Danmarks Tekniske Universitet

Skriftlig prøve, den 23. maj 2011.

Kursusnavn: Algoritmer og datastrukturer Kursus nr. 02326.

Varighed: 4 timer

Tilladte hjælpemidler: Alle skriftlige hjælpemidler.

Vægtning af opgaverne: Opgave 1 - 20%, Opgave 2 - 20%, Opgave 3 - 20%, Opgave 4 - 15%, 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.

Eksamenssættet fortsætter på næste side

Opgave 1 (kompleksitet)

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

	Ja	*Nej*
$\frac{1}{20}n^4 - 100n^3 = O(n^3)$	☐	x
$(\log n)^2 = O(n)$	x	☐
$\frac{1}{2}n^5 = O(n^4)$	☐	x
$2^n = O(3^n)$	x	☐
$n^4(n-1)/5 = O(n^5)$	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 $f(n) = O(g(n))$.

$5000(\log n)^2$

$4n$

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

$n^{1/100}$

$4n \log n$

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 (sæt kun ét kryds): Dit svar skal være så tæt som muligt

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

X $O(n^2)$ G $O(2^n)$ H $O(n^4)$ I $O(\sqrt{n})$

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:    $i = 2 \cdot i$
7: **end while**

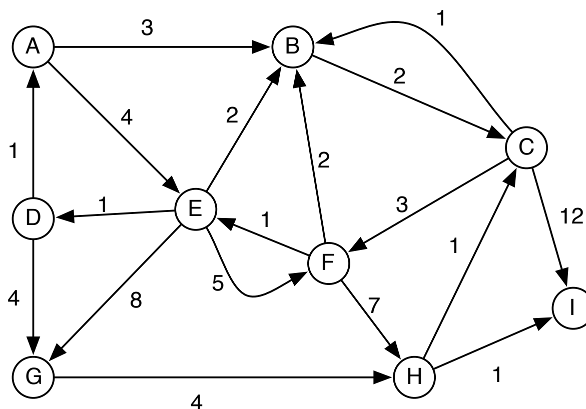
Køretiden af algoritmen er (sæt kun ét kryds): Dit svar skal være så tæt som muligt

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

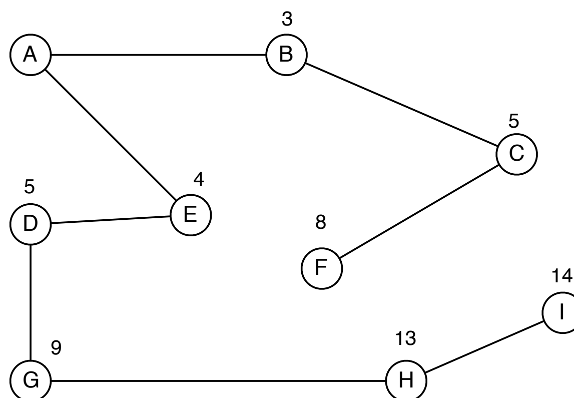
[F] $O(n^2)$ [G] $O(2^n)$ [H] $O(n^4)$ [I] $O(\sqrt{n})$

Opgave 2 (grafer)

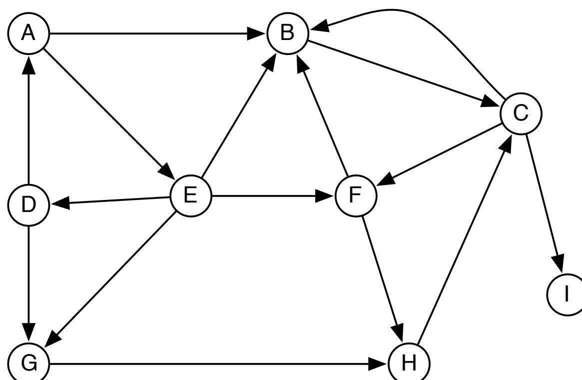
2.1 Angiv et korteste veje træ for nedenstående graf når korteste veje beregningen sker med hensyn til startknuden A. Angiv for hver knude afstanden fra knuden A.



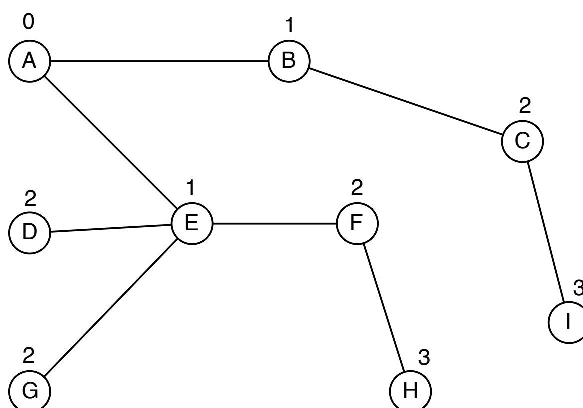
Angiv korteste veje træet og afstandene her:



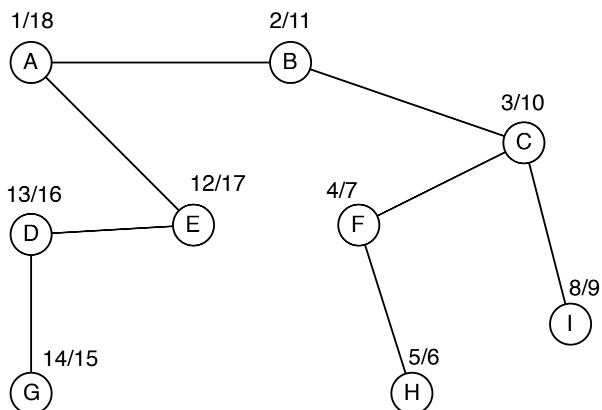
2.2 Betragt nedenstående graf G.



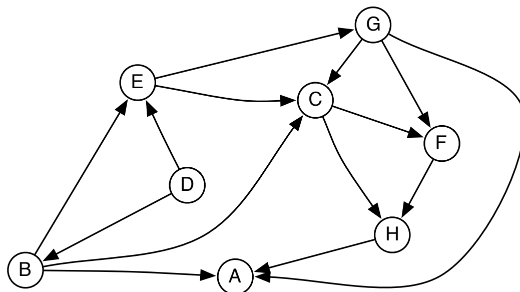
a) Angiv et BFS træ for grafen G når BFS genemløbet starter i knuden A. Angiv BFS-dybde/lag for hver knude. Det antages at incidenslisterne er sorteret i alfabetisk orden.



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.

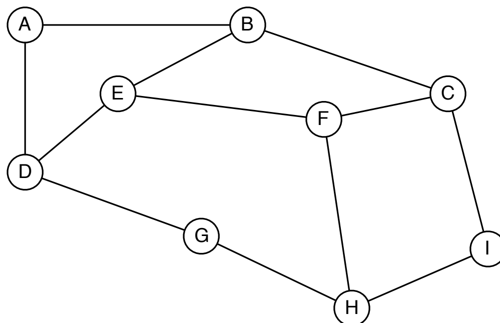


2.3 Angiv en topologisk sortering af knuderne i nedenstående graf.



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

2.4 Er nedenstående graf todelt? Hvis ja, så angiv en opdeling af knuderne. Hvis nej, så forklar hvorfor.



Grafen er ikke todelt da knuderne D, E, F, H, G danner en kreds med et ulige antal knuder.

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 SLET-LAVESTEINDKOMST 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 min-hobens 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 $\mathit{left}[x]$, og dens højre barn betegnes $\mathit{right}[x]$. Hvis knuden x ikke har nogle børn, har $\mathit{left}[x]$ og $\mathit{right}[x]$ 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 $\mathit{size}[x]$, der indeholder et heltal.

Betragt følgende algoritme:

Algorithm 3 ZERO(x)

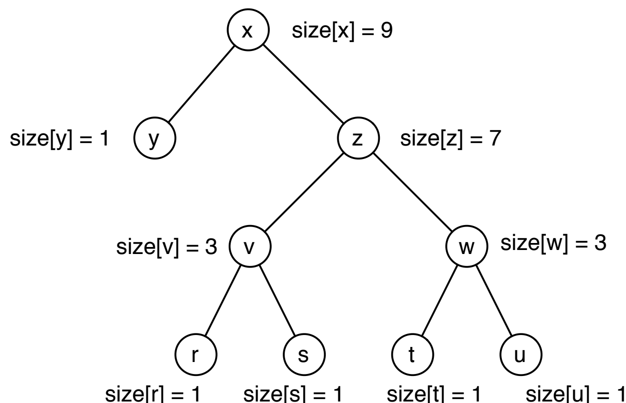
```
1: if (x ≠ NIL) then
2:     size[x] = 0
3:     ZERO(left[x])
4:     ZERO(right[x])
5: end if
```

Lad x være rodknuden for træet T. Efter udørelsen af proceduren ZERO(x) vil alle felterne $\mathit{size}[x]$ 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 Lad $T(x)$ betegne 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 $O(n)$. Angiv køretiden af din algoritme.

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 $O(n)$.

