Institut for Matematik og Datalogi
Syddansk Universitet, Odense

19. marts 2020

DM507 — Algoritmer og datastrukturer

Eksaminatorie-timer uge 12, Forår 2020

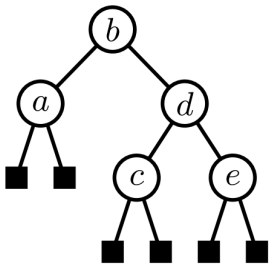
Instruktorerne for DM507

Indhold

1 Opgave 1 - Eksamen juni 2015, opgave 8

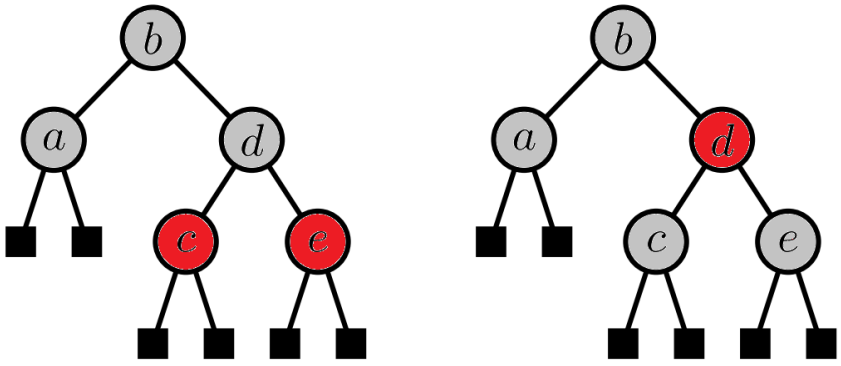
Spørgsmål a

Angiv en farvning af knuderne i træet nedenfor, som gør det til et rød-sort træ.

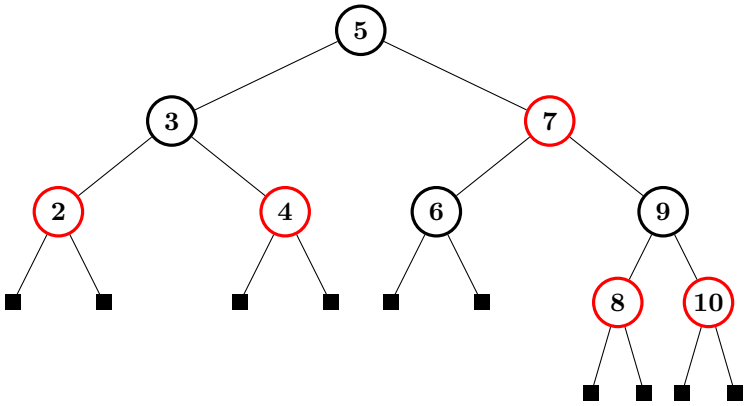


- b er triviel, da b er roden, og roden i et rød-sort træ skal være sort.
- Derefter undersøges knuden a.
  Hvis knuden a bliver farvet rød, så vil der skabes en rod-blad sti, b-a-nil, som indeholder 2 sorte. Der gælder, at alle rod-blad stier, i et ikke-tomt rød-sort træ, vil ramme mindst 2 sorte knuder: roden og bladet. Derfor må ingen andre indre knuder udover roden være sort, da dette vil skabe en rod-blad sti med mere end 2 sorte knuder. Dette betyder, at c,d og e skal farves røde, hvis a farves rød. Dette bryder "ingen to røde i streg"da d-c og d-e kommer i streg. Altså skal a farves sort.
- I forhold til c, d og e's farvning er der nogle mulighed. Alle 3 knuder må ikke farves sorte, da det skaber en rod-blad sti med flere sorte end b-a-nil stien. c og e må ikke farves forskellige farver, da dette betyder, at der vil være én sort mere på b-d-c-nil stien end b-d-e-nil stien, eller én mindre, afhængig af hvilken knude der farves sort.Da alle 3 heller ikke kan være røde, så efterlader det 2 muligheder. d er rød, imens c og e er sorte, eller d er sort, imens c og e er røde.

Dvs. de 2 farvninger af knuderne i træet ovenfor, som gør træet til et rød-sort træ er:

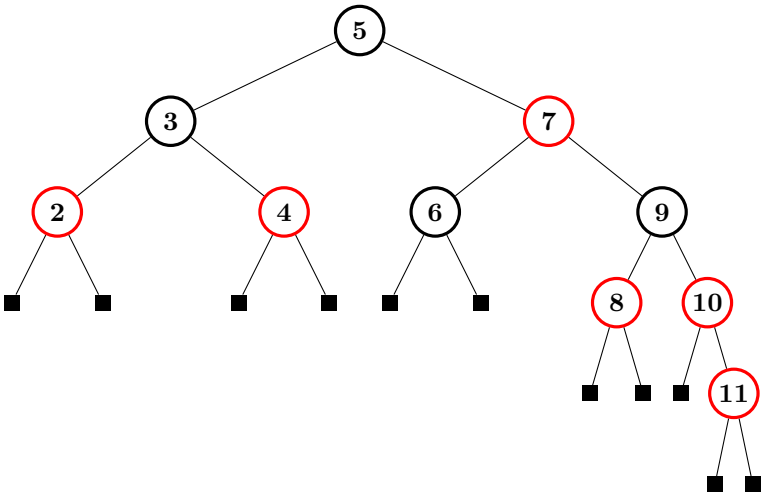


Spørgsmål b

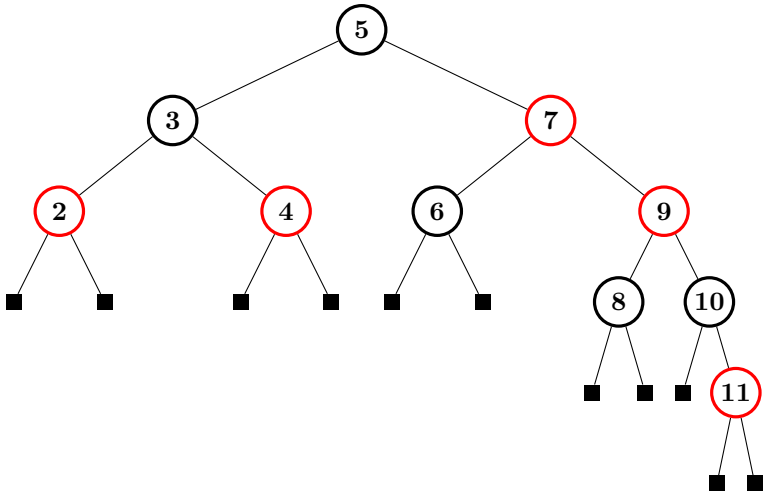


I ovenstående rød-sort træ indsættes 11 under brug af algoritmen fra lærebogen [1, s. 315].

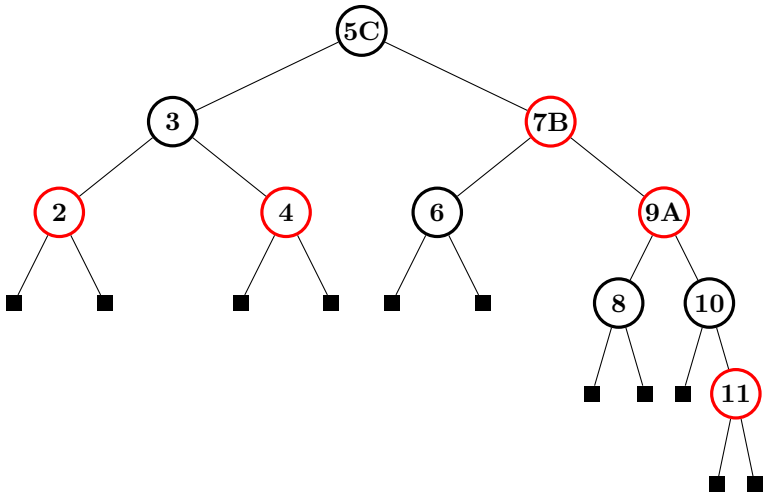
Først så indsættes 11 i træet og farves rødt. Da $11 > 5$, $11 > 7$, $11 > 9$ og $11 > 10$, så kommer 11 som det højre barn på knuden 10.



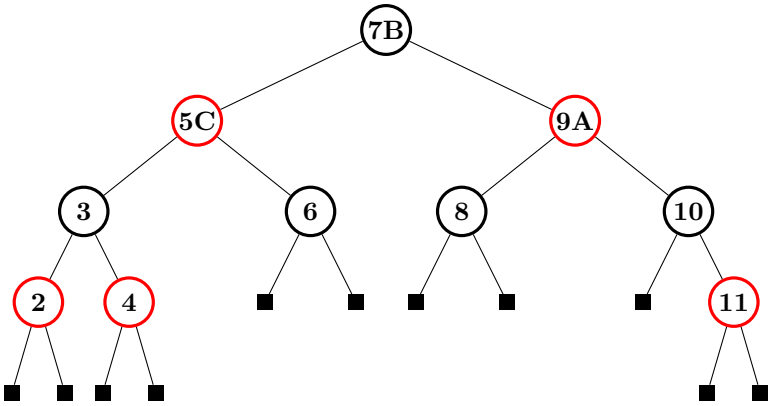
Da både 11 og 10 er røde, så opstår der et "to røde i streg"problem. Der kigges på Dictonaries slides [2, s. 25] for at undersøge hvilken en af de 3 "cases", der er opstået. Da 11's onkel, 8, er rød, så er det case 1. I case 1 farves 11's forælder og onkel sort, altså 10 og 8, og 11's bedsteforælder 9 farves rød.



Der opstår dog endnu et "to røde i træk"problem med 7 og 9, så igen kigges der på hvilken case, som er opstået. Da 9's onkel 3 er sort, så er det enten case 2 eller case 3. Det er case 3, da 7 og 9 er samme type barn, altså begge højre børn. Hvis de var forskellige typer børn, så ville der kigges på case 2. Knuderne A, B og C fra slides markeres,



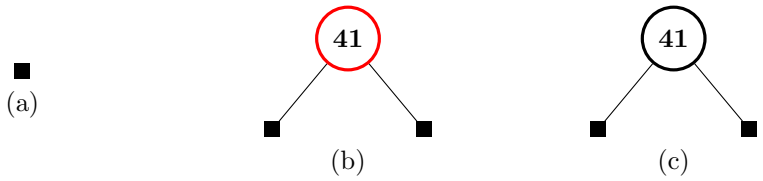
og træet roteres som beskrevet i slides (dvs. en venstre rotation på 5 (C)).



