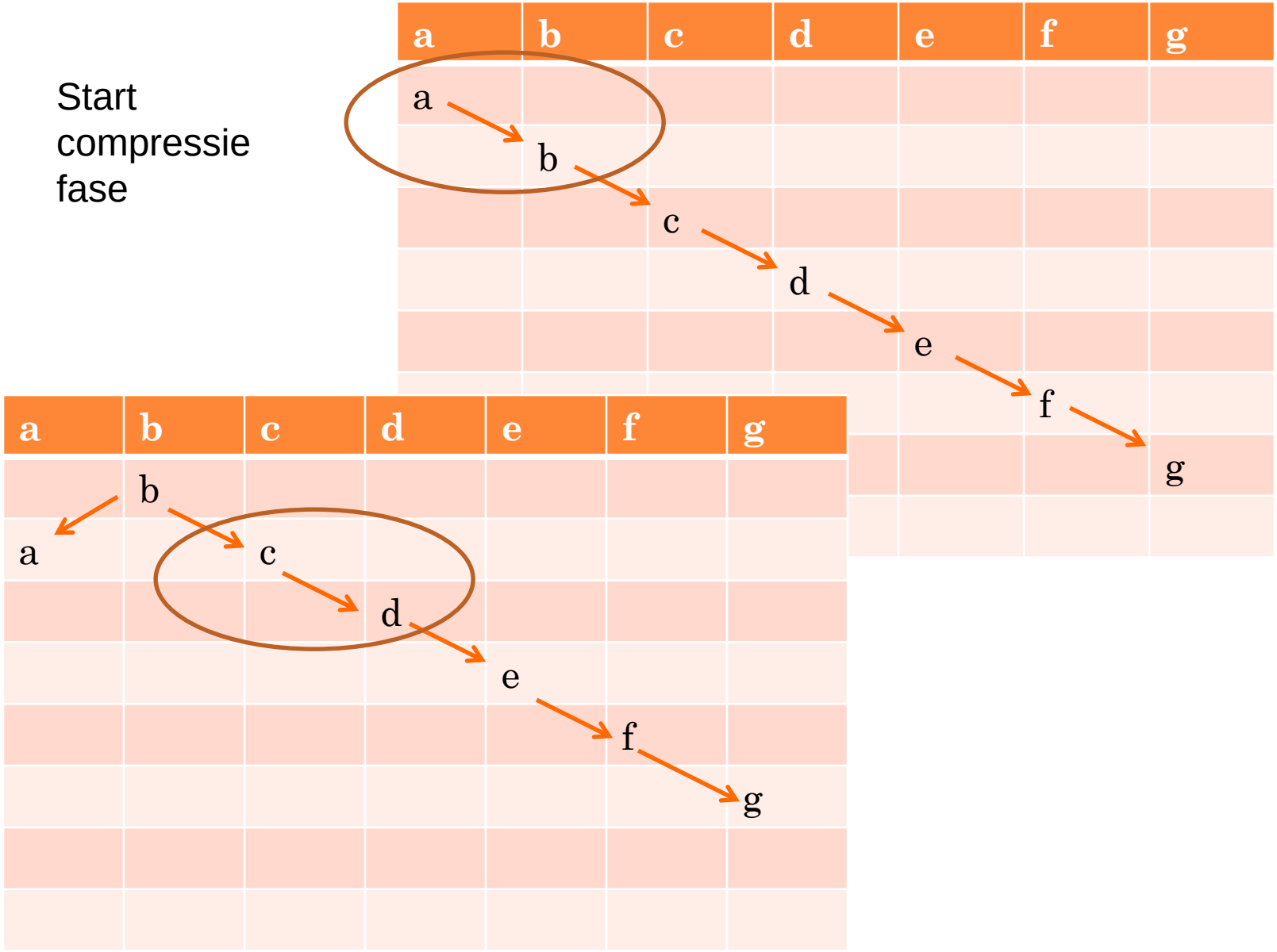
DSW VOORBEELD VERVOLG-1



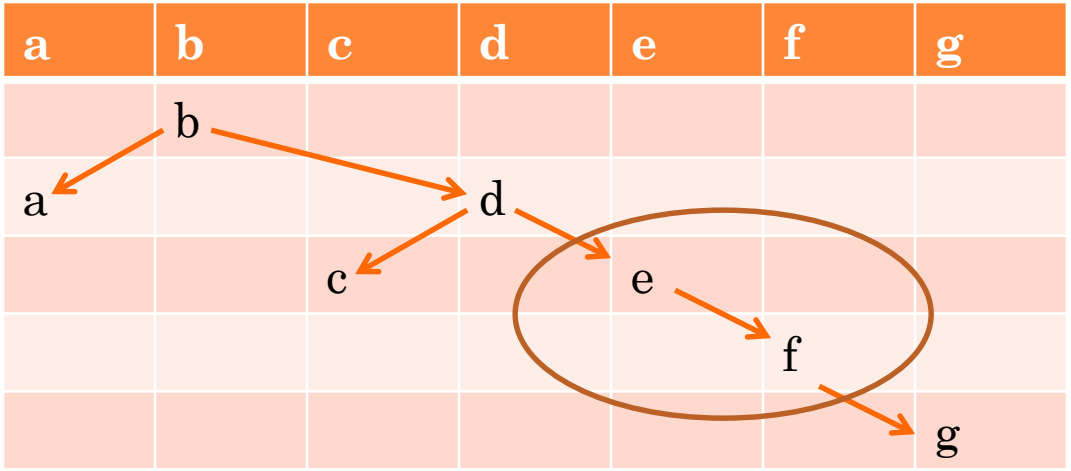
Doorgaan tot linker subtree voorste ook leeg -> backbone klaar

