Datastructuren 24 maart 2021

Dit tentamen bevat vier opgaven.
(meestal)

Graag elke opgave op een nieuwe pagina beginnen.
(wel zo overzichtelijk)

Pseudo-code mag do/od gebruiken of er meer als C++ uitzien, ...
(dáár hoef je je niet druk om te maken)

Geef steeds voldoende uitleg.
(ik kan niet atijd goed raden wat je bedoelt)

Succes.
(natuurlijk)

1) stapelwandeling: pre-orde, in-orde

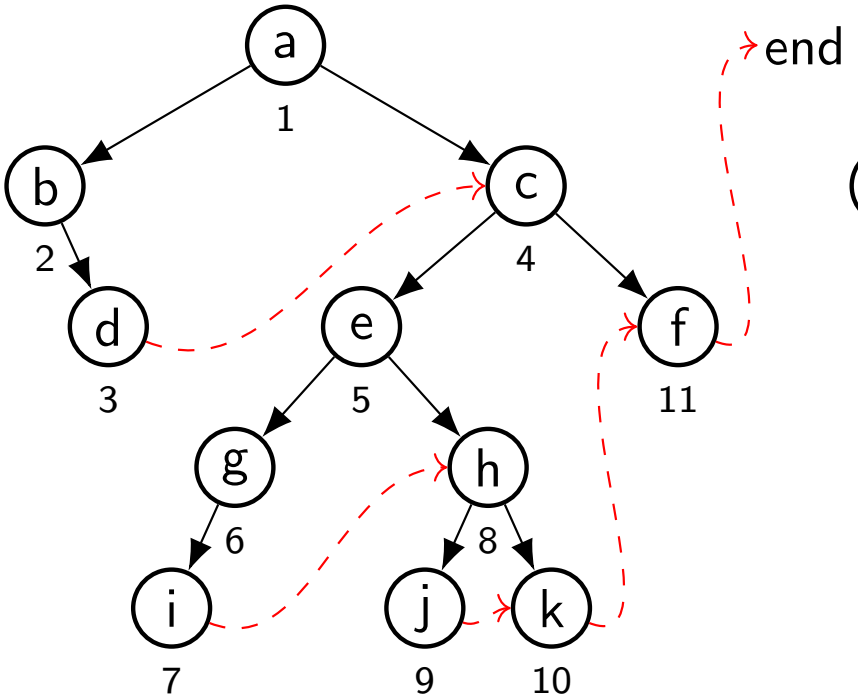
```
iterative-preorder( root )
    S : Stack
    S.create()
    S.push( root )
    while ( not S.isEmpty() )
    do  node = S.pop()
        while (node != nil)
        do  visit( node ) // pre-orde
            S.push( node.right )
            node = node.left
        od
    od
end // iterative-preorder
```

```
iterative-inorder( root )
    S : Stack
    S.create()
    node = root
    // move to first node (left-most)
    while (node != nil)
    do  S.push( node )
        node = node.left
    od
    while ( not S.isEmpty() )
    do  node = S.pop()
        visit( node ) // inorder
        node = node.right
        while (node != nil)
        do  S.push( node )
            node = node.left
        od
    od
end // iterative-inorder
```

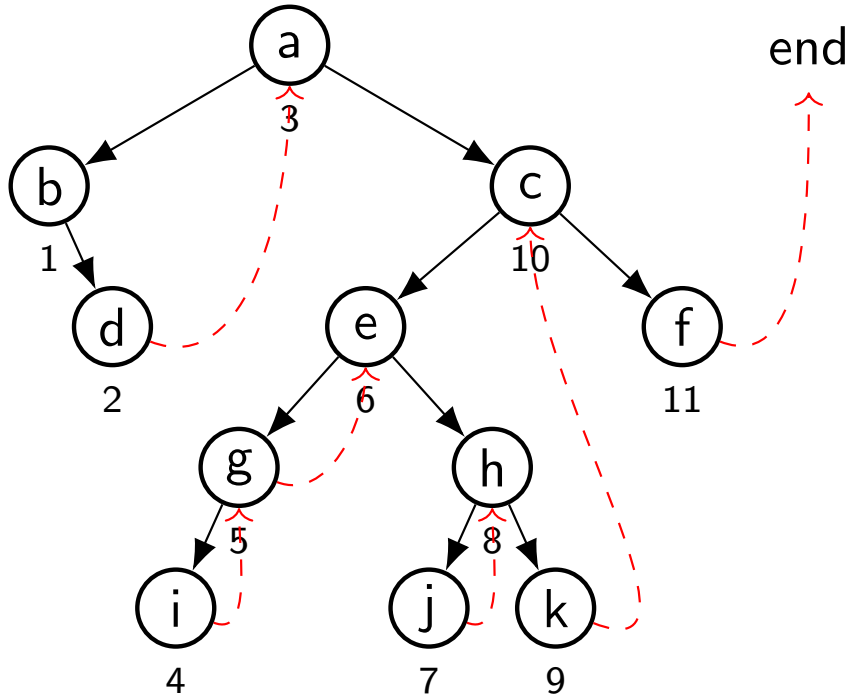
a. Leg uit welke knopen van de boom er op de stapel staan op het moment dat een bepaalde knoop `node` bezocht wordt.

which nodes on stack

pre: right children



in: left parents



Getekend zijn 'draden' (naar opvolgers)

Bij pre-orde worden ook rechter nil-pointers op de stapel gezet.