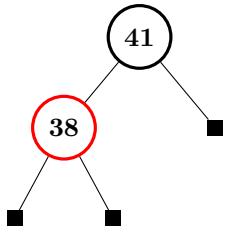
Så er træet i balance, da alle rød-sort træ-kravene er opfyldt. Opgaven spørger specifikt om hvilket træ, ud af 4 valgmuligheder, som opstår ved indsættelse. Dette vil være T_3.

2 Opgave 2 - Cormen et al. opgave 13.3-2

Vis rød-sort træerne der følger efter gentagne indsættelser af nøglerne 41, 38, 31, 12, 19, 8 i et rød-sort træ, som i starten er tomt.

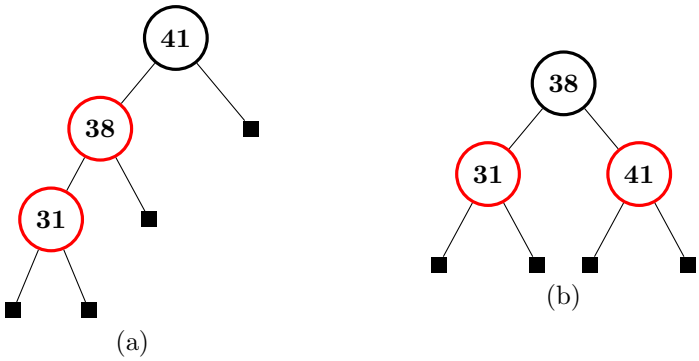


Figur 1: Indsættelse af nøgle 41: (a) I starten er rød-sort træet tomt. (b) Søg efter indsættelsesplacering for knude med nøgle 41, som overholder inorder krav. I dette tilfælde er det trivielt, da træet er tomt - der er kun en mulig placering. (c) Når en ny knude indsættes i et rød-sort træ, så farves den nye knude altid rød. Eneste mulige overtrædelse vil være to røde naboer og dette fikses vha. omfarvning og/eller rebalancering. Hvis man når roden, så farves den sort. I dette tilfælde starter vi ved roden, så knuden farves bare sort.



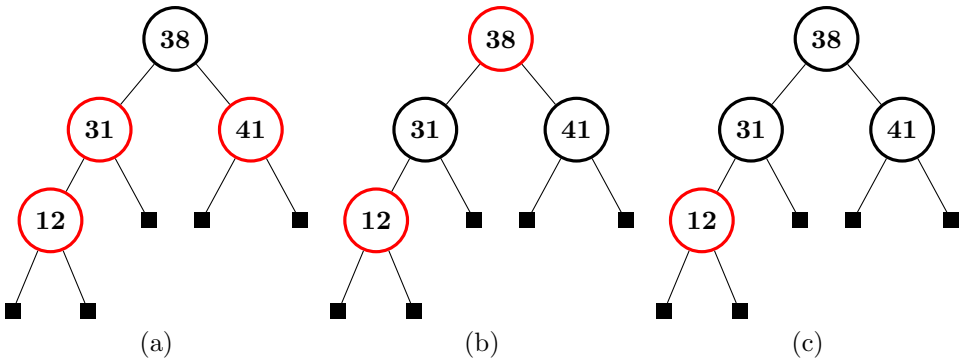
(a)

Figur 2: Indsættelse af nøgle 38: (a) Da 38 er mindre end 41, så skal den nye knude være i rodens venstre undertræ. Dette undertræ er et `NIL` undertræ, så indsættelsesplaceringen er fundet. Den nye knude farves rød, og da der ikke er to røde naboer, så er vi færdige.

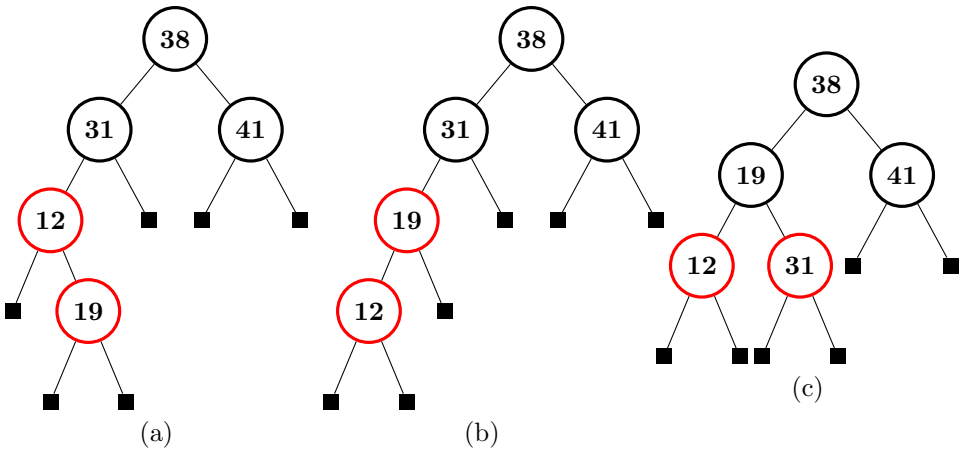


(a) (b)

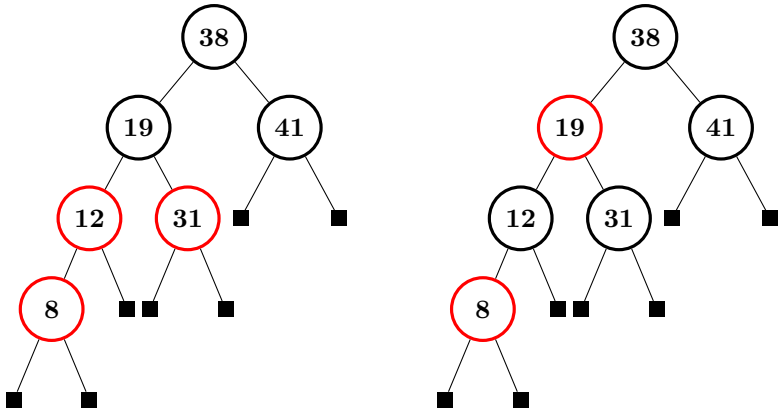
Figur 3: Indsættelse af nøgle 31: Da 31 er mindre end 41, så skal den nye knude være i rodens venstre undertræ. Da 31 også er mindre end 38, så skal den nye knude være i 38 knudens venstre undertræ, som er et `NIL` undertræ, og dermed er indsættelsespositionen fundet. (b) Da der er en rød-rød overtrædelse mellem 31 og 38 knuderne, så skal der foretages en rebalancering. Da 31 knuden er et venstre barn og dens onkel (rodens højre barn) er sort, så er vi i Case 3 (se [2, s. 25]). Dette indebærer en højre rotation omkring 41 knuden (roden) samt at 41 knuden farves rød og 38 knuden (ny rod) farves sort. Dette færdiggør rebalanceringsproceduren.



Figur 4: Indsættelse af nøgle 12: Da 12 er mindre end 38, så skal den nye knude være i rodens venstre undertræ. Da 12 er mindre end 31, så skal den nye knude være i 31 knudens venstre undertræ, som er et `NIL` undertræ, og dermed er indsættelsespositionen fundet. (b) Da der er en rød-rød overtrædelse mellem 12 og 31 knuderne, så skal der foretages en rebalancering/omfarvning. Da 12 knudens onkel (rodens højre barn) er rød, så er vi i Case 1 (se [2, s. 25]). Dette indebærer at 12 knudens forælder og onkel farves sorte og endvidere at deres forælder (altså 12 knudes bedsteforælder) farves rød. (c) Den nu røde 38 knude kan medvirke i en ny rød-rød overtrædelse, som skal fikses, men da dette er roden farves denne bare sort (dermed får alle rod-blad stiger én sort mere). Dette færdiggør rebalanceringsproceduren.



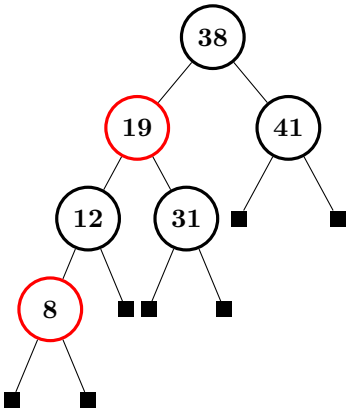
Figur 5: Indsættelse af nøgle 19: Da 19 er mindre end 38, så skal den nye knude være i rodens venstre undertræ. Da 19 er mindre end 31, så skal den nye knude være i 31 knudens venstre undertræ. Da 19 er større end 12, så skal den nye knude være i 12 knudens højre undertræ, som er et `NIL` undertræ, og dermed er indsættelsespositionen fundet. (b) Da der er en rød-rød overtrædelse mellem 12 og 19 knuderne, så skal der foretages en rebalancering. Da 19 knuden er et højre barn og dens onkel (31 knudes højre barn) er sort, så er vi i Case 2 (se [2, s. 25]). Dette indebærer en venstre rotation omkring 12 knuden. Rebalanceringsproceduren forsætter fra 12 knuden. (c) Oberserver fra [2, s. 25], at Case 2 altid efterfølgende leder til Case 3. Dette indebærer en højre rotation omkring 31 knuden samt at 31 knuden farves rød og 19 knuden farves sort.



Figur 6: Indsættelse af nøgle 8: Da 8 er mindre end 38, så skal den nye knude være i rodens venstre undertræ. Da 8 er mindre end 19, så skal den nye knude være i 19 knudens venstre undertræ. Da 8 er mindre end 12, så skal den nye knude være i 12 knudens venstre undertræ, som er et `NIL` undertræ, og dermed er indsættelsespositionen fundet. (b) Da 8 knudens onkel (19 knudens højre barn) er rød, så er vi i Case 1 (se [2, s. 25]). Dette indebærer at 8 knudens forælder og onkel farves sorte og endvidere at deres forælder (altså 8 knudes bedsteforælder) farves rød. Den nu røde 19 knude kan medvirke i en ny rød-rød overtrædelse, men det gør den ikke, så dermed er rebalanceringsproceduren færdig.

3 Opgave 3 - Cormen et al. opgave 13.4-3

Vis rød-sort træet der følger efter at slette knuderne med følgende nøgler: 8, 12, 19, 31, 38 og 41 (i den række følge) fra træet vi fandt frem til i opgave 2 (Corment et al opgave 13.3-2).



