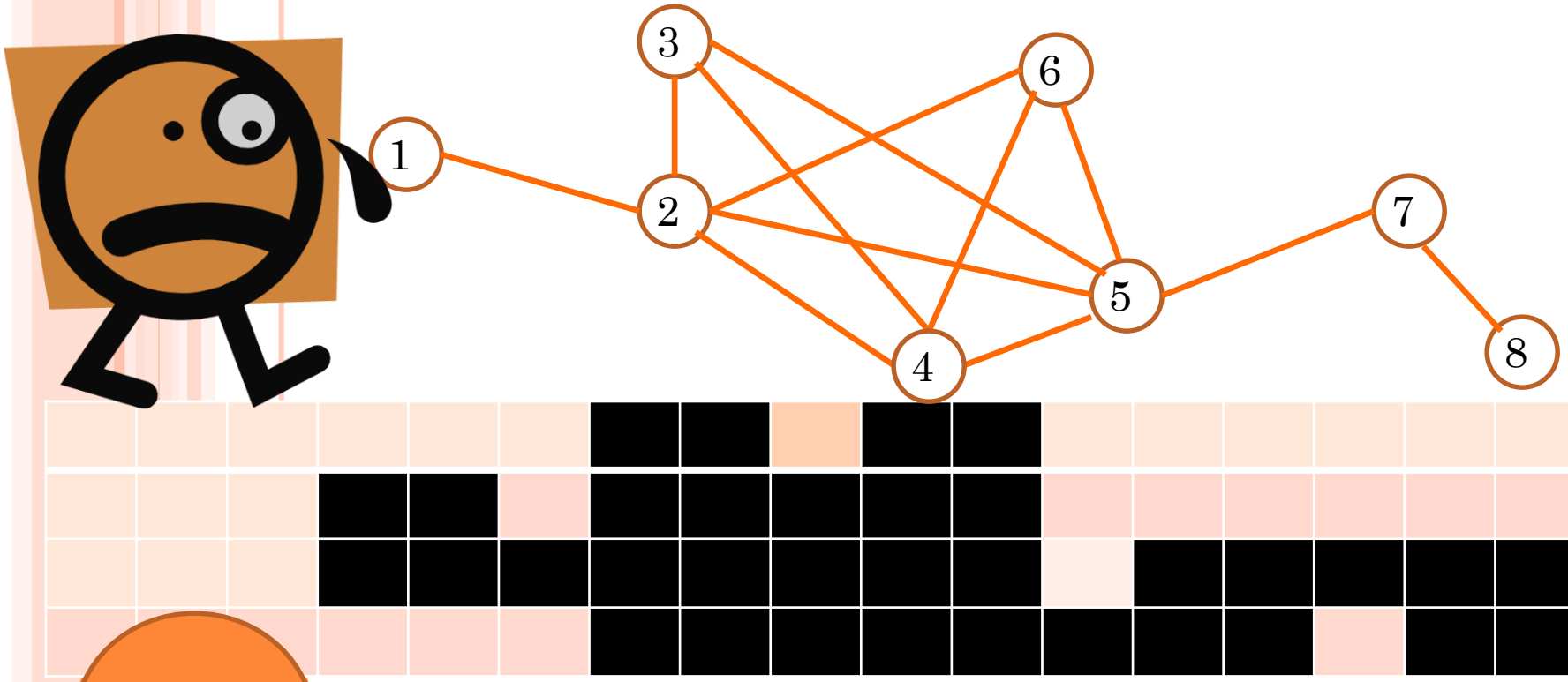


Interval grafen en bomen

datastructuren

Dr. D.P. Huijsmans

College 11 13 november 2013

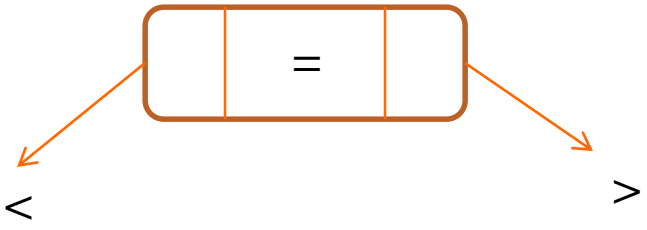
Universiteit Leiden LIACS

Zoeken en vergelijken van tijdsintervallen

- Bij het onderling vergelijken van intervallen komt meer kijken dan vergelijken van puntwaarden
- Waarin zit het verschil?
- Mogelijke relaties tussen 2 tijdspunten A en B:

  A<B; A=B;A>B (< voordat; = gelijktijdig; > nadat)

- Deze 3 mogelijke relaties passen goed bij een BST waar de knoopwaarde en twee kindrichtingen kleiner dan en groter dan inpassen



Het aantal mogelijke tijdsinterval relaties

- Gegeven twee gesloten intervallen [a,b] en [c,d] met a,b,c,d ε INT met restricties a<=b en c<=d
- Hiermee zijn van de in totaal 24 permutaties (4!) de 6 volgende nog maar mogelijk (elk met 5 tot 7 situaties naargelang tijdstippen onderling gelijk zijn of niet):
- abcd: [a..b] [c..d]; [a..b=c..d]; [a=b=c..d]; [a=b=c=d];
- [a=b] [c..d]; [a..b] [c=d]; [a..b=c=d]
- acbd: [a..[c..b]..d]; [a=c..b]..d]; [a..[c..b=d]; [a=c..b=d]; [a=c=b=d]
- acdb: [a..[c..d]..b]; [a=c..d]..b]; [a..[c..d=b]; [a=c..d=b]; [a=c=d=b]
- cabd: [c..[a..b]..d]; [c=a..b]..d]; [c..[a..b=d]; [c=a..b=d]; [c=a=b=d]
- cadb: [c..[a..d]..b]; [c=a..d]..b]; [c..[a..d=b]; [c=a..d=b]; [c=a=d=b]
- cdab: [c..d] [a..b]; [c..d=a..b]; [c=d=a..b];[c=d=a=b];
- [c=d] [a..b]; [c..d] [a=b]; [c..d=a=b]

Hoe zouden we deze 34 configuraties beschrijven?

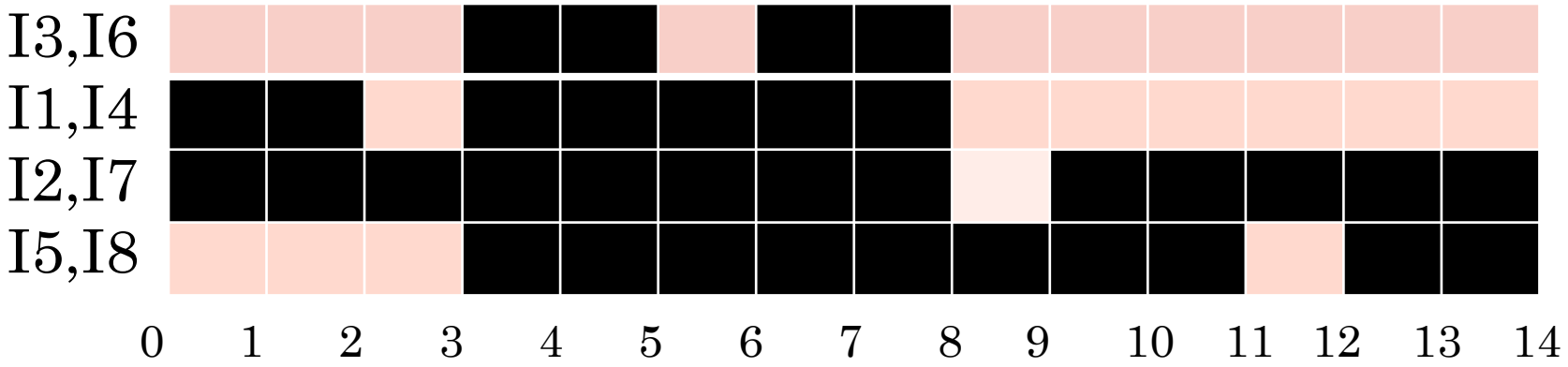
De topologisch verschillende tijdsinterval configuraties

