Danmarks Tekniske Universitet

Skriftlig prøve, den 21. maj 2010.

Kursusnavn Algoritmer og datastrukturer I Kursus nr. 02105.

Tilladte hjælpemidler: Alle skriftlige hjælpemidler.

Vægtning af opgaverne: Opgave 1 - 24%, Opgave 2 - 16%, Opgave 3 - 20%, Opgave 4 - 20 %, Opgave 5 - 20 %.

Vægtningen er kun en cirka vægtning.

Alle opgaver besvares ved at udfylde de indrettede felter nedenfor. Som opgavebesvarelse afleveres blot denne og de efterfølgende sider i udfyldt stand. Hvis der opstår pladsmangel kan man eventuelt benyttes ekstra papir som så vedlægges opgavebesvarelsen.

Opgave 1 (kompleksitet)

1.1 Angiv for hver af nedenstående udsagn om de er korrekte:

	Ja	*Nej*
$\frac{1}{2}n^5 = O(n^3)$	☐	☒
$n(\log n)^2 = O(n^2)$	☒	☐
$\frac{1}{2}n^5 = O(n^6)$	☒	☐
$2^n + n^2 = \Omega(n^2)$	☒	☐
$n^2(n-1)/5 = \Theta(n^2)$	☐	☒

1.2 Skriv følgende liste af funktioner op i voksende rækkefølge efter asymptotisk vækst. Dvs. hvis funktionen $g(n)$ følger umiddelbart efter funktionen $f(n)$ i din liste, så skal der gælde at $f(n) = O(g(n))$.

$n^2 \log n$

$4000 \log n$

$\frac{1}{3}n^{5/2} + n$

$\frac{2}{3}n^2$

$4\sqrt{n}$

Svar: $4000 \log n$ $4\sqrt{n}$ $\frac{2}{3}n^2$ $n^2 \log n$ $\frac{1}{3}n^{5/2} + n$

1.3 Lad A_1 være en algoritme, der har køretid $O(n^2)$, og A_2 en algoritme, der har køretid $O(n^3)$. Lad A_3 være en algoritme der først kalder A_1 og dernæst A_2 og ellers intet gør. Hvad er køretiden af A_3?

☐ A $O(n)$ ☐ B $O(n^2)$ ☒ X $O(n^3)$ ☐ D $O(n^5)$

1.4 Betragt nedenstående algoritme.

Algorithm 1 Løkke1(n)

```
1: for i = 1 to n do
2:   for j = 1 to n do
3:     print i + j
4:   end for
5: end for
```

a) Køretiden af algoritmen er

☐ A $\Theta(\log n)$ ☐ B $\Theta(n)$ ☐ C $\Theta(n \log n)$ ☐ D $\Theta(n^2 \log n)$ ☐ E $\Theta(n^3)$

☒ X $\Theta(n^2)$ ☐ G $\Theta(2^n)$ ☐ H $\Theta(n^4)$ ☐ I $\Theta(\sqrt{n})$

b) Hvis linie 2 ændres til "**for** $j = 1$ **to** i" så bliver køretiden:

[A] asymptotisk langsommere

[X] asymptotisk den samme

[C] asymptotisk hurtigere

1.5 Betragt nedenstående algoritme. $\lfloor n/2 \rfloor$ betyder $n/2$ rundet ned til nærmeste heltal.

Algorithm 2 Løkke2(n)

```
1: if (n > 0) then
2:     Løkke2(⌊n/2⌋)
3: end if
```

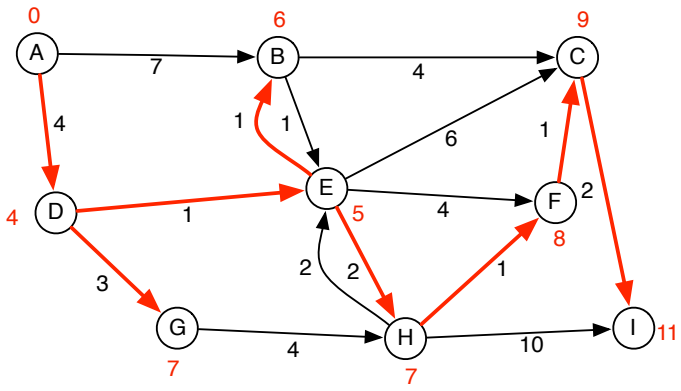
Køretiden af algoritmen er

[X] $\Theta(\log n)$ [B] $\Theta(n)$ [C] $\Theta(n \log n)$ [D] $\Theta(n^2 \log n)$ [E] $\Theta(n^3)$

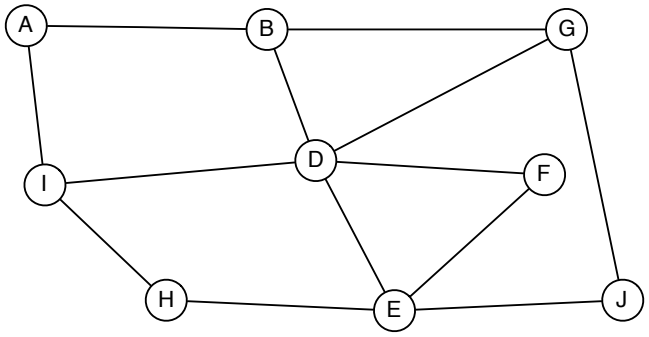
[F] $\Theta(n^2)$ [G] $\Theta(2^n)$ [H] $\Theta(n^4)$ [I] $\Theta(\sqrt{n})$

Opgave 2 (grafer)

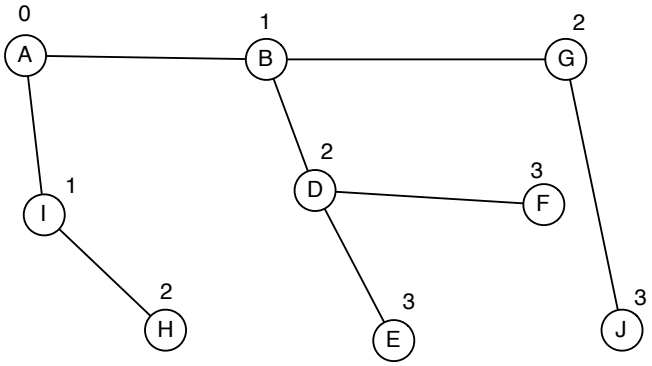
2.1 Angiv et korteste veje træ for nedenstående graf når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



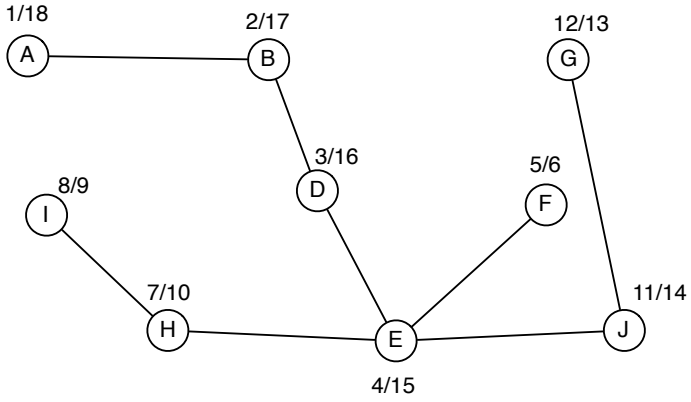
2.2 Betragt nedenstående graf G.



