Algoritmer og datastrukturer
Course No. 02105
Cheat Sheet 2012

May 15, 2012

Contents

1 Kompleksitet

1.1 Køretid

O : øvre grænse, denne skal altid være langsomst. Eksempel: $n^3 = O(n^4)$. I dette tilfælde er $n^4 langtsommere end n^3$.
Ω : nedre grænse, denne skal altid være hurtigst. Eksempel: $n^4 = \Omega(n^3)$. I dette tilfælde er $n^3 hurtigere end n^4$.
Θ : tæt grænse, denne skal altid være den samme som den du får opgivet. Eksempel: $n^3 = \Theta(n^3)$.
Generelt eksempel:

$$\mathrm{T(n)} = 7n^2 + 3n + 42$$
$$\mathrm{T(n)} = O(n^2), O(n^3), \Omega(n^2), \Omega(n), \Theta(n^2).$$
$$\mathrm{T(n)} \neq O(n), \Omega(n^3), \Theta(n), \Theta(n^3).$$

1.2 Asymptotisk vækst

Funktioner er listet med hurtigste funktioner først.

$$1 \to log(n) \to \sqrt{n} \to n \to n \times log(n) \to n^2 \to 2^n \to n!$$

1.3 Eksempel på køretid

Algorithm 1 Algo1(n)

```
for i = 1 to n do                          ▷ n
    for j = 1 to n do                      ▷ n
        print i + j                        ▷ konstant
    end for
end for
```

Køretiden er $T(n) = \Theta(n^2)$ da man ser bort fra konstanten.

Algorithm 2 Algo2(n)

```
if n > 0 then
    Algo2([n/2])                           ▷ log(n)
end if
```

Køretiden er $T(n) = \Theta(log(n))$ da det er et rekursivt kald.

Algorithm 3 Algo3(n)

```
i = 1
while i ≤ n do                             ▷ log(n)
    j = 1
    while j ≤ n do                         ▷ n
        j = j + 1
    end while
    i = 2i
end while
```

Køretiden er $T(n) = \Theta(n \times log(n))$ da det inderste while loop kører til n og det yderste bliver halveret da i bliver fordoblet, derfor $log(n)$ (kan sammenlignes med et rekursivt kald).

2 Grafer

2.1 BFS

- Første knude er i lag 0
- Kun alfabetisk hvis skrevet i opgaven
- Lag: distancen fra startknuden, typisk knude A
- Husk at følge pilene

Given graf G:

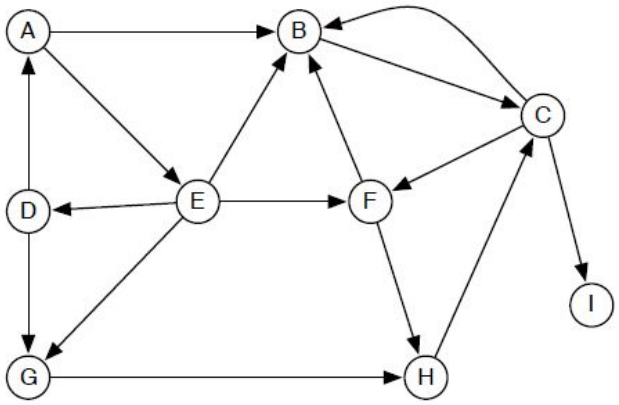

Figure 1: Grafen G

Løsning:

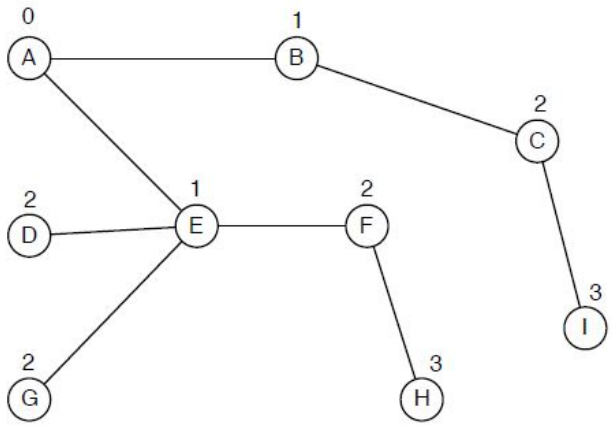

Figure 2: BFS løsning

2.1.1 Køretid

- Køretid: O(n + m)