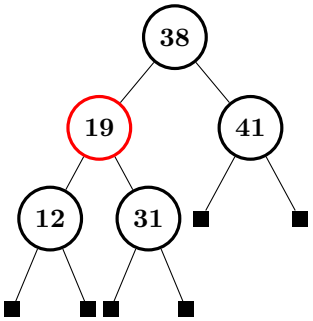
Figur 7: Det endelige træ fra opgave 2

Den generelle strategi for at slette en ny knude i et rødsort træ er følgende:

1. Først slettes knuden næsten som i et binært søgetræ
2. Herefter fjernes evt. opstået ubalance (overtrædelser af strukturkravene for rød-sorte træer)

Dette kan findes på slides 26-30 i [2]. Til at slette i et rød-sort træ bruges `RB-Transplant`, `RB-Delete` og `RB-Delete-Fixup` fra siderne 323-326 i [1]. `RB-transplant` er mere eller mindre den samme som fra binære søgetræer. `RB-Delete` benytter også de samme principper som ved binære søgetræer, men gemmer mere information for at kunne rette op på overtrædelser af kravene for et rød-sort træ efter at knuden er slettet. `RB-Delete-Fixup` er den metode der bruges til at rette op på eventuelle overtrædelser. På slides 26-30 i [2] er fokus på ideen bag `RB-Delete-Fixup`.

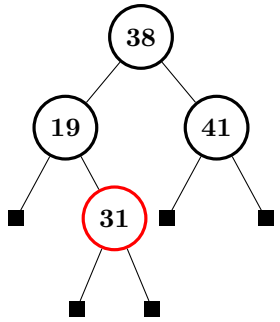
Først skal knuden med nøglen 8 slettes. Kigger man på `RB-Delete` på side 324 i [1], kan man se at der ligesom ved et binært søgetræ vi flytter bladet (som er en reference til `T.nil`) op på 8's plads med `RB-Transplant` (som kan findes på side 323). Da knuden var rød er træet stadig et gyldigt rød-sort træ og vi kalder derfor ikke `RB-Delete-Fixup`.



Figur 8: Træet efter at knuden med nøglen 8 er blevet slettet

Herefter skal knuden med nøglen 12 slettes. Her sker samme procedure som ved knuden med nøglen 8, da 12 heller ikke har nogen børn. Men da knuden med nøglen 12 var sort kaldes `RB-Delete-Fixup` med x = `T.nil` på linje 22 (i `RB-Delete`). I `RB-Delete-Fixup` går vi ind i `while-løkken`, da x ikke er træets rod og dens farve er sort (alle blade er sorte), og forsætter ind i første `if-statement` da x er venstre barn til knuden med nøglen 19. Herefter går vi ind i `if-statmentet` på linje 9 siden at knuden med nøglen 31 har to sorte børn, og her ændrer vi 31's farve til rød, Til sidst ændres x til at være knuden med nøglen 19. Så rammer vi bunden af `while-løkken`, og da knuden med nøglen 19 ikke er sort går vi ud af `while-løkken`. Efterfølgende ændres 19's farve til at være sort i linje 23 og sletningen er færdig.

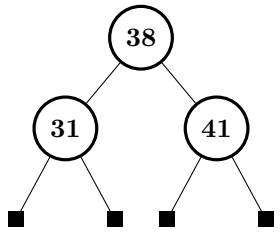
Kigger man på slide 30 i [2] er det oprykkede blad blevet sværtet (og tæller derfor som 2 sorte). Vi er i case 2, da bladet er sort og dens søskend (knuden med nøglen 31) har 2 sorte børn. Som man kan se på slide 30 ændrer vi x til at være det oprykkede blads forældre (dvs. knuden med nøglen 19 bliver den nye sværtet knude) og d = 31 til at være rød. Nu er vi i et af de nemme stoptilfælde (slide 28 i [2]), da 19 er rød, og vi kan genoprette balancen i træet ved at farve denne sort.



Figur 9: Træet efter at knuden med nøglen 12 er blevet slettet

Nu skal 19 slettes, og vi ender med at gøre det samme som i de foregående tilfælde bortset fra vi skal transplantere 31 op på 19's plads i stedet for `T.nil`. I det her tilfælde skal vi også kalde `RB-Delete-Fixup` med x = 31 som argument, men da 31 er rød ender vi bare med at gøre denne sort.

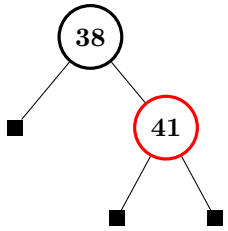
Kigger man på slides bliver den oprykkede knude med nøglen 31 til den sværtede knude og da 31 er rød er vi i et af de nemme stop tilfælde hvor vi bare kan gøre 31 sort.



Figur 10: Træet efter at knuden med nøglen 19 er blevet slettet

Med kun 3 knuder tilbage skal vi nu slette 31 og vi kommer til at gøre det samme som da vi skulle slette 12. Det vil sige flytte `T.nil` op på 31's plads og herefter gøre 41 rød som følge af case 2 i `RB-Delete-Fixup` (se kommentarer til højre i koden på sidde 326 i [1]).

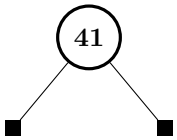
Med fokus på slide 30 i [2] er vi i case 2, da det oprykkede blad (som er sværtet) er sort og dens søskend 41 er sort med 2 sorte børn. Derfor gør vi d = 41 rød og lader 38 blive den sværtede knude, men da 38 er roden er vi i et af de nemme stoptilfælde og vi kan derfor bare fjerne sværtningen.



Figur 11: Træet efter at knuden med nøglen 31 er blevet slettet

Med kun 2 knuder tilbage skal vi slette roden med nøglen 38. Proceduren er det samme som ved de andre knuder; vi går ind i den første `if-statement` i `RB-delete` (linje 3) og transplantere 41 op som den nye rod. Eftersom den gamle rod var sort (som roden skal være i et rød-sort træ) kalder vi `RB-Delete-Fixup` med knuden med nøglen 41 som argument. Da knuden er rød hopper vi over `while-løkken` og farver knuden sort inden vi er færdige.

I forhold til [2] bliver 41 den sværtede knude efter transplantationen. Her kan vi bare gøre 41 sort, da den er rød, og fjerne sværtningen (et af de nemme stoptilfælde).



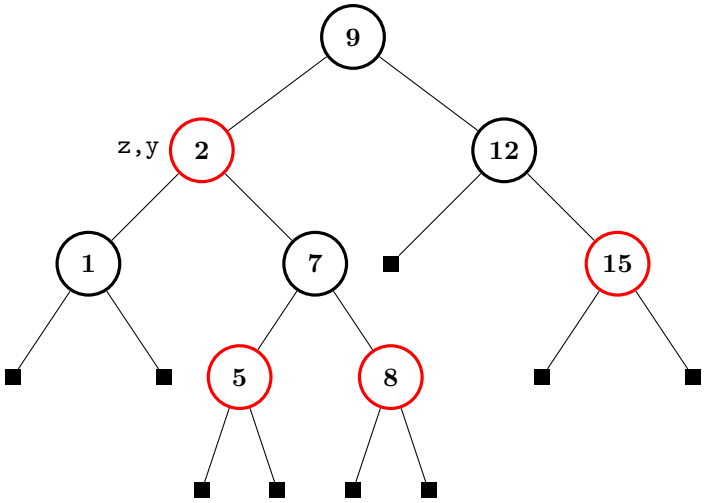
Figur 12: Træet efter at knuden med nøglen 38 er blevet slettet

Til sidst skal roden fjernes og dens venstre barn, `T.nil`, bliver transplanteret op som den nye rod. `RB-Delete-Fixup` bliver kaldt, men ændrer ikke noget, da `T.nil` er roden i træet og allerede har farven sort.

4 Opgave 4 - Eksamen juni 2009, opgave 1b

Vi ser på rød sort træet nedenfor og hvad der sker når vi sletter knuden med nøgle 2.

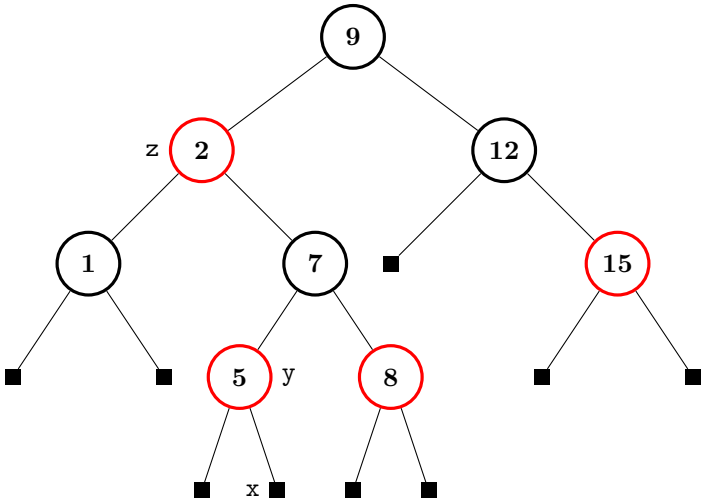
Vi kan bruge pseudokoden for *RB-Delete* til at slette knuden med nøgle 2 og stadig opretholde alle reglerne for Rød-Sort træer (findes på side 324 i [1]). I gennemgangen nedenfor når der refereres til et linje-tal så er det til pseudokoden i bogen.



Figur 13: Rød-Sort træet inden sletning

Som input til *RB-Delete* giver vi træet fra figur 13 kaldet T i pseudokoden og knuden med nøglen 2, kaldet z i pseudokoden.

