1. Zie de diktaat. Hieronder kort opmerkingen. Voeg plaatjes toe.

1a) Draden worden gebruikt om een wandeling te maken zonder externe datastructuur (of recursie). We moeten de draden wel aanpassen bij dynamisch gebruik van de boom, dus bij toevoegen en verwijderen.

b) Sla in elke knoop het aantal sleutels op in de linker subboom. We kunnen nu recursief zoeken. Als het aantal sleutels links ℓ is dan

$k \leq \ell$ ga links verder

$k = \ell + 1$ de huidige knoop bevat de k^e sleutel

$k > \ell + 1$ zoek naar de $k - \ell - 1$-ste sleutel in de rechter subboom.

Natuurlijk is het ook mogelijk het aantal sleutels onder de knnop zelf op te slaan, dan kijken we gewoon bij het linker kind.

Als we een knoop toevoegen dan wordt de waarde opgehoogd in elke knoop waar we naar links gaan.

c) Wordt toegepast bij toevoegen in LR of RL situaties; dwz. de huidige knoop is de laagste knoop uit balans, en de toegevoegde sleutel is terechtgekomen in de rechts-links subboom.

2a) Een B-boom van orde m heeft ten hoogste $m - 1$ sleutels per knoop, en ten minste $\lceil \frac{m}{2} \rceil - 1$ sleutels per knoop. Dat laatste geldt niet voor de wortel, die geen minimum voor het aantal sleutels kent.

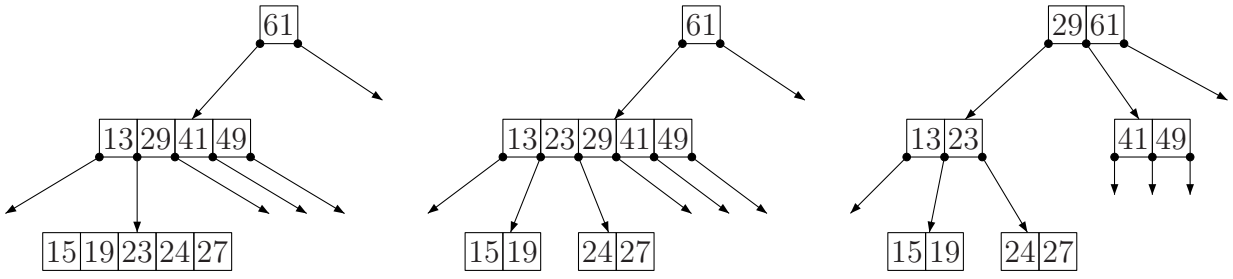
Het aantal kinderen ligt tussen de $\lceil \frac{m}{2} \rceil$ en m.

Er zijn *maximaal* $1 + m + m^2$ knopen, met elk $m - 1$ sleutels. *You do the math*: $m^3 - 1$.

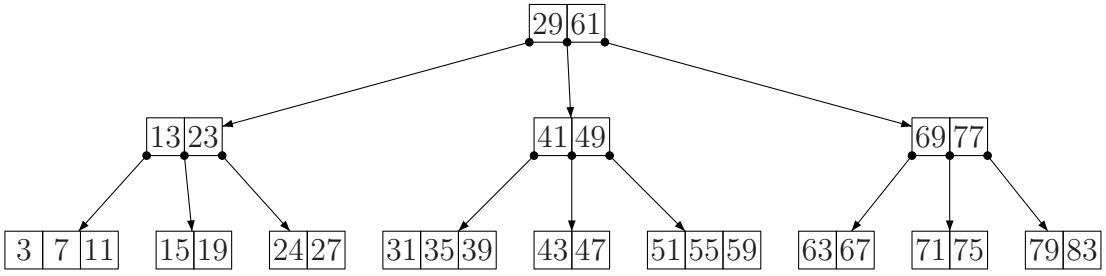
Laten we even schrijven $k = \lceil \frac{m}{2} \rceil$. Dan zijn er minimaal een wortel met één sleutel, en $2 + 2k$ 'gewone' knopen, met elk $k - 1$ sleutels. Bij elkaar $2k^2 - 1$ sleutels.

2b)i Een B-boom van orde 5 heeft per knoop tussen de 2 en 4 sleutels (de wortel mag ook één sleutel hebben).

In de originele boom wordt 24 toegevoegd in het blad $[15, 19, 23, 27]$. We krijgen dan een tussenoplossing, omdat er maximaal 4 sleutels in een knoop passen: Het blad scheurt in twee en het middelste element 23 gaat omhoog. Alweer heeft een knoop een sleutel te veel. Middelste element wordt weer verplaatst naar de ouder (de wortel van de boom).

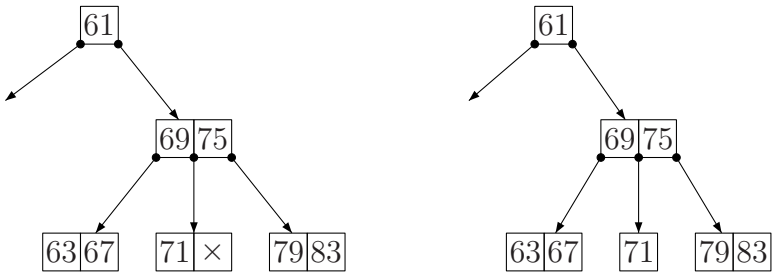


Voor de volledigheid, de boom ziet er nu uit als volgt.

