Eksamen i Algoritmer og Datastrukturer (NDAA04010U)

(version med svar og forklaringer)

Københavns Universitet

2024-04-08

Instruktioner

Du har 4 timer til at besvare 21 opgaver, beskrevet nedenfor. Karakteren baseres på det samlede antal point (vægten af hver opgave er angivet) og på en helhedsbedømmelse. Opgavesættet har 19 nummererede sider. CLRS refererer til kursets lærebog, Introduction to Algorithms, 4th edition. Visse opgaver refererer til digitale kapitler fra CLRS 3rd edition, der ikke indgår i 4th edition.

Skriftlige hjælpemidler samt brug af lokalt installeret software er tilladt. Det er *ikke* tilladt at tilgå internet eller at kommunikere med andre.

Multiple-choice opgaver besvares i ITX systemets multiple-choice modul. I hvert multiple-choice spørgsmål er det enten angivet at der er præcis ét korrekt svar eller at der er en *mængde* af ét eller flere korrekte svar. I begge tilfælde er det tilladt at vælge mere end ét svar. Ved retning af multiple-choice opgaver gives der negative point for forkerte svar. Eksempler:

- Opgave X har 8 svarmuligheder, hvoraf 4 er korrekte. Hvert kryds ved en korrekt svarmulighed giver 1 point, og hvert kryds ved en forkert svarmulighed giver -1 point. Hvis du fx sætter kryds ved alle de korrekte svarmuligheder og én forkert svarmulighed får du 3 point.
- Opgave Y har 5 svarmuligheder, hvoraf 1 er korrekt. Kryds ved den korrekte svarmulighed giver 4 point, og kryds ved en forkert svarmulighed giver -1 point.

Hvis du ikke sætter nogen krydser får du altid 0 point, og sætter du krydser ved alle svarmuligheder får du også altid 0 point.

Svar på skriftlige opgaver afleveres som et Word dokument i ITX systemet. Der er mulighed for at tegne og skrive på tablet, og indsætte i dokumentet. Efter forsiden skal der være *én side til hvert delspørgmål, i samme rækkefølge som i opgavesættet.* Kopiér gerne den udleverede skabelon ind i besvarelsen. Delspørgsmålets nummer og bogstav (fx 19.a) skal fremgå. Totalt skal Word dokumentet have 10 sider — hvis du ikke svarer på et delspørgsmål skal der være en tom side.

Multiple-choice

Dine svar skal indtastes i ITX-systemets multiple-choice modul.

1 Sortering (4%)

Betragt en variant af MERGE-SORT der deler input op i *tre* dele, der er lige store (op til afrunding), sorterer hver del rekursivt, og derefter fletter de tre dele sammen med to kald til MERGE (CLRS sektion 2.3). Pseudokoden ser således ud:

```
MERGE3-SORT(A, p, s)
1  if p < s
2       q = ⌊(2 · p + s)/3⌋
3       r = ⌊(p + 2 · s)/3⌋
4       MERGE3-SORT(A, p, q)
5       MERGE3-SORT(A, q + 1, r)
6       MERGE3-SORT(A, r + 1, s)
7       MERGE(A, p, q, r)
8       MERGE(A, p, r, s)
```

Hvilken rekursionsligning beskriver køretiden $T(n)$ af MERGE3-SORT på et input af størrelse n (hvor $n = s - p + 1$)? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

1. $T(n) = T(n/2) + \Theta(n)$
2. $T(n) = T(2n/3) + \Theta(n)$
3. $T(n) = 2 \cdot T(n/3) + \Theta(n)$
4. $T(n) = 2 \cdot T(2n/3) + \Theta(n)$
5. $T(n) = 3 \cdot T(n/2) + \Theta(n)$
6. $T(n) = 3 \cdot T(n/3) + \Theta(n)$ ✓

Svar: De tre rekursive kald tager hver tid $T(n/3)$ mens de to kald til MERGE tager samlet tid $\Theta(n)$.

2 Rekursionsligninger (4%)

Hvilken af nedenstående rekursionsligninger, nummereret $i = 1, \ldots, 6$, har en løsning T_i hvor $T_i(n) = O(n^2)$? Antag at $T_i(n) = 1$ for $n \leq 1$. Vælg ét eller flere korrekte svar.

Eksempler. Hvis $T_0(n) = \Theta(n)$ gælder også $T_0(n) = O(n^2)$. Hvis $T_0(n) = \Theta(2^n)$ gælder *ikke* $T_0(n) = O(n^2)$.