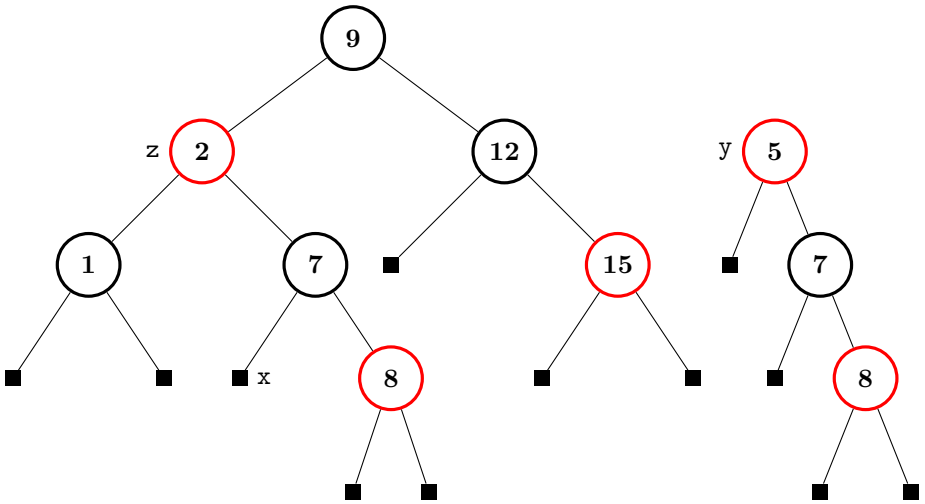
Skridt 1: Vi går forbi `if-statement` på linje 3 og `elseif-statement` på linje 6 da sandt-falsk udsagnet i disse giver falsk (knuden med nøgle 2 har både højre og venstre barn). Derefter ender vi i `else-statement` på linje 9 og sætter y = *Tree-Minimum(z.right)*, vi finder altså successor til knuden z. y er nu knuden med nøgle 5. x er højre barn af y, altså `NIL`.



Figur 14: Rød-Sort træet efter skridt 1

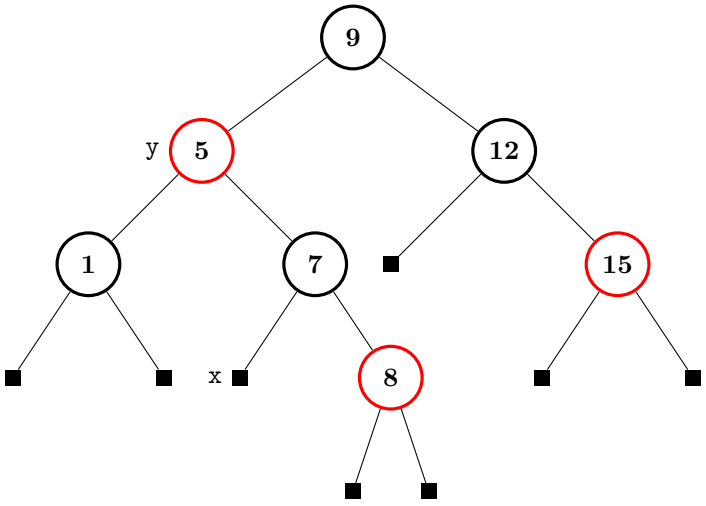
Skridt 2: Vi går forbi `if-statement` på linje 12 da y ikke er direkte barn af z. Vi ender i `else-statement` på linje 14 og kalder *RB-Transplant* for at indsætte en pointer til `NIL` på

pladsen hvor knuden med nøgle 7 før pegede på y. Derefter laver vi y's højre barn om til z's højre barn og retter parent pointer.



Figur 15: Rød-Sort træet efter skridt 2 både 5 og 2 referere til 7 som deres højre barn, men 7 peger kun på 5 med sin parent pointer. Selvom det ser sådan ud er der ikke to versioner af knuderne 7 og 8, det er bare referencer til samme element

Skridt 3: Vi starter nu på linje 17 og kalder *RB-Transplant* som sætter y ind på z's plads og får roden til at pege på y. Herefter sættes y's venstre barn til at være z's højre barn. Parent pointere ændres også hvor knuder får ny forældre.



Figur 16: Rød-Sort træet efter skridt 3

Skridt 4: Vi kontrollerer hvilken farve den knude vi har slettet havde, linje 21. Da den ikke

var sort gør vi ingenting og vi er færdige med sletning af knuden med nøgle 2.

5 Opgave 5 - Eksamen juni 2016, opgave 4

Vi er givet en hashtabel:

0	1	2	3	4	5	6	7	8	9	10
67	20	17		33		16	2			15

som vi skal indsætte to nøgler i. Vi skal bruge hashfunktionen

$$h'(x) = (7x + 4) \bmod 11$$

Og hvis to nøgler kommer til at få den samme hashværdi (dvs. begge prøver på at blive indsat på den samme plads), så skal vi bruge `linear probing` / `linear hashing` for at løse problemet, dvs. vi skal i princippet bare gå videre gennem hashtabellen indtil vi finder en tom plads (og evt. gå tilbage til starten hvis vi rammer slutningen af hashtabellen).

En anden måde at se denne metode på er ved at definere en ny hashfunktion $h(x, i)$, som bruger vores gamle hashfunktion $h'(x)$ som en `auxiliary hash function`. Her er x stadig den nøgle vi vil have en hashværdi for, og i er det "forsøg" vi er igang med. Til at starte med er $i = 0$, men hvis den hashværdi vi får allerede er optaget i hashtabellen prøver vi med $i = 1$, $i = 2$ osv. m er tabellens størrelse:

$$h(x, i) = (h'(x) + i) \bmod m = ((7x + 4) \bmod 11 + i) \bmod 11$$

Vi prøver nu at indsætte den første nøgle, 18. Da det er første forsøg på at indsætte 18, har vi at $i = 0$:

$$h(18, 0) = ((7 \cdot 18 + 4) \bmod 11 + 0) \bmod 11 = 9$$

Vi kan se, at plads 9 i hashtabellen ikke er optaget, så her bliver 18 indsat uden problemer:

0	1	2	3	4	5	6	7	8	9	10
67	20	17		33		16	2		18	15

Vi skal nu indsætte 26. Vi prøver først med $i = 0$:

$$h(26, 0) = ((7 \cdot 26 + 4) \bmod 11 + 0) \bmod 11 = 10$$

Vi kan se, at vores første forsøg på at indsætte 26 går galt, da plads 10 allerede er optaget. Man kan nok allerede nu se, at fordi det er `linear hashing` vi bruger, så vil det næste

forsøg være 0 (da 10 var den sidste plads), så 1, så 2, og først når vi når til plads 3 er der en ledig plads, så her vil 26 blive indsat. For komplethedens skyld, lad os også se de udregninger som der bliver lavet (vi prøver med $i = 1$, $i = 2$, $i = 3$, $i = 4$):

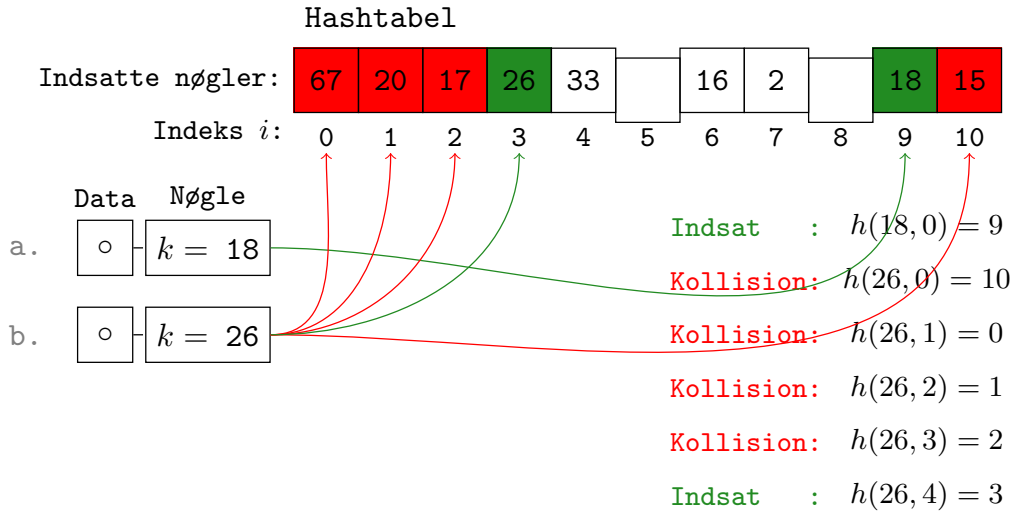
$$h(26,1) = ((7 \cdot 26 + 4) \bmod 11 + 1) \bmod 11 = 0$$
$$h(26,2) = ((7 \cdot 26 + 4) \bmod 11 + 2) \bmod 11 = 1$$
$$h(26,3) = ((7 \cdot 26 + 4) \bmod 11 + 3) \bmod 11 = 2$$
$$h(26,4) = ((7 \cdot 26 + 4) \bmod 11 + 4) \bmod 11 = 3$$

Efter indsættelse vil vores hashtabel se således ud:

0	1	2	3	4	5	6	7	8	9	10
67	20	17	26	33		16	2		18	15

og er dermed også det endelige svar.

En figur der viser alt dette ske på én gang findes nedenfor:

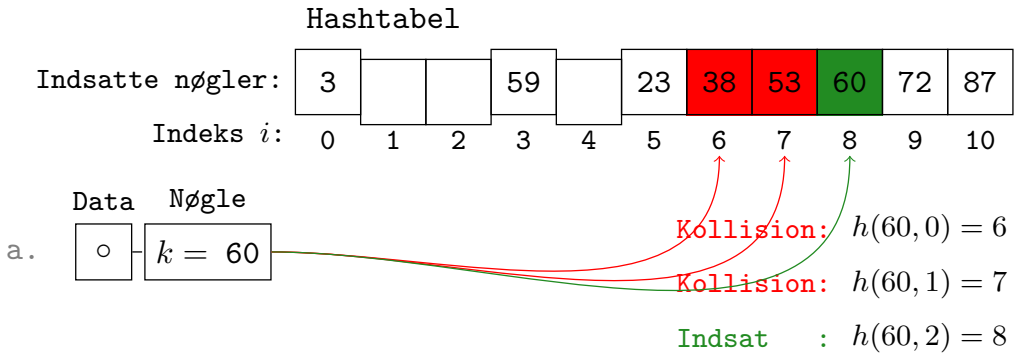


6 Opgave 6 - Eksamen juni 2013, opgave 3b

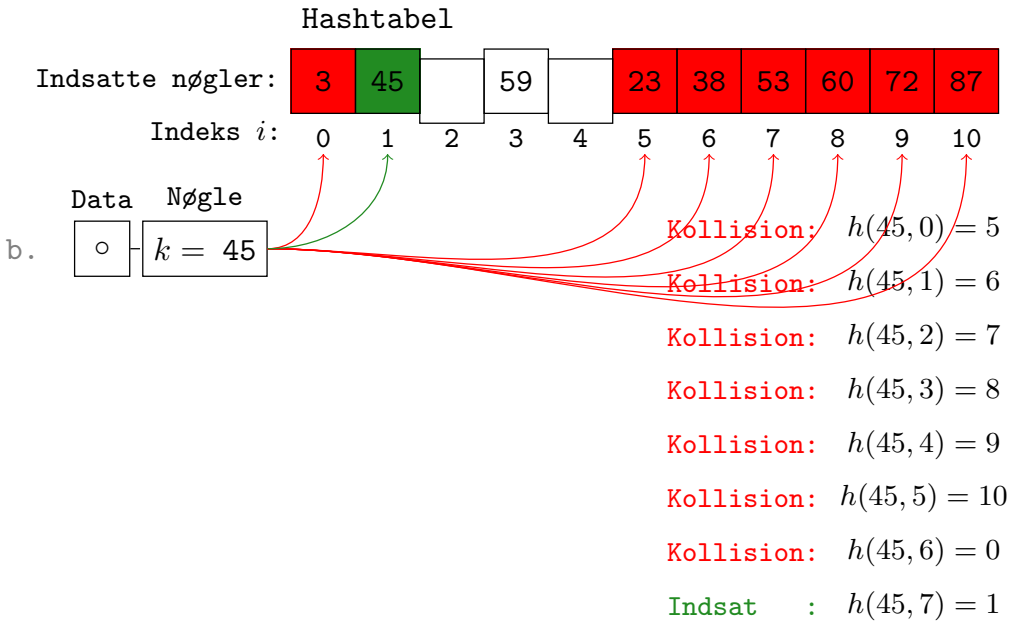
I følgende opgave er vi givet nøglerne 60 og 45 som skal indsættes i en hashtabel af størrelse $m = 11$. Til dette formål anvender vi *open adressing* med *linear probing* til at indsætte nøglerne i den givne hashtabel (Figur 17 før indsættelse af nøglen $k = 60$). Dette betyder at vi beregner indeks for indsættelse af nøgler i en hashtabel vha. følgende *hashfunktion*:

$$h(k,i) = (h'(k) + i) \bmod m,$$

med $h'(k) = (3k+2) \bmod m$. Her er $h'(k)$ vores givne *auxilary hashfunktion*, k er nøglen der indsættes og i er et indeks der øges for hver *kollision* med en anden nøgle i hashtabellen. Ved indsættelse af nøglerne $k = 60$ og $k = 45$ i den givne hashtabel har vi følgende forløb illustreret i Figur 17-18.



Figur 17: Skridt a. Indsættelse af nøglen $k = 60$.



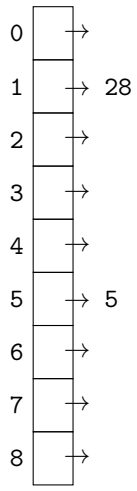
Figur 18: Skridt b. Indsættelse af nøglen $k = 45$.

7 Opgave 7 - Cormen et al. opgave 11.2-2

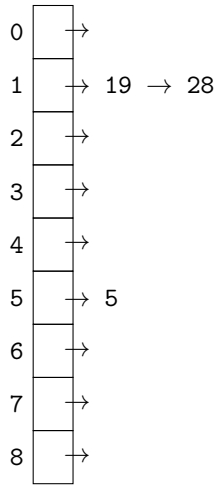
Vis hvad der sker, når nøglerne $5, 28, 19, 15, 20, 33, 12, 17$ og 10 indsættes i en hashtabel med chaining ved kollision. Tabellen har 9 pladser, og bruger hashfunktion $h(k) = k \bmod 9$.

I chaining ved kollison, så indeholder hver plads ikke et element, men derimod indeholder den en liste af elementer. Hver gang et element rammer en plads, så bliver det element sat forrest i listen.[2, p. 35]

Først indsættes 5. Da $h(5) = 5 \bmod 9 = 5$, så indsættes 5 i 5-pladsens liste.
Derefter indsættes 28. Da $h(28) = 28 \bmod 9 = 1$, så indsættes 28 i 1-pladsens liste.
På nuværende tidspunkt ser hashtabellen således ud:

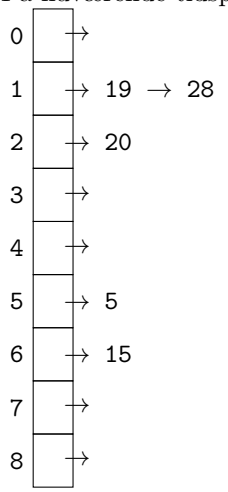


Derefter indsættes 19. h(19) = 19 mod 9 = 1. Da 28 allerede er i 1-pladsens liste, så bliver 19 sat forrest i 1-pladsens liste.

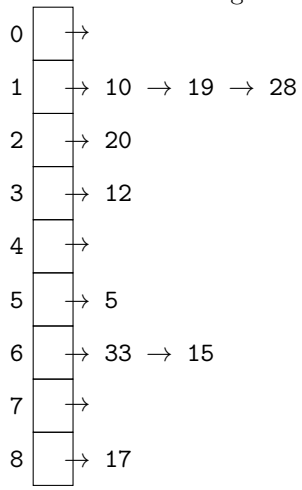


Derefter indsættes 15. Da h(15) = 15 mod 9 = 6, så indsættes 15 i 6-pladsens liste.
20 indsættes. Da h(20) = 20 mod 9 = 2, så indsættes 20 i 2-pladsens liste.

På nuværende tidspunkt ser hash tabellen således ud:



33 indsættes. $h(33) = 33 \bmod 9 = 6$. Da 15 allerede er i 6-pladsens liste, så bliver 33 sat forrest i 6-pladsens liste.
12 indsættes. Da $h(12) = 12 \bmod 9 = 3$, så indsættes 12 i 3-pladsens liste.
17 indsættes. Da $h(17) = 17 \bmod 9 = 8$, så indsættes 17 i 8-pladsens liste.
10 indsættes. $h(10) = 10 \bmod 9 = 1$. Da 19 og 28 allerede er i 1-pladsens liste, så bliver 10 sat forrest i 1-pladsens liste.
Altså ser den endelige hashtabel således ud:



8 Opgave 8 - Cormen et al. opgave 11.4-1

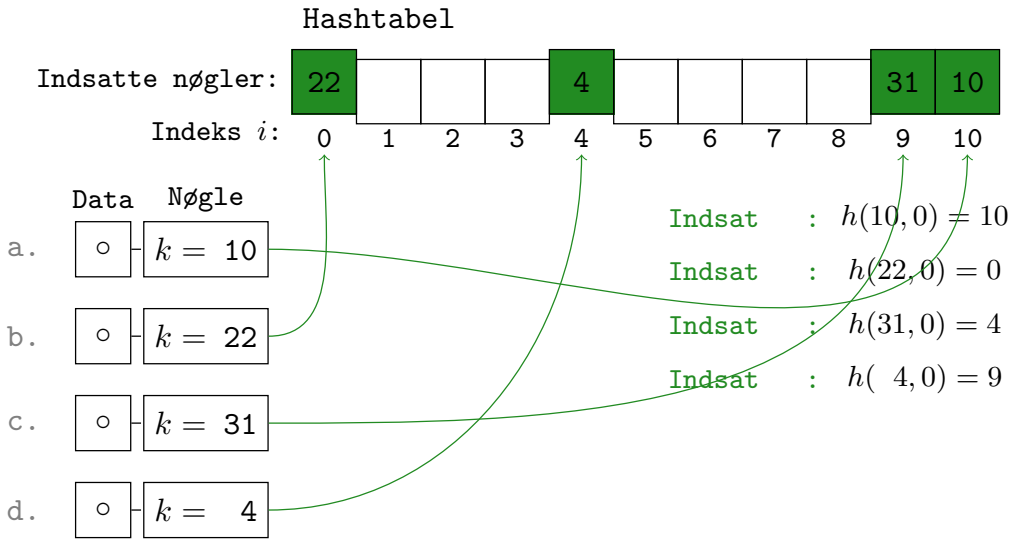
I følgende opgave er vi givet nøglerne $10, 22, 31, 4, 15, 28, 17, 88, 59$ som skal indsættes i en hashtabel af størrelse $m = 11$. Til dette formål anvender vi *open adressing* med henholdsvis *linear probing*, *quadratic probing* og *double hashing* til at indsætte nøglerne i en tom hashtabel. Anvendelsen af hver af disse metoder er beskrevet i de følgende delsektioner.

8.1 Delspørgsmål: Linear Probing

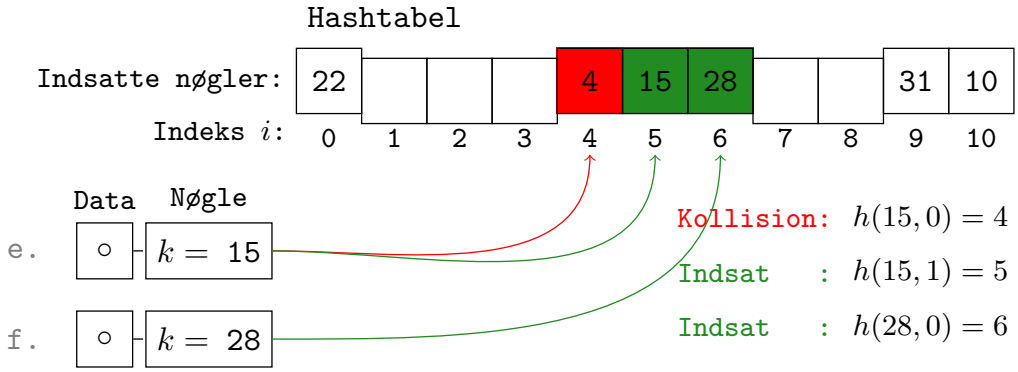
Ved brug af *linear probing* beregner vi indeks for indsættelse af nøgler i en hashtabel vha. følgende *hashfunktion*:

$$h(k, i) = (h'(k) + i) \bmod m,$$

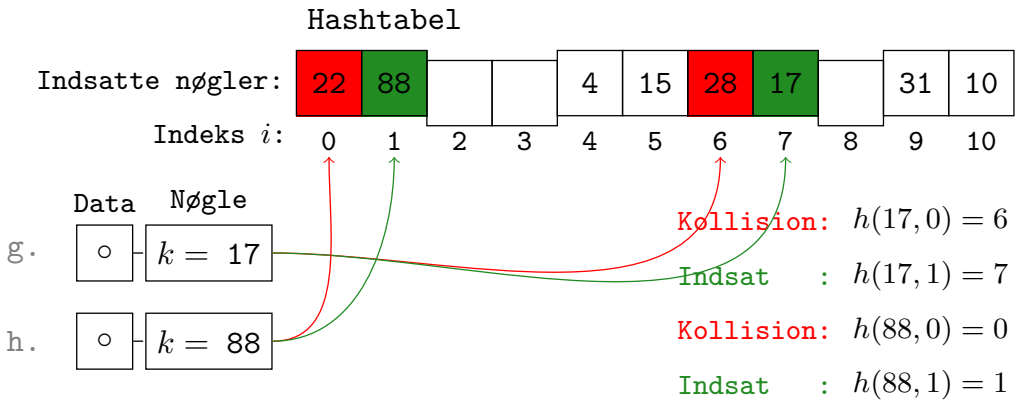
med $h'(k) = k \bmod m$. Her er $h'(k)$ vores givne *auxilary hashfunktion*, k er nøglen der indsættes og i er et indeks der øges for hver *kollision* med en anden nøgle i hashtabellen. Ved indsættelse af nøglerne har vi følgende forløb illustreret i Figur 19-22.