1) post-order, zonder stapel

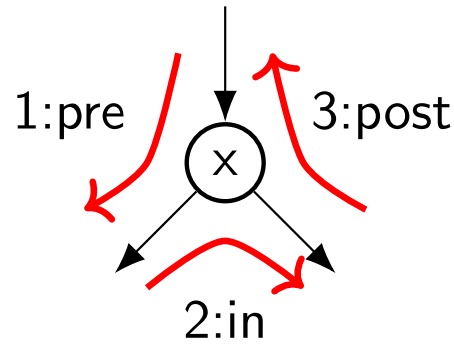
b. Representatie met `left`-, `right`- en ook een `parent`-pointer

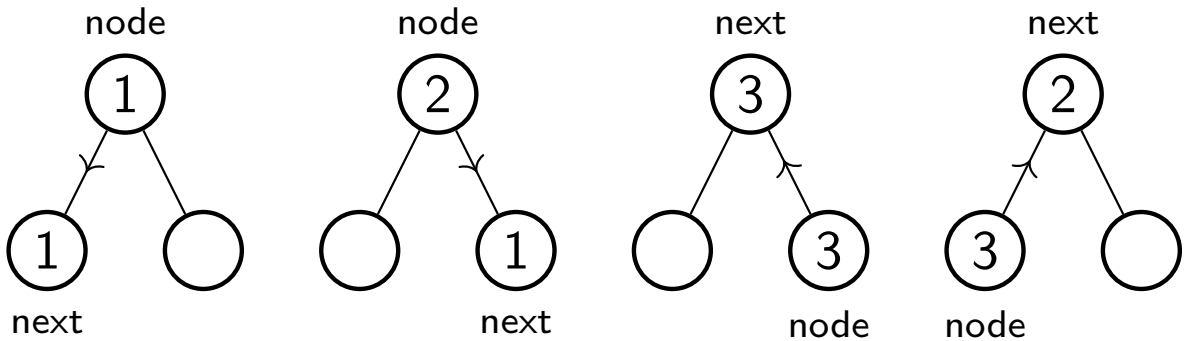
Geef een passend algoritme voor de post-orde wandeling.

'normaal': pad naar wortel op stapel. hier: via `parent` beschikbaar.

Euler traversal (algoritme)

visit	test	direction	next
1	has left-child	down-left	1
		(stay)	2
2	has right-child	down-right	1
		(stay)	3
3	at left-child	up	2
	at right-child	up	3