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

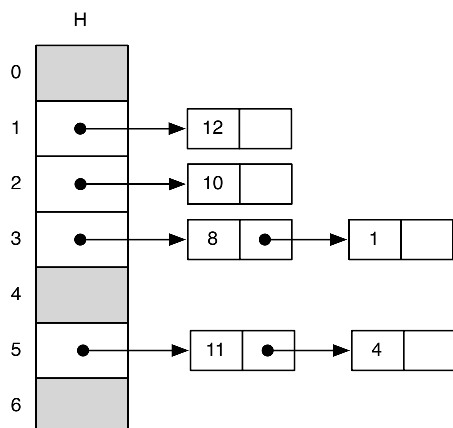
[X] D U T N R M

[D] D T U I N F

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

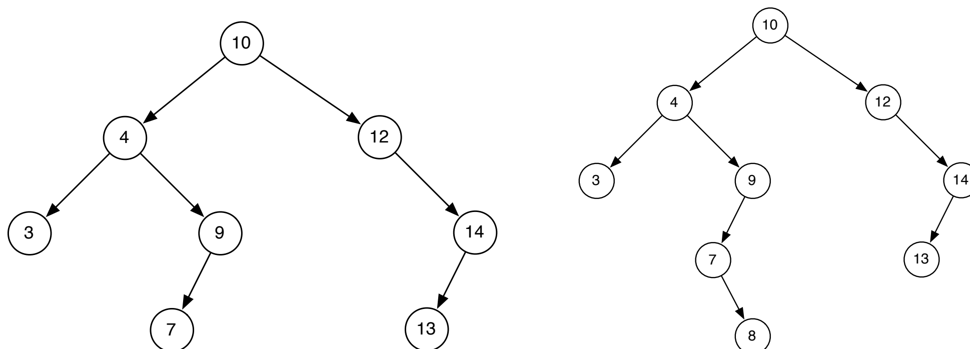
[F] D U T N O M

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.

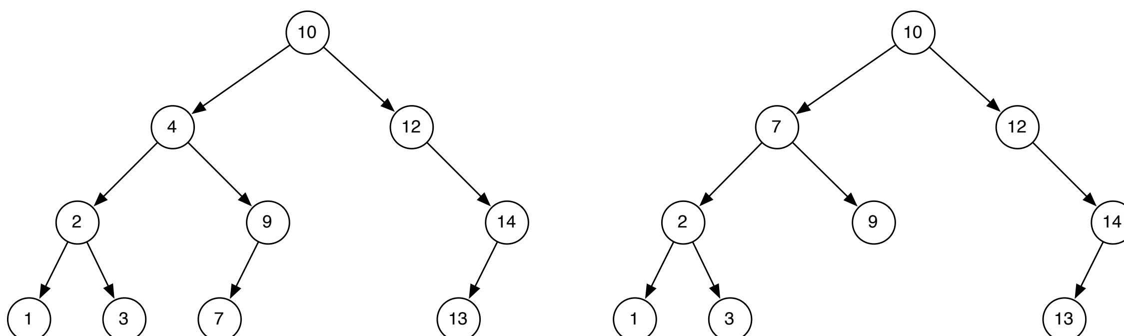


5.3 Denne opgave omhandler (ubalancerede) binære søgetræer, som beskrevet i de udleverede noter CLRS kapitel 12.

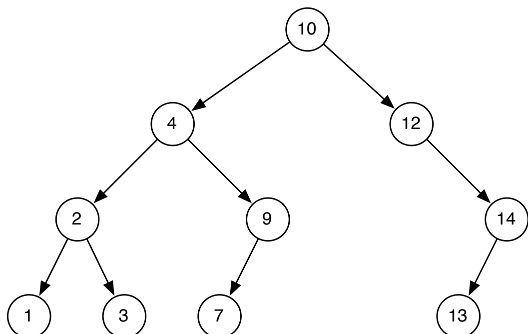
Opgave a Angiv hvordan det binære søgetræ nedenfor ser ud efter indsættelse af et element med nøgle 8.



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

5.4 Lad A være en sorteret tabel (array) af n heltal. Proceduren COUNT(x,y) tager to heltal x og y som argumenter, hvor $x < y$. Proceduren COUNT(x,y) skal returnere antallet af tal i tabellen der både er større end x og mindre end y.

Eksempel: Hvis $A = \langle 5, 10, 33, 49, 66, 75, 82, 95, 100 \rangle$, så returnerer COUNT(33,85) tallet 4.

Beskriv hvordan COUNT(x,y) kan implementeres (giv pseudokode eller forklaring), så den har kompleksitet $O(\log n)$.

Hvis du ikke kan få køretiden ned på $O(\log n)$, så angiv den bedst mulige implementation af proceduren (i tekst eller pseudokode), som du kan finde på

Vi laver først en binær søgning efter det største tal der er mindre end x. Lad i være indeks på det fundne tal. Herefter laver vi en binær søgning efter det mindste tal der er større end y. Lad j være indeks på det fundne tal. Algoritmen returnerer tallet $j - i + 1$, som er antallet af tal i A som både er større end x og mindre end y.

Algoritmen laver to binære søgninger, hvilket tager $2 \cdot O(\log n)$ tid. Til sidst laver algoritmen et konstant antal operationer for at beregne hvor mange tal der er mellem x og y ud fra i og j. Dvs. algoritmen har en køretid på $2 \cdot O(\log n) + O(1) = O(\log n)$.