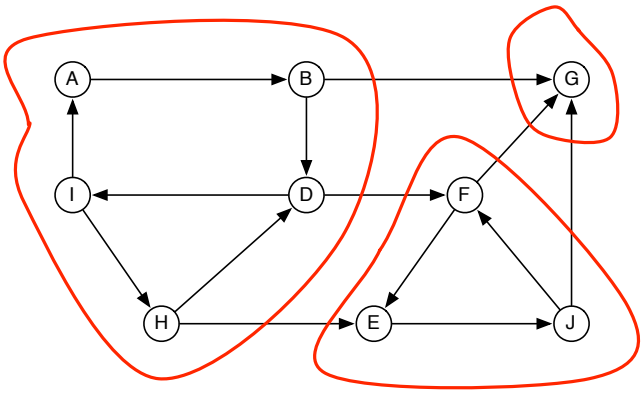
a) Angiv et BFS træ for grafen G når BFS genemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



b) Angiv et DFS træ for grafen G, når DFS genemløbet starter i knuden A. Angiv en DFS nummerering af knuderne (en DFS nummerering er den rækkefølge knuder bliver besøgt i). Det antages at incidenslisterne er sorteret i alfabetisk orden.



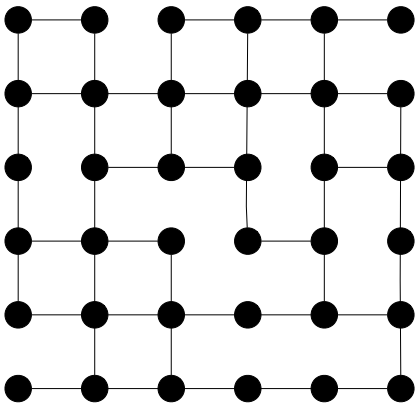
2.3 Angiv de stærke sammenhængskomponenter (eng. strongly connected components) i nedenstående graf.



Opgave 3 (modellering og anvendelse af algoritmer/datastrukturer)

Gittergrafer

En $k \times k$ *gittergraf* er en graf hvor knuderne er arrangeret i k rækker hver indeholdende k knuder. Kanterne i en gittergraf kan kun gå mellem knuder der er lige til højre eller venstre for hinanden eller lige ovenover hinanden (der behøver ikke at være kanter mellem alle knuder der er over eller ved siden af hinanden). Nedenstående figur viser en 6×6 gittergraf.



3.1 Lad n og m betegne henholdvis antallet af knuder og kanter i en $k \times k$ gittergraf. Udtryk n som en funktion af k.

$n = k^2$

3.2 Hvor stor kan m maksimalt være (sæt kun ét kryds)?

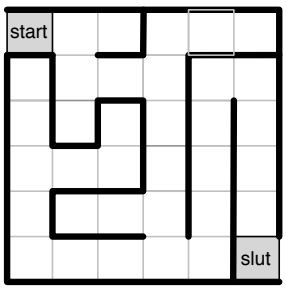
[A] $O(\sqrt{n})$ [X] $O(n)$ [C] $O(n \log n)$ [D] $O(n^2)$

Labyrinter

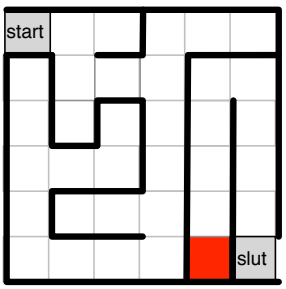
Vi siger at en labyrint er *korrekt konstrueret* hvis

1. der er ét startfelt og ét slutfelt,
2. der er netop én vej fra start til slut,
3. ethvert sted i labyrinten kan nås fra start, og
4. der ikke er nogle kredse/loops i labyrinten.

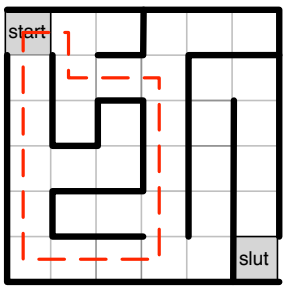
Nedenfor er 3 eksempler på laybrinter. Start og slutfelterne er markeret med grå. Labyrint 1 er korrekt konstrueret. Labyrint 2 er *ikke* korrekt konstrueret, da der ikke er nogen vej fra start til slut og nogle steder/felter ikke kan nås fra start (f.eks. kan det røde felt ikke nås). Labyrint 3 er *ikke* korrekt konstrueret, da der er en kreds.