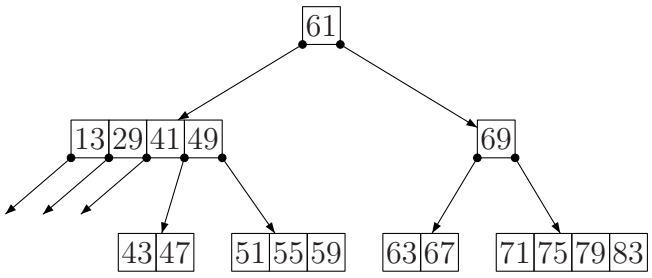


2b)ii Niks aan de hand. Sleutel 50 past gewoon in blad $[51, 55, 59]$ en komt daar vooraan te staan.

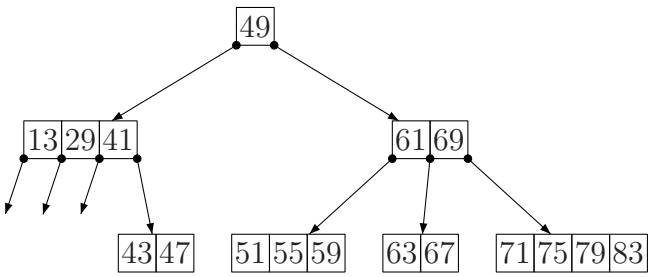
2c) Knopen worden uit een blad verwijderd. We verwisselen eerst 77 met zijn voorganger 75 (verwisselen met opvolger mag ook). Verwijder nu 77 (al gemarkeerd). We krijgen een blad met één sleutel. Lenen van de buren werkt niet.



Daarom moeten we samenvoegen met één van de broers (en de bijbehorende tussenliggende sleutel uit de ouder).



Helaas levert dit een knoop met een enkele sleutel. Lenen bij de buren (broer) is nu wel mogelijk. Dit gebeurt *via* de ouder: de waarde in de wortel verandert!



3. Zie het tentamen van januari 2016.

4a) Als $Flink(k) = r$ dan is de suffix ter lengte $r-1$ voorafgaand aan positie k gelijk aan de prefix (van gelijke lengte) voorafgaand aan positie r. Meer precies $P_1 \dots P_{r-1} = P_{k-r+1} \dots P_{k-1}$.

b) We vinden deze tabel voor Flink

1	2	3	4	5	6	7	8	k
a	b	a	c	a	a	b	a	P_k
0	1	1	2	1	2	2	3	Flink(k)

Opm. Probeer niet alle prefixen uit, maar gebruik het 'efficiente' algoritme. Dat algoritme wordt hier niet gevraagd, maar informeel: om de failure link van positie k te vinden volgen we de failure links van positie $k-1$ op zoek naar de letter van positie $k-1$. Als we die vinden tellen we 1 bij de gevonden positie op.

Opm. Inderdaad: Flink(8) = 3 waar ab het langste prefix is dat ook een suffix is vóór positie 8. Dat prefix aba ook een suffix is tot en met positie 8 is hier niet belangrijk, dat zou op positie 9 weer belangrijk zijn.

c) Positie is hier de positie van de tekst T.

Stap 1. Match positie 1,2,3. Mismatch P_4 op positie 4. Flink(4) = 2. Vervolg met P_2 op (dezelfde) positie 4.

Stap 2. Match letter P_2 positie 4. Mismatch P_3 op positie 5. Flink(3) = 1. Vervolg met P_1 op positie 5.

Stap 3. Mismatch. Flink(1) = 0. Nu: vervolg met P_1 op de volgende positie 6.

Stap 4. Match letter P_1 positie 6. Mismatch P_2 op positie 7. Flink(2) = 1. Vervolg met P_1 op positie 7.

Stap 5, net als stap 1. Match positie 7,8,9. Mismatch P_4 op positie 10. Flink(4) = 2. Vervolg met P_2 op (dezelfde) positie 10.

Stap 6. Mismatch P_2 op positie 10. Flink(2) = 1. Vervolg met P_1 op positie 10.

Stap 7. Match letter P_1 positie 10. Mismatch P_2 op positie 11. Flink(2) = 1. Vervolg met P_1 op positie 11.

Dan nog twee stappen. Mismatch P_6 op positie 16. Flink(6) = 2. Mismatch P_2 op positie 16. Letters tekst zijn op. Patroon niet gevonden.

PS. Sorry (jan'20) Ik ben er op gewezen dat de tekst uit de opgave nog langer is dan hier in de uitwerking. Nog doorzoeken dus.

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
T	a	b	a	b	c	a	a	b	a	a	a	b	a	c	a	b
1	a_1	b_2	a_3	c_4												
2				b_2	a_3											
3					a_1											
4						a_1	b_2									
5							a_1	b_2	a_3	c_4						
6										b_2						
7										a_1	b_2					
8											a_1	b_2	a_3	c_4	a_5	a_6
9																b_2

d) Een situatie waar het naieve algoritme heel slecht werkt is als de mismatch steeds in de laatste letter van het patroon zit. Dan wordt eigenlijk elke letter van het patroon met elke letter van de tekst vergeleken. Voorbeeld: $P = AA \ldots AB$ en $T = AA \ldots AB$. KMP schuift daar netjes een letter op elke stap (bepaal de failute links).