- Interval1 [a..b] relatie met interval2 [c..d]:
  - Interval1 voor interval2
  - Interval1 leidt interval2 (raakt aan voorkant)
  - Interval1 stopt binnen interval2
  - Interval1 omsluit interval2
  - Interval1 gelijk aan interval2 (+gelijke tijdstippen)
  - Interval1 is binnen interval2
  - Interval1 start binnen interval2
  - Interval1 volgt interval2 (raakt aan achterzijde)
  - Interval1 na interval2
- Het is duidelijk dat deze 9-deling
- (10-deling zelfs als we identiek interval en identieke tijdstippen willen onderscheiden)
- niet past op een BST structuur

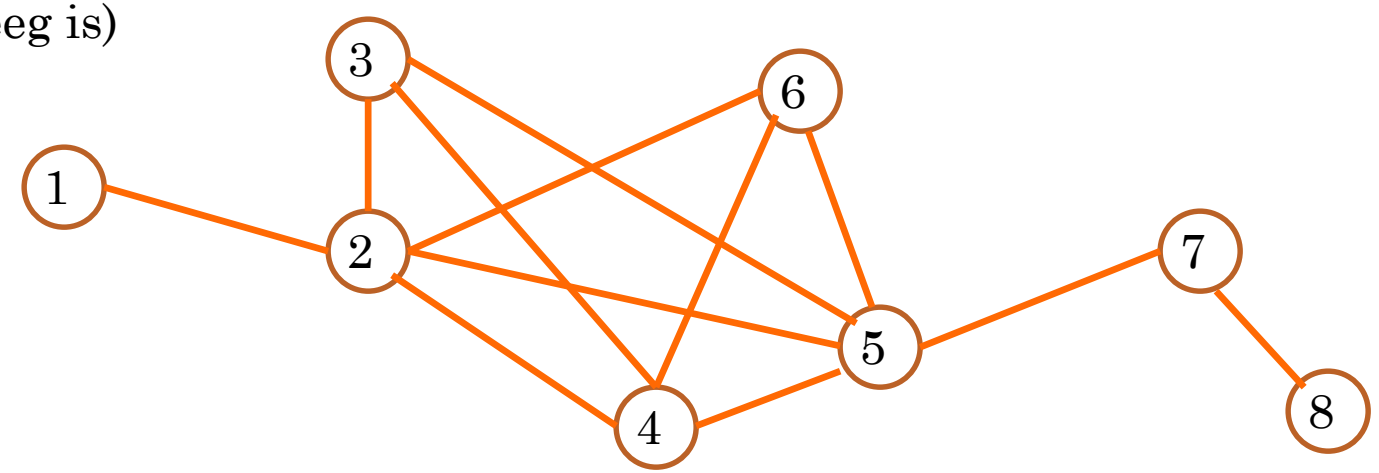
Voorbeeld voor interval query

- Laten we aan de hand van een voorbeeld kijken wat we met intervallen en apart de begin- en eindpunten van intervallen kunnen doen.
- Voorbeeld:
- I1=[0,2]; I2=[0,8]; I3=[3,5]; I4=[3,8]; I5=[3,11]; I6=[6,8]; I7=[9,14]; I8=[12,14]
- Gevraagd wordt met welke intervallen I9=[4,7] een niet lege relatie heeft

Voorbeeld interval situatie



Hierbij past de volgende Interval Graaf
(ongerichte graaf waarvan de zijdes een 1-op-1 verband hebben met een verzameling lineair geordende intervallen, zodanig dat 2 knopen alleen dan een zijde delen als de doorsnede van hun intervallen niet leeg is)



Matrix formulering interval graaf

- Bij de Interval Graaf past een matrix met 1-en en 0-en voor de wel/niet aanwezigheid van zijden (interval doorsnijdingen)
- De matrix van een Interval Graaf heeft als kenmerk dat z'n rijen zo gepermuteeerd kunnen worden dat de kolommen de eigenschap opeenvolgende 1-en hebben

0	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

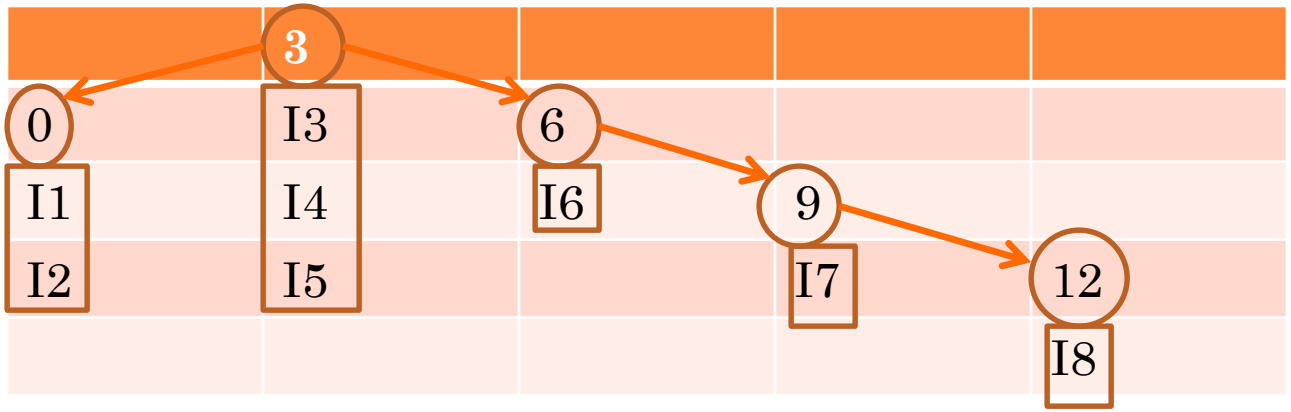
0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Nut Interval Graaf bij opzoeken

- Als de intervallen in een array staan, kost het aflopen en testen tegen een zoekinterval of er een niet-lege doorsnede is O(N) tijd
- Met behulp van de interval graaf waarin staat voor elke i,j interval relatie of deze wel/niet leeg is, kost het opzoeken nog wel O(N) tijd, maar het resultaat van wel/niet een doorsnede is direct bekend en kost maar een bit opslag per kruispunt.

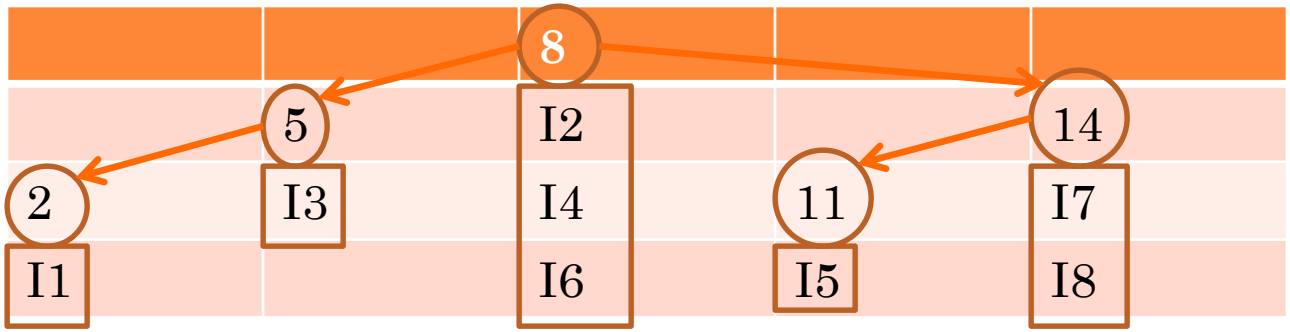
Sneller zoeken in Intervallen m.b.v. twee Interval BST's

- Voor de gesorteerde beginpunten van de intervallen zetten we een Interval BST op met per knoop een lijstje van interval-id's:
- Voor ons voorbeeld:
- Gesorteerd Beginpunt: 0, 0, 3, 3, 3, 6, 9,12
- Interval: I1,I2,I3,I4,I5,I6,I7,I8
- Vul Interval-L BST m.b.v. binary search:



Interval-R BST voor de eindpunten

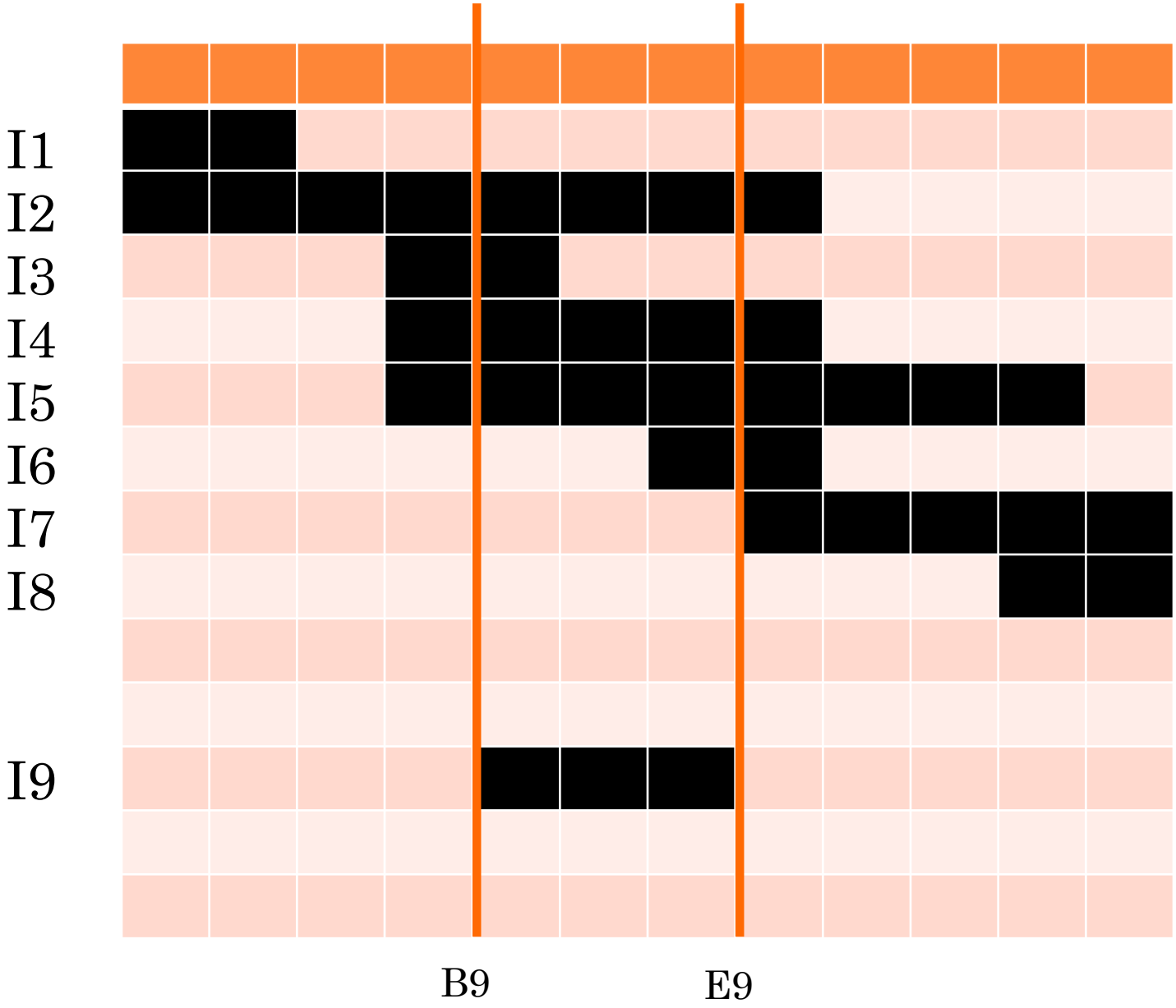
- Gesorteerd op eindpunten wordt voor ons voorbeeld de Interval-R BST invoer:
- Gesorteerd Eindpunt: 2, 5, 8, 8, 8, 11, 14, 14
- Interval-ID:I1, I3, I2, I4, I6, I5, I7, I8



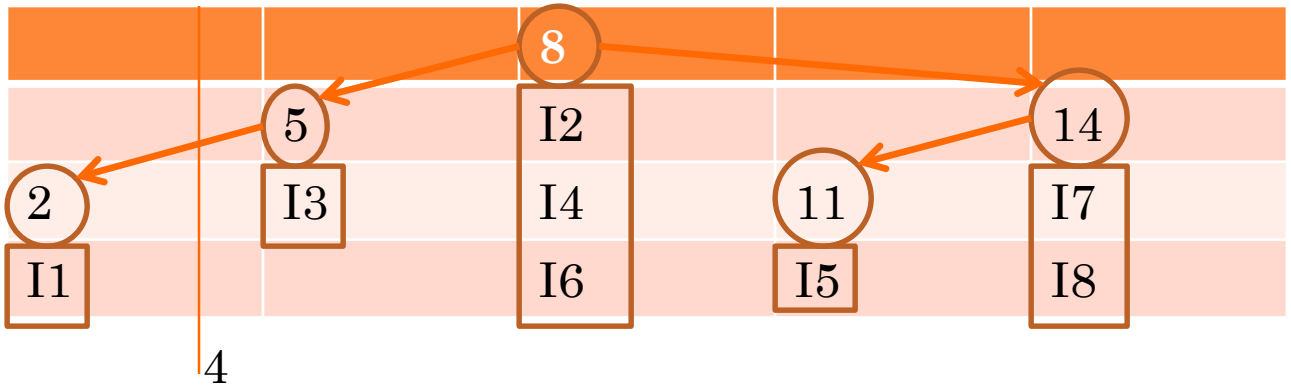
Zoekalgoritme met Interval-LR BST's

- Het zoeken naar intervallen die bekeken moeten worden om een doorsnede te bepalen gaat als volgt:
- 1) **zoek met startpunt in eindpunten boom** (Interval-R BST); alleen intervallen die eindpunten op of vanaf het **start punt** hebben komen nog in aanmerking
- 2) **zoek met eindpunt in beginpunten boom** (Interval_L BST); alleen die intervallen die beginpunten voor of tot aan het **eind punt** hebben komen nog in aanmerking
- Neem de doorsnede van de 2 interval lijsten en doorzoek deze per interval op relatie.

Overdekking bij ons voorbeeld

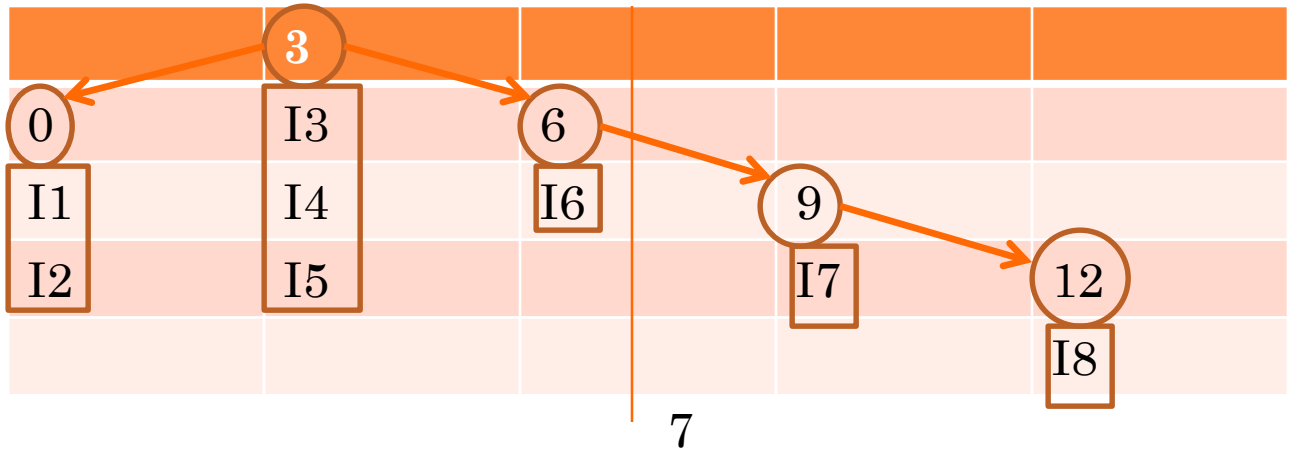


Voorbeeld: zoeken naar intervallen die qua eindpunt bij een gegeven beginpunt nog doorsnij kansen hebben



