Danmarks Tekniske Universitet

Skriftlig prøve, den 26. maj 2009.

Kursusnavn Algoritmik og datastrukturer I Kursus nr. 02105.

Tilladte hjælpemidler: Alle skriftlige hjælpemidler.

Vægtning af opgaverne: Opgave 1 - 24%, Opgave 2 - 16%, Opgave 3 - 23%, Opgave 4 - 12 %, Opgave 5 - 25 %.

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*
$n^7 = O(n^3)$	☐	☒
$n(\log n)^3 = O(n^2)$	☒	☐
$2^n = O(n^2)$	☐	☒
$n^2 + \frac{1}{2}n = \Omega(n)$	☒	☐
$n(n-3)/17 = \Theta(n^2)$	☒	☐

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 $f(n) = O(g(n))$.

$4\sqrt{n}$

$2/\log n$

$\frac{1}{2}n^3$

$\frac{2}{3}n$

$(\log n)^5$

Svar: $2/\log n,\ (\log n)^5,\ 4\sqrt{n},\ \frac{2}{3}n,\ \frac{1}{2}n^3$

1.3 Antag at du har en algoritme hvis køretid er præcist $5n^2$. Hvor meget langsommere kører algoritmen hvis du forbdobler inputstørrelsen?

☐ A dobbelt så langsom ☒ **B 4 gange langsommere** ☐ C 5 gange langsommere

☐ D 10 gange langsommere ☐ E 20 gange langsommere

1.4 Betragt nedenstående algoritme.

Algoritme Løkke1(n)
1. $x = 1$
2. **for** $i = 1$ **to** n
3. **for** $j = 1$ **to** n
4. **for** $k = 1$ **to** n
5. $x = x + 1$

a) Køretiden af algoritmen er

☐ A $\Theta(\log n)$ ☐ B $\Theta(n)$ ☐ C $\Theta(n \log n)$ ☐ D $\Theta(n^2 \log n)$ ☒ $\underline{\Theta(n^3)}$

☐ F $\Theta(n^{3/2})$ ☐ G $\Theta(2^n)$ ☐ H $\Theta(n^4)$ ☐ I $\Theta(\sqrt{n})$

b) Hvis linie 4 ændres til "**for** $k = j$ **to** n" så bliver køretiden:

[A] asymptotisk langsommere

[X] **asymptotisk den samme**

[C] asymptotisk hurtigere

1.5 Betragt nedenstående algoritme.

Algoritme Løkke2(n)
1. $i = 1$
2. **while** $i \leq n$ **do**
3. $j = 1$
4. **while** $j \leq n$ **do**
5. $j = j + 1$
6. $i = 2i$

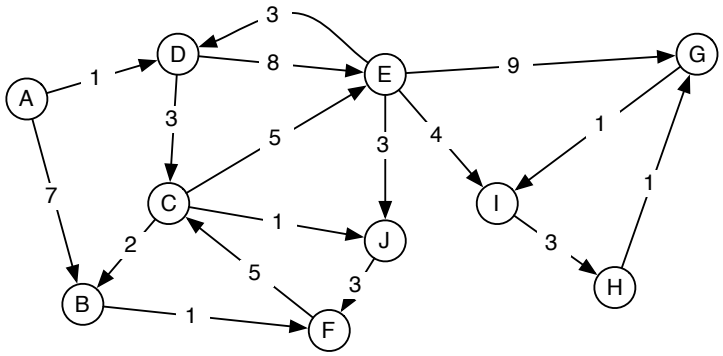
Køretiden af algoritmen er

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

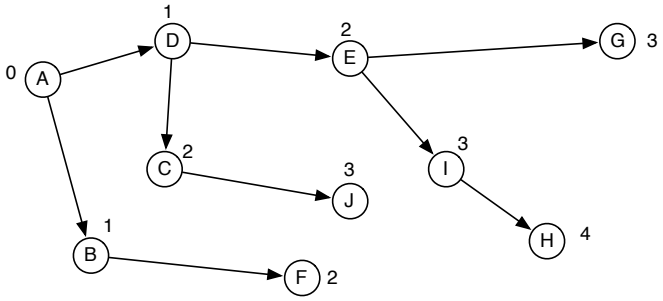
[F] $\Theta(n^{3/2})$ [G] $\Theta(2^n)$ [H] $\Theta(n^4)$ [I] $\Theta(\sqrt{n})$

Opgave 2 (grafer)

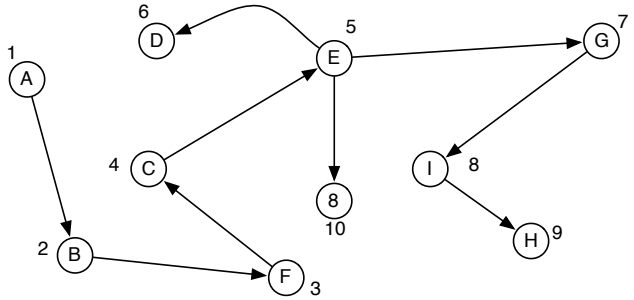
2.1 Betragt nedenstående graf G med ikke-negative kantvægte. Det antages at incidenslisterne er sorteret alfabetisk.



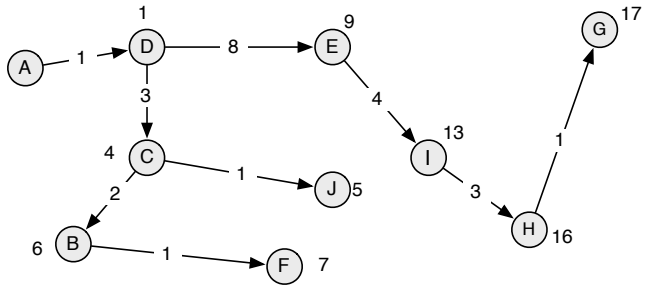
a) Angiv et BFS træ for grafen G når BFS genemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



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.

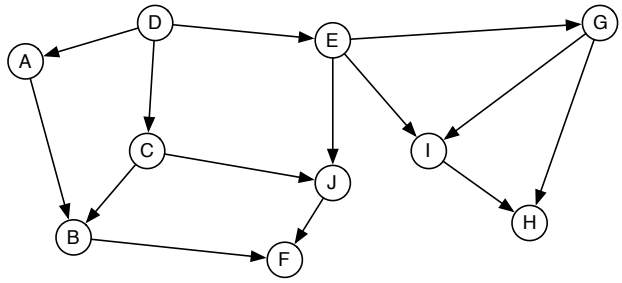


c) Angiv et korteste veje træ for grafen G når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



Obs. Det er også korrekt at have en kant fra C til E i stedet for D til E.

2.2 Betragt nedenstående DAG D.



Hvilke af følgende rækkefølger angiver en topologisk sortering af knuderne i grafen D:

[1] D E A C B J F I G H [2] D A E C B I J G H [3] D A B C E J F G I H

[X] **D E A C B J F G I H** [5] D A E C B J G H [6] D E A C F B J G I H

Opgave 3 (modellering, anvendelse og analyse af algoritmer)

Du er konsulent for langtursbusfirmaet "Blåhundebusserne" der ønsker at lave en hjemmeside hvor folk kan finde og bestille deres rejse. Man skal både kunne søge på den billigste rejse mellem to valgfrie destinationer og på den rejse med færrest antal omstigninger (busskift). Der er S forskellige stoppesteder/stationer og B forskellige busruter. For hver busrute kender du startsted og destination, samt prisen for at benytte ruten. En bus stopper ikke undervejs, men kører direkte fra startsted til destination. Du skal ikke tage hensyn til tidspunkt for afgang og ankomst i denne opgave.

3.1 Giv en effektiv algoritme der finder den billligste rejse mellem to givne destinationer s og t. Rejsen kan være en kombination af flere forskellige busruter. Prisen for rejsen bliver så summen af priserne for de ruter der benyttes. Angiv køretiden af din algoritme i asymptotisk notation. Din analyse skal være så tæt som mulig.

Problemet kan modelleres med en vægtet graf. Stationerne er knuder og ruterne er kanter. Der er en kant fra knude A til knude B, hvis der er en busrute fra station A til station B. Kantvægten på en kant er prisen for den tilsvarende busrute.

Problem kan nu løses vha. en korteste vej forespørgsel: Den billigste rejse fra A til B svarer til den korteste vej fra A til B i den vægtede graf. Vi kan bruge Dijkstras algoritme til korteste vej.

Lad m være antal busruter (= antal kanter) og $n$være antal stationer (= antal knuder). Det tager linær tid at lave grafen, dvs. $O(m+n)$: Vi laver en graf med incidenslister. For at lave knudetabellen gives hver station et nummer (der kan bruges en hashtabel). Incidenslisterne laves nu ved at gennemgå hver busrute og indsætte en kan ml. startsted og destination med vægt lig prisen for ruten. Dijkstras algoritme tager $O(n+m \log n)$ tid. I alt tager det $O(n+m \log n)$ tid.

3.2 Giv en effektiv algoritme der finder den rejse mellem to givne destinationer s og t der har færrest antal omstigninger. Angiv køretiden af din algoritme i asymptotisk notation. Din analyse skal være så tæt som mulig.

Problemet kan modelleres med samme graf som ovenfor. Den rejse med færrest omstigninger mellem A og B svarer til den sti mellem A og B med færrest kanter. Vi kan derfor løse problemet vha. BFS-agoritmen. Som ovenfor tager det $O(m+n)$ tid at konstrurere grafen og BFS tager også $O(m+n)$ tid, da vi bruger incidenslister til at repræsentere grafen.

3.3 Firmaet ønsker også hjælp til at finde ud af hvornår busserne skal stoppe og tanke. Givet en busrute, kender vi placeringen af alle tankstationerne på ruten. Vi ved også hvor mange km k bussen kan køre på en fuld tank. Vi ønsker at lave så få stop for at tanke som muligt. Vi antager at bussen starter med en fuld tank. Vi kalder dette for *kør-og-tank* problemet.

Eksempel. Ruten er 300km. Bussen kan køre 110 km på en fuld tank. Tankstationerne på ruten ligger efter 10, 50, 80, 140, 180, 250, og 270 km.
I dette tilfælde skal bussen skal tanke mindst 3 gange, f.eks. efter 50, 140 og 250 km.

En af chaufførerne kommer med følgende strategi:

Grådig strategi: Kør så langt som muligt før der tankes.

a) Angiv hvor bussen stopper for at tanke hvis strategien bruges på eksemplet ovenfor.

Efter 80, 180 og 270 km.

b) Giv et argument for at den grådige strategi er korrekt eller giv et modeksempel.

Den grådige strategi er optimal. Lad L være længden af ruten. Vi siger at en sekvens af tankstationer er lovlig, hvis afstanden mellem to efter hinanden følgende stationer er højst k, og afstanden fra startsted til første tankstation, samt afstangen fra sidste tankstation til destinationen er højst k.

Lad $S = \{x_{s_1}, \ldots, x_{s_n}\}$ være sekvensen af tankstationer valgt af den grådige strategi. Lad $R = \{x_{t_1}, \ldots, x_{t_m}\}$ være en anden lovlig sekvens. Vi vil vise at $n \leq m$. Da dette gælder for enhver anden sekvens medfører det at S er en optimal sekvens. Vi vil først vise at efer hvert stop er bussen i S mindst lige så som i løsning R:

(*) For $j = 1, 2, \ldots, m$ har vi $x_{s_j} \geq x_{t_j}$.

Vi bruger induktion over j til at vise (*).
Basistilfælde $j = 1$: følger direkte af definitionen af den grådige strategi.
Induktionsskridt: Antag at det gælder for $i < j$ (induktionshypotesen). Vi har

$$\begin{aligned} k &\geq x_{t_j} - x_{t_{j-1}} && \text{da } R \text{ er lovlig} \\ &\geq x_{t_j} - x_{s_{j-1}} && \text{da } x_{s_{j-1}} \geq x_{t_{j-1}} \text{ i følge induktionshypotesen} \end{aligned}$$

Altså kan bussen nå fra $x_{s_{j-1}}$til x_{t_j}på en fuld tank. Da bussen i følge den grådige strategi kører så langt som muligt før den tanker har vi at $x_{s_j} \geq x_{t_j}$.

(*) medfører at $x_{s_m} \geq x_{t_m}$. Antag at $m \leq n$. Så er $x_{s_m} \leq L - k$, ellers ville bussen ikke behøve at tanke på tankstation $x_{s_{m+1}}$. Men så er $x_{t_m} \leq x_{s_m} \leq L - k$, og så er R ikke lovlig. Altså må $m \geq n$ og den grådige strategi er optimal.

Opgave 4 (rekursion)

4.1 Denne opgave omhandler en algoritme for fakultetsfunktionen ($n! = n \cdot (n-1) \cdots 1$). Nedenfor er 3 forsøg på at lave en rekursiv algoritme der beregner $n!$.

Algoritme Fak1(n)
  if $n = 1$
    return n
  $x = n \cdot (\text{Fak1}(n) - 1)$
  return x

Algoritme Fak2(n)
  if $n = 0$
    return n
  $x = n \cdot \text{Fak2}(n-1)$
  return x

Algoritme Fak3(n)
  if $n = 1$
    return n
  return $n \cdot \text{Fak3}(n-1)$

Hvilken af algoritmerne beregner $n!$ korrekt når n er et positivt heltal: Fak3

4.2 Nedenstående algoritme tager et positivt heltal som input.

Algoritme Rekur(n)
1. **if** $n \leq 0$
2. **return** 0
3. **return** n + Rekur($n-1$) + 2

Giv iterativ variant af algoritmen:

Rekur-Iter(n)
1. $x = 0$, $i = 1$.
2. **while** $i \leq n$ **do**
3. $x = x + 2 + i$
4. $i = i + 1$
5. **return** x

Opgave 5 (datastrukturer)

5.1 I hvilke af nedenstående datastrukturer kan man lave søgning i worst case $O(\log n)$ tid, hvor n er antal elementer i datastrukturen (du må gerne sætte mere end et kryds)?

[1] Hashtabel	[2] sorteret hægtet liste	[3] hob
[X] **sorteret tabel**	[5] usorteret tabel	[6] binært søgetræ

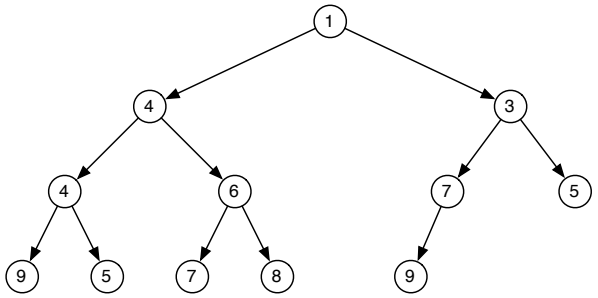
Jeg har givet 3 point hvis man har sat kryds ved både sorteret tabel og binært søgetræ, og 4 point hvis man kun har sat kryds ved sorteret tabel, da opgaven var utilsigtet svær.

5.2 I hvilke af nedenstående datastrukturer kan man lave indsættelse i worst case $O(1)$ tid, hvor n er antal elementer i datastrukturen (du må gerne sætte mere end et kryds)?

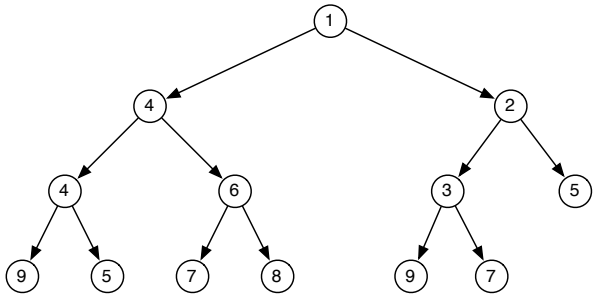
[X] **Hashtabel**	[X] **usorteret hægtet liste**	[3] hob
[4] sorteret tabel	[5] binært søgetræ	

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

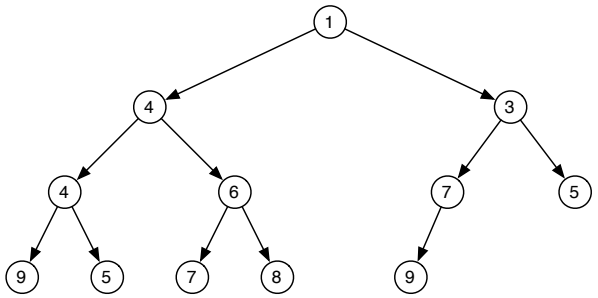
a) Angiv hvordan hoben nedenfor ser ud efter indsættelse af et element med nøgle 2.



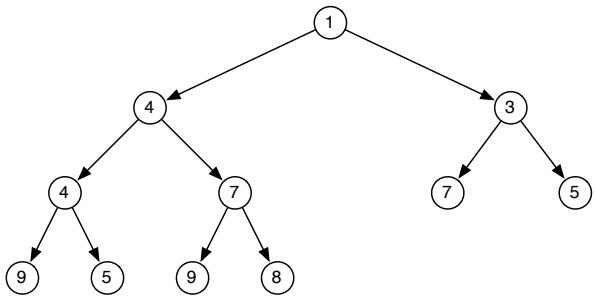
Løsning:



b) Angiv hvordan hoben nedenfor ser ud efter sletning af elementet med nøglen 6.



Løsning:



c) Giv en algoritme der i lineær tid checker om nøglerne i knuderne i et komplet binært træ opfylder hoborden (se side 59 i bogen for definitionen af hoborden).

Algoritmen tager en knude som input og returnerer 1 hvis træet rodfæstet i v opfylder hoborden og 0 ellers.

Agoritme Hoborden(v)

```
if v = NIL { return 1 }
else {
  if (left(v) ≠ NIL and key[left(v)] < key[v])
    { return 0 }
  if (right(v) ≠ NIL and key[right(v)] < key[v])
    { return 0 }
  return min(Hoborden(left(v)), Hoborden(right(v)))
}
```

Et kald til algoritmen tager konstant tid i sig selv (uden de rekursive kald). Det tager konstant tid at checke hver betingelse i if-sætningerne og at returnere 0 eller 1. For hver knude laves der to rekursive kald. Men for enhver knude u kaldes Hoborden(u) kun en gang. Algoritmen tager derfor lineær tid i antallet af knuder i træet.

5.4 I kør-og-tank problemet fra opgave 3.3 kender vi placeringen af tankstationerne på ruten. Antag at placeringen af en tankstation t er angivet som afstanden d_t fra starten til t. Givet en bus' position p og hvor mange km k den kan køre på en fuld tank ønsker vi at finde den tankstation der ligger længst væk af de tankstationer bussen kan nå inden den løber tør for diesel. Dvs. givet p og k ønsker vi at finde t således $d_t \leq p + k$ og d_t er størst mulig. Lad n være antallet af tankstationer. Hvilken datastruktur kan bruges til at finde det ønskede t i $O(\log n)$ tid givet p og k?

Der kan bruges et binært søgetræ eller en sorteret tabel. Forespørgslen find det maksimale $t$således at $d_t \leq p + k$ svarer til en predecessor forespørgsel: pred($p + k$) i en datastruktur der indeholder alle tankstationerne som elementer med placeringen af tankstationerne som nøgler. En predecessor forespørgsel kan besvares i $O(\log n)$tid i et binært søgetræ. I en sorteret tabel kan den besvares i $O(\log n)$ tid ved binær søgning.

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
```

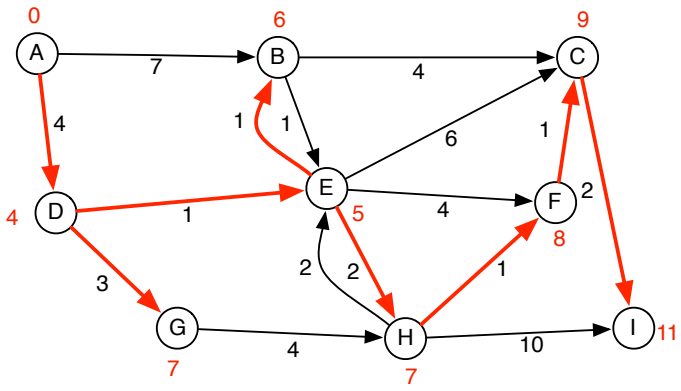
Køretiden af algoritmen er

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

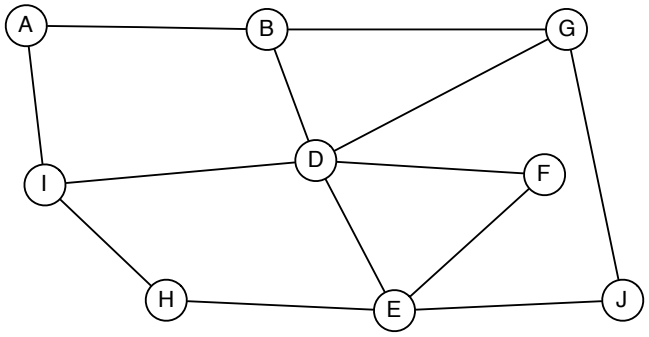
[F] $\Theta(n^2)$ [G] $\Theta(2^n)$ [H] $\Theta(n^4)$ [I] $\Theta(\sqrt{n})$

Opgave 2 (grafer)

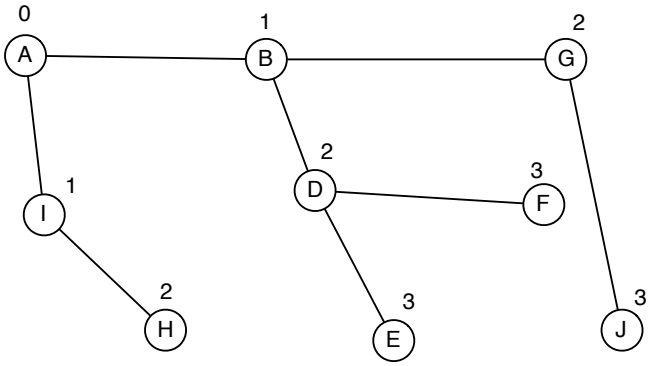
2.1 Angiv et korteste veje træ for nedenstående graf når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



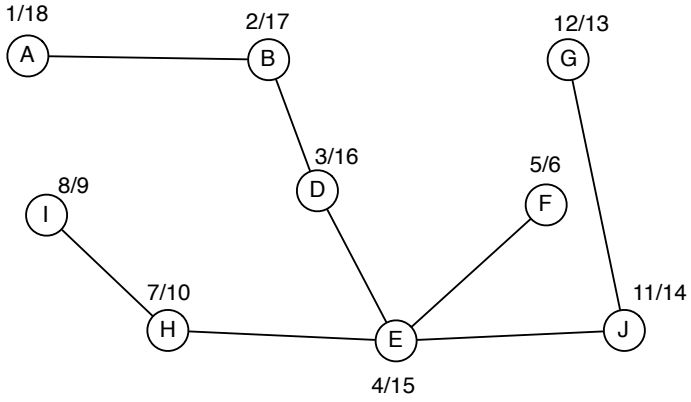
2.2 Betragt nedenstående graf G.



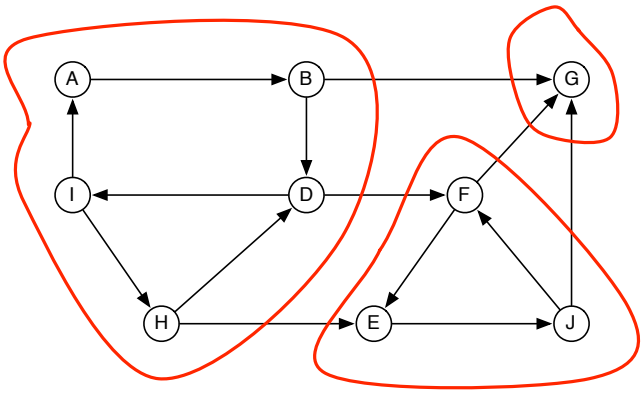
a) Angiv et BFS træ for grafen G når BFS genemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



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.



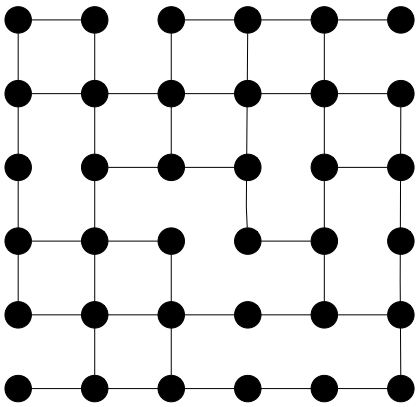
2.3 Angiv de stærke sammenhængskomponenter (eng. strongly connected components) i nedenstående graf.



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)?

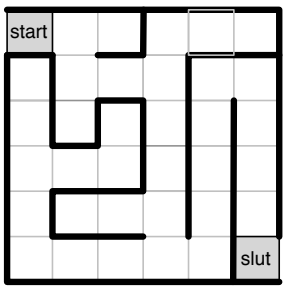
[A] $O(\sqrt{n})$ [X] $O(n)$ [C] $O(n \log n)$ [D] $O(n^2)$

Labyrinter

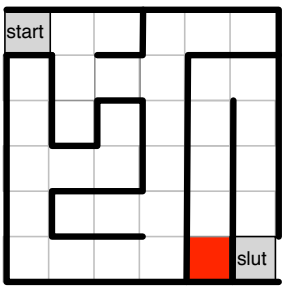
Vi siger at en labyrint er *korrekt konstrueret* hvis

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.

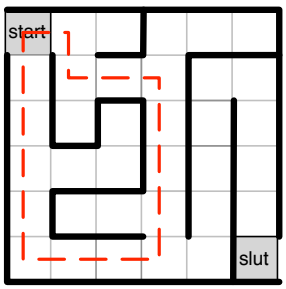
Nedenfor er 3 eksempler på laybrinter. Start og slutfelterne er markeret med grå. Labyrint 1 er korrekt konstrueret. Labyrint 2 er *ikke* korrekt konstrueret, da der ikke er nogen vej fra start til slut og nogle steder/felter ikke kan nås fra start (f.eks. kan det røde felt ikke nås). Labyrint 3 er *ikke*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



Labyrint 1



Labyrint 2



Labyrint 3

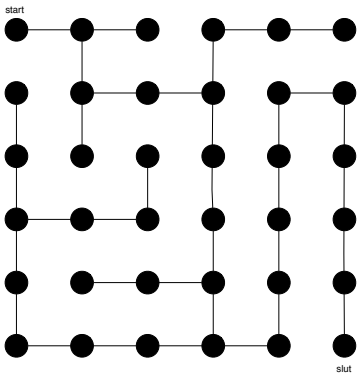
Vi siger at labyrinten er tegnet i et $k \times k$ gitter hvis der er k rækker af felter med hver k felter i. Alle ovenstående labyrinter er tegnet i et 6×6 gitter.

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
```