1. $T_1(n) = T_1(n/2) + \Theta(n^2)$ ✓
2. $T_2(n) = 4 \cdot T_2(n/2) + \Theta(n^2)$
3. $T_3(n) = 3 \cdot T_3(n/2) + \Theta(n)$ ✓
4. $T_4(n) = T_4(n/3) + \Theta(n)$ ✓
5. $T_5(n) = 2 \cdot T_5(n/3) + \Theta(n)$ ✓
6. $T_6(n) = 2 \cdot T_6(n-2) + \Theta(n)$

Svar: Rekursionsligning 1–5 kan løses med master theorem og har løsninger henholdsvis $T_1(n) = \Theta(n^2)$, $T_2(n) = \Theta(n^2 \lg n)$, $T_3(n) = \Theta(n^{\lg_2 3})$, $T_4(n) = \Theta(n)$, og $T_5(n) = \Theta(n)$. Rekursionsligning 6 kan løses som $T_6(n) = \Theta(2^{n/2})$ ved hjælp af substitutionsmetoden.

3 Nedre grænser for sortering (4%)

MERGE-SORT og (deterministisk) QUICKSORT som beskrevet i CLRS sektion 2.3 og 7.1 er eksempler på sammenligningsbaserede sorteringsalgoritmer (*eng., comparison sort algorithms*). Hvilke konklusioner kan drages som følge af den nedre grænse for sortering, Theorem 8.1 i CLRS? (Du skal ikke tage stilling til om konklusionen er sand, blot om den er en logisk konsekvens af Theorem 8.1.)

1. Der findes et input hvor QUICKSORT kører i tid $O(n \lg n)$ i værste fald
2. Der findes et input hvor MERGE-SORT kører i tid $O(n \lg n)$ i værste fald
3. Der findes et input hvor QUICKSORT kører i tid $\Omega(n \lg n)$ i værste fald ✓
4. Der findes et input hvor MERGE-SORT kører i tid $\Omega(n \lg n)$ i værste fald ✓
5. I gennemsnit over alle input kører MERGE-SORT i tid $\Omega(n \lg n)$
6. I gennemsnit over alle input kører QUICKSORT i tid $O(n \lg n)$

Svar: Theorem 8.1 giver ikke nogen $O(n \lg n)$ øvre grænse på antal sammenligninger (og en sådan grænse findes ikke, da der jo eksisterer langsommere sorteringsalgoritmer). Det udelukker svar 1, 2 og 6. Derimod udtaler Theorem 8.1 sig om tiden for det værste input for såvel MERGE-SORT som QUICKSORT. Svarmulighed 5 er et korrekt udsagn, men det er ikke en konsevkens af Theorem 8.1, der kun udtaler sig om værste fald.

4 Løkkeinvarianter (4%)

Betragt følgende funktion, i pseudo-kode notationen fra CLRS, hvor input A er et array af reelle tal og n er længden af A.

VARIANCE(A, n)
1 $r = 0$
2 $s = 0$
3 **for** $i = 1$ **to** n
4 $\quad r = r + A[i] \cdot A[i]$
5 $\quad s = s + A[i]$
6 **return** $r/n - (s \cdot s)/(n \cdot n)$

Eksempel: For $n = 4$ og $A = \langle \frac{1}{2}, 2, 0, -\frac{1}{2} \rangle$ returnerer VARIANCE værdien $7/8$.

Hvilke udsagn er gyldige løkke-invarianter, der er sande efter udførelsen af linje 5 i **for**-løkken? Vælg ét eller flere korrekte svar.

1. $s \leq r$
2. $s \geq 0$
3. $r \geq 0$ ✓
4. $i \geq 0$ ✓
5. $s = \sum_{j=1}^{n} A[j]$
6. $r = \sum_{j=1}^{i} A[j]^2$ ✓
7. $i \leq n$ ✓

Svar: Udsagn 1 gælder ikke efter 1 iteration hvis $A[1] = 1/2$. Udsagn 2 gælder ikke efter 1 iteration hvis $A[1] = -1/2$. Udsagn 6 gælder per induktion pga. initialiseringen af $r = 0$ og da r øger sin værdi med $A[i]^2$ i iteration i. Fordi $A[i] \cdot A[i] \geq 0$ medfører det at udsagn 3 er korrekt. Udsagn 5 gælder efter den sidste iteration, men er ikke en invariant. Udsagn 4 og 7 er invarianter da løkken starter med $i = 1$ og terminerer når $i = n$.