Labyrint 1



Labyrint 2



Labyrint 3

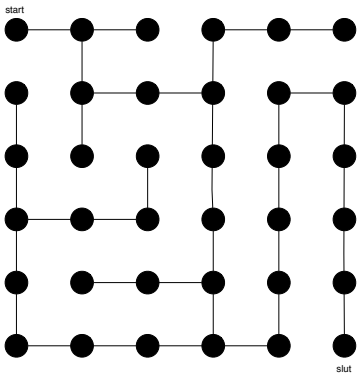
Vi siger at labyrinten er tegnet i et $k \times k$ gitter hvis der er k rækker af felter med hver k felter i. Alle ovenstående labyrinter er tegnet i et 6×6 gitter.

De næste 3 opgaver går ud på at modellere labyrinter som grafer, og konstruere en algoritme der ved hjælp af denne modellering kan afgøre om en labyrint er korrekt konstrueret.

3.3 Beskriv hvordan en labyrint tegnet i et $k \times k$ gitter kan modelleres som en $k \times k$ gittergraf.

Lad hvert felt i labyrinten være repræsenteret af en knude i grafen. Lad der være en kant mellem to knuder, hvis de to respektive felter er tilstødende og der ikke er en mur mellem felterne. Hver knude har tilknyttet et typefelt, hvor der kan angives om en knude er en start- eller slutknude.

3.4 Tegn Labyrint 1 som en gittergraf:



3.5 Giv en effektiv algoritme, der givet en labyrint modelleret som en $k \times k$ gittergraf afgør om labyrinten er korrekt konstrueret. Argumenter for at din algoritme løser problemet korrekt.

Angiv køretiden af din algoritme i O-notation som en funktion af k. Din analyse skal være så tæt som mulig. Begrund dit svar.

Der er fire krav til en korrekt konstrueret labyrint. Disse har følgende betydning for grafen.

1. Der er én knude af typen *start* og én knude af type *slut*.
2. Grafen er kredsfri. Hvis der er flere end en sti fra start til slut så må grafen indeholde en kreds.
3. Grafen er sammenhængende.
4. Svarer til punkt 2.

Vi ved at BFS algoritmen bruger en farve til at holde styr på, om en knude allerede har været udforsket. Hvis BFS ser en knude, som allerede har været udforsket, og som ikke er den aktuelle knudes forgænger, så må der findes en kreds i grafen. Vi modificerer BFS til at afgøre om der findes en kreds baseret på dette. Konkret indsætter vi følgende som det første i den inderste for-løkke af BFS:

```
1: if (v.color = BLACK and v ≠ u.π) then
2:     print "Labyrinten indeholder en kreds."
3:     exit
4: end if
```

Den nuværende knude er u og i en iteration undersøger algoritmen knuden v hvis der er en kant fra u til v. Hvis v har farven sort, så har den været udforsket før. Da u har en kant tilbage til knuden der blev udforsket før u, så bruger vi kravet $v \neq u.\pi$ til at afgøre om v er den knude hvorfra u blev opdaget. Hvis dette er tilfældet, så er der ikke en kreds.

Den endelige algoritme ser nu ud som følger.

- Kør den modificerede udgave af BFS.
- Iterer over alle knuder og kontroller, at de har været besøgt af BFS. Kontroller samtidig at der kun er en startknude og en slutknude.

Den modificerede udgave af BFS har samme køretid som BFS, dvs. $O(n+m)$. Trin 2 af algoritmen har køretiden $O(n)$. Dvs. algoritmen har en køretid på $O(n+m)$. Vi bemærker her, at en graf uden kreds er et træ, dvs. vi ved at gittergrafen for en korrekt konstrueret labyrint har $m = n - 1$ kanter, så algoritmens køretid er faktisk $O(n)$.