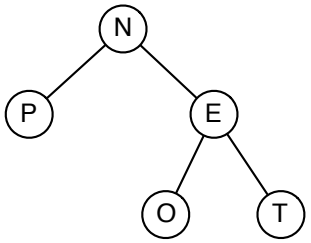


Figure 1: Træ med knuder, der har farver angivet som bogstaver.

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)?

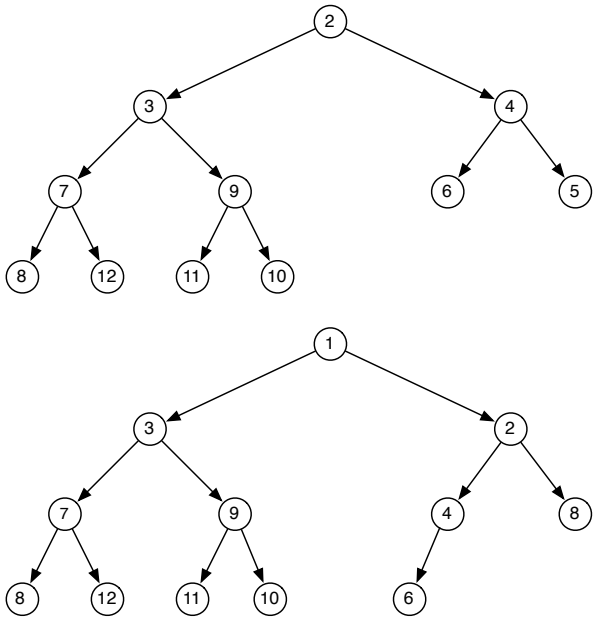
[1] sorteret hægtet liste (eng. sorted linked list)	[2] hob	[3] binært søgetræ
[X] sorteret tabel (eng. sorted array)	[5] Hashtabel	[X] balanceret binært søgetræ

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)?

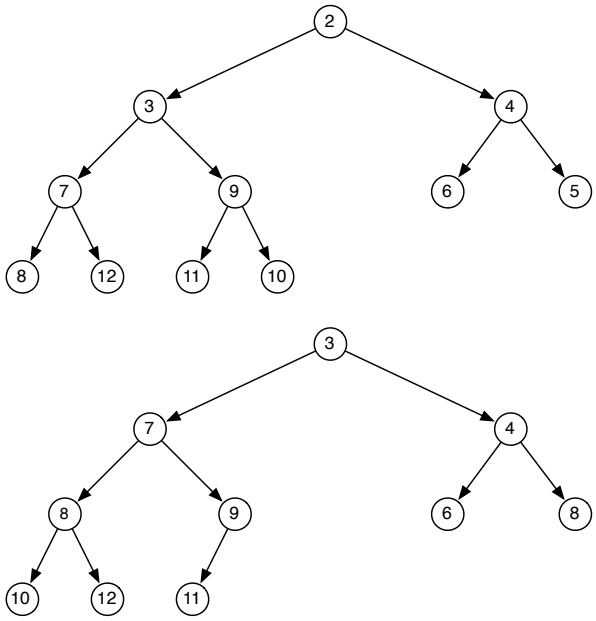
[1] Hashtabel	[X] sorteret hægtet liste	[3] hob
[X] sorteret tabel	[5] usorteret hægtet liste	[X] binært søgetræ

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.



Danmarks Tekniske Universitet

Skriftlig prøve, den 23. maj 2011.

Kursusnavn: Algoritmer og datastrukturer I Kursus nr. 02105.

Varighed: 4 timer

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*
$\frac{1}{20}n^4 - 100n^3 = O(n^3)$	☐	☒
$(\log n)^2 = O(n)$	☒	☐
$\frac{1}{2}n^5 = \Omega(n^6)$	☐	☒
$2^n = O(3^n)$	☒	☐
$n^4(n-1)/5 = \Theta(n^5)$	☒	☐

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 $f(n) = O(g(n))$.

$5000(\log n)^2$

$4n$

$\frac{1}{4}n^2 - 10000n$

$n^{1/100}$

$4n \log n$

Svar: $5000(\log n)^2$ $n^{1/100}$ $4n$ $4n \log n$ $\frac{1}{4}n^2 - 10000n$

1.3 Antag at du har en algoritme hvis køretid er præcist $2n^3$. Hvor meget langsommere kører algoritmen hvis du forbdobler inputstørrelsen?

☐ A dobbelt så langsom ☐ B 3 gange langsommere ☐ C 4 gange langsommere

☒ X 8 gange langsommere ☐ E 16 gange langsommere

1.4 Betragt nedenstående algoritme.

Algorithm 1 Løkke1(n)

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

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 $\Theta(\sqrt{n})$

1.5 Betragt nedenstående algoritme.

Algorithm 2 Løkke2(n)

```
1: i = 1
2: while (i < n) do
3:    for j = 1 to n do
4:       i = i + 1
5:    end for
6:    i = 2 · i
7: end while
```

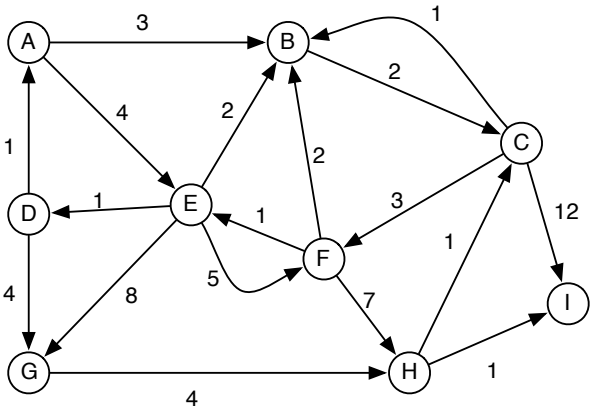
Køretiden af algoritmen er

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

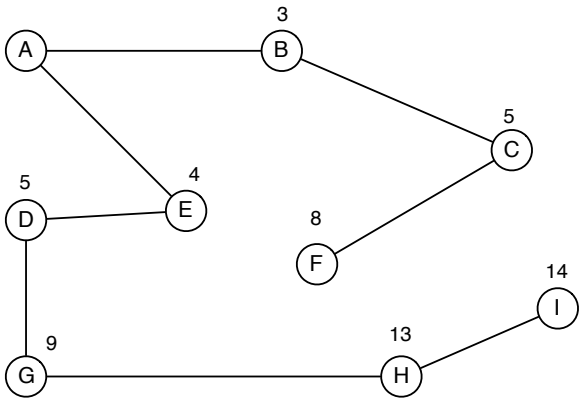
[F] $\Theta(n^2)$ [G] $\Theta(2^n)$ [H] $\Theta(n^4)$ [I] $\Theta(\sqrt{n})$

Opgave 2 (grafer)

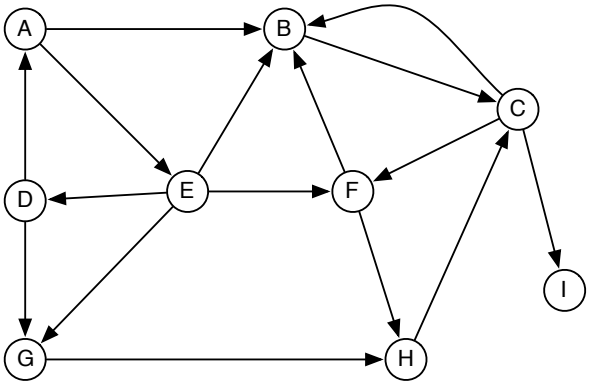
2.1 Angiv et korteste veje træ for nedenstående graf når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



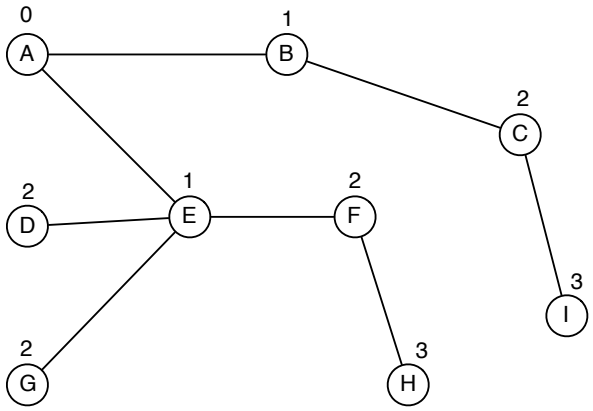
Angiv korteste veje træet og afstandene her:



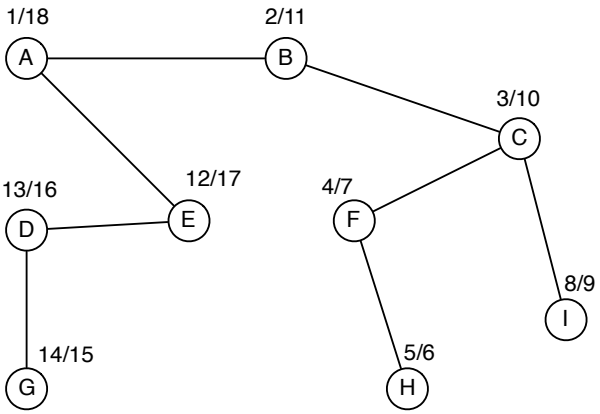
2.2 Betragt nedenstående graf G.



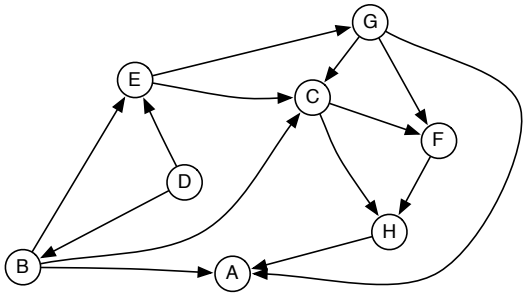
a) Angiv et BFS træ for grafen G når BFS genemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



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 en topologisk sortering af knuderne i nedenstående graf.



D, B, E, G, C, F, H, A.

Opgave 3 (modellering og anvendelse af algoritmer/datastrukturer)

I det lille land Algostan har man besluttet, at lave et registersystem. I første version af systemet skal registret kun kunne håndtere følgende tre operationer:

- INIT(n): Initialiser registret, så det kan håndtere et maksimum på n personer i registret.
- INDSÆT(*CPR, indkomst*): Indsæt en person med cpr-nummer CPR i registret. Til personen skal der tilknyttes en oplysning om årlig indkomst *indkomst* i kr. Det antages, at registret ikke allerede indeholder en person med cpr-nummer CPR i forvejen.
- SLETLAVESTEINDKOMST(): Returner cpr-nummeret for en person, der har den laveste indkomst, samt slet denne person fra registret.

3.1 Beskriv hvordan første version af registret kan laves, så INIT(n) tager tid $O(n)$, og de to andre operationer tager tid $O(\log n)$, hvor n er det maksimale antal personer der kan være i registret. Husk at begrunde dit svar og at argumentere for køretiderne.

Vi bruger en min-hob H som datastruktur med indkomster som nøgler og CPR nummer som satellitdata.

INIT(n) allokerer et array af længde n som skal bruges til at holde elementerne i min-hoben. Dette tager $O(n)$ tid.

INDSÆT(CPR, *indkomst*) kalder MIN-HEAP-INSERT(H, *indkomst*), som vi ved tager $O(\log n)$ tid.

SLETLAVESTEINDKOMST() kalder HEAP-EXTRACT-MIN(H) som også tager $O(\log n)$ tid. Returværdien fra HEAP-EXTRACT-MIN bliver ikke brugt til noget.

Anden version af registret skal ud over ovenstående operationer også understøtte følgende operation:

- FindAlleUnder(k): Udskriv personnumrene på alle personer der tjener mindre end k kr om året.

3.2 Beskriv hvordan anden version af registret kan laves, så FindAlleUnder(k) tager $O(n_k)$ tid, hvor n_k er antal personer i registret der tjener mindre end k kr om året. Procedurene Init, Indsæt og SletLavesteIndkomst skal have samme køretid som i opgave 3.1. Argumenter for at din procedure er korrekt og angiv køretiden for proceduren.

Hvis du ikke kan få køretiden ned på $O(n_k)$, så angiv den bedst mulige implementation af proceduren (i tekst eller pseudokode), som du kan finde på (husk stadig argumentation for korrekthed og køretid).

Vi beholder vores min-hob og laver en algoritme der udnytter min-hob egenskaben $A[\text{Parent}(i)] \geq A[i]$. Algoritmen udskriver CPR nummeret for de elementer der har en indkomst under k startende fra minhobens rod. Hvis algoritmen møder et element hvor indkomsten er større end eller lig med k, så skal dens efterkommenre i min-hoben ikke udskrives. Algoritmen anvender en kø Q som er implementeret vha. en hægtet liste. Pseudokode for FindAlleUnder(k) ses herunder.

```
1: Enqueue(Q, 0)
2: while (Q ≠ ∅) do
3:    i = Dequeue(Q)
4:    if (H[i].key < k) then
5:       print H[i].CPR
6:       Enqueue(Q,Left(i))
7:       Enqueue(Q,Right(i))
8:    end if
9: end while
```

Algoritmen tilføjer kun et element til køen én gang. Alle elementer der bliver tilføjet til køen har enten en indkomst under k eller er barn af et element med indkomst under k. I værste tilfælde da vil de n_k elementer med en indkomst under k udgøre et komplet binært træ. Børnene af bladene i dette træ tilføjes også til Q, så algoritmen tilføjer i alt $2n_k$ elementer til Q. Da Enqueue og Dequeue tager konstant tid fås en køretid på $2n_k = O(n_k)$.

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. Hver knude i træet har et felt $size[x]$, der indeholder et heltal.

Betragt følgende algoritme:

Algorithm 3 ZERO(x)

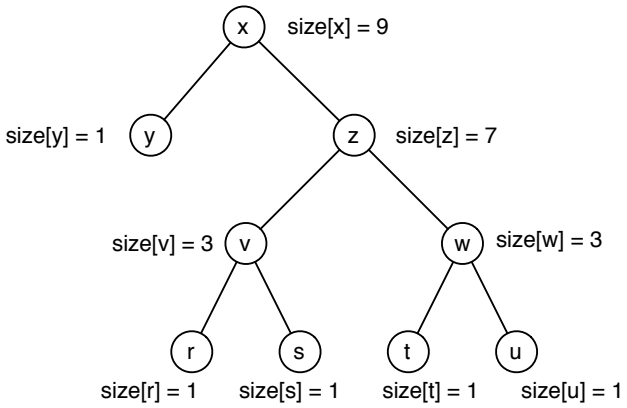
```
1: if (x ≠ NIL) then
2:     size[x] = 0
3:     ZERO(left[x])
4:     ZERO(right[x])
5: end if
```

Lad x være rodknuden for træet T. Efter udørelsen af proceduren ZERO(x) vil alle felterne $size[x]$ være 0.

4.1 Angiv køretiden af proceduren ZERO(x) i O-notation, hvor x er roden i et træ med n knuder. Begrund dit svar.

Linie 2 i algoritmen tager konstant tid. Det rekursive kald i linie 3 får x's venstre barn som input og kaldet i linie 4 får x's højre barn som input. Da inputknuden x er rod i et træ vil den aldrig komme i betragtning igen. Dvs. hver knude får kun sin *size* værdi ændret én gang, så køretiden for algoritmen er $O(n)$.

4.2 I de følgende opgaver betegner $T(x)$ undertræet med rod x i T. Proceduren INITSIZE(x), skal givet roden x i et træ T med n knuder initialisere felterne $size[y]$ for alle knuder y i T, så $size[y]$ bliver lig med antallet af knuder i $T(y)$. Se eksemplet nedenfor.



Giv pseudokode for InitSize(x), så køretiden er $O(n)$. Argumenter for køretiden af din algoritme, samt for at den er korrekt.

Basistilfældet for vores rekursive algoritme er når x er NIL, dvs. når input er et tomt træ. I dette tilfælde skal der ikke skrives noget til knuden, såalgoritmen skal ikke foretage noget. Basistilfældet for et ikke tomt træer når x er et blad, dvs. når $left[x] =$ NIL og $right[x] =$ NIL. I dette tilfælde er størrelsen af deltræet rodfæstet i x lig med 1.

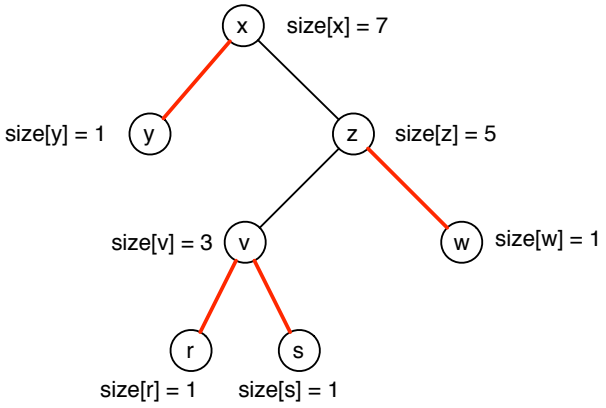
Når x ikke er et blad, så er størrelsen af deltræet rodfæstet i x lig med summen af størrelsen af deltræerne rodfæstet i x's højre og venstre barn plus 1 for at medregne x. For det ikke trivielle tilfælde skal vi altså kende størrelsen af deltræerne rodfæstet i x's børn for at beregne størrelsen af deltræet rodfæstet i x, så vores algoritme kaldes først rekursivt på x's børn. Pseudokode ses herunder.

InitSize(x):

```
1: if (x ≠ NIL) then
2:    if (left[x] = NIL and right[x] = NIL) then
3:       size[x] = 1
4:    else
5:       InitSize(left[x])
6:       InitSize(right[x])
7:       size[x] = size[left[x]] + size[right[x]] + 1
8:    end if
9: end if
```

Hvis x er et blad så tager linie 2 konstant tid. Hvis x ikke er et blad bliver algoritmen kørt rekursivt på x's højre og venstre barn. Således bliver x aldrig input til algoritmen igen, så linie 6 bliver kun kørt én gang pr. knude der ikke er et blad. Derfor har algoritmen køretiden $O(n)$.

For et træ T siger vi at en kant (u, v), hvor u er forælder til v, er *rød*, hvis antallet af knuder i undertræet $T(u)$ er mindst dobbelt så stort som antallet af knuder i $T(v)$. Dvs. efter udførelse af InitSize gælder $size[u] \geq 2size[v]$ for alle *røde* kanter (u, v) i T. Se eksempel nedenfor.



4.3 Konstruer en rekursiv algoritme RØDKANT(x) der givet en rodknude x for et træ returnerer antallet af *røde* kanter i træet. Argumenter for at din algoritme er korrekt. Angiv tidskompleksiteten/køretiden af din løsning i O-notation. Begrund dit svar.

Vi laver en rekursiv algoritme, som givet en knude x afgør om knuden har 0, 1 eller 2 røde kanter til sine børn, og returnerer dette antal plus summen af røde kanter i deltræerne rodfæstet i x's børn.

Algoritmens basistilfælde er hvis x er det tomme træ eller x er et blad. I dette tilfælde har træet ingen røde kanter. Det ikke trivielle tilfælde er når x er rod i et træ med mere end en knude. I dette tilfælde skal algoritmen kontrollere om størrelsen af et undertræ rodfæstet i et barn y er mindre mindre end $size[v]/2$, og i så fald regne disse kanter som røde. Pseudokode er givet herunder.