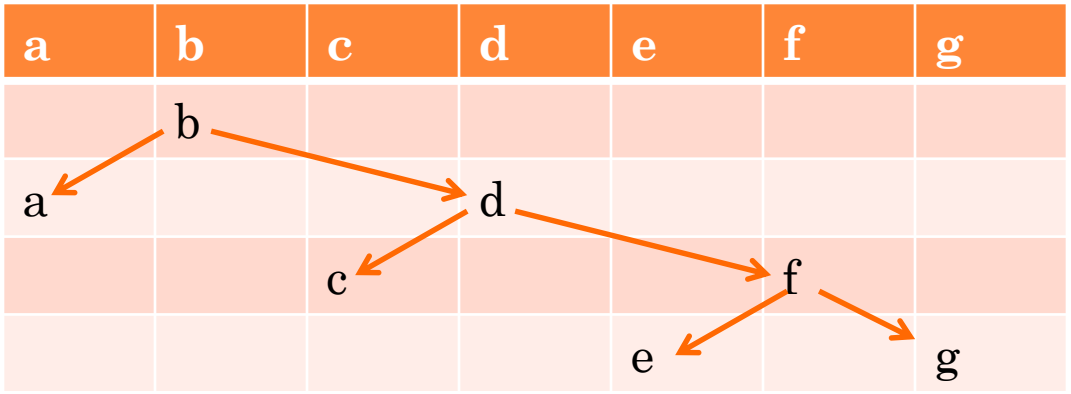
DSW VOORBEELD BACKBONE -> BALANCED

Start compressie fase



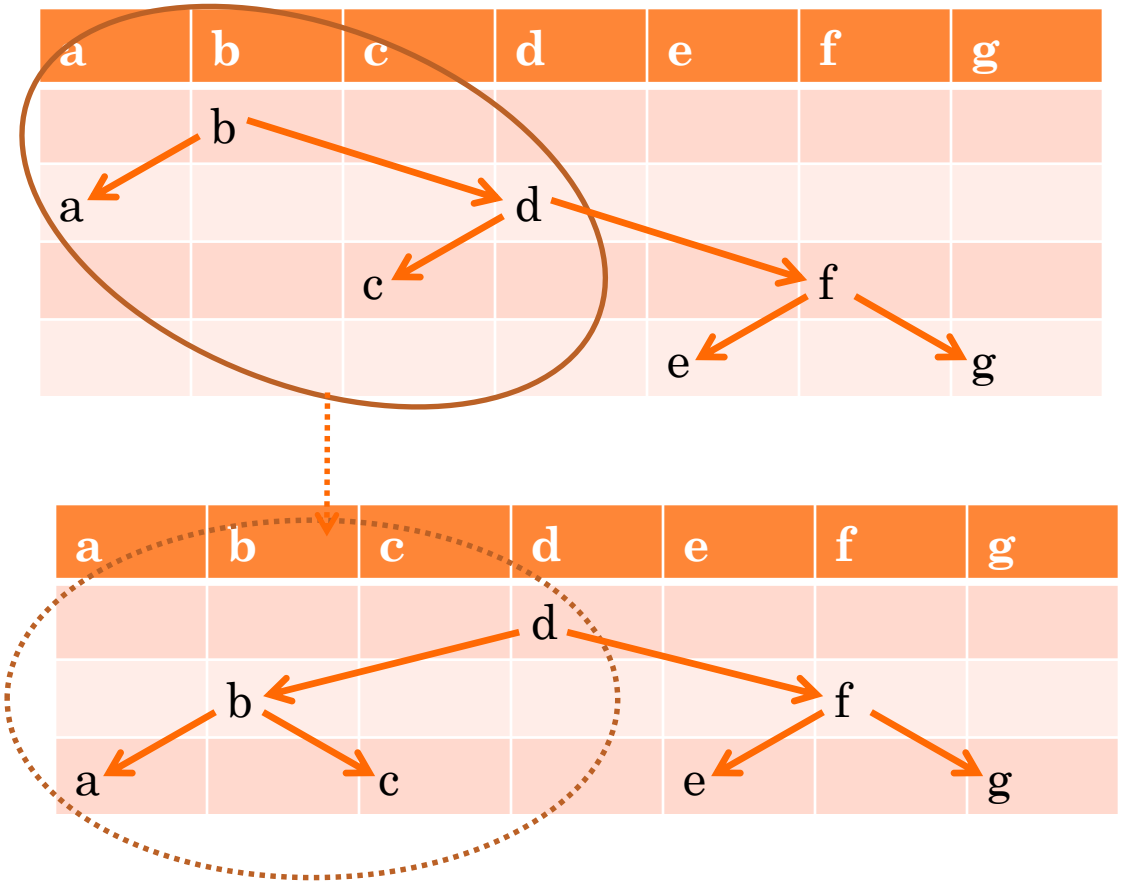
DSW VOORBEELD BACKBONE -> BALANCED 2





Hiermee laagste nivo afgewerkt

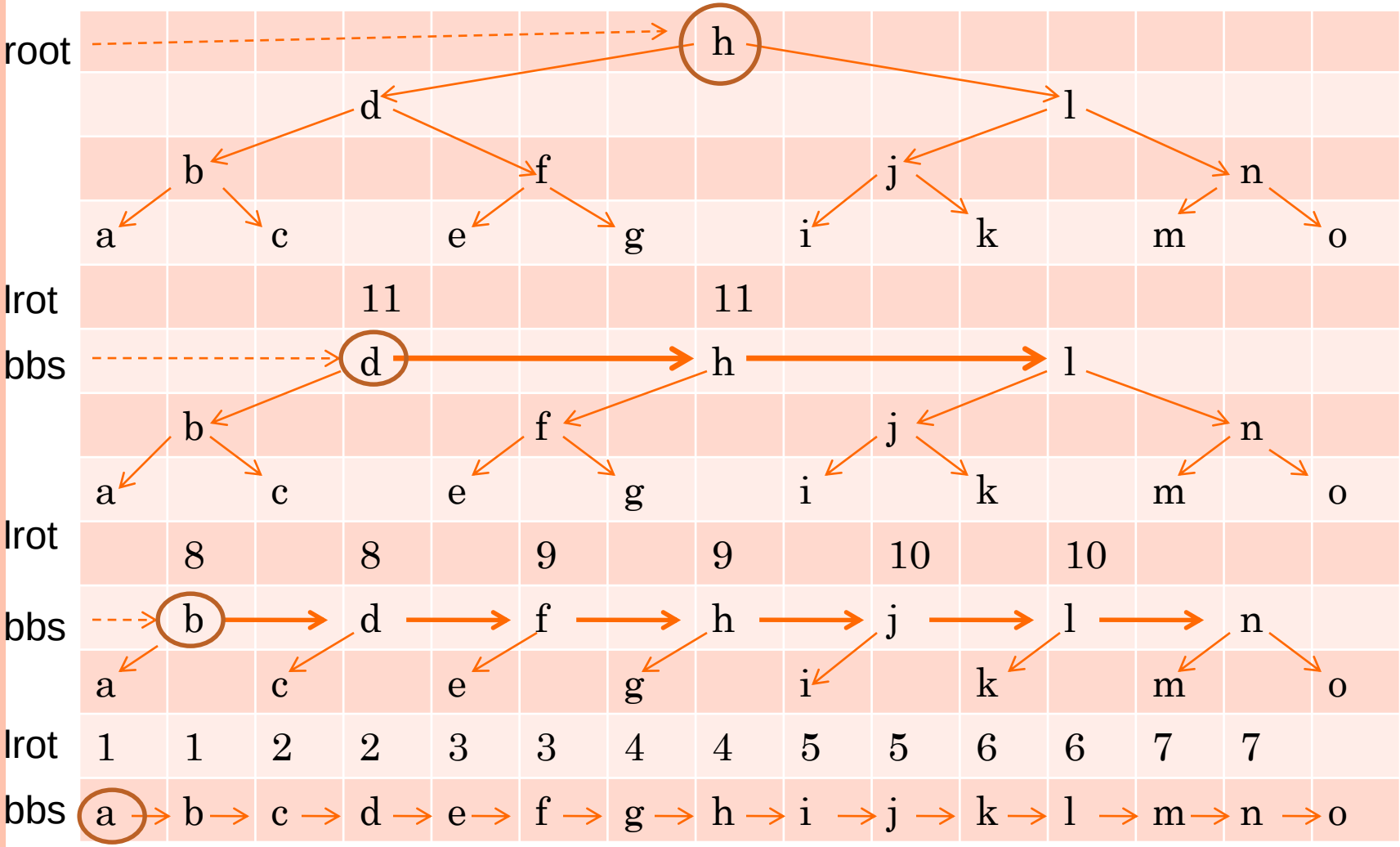
DSW VOORBEELD BACKBONE -> BALANCED 3



Hoogste nivo laatste linksrotatie -> klaar!

DSW BALANCERING GEZIEN VANUIT BACKBONESTRUCTUUR

Volgorde en betrokkenheid bij linksrotaties genummerd



Alternatief gebruik BST

- Als je een BST die na herhaald invoegen van nieuwe elementen en deleten van oude elementen is ontstaan zonder balanceren efficient voor veel opzoekwerk wil gebruiken,
- Wat zou je dan kunnen doen i.p.v. balanceren?