Opgave 4 (Træer og rekursion)

Denne opgave handler om rekursion og rodfæstede binære træer. Hver knude har *enten to eller ingen* børn. Knuden x's venstre barn betegnes $left[x]$, og dens højre barn betegnes $right[x]$. Hvis knuden x ikke har nogle børn, har $left[x]$ og $right[x]$ den specielle NIL værdi. Hvis knude x ikke har nogle børn, kaldes den et blad. Ellers kaldes den en intern knude. Såfremt rodknuden for et træ er NIL, er træet tomt. Til hver knude i træet er der knyttet en farve; knuden x har farven $farve[x]$.

Betragt følgende algoritme:

Algorithm 3 UDSKRIV(x)

```
1: if (x ≠ NIL) then
2:    print farve[x]
3:    if (left[x] ≠ NIL) then
4:       UDSKRIV(left[x])
5:       UDSKRIV(right[x])
6:    end if
7: end if
```

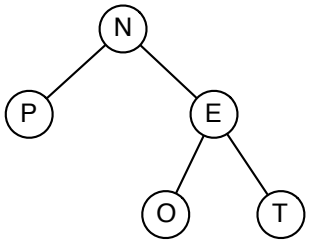


Figure 1: Træ med knuder, der har farver angivet som bogstaver.

Lad x være rodknuden for træet i figur 1. Et kald til algoritmen UDSKRIV(x) vil da udskrive farvesekvensen "NPEOT".

4.1 Du skal ændre algoritmen UDSKRIV, således at det rekursive gennemløb ved kaldet til UDSKRIV(x) for rodknuden x i træet fra figur 1 udskriver farvesekvensen "NETOP" i stedet. Argumenter for at din algoritme er korrekt.

Ovenstående algoritme udskriver knuderne i preorder, dvs. en knude bliver udskrevet før dens børn. Den modificerede algoritme gør det samme, men den rekurserer på det højre barn før det venstre barn.

```
1: if (x ≠ NIL) then
2:    print farve[x]
3:    if (left[x] ≠ NIL) then
4:       UDSKRIV(right[x])
5:       UDSKRIV(left[x])
6:    end if
7: end if
```

Med den definition af binære træer, vi benytter i denne opgave, kan vi beregne antallet af knuder og antallet af blade i et træ ved hjælp af følgende to algoritmer:

Algorithm 4 NODES(x)

```
1: if (x = NIL) then
2:     return 0
3: else
4:     return 1 + NODES(left[x]) + NODES(right[x])
5: end if
```

Algorithm 5 LEAVES(x)

```
1: if (x = NIL) then
2:     return 0
3: else if left[x] = NIL then
4:     return 1
5: else
6:     return LEAVES(left[x]) + LEAVES(right[x])
7: end if
```

4.2 En intern knude i et træ er en knude, som ikke er et blad. Konstruer en algoritme INTERN(x), der givet en rodknude x for et træ returnerer antallet af interne knuder i træet. Angiv tidskompleksiteten/køretiden af din løsning i O-notation. Begrund dit svar.

Vi laver en algoritme som udnytter de to algoritmer der er givet. Antallet af interne knuder er antallet af knuder minus antallet af blade. Vi får derfor følgende algoritme.

```
1: if (x = NIL) then
2:     return 0
3: else
4:     return NODES(x) - LEAVES(x)
5: end if
```

Køretiden for NODES er $O(n)$ da hvert rekursive kald er på et distinkt barn af en knude, så hver knude bliver betragtet én gang, og hvert rekursive kald består af et konstant antal operationer. Samme argument gælder for LEAVES, så køretiden for INTERN er $O(n) + O(n) = O(n)$.