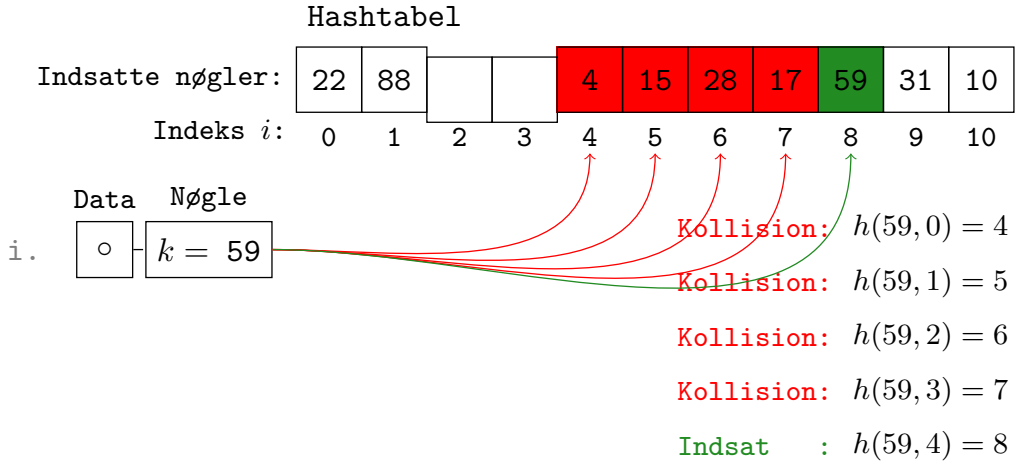
Figur 19: Skridt a-d. Indsættelse af nøglerne $k = 10, 22, 31, 4$.



Figur 20: Skridt e-f. Indsættelse af nøglerne $k = 15, 28$.



Figur 21: Skridt g-h. Indsættelse af nøglerne $k = 17, 88$.



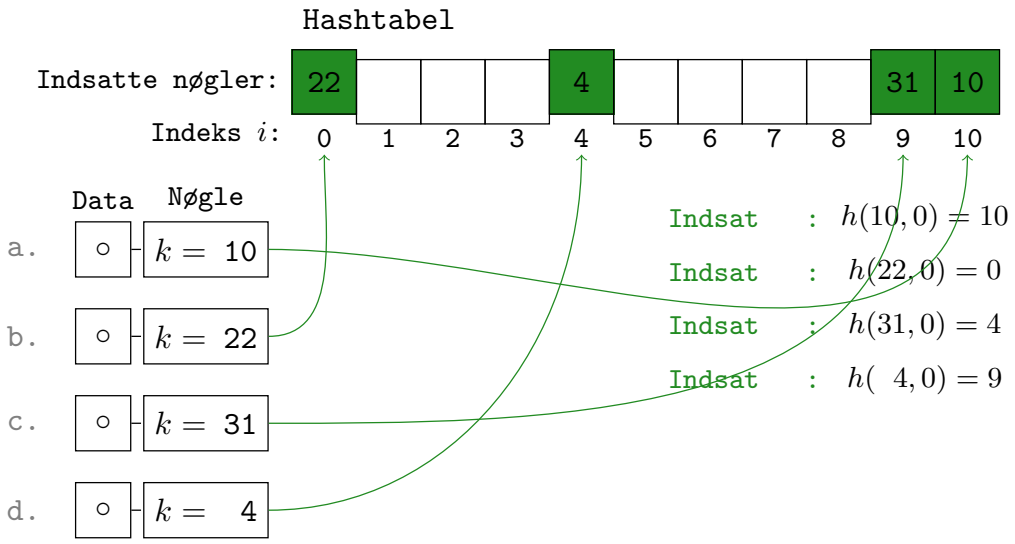
Figur 22: Skridt i. Indsættelse af nøglen $k = 59$.

8.2 Delspørgsmål: Quadratic Probing

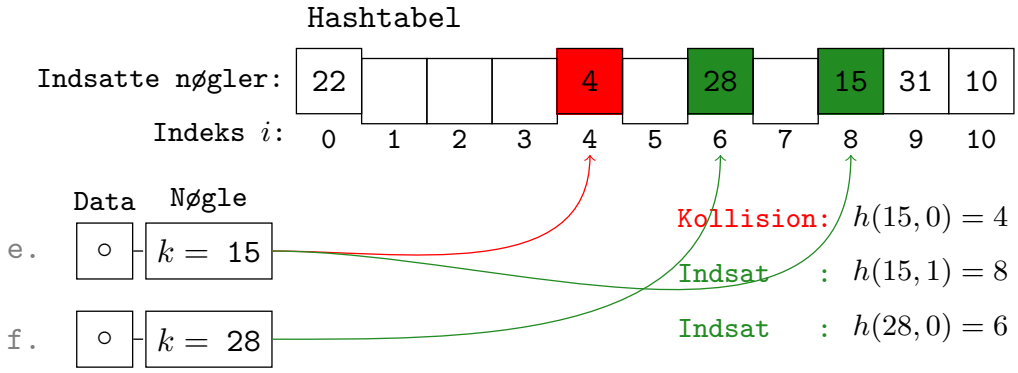
Ved brug af *quadratic probing* beregner vi indeks for indsættelse af nøgler i en hashtabel vha. følgende *hashfunktion*:

$$h(k, i) = (h'(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod m,$$

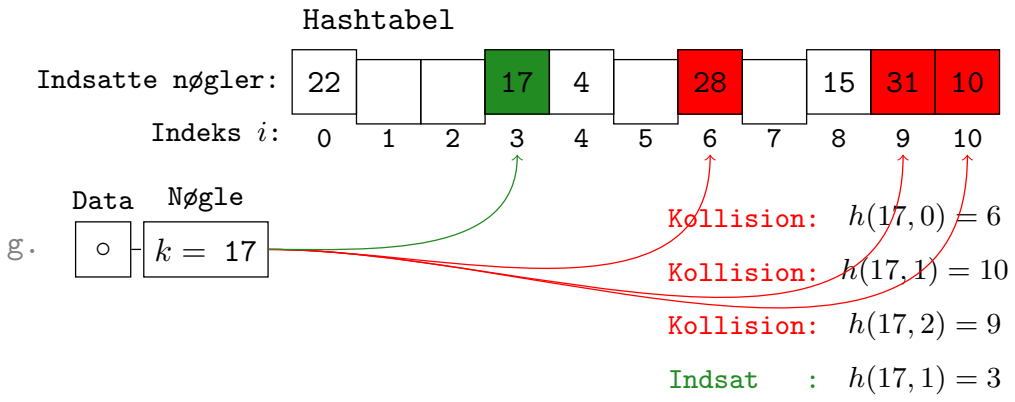
med $h'(k) = k \bmod m$, $c_1 = 1$ og $c_2 = 3$. Her er $h'(k)$ vores givne *auxilary hashfunktion*, k er nøglen der indsættes og i er et indeks der øges for hver *kollision* med en anden nøgle i hashtabellen. Ved indsættelse af nøglerne har vi følgende forløb illustreret i Figur 23-27.



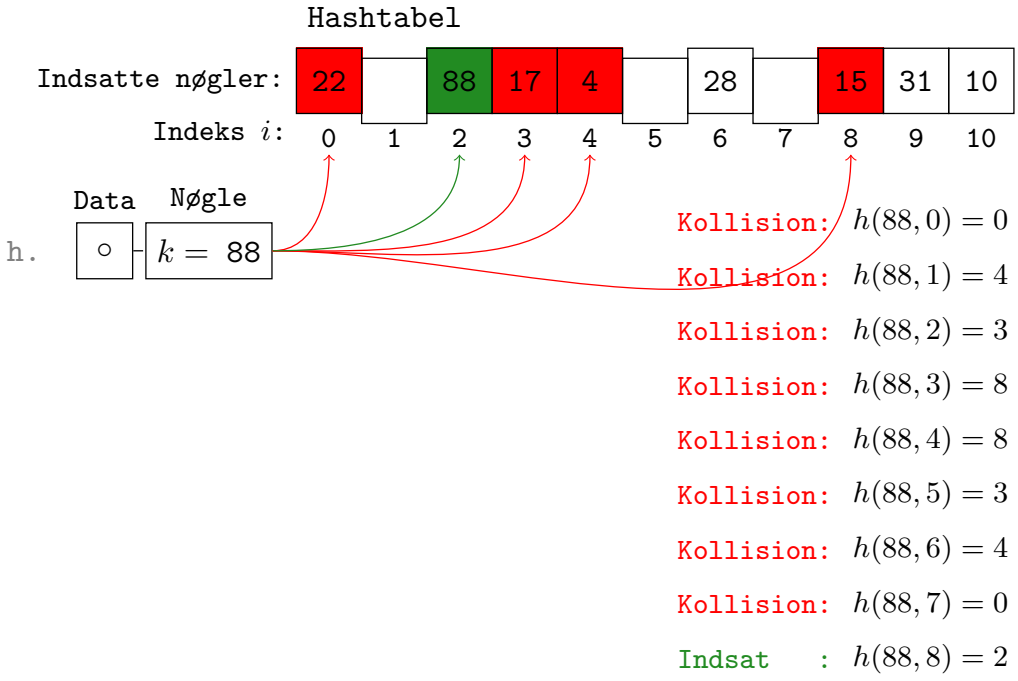
Figur 23: Skridt a-d. Indsættelse af nøglerne $k = 10, 22, 31, 4$.