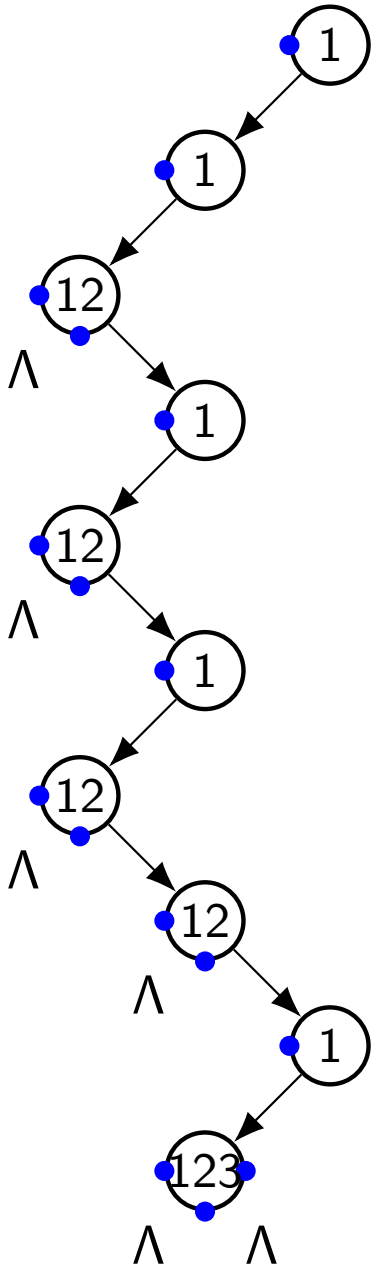


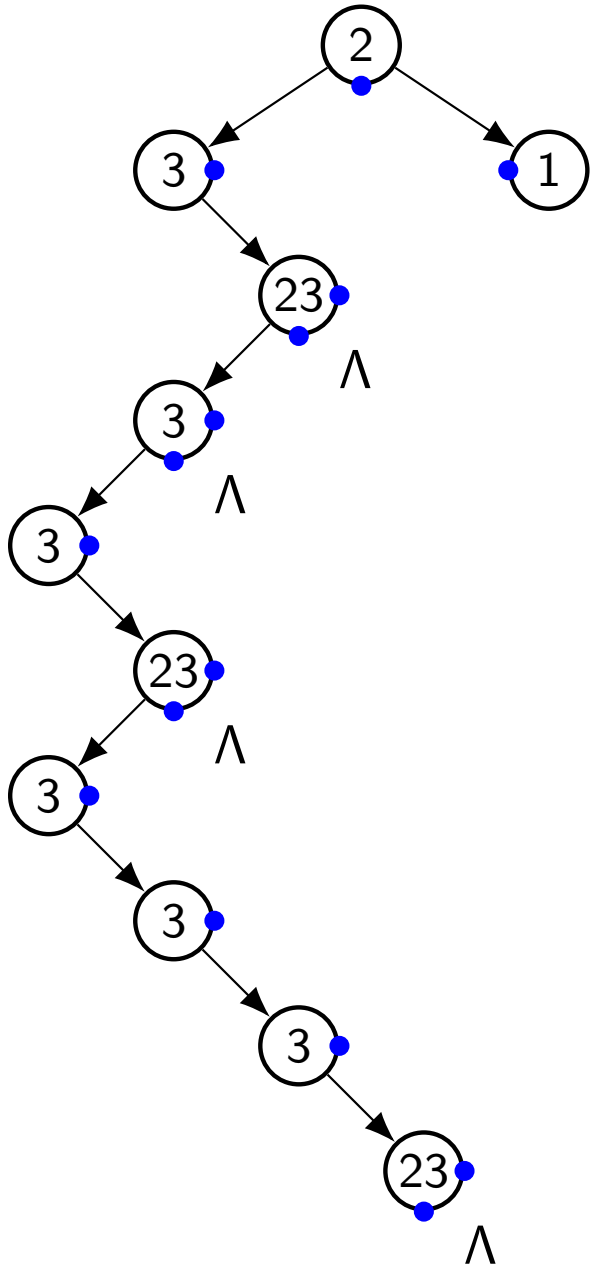


go up hard in binary trees:

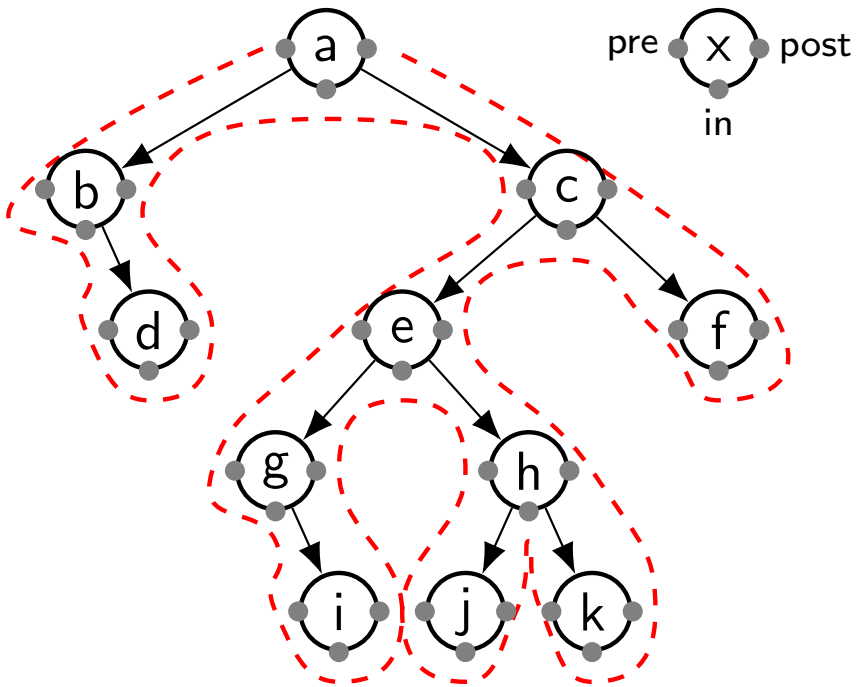
– find parent – at left or right child?