- Het enige eindpunt dat voor beginpunt 4 in de eindpunten boom zit is interval I1 dus I1 komt voor I9 en kan I9 niet doorsnijden (I1 is al afgelopen voordat I9 begint)
- Alle intervallen die eindigen nadat I9 begonnen is te weten I2 t/m I8 komen mogelijk nog wel in aanmerking

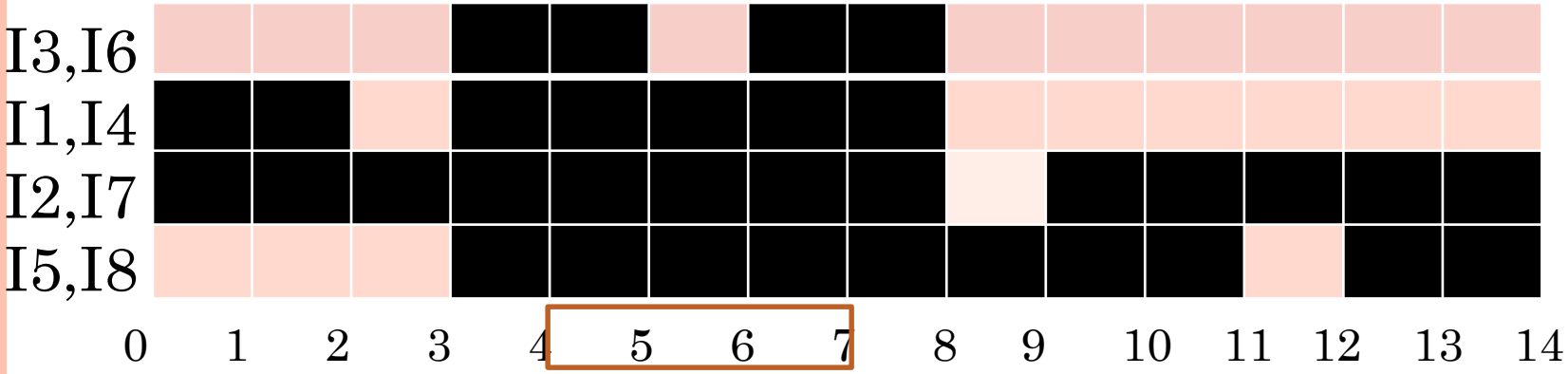
Voorbeeld: zoeken naar intervallen die qua beginpunt bij een gegeven eindpunt nog doorsnij kansen hebben



- De beginpunten die na eindpunt 7 in de beginpuntenboom zitten hebben geen doorsnijdingsmogelijkheid met I9;
- I9 komt voor I7 en I8;
- Intervallen I1 t/m I6 komen nog wel in aanmerking

Voorbeeld: eindselectie interval doorzoekverzameling voor I9[4,7]

- Kandidaten eindpunten boom:
- I2,I3,I4,I5,I6,I7,I8 (I1[0,2] en I9: I9 na I1)
- Kandidaten beginpuntenboom:
- I1,I2,I3,I4,I5,I6 (I7[9,14] en I8 [12,14] en I9: I9 voor I7,I8)
- Te doorzoeken doorsnede van Intervallen:
- I2,I3,I4,I5,I6
- Resultaat vergelijking relatie met I9 [4,7]:
- I2 [0,8] en I9: [4,7] I9 binnen I2
- I3 [3,5] en I9: [4,5] I9 start binnen I3
- I4 [3,8] en I9: [4,7] I9 binnen I4
- I5 [3,11] en I9: [4,7] I9 binnen I5
- I6 [6,8] en I9: [6,7] I9 stopt binnen I6



Voordeel Interval-LR BST's bij niet doorsnijdende intervallen

- Uit het voorbeeeld was al te zien dat als het gaat om snel te bepalen welke intervallen geen doorsnede hebben met een test interval het:
- Direct volgt uit alleen de Interval-L BST zoektocht welke Intervallen in z'n geheel voor het test-interval liggen.
- Direct volgt uit alleen de Interval-R BST zoektocht welke intervallen in z'n geheel na het test-interval liggen.
- Als de kans op een doorsnijding klein is, hoeft er dus vaak maar in 1 van de 2 Interval BST's gezocht te worden!

Interval zoeken in k-D

- De aanpak voor interval zoeken in 1D middels 2 BST's voor de begin- en eindpunten kan naar believen uitgebreid worden naar meer dimensies;
- Per dimensie kan eenzelfde Interval-LR BST paar gebruikt worden
- Per dimensie worden de genummerd kD-intervallen middels hun component in die dimensie geselecteerd
- Er is alleen een extra merge stap nodig na het ophalen van de kandidaten uit alle kD-dimensies apart
- Niet-doorsnijdende selekties kunnen weer sneller per dimensie middels 1 van de 2 Interval-LR BST's

Alternatief voor R-tree

- De Interval-LR BST's zijn hiermee een bruikbaar alternatief voor de al eerder besproken R-tree omdat voor de R-tree rechthoekzijden langs de assen gebruikt worden als MER (Minimum Enclosing Rectangle) en dit in 2D toch al een aparte verzameling intervallen in x-richting en in de y-richting betekent.

- Conclusie: Voor 2D, 3D en 3D,T modelleringen kan de Interval-LR BST's aanpak makkelijk ingezet worden

Efficientie BST aanpak intervallen

- Vergeleken met een Interval Graaf of aflopen van een array met intervallen, die beide een O(N) complexiteit hebben,
- Heeft BST aanpak met 2 bomen voor begin- en eindpunt een complexiteit van 2logN+R, waarbij R het aantal overlappende intervallen is
- Alleen als R<<N loont dit als je op zoek bent naar overlappende intervallen, voor intervallen geheel voor of geheel na een query interval (of tijdstip) is de complexiteit beter n.l. O(logN) omdat dan maar naar 1 van de twee Interval-BST's gekeken hoeft te worden.

Open- of gesloten interval

- Aantal configuraties nog veel groter
- Oplossing?

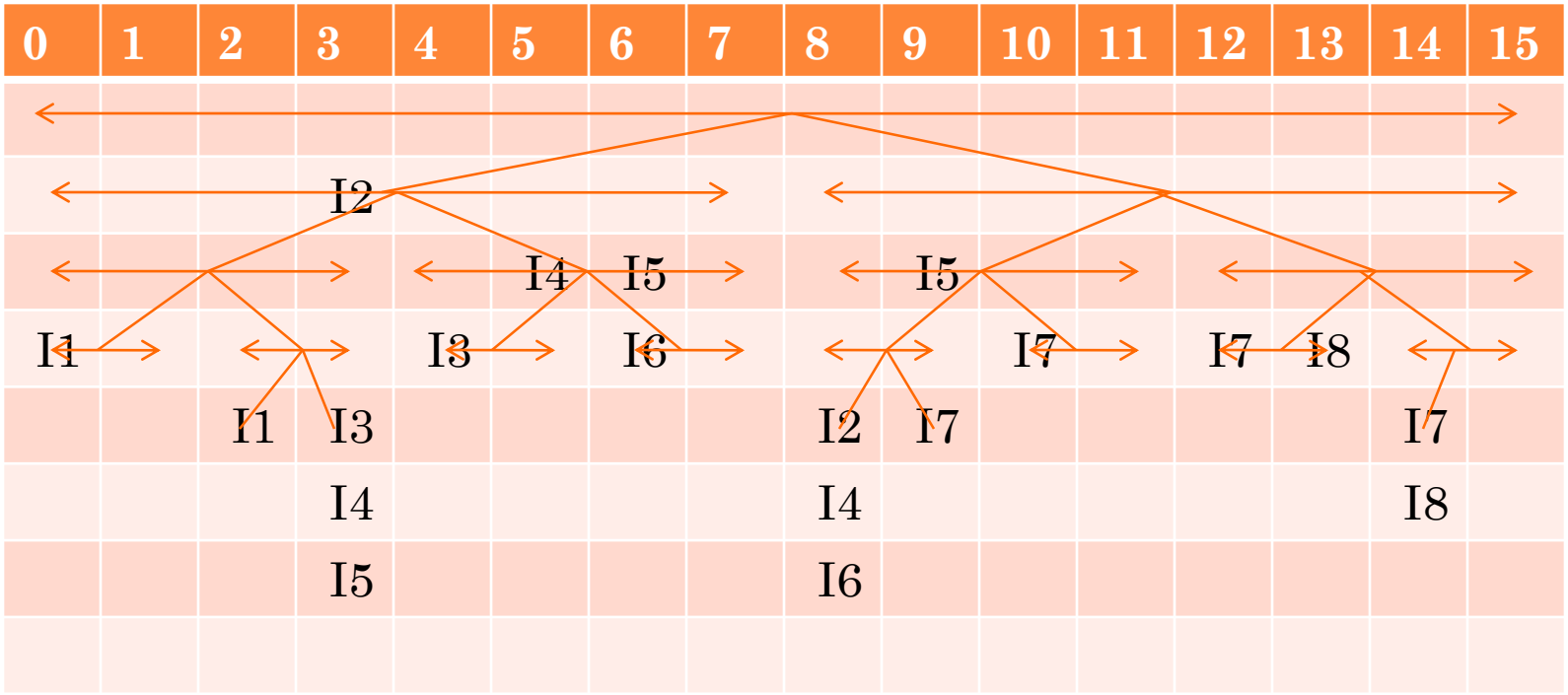
Een mogelijke grid aanpak voor intervallen