2.2 DFS

- Første knude har vægt 1 samt slutvægt
- Køres alfabetisk
- Alle knuder har 2 vægte
- En knude besøges 2 gange i træk ved "dead end"
- Husk at returnere til start knuden

Given graf G:

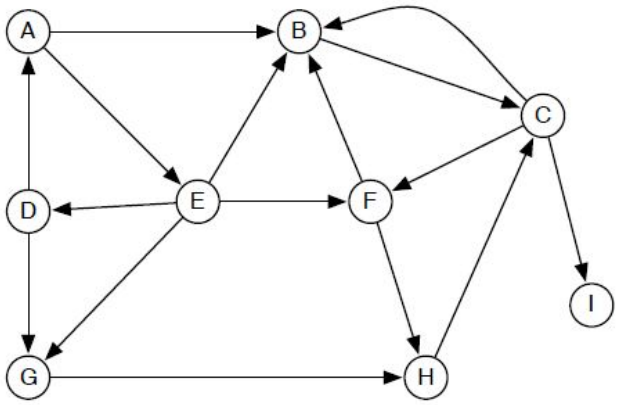

Figure 3: Grafen G

Løsning:

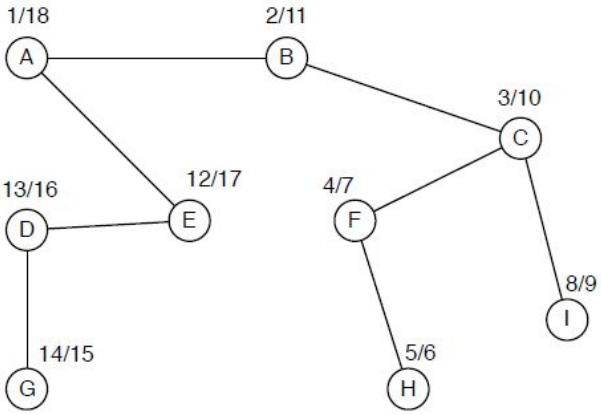

Figure 4: DFS løsning

2.2.1 Køretid

- Køretid: O(m+n))

2.3 Korteste vej

- Startknuden har altid afstand 0.
- Pilene skal følges i ruten, dog bruges pilene ikke i svaret.
- Bogstav rækkefølge benyttes ikke.
- Forsikre altid at den rute der vælges er den korteste til den givne knude.

Given graf:

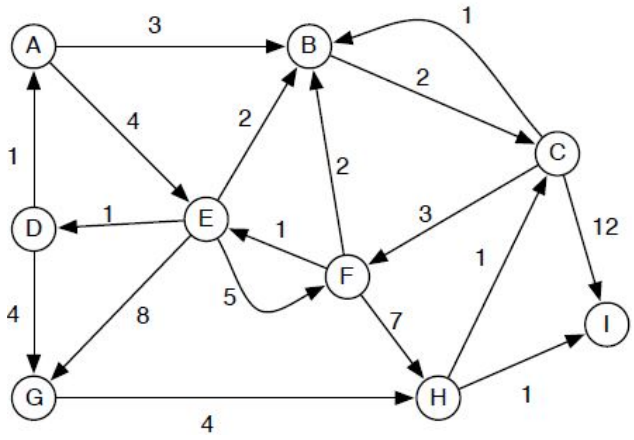

Figure 5: Kortest vej træ

Løsningen:

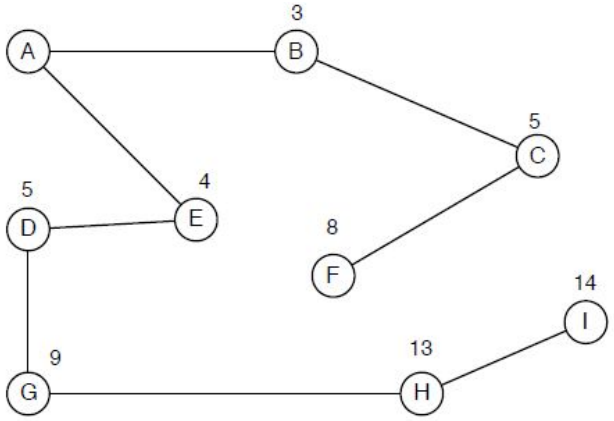

Figure 6: Kortest vej træ færdig

2.3.1 Køretid

- Kommer an på valg af datastruktur, se slide 12 om Korteste veje i grafer, hvor bl.a. Dijkstras algoritme er beskrevet.