RØDKANT(x):

```
1:  if (x = NIL or (left[x] = NIL and right[x] = NIL)) then
2:     return 0
3:  else
4:     red_edges =RØDKANT(left[x]) +RØDKANT(right[x])
5:     if (size[left[x]] ≤ size[x]/2) then
6:        red_edges = red_edges + 1
7:     end if
8:     if (size[right[x]] ≤ size[x]/2) then
9:        red_edges = red_edges + 1
10:    end if
11:    return red_edges
12: end if
```

Algoritmen er korrekt fordi den, i det ikke trivielle tilfælde, først tæller antallet af røde kanter i deltræerne rodfæstet i x's børn i linie 4, og herefter afgør om de to kanter fra x til dens børn er røde (linie 5 og 8). Hvis x er et blad tager linie 2 konstant tid. Linie 5 til 11 er et konstant antal aritmetiske operations og sammenligninger, så dette tager også konstant tid for hvert rekursive kald. Da algoritmen bliver kaldt rekursivt på x's børn bliver en knude aldrig betragtet mere end én gang, så algoritmens køretid er $O(n)$.

4.4 Angiv i O-notation en øvre grænse for antallet af røde kanter, der kan være på stien fra en knude til roden i et træ T. Begrund dit svar.

Der kan højst være $O(\log n)$ røde kanter på en sti fra roden til et blad.

Lad v være en knude med to børn x og y. Hvis kanten (v, x) er rød, så betyder det at $size[x]$ er højst $\frac{size[v]}{2}$, så for hver gang vi følger en rød kant ned i træet halverer vi antallet af knuder i træet rodfæstet i den nuværende knude. Da n kan halveres $O(\log n)$ gange før $n = 1$, så følger det at en rød sti fra roden til et blad højst kan være $O(n)$ dyb.

Opgave 5 (datastrukturer)

5.1 Lad S være en stak. Udfør følgende operationer fra venstre til højre: et bogstav i betyder *Push(S,i)* og $*$ betyder *Pop(S)*.

D * T U * * I N * F O R * M * A T I K

Angiv sekvensen af bogstaver der bliver "pop"'et (returneret af *Pop(S)*) af disse operationer:

A D U T I R M

B D U T N R M K I T A O F

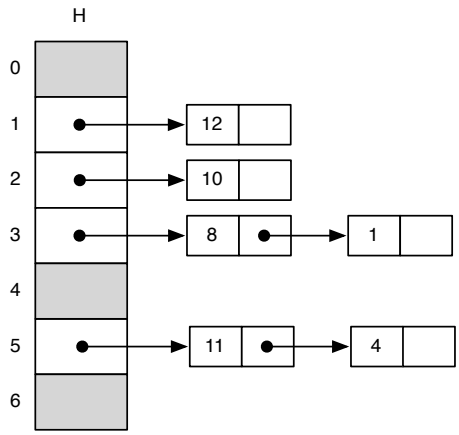
X D U T N R M

D D T U I N F

E D U T N R M I T A O F

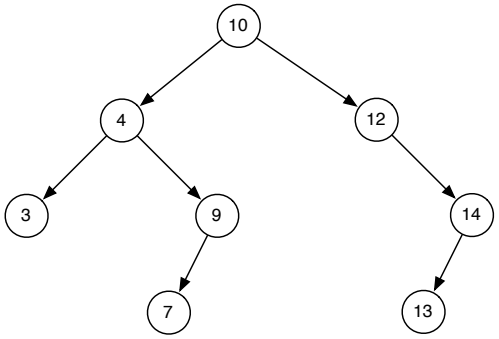
F D U T N O M

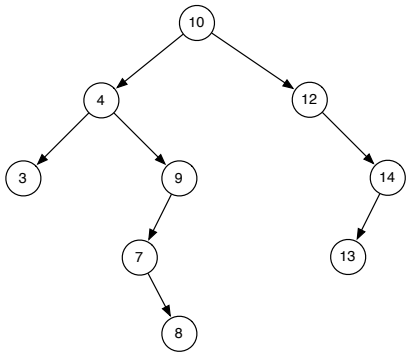
5.2 Lad H være en kædet hashtabel (chained hashing) af størrelse 7 med hashfunktion $h(x) = 3x \mod 7$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 4, 1, 12, 8, 10, 11.



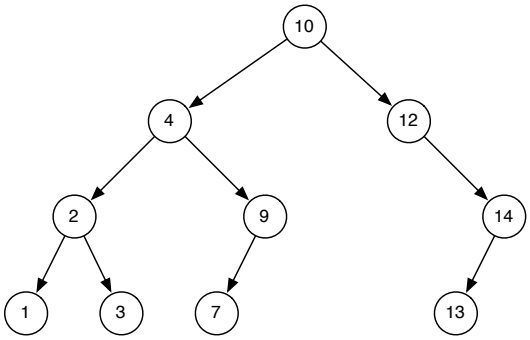
5.3 Denne opgave omhandler (ubalancerede) binære søgetræer, som beskrevet i de udleverede noter CLRS kapitel 12.

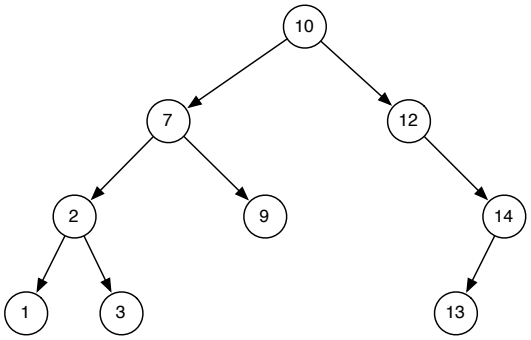
Opgave a Angiv hvordan det binære søgetræ nedenfor ser ud efter indsættelse af et element med nøgle 8.



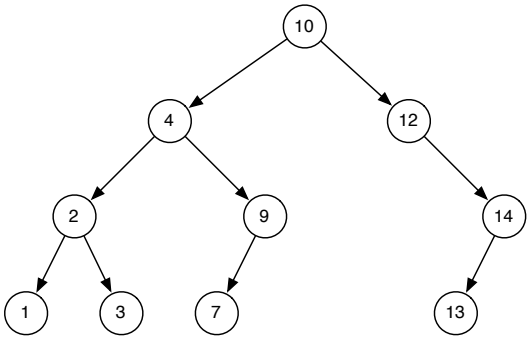


Opgave b Angiv hvordan det binære søgetræ nedenfor ser ud efter sletning elementet med nøgle 4.





Opgave c Angiv den rækkefølge af knuderne bliver skrevet ud i når man laver et preorder gennemløb af ovenstående træ.



Preorder gennemløb: $10, 4, 2, 1, 3, 9, 7, 12, 14, 13$

Danmarks Tekniske Universitet

Skriftlig prøve, den 16. maj 2012.

Kursusnavn: Algoritmer og datastrukturer I Kursus nr. 02105.

Tilladte hjælpemidler: Skriftlige hjælpemidler.

Varighed: 4 timer

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 benytte ekstra papir som så vedlægges opgavebesvarelsen.

Opgave 1 (kompleksitet)

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

	Ja	*Nej*
$\frac{1}{20}n^2 + 100n^3 = O(n^2)$	☐	☒
$(\log n)^2 + n = O(n)$	☒	☐
$(\log n)^2 + n = \Theta(n)$	☒	☐
$2^{\log_2 n} = O(n)$	☒	☐
$n^3(n-1)/5 = \Theta(n^3)$	☐	☒

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 $f(n) = O(g(n))$.

$n(\log n)^2$

$n^2 \log n$

$n^3 + 100n - 5000n^2$

2^4

$n^{1/4}$

2^n

Svar: $2^4, n^{1/4}, n(\log n)^2, n^2 \log n, n^3 + 100n - 5000n^2, 2^n$

1.3 Antag at du har en algoritme hvis køretid er præcist $3n^4$. Hvor meget langsommere kører algoritmen hvis du fordobler inputstørrelsen?

A dobbelt så langsom	B 3 gange langsommere	C 4 gange langsommere
D 8 gange langsommere	E 12 gange langsommere	**F** 16 gange langsommere

1.4 Betragt nedenstående algoritme.

Algorithm 1 Løkke1(n)

```
1: for i = 1 to n do
2:     j = 1
3:     while j ≤ n/2 do
4:         print "⋆"
5:         j = j + 1
6:     end while
7: end for
```

Køretiden af algoritmen er

A $\Theta(\log n)$	B $\Theta(n)$	C $\Theta(n \log n)$	D $\Theta(n^2 \log n)$	E $\Theta(n^3)$
F $\Theta(n^2)$	G $\Theta(2^n)$	H $\Theta(n^4)$	I $\Theta(\sqrt{n})$	

1.5 Betragt nedenstående algoritme.

Algorithm 2 Løkke2(n)

```
1: for i = 1 to n do
2:   for j = 1 to n do
3:     for k = j to n do
4:       print "⋆"
5:     end for
6:   end for
7: end for
```

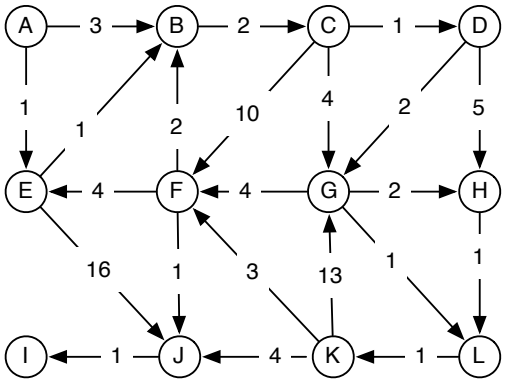
Køretiden af algoritmen er

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

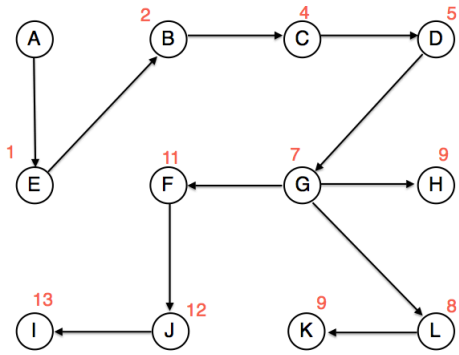
[F] $\Theta(n^2)$ [G] $\Theta(2^n)$ [H] $\Theta(n^4)$ [I] $\Theta(\sqrt{n})$

Opgave 2 (grafer)

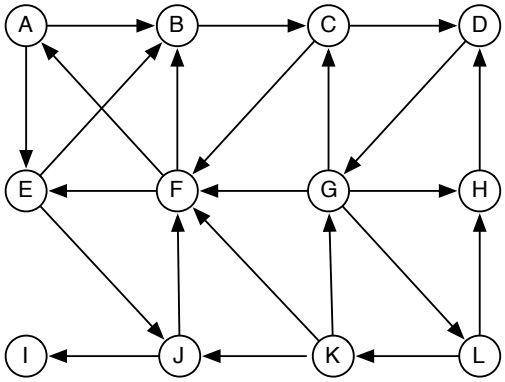
2.1 Angiv et korteste veje træ for nedenstående graf når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



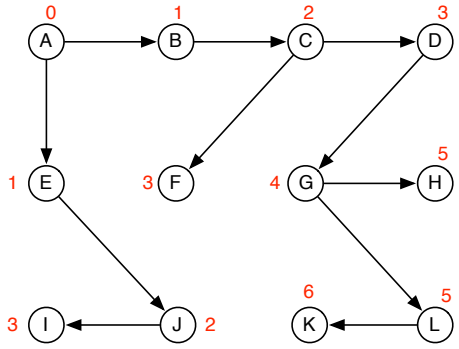
Angiv korteste veje træet og afstandene her:



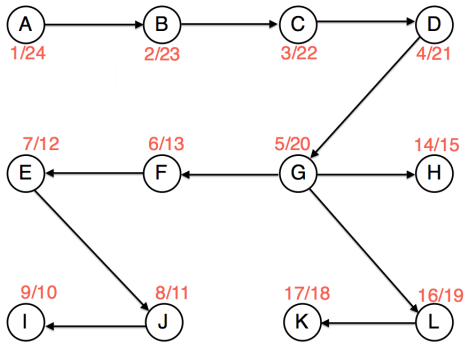
2.2 Betragt nedenstående graf G.



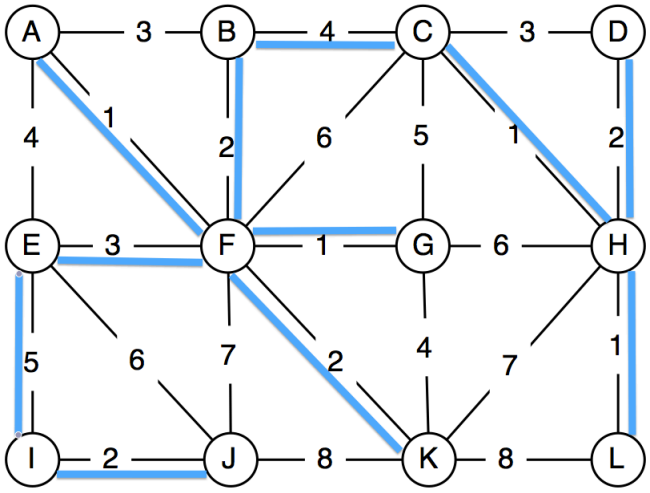
a) Angiv et BFS træ for grafen G når BFS gennemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



b) Angiv et DFS træ for grafen G, når DFS gennemløbet starter i knuden A. Angiv en DFS nummerering af knuderne. Det antages at incidenslisterne er sorteret i alfabetisk orden.



2.3 Angiv et mindste udspændende træ i nedenstående graf.



Opgave 3 (modellering og anvendelse af algoritmer/datastrukturer)

Kabeltv-firmaet AlgoNet udbyder kabeltv til alle huse i den lille by AlgoCity. Firmaet sender kabeltv fra deres hovedstation i byen ud til alle de huse der ønsker det. De har et netværk af kabler de bruger til at sende tv-signal til alle kunderne i byen. Kablerne går mellem nogle bokse. Der er en boks i alle de huse der modtager kabeltv (dvs. hos alle kunderne), og ingen bokse andre steder. Der kan godt være mange kabler tilsluttet samme boks, så der kan sendes signal både ud og ind af en boks. Der er X huse og K kabler i netværket. Firmaet kender også den præcise længde af hvert kabel.

Opgave 3.1: Rutning

Firmaet ønsker at finde ud af hvad vej de skal sende signalerne fra hovedstationen til de huse der ønsker kabeltv. Der skal være så lidt forringelse af signalet som muligt, så derfor ønsker de at hver kunde skal modtage signalet af så kort en rute som muligt (længden af en rute er den samlede længde af kablerne på ruten).

Giv en algoritme der finder den bedste måde at rute signalerne på, så hver kundes signal er så godt som muligt. Angiv køretiden af din algoritme og argumenter for at den er korrekt.

Svar Modeller problemet som en graf. Hvert hus har præcis en boks, og modelleres som en knude i grafen. Hovedstationen modelleres også som en knude i grafen. Hvert kabel forbinder to bokse (eller en boks og hovedstationen), og modelleres som en ikke-orienteret kant mellem de to tilsvarende knuder i grafen med en kantlængde lig kabelets længde. Den færdige graf har altså $O(X)$ knuder og $O(K)$ kanter.

Den korteste rute fra hovedstationen til enhver boks kan nu findes korrekt ved at anvende Dijkstras Algoritme, eftersom algoritmen garanterer at den finder den korteste sti fra en knude til alle andre knuder. Køretiden er $O(X + K \log X)$ (se slides) hvis grafen er implementeret med incidenslister og en prioritetskø implementeret med en hob. Kanterne i korteste vej træet som Dijkstras Algoritme producerer er de kanter som signalet fra hovedstationen skal rutes ad.

Opgave 3.2: Forringelse af signal i bokse

Firmaet har fundet ud af at der også sker en forringelse af signalet, når det går gennem en boks. Forringelsen af signalet når det går gennem en boks, svarer til den forringelse der sker når det løber gennem 5 meter kabel.

Giv en algoritme der finder den bedste måde at rute signalerner på, så hver kundes signal er så godt som muligt. Angiv køretiden af din algoritme og argumenter for at den er korrekt.

Svar Forringelsen af signalet sker når det går igennem en boks (og ikke når det går ind i en boks). Vi kan modellere forringelserne i boksene som en 5 meter forlængelse af de udgående kabler fra hver boks (idet det er så meget signalet bliver forringet ved at gå igennem en boks). Ved at løse problemet med disse nye kantlængder giver Dijkstras Algoritme et nyt korteste vej træ der tager højde for forringelsen i boksene.

For at kunne forlænge udgående kanter er det nødvendigt at modellere hver kabel som to orienterede kanter i stedet for en enkelt ikke-orienteret kant. Alle udgående kanter fra en boks tildeles en kantlængde der er 5 meter længere end kablets længde. Herefter anvendes Dijkstras Algoritme som tidligere fra hovedstationen. Køretiden er som før $O(X + K \log X)$ fordi antallet af kanter i grafen højst fordobles.

Opgave 3.3: Vedligeholdelse

Firmaet skal spare, og ønsker derfor ikke længere at vedligeholde hele kabelnettet. De vil derfor gerne finde en mængde af kabler der er billigst mulig at vedligeholde, men som samtidig forbinder alle nuværende kunder. Prisen for at vedligeholde et kabel er proportional med længden af kablet, dvs. det er den samlede længde af kablerne der afgør hvor dyrt det er at vedligeholde dem.

Giv en algoritme der finder den billigste mængde af kabler at vedligeholde. Angiv køretiden af din algoritme og argumenter for at den er korrekt.

Svar Betragt som før problemet som en graf hvor husene og hovedstationen er knuder og hvert kabel modelleres som en ikke-orienteret kant i grafen med en kantlængde svarende til kablets længde. Firmaet ønsker nu at minimere den samlede længde af kabler.

Problemet kan løses ved at finde det mindste udspændede træ i grafen, hvor vægten af en kant er lig med kantlængden (og dermed kablets længde). Det mindste udspændende træ er netop det træ der forbinder alle knuder (så alle kunder er forbundet), og hvor summen af kantvægtene er minimeret (svarende til at den samlede længde af kabler er kortest mulig). Hvis kun de kanter der er med i det mindste udspændende træ vedligholdes minimeres vedligeholdessesudgiften dermed.

Hvis det mindste udspændende træ findes med Prims Algoritme er køretiden $O(K \log X)$ hvis grafen er implementeret med incidenslister og en prioritetskø implementeret med en hob.

Opgave 4 (Træer)

Denne opgave omhandler 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 vægt; knuden x har vægten $weight[x]$.

4.1 Ens vægte

Giv en effektiv algoritme EnsVægt(x) der givet rodknuden til et vægtet binært træ returnerer 1 hvis der er mindst to knuder i træet der har samme vægt og 0 ellers.

Angiv køretiden af din algoritme og argumenter for at den er korrekt.

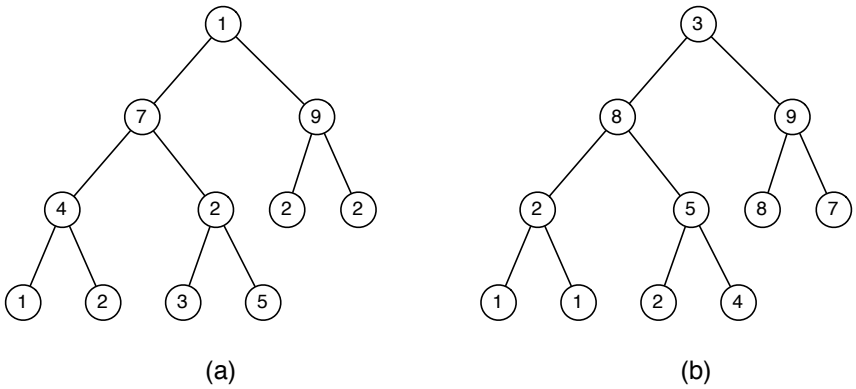
Svar Lad algoritmen gennemløbe alle knuder i træet en gang ved brug af DFS. Vedligehold desuden en datastruktur D der supporterer Search og Insert operationerne over knudernes vægte. For hver knude x der besøges i træet tjekkes først om vægten allerede findes i D med Search($weight[x]$). Hvis vægten findes returnerer algoritmen 1. Ellers indsættes vægten i D med Insert($weight[x]$). Når gennemløbet af knuderne er afsluttet returneres 0.

Algoritmen returnerer 1 første gang en knude tjekkes hvor en knude med samme vægt tidligere er blevet tjekket. Hvis der ikke er sådanne kollisioner returneres 0. Køretiden for DFS er $O(|V|)$ (hvor V er sættet af knuder i træet), og køretiden for Search og Insert operationerne er afhængig af hvilken datastruktur D der vælges. Bemærk at der laves $O(|V|)$ Search og Insert operationer. Anvendes som D et balanceret binært søgetræ tager D-operationerne $O(\log |V|)$ tid per styk, så den samlede køretid er $O(|V| \log |V|)$ i worst case. Anvendes i stedet som D en dynamisk hægtet hashtabel med en universel hashfunktion tager D-operationerne $O(1)$ forventet tid per styk, så den samlede forventede køretid for algoritmen er $O(|V|)$.

4.2 Vægtbalancerede træer

To knuder i et træ er søskende, hvis de har samme forælder. Dvs. $left[x]$ og $right[x]$ er søskende. Vi siger at en knude v i et vægtet binært træ er *vægtbalanceret* hvis vægten af v højst er dobbelt så meget som vægten af dens søskende. Et træ er vægtbalanceret hvis alle knuder i træet er vægtbalancerede.

4.2.1 Hvilke af nedenstående træer er vægtbalancerede?



Svar (a) er vægtbalanceret. I (b) findes et søskendepar med vægt 2 og 5, som strider mod definitionen.

4.2.2 Skriv pseudokode for en *rekursiv* algoritme VægtBalance(x), der givet rodknuden til et vægtet binært træ der returnerer 1 hvis træet er vægtbalanceret og 0 ellers.

Angiv køretiden af din algoritme og argumenter for at den er korrekt.

Svar Algoritmen lader en knude x bestemme om børnene er vægtbalancerede. Eftersom et blad ikke har nogen børn er de altid vægtbalancerede. I alle andre tilfælde sammenlignes børnenes vægt, og der returneres 0 hvis det ene barn vejer mere end dobbelt så meget som det andet. Er det ikke tilfældet køres algoritmen rekursivt på børnene, og den mindste returværdi fra disse returneres. Da der kun returneres 0 eller 1 vil det altid være en af disse værdier der returneres fra algorimen, og da den mindste værdi returneres i de rekursive kald vil 0 blive returneret netop hvis der findes et ikke-vægtbalanceret søskendepar.

Fordi hver knude kun tjekkes en gang og der kun udføres et konstant antal sammenligninger per tjek er køretiden af algoritmen $O(|V|)$, hvor V er antallet af knuder i træet.

Algorithm 3 VægtBalance(x)

```
1: if (left[x] = NIL) then
2:    return 1
3: else
4:    if (max{weight[left[x]], weight[right[x]]} > 2 * min{weight[left[x]], weight[right[x]]}) then
5:       return 0
6:    else
7:       return min{VægtBalance(left[x]), VægtBalance(right[x])}
8:    end if
9: end if
```

Opgave 5 (datastrukturer)

5.1 Lad K være en kø. Udfør følgende operationer fra venstre til højre: et bogstav i betyder *Enqueue(K,i)* og $*$ betyder *Dequeue(K)*.