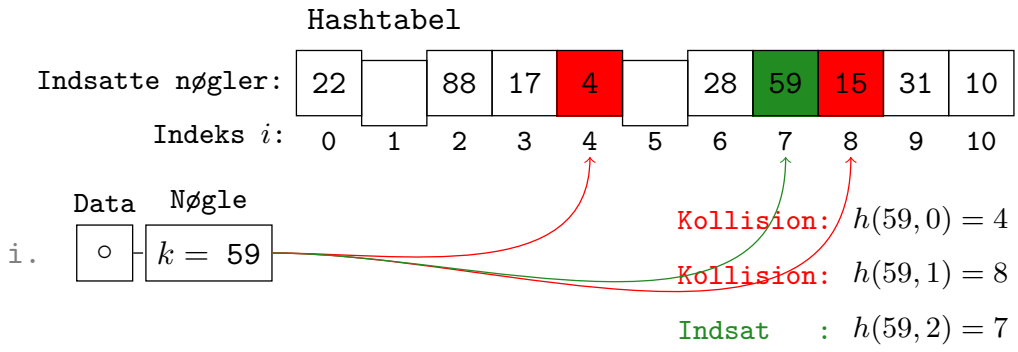
Figur 24: Skridt e-f. Indsættelse af nøglerne $k = 15, 28$.



Figur 25: Skridt g. Indsættelse af nøglen $k = 17$.



Figur 26: Skridt h: Indsættelse af nøglen $k = 88$.



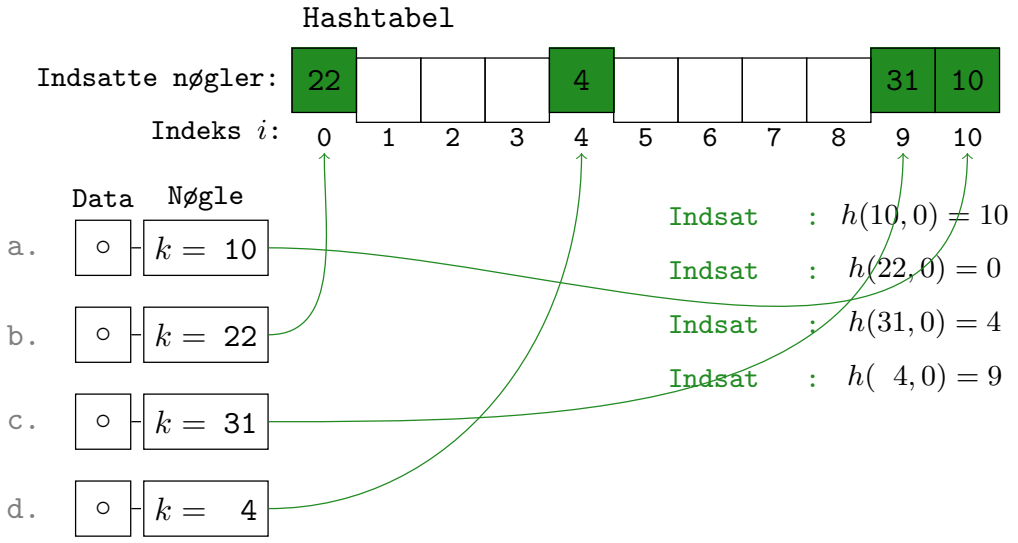
Figur 27: Skridt i. Indsættelse af nøglen $k = 59$.

8.3 Delspørgsmål: Double Hashing

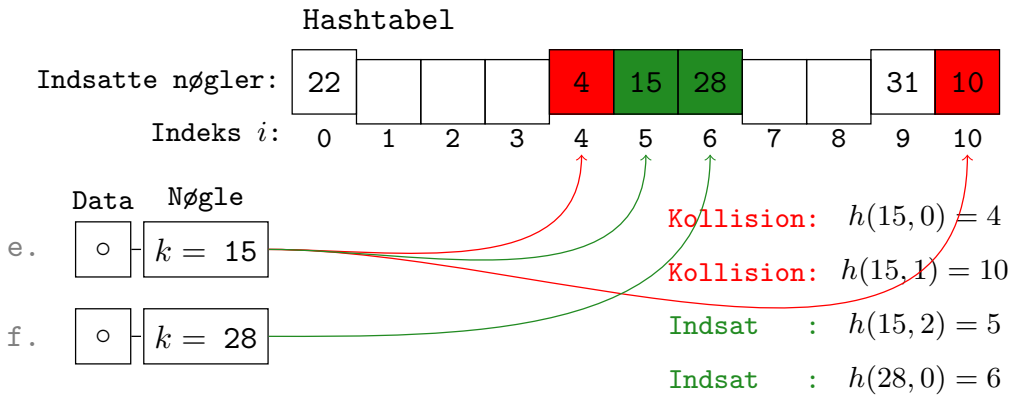
Ved brug af *double hashing* beregner vi indeks for indsættelse af nøgler i en hashtabel vha. følgende *hashfunktion*:

$$h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m,$$

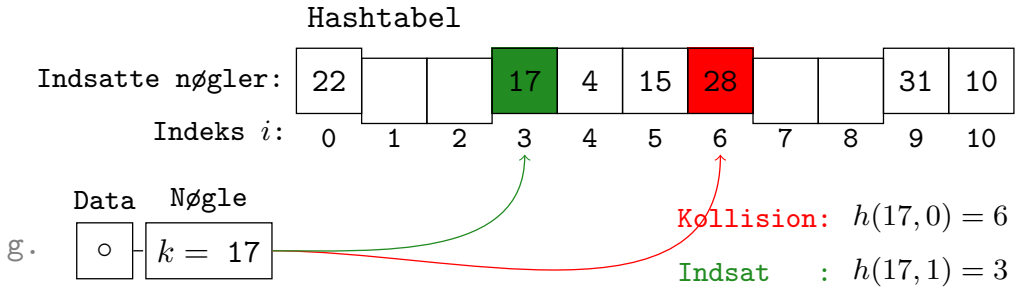
med $h_1(k) = k \bmod m$ og $h_2(k) = 1 + (k \bmod (m-1))$. Her er $h_1(k)$ og $h_2(k)$ vores givne *auxilary hashfunktioner*, k er nøglen der indsættes og i er et indeks der øges for hver *kollision* med en anden nøgle i hashtabellen. Før vi fortsætter bemærker vi at når $i = 0$, dvs. der er ingen kollisioner har været, så reducerer hashfunktionen til følgende: $h(k, 0) = (h_1(k) + 0 \cdot h_2(k)) \bmod m = h_1(k) \bmod m$. Med andre ord, kun $h_1(k)$ har indflydelse på det beregnede indeks, når der ikke har været nogen kollisioner med andre nøgler. Ved indsættelse af nøglerne har vi følgende forløb illustreret i Figur 28-31.



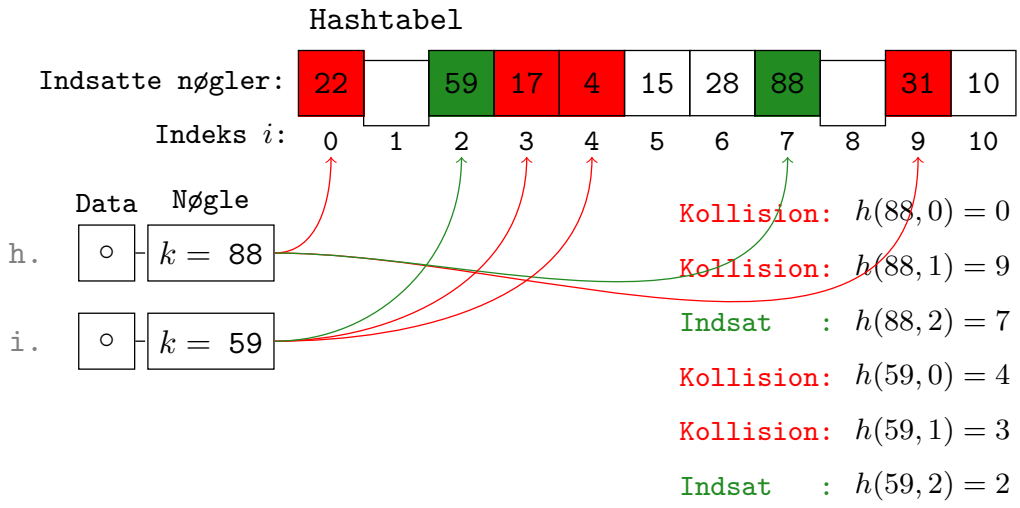
Figur 28: Skridt a-d. Indsættelse af nøglerne $k = 10, 22, 31, 4$.



Figur 29: Skridt e-f. Indsættelse af nøglerne $k = 15, 28$.



Figur 30: Skridt g. Indsættelse af nøglen $k = 17$.



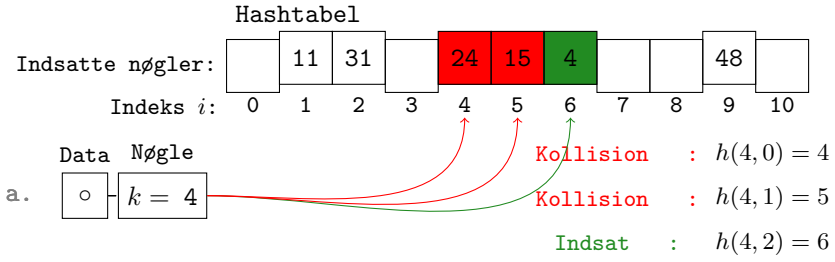
Figur 31: Skridt h-i. Indsættelse af nøglerne $k = 88, 59$.