5 Invarianter for dynamiske tabeller (4%)

Betragt dynamiske tabeller (*eng., dynamic tables*) som beskrevet i CLRS sektion 16.4.1, altså versionen *uden* en TABLE-DELETE operation. Hvilke af følgende er gyldige invarianter for algoritmen efter hver TABLE-INSERT operation? Vælg ét eller flere korrekte svar.

1. $T.size = 0$

2. $T.size \leq T.num$

3. $T.size \leq 2 \cdot T.num$ ✓

4. $T.num = T.size$

5. $T.num \leq T.size$ ✓

6. $T.num \leq 2 \cdot T.size$ ✓

Svar: Udsagn 1 og 4 er betingelser der evalueres i TABLE-INSERT og ikke invarianter. Udsagn 5 er en invariant fordi den gælder initielt (når $T.size = 0$)og fordi $T.size$ fordobles når $T.size = T.num$, inden værdien af $T.num$ øges. Dette medfører at udsagn 6 også er sandt. Udsagn 2 er forkert og gælder fx ikke efter den anden fordobling når $T.size = 4$ og $T.num = 3$. Udsagn 3 gælder fordi det er sandt efter den første indsættelse ($T.num = T.size = 1$), og da $T.num$ kun fordobles når $T.num = T.size$, mens $T.size$ aldrig bliver mindre (fordi vi ikke har TABLE-DELETE).

6 Amortiseret analyse (4%)

Betragt dynamiske tabeller (*eng., dynamic tables*) som beskrevet i CLRS sektion 16.4.2, altså versionen *med* en TABLE-DELETE operation. Vi betragter potentialfunktionen defineret i CLRS ligning (16.5), hvor $\alpha(T) = T.num/T.size$:

$$\Phi(T) = \begin{cases} 2(T.num - T.size/2) \text{ if } \alpha(T) \geq 1/2, \\ T.size - T.num \text{ if } \alpha(T) < 1/2 \ . \end{cases}$$

Hvilke af følgende udsagn er sande? Vælg ét eller flere korrekte svar.

1. En TABLE-INSERT operation øger altid værdien af $\Phi(T)$

2. En TABLE-DELETE operation øger altid værdien af $\Phi(T)$

3. En TABLE-INSERT operation mindsker altid værdien af $\Phi(T)$

4. En TABLE-DELETE operation mindsker altid værdien af $\Phi(T)$

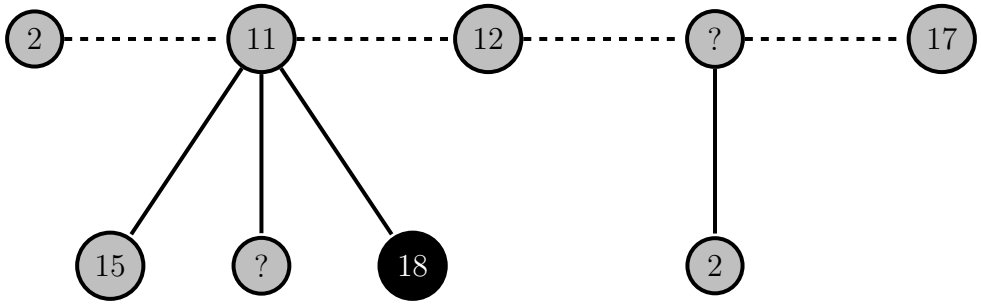
5. Værdien af $\Phi(T)$ er altid større eller lig med nul ✓

6. Værdien af $\Phi(T)$ er altid mindre end eller lig med $T.size$ ✓

Svar: Udsagn 1-4 er falske fordi effekten på potentialet af en operation afhænger af $\alpha(T)$, med en mindskning i det ene tilfælde og en øgning i det andet tilfælde. Udsagn 5 er sandt da $2T.num \geq T.size \geq T.num$. Udsagn 6 er sandt da $2(T.num - T.size/2) \leq 2(T.size - T.size/2) = T.size$.

7 Fibonaccihobe (4%)

Denne opgave omhandler Fibonaccihobe (*eng., Fibonacci heaps*) som beskrevet i CLRS digitalt kapitel 19 (dvs. min-heap ordnede). Som i CLRS figur 19.2.a illustreres rodlisten med stiplede linjer og markerede knuder tegnes med sort. Hvilke nøgler er mulige på mindst én af knuderne markeret med spørgsmålstegn? Vælg ét eller flere korrekte svar.