- Stel we leggen een elementair grid met interval grenzen aan over het domein en bouwen daar een hierarchische binaire boom boven op (quad-tree, maar dan voor 1 dimensie)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1	I1	I1	I3	I3	I3	I6	I6	I6				I8	I8	I8	
I2	I2	I2	I2	I2	I2	I2	I2	I2	I7	I7	I7	I7	I7	I7	
			I4	I4	I4	I4	I4	I4							
			I5	I5	I5	I5	I5	I5	I5	I5	I5				

- Elementair grid: lijst van intervallen per grid cel

Interval Bin tree vullen tot op bladniveau; toestand na samennemen



zoekinterval 4 7 I2,I3,I4,I5,I6

Intervallen verwijderd uit bladinterval als ook in ouderinterval

Complexiteit Interval BT en elementair grid

- Voorbeeld opzoeken I9 [4,7]:
- - in Interval BT: 5 v/d 31 knopen bezocht levert 5 doorsnijdingsintervallen (1.25 knoop per interval) ziet er efficient uit

- - in elementair grid: gridcel 4,5,6 en 7 bezocht levert 5 doorsnijdingsintervallen (0.8 knoop per interval) is mogelijk nog efficienter (de doorsnijdingsinterval subsets van de 4 gridcellen moeten hierbij echter wel tot 1 doorsneeverzameling beperkt worden)

Interval Grids in meer dimensies

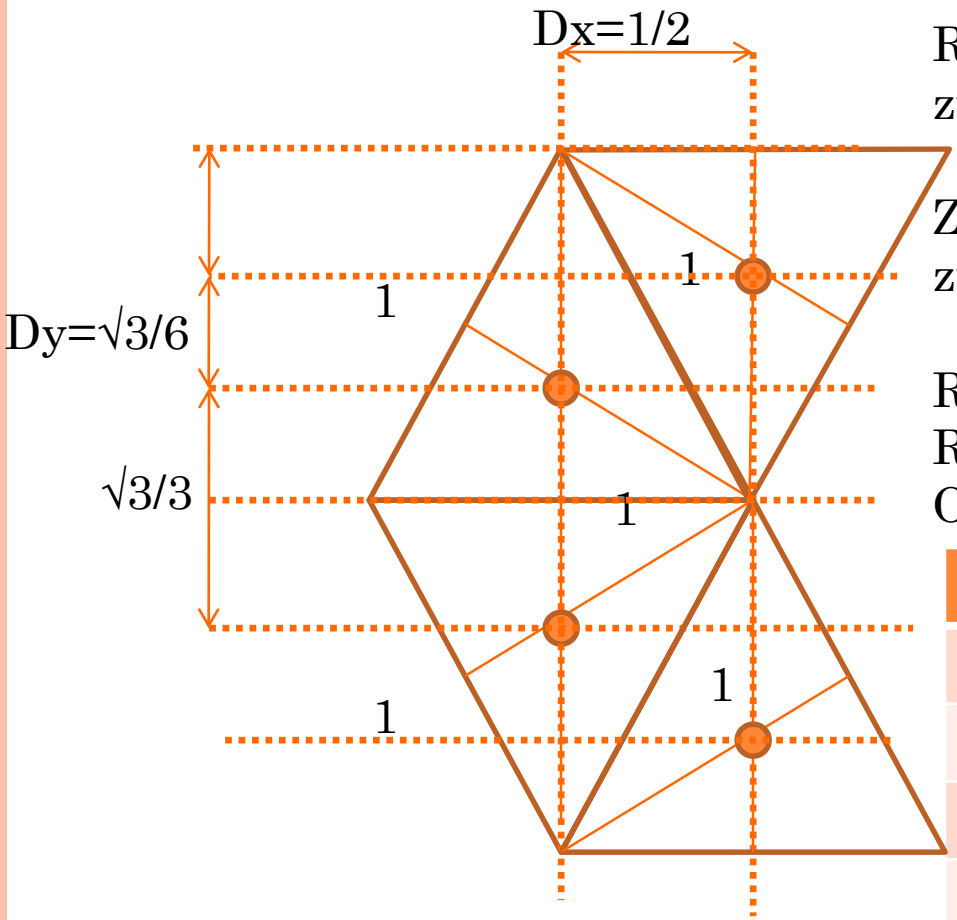
- Elementaire grids en hierarchische grid cellen lenen zich voor meer-dimensionale ruimten
- We kennen al quad- en octtrees
- Het aantal elementaire cellen groeit exponentieel met het aantal dimensies:
- 1D: N cellen
- 2D: N^2 cellen (optimum $\sqrt{N}x\sqrt{N}$ cellen)
- 3D: N^3 cellen
- kD: N^k cellen
- Voordeel is dat per cel de opzoektijd O(1) is
- Als de intervallen sterk clusteren en voor hogere dimensies loont het al gauw om een hierarchische k-dimensionale BT op te bouwen

PAUZE

Benodigde gegevens voor visualisatie triominogrid vanuit vierkant grid

- Voor de 5e programmeeropdracht volgen hierna de belangrijkste gegevens om:
- Vanuit een vierkant grid de naaste buren op het driehoeksgrid te vinden
- Voor een visualisatie (b.v. met Qt) de fysieke zwaartepunten en hoekpuntposities horend bij een positie i,j op het vierkantsgrid te bepalen.
- De triomino becijfering van 3 cijfers i,j,k uit [0..5] krijg je in een file met triomino-ID; de cijfer volgorde kan ook makkelijk gegenereerd worden door van de mogelijke cijfervolgordes alleen die te houden die niet aflopend zijn.