Euler traversal omlaag, omhoog





Euler traversal (gestructureerd)



Euler traversal

```
start at root
while (node not nil)
do  pre-visit (1)
    while (has left child)
    do  go left "push"
        pre-visit (1)
    od
    in-visit (2)
    while (not has right)
    do  repeat
            post-visit (3)
            go up "pop"
            if nil then exit fi
        until (was left)
        in-visit (2)
    od
    go right "push"
od
```

pseudo Drozdek

postorder (pseudo Drozdek)

```
iterativePostorder
    S : Stack
    node = root, last = root
    while (node != nil)
    do      // go down left, 1st visit
        while (node.left != nil)
        do  S.push(node)
            node = node.left
        od
            // go up, 3rd visit
        while (node.right == nil or node.right == last)
        do  visit(node);
            last = node;
            if (S.isEmpty() ) then return fi // exit
            node = S.pop()
        od
            // (2nd visit) go right via parent
        S.push(node)
        node = node.right
    od
end // iterativePostorder
```

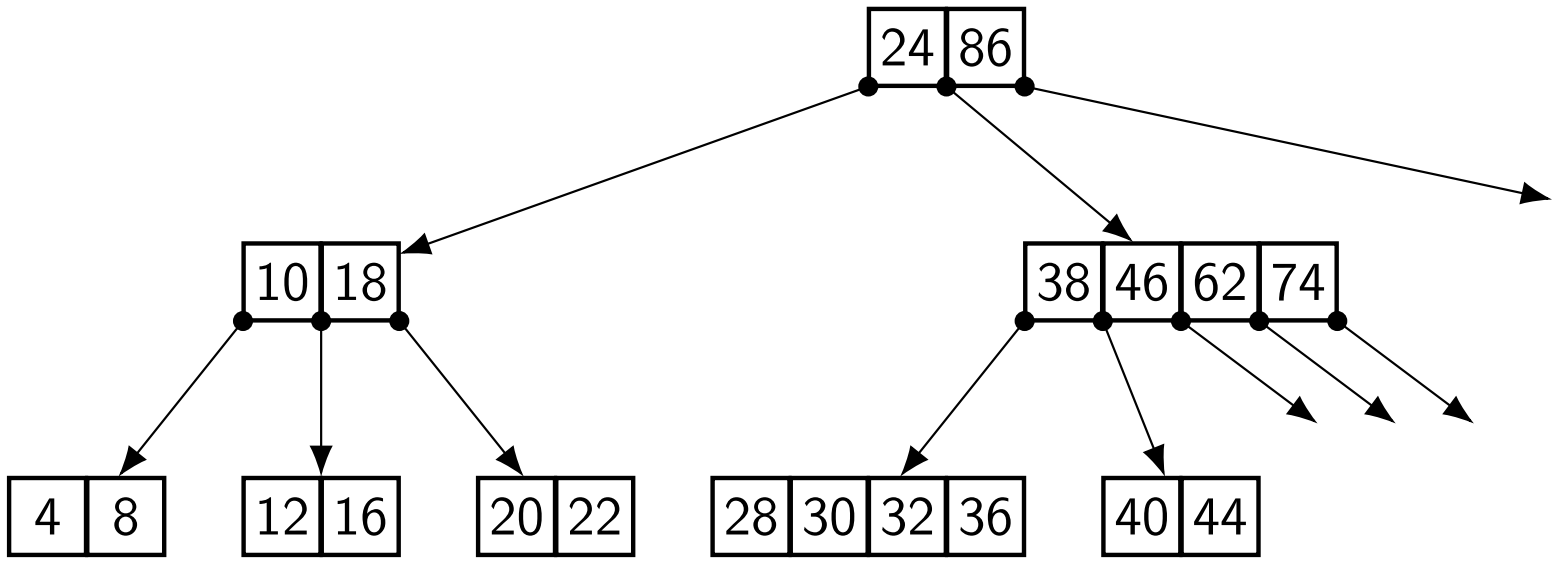
2) B-boom: wortel

a. Op welke manier wordt, in de definitie van een B-boom, de wortel anders behandeld dan de overige knopen? Waarom is dat?

In elke knoop is het aantal kinderen tussen de $m/2$ en m. De ondergrens $m/2$ geldt niet voor de wortel. Bij het toevoegen van een nieuwe sleutel kan de wortel splitsen en dat leidt tot een wortel met twee kinderen. In dat geval geldt de ondergrens dus niet.
Als de wortel splitst wordt de hele boom dus een nivo hoger.