1. 2 ✓
2. 6
3. 10
4. 14 ✓
5. 18 ✓
6. 22 ✓

Svar: Knuderne skal overholde min-heap egenskaben, dvs. den ukendte knude til venstre skal indeholde en værdi som er større end eller lig med forældreknuden 11, og den ukendte knude til højre skal indeholde en værdi som er mindre end eller lig med dens børns værdier, dvs. 2. Derfor er 6 og 10 de eneste værdier, der ikke passer i nogen af knuderne.

8 Tidskompleksitet for Fibonaccihobe (4%)

Antag at der laves en Fib-Heap-Extract-Min operation på en Fibonaccihob H(*eng., Fibonacci heap*) med n nøgler som beskrevet i CLRS digitalt kapitel 19. Før operationen har potentialfunktionen værdien $\Phi(H)$ og efter operationen har den værdien $\Phi(H')$, hvor H' er den nye Fibonaccihob. Hvad kan vi med sikkerhed sige om tidsforbruget af Fib-Heap-Extract-Min operationen der omdanner H til H'? Vælg ét eller flere korrekte svar.

1. Tiden er $O(n)$ ✓
2. Tiden er $O(\lg n)$

3. Tiden er $O(\lg n + \Phi(H) - \Phi(H'))$ ✓

4. Tiden er $O(1 + \Phi(H) - \Phi(H'))$

5. Tiden er $O(1)$

Svar: Den amortiserede analyse i CLRS viser at tidsforbruget er højst (en konstant gange med) $\lg n + \Phi(H) - \Phi(H')$. Da $0 \leq \phi(H) \leq n$ medfører det en tidsgrænse på $O(n)$, dvs. svarmulighed 1 og 3 er korrekte. Svarmulighed 2 og 5 er ikke korrekte hvis rodlisten er længere end $O(\lg n)$, hvilket kan ske efter en række indsættelser, da CONSOLIDATE løber igennem rodlisten. Svarmulighed 4 er ikke korrekt hvis der er en knude af grad $\lg n$ i rodlisten, da CONSOLIDATE bruger tid proportionalt med graden, men ikke nødvendigvis mindsker potentialet.

9 Stavudskæring (4%)

Betragt algoritmen EXTENDED-BOTTOM-UP-CUT-ROD, CLRS sektion 14.1, side 372. Lad $n = 6$ og $p = \langle 4, 4, 13, 40, 43, 49 \rangle$. Hvilke arrays r og s returnerer algoritmen når den kaldes på input (p, n)? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

1. $r = \langle 0, 4, 8, 13, 40, 44, 48 \rangle$ og $s = \langle 1, 1, 3, 4, 1, 4 \rangle$

2. $r = \langle 0, 4, 8, 13, 40, 44, 48 \rangle$ og $s = \langle 1, 1, 3, 4, 1, 6 \rangle$

3. $r = \langle 0, 4, 8, 13, 40, 44, 49 \rangle$ og $s = \langle 1, 1, 3, 4, 1, 4 \rangle$

4. $r = \langle 0, 4, 8, 13, 40, 44, 49 \rangle$ og $s = \langle 1, 1, 3, 4, 1, 6 \rangle$ ✓

Svar: Ses ved at håndkøre algoritmen.

10 Paradigmer (4%)

Betragt følgende pseudokode, der tager et array A og to indekser p, r som parametre.

```
MyAlgorithm(A, p, r)
1  if p < r
2       q = ⌊(p + r)/2⌋
3       x = MyAlgorithm(A, p, q)
4       y = MyAlgorithm(A, q + 1, r)
5       z = MyAlgorithm(A, p + 1, r)
6       return x + y + z
7  else
8       return A[p]
```

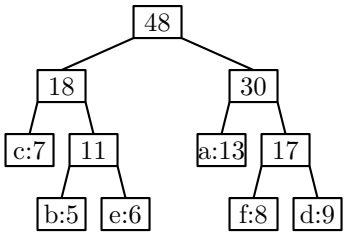
Hvilket af nedenstående paradigmer beskriver bedst MyAlgorithm? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