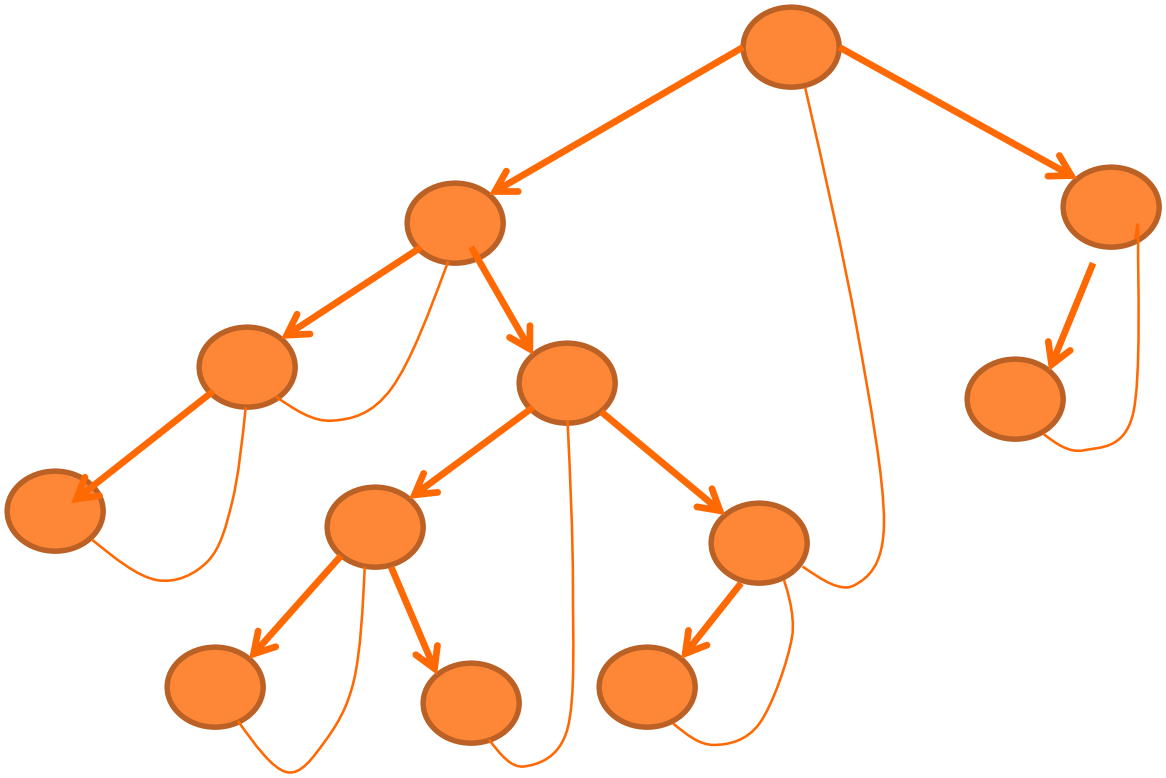
Alternatief gebruik BST

- Als je een BST die na herhaald invoegen van nieuwe elementen en deleten van oude elementen is ontstaan zonder balanceren efficient voor veel opzoekwerk wil gebruiken,
- Wat zou je dan kunnen doen i.p.v. balanceren?
- Je zou deze mogelijk slecht gebalanceerde BST Inorder uit kunnen lezen naar een array (die de gesorteerde rij maar op 1 manier zal representeren) en daarop de binary searches kunnen uitvoeren
- Ook kan vanuit dit array een goed gebalanceerde BST kunnen worden opgebouwd ter vervanging van de uit balans geraakte

BST: Threaded vorm Versnellen inOrder aflopen zoekpad

- Variant van BST
- Oponthoud door nil-pointers tijdens doorloop
- Elke leaf heeft al 2 nil pointers
- Thread=zet adres InOrder opvolger in boom op plaats nil-pointer:
- Nil in linker child -> thread naar ouder knoop
- Nil in rechter child -> thread naar voorouder knoop waarvan huidige knoop verst in linker subtree zit
- Wat voor een adres in child pointer staat (pointer naar kind/1e niet lege InOrder opvolger) kan m.b.v. een extra bit onderscheiden worden

Voorbeeld threaded BST



Is Binary Search snelste zoekmethode?

- Hoe zoek je zelf in b.v. een telefoongids?

Is Binary Search snelste zoekmethode?