2) B-boom (orde 5): verwijderen

orde 5: tussen 3 en 5 kinderen / tussen 2 en 4 sleutels

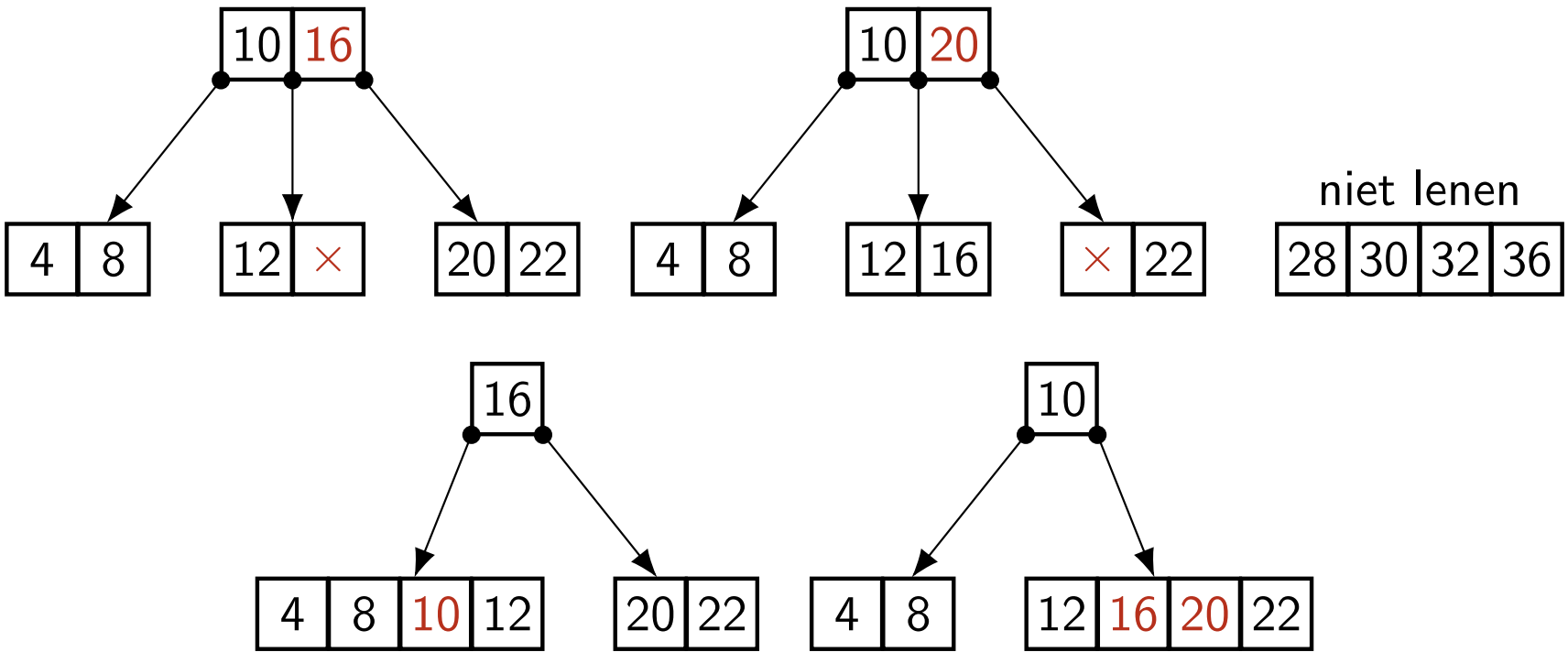


b. i Verwijder sleutel 18.