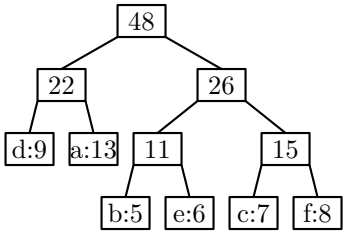
1. Del og kombiner (eng. *divide and conquer*) ✓
2. Dynamisk programmering (eng. *dynamic programming*)
3. Grådige algoritmer (eng. *greedy algorithms*)

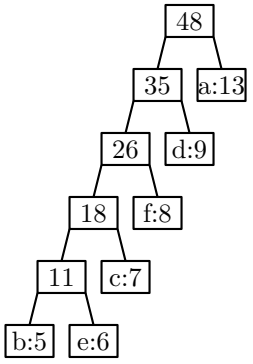
Svar: Algoritmen virker ved at kalde sig selv rekursivt på tre mindre probleminstanser og derefter kombinere resultaterne. Den bruger derfor del og kombiner. Der gemmes ikke nogen resultater, så koden bruger ikke dynamisk programmering. Der foregår heller ikke noget grådigt valg (ingen betingelser afhænger af indholdet i A), så det passer heller ikke godt med en grådig algoritme.

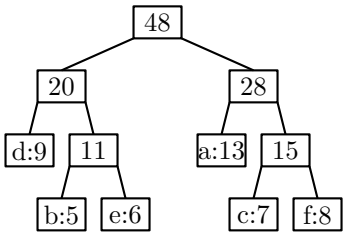
11 Huffman-koder (4%)

Betragt bogstaverne a, b, c, d, e, f og tilhørende frekvenser a:13, b:5, c:7, d:9, e:6, f:8. Vi kører Huffmans algoritme på frekvenserne, som beskrevet i CLRS afsnit 15.3. Hvilket træ får vi som resultat? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

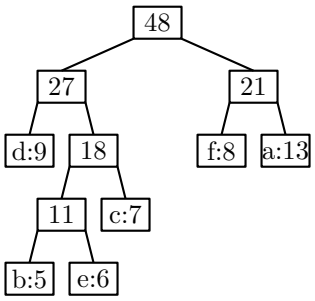
1.


2.


3.


4. 

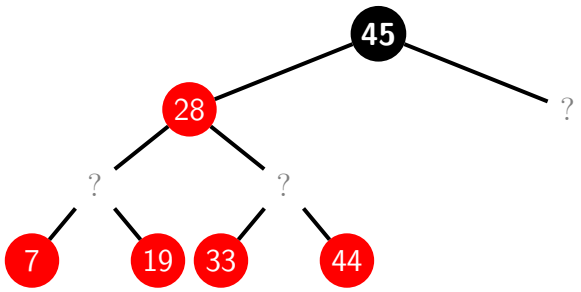
✓

5. 

Svar: Ses ved at håndkøre algoritmen.

12 Rød-sorte søgetræer (4%)

Betragt et rød-sort binært søgetræ T (eng. *red-black binary search tree*), som beskrevet i CLRS kapitel 13. Betragt nedenstående træ, hvor alle blade er udeladt (som på CLRS figur 13.1.c) og tre indre knuder er uspecificerede:



Hvilke knuder er mulige på mindst én af positionerne markeret med ?, hvis træet skal overholde invarianterne for et rød-sort binært søgetræ? Vælg ét eller flere korrekte svar.

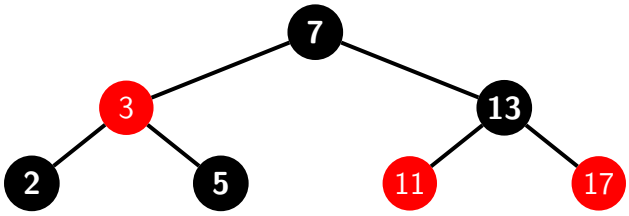
1. En rød knude med nøglen 13
2. En sort knude med nøglen 17 ✓
3. En rød knude med nøglen 29
4. En sort knude med nøglen 37 ✓
5. En rød knude med nøglen 48

6. En sort knude med nøglen 84 ✓

Svar: Alle de markerede knuder skal være sorte. De to til venstre fordi deres forælder er rød. Den sidste fordi der skal være lige mange sorte knuder på alle stier fra roden til bladene. Nøglerne 17, 37 og 84 er alle konsistente med invarianten at træet er et søgetræ (17 ligger mellem 7 og 19, 37 mellem 33 og 44, 84 er større end 45).