D * T U * * I N * F O R * M * A T I K

Angiv sekvensen af bogstaver der bliver "dequeue"et (returneret af *Dequeue(K)*) af disse operationer:

A D U T I R M

B D U T N R M K I T A O F

C D U T N R M

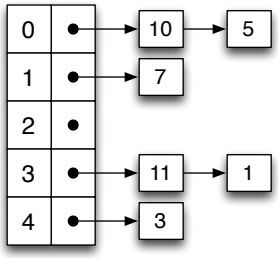
D D T U I N F

E D U T N R M I T A O F

F D U T N O M

5.2 Lad H være en hægtet hashtabel (chained hashing) af størrelse 5 med hashfunktion $h(x) = 3x \mod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 5, 1, 11, 7, 10, 3.

Svar Hashtabellen ser ud som følger. Venstre række i tabellen angiver indexet. Pilene er pointere til det næste element i den hægtede liste. Hvis der ikke er en pil videre fra et felt betyder det at at pointeren er NIL.

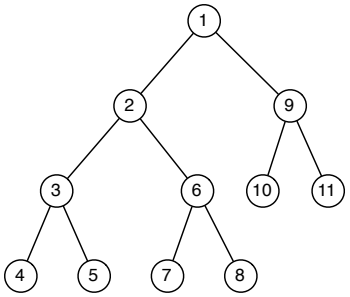


5.3 Lad H være en hashtabel med linær probering (linear probing) af størrelse 5 med hashfunktion $h(x) = x \mod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 2, 7, 16.

Svar Hashtabellen ser ud som følger. Øverste række i tabellen angiver indexet, nederste angiver tallene der er indsat. Bemærk at der ikke er noget element på index 0.

0	1	2	3	4
	6	2	7	16

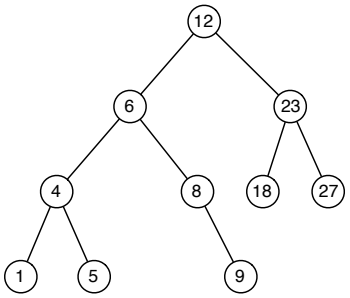
Opgave 5.4 Angiv den rækkefølge knuderne bliver skrevet ud i, når man laver et inorder gennemløb af nedenstående træ.



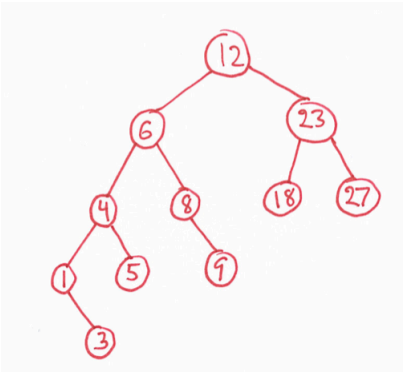
Inorder gennemløb: $4, 3, 5, 2, 7, 6, 8, 1, 10, 9, 11$

5.5 Denne opgave omhandler (ubalancerede) binære søgetræer, som beskrevet i de udleverede noter CLRS kapitel 12.

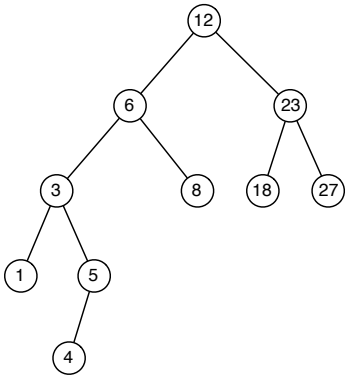
Opgave a Angiv hvordan det binære søgetræ nedenfor ser ud efter indsættelse af et element med nøgle 3.



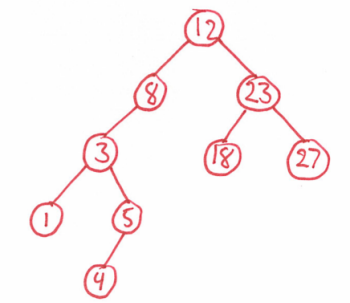
Svar



Opgave b Angiv hvordan det binære søgetræ nedenfor ser ud efter sletning elementet med nøgle 6.



Svar



Danmarks Tekniske Universitet

Skriftlig prøve, den 29. maj 2013.

Kursusnavn: Algoritmer og datastrukturer I Kursus nr. 02105.

Hjælpemidler: Skriftlige hjælpemidler. Det er **ikke** tilladt at medbringe lommeregner.

Varighed: 4 timer

Vægtning af opgaverne: Opgave 1 - 24%, Opgave 2 - 16%, Opgave 3 - 24%, Opgave 4 - 16%, 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 benytte ekstra papir som så vedlægges opgavebesvarelsen.

Vejledende løsning

Opgave 1 (kompleksitet)

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

	Ja	*Nej*
$n(n+1)/100 = O(n^3)$	[x]	☐
$n(n+1)/2 + \log n = O(n^2)$	[x]	☐
$\frac{n}{\log n} = \Theta(n)$	☐	[x]
$2^n = O(n)$	☐	[x]
$n + n(n-1)/5 = \Theta(n^2)$	[x]	☐

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 $f(n) = O(g(n))$.

$n^2 \log n$

$\frac{n^3}{10000}$

$\frac{n^3}{\log n} + 100n + 5000n^2$

$2^{\log_2 n}$

$2^4 \cdot n^{1/4}$

$2n!$

Svar: $2^4 \cdot n^{1/4}, \quad 2^{\log_2 n}, \quad n^2 \log n, \quad \frac{n^3}{\log n} + 100n + 5000n^2, \quad \frac{n^3}{10000}, \quad 2n!$

1.3 Betragt nedenstående algoritme, der som input tager et array med n heltal.

Algorithm 1 Løkke1(A[1 . . . n])

```
1: for i = 1 to n − 1 do
2:    for j = i + 1 to n do
3:       if A[i]=A[j] return false
4:    end for
5: end for
6: Return true
```

Forklar hvad algoritmen beregner/gør:

Algoritmen returnerer true hvis alle tal i rækken er forskellige og false ellers.

Køretiden af algoritmen er

[A] $\Theta(\log n)$ [B] $\Theta(n)$ [C] $\Theta(n \log n)$ [D] $\Theta(n^2 \log n)$ [E] $\Theta(n^{1.5})$

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

1.4 Antag at du har en algoritme hvis køretid er præcist $7n^3$. Hvor meget langsommere kører algoritmen hvis du fordobler inputstørrelsen?

A dobbelt så langsom | B 3 gange langsommere | C 7 gange langsommere

D 8 gange langsommere | E 14 gange langsommere | F 4 gange langsommere

1.5 Betragt nedenstående algoritme.

Algorithm 2 Løkke2(n)

```
1: if n > 0 then
2:    for i = 1 to n do
3:       print "⋆"
4:    end for
5:    Løkke2(n − 1)
6: end if
```

a) Hvor mange stjerner udskriver algoritmen Løkke2 når den bliver kaldt med parameteren $n = 4$: **10**

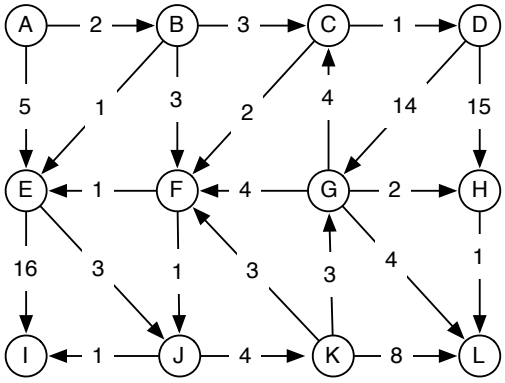
b) Køretiden af algoritmen er

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

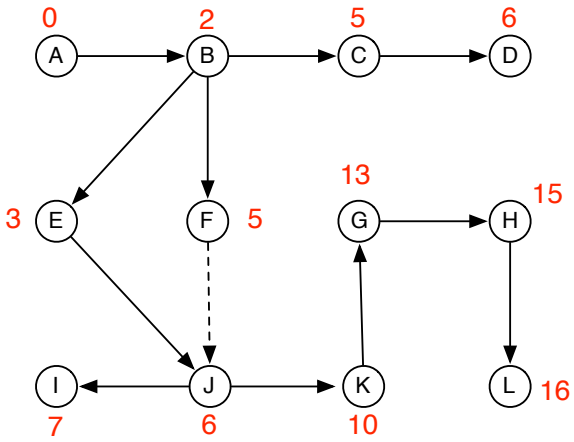
F $\Theta(n^2)$ | G $\Theta(2^n)$ | H $\Theta(n^4)$ | I $\Theta(\sqrt{n})$

Opgave 2 (grafer)

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

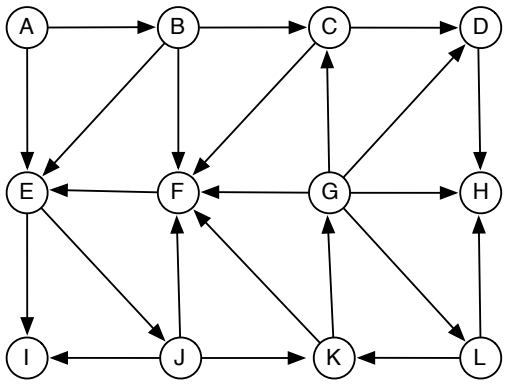


Angiv korteste veje træet og afstandene her:

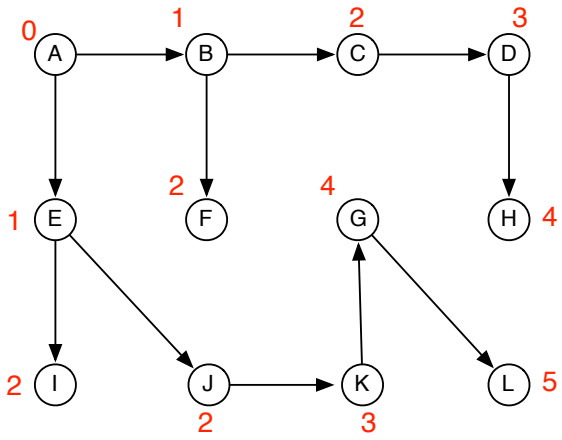


Den stiplede kant er ikke med i i korteste veje træet. Men kanten fra E til J kan erstattes med den stiplede kant.

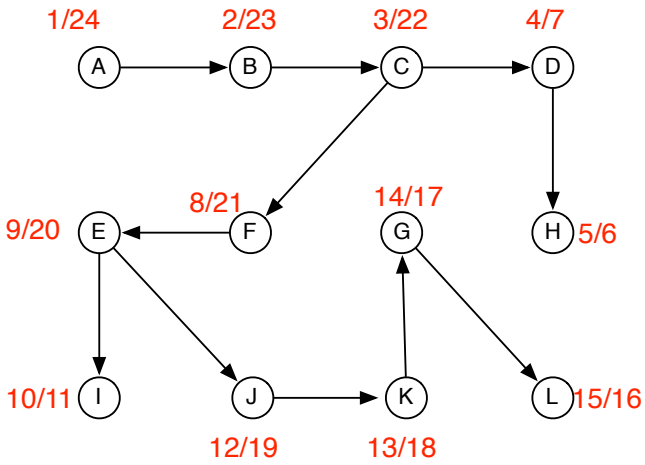
2.2 Betragt nedenstående graf G.



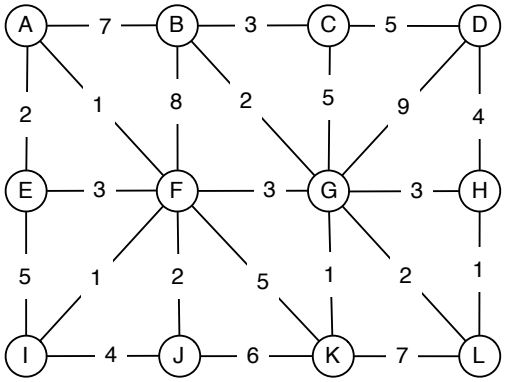
a) Angiv et BFS træ for grafen G når BFS gennemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



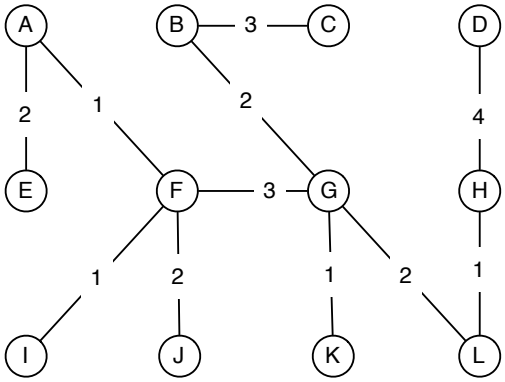
b) Angiv et DFS træ for grafen G, når DFS gennemløbet starter i knuden A. Angiv *discovery time* og *finishing time* for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



2.3 Angiv et mindste udspændende træ i nedenstående graf.

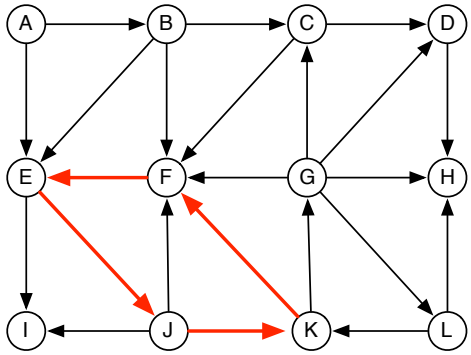


Angiv det mindste udspændende træ og værdien af det her:



Værdi: **22**

2.4 Er nedenstående graf en DAG (*directed acyclic graph*)? Hvis ja, så angiv en topologisk sortering af knuderne. Hvis nej, så forklar hvorfor.



Grafen er ikke en DAG, da den indeholder kredse (se kreds markeret med rødt).

Opgave 3 (modellering og anvendelse af algoritmer/datastrukturer)

Rejseselskabet ØerЯos planlægger rejser til øgruppen Algo. Nogle af øerne er forbundet med broer, andre må man sejle imellem. Der er X øer, B broer, og F færger. Hver færge sejler frem og tilbage mellem to øer.

Opgave 3.1: Rejser for søsyge

Da nogle af rejseselskabets kunder nemt bliver søsyge, ønsker firmaet at finde det største antal øer man kan besøge uden at skulle sejle imellem dem. På den måde kan de rejsende nøjes med at sejle til den første ø i gruppen og derefter komme til de andre øer via broerne.

Giv en effektiv algoritme der finder det største antal øer man kan besøge uden at sejle imellem dem.

Angiv køretiden af din algoritme i asymptotisk notation (ved hjælp af parametrene X, B og F) og argumenter for at den er korrekt.

Svar Problemet kan modelleres som en uorienteret graf. Øer er knuder og broerne er kanter. Der er en knude for hver ø. Der er en kant mellem to knuder, hvis der er en bro mellem de to øer som knuderne svarer til. Der er X øer og B kanter.

Problemet svarer nu til at finde den største sammenhængskomponent i grafen. Dette kan gøres ved hjælp af enten BFS- eller DFS-algoritmen. Kør BFS (eller DFS) fra en knude. Modificer algoritmen så den har en tæller, der tælles op hver gang en ny knude mødes. Til sidst returneres antallet af mødte knuder, dvs. antallet af knuder i den sammenhængskomponent.

Hvis der stadig er knuder i grafen der ikke er besøgt, så kør BFS (eller DFS) igen fra en af disse (med tælleren nulstillet). Bliv ved med at køre BFS (DFS) indtil alle knuder er besøgt. Nu kendes størrelserne af alle sammenhængskomponenterne og det største tal returneres.

BFS-algoritmen tager $O(X+B)$ tid. Modifikationen ændrer ikke på køretiden, da tælleren kan opdateres i konstant tid i hver iteration (det samme gælder DFS).

Opgave 3.2: Billigste færgeture

Andre af rejseselskabets kunder ønsker at besøge en gruppe af Y øer, hvor der ikke er nogle broer imellem. Det er muligt at besøge alle øerne ved at tage færger imellem dem. Firmaet kender prisen for alle færgeture. Alle billetterne til færgerne er returbilletter. Dvs. man betaler for både at sejle frem og tilbage når man køber en færgebillet. Der er G færgeruter. Firmaet vil gerne tilbyde den billigste rejse, så de vil gerne minimere den pris man betaler for færgebilletter.

Giv en algoritme der finder ud af hvilke færgebilletter der skal købes for at alle Y øer kan besøges, og så den samlede pris for færgebilletter bliver mindst mulig. Angiv køretiden af din algoritme i asymptotisk notation (ved hjælp af parametrene Y og G) og argumenter for at den er korrekt.

Svar Problemet modelleres som en vægtet uorienteret graf. Øer er knuder og færgertuerne er kanter. Der er en knude for hver ø. Der er en kant mellem to knuder, hvis der er en færgerute mellem de to øer som knuderne svarer til. Vægten af kanten er prisen for færgebilletten. Der er Y øer og G kanter. Da priserne er for en returbillet, svarer problemet til at finde det mindste udspændende træ i grafen. Når man har fundet et mindste udspændende træ kan man besøge alle øer ved at følge en rute der svarer til et trægennemløb af træet. Hver kant besøges kun 2 gange (en gang i hver retning) og det svarer til at der betales for netop en returbillet for hver færgerute i det mindste udspændende træ. Prisen for færgebilletter i alt bliver derfor vægten af det mindste udspændende træ. Denne kan findes i lineær tid ($\mathrm{O}(Y)$ tid) ved et trægennemløb når først MST er fundet.

Det er ikke muligt at gøre det billigere end prisen for et mindste udspændende træ, da enhver rute der besøger alle øerne må være sammenhængende, og et mindste udspændende træ er den billigste sammenhængende delgraf i grafen.

Hvis Prims algoritme bruges bliver køretiden $O(G \log Y)$.

Opgave 3.3: Billigste tur til alle øer

Som i forrige opgave ønsker firmaet at minimere den pris man betaler for færgebilletter. Men nu er der broer mellem nogle af øerne. Da broerne er gratis er der ingen grund til at tage en færge mellem to øer hvis der er en bro. Der er X øer, B broer, og F færger.

Giv en algoritme der finder ud af hvilke færgebilletter der skal købes for at alle X øer kan besøges, og så den samlede pris for færgebilletter bliver mindst mulig. Angiv køretiden af din algoritme i asymptotisk notation (ved hjælp af parametrene X, B og F) og argumenter for at den er korrekt.

Svar Problemet modelleres som en graf G, hvor alle øer er knuder og kanterne svarer til broer og færgeruter. Kanterne der svarer til broer har vægt 0, og kanterne der svarer til færgeruter har vægt lig med prisen for billetten. Dvs. der er en kant mellem to knuder hvis der er en bro eller færgerute imellem dem.

Problemet kan nu løses ved hjælp af algoritmen fra opgave 3.2.

Grafen G har X knuder og højst $F + B$ kanter. Bruges Prims algoritme til at finde MST i G tager $O((F + B) \log X)$ tid.

Alternativt svar Problemet modelleres som en graf G_1, hvor alle øer er knuder og kanterne svarer til broer som i opgave 3.1. Alle sammenhængskomponenter findes som i opgave 3.1. Der laves nu en ny graf G_2, hvor knuderne er sammenhængskompenterne i G_1. Kanterne i G_2 er færgeruterne. Der er en kant mellem to knuder u og v hvis der er en færgerute mellem en ø i u og en ø i v. Vægten af kanten er prisen for den billigste sådanne færgerute. Problemet kan nu løses ved hjælp af algoritmen fra opgave 3.2.

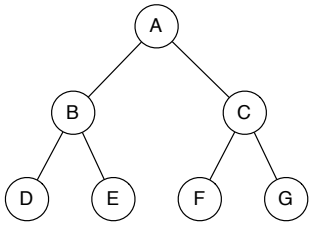
At finde alle sammenhængskomponenterne i G_1 tager $O(X + B)$ tid. Når sammenhængskomponenterne er fundet tager det $O(X + F)$ tid at konstruere G_2. Grafen G_2 har højst X knuder og højst F kanter. Bruges Prims algoritme til at finde MST i G_2 tager denne del $O(F \log X)$ tid.

I alt bliver køretiden: $O(X + B) + O(X + F) + O(F \log X) = O(X + B + F \log X)$.

Opgave 4 (Træer)

4.1 Trægennemløb

Betragt nedenstående træ.



Angiv hvilke af følgende sekvenser af bogstaver der bliver udskrevet ved et preorder, inorder og postorder gennemløb af ovenstående træ.

[1] A B C D E F G [2] G F E D C B A [3] A B D E C F G

[4] D B E A F C G [5] D E B F G C A [6] C B D A F E G

Preorder: **3**

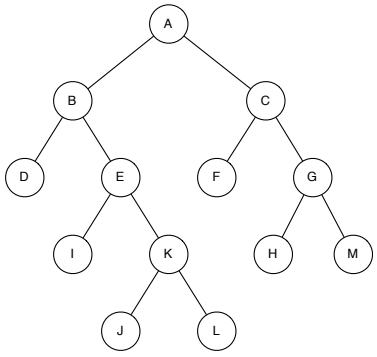
Inorder: **4**

Postorder: **5**

4.2 Korteste sti

Denne opgave omhandler 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.

4.2.1 Betragt nedenstående træ.



Hvad er længden af den korteste rod-til-blad sti i træet: **2**

Hvad er længden af den længste rod-til-blad sti i træet: **4**

4.2.2 Giv en rekursiv algoritme KortesteSti(x) der givet rodknuden til et binært træ returnerer længden af den korteste rod-til-blad sti i træet. Angiv gerne algoritmen i pseudokode. Angiv køretiden af din algoritme i asymptotisk notation og argumenter for at den er korrekt.

Svar

Algorithm 3 Træ(x)

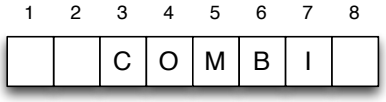
```
1: if (x = NIL or left[x] = NIL) then
2:     return 0
3: else
4:     return min{Træ(left[x]), Træ(right[x])}+1
5: end if
```

Algoritmen finder længden af den korteste rod-til-blad sti i børnenes deltræer, tager minimum af de to og lægger en til svarende til kanten fra knuden x til barnet. Hvis det er et tomt træ returneres 0. Hvis x er et blad returneres også 0.

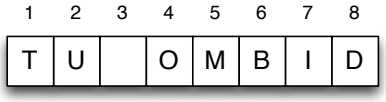
Køretiden af algoritmen er $\Theta(n)$ hvor n er antallet af knuder i træet. Hvert kald tager konstant tid og der kaldes kun en gang for hver knude.

Opgave 5 (datastrukturer)

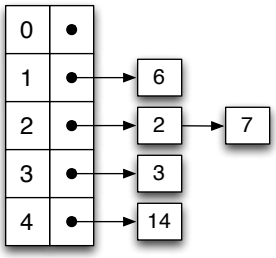
5.1 Betragt nedenstående kø K implementeret ved hjælp af en tabel (et array). K.head = 3 og K.tail = 8.