- Vergelijk met zoeken in telefoongids
- Rekening houden met de verdeling over de plaatsnamen en het alfabet kies je een ingang meer voorin of meer achterin i.p.v. het midden
- Als je kennis hebt van de verdeling van de gesorteerde waardes kun je een betere gok doen dan steeds het midden
- De eerste stappen kun je na een aantal malen zoeken omzeilen m.b.v. een ingangstabel b.v. index waarbij de K of postcode 5000 begint
- Je kunt zo makkelijk nog enkele (begin)stappen uitsparen

Opgeslagen waarde en adres van deze waarde

- Als onbekend is onder welk adres een waarde is opgeslagen helpt kennis dat de waardes gesorteerd zijn opgeslagen (binary search opzoek strategie mogelijk)
- Impliciet is er dan al iets van de waarde in het adres (index/key) doorgekomen
- Wat kun je doen als de op te bergen/op te roepen waarde expliciet samenhangt met het adres?

O(1) Zoekmethoden in data?

- Om in 1 keer een bepaalde key in data te vinden moet je data opslag structuren opzetten die een **intern adres** krijgen **dat direct met de op te zoeken key waarde samenhangt**.

- Deze zogenaamde hash technieken komen later in dit college aan de orde
- Waardes hoeven dan ook niet gesorteerd te zijn

Drozdek

- H.6: 6.5, 6.6, 6.7 en 6.8