VERBAND ZIJDE DRIEHOEK EN ROOSTERAFSTAND



Roosterafstand = afstand tussen zwaartepunten buurdriehoeken

Zwaartepunt = snijpunt zwaartelijnen op 1/3 van basis

Roosterafstand Dy = √3/6 (zijde=1)
Roosterafstand Dx = 1/2
Oppervlak =√3 /2

	+		+		+
+		+		+	
+		+		+	
	+		+		+
	+		+		+

Dx=1/2
Dy=√3/6

Door zwaartepunten Δ betegeling kan een rechthoekig Dy,Dx grid gelegd worden

Fysieke locatie tegel[i,j] in Δ-grid

Rij 0 start met tegel op top;
rij 1 naar rechts verschoven met tegel op basis
Maat zijde tegel is z
Zwaartepunt tegel[0,0] op $y_z = \sqrt{3}z/6$ en $x_z = z/2$
Twee modulo functies zijn nodig vanwege:
-- Samenneming van 3 halfgridrijen naar 2 vierkantgridrijen
-- inspringstructuur die zich om de 4 regels herhaalt
Zwaartepunt tegel[i,j]: $k = (i \bmod 2)$ $l = (i \bmod 4)$ $n = i/2$

$$y_z = \sqrt{3}z/6 + n\sqrt{3}z/2 + kz\sqrt{3}/6$$
$$= \sqrt{3}z/6\,(3n + k + 1)$$

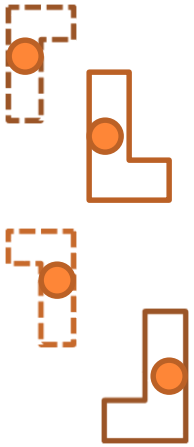
$x_z =$

$l = 0$	$z/2 + j*z/2 = z/2(j+1)$
$l = 1$	$z + j*z/2 = z/2(j+2)$
$l = 2$	$z + j*z/2 = z/2(j+2)$
$l = 3$	$z/2 + j*z/2 = z/2(j+1)$

L-TEMPLATE VOOR AFTASTEN NAASTE BUREN Δ-BETEGELING OP VIERKANT GRID

Aftastvolgorde naaste 3 buren:
Als rij i modulo 4 =

1
2
3
0



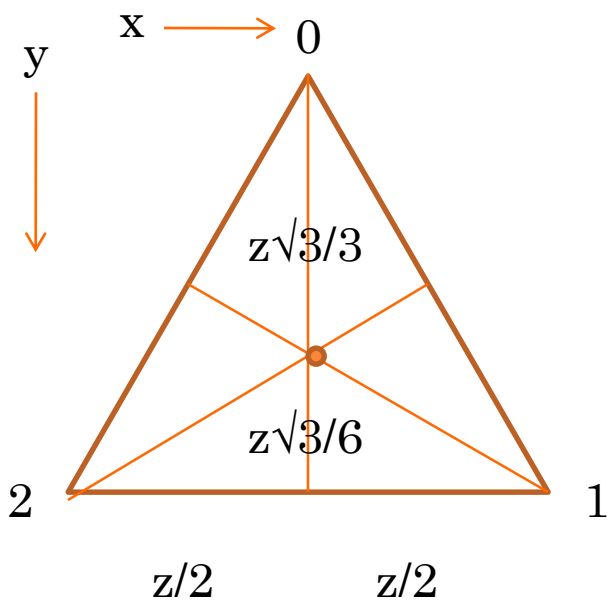
Naaste buren configuratie1:
i-1,j
i-1,j+1
i+1,j

Naaste buren configuratie2:
i-1,j
i+1,j
i+1,j+1

Naaste buren configuratie3:
i-1,j
i-1,j-1
i+1,j

Naaste buren configuratie0:
i-1,j
i+1,j
i+1,j-1

HOEKPUNTEN DRIEHOEKSTEGELS

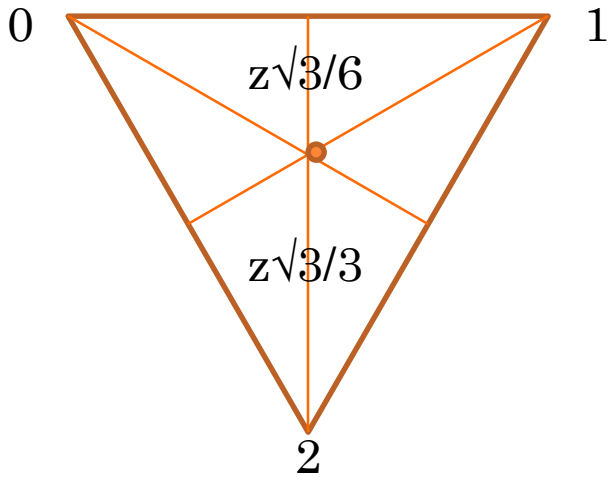


Als zwaartepunt Δ-tegel is y_z,x_z dan zijn met zijde z de coordinaten van de hoekpunten bij basisorientatie:

Hoekpunt0 = $y_z - z\sqrt{3}/3$, x_z

Hoekpunt1 = $y_z + z\sqrt{3}/6$, $x_z - z/2$

Hoekpunt2 = $y_z + z\sqrt{3}/6$, $x_z + z/2$



Als zwaartepunt Δ-tegel is y_z,x_z dan zijn met zijde z de coordinaten van de hoekpunten bij toporientatie:

Hoekpunt0 = $y_z - z\sqrt{3}/6$, $x_z - z/2$

Hoekpunt1 = $y_z - z\sqrt{3}/6$, $x_z + z/2$

Hoekpunt2 = $y_z + z\sqrt{3}/3$, x_z

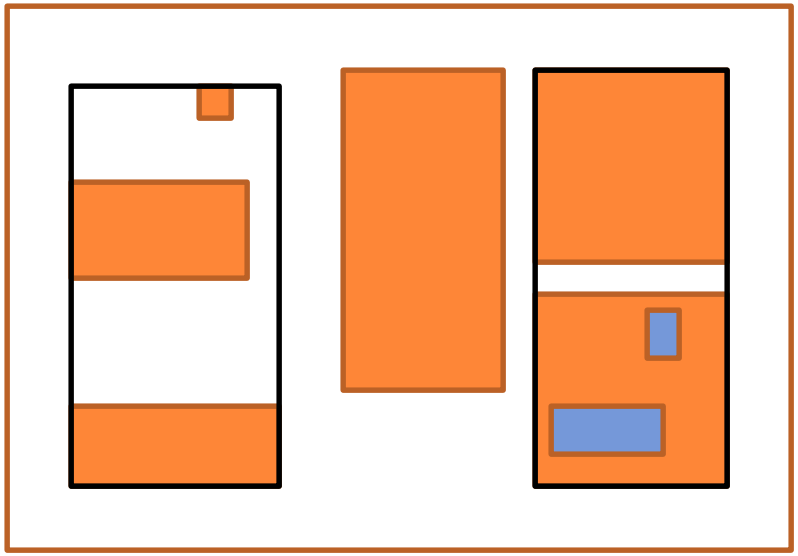
R(echthoek)-Trees

- R-bomen zijn opgezet om te kunnen werken met:
- posities van rechthoeken
- nestingen van rechthoeken
- Overlap van rechthoeken
- Omsluitende rechthoek van rechthoeken

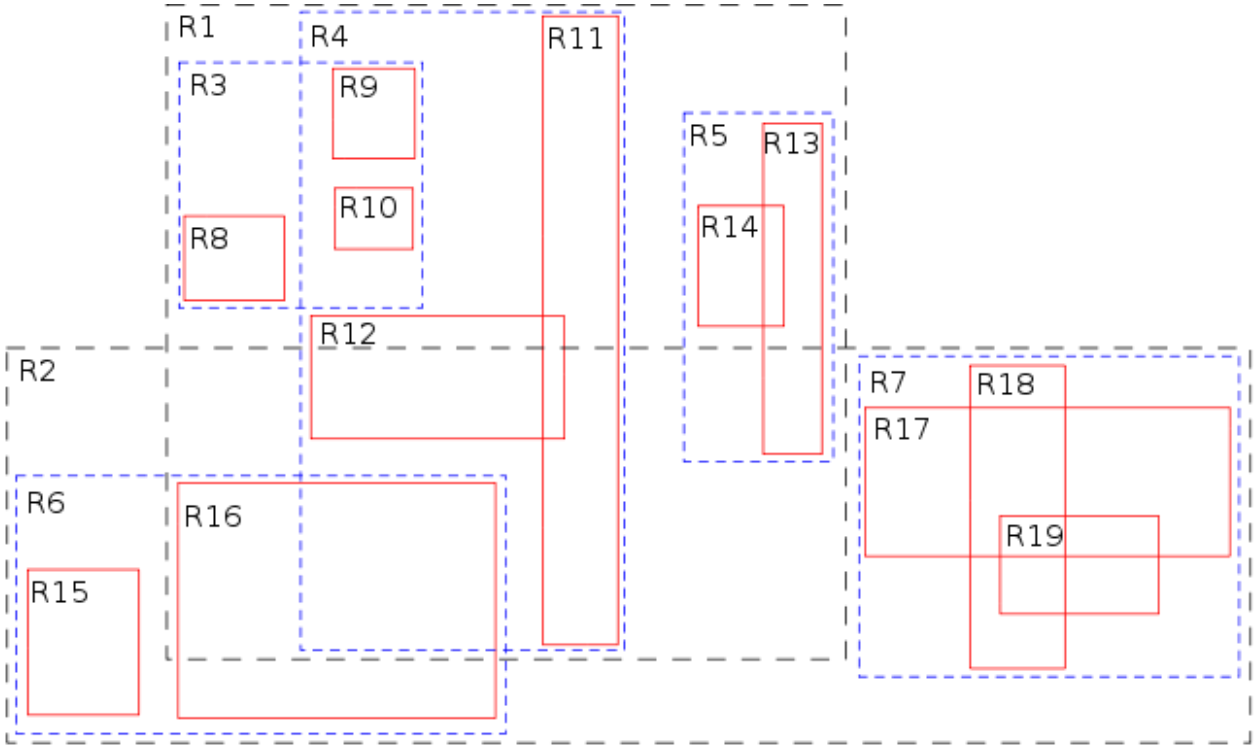
- Voor een architect is een geneste opdeling van een ruimte met een beperkt aantal ruimtes daarbinnen zo veelvoorkomend dat een multi-way opzet (hierarchisch) meer voor de hand ligt dan een binaire opzet (gesorteerd maar 1 dim)