Angiv hvordan køen ser ud efter følgende operationer: Enqueue(D), Enqueue(T), Dequeue(), Enqueue(U).



5.2 Lad H være en hægtet hashtabel (chained hashing) af størrelse 5 med hashfunktion $h(x) = x \mod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 7, 3, 14, 2

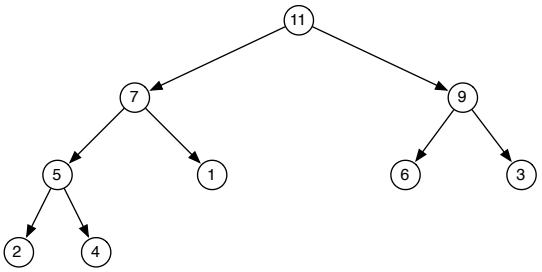


5.3 Lad H være en hashtabel med linær probering (linear probing) af størrelse 5 med hashfunktion $h(x) = x \mod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 7, 3, 14, 2

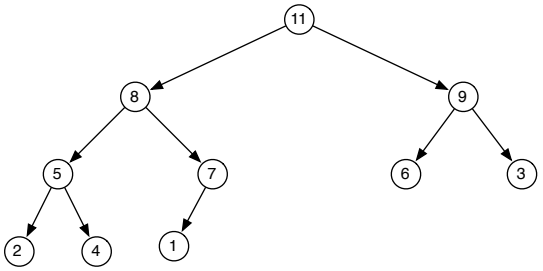
0	2
1	6
2	7
3	3
4	14

5.5 Denne opgave omhandler binære max-hobe.

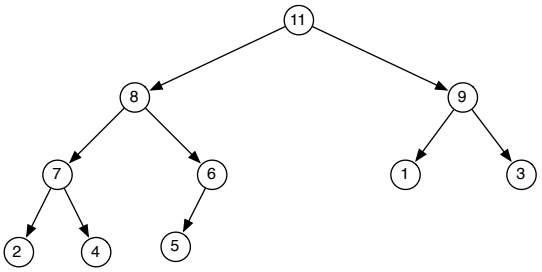
Opgave a Angiv hvordan den binære hob nedenfor ser ud efter indsættelse af et element med nøgle 8.



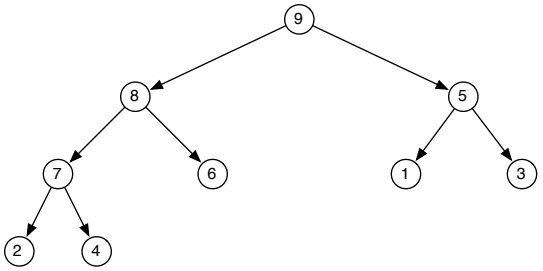
Svar



Opgave b Angiv hvordan den binære hob nedenfor ser ud efter én Extract-Max operation.



Svar



Sample/practice exam 2016, questions

Algoritmer og datastrukturer (Danmarks Tekniske Universitet)

Technical University of Denmark

Example exam.

Course name: Algorithms and data structures.

Course number: 02110.

Aids allow: All written materials are permitted.

Exam duration: 4 hours

Weighting: Question 1: 30% - Question 2: 15% - Question 3: 20% - Question 4: 10% - Question 5: 15% - Question 6: 10%.

The weighting is only an approximative weighting.

All questions should be answered by filling out the room below the question. As exam paper just hand in this and the following pages filled out. If you need more room you can use extra paper that you hand in together with the exam paper.

Question 1

1.1 Which of the following statements are correct: :

A The worst-case running time of the Jarvis' March algorithm is $O(n^2)$.
B Jarvis' March is always faster than Graham's scan.
C If the points are sorted by polar angles then Jarvis' march runs in linear time.
D Graham's scan is always faster than Jarvis' March.

1.2 Which of the following statements are true: :

A A subtree of a red-black tree is itself a red-black tree (except the root might be red).
B The sibling of a leaf node is either a leaf or red.
C the *longest* simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the *shortest* path from x to a descendant leaf.

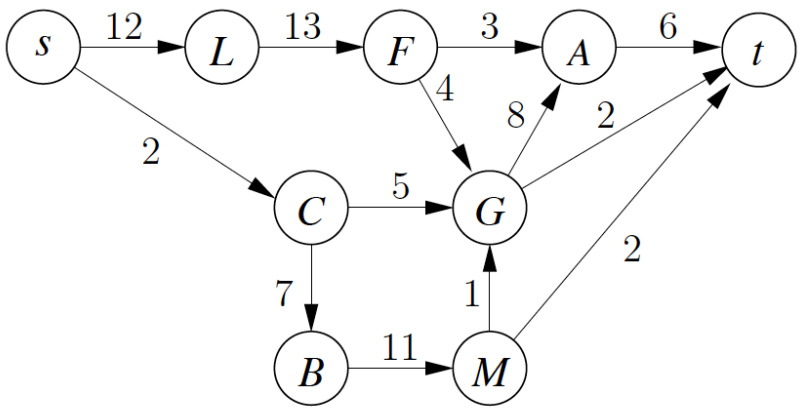
1.3 Draw the compressed trie for the strings `this`, `that`, `hat`, `thing`, `fat` (don't replace the labels by indexes into the string, just write the labels on the edges):

1.4 Draw the finite string matching automata for the string `hejhejsa`:

1.4 Show the red-black tree that results from inserting the keys 41, 38, 31, 12, 19 , 8 in this order into an initially empty tree.

Question 2 (flow)

Consider the network below with capacities on the edges.



Question a Give a maximum flow from s to t in the network (write the flow for each edge along the edges on the graph above), give the value of the flow, and give a minimum $s - t$ cut (give the partition of the vertices).

value of flow: _____

minimum cut: _____

Question b Use Edmonds-Karp's algorithm to compute a maximum flow on the two graph. For each augmenting path write the nodes on the path and the value you augment the path with in the table below.

augmenting path	value

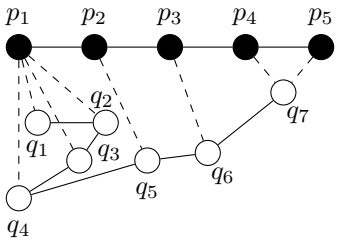
Question 3

Consider Professor Bille going for a walk with his personal zombie. The professor follows a path of points $p_1, \ldots, p_n$ and the zombie follows a path of points $q_1, \ldots, q_m$. We assume that the walk is partitioned into a number of small steps, where the professor and the zombie in each step either both move from p_i tp p_{i+1} and from q_j to q_{j+1}, respectively, or only one of them moves and the other one stays.

The goal is to find the smallest possible length L of the leash, such that the professor and the zombie can move from p_1 and q_1, resp., to p_n and q_n. They cannot move backwards, and we only consider the distance between points. The distance L is also known as the discrete Fréchet distance.

We let $L(i, j)$ denote the smallest possible length of the leash, such that the professor and the zombie can move from p_1 and q_1 to p_i and q_j, resp. For two points p and q, let $d(p, q)$ denote the distance between the them.

In the example below the dotted lines denote where Professor Bille (black nodes) and the zombie (white nodes) are at time 1 to 8. The minimum leash length is $L = d(p_1, q_4)$.



Q1: Recurrence Give a recursive formula for $L(i, j)$.

Q2: Algorithm Give an algorithm that computes the length of the shortest possible leash. Analyze space and time usage of your solution.

Q3: Print solution Extend your algorithm to print out paths for the professor and the zombie. The algorithm must return where the professor and the zombie is at each time step. Analyze the time and space usage of your solution.

Question 4

At the halloween party at a well-known academic institution north of Copenhagen not all went smooth and some students had to be taken to medical emergency treatment at *Rigshospitalet*. In total 150 had to get a transfusion of one bag of blood. The hospital had 155 bags in stock. The distribution of blood groups in the supply and amongst the students is shown in the table below.

Blood type	A	B	0	AB
Bags in stock	44	31	42	38
Demand	37	33	40	40

Type **A** patients can only receive blood of type **A** or type **0**; type **B** patients can receive only type **B** or type **0**; type **0** patients can receive only type **0**; and type **AB** patients can receive any of the four types.

Model the problem as a flow network problem. Draw the corresponding network, and interpret the meaning of the nodes, and edges (edges capacities). Describe how to check whether every student can get a transfusion, otherwise how many can get one.

You do not have to solve the problem explicitly. Remember to argue that your algorithm is correct.

The exam set continues on the next page

Question 5

In chemical databases for circular molecules, each molecule is represented by a circular string of chemical characters. To allow faster lookup and comparisons of molecules, one wants to store each circular string by a canonical linear string. A natural choice for a canonical in ear strings the one that is lexicographically smallest. That gives the following computational problem.

Assume we are given a string $T = x_1 \dots x_n$ of length n. A *shift* of T by s, $0 \leq s < n$, is the string $T^s = x_{s+1}x_{s+2} \dots x_n x_1 x_2 \dots x_s$. In this problem we want to find the *lexicographically smallest shift*, i.e. the shift s where T^s is lexicographically smallest among $T^0, \dots, T^{n-1}$. Eg. $T^2 = T^7 = \texttt{aababaabab}$ are the lexicographically smallest shifts of the string

$$T = \texttt{abaababaab}$$

Question a State all s where T^s is a lexicographically smallest shift of the string

$$T = \texttt{bcabaabcabaabcabaa}$$

Question b Describe an algorithm that given a string T of length n over an alphabet of size $O(1)$ computes all s where T^s is a lexicographically smallest shift of T. State the algorithms running time.

The exam set continues on the next page

Question 6

Let S be a set of n line segments in the plane. Give an algorithm to compute the convex hull of S. Analyze the time complexity of your algorithm and argue it is correct.

Technical University of Denmark

Written examination, May 17, 2017.

Course name: Algorithms and Data Structures 1

Course number: 02105

Aids: Written aids. It is **not** permitted to bring a calculator.

Duration: 4 hours.

Weights: Exercise 1 - 24 %, Exercise 2 - 20 %, Exercise 3 - 16 %, Exercise 4 - 15 %, Exercise 5 - 25 %. The weights are approximate. The grade is given based on an overall assessment.

All exercise should be answered by filling out the areas below the description. As your solution to the exam, just hand in the page and the following pages. If you need more space, you may use extra pieces of paper and hand these in along with your solution.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

Name: _____

Student ID: _____

1 Complexity

1.1 (6 %) For each statement below, mark whether or not it is correct:

	Yes	No
$\frac{1}{4}n^3 + n^2 \log n + 17 \cdot n^2 = \Theta(n^3)$	☐	☐
$10^{17} + (n^3)^2 + \log^2 n = O(n^5)$	☐	☐
$42 \log n + \sqrt{n} + n^{1/3} = \Omega(\sqrt{n})$	☐	☐
$2^{\log n} + \frac{n^{1,5}}{30^{30}} + \log^{10} n = \Theta(n)$	☐	☐
$(3 \log^2 n + 55 \log(n^{10}) + 8 \log n) \cdot \log n = \Omega(\log^{10} n)$	☐	☐

1.2 (6 %) Arrange the following functions in increasing order according to asymptotic growth. That is, if $g(n)$ immediately follows $f(n)$ in your list, it must hold that $f(n) = O(g(n))$.

$$8\sqrt{n} \qquad \log(2^n) \qquad 7n^3 \qquad 1^n \qquad 3\log^2 n \qquad n^{4/2}$$

Solution:

1.3 (12 %) State the running time for each of the following algorithms. Write your answer in O-notation as a function of n.

```
ALG1(n)
c = 0
for i = 1 to ⌊n/3⌋ do
  c = c + 1
end for
for j = 1 to ⌊n/5⌋ do
  c = c + 1
end for
```

Solution:

```
ALG2(n)
if n ≤ 1 then
  return 1
else
  return 1+ALG2(n − 2)
end if
```

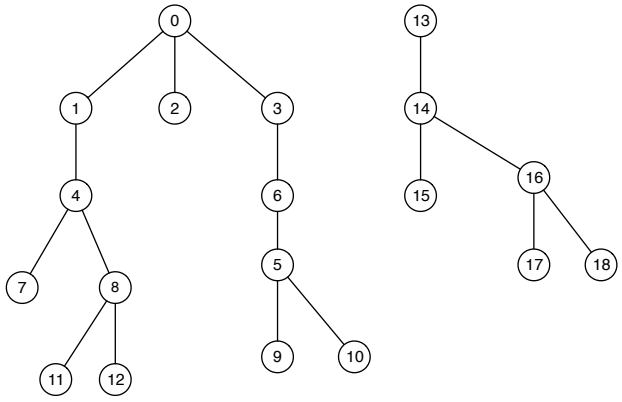
Solution:

```
ALG3(n)
for j = 1 to n do
  i = n
  while i ≥ 3 do
    i = ⌊i/2⌋
  end while
end for
```

Solution:

2 Data Structures

Consider the following forest of trees representing a family of sets in a union find data structure constructed using the quick union algorithm.



2.1 (4 %) State the results of the following FIND(·) operations.

FIND(2) : _____ FIND(8) : _____

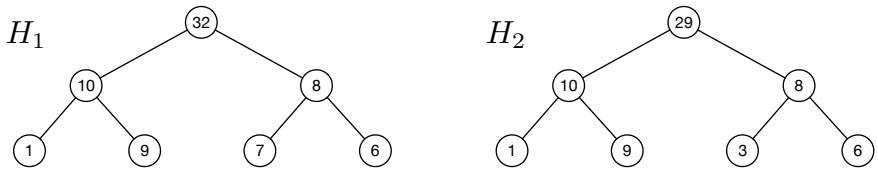
FIND(14) : _____ FIND(12) : _____

FIND(0) : _____ FIND(17) : _____

2.2 (4 %) Assume that we now use path compression on the forest above. Show the forest of trees after a FIND(8) operation.

Solution:

2.3 (4 %) Consider the following max heaps H_1 and H_2.



Draw H_1 after an INSERT operation with key 12.

Solution:

2.4 (4 %) Draw H_2 after an EXTRACT-MAX operation.

Solution:

2.5 (4 %) We want to support the DELETE-MAX() operation on each of the following data structures. DELETE-MAX() should remove the largest element in the data structure and then reestablish the data structure over the remaining elements. As an example, if the data structure is a min heap with elements $\{6, 32, 18, 7, 2\}$, DELETE-MAX() removes element 32 and re-establishes a min heap with the elements $\{6, 18, 7, 2\}$. For each of the data structures, state the time needed to perform a DELETE-MAX() operation in O-notation as a function of n, where n is the number of elements in the data structure.

Singly-linked list: _____

Doubly-linked list: _____

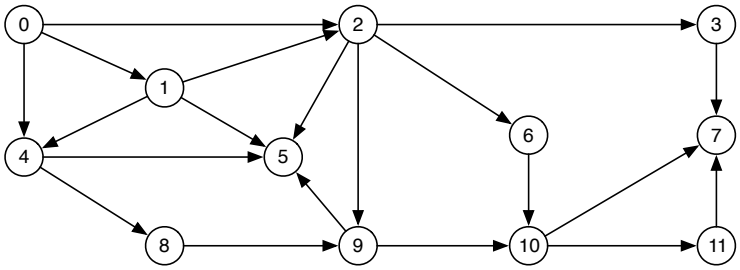
Max heap: _____

Min heap: _____

Binary search tree: _____

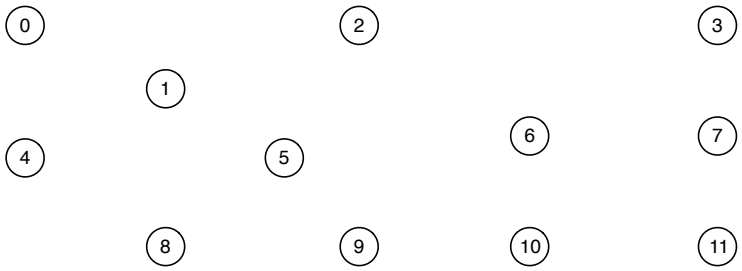
3 Graphs

Consider the following graph G.



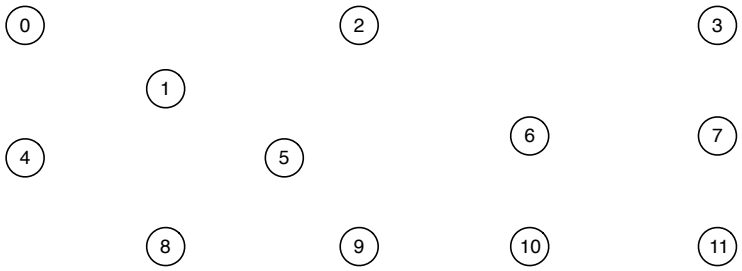
3.1 (4 %) Show the DFS tree for G when starting in node 0 and state the discovery and finish times for each node. Assume that the adjacency lists are sorted in increasing order.

Solution:

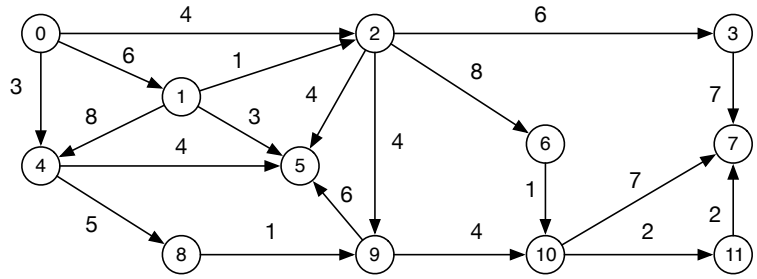


3.2 (4 %) Show the BFS tree for G when starting in node 0 and state the distance for each node. Assume that the adjacency lists are sorted in increasing order.

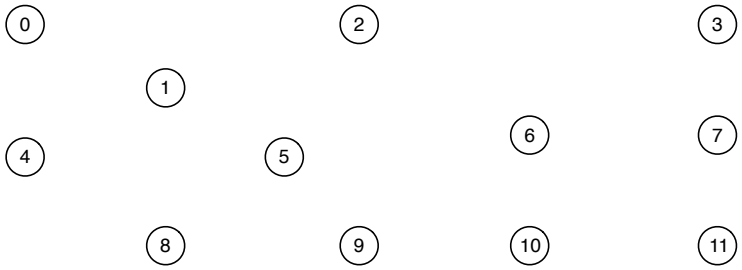
Solution:



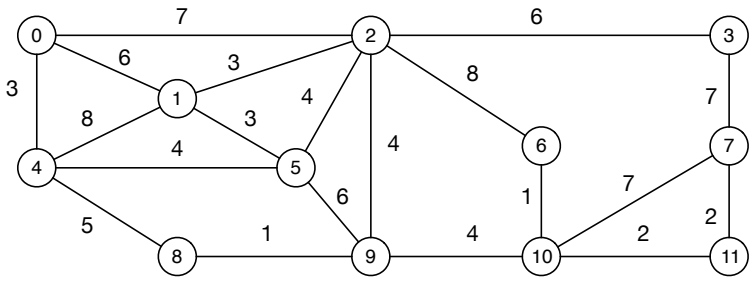
3.3 (4 %) Consider the graph below. Show a shortest path tree for the graph starting at node 0. State the length of the shortest path from node 0 to each node.



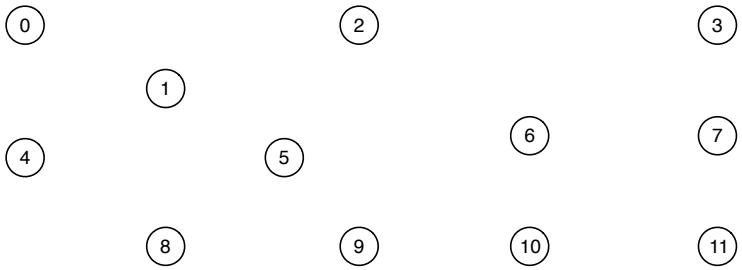
Solution:



3.4 (4 %) Consider the graph below. Show a minimum spanning tree and state the total weight of the tree.



Solution:



Total weight: _____

4 Trees

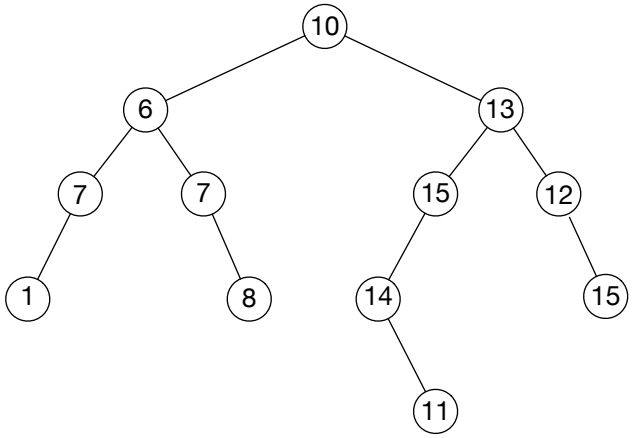
This exercise is about rooted binary trees. Any node x has fields $x.parent$, $x.left$ and $x.right$ denoting the parent, left child, and right child of x, respectively. The root r has $r.parent = null$. Furthermore, any node has a *weight* given by the field $x.weight$.

4.1 (1 %) Let x be a node in the tree. The node x is *skewed* according to the following conditions.

- If x is a leaf then x is skewed.
- If x is an internal node with two children then x is skewed if and only if $x.left.weight < x.weight < x.right.weight$.
- If x is an internal node with a left child but no right child then x is skewed if and only if $x.left.weight < x.weight$.
- If x is an internal node with a right child but no left child then x is skewed if and only if $x.weight < x.right.weight$.

Consider the following tree. Mark all skewed nodes in the tree with a cross.

Solution:



4.2 (6 %) Give an algorithm, SKEWED(X), that given a node x returns true if and only if the node is skewed. Write your algorithm in pseudocode and analyze the running time of your algorithm as a function of n, where n is the number of nodes in the tree.

Solution:

4.3 (8 %) Give a recursive algorithm, SKEWCOUNT(x), that, given the root node, computes the number of skewed nodes in the tree. Write your algorithm in pseudocode and analyze the running time of your algorithm as a function of n, where n is the number of nodes in the tree.