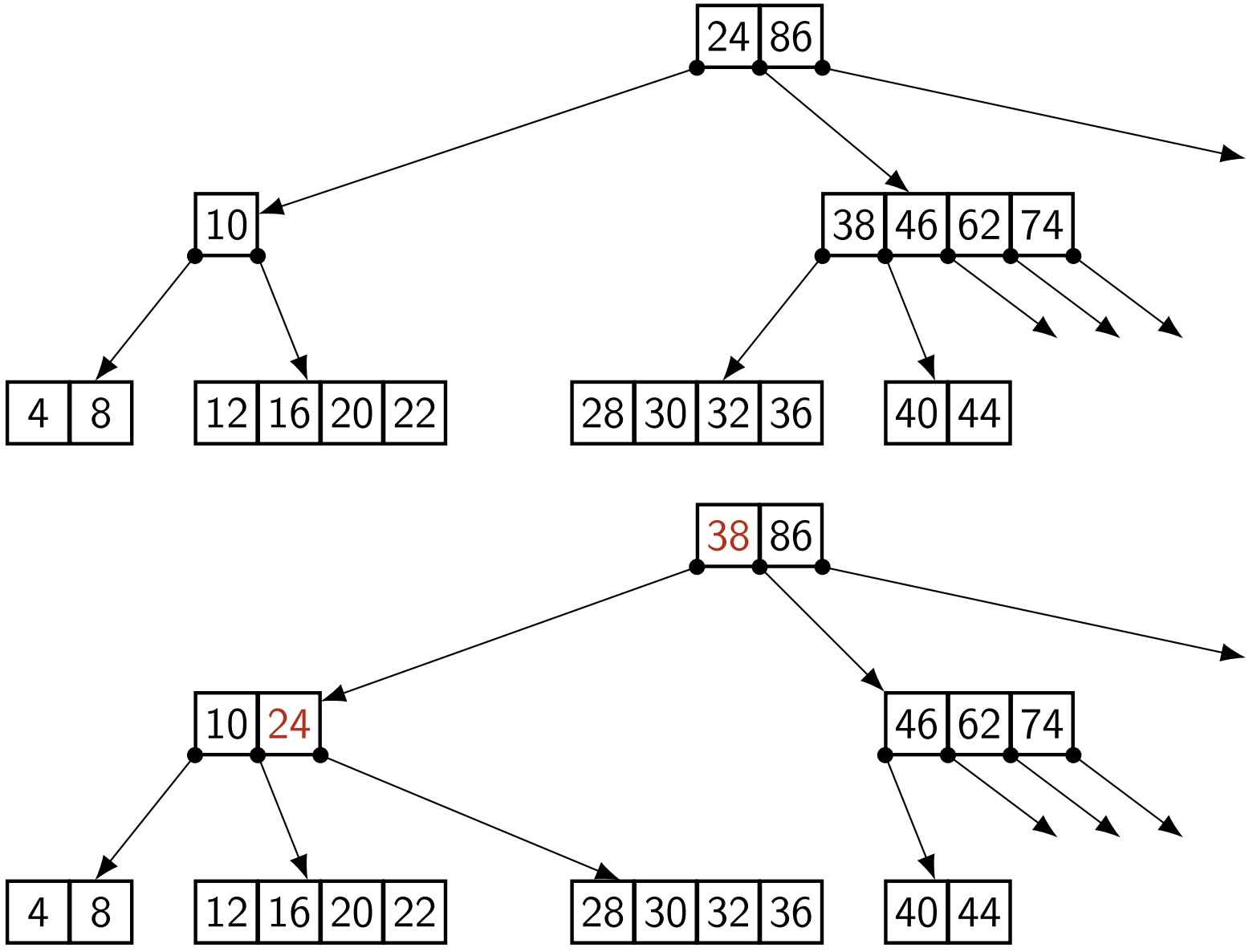
ii Voeg sleutel 18 opnieuw toe.

2b) verwijder sleutel

Altijd in blad! verwissel voorganger / opvolger
te klein: lenen broers anders: samenvoegen, via ouder
(er zijn dus keuzes: voorganger/opvolger, en linker/rechterbroer)