R-TREE
INFORMATIE IN HOGERE NIVEAUS

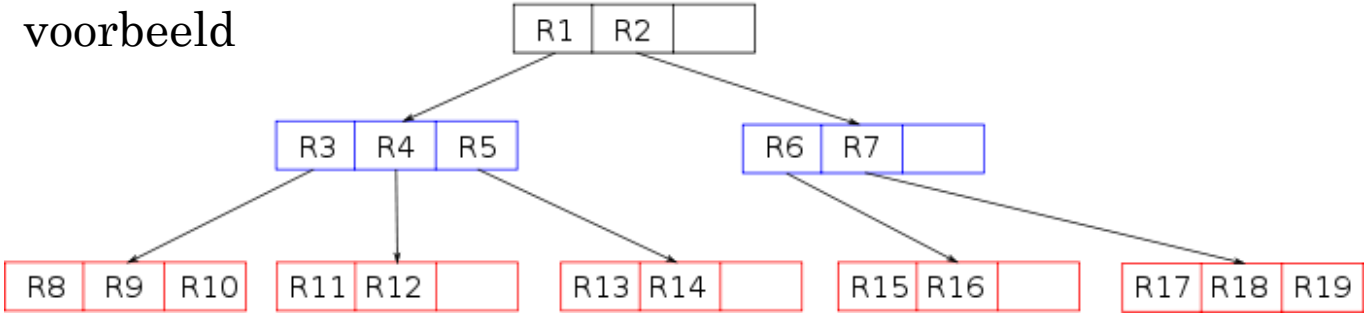
Hierarchie middels MER's (minimal Enclosing Rectangles)
Geneste niveaus



Organisatie van een R-tree



Wikipedia voorbeeld



Echte data MER's in bladeren (op 1 niveau)

Gezamenlijke MER's Op hogere niveaus

Bladinfo in blok op schijf

R-Tree bewerkingen

- Zoeken:
- zoekgebied beschreven door MER (zoek-MER)
- Doorzoek die kinderen waarbij er overlap is tussen MER in detreffende knoop en zoek-MER
- Op bladniveau: bekijk elke overlap van zoek-MER en blad-MER's
- Lastigst: gebalanceerd houden R-Tree
- Randvoorwaarden: bladinfo in 1 blok en blok minstens halfvol

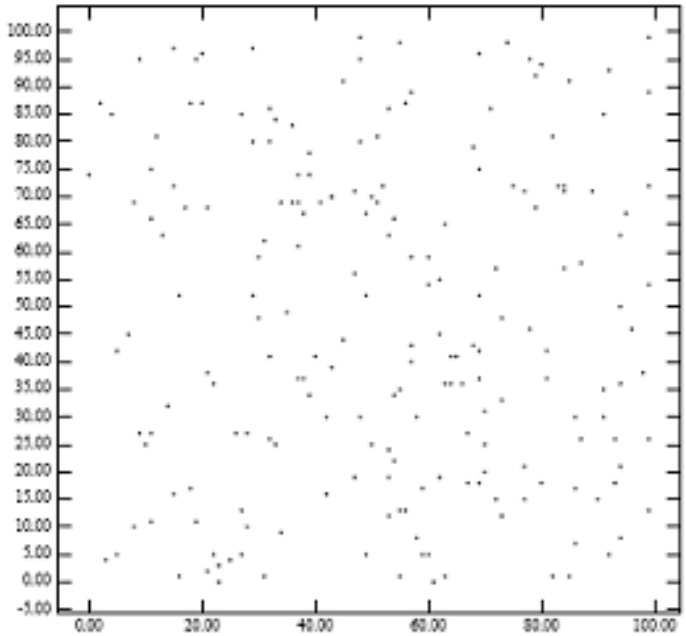
- Nadeel R-Tree: optreden slicing (smalle MER's)

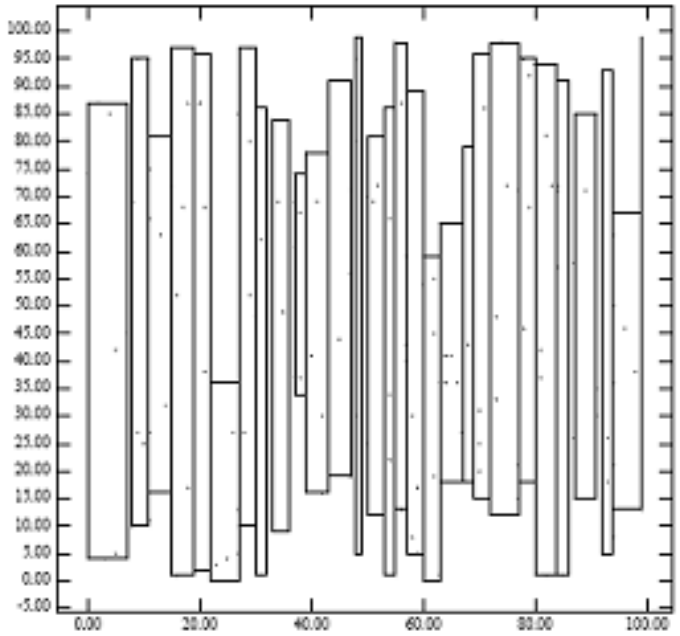
R-Tree insert

- Toevoegen MER aan R-Tree:
- - probeer een bestaande MER te vinden die de minste uitbreiding nodig heeft
- - als bladknoop vol: in 2-en splitsen (opsplitsen is niet triviaal -> varianten R-tree)
- - als aantal kinderen niveau er boven te groot: reorganisatie R-Tree met mogelijk 1 extra tussen niveau voor alle bladeren (aanpak niet triviaal -> varianten R-Tree)
- Quadratic split (Guttmann): begin twee nieuwe groepen met de twee slechtst bij elkaar passende
- R*-Tree split: minimaliseer MER overlap op alle niveaus en ga slicing tegen