Solution:

5 Super Mario Run

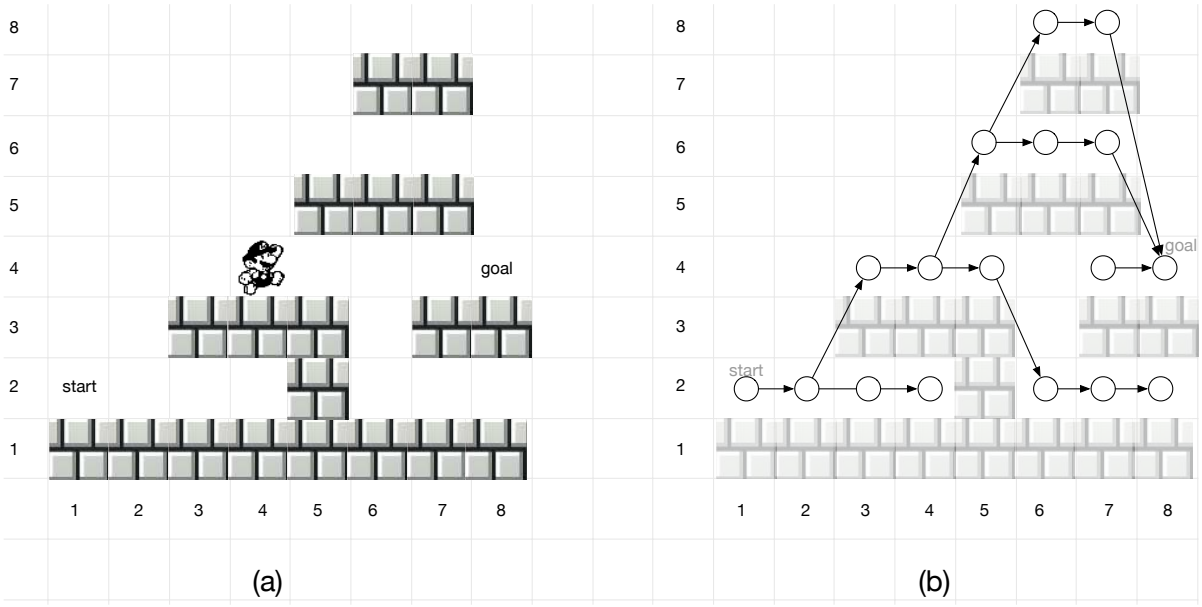


Figure 1: 8×8 Mario world and corresponding Mario graph.

A *Mario world* M consists of a $k \times k$ grid. Each field in the grid is *empty* or *brick*. Two empty fields are marked as *start* and *goal* (see Fig. 5(a)). The goal of the game is to move the player, called Mario, from the start field to the goal field. When Mario is in field (x, y) he has the following options:

Forward Mario moves to the field $(x+1, y)$. This move is possible if $(x+1, y)$ is empty and $(x+1, y-1)$ is brick.

Jump Mario moves to the field $(x+1, y+2)$. This move is possible if $(x+1, y+2)$ is empty, $(x, y+1)$ is empty, and $(x+1, y+1)$ is brick.

Fall Mario moves to the field $(x+1, y')$ where $y' < y$ is the maximal value y' such that $(x+1, y'-1)$ is brick and $(x+1, y), (x+1, y-1), \ldots, (x+1, y')$ are empty.

For instance, Mario in field $(4, 4)$ can move forward to $(5, 4)$ or jump to field $(5, 6)$. A move is *valid* if the move can be done according to the above rules. All fields that can be reached via valid moves from the start field are the *valid fields*. For example, $(4, 4)$ is a valid field, since it can be reached from the start field in $(1, 2)$ doing the sequence of valid moves: forward, forward, jump.

A Mario world M defines a directed graph G with n vertices and m edges (see Fig. 5(b)). All valid fields correspond to a vertex and the valid moves define the edges (there is an edge from field (x, y) to field (x', y') if Mario can move from (x, y) to (x', y') with a valid move). The edges corresponding to forward, jump and fall are denoted by *forward edges*, *jump edges* and *fall edges*.

5.1 (3 %) Let M be a Mario world defined on a $k \times k$ grid and let G be the corresponding Mario graph.

Give the maximum number of nodes n that can appear in G as a function of k.

Solution: ☐ $O(\log k)$ ☐ $O(\sqrt{k})$ ☐ $O(k)$ ☐ $O(k \log k)$ ☐ $O(k^2)$ ☐ $O(2^k)$

Give the maximum number of edges m that can appear in G as a function of k.

Solution: ☐ $O(\log k)$ ☐ $O(\sqrt{k})$ ☐ $O(k)$ ☐ $O(k \log k)$ ☐ $O(k^2)$ ☐ $O(2^k)$

5.2 (6 %) We are now interested in checking if it is possible to go from the start field to the goal field. A Mario graph is *valid* if there exists a path from start to goal. Give an algorithm that given a Mario graph G, decides whether or not there is a path from the start field to the goal field. Analyze the running time of your algorithm as a function of n and m.

Solution:

5.3 (6 %) It's tough work for Mario to jump and fall. Thus, we associate a *price* with each edge e, $price(e)$, such that a forward edge costs 1, a jump edge costs $2\frac{1}{2}$ and a fall edge costs 4. We define the price of a path to be the sum of the prices on the edges of the path. Give an algorithm which, given a valid Mario graph, finds a cheapest path from the start field to the goal field. Analyze the running time of your algorithm as a function of n and m.

Solution:

5.4 (6 %) Now, we're also interested in collecting a *mushroom* during the game. A *mushroom graph* is a valid Mario graph where a single node c is marked as a *mushroom node* and there exists a path from start to goal going through c. Give an algorithm which, given a mushroom graph, finds a cheapest path from the start field to the goal field going through c. Analyze the running time of your algorithm as a function of n and m.

Solution:

5.5 (4 %) A *k-mushroom graph* is a valid Mario graph where k nodes $c_0, \ldots, c_{k-1}$ are marked as mushroom nodes and for each mushroom c_i, $0 \leq i \leq k-1$, there exists a path from start to goal going through c_i. A *mushroom path* in a *k-mushroom graph* is a path from start to goal going through *at least* one of the k mushrooms. Give an algorithm which, given a k-mushroom graph, finds a cheapest mushroom path from the start field to the goal field. Analyze the running time of your algorithm as a function of n, m and k.

Solution:

Technical University of Denmark

Written examination, May 21, 2019.

Course name: Algorithms and Data Structures

Course number: 02326

Aids: Written aids. Calculators are **not** permitted.

Duration: 4 hours.

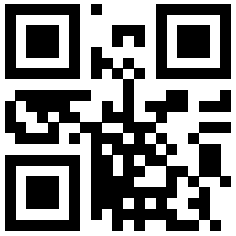
Weights: Exercise 1 - 20 %, Exercise 2 - 25 %, Exercise 3 - 20 %, Exercise 4 - 13 %, Exercise 5 - 22 %. The weights are approximate. The grade is based on an overall assessment.

All exercises should be answered by filling out the areas below the description. As your solution to the exam, just hand in this page and the following pages. If you really need more space, you may use extra pieces of paper and hand these in along with your solution.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

Name: _____

Student ID: _____



1 Complexity

1.1 (5 %) For each statement below, mark whether or not it is correct.

	Yes	No
$2 + 2n^2 + 2\log^2 n = \Theta(n^2)$	☐	☐
$\frac{2\cdot\sqrt{n}}{10^7} + 2 \cdot \log(n^7) = O(\sqrt{n})$	☐	☐
$\log(2^n) + \log(2n) = \Omega(n)$	☐	☐
$80^2 \cdot n + \frac{n^2}{10^7} = \Theta(n)$	☐	☐
$\log(\log n) + 5n^{1/3} + 6n^{1/5} = \Omega(n^{1/4})$	☐	☐

1.2 (5 %) Arrange the following functions in increasing order according to asymptotic growth, that is, if $g(n)$ immediately follows $f(n)$ then $f(n) = O(g(n))$.

$$2 \cdot 3^n \qquad 18^4 \qquad \sqrt{n}\log n \qquad 54n^3 \qquad n^2 \qquad \log(n^3)$$

Solution:

1.3 (10 %) State the running time for each of the following algorithms. Write your solution in O-notation as a function of n.

```
ALG1(n)
c = 0
j = n − 4
while j ≤ n do
   j = j + 1
   for i = 1 to n do
      c = j + i
   end for
end while
```

Solution:

```
ALG2(n)
if n ≤ 1 then
   return 1
else
   return 1 + ALG2(1) + ALG2(1)
end if
```

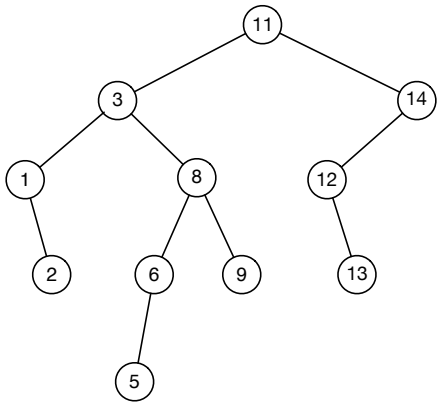
Solution:

```
ALG3(n)
c = 0
for i = 1 to ⌈log n⌉ do
   for j = i to n do
      c = i + j
   end for
end for
```

Solution:

2 Data Structures and Algorithms

Consider the following binary search tree T.



2.1 (5 %) Write the ordering of the vertices from a preorder, postorder, and inorder traversal of T.

PREORDER : _____

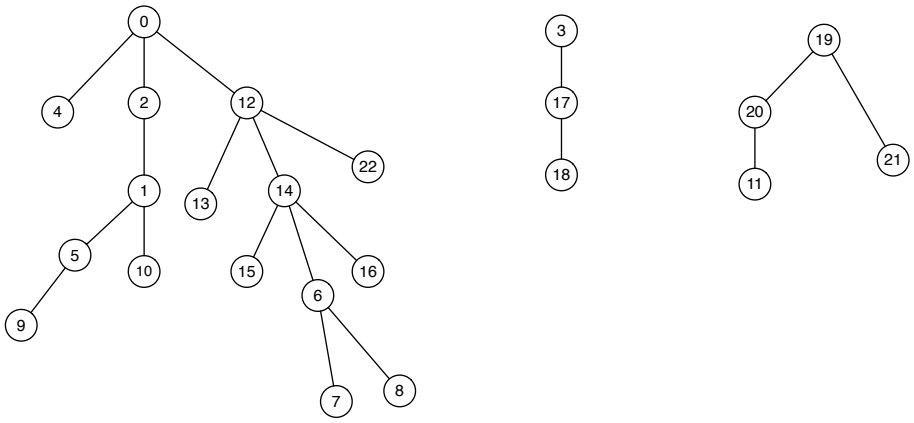
POSTORDER : _____

INORDER : _____

2.2 (5 %) Show T after deleting the vertex with key 3.

Solution:

Consider the following forest of trees representing a family of sets in a union find data structure constructed using the quick union algorithm.



2.3 (5 %) State the results of the following FIND(·) operations.

FIND(2) : _____ FIND(8) : _____

FIND(18) : _____ FIND(11) : _____

FIND(0) : _____ FIND(14) : _____

2.4 (5 %) Assume that we now use path compression on the forest above. Show the forest of trees after a FIND(8) operation.

Solution:

2.5 (5 %) We want to support the operation TOP(k) on each of the following data structures. Let n be the total number of elements in the data structure. Given a parameter $k \leq n$, TOP(k) returns the k largest elements in the data structure. For instance, if the data structure consists of the elements $\{6, 32, 18, 7, 2\}$, TOP(2) should return 32 and 18 since these are the 2 largest elements among the 5 elements. State for each of the following data structures, the running time of an efficient algorithm for the TOP(k) operation in O-notation as a function of k and/or n.

Single-linked list: _____

Unsorted array: _____

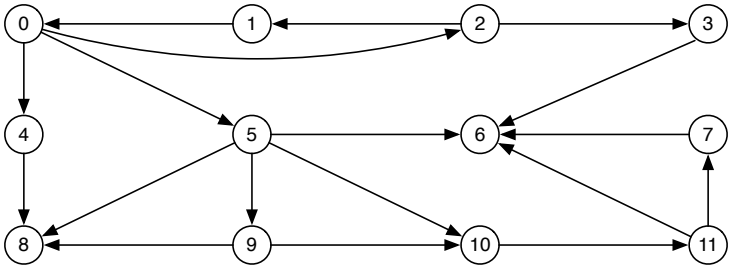
Sorted array: _____

Max-heap: _____

Binary search tree: _____

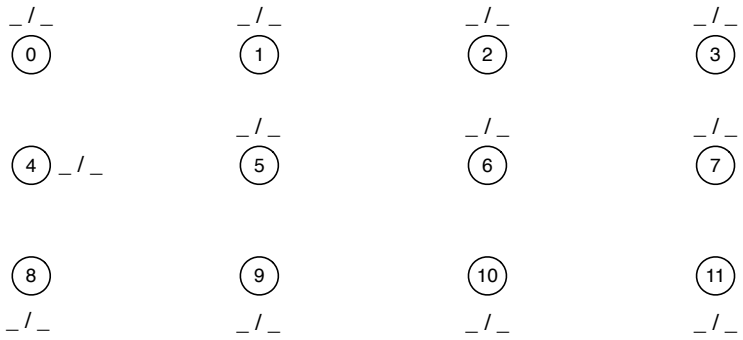
3 Graphs

Consider the following graph G.



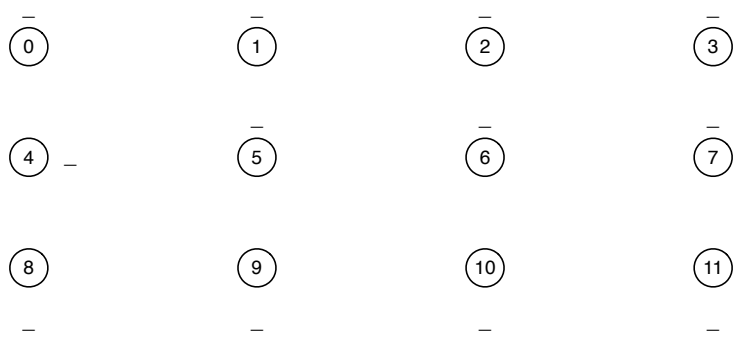
3.1 (5 %) Show the DFS tree for G when starting in vertex 0 and write the discovery and finish times for each vertex. Assume that the adjacency lists are sorted in increasing order. Write the discovery and finish times for each vertex in the area marked by "_ / _" next to each vertex.

Solution:

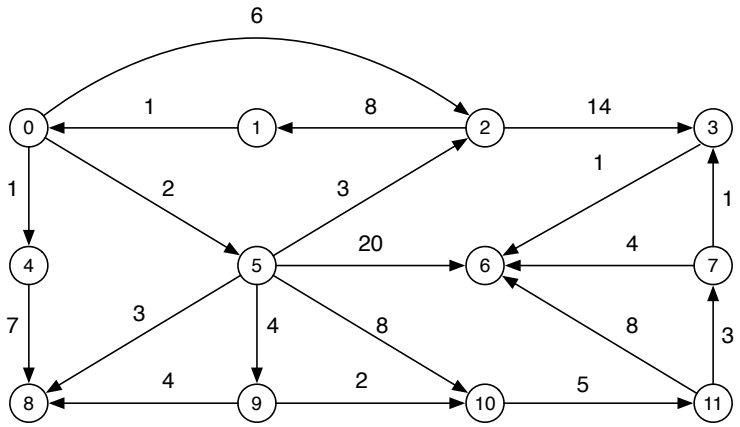


3.2 (5 %) Show the BFS tree for G when starting in vertex 0. Assume that the adjacency lists are sorted in increasing order. Write the BFS layer for each vertex in the area marked by "_" next to each vertex.

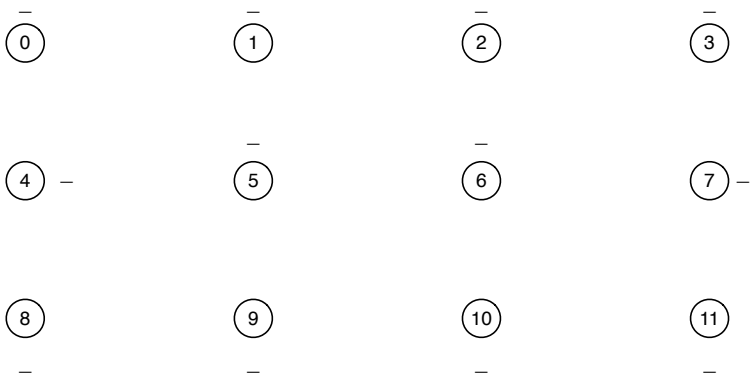
Solution:



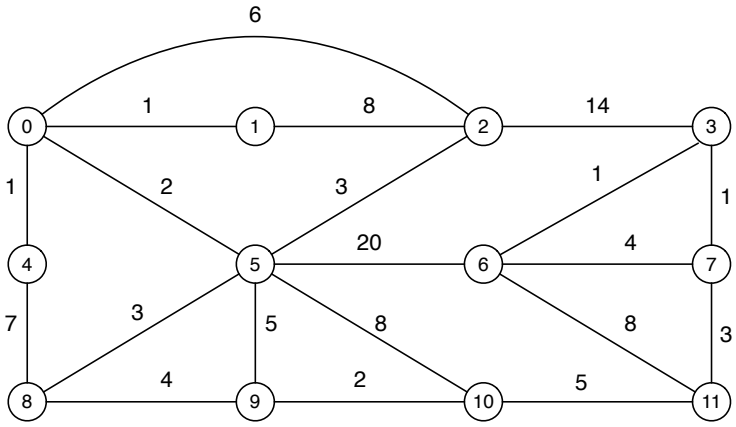
3.3 (5 %) Consider the following graph. Show a shortest path tree for the graph starting at vertex 0. Write the length of the shortest path for each vertex in the area marked by "_" next to each vertex.



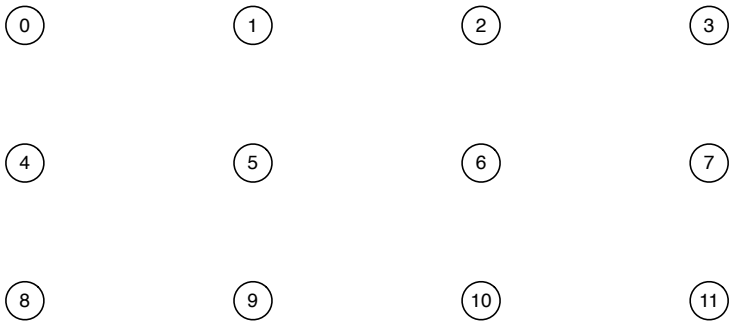
Solution:



3.4 (5 %) Consider the following graph. Show a minimum spanning tree for the graph. State the total weight of the tree.



Solution:



Total weight: _____

4 Trees

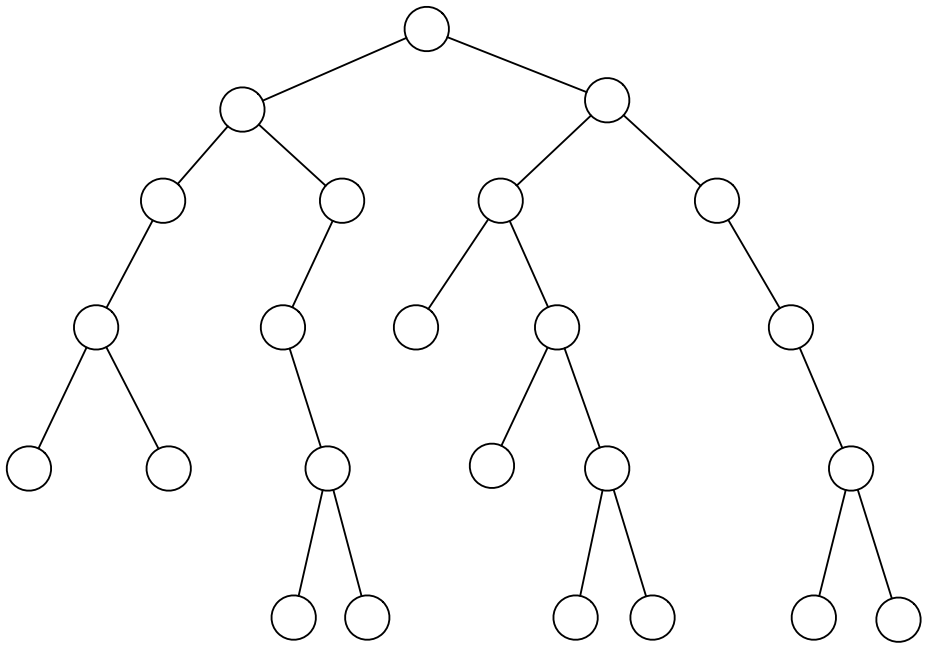
This exercise is about rooted binary trees. Each node x has fields $x.parent$, $x.left$ and $x.right$, denoting the parent, left child, and right child of x. For the root $root$, $root.parent = null$. Throughout the exercise, we let n denote the size of the input tree.

4.1 (1 %) Recall that a node has a *sibling* if its parent has two children. A node x is a *twig* if it satisfies the following two properties:

- x has depth at least 2.
- x has exactly two children that are both leaves.
- x and the parent of x have no siblings.

Consider the tree below. Mark all twigs in the tree directly on the vertices in the tree.

Solution:



4.2 (4 %) Give an algorithm, SIBLING(x), that given a non-root vertex x returns true if and only if x has a sibling. Write your algorithm in pseudocode. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

4.3 (4 %) Give an algorithm, TWIG(x), that given a vertex x of depth at least 3 returns true if and only if x is a twig. In addition to the SIBLING algorithm from the previous exercise, you can assume that you have a constant time algorithm LEAF(X) that given a vertex x, returns true if and only if x is a leaf. Write your algorithm in pseudocode. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

4.4 (4 %) Give a recursive algorithm, TWIGCOUNT(x), that given the root, returns the number of twigs in the tree. Write your algorithm in pseudocode. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

5 Ruling the Galaxy

A *galaxy* is a set of P *planets* and a set of H *hyperspace links*. Each hyperspace link is a pair (x, y) of distinct planets x and y and we say that planet x is *hyperspace linked* to y. Furthermore, a *hyperspace sequence* from planet z_1 to planet z_j is a sequence of planets $z_1, \ldots, z_j$ such that (z_i, z_{i+1}) is a hyperspace link for $i = 1, \ldots, j-1$. We assume that any pair of planets has at least one hyperspace sequence between them. For example, the sets $\{a, b, c, d, e\}$ and $\{(a, b), (b, c), (c, d), (d, e), (c, e), (c, a)\}$ is a galaxy with $P = 5$ planets and $H = 6$ hyperspace links. The sequence a, b, c, e is a hyperspace sequence from a to e.

5.1 (2 %) Describe how to model a galaxy as a graph

Solution:

5.2 (1 %) Draw the graph corresponding to the galaxy in the example.

Solution:

5.3 (6 %) Given a non-negative integer t, a *t-hub* is a planet that is hyperspace linked to at least t other planets. Give an algorithm, that given a galaxy and a non-negative integer t, computes the number of t-hubs in the galaxy. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

5.4 (5 %) We are now interested in optimizing the cost of maintaining the empire while ruling the galaxy. We associate to each hyperspace link $\ell = (x, y)$ a *maintenance cost*, denoted $m(\ell)$, that indicates the cost of keeping the hyperspace link open for the fleet. An *hyperspace link set* is a subset of hyperspace links such that any two planets have a hyperspace sequence between them and an *optimal hyperspace link set* is a hyperspace link set such that the total cost of its hyperspace links is minimal.