2.4 Topologisk sortering (DAG)

- Ingen pile må pege på startknude
- Skal kunne sorteres så alle pile peger fremad.
- Må ikke indeholde sammenhængskomponenter

Given graf:

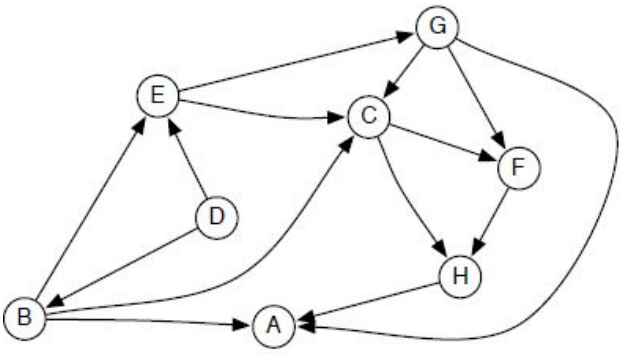

Figure 7: Given graf

Løsning: D, B, E, G, C, F, H, A
Bevis, ingen sammenhængskomponenter:

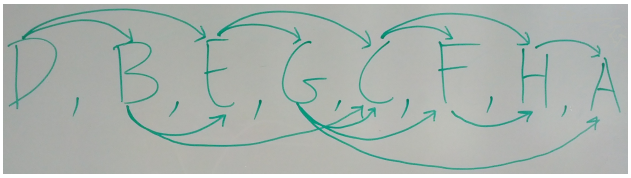

Figure 8: Bevis

2.4.1 Køretid

- Kommer an på valg af datastruktur, se slide 7 om DFS og topologisk sortering.

2.5 Sammenhængskomponenter

- Knuder hvor der kan køres i ring (kreds)
- Hvis en knude står alene er det også en kreds

Løsning: A, B, D, I, H | E, F, J | G

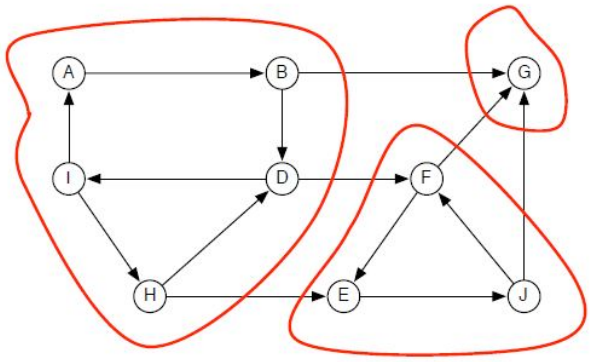

Figure 9: Sammenhængskomponenter

3 Hobe og binære søgetræer

3.1 Hobe

3.1.1 MinHob

- Ved insert indsættes knuden på næste plads og der bobles op
- Vigtigt at hob ordenen overholdes

Graf hvor 1 bliver indsat på den ledige plads:

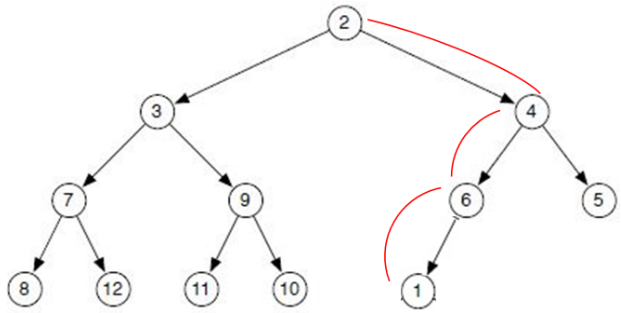

Figure 10: minHob

Graf hvor knude 1 er blevet boblet op:

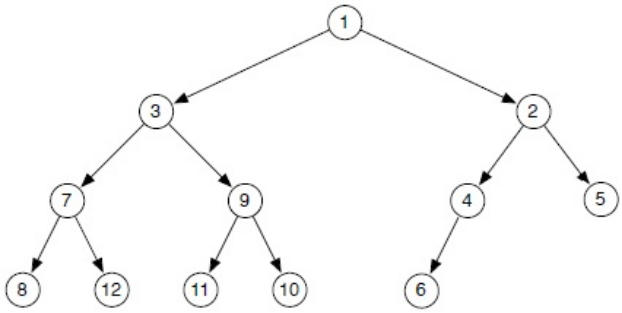

Figure 11: minHob indsættelse af 1

- Ved delete slettes den ønskede knude, og erstates med den sidste knude i grafen.
- Vigtigt at hob ordenen overholdes
- Hvis hob ordenen ikke er overholdt efter udskiftning af knuder, 'boble' op eller ned.
- Ved ExtractMin udtrækker man den øverste knude i træet.
- Knuden udskiftes med den sidste knude i træet, så hob ordenen overholdes.
- Da denne knude højst sandsynligt er større end dens børn, skal denne knude bobles ned med den mindste af knudens børn.

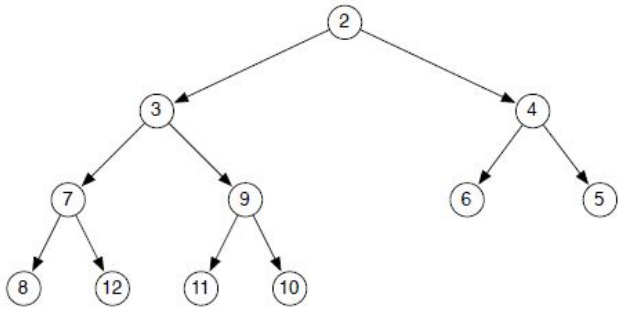

Figure 12: minHob extract 2

Løsning:

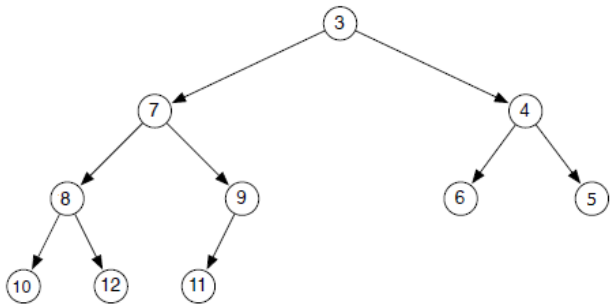

Figure 13: minHob nøgle 2 extracted

3.1.2 MaxHob

Samme operationer som MinHob:

- Ved insert indsættes knuden på næste plads og der bobles op
- Vigtigt at hob ordenen overholdes
- Ved delete slettes den ønskede knude, og erstates med den sidste knude i grafen
- Vigtigt at hob ordenen overholdes
- Hvis hob ordenen ikke er overholdt efter udskiftning af knuder, 'boble' op eller ned
- Ved ExtractMax, samme fremgangsmetode for ExtractMin

3.2 Binære søgetræ

3.2.1 Insert

- Venstre del træ er alle mindre end roden
- Højre del træ er alle større end roden.

Given binært træ:

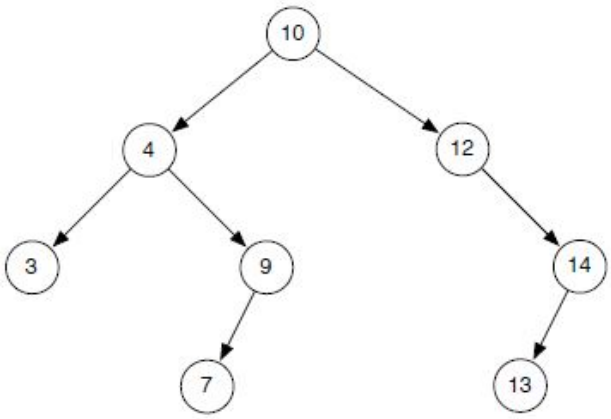

Figure 14: Binært søgetræ

Indsæt key 8:
Løsning: $8 < 10$, $8 > 4$, $8 < 9$, $8 > 7 \rightarrow$ indsæt 8 som 7's højre barn:

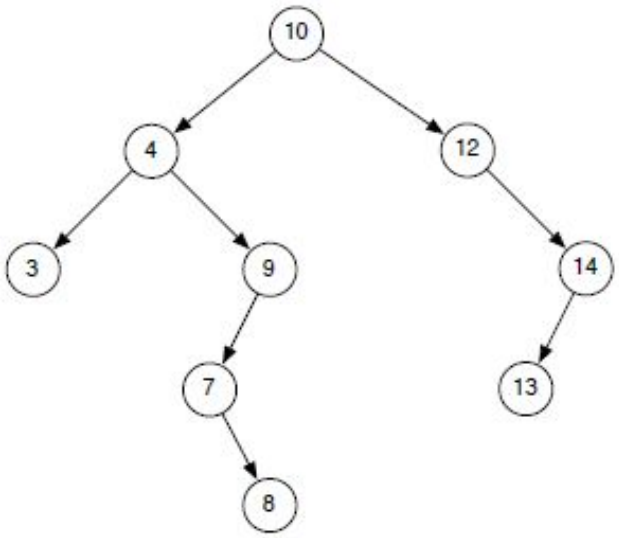

Figure 15: Binært søgetræ insert 8

3.2.2 Delete

- Nul børn: Slet knude v
- Ét barn: split v ud (slet v og ryk barn op)

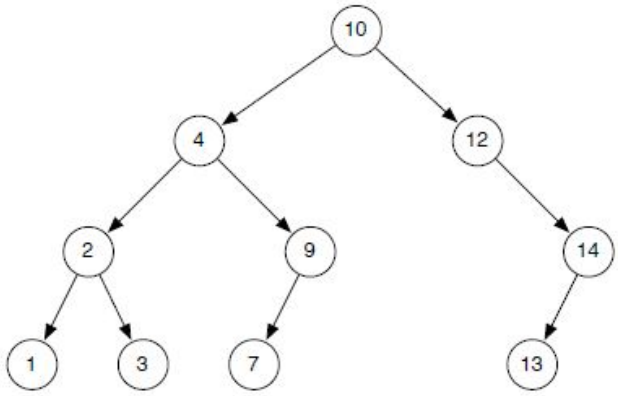

Figure 16: Binært søgetræ

- To børn: Brug successor til knuden v. Successor: Mindste knude $\geq$ v.

Slet knude 4:
Løsning: Knude 4 har to børn. Erstat knude 4 med knudens successor.

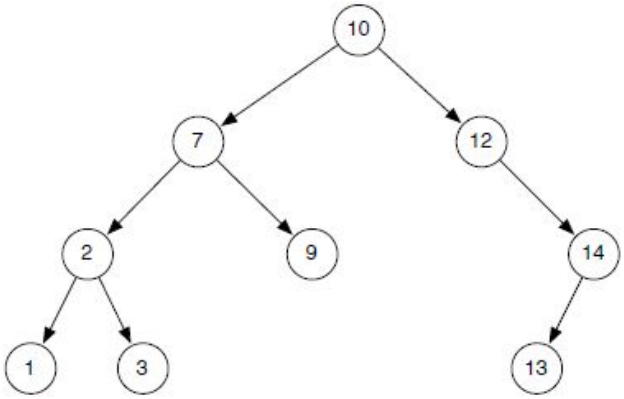

Figure 17: Binært søgetræ delete 4

3.2.3 Pre-order

- Besøg knude
- Besøg rekursivt venstre deltræ
- Besøg rekursivt højre deltræ
- Gå træet igennem til venstre først:

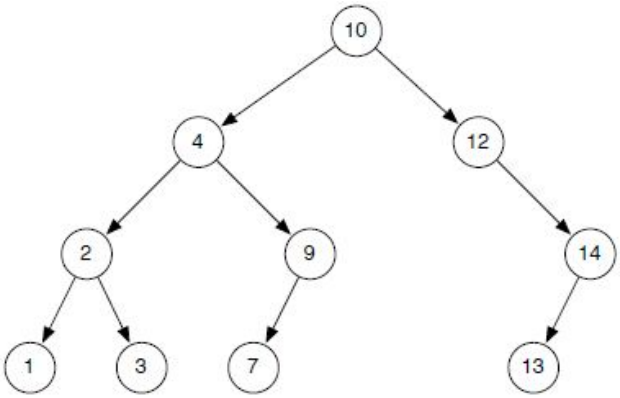

Figure 18: Binært søgetræ

Pre-order gennemløb: 10, 4, 2, 1, 3, 9, 7, 12, 14, 13

3.2.4 Post-order

- Besøg rekursivt venstre deltræ
- Besøg rekursivt højre deltræ
- Besøg knude

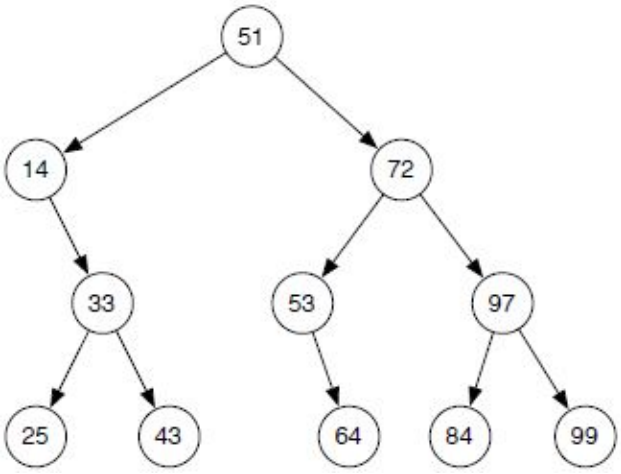

Figure 19: Binært søgetræ

Post-order gennemløb: 25, 43, 33, 14, 64, 53, 84, 99, 97, 72, 51

4 Datastrukturer

4.1 Stakke

4.1.1 Push(S,i), Pop(S)

Push betyder at man ligger bogstavet på stakken og pop fjerner det igen. En stack er en FIFO (First In First Out) struktur hvilket betyder at man fjerne det element man lagde på sidst.

Ifølgende eksempel betyder et bogstav i, PUSH(S,i) og * betyder POP(S).

D*TU**IN*FOR*M*ATIK

Løsning: DUTNRM

4.2 Binært søgetræ

4.2.1 Operationer, køretid

- Insert, search, delete, max, min, predecessor, successor: $\Theta(h)$
- Binært søgetræ: h = n
- Balanceret Binært søgetræt: h = log(n)

4.3 Hobe

4.3.1 Operationer, køretid

- Maximum: O(1)
- Insert: O(1) + tid for at 'boble op' = O(log(n))
- ExtractMin/Max: O(1) + tid for at 'boble op' = O(log(n))
- Delete: O(log(n))
- Increase-Key: O(log(n))

4.4 Hashing

4.4.1 Operationer, køretid

Seperat hægtning:

- Hash: Afbild nøgle til heltal mellem 0 og n-1: O(1)
- Insert: Indsæt i starten af den i'te liste: O(1), *O(n) hvis vi skal checke om det allerede findes*
- Delete: Slet fra i'te liste: O(n_i), n_i = *elementer i den i'te liste*
- Søg: Behøver kun at søge i den i'te liste: O(n_i)

4.5 Incidenslister

4.5.1 Pladsforbrug, køretid

- Knuder: n, Kanter: m
- Pladsforbrug: $\Theta(n+m)$
- Degree??
- $\Theta(m+n)$ tid for at identificere alle kanter.

4.6 Incidensmatrix

4.6.1 Pladsforbrug, køretid

- Pladsforbrug: $\Theta(n^2)$
- $\Theta(1)$ tid for at checke om (u, v) er en kant.
- $\Theta(n^2)$ tid for at identificere alle kanter.