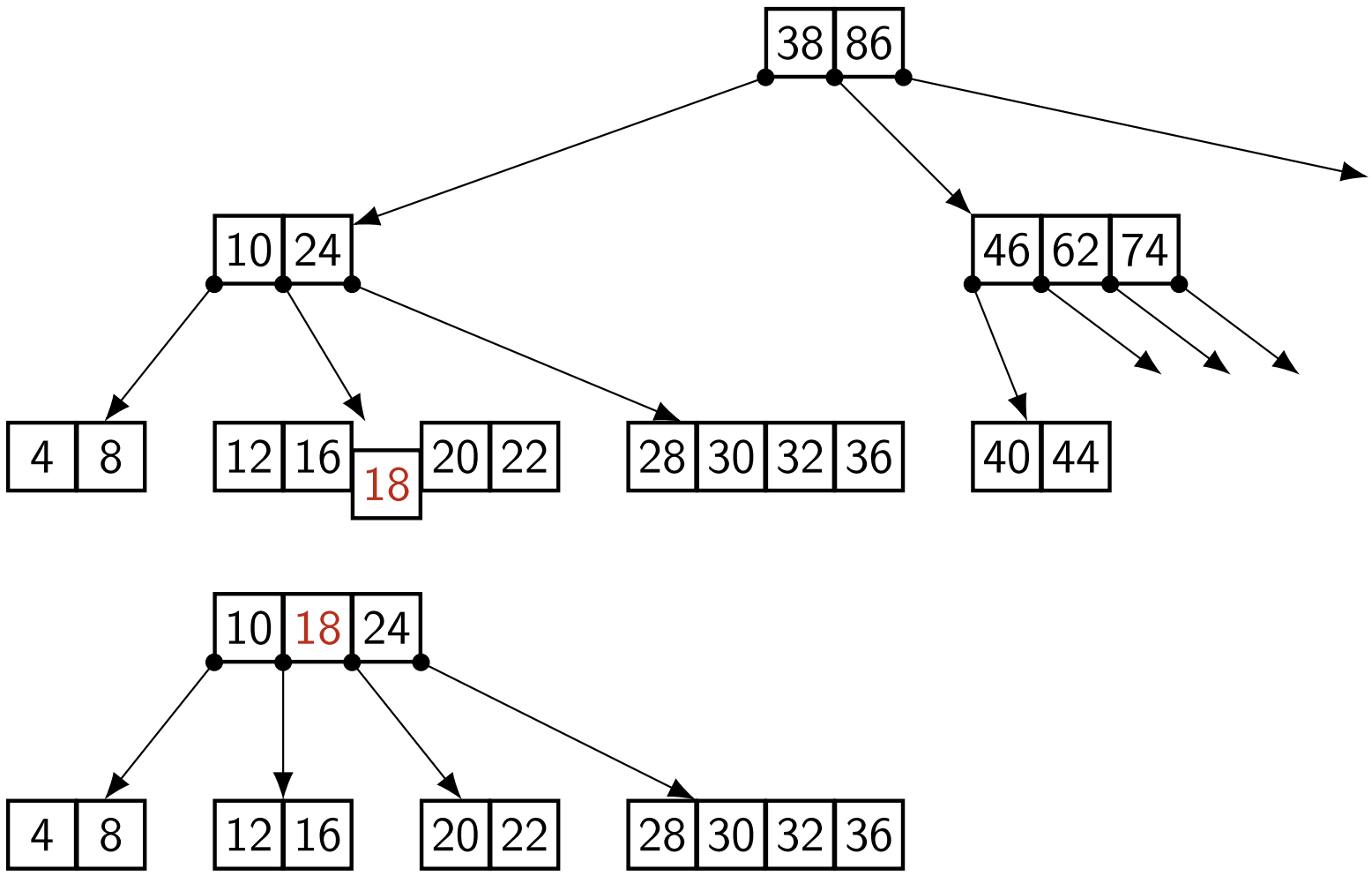


2b) verwijderen: en weer knoop te klein



2b) voeg sleutel toe

Altijd in blad! op de juiste plek
te groot: splitsen (nee: niet naar broer verplaatsen)



2) Hoeveel sleutels

orde m: tussen $\lceil \frac{m}{2} \rceil$ en m kinderen / tussen $\lceil \frac{m}{2} \rceil - 1$ en $m - 1$ sleutels

c. We kijken naar een B-boom van orde m. Bij toevoegen van een sleutel moet soms een knoop gesplitst worden.
- Hoeveel sleutels heeft de knoop op dat moment?
- En hoeveel sleutels hebben de twee nieuwe knopen?

– met één teveel dus m sleutels

neem m even:
– $\frac{m}{2}$ en $\frac{m}{2} - 1$ sleutels (want één naar ouder)
als m oneven: $\lceil \frac{m}{2} \rceil = \frac{m}{2} + \frac{1}{2} = \frac{(m+1)}{2}$
– twee keer $\frac{(m-1)}{2}$ sleutels dat is precies $\lceil \frac{m}{2} \rceil - 1 = \frac{(m-1)}{2}$

3) Max-heap

a. Geef de definitie van een *binary heap*, en beschrijf de twee basis-operaties *bubble-up* en *trickle down*.

Een binary heap is een complete binaire boom (waarbij alle knopen tot aan het laatste nivo helemaal gevuld zijn, en het laatste nivo van links af gevuld wordt) (en daarom gerepresenteerd als array) en waarvan de knopen de heap ordening hebben (de waarde van elke knoop is groter dan van de kinderen).

bubble-up: verwissel de waarde van een knoop herhaald met die van de ouder totdat de waarde kleiner is dan die van de ouder.

trickle down: verwissel de waarde van een knoop herhaald met die van het grootste kind totdat de waarde groter is dan die van de kinderen.

3) Max-heap Insert, DeleteMax

b. Illustreer hoe de priority queue operaties *DeleteMax* (en . . .) worden uitgevoerd, aan de hand van de volgende heap:
[90, 75, 80, 70, 55, 40, 60, 15, 45, 10, 30, 20].

1	2	3	4	5	6	7	8	9	10	11	12
90	75	80	70	55	40	60	15	45	10	30	20

1	2	3	4	5	6	7	8	9	10	11	12
20	75	80	70	55	40	60	15	45	10	30	

1	2	3	4	5	6	7	8	9	10	11	12
80	75	60	70	55	40	20	15	45	10	30	

b. . . . en *Insert*(50)

1	2	3	4	5	6	7	8	9	10	11	12	13
90	75	80	70	55	40	60	15	45	10	30	20	50

1	2	3	4	5	6	7	8	9	10	11	12	13
90	75	80	70	55	50	60	15	45	10	30	20	40

3) Heapify

c. Heapify/Makeheap is een methode om van een array een heap te maken (dus de sleutels in heap-order te zetten).
Als op de juiste manier gedaan kan dat in lineaire tijd:
Voeg van achter naar voren(!) de sleutels toe aan de (onderliggende) heapstructuur, steeds met trickle down.