Give an algorithm, that given an galaxy, computes an optimal hyperspace link set. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

5.5 (5 %) The empire has a new Death Star weapon system that is extremely powerful, but also slow to move from planet to planet. We want to compute how fast the Death Star can reach any planet in the galaxy from the empires home planet. We associate to each hyperspace link $\ell = (x, y)$, a *travel time*, denoted $t(x, y)$, that indicates the time needed for the Death Star to travel between x and y. The travel time for a hyperspace sequence is the sum of travel time of the hyperspace links in the hyperspace sequence. The *home planet*, h, is a special planet where the Death Star is docked between operations. The *response time to planet* x is the minimum travel time for the Death Star from h to x. The *galactic response time* is the maximum response time for any planet in the galaxy.

Give an algorithm, that given a galaxy and a home planet h, computes the galactic response time. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

5.6 (3 %) To improve the response times the empires now invests in k Death Stars, located at distinct home planets $h_1, \ldots, h_k$. In this setting, the response time to a planet x is now the shortest response time for any of the Death Stars, i.e., the minimum travel time for any of the k Death Stars to x. The galactic response time is the maximum response time for any planet in the galaxy. Give an algorithm, that given a galaxy and k home planets $h_1, \ldots, h_k$, computes the galactic response time. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

Technical University of Denmark

Written test examination, May xx, 2020.

Course name: Algorithms and Data Structures 1

Course number: 02105

Aids: all aids, open internet.

Duration: 2 hours.

Weights: XX.

All exercises should be answered by filling out the areas below the description. As your solution to the exam, hand in this page and the following pages. If you really need more space, you may use extra pieces of paper and attach these to the end of your solution.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

"Give an algorithm" means that you should describe your solution in a short, precise, and unambiguous manner, argue correctness of your solution, and analyze the complexity of your solution. Unless specified otherwise, the description should be in natural language and not pseudocode. The analysis should explain how you derived the complexity bound.

"Argue correctness of your algorithm" means that you should provide a short argument for correctness of your algorithm.

"Analyse the running time of your algorithm in the relevant parameters (parameters x, y, ..) of the problem" means that you should analyze the running time using the explicitly stated parameters of the problem (parameters x, y, ..).

Name: _____

Student ID: _____

1 Movie Reviews

Consider analyzing a collection R of reviews of a popular movie. Each review has a *score* that is non-negative integer. Throughout the exercise, we let n denote the size of R and we assume that R is given as an array, where each entry contains the score of a review.

1.1 Give an algorithm that given R computes the total number of reviews with a score in the range $[n, n^2]$. Analyse the running time of your algorithm in terms of parameter n.

Solution:

1.2 Give an algorithm that given R computes the total number of *unique* reviews scores in the range $[n, n^2]$ (i.e., multiple identical review scores should only be counted once). Analyze the running time of your algorithm in terms of parameter n.

Solution:

1.3 We now want to find the position in the array representing R containing a particular review score s, that occurs exactly once in R. Suppose that you are given a black-box algorithm TEST(l, r, s) that can determine if the score s exists in the range $R[l, r]$. The algorithm TEST runs in $O(\log n)$ time. Give an algorithm, that given R and s, computes the position of s in R. Use the TEST algorithm to help you. Analyze the running time of your algorithm in terms of parameter n.

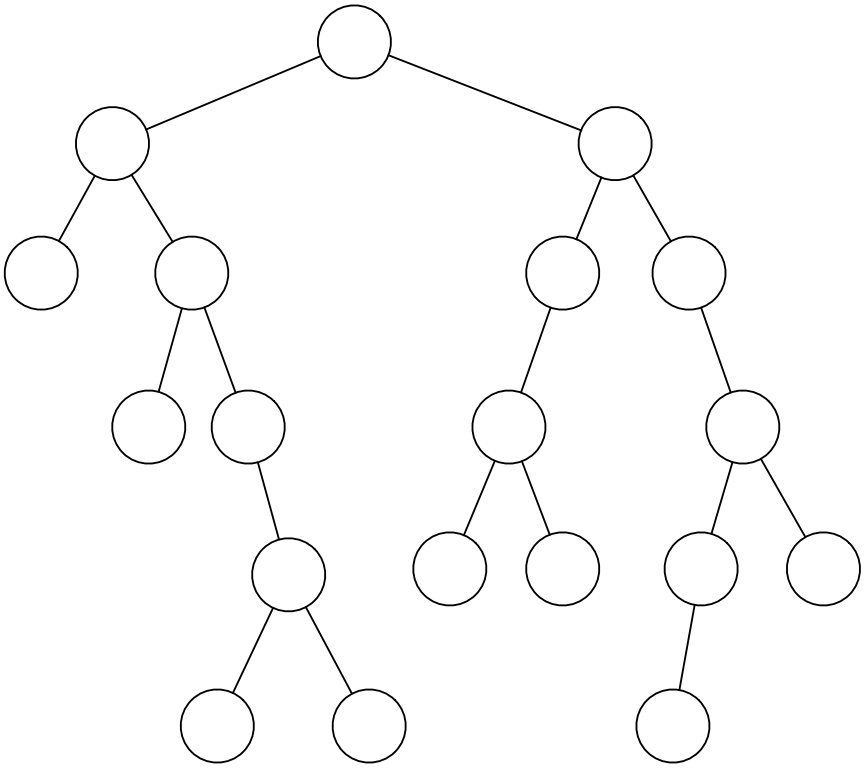
Solution:

2 Trees distances

This exercise is about rooted binary trees. Each node x has fields $x.parent$, $x.left$ and $x.right$ denoting the parent, left child, and right child of x. For the root $root$, $root.parent = null$. Throughout the exercise, we let n denote the size of the input tree.

2.1 A *descendant leaf* of a node x is a descendant of node x that is also a leaf. A *nearest descendant leaf* is a descendant leaf of minimal distance to x. The *leaf distance* for x is the distance from x to a nearest descendant leaf (if x is a leaf itself the distance is of course 0). Consider the tree below. Write the leaf distance for each node x in the tree. Do so directly on the vertices in the tree.

Solution:



2.2 Give a recursive algorithm, LEAFDIST(x), that given the root computes the leaf distance for all nodes in the tree. Assume that all nodes have a field *leafdist* and after the call to LEAFDIST on the root, it should hold for all nodes y that y.*leafdist* is the leaf distance for y. Write your algorithm in pseudocode. Analyse the running time of your algorithm in the parameter n.

Solution:

2.3 Now, assume that for all nodes y, $y.leafdist$ is the leaf distance for y. Give a recursive algorithm CLOSESTLEAF(x), that given the root node, prints out all nearest descendant leafs of x. Write your algorithm in pseudocode. Analyse the running time of your algorithm in the parameter n.

Solution:

3 Zombies

A *zombie outbreak* consists of a set of persons and a set of infections. Each infection is a pair of persons (p_1, p_2) and we say that p_1 has *infected* p_2. Note that multiple persons may infect the same person. For instance, $\{A, B, C, D, E\}$ and $\{(A, B), (B, C), (C, D), (A, E), (E, C)\}$ is a zombie outbreak with 5 persons and 5 infections. Throughout the exercise, let P denote the number of persons and I the number of infections.

3.1 Describe how to model a zombie outbreak as a graph

Solution:

3.2 Draw the graph corresponding to the zombie outbreak in the example.

Solution:

3.3 We are interested in investigating the origin of the outbreak. A person p is a *patient-zero* if p did not get infected by any other person. Give an algorithm, that given a zombie outbreak, prints out all patient-zero persons. Argue correctness of your algorithm. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

3.4 We are now interested in reconstructing the details of the zombie outbreak and want to ensure that the data is correct. An *infection chain* is a sequence of persons $p_0, \ldots, p_{k-1}$ such that p_i has infected p_{i+1} for $i = 0, \ldots, k-2$. A zombie outbreak is *inconsistent* if there is an infection chain that starts and ends with the same person. If a zombie outbreak is not inconsistent it is *consistent*. Give an algorithm, that given a zombie outbreak, determines if the zombie outbreak is inconsistent or consistent. Argue correctness of your algorithm. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

3.5 We now associate to every infection i an *infection speed*, $speed(i)$. We define the infection speed of an infection chain to be the sum of the infection speeds of each infection in the chain. We are interested in computing the fastest infection chain between two persons in a consistent zombie outbreak. Given an algorithm, that given a consistent zombie outbreak and two persons p_1 and p_2 computes a fastest infection chain from p_1 to p_2. Argue correctness of your algorithm. Analyse the running time of your algorithm in the relevant parameters of the problem.

Solution:

02105 - Algorithms and Data Structures May 2022 (MC part)

Der anvendes en scoringsalgoritme, som er baseret på "One best answer"

Dette betyder følgende:
- Der er altid netop ét svar som er mere rigtigt end de andre
- Studerende kan kun vælge ét svar per spørgsmål
-

The following approach to scoring responses is implemented and is based on "One best answer"
- There is always only one correct answer – a response that is more correct than the rest
- Students are only able to select one answer per question

Page 1

For each question below, mark whether or not the statement is correct.

	Yes	No
$42n + \frac{42n}{3} + \frac{50n}{7} = \Theta(n)$	☑	☐
$n^2(n^2 + 8n^3) = O(n^5 \log n)$	☑	☐
$5\sqrt{n} + \log^2 n = \Omega(\log^3 n)$	☐	☑
$\log n + \log^2(\sqrt{n}) = O(\log n)$	☑	☐
$2^n \cdot 2^n = O(2^{n+7})$	☐	☑

Mark the running time in O-notation of each of the following algorithms as a function of n. Mark the best bound.

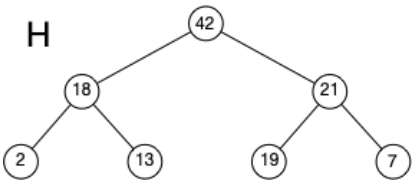
```
ALG1(n)
s = 1
for i = 1 to 2⌈√n⌉ + 42⌈log n⌉ do
   for j = i to ⌈√n⌉ do
      s = i + j
   end for
end for
```

```
ALG2(n)
c = 0
j = n
while j ≥ ⌈log n⌉ do
   j = j − 1
end while
```

```
ALG3(n)
if n ≤ 1 then
   return 1
else
   return ALG3(⌈n/2⌉) + ALG3(⌈n/2⌉)
end if
```

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^2 \log n)$	$O(n^3)$	$O(2^n)$
Alg1	☐	☐	☐	☐	☐	☑	☐	☐	☐
Alg2	☐	☐	☐	☑	☐	☐	☐	☐	☐
Alg3	☐	☐	☐	☐	☐	☐	☑	☐	☐

Consider the following binary heap H.

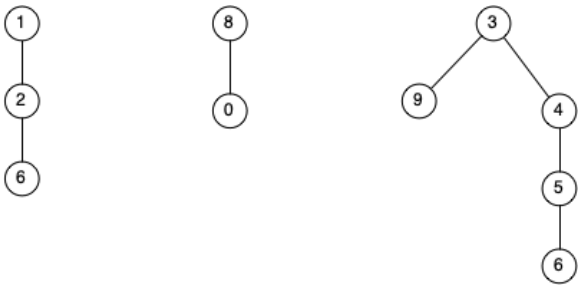


Let H_1 be the result of applying an Extract-Max operation on H and let H_2 be the result of applying an Insert operation with key 20 on H. Construct the arrays representations of H, H_1, and H_2 and state the content of each of the following entries (recall that a heap with n items in the array representation consists of an array of length n+1 where index 0 is not used).

	2	7	13	18	19	20	21	42
H[3]	○	○	○	○	○	○	●	○
H[4]	●	○	○	○	○	○	○	○
H[5]	○	○	●	○	○	○	○	○
H[6]	○	○	○	○	●	○	○	○
H_1[3]	○	○	○	○	●	○	○	○
H_1[4]	●	○	○	○	○	○	○	○
H_1[5]	○	○	●	○	○	○	○	○
H_1[6]	○	●	○	○	○	○	○	○
H_2[3]	○	○	○	○	○	○	●	○
H_2[4]	○	○	○	●	○	○	○	○
H_2[5]	○	○	●	○	○	○	○	○
H_2[6]	○	○	○	○	●	○	○	○

Union Find

Consider the following forest of trees representing a family of sets in a union find data structure constructed using the quick union algorithm.



State the result of the following Find(.) operations.

	0	1	2	3	4	5	6	7	8	9
Find(0)	○	○	○	○	●	○	○	○	○	○
Find(1)	●	○	○	○	○	○	○	○	○	○
Find(2)	○	○	●	○	○	○	○	○	○	○
Find(3)	○	●	○	○	○	○	○	○	○	○
Find(4)	○	○	○	○	○	○	●	○	○	○
Find(5)	○	○	○	○	○	○	○	○	●	○

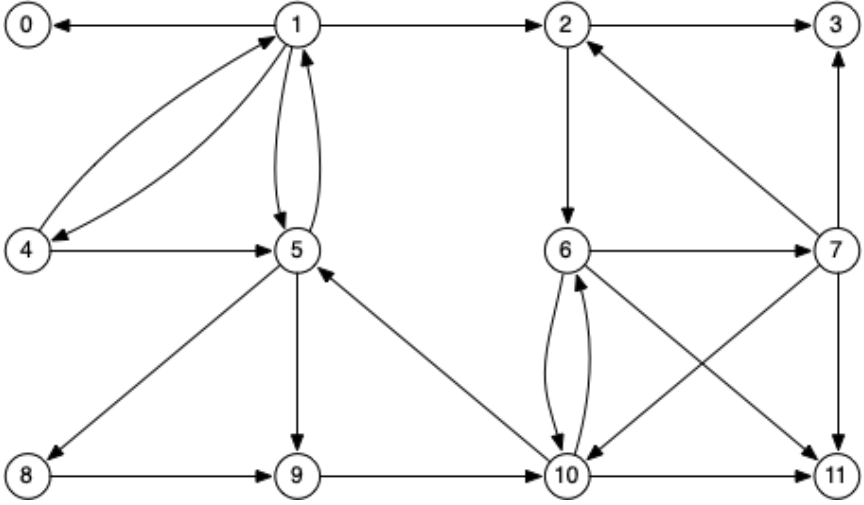
Suppose we now use path compression on the forest above. Construct the forest of trees F after a Find(5) operation and answer the following question for F.

	1	2	3	4	5	6	7	8
The total number of leaves in all trees in F is	○	○	○	●	○	○	○	○
The maximum depth of a tree in F is	○	○	○	○	○	●	○	○
The minimum depth of a tree in F is	○	○	○	○	●	○	○	○
The maximum degree of a root of a tree in F is	○	○	●	○	○	○	○	○
The minimum degree of a root of a tree in F is	●	○	○	○	○	○	○	○

Consider the following data structures for representing a set of integers. We are interested in supporting the operation Max-Min-Sum(), that returns the sum of the smallest and largest integer in the set. For instance, if a data structure stores the set {32, 6, 18, 7, 2} then Max-Min-Sum() should return 34. For each data structure below mark the runtime of the Max-Min-Sum() operation in O-notation as a function of n, where n is the number of elements in the data structure. Mark the best bound

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$
Sorted array	●	○	○	○	○	○	○
Max-heap	○	○	○	●	○	○	○
Min-heap	○	○	○	●	○	○	○
Array (not sorted)	○	○	○	○	○	●	○
Binary search tree	○	○	○	●	○	○	○

Graph Search
Consider the following graph G.



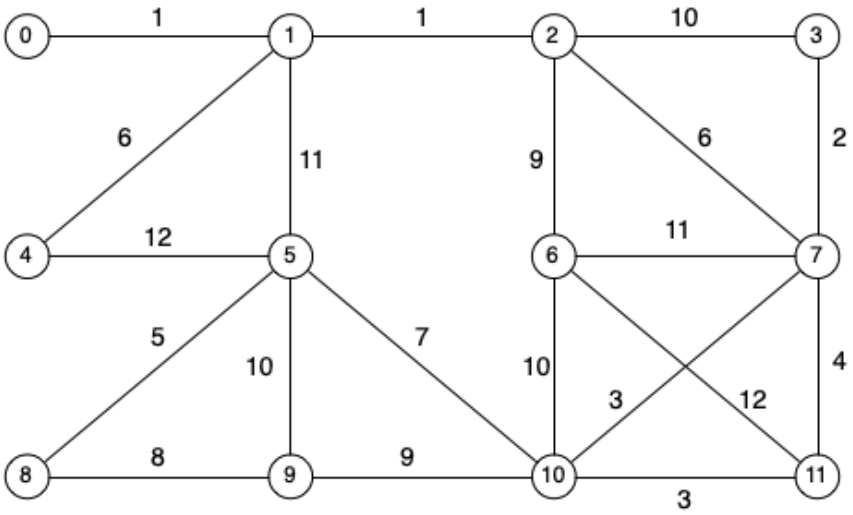
Construct the DFS tree T_1 for G when starting in vertex 4. Assume that each adjacency list is sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7	8	9
The depth of T_1 is	○	○	○	○	○	○	●	○	○
The number of leaves of T_1 is	○	○	●	○	○	○	○	○	○
The maximum number of children of a node in T_1 is	○	●	○	○	○	○	○	○	○

Construct the BFS tree T_2 for G when starting in vertex 7. Assume that each adjacency list is sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7	8	9
The depth of T_2 is	○	○	○	●	○	○	○	○	○
The number of leaves of T_2 is	○	○	○	○	○	○	●	○	○
The maximum number of children of a node in T_2 is	○	○	○	●	○	○	○	○	○

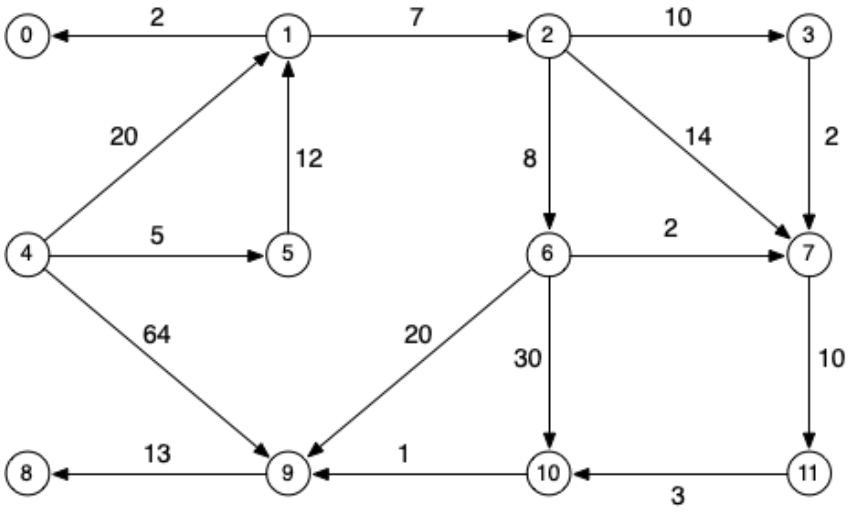
Consider the following graph.



Construct a minimum spanning tree T for the graph and answer the following questions.

	1	2	3	4	5	6	7	8	9
The total number of edges on a longest path of T is	○	○	○	○	○	○	●	○	○
The number of leaves of T is	○	○	○	○	○	○	●	○	○
The maximum degree of a node in T is	○	○	○	●	○	○	○	○	○

Consider the following graph



Construct a shortest path tree T for the graph when starting at vertex 4. Answer the following questions for T

	1	2	3	4	5	6	7	8	9
The total number of edges on a longest path in T is	○	○	○	○	○	○	○	●	○
The number of leaves in T is	○	○	○	●	○	○	○	○	○
The maximum number of children of a node in T is	○	●	○	○	○	○	○	○	○

Algorithms and Data Structures 02105 multiple-choice part - Test Exam

The following approach to scoring responses is implemented and is based on "One best answer"

- There is always only one correct answer – a response that is more correct than the rest
- Students are only able to select one answer per question

Page 1

For each question below, mark whether or not the statement is correct.

	Yes	No
$3n^4 + 2n^3 = O(n^3)$	○	●
$4n + 45 \log n = \Theta(n)$	●	○
$\frac{1}{3}n^5 = \Omega(n^4)$	●	○
$2^{n+2} = \Theta(2^n)$	●	○
$(\frac{1}{3}n^5 + n^2 + n)n = O(n^6)$	●	○

State the running time in O-notation of each of the following algorithms as a function n. Choose the best bound.

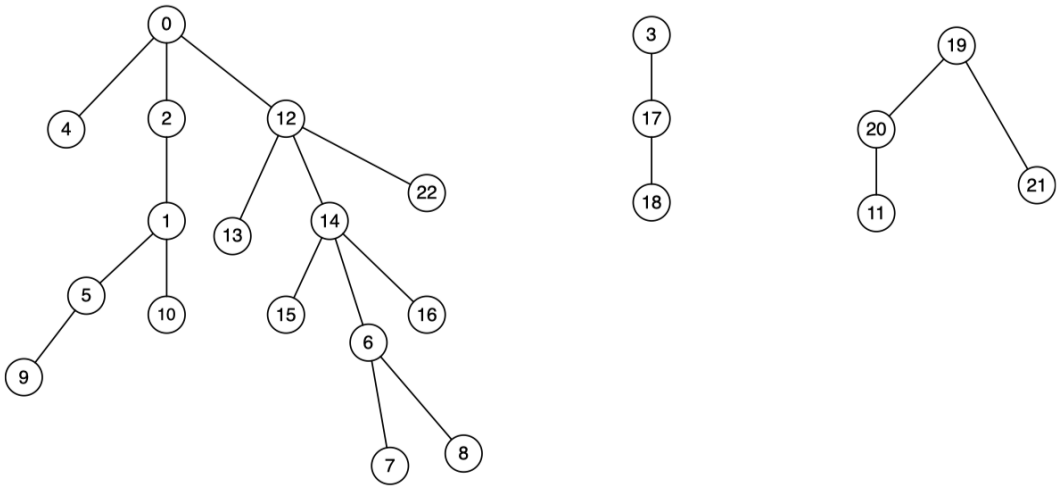
```
ALG1(n)
c = 0
j = n − 4
while j ≤ n do
  j = j + 1
  for i = 1 to n do
    c = j + i
  end for
end while
```

```
ALG2(n)
if n ≤ 1 then
  return 1
else
  return 1 + ALG2(1) + ALG2(1)
end if
```

```
ALG3(n)
c = 0
for i = 1 to ⌈log n⌉ do
  for j = i to n do
    c = i + j
  end for
end for
```

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^2 \log n)$	$O(n^3)$	$O(2^n)$
Alg1	○	○	○	●	○	○	○	○	○
Alg2	●	○	○	○	○	○	○	○	○
Alg3	○	○	○	○	●	○	○	○	○

Union Find
Consider the following forest of trees representing a family of sets in a union find data structure constructed using the quick union algorithm.



State the result of the following Find(.) operations.

	0	2	3	8	11	14	18	19	20	21
Find(2)	☑	○	○	○	○	○	○	○	○	○
Find(18)	○	○	☑	○	○	○	○	○	○	○
Find(0)	☑	○	○	○	○	○	○	○	○	○
Find(8)	☑	○	○	○	○	○	○	○	○	○
Find(11)	○	○	○	○	○	○	○	☑	○	○
Find(14)	☑	○	○	○	○	○	○	○	○	○

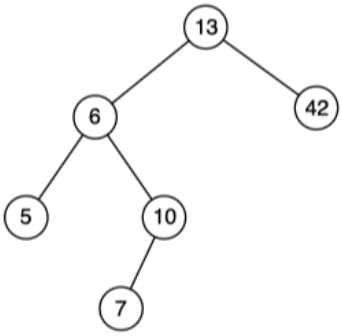
Suppose we now use path compression on the forest above. Construct the forest of trees F after a Find(8) operation and answer the following question for F.