4.3 Et træ har en rød rodvej, hvis der er en vej i træet fra roden x til et blad, hvor alle knuderne på vejen fra x til bladet har farven rød. Konstruer en rekursiv algoritme RØDRODVEJ(x), der returnerer tallet 1, hvis træet med roden x har en rød rodvej. Hvis en sådan vej ikke eksisterer, skal algoritmen returnere 0. Argumenter for at din algoritmen er korrekt. Angiv tidskompleksiteten/køretiden af din løsning i O-notation. Begrund dit svar.

Basistilfældet for vores rekursive algoritme er når inputknuden er det tomme træ, dvs. $x = \text{NIL}$. I dette tilfælde definerer vi, at der ingen rød rodvej er.

Herefter kigger vi på de ikke trivielle tilfælde. Hvis en knude x er rød og der er en rød rodvej fra mindst en af dens børn, så er der også en rød rodvej i deltræet rodfæstet i x. Hvis x ikke har nogen børn så er der også en rød rodvej i deltræet rodfæstet i x. Disse krav er implementeret i algoritmens linie 4 og 5.

```
1: if (x = NIL) then
2:    return 0
3: else
4:    if (farve[x] = RØD) then
5:       if (left[x] = NIL or RØDRODVEJ(left[x]) = 1 or RØDRODVEJ(right[x]) = 1) then
6:          return 1
7:       else
8:          return 0
9:       end if
10:   else
11:      return 0
12:   end if
13: end if
```

Algoritmens køretid er $O(n)$ da hvert rekursive kald er på et distinkt barn af en knude, så hver knude bliver betragtet én gang, og hvert rekursive kald består af et konstant antal operationer.

Opgave 5 (datastrukturer)

5.1 I hvilke af nedenstående datastrukturer kan man finde predecessor af et tal x (dvs. det største tal der er $\leq x$) i worst case $O(\log n)$ tid, hvor n er antal elementer i datastrukturen (du må gerne sætte mere end et kryds)?

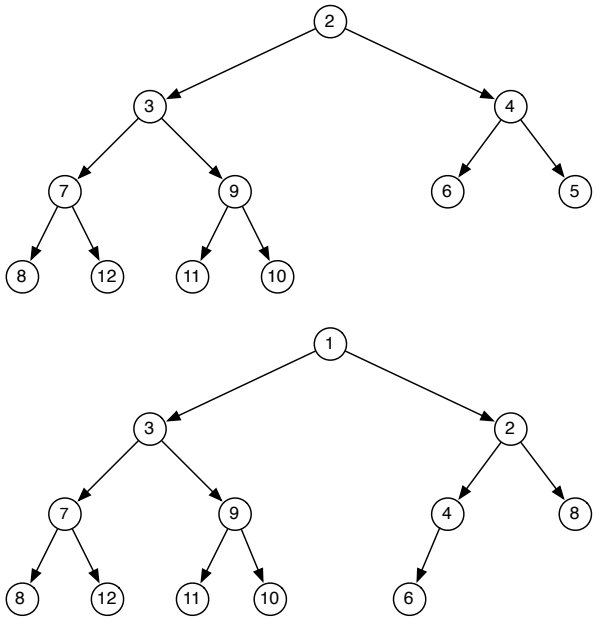
[1] sorteret hægtet liste (eng. sorted linked list)	[2] hob	[3] binært søgetræ
[X] sorteret tabel (eng. sorted array)	[5] Hashtabel	[X] balanceret binært søgetræ

5.2 I hvilke af nedenstående datastrukturer kan man udskrive alle elementerne i sorteret rækkefølge i $O(n)$ tid, hvor n er antal elementer i datastrukturen (du må gerne sætte mere end et kryds)?

[1] Hashtabel	[X] sorteret hægtet liste	[3] hob
[X] sorteret tabel	[5] usorteret hægtet liste	[X] binært søgetræ

5.3 Denne opgave omhandler minhobe, som beskrevet i bogen i afsnit 2.5.

a) Angiv hvordan hoben nedenfor ser ud efter indsættelse af et element med nøgle 1.



b) Angiv hvordan hoben nedenfor ser ud efter én ExtractMin operation.