1	2	3	4	5	6	7	8	9
50	10	30	90	20	40	70	80	60

De sleutels op adressen 5 tm. 9 hebben geen kinderen, en staan in eerste instantie al goed. We beginnen met sleutel 90 op adres 4. Deze is groter dan de kinderen op adres 8 en 9. Etc.
Elk paar kinderen onderstreept, evt. verwisselen met grootste.

1	2	3	4	5	6	7	8	9	
50	10	30	90	20	40	70	80	60	$90 >$ kinderen 80,60
50	10	70	90	20	40	30	80	60	$70 \leftrightarrow 30$
50	90	70	80	20	40	30	10	60	$10 \leftrightarrow 90 \leftrightarrow 80$
90	80	70	50	20	40	30	10	60	$50 \leftrightarrow 90 \leftrightarrow 80$

4) Hashen

a. Welke twee soorten clustering onderscheiden we bij hashen met open adressering? Geef een korte beschrijving.

primaire en secundaire clustering.

primair (bij lineair hashen). Zoekpaden van adres worden samengevoegd met dat van buren, zodat clusters ontstaan, die zichzelf versterken.

secundaire clustering. Sleutels op hetzelfde adres volgen herzelfde zoekpad, dus bij veel synomiemen lange zoekpaden.

4) Hashen

Beschouw hashen in een zgn. 'open' hashtabel met twee hash-functies h en p. Het $i+1$-ste bezochte adres $h(K,i)$ is zoals gewoonlijk $h(K) - i \cdot p(K)$ (modulo M).

b. In een tabel `T[0..10]`, dus $M = 11$, worden achtereenvolgens de sleutels 16, 29, 36, 40, 9, 20, 28 geplaatst, met adresfunctie $h(K) = K \bmod 11$ en lineair hashen (stapgrootte 1).
(i) Laat zien welke tabel ontstaat, maar geef op een overzichtelijke manier ook alle plekken waar de sleutels geprobeerd worden.
(ii) Hoeveel stappen kost het om te zien of 18 in T opgeslagen is?

c. Idem, nu met een stapfunctie $p(K) = 1 + (K \bmod 10)$.

4) lineair Hashen

b. $M = 11$, sleutels 16, 29, 36, 40, 9, 20, 28, adresfunctie $h(K) = K \bmod 11$ en lineair hashen (stapgrootte 1).

K	16	29	36	40	9	20	28	18	adressen
$h(K)$	5	7	3	7	9	9	6	7	

0	1	2	3	4	5	6	7	8	9	10	
			36		16		29				vrije adressen
			36		16	40	29				$40 \rightsquigarrow 7\times, 6\checkmark$
			36		16	40	29		9		$9 \rightsquigarrow 9\checkmark$
			36		16	40	29	20	9		$20 \rightsquigarrow 9\times, 8\checkmark$
			36	28	16	40	29	20	9		$28 \rightsquigarrow 6\times, 5\times, 4\checkmark$
		$\times$	$\times$	$\times$	$\times$	$\times$	$\times$				$18 \rightsquigarrow 7\times, 6\times, \ldots, 2\times$

18 wordt niet gevonden, te beginnen op 'huisadres' 7 vinden we een leeg veld op adres 2.

4) dubbel Hashen

b. $M = 11$, sleutels 16, 29, 36, 40, 9, 20, 28, adresfunctie $h(K) = K \bmod 11$ en stapfunctie $p(K) = 1 + (K \bmod 10)$.

K	16	29	36	40	9	20	28	18	
$h(K)$	5	7	3	7	9	9	6	7	thuisadres
$p(K)$	7	10	7	1	9	1	9	9	stapgrootte

Er lijkt hier veel hetzelfde te gaan, tot aan de plaatsing van 28(?!)

0	1	2	3	4	5	6	7	8	9	10	
			36		16		29				vrije adressen
			36		16	40	29				$40 \rightsquigarrow 7\times, 7{-}1 = 6\checkmark$
			36		16	40	29		9		$9 \rightsquigarrow 9\checkmark$
			36		16	40	29	20	9		$20 \rightsquigarrow 9\times, 9{-}1 = 8\checkmark$
			36		16	40	29	20	9	28	$28 \rightsquigarrow 6\times, 6{-}9 \equiv 8\times, 10\checkmark$
$\times$							$\times$		$\times$		$18 \rightsquigarrow 9\times, 7{-}9 \equiv 9\times, 0$

18 wordt niet gevonden, te beginnen op 'huisadres' 7, met stap $9 \equiv -2$ (dus 2 naar rechts) vinden we een leeg veld op adres 0.