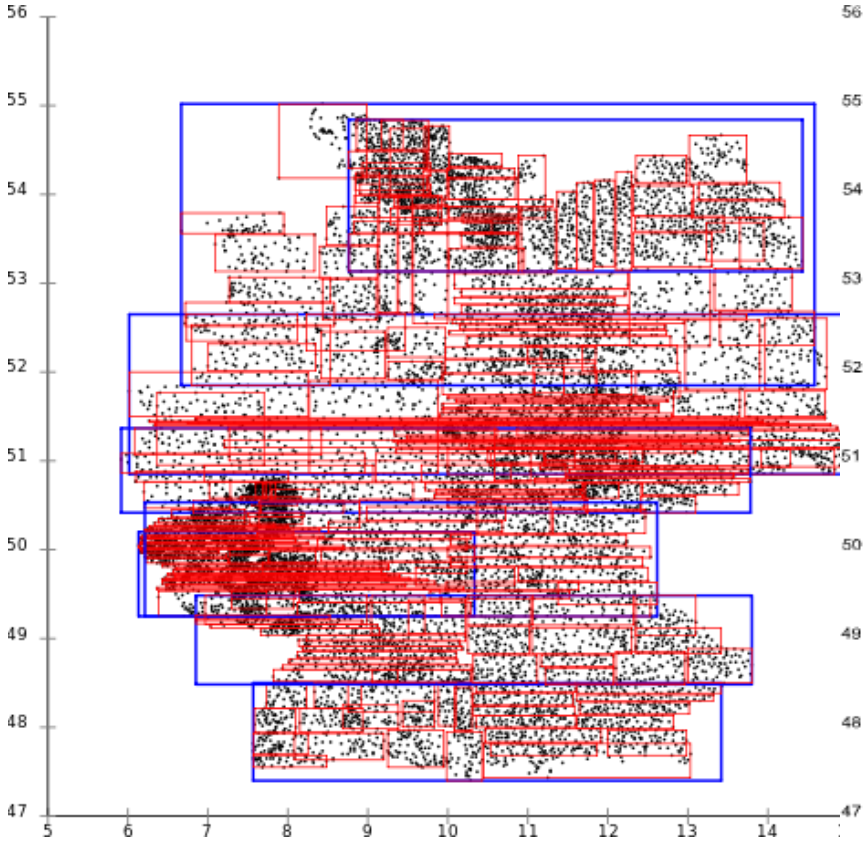
SLICING

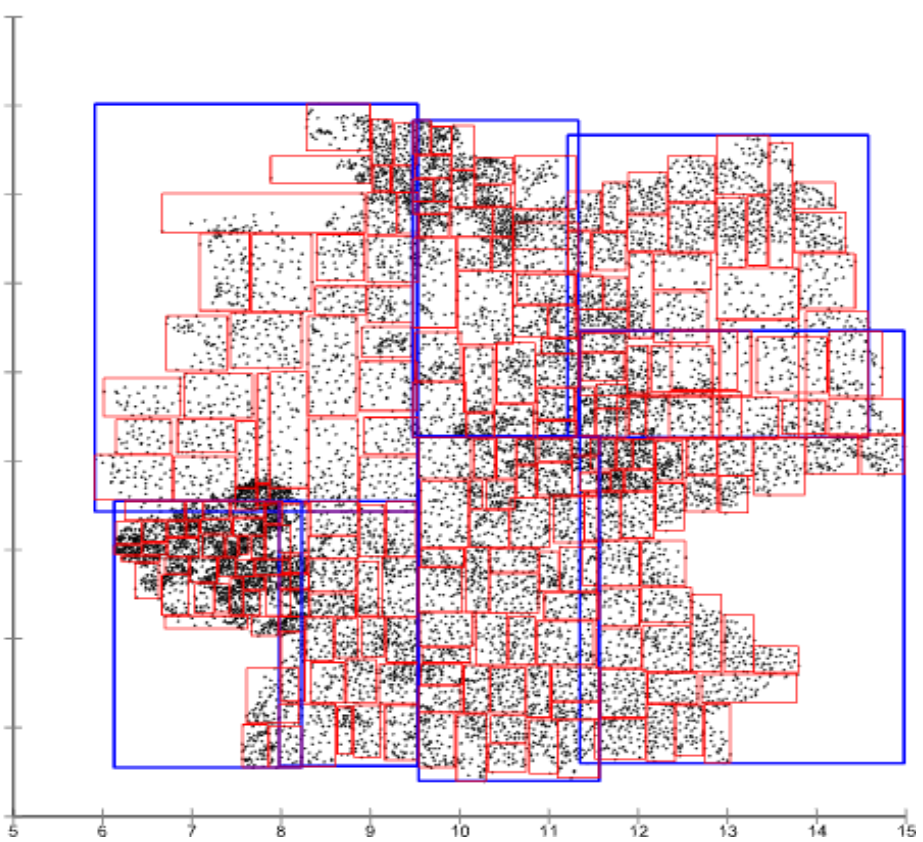
- Veelgebruikte manier om MER's in groepen op te delen maakt gebruik van sortering op 1 van de hoekpunten (b.v. LO-x of LB-y)
- Veel kleine objecten -> slicing (smalle MER's)





Voorbeelden vulling





Oorspronkelijke R-Tree

R*-Tree variant

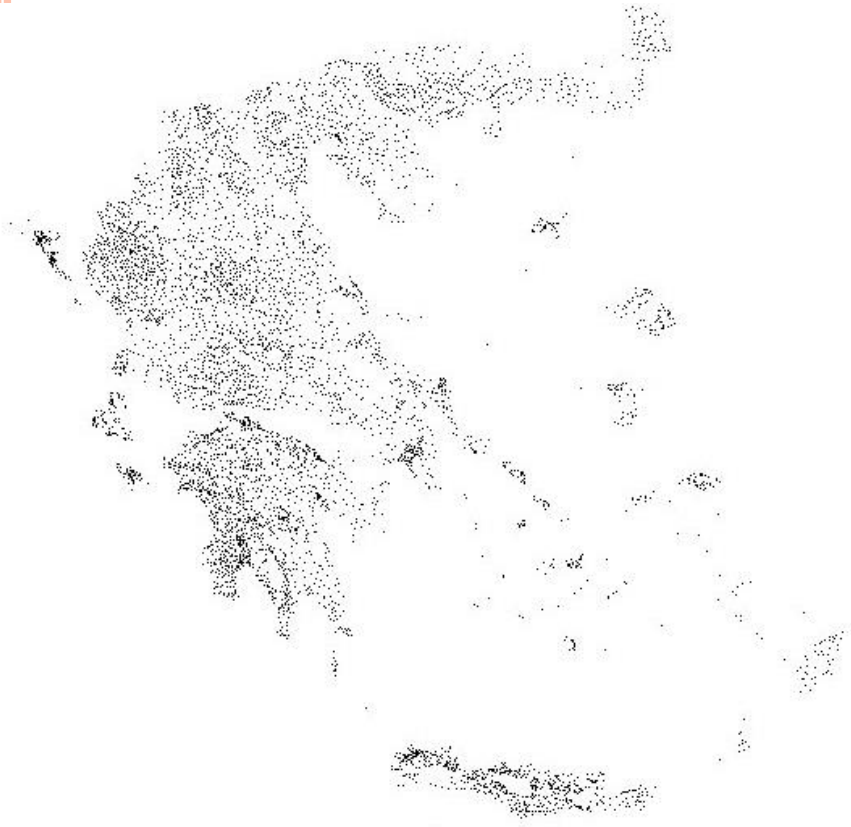
Voorbeelden: wikipedia Duitse Postcodegebieden

R-Tree delete

- Als knoop vol genoeg:
- - verwijder MER uit blad knoop
- - pas MER knoop erboven aan
- Anders:
- - delete blad knoop en voeg overgebleven MER's opnieuw toe aan R-Tree (mogelijk met verlaging aantal niveaus in boom)

Code en testdata voor R-trees

- www.rtreeportal.org heeft codes en datasets:



Griekse steden

Duitse autowegen

Alternatief voor MER?

- Nadeel: 4 hoekpunten; sortering op 1 ervan voor onderverdeling
- Alternatieve karakterisering punt of polygon?

De Priority R-Tree (PR-Tree)
Een verbeterde R-Tree

Als goed voorbeeld van een 2D interval (oppervlak van MER) boom dat een verbetering is van de vele R-tree varianten laat ik nu nog iets zien van de Priority R-Tree.

Voor de nu volgende presentatie over de PR-Tree gebruik ik een voordracht van een student van de bekende Prof. Hanan Samet (specialiteit ruimtelijke datastructuren) die het artikel Priority R-Tree van Lars Arge, Mark de Berg, Herman J. Haverkort en Ke Yi qua werking heel visueel uitlegt.

De pdf file is op Internet te vinden onder:
www.cs.umd.edu/class/spring2005/cmsc828s/slides/prtree.pdf