9 Opgave 9 - Eksamen januar 2008, opgave 1c

I denne opgave får vi givet en hashtabel med 10 pladser, samt følgende hashfunktionen:

$$h(k) = k \bmod 10$$

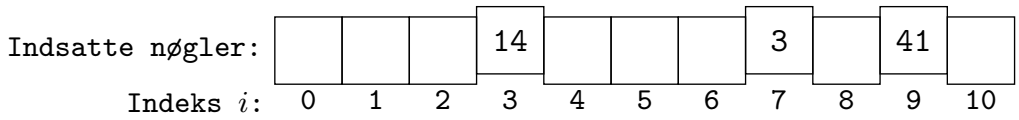
Vi skal med denne hashfunktion og linear probing prøve at indsætte et nyt element med nøglen 4. For at finde det nye elements plads udregner vi først $h(4) = 4$ og prøver at indsætte det nye element på denne plads, men der står allerede et element med nøglen 24. Det er her linear probing kommer i spil, hvor vi linært prøver de næste pladser i hashtabellen indtil en fri plads er fundet. Det vil sige, at vi nu prøver at indsætte på plads 5, men der opstår en kollision eftersom at pladsen ikke er fri. Vi forsætter med linear probing til plads 6. Da plads 6 er frit indsætter vi det nye element her og den færdige hashtabel kan ses sammen med en illustration af processen i figuren nedeunder.



Figur 32: Hashtabellen med det nye element sat ind

10 Opgave 10 - Eksamen januar 2006, opgave 1a

Vi bliver givet hashtabellen i figur 33 og bliver bedt om at indsætte et element med nøglen 18, via open adressering og double hashing (info om double hashing kan bl.a. findes på side 272 i [1]).



Figur 33: Hashtabel inden indsætning

Hash-funktionen vi skal bruge er beskrevet ved:

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod 11$$

hvor

$$h_1(k) = k \bmod 11$$
$$h_2(k) = 1 + (k \bmod 10)$$

Skridt 1: Vi udregner

$$h(18,0) = (18 \bmod 11) + 0 \cdot (1 + (18 \bmod 10)) \bmod 11 = 7$$

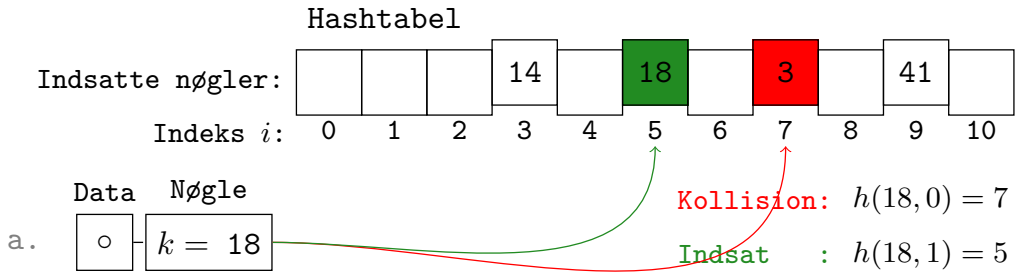
Vi forsøger at indsætte elementet med nøgle 18 på plads 7 og ser at vi får kollision med et element med nøglen 3.

Skridt 2: Vi udregner

$$h(18,1) = (18 \bmod 11) + 1 \cdot (1 + (18 \bmod 10)) \bmod 11 = 5$$

Vi indsætter element med nøglen 18 på plads 5.

Figur 34 viser dette forløb og resultatet.



Figur 34: Hashtabel efter indsætning

11 Opgave 11 - Eksamen juni 2009, opgave 1c

Vi er givet en hashtabel H:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32				20			23	7		39					

Som vi skal indsætte nøglen 71 i.

Hvis der opstår kollisioner (dvs. der allerede er en nøgle på pladsen vi prøver at indsætte på), skal vi lede efter en ny plads ved at bruge `kvadratisk probing` / `quadratic hashing`. Ved `quadratic hashing` bruger vi en hashfunktion $h(x, i)$, som bruger en `auxiliary hash function` $h'(x)$, og to konstanter, c_1 og c_2. Vi er givet `auxiliary hash function` $h'(x)$ som:

$$h'(x) = x \bmod 16$$

og de to konstanter som $c_1 = \frac{1}{2}$ og $c_2 = \frac{1}{2}$.

Vi kan nu definere vores hashfunktion $h(x, i)$ (her er m størrelsen af hashtabellen):

$$h(x,i) = (h'(x) + c_1 \cdot i + c_2 \cdot i^2) \bmod m = (x \bmod 16 + \frac{1}{2} \cdot i + \frac{1}{2} \cdot i^2) \bmod 16$$

Vi prøver nu at indsætte nøglen 71. Da det er første forsøg, er $i = 0$:

$$h(71, 0) = (71 \bmod 16 + \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0^2) \bmod 16 = 7$$

7 **er optaget**, så vi finder nu en ny hashværdi ved at sætte $i = 1$:

$$h(71, 1) = (71 \bmod 16 + \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1^2) \bmod 16 = 8$$

8 **er også optaget**. Vi prøver med $i = 2$:

$$h(71, 2) = (71 \bmod 16 + \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 2^2) \bmod 16 = 10$$

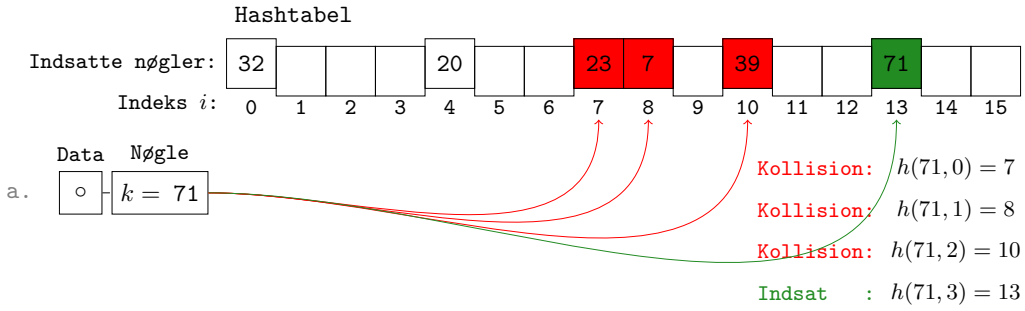
10 **er også optaget**. Vi prøver med $i = 3$:

$$h(71, 3) = (71 \bmod 16 + \frac{1}{2} \cdot 3 + \frac{1}{2} \cdot 3^2) \bmod 16 = 13$$

13 **er ikke optaget**, så her kan vi indsætte 71:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32				20			23	7		39			71		

En figur der viser alt dette ske på en gang findes nedenfor:

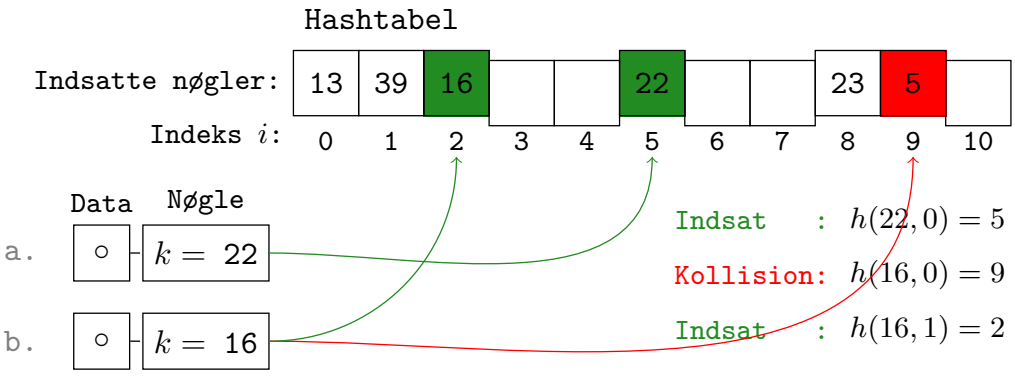


12 Opgave 12 - Eksamen juni 2015, opgave 4

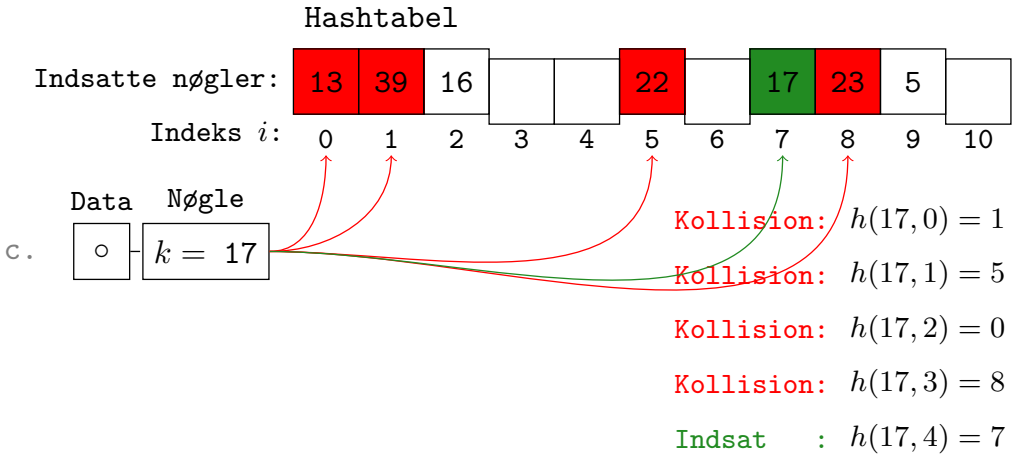
I følgende opgave er vi givet nøglerne $22, 16, 17$ som skal indsættes i en hashtabel af størrelse $m = 11$. Til dette formål anvender vi *open adressing* med *quadratic probing* til at indsætte nøglerne i den givne hashtabel (Figur 35 før indsættelse af nøglerne $k = 22, 16$). Dette betyder at vi beregner indeks for indsættelse af nøgler i en hashtabel vha. følgende *hashfunktion*:

$$h(k, i) = (h'(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod m,$$

med $h'(k) = (3k{+}5) \bmod m$, $c_1 = 3$ og $c_2 = 1$. Her er $h'(k)$ vores givne *auxilary hashfunktion*, k er nøglen der indsættes og i er et indeks der øges for hver *kollision* med en anden nøgle i hashtabellen. Ved indsættelse af nøglerne har vi følgende forløb illustreret i Figur 35-36.



Figur 35: Skridt a-b. Indsættelse af nøglerne $k = 22, 16$.



Figur 36: Skridt c. Indsættelse af nøglen $k = 17$.

Litteratur

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[2] Rolf Fagerberg. Dictionaries. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/dictionarySlides.pdf`, 2020.