	1	2	3	4	5	8	10	12	14	16
The total number of leaves in all trees in F is	○	○	○	○	○	○	○	☑	○	○
The maximum depth of a tree in F is	○	○	○	☑	○	○	○	○	○	○
The minimum depth of a tree in F is	☑	○	○	○	○	○	○	○	○	○
The maximum degree of the root of a tree is	○	○	○	○	☑	○	○	○	○	○
The minimum degree of a root of a tree in F is	☑	○	○	○	○	○	○	○	○	○

Consider the following data structures for representing a set of integers. We are interested in supporting the operation Member(x), that given an integer x should return yes if x is in the set and no otherwise. For instance, if a data structure stores the set {32, 6, 18, 7, 2} then Member(6) should return yes while Member(4) should return no. For each data structure state the runtime of Member(x) operation in O-notation as a function of n, where n is the number of elements in the data structure.

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$
Sorted array	○	☑	○	○	○	○	○
Max-heap	○	○	○	☑	○	○	○
Min-heap	○	○	○	☑	○	○	○
Array (not sorted)	○	○	○	☑	○	○	○
Binary search tree	○	○	○	☑	○	○	○

Binary Search Trees
Consider the following binary search tree.



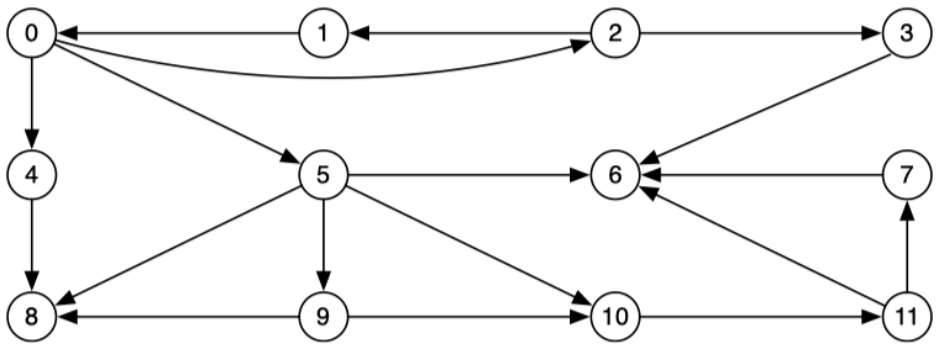
Construct the preorder, inorder, and postorder traversal of the tree and answer the following questions.

	5	6	7	10	13	42
The key of the first node is the preorder traversal is	○	○	○	○	●	○
The key of the first node in the inorder traversal is	●	○	○	○	○	○
The key of the first node in the postorder traversal is	●	○	○	○	○	○
The key of the last node in the preorder traversal is	○	○	○	○	○	●
The key of the last node in the inorder traversal is	○	○	○	○	○	●
The key of the last node in the postorder traversal is	○	○	○	○	●	○
The key of the 4th node in the preorder traversal is	○	○	○	●	○	○
The key of the 4th node in the inorder traversal is	○	○	○	●	○	○
The key of the 4th node in the inorder traversal is	○	●	○	○	○	○

Insert a new node with key 9 into the above tree and let T be the resulting binary search tree. Answer the following questions.

	0	1	2	3	4	5
The height of T is	○	○	○	○	●	○
The number of nodes with 2 children in T is	○	○	●	○	○	○
The number of nodes with 1 child in T is	○	○	●	○	○	○

Graph Search
Consider the following graph G.



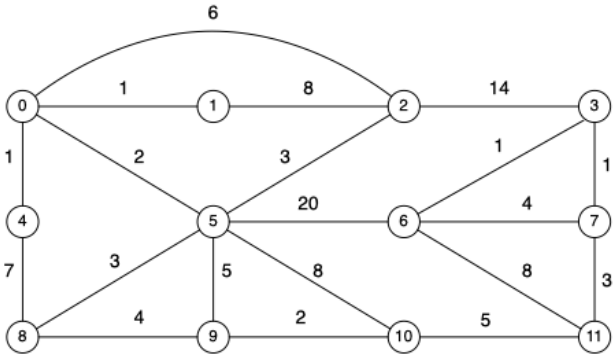
Construct the DFS tree for G when starting in node 0. Assume that the adjacency list are sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7
The depth of the tree is	○	○	○	○	●	○	○
The number of leaves of the tree is	○	○	○	○	●	○	○
The maximum number of children of a node in the tree is	○	○	●	○	○	○	○

Construct the BFS tree for G when starting in node 0. Assume that the adjacency list are sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7
The depth of the tree is	○	○	○	●	○	○	○
The number of leaves of the tree is	○	○	○	○	○	●	○
The maximum number of children of a node in the tree is	○	○	●	○	○	○	○

Minimum Spanning Trees
Consider the following graph.



Construct a minimum spanning tree T for the graph.

Answer the following questions.

	2	3	4	5	6	7	8
The total number of edges on a longest path of T is	○	○	○	○	○	○	●
The number of leaves of T is	○	○	●	○	○	○	○
The maximum degree of a node in T is	○	●	○	○	○	○	○

State the total weight of T.

26

Technical University of Denmark

Written examination, May 17, 2022.

Course name: Algorithms and Data Structures 1

Course number: 02105

Aids: all aids, open internet.

Duration: 2.5 hours.

Weight: Exercise 1 - 35%, exercise 2 - 25%. The weight is approximate. The grade is based on an overall assessment of the multiple-choice part, the implementation part, and the written part of the exam.

Instructions for exercise 1.

Write your answers to the exercises and submit them using the digital exam system. Your answer should be in the form of a `pdf`-file containing exactly 1 page in the a4-format. The top of the page should clearly list your full name and your study id. The rest of page should contain your answers to the each of the exercises. Number each of your answers with the number of the corresponding exercise. Use a clearly readable font of size 10pt or more and margins of 2cm or more.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

"Give an algorithm" means that you should describe your solution in a short, precise, and unambiguous manner and analyze the complexity of your solution. Unless specified otherwise, the description should be in natural language and not pseudocode. The analysis should explain how you derived the complexity bound.

"Argue correctness of your algorithm" means that you should provide a short argument for correctness of your algorithm.

"Analyze the running time of your algorithm in the relevant parameters (parameters x, y, ..) of the problem" means that you should analyze the running time using the explicitly stated parameters of the problem (parameters x, y, ..).

Instructions for exercise 2.

Read the exercise text in the CodeJudge system and implement a solution. Upload your solution to the CodeJudge system.

1 Skiing

We are interested in planning trip on a mountain at a ski resort. The ski resort consists of x *positions* $p_0, \ldots, p_{x-1}$ and y *ski slopes* $s_0, \ldots, s_{y-1}$. Each ski slope s is defined by the following:

- a *start position* $\mathsf{start}(s)$,
- an *end position* $\mathsf{end}(s)$, and
- a *completion time* $\mathsf{time}(s)$ that indicates how many minutes it takes to traverse s on ski.

Note that ski slopes may go both up or down the mountain.

1.1 Briefly describe how to model a ski resort as a graph.

1.2 We are interested in determining if it is possible to get to the bottom of the mountain before the ski slopes close. A *trip* from position a to position b is a sequence of positions starting in a and ending in b that are pairwise connected by ski slopes (i.e., each position p in the sequence is connected to the next position p' by a ski slope s such that $p = \mathsf{start}(s)$ and $p' = \mathsf{end}(s)$. The completion time of a trip is the sum of the completion time of each of the ski slopes of the sequence.

Give an algorithm that given a description of a ski resort and a time threshold t determines whether or not there is a trip from p_0 to p_{x-1} with completion time at most t. Argue correctness of your algorithm. Analyze the running time of your algorithm in terms of parameters x and y.

1.3 We now also store a *level* for each ski slope s, denoted $\mathsf{level}(s)$, that indicates the difficulty of traversing s. A high level for s indicates that s is difficult to traverse. Each level is an integer between 1 and x. Give an algorithm, that given a ski resort and a time threshold t determines the smallest level ℓ such that there is a trip from p_0 to p_{x-1} with completion time at most t that only uses ski slopes of level at most ℓ. Argue correctness of your algorithm. Analyze the running time of your algorithm in terms of parameters x and y.

2 Implementation Exercise

See the exercise text in the CodeJudge system.

02105 - Algorithms and Data Structures May 2022 (MC part)

Der anvendes en scoringsalgoritme, som er baseret på "One best answer"

Dette betyder følgende:
- Der er altid netop ét svar som er mere rigtigt end de andre
- Studerende kan kun vælge ét svar per spørgsmål
-

The following approach to scoring responses is implemented and is based on "One best answer"
- There is always only one correct answer – a response that is more correct than the rest
- Students are only able to select one answer per question

Page 1

For each question below, mark whether or not the statement is correct.

	Yes	No
$42n + \frac{42n}{3} + \frac{50n}{7} = \Theta(n)$	☑	☐
$n^2(n^2 + 8n^3) = O(n^5 \log n)$	☑	☐
$5\sqrt{n} + \log^2 n = \Omega(\log^3 n)$	☐	☑
$\log n + \log^2(\sqrt{n}) = O(\log n)$	☑	☐
$2^n \cdot 2^n = O(2^{n+7})$	☐	☑

Mark the running time in O-notation of each of the following algorithms as a function of n. Mark the best bound.

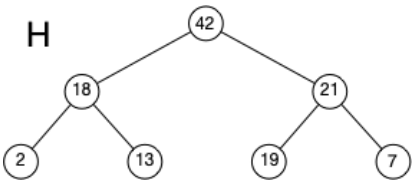
```
ALG1(n)
s = 1
for i = 1 to 2⌈√n⌉ + 42⌈log n⌉ do
   for j = i to ⌈√n⌉ do
      s = i + j
   end for
end for
```

```
ALG2(n)
c = 0
j = n
while j ≥ ⌈log n⌉ do
   j = j − 1
end while
```

```
ALG3(n)
if n ≤ 1 then
   return 1
else
   return ALG3(⌈n/2⌉) + ALG3(⌈n/2⌉)
end if
```

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^2 \log n)$	$O(n^3)$	$O(2^n)$
Alg1	☐	☐	☐	☐	☐	☑	☐	☐	☐
Alg2	☐	☐	☐	☑	☐	☐	☐	☐	☐
Alg3	☐	☐	☐	☐	☐	☐	☑	☐	☐

Consider the following binary heap H.

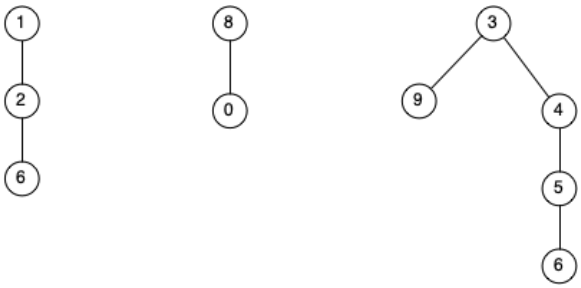


Let H_1 be the result of applying an Extract-Max operation on H and let H_2 be the result of applying an Insert operation with key 20 on H. Construct the arrays representations of H, H_1, and H_2 and state the content of each of the following entries (recall that a heap with n items in the array representation consists of an array of length n+1 where index 0 is not used).

	2	7	13	18	19	20	21	42
H[3]	○	○	○	○	○	○	●	○
H[4]	●	○	○	○	○	○	○	○
H[5]	○	○	●	○	○	○	○	○
H[6]	○	○	○	○	●	○	○	○
H_1[3]	○	○	○	○	●	○	○	○
H_1[4]	●	○	○	○	○	○	○	○
H_1[5]	○	○	●	○	○	○	○	○
H_1[6]	○	●	○	○	○	○	○	○
H_2[3]	○	○	○	○	○	○	●	○
H_2[4]	○	○	○	●	○	○	○	○
H_2[5]	○	○	●	○	○	○	○	○
H_2[6]	○	○	○	○	●	○	○	○

Union Find

Consider the following forest of trees representing a family of sets in a union find data structure constructed using the quick union algorithm.



State the result of the following Find(.) operations.

	0	1	2	3	4	5	6	7	8	9
Find(0)	○	○	○	○	●	○	○	○	○	○
Find(1)	●	○	○	○	○	○	○	○	○	○
Find(2)	○	○	●	○	○	○	○	○	○	○
Find(3)	○	●	○	○	○	○	○	○	○	○
Find(4)	○	○	○	○	○	○	●	○	○	○
Find(5)	○	○	○	○	○	○	○	○	●	○

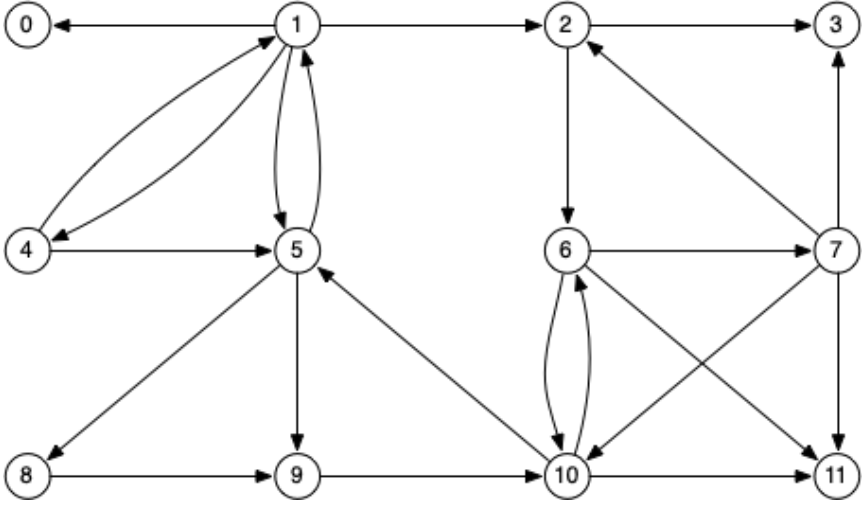
Suppose we now use path compression on the forest above. Construct the forest of trees F after a Find(5) operation and answer the following question for F.

	1	2	3	4	5	6	7	8
The total number of leaves in all trees in F is	○	○	○	●	○	○	○	○
The maximum depth of a tree in F is	○	○	○	○	○	●	○	○
The minimum depth of a tree in F is	○	○	○	○	●	○	○	○
The maximum degree of a root of a tree in F is	○	○	●	○	○	○	○	○
The minimum degree of a root of a tree in F is	●	○	○	○	○	○	○	○

Consider the following data structures for representing a set of integers. We are interested in supporting the operation Max-Min-Sum(), that returns the sum of the smallest and largest integer in the set. For instance, if a data structure stores the set {32, 6, 18, 7, 2} then Max-Min-Sum() should return 34. For each data structure below mark the runtime of the Max-Min-Sum() operation in O-notation as a function of n, where n is the number of elements in the data structure. Mark the best bound

	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$
Sorted array	●	○	○	○	○	○	○
Max-heap	○	○	○	●	○	○	○
Min-heap	○	○	○	●	○	○	○
Array (not sorted)	○	○	○	○	○	●	○
Binary search tree	○	○	○	●	○	○	○

Graph Search
Consider the following graph G.



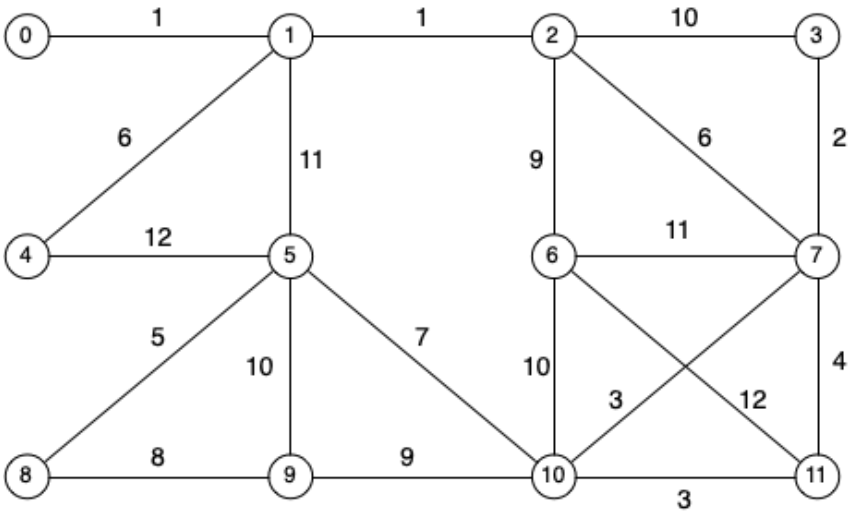
Construct the DFS tree T_1 for G when starting in vertex 4. Assume that each adjacency list is sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7	8	9
The depth of T_1 is	○	○	○	○	○	○	●	○	○
The number of leaves of T_1 is	○	○	●	○	○	○	○	○	○
The maximum number of children of a node in T_1 is	○	●	○	○	○	○	○	○	○

Construct the BFS tree T_2 for G when starting in vertex 7. Assume that each adjacency list is sorted in increasing order. Answer the following questions.

	1	2	3	4	5	6	7	8	9
The depth of T_2 is	○	○	○	●	○	○	○	○	○
The number of leaves of T_2 is	○	○	○	○	○	○	●	○	○
The maximum number of children of a node in T_2 is	○	○	○	●	○	○	○	○	○

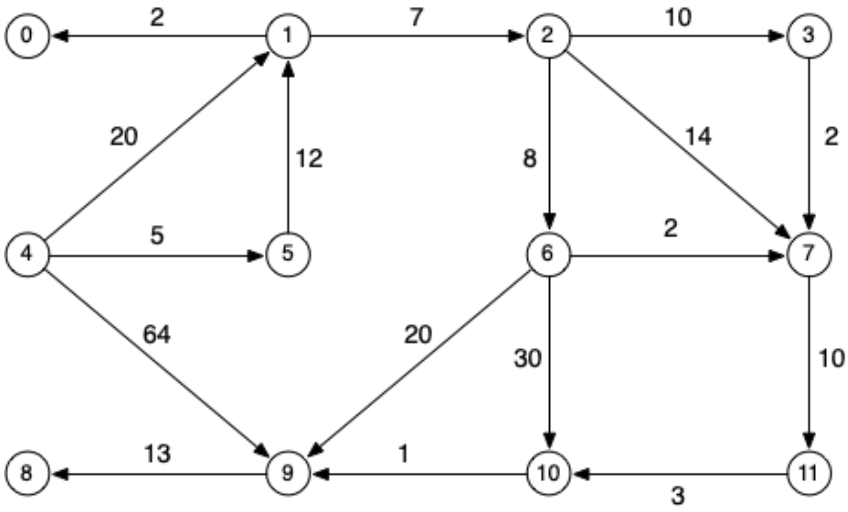
Consider the following graph.



Construct a minimum spanning tree T for the graph and answer the following questions.

	1	2	3	4	5	6	7	8	9
The total number of edges on a longest path of T is	○	○	○	○	○	○	●	○	○
The number of leaves of T is	○	○	○	○	○	○	●	○	○
The maximum degree of a node in T is	○	○	○	●	○	○	○	○	○

Consider the following graph



Construct a shortest path tree T for the graph when starting at vertex 4. Answer the following questions for T

	1	2	3	4	5	6	7	8	9
The total number of edges on a longest path in T is	○	○	○	○	○	○	○	●	○
The number of leaves in T is	○	○	○	●	○	○	○	○	○
The maximum number of children of a node in T is	○	●	○	○	○	○	○	○	○

Technical University of Denmark

Written examination, May 17, 2022.

Course name: Algorithms and Data Structures 1

Course number: 02105

Aids: all aids, open internet.

Duration: 2.5 hours.

Weight: Exercise 1 - 35%, exercise 2 - 25%. The weight is approximate. The grade is based on an overall assessment of the multiple-choice part, the implementation part, and the written part of the exam.

Instructions for exercise 1.

Write your answers to the exercises and submit them using the digital exam system. Your answer should be in the form of a `pdf`-file containing exactly 1 page in the a4-format. The top of the page should clearly list your full name and your study id. The rest of page should contain your answers to the each of the exercises. Number each of your answers with the number of the corresponding exercise. Use a clearly readable font of size 10pt or more and margins of 2cm or more.

Asymptotic bounds should be as tight as possible. Unless otherwise specified, the base of all logarithms is 2 and $\log^k n$ means $(\log n)^k$.

"Give an algorithm" means that you should describe your solution in a short, precise, and unambiguous manner and analyze the complexity of your solution. Unless specified otherwise, the description should be in natural language and not pseudocode. The analysis should explain how you derived the complexity bound.

"Argue correctness of your algorithm" means that you should provide a short argument for correctness of your algorithm.

"Analyze the running time of your algorithm in the relevant parameters (parameters x, y, ..) of the problem" means that you should analyze the running time using the explicitly stated parameters of the problem (parameters x, y, ..).

Instructions for exercise 2.

Read the exercise text in the CodeJudge system and implement a solution. Upload your solution to the CodeJudge system.

1 Skiing

We are interested in planning trip on a mountain at a ski resort. The ski resort consists of x *positions* $p_0, \ldots, p_{x-1}$ and y *ski slopes* $s_0, \ldots, s_{y-1}$. Each ski slope s is defined by the following:

- a *start position* $\mathsf{start}(s)$,
- an *end position* $\mathsf{end}(s)$, and
- a *completion time* $\mathsf{time}(s)$ that indicates how many minutes it takes to traverse s on ski.

Note that ski slopes may go both up or down the mountain.

1.1 Briefly describe how to model a ski resort as a graph.

1.2 We are interested in determining if it is possible to get to the bottom of the mountain before the ski slopes close. A *trip* from position a to position b is a sequence of positions starting in a and ending in b that are pairwise connected by ski slopes (i.e., each position p in the sequence is connected to the next position p' by a ski slope s such that $p = \mathsf{start}(s)$ and $p' = \mathsf{end}(s)$. The completion time of a trip is the sum of the completion time of each of the ski slopes of the sequence.

Give an algorithm that given a description of a ski resort and a time threshold t determines whether or not there is a trip from p_0 to p_{x-1} with completion time at most t. Argue correctness of your algorithm. Analyze the running time of your algorithm in terms of parameters x and y.

1.3 We now also store a *level* for each ski slope s, denoted $\mathsf{level}(s)$, that indicates the difficulty of traversing s. A high level for s indicates that s is difficult to traverse. Each level is an integer between 1 and x. Give an algorithm, that given a ski resort and a time threshold t determines the smallest level ℓ such that there is a trip from p_0 to p_{x-1} with completion time at most t that only uses ski slopes of level at most ℓ. Argue correctness of your algorithm. Analyze the running time of your algorithm in terms of parameters x and y.

2 Implementation Exercise

See the exercise text in the CodeJudge system.