13 Rotationer i rød-sorte søgetræer (4%)

Betragt dette rød-sorte binære søgetræ T (eng. *red-black binary search tree*), hvor bladene er udeladt på tegningen:



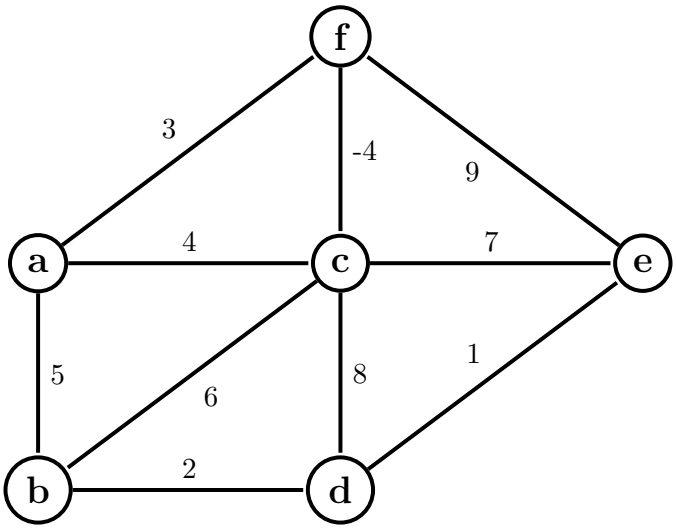
Antag at vi kører algoritmen LEFT-ROTATE$(T, 7)$ beskrevet i CLRS kapitel 13.2, altså med knuden 7 som input. Hvad sker med træet T? Vælg ét eller flere korrekte svar.

1. 3 bliver den nye rodknude og farves sort
2. 13 bliver den nye rodknude ✓
3. Højden forbliver den samme
4. Højden stiger med 1 ✓

Svar: LEFT-ROTATE$(T, 7)$ gør 13 til rod med 7 og 17 under sig. Knude 11 bliver højre barn til 7. Ingen knuder skifter farve. Efter rotationen er knude 2 og 5 et skridt længere væk fra roden end inden.

14 Mindste udspændende træ (4%)

Betragt mindste udspændende træ (eng. *minimum spanning tree*) problemet på denne graf:



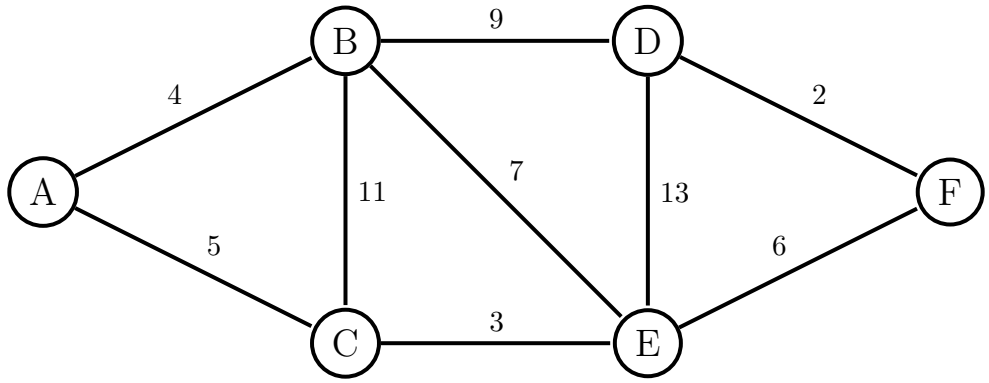
Hvilke kanter er med i det mindste udspændende træ (som er unikt)?

1. {a,b} ✓
2. {a,c}
3. {a,f} ✓
4. {b,c}
5. {b,d} ✓
6. {c,d}
7. {c,e}
8. {c,f} ✓
9. {d,e} ✓
10. {e,f}

Svar: Mindste udspændende træ kan findes ved at køre fx Kruskál's eller Prim's algoritme. Kruskal's algoritme tilføjer kanterne i vægtrækkefølge, dvs. starter med {c,f}, {d,e}, {b,d}, {a,f}, springer over {a,c} fordi det ville skabe en kreds, og afslutter med {a,b}.

15 Snit og sikre kanter (4%)

Betragt følgende ikke-orienterede, vægtede og sammenhængende graf $G = (V, E)$ med knudemængden $V = \{A, B, C, D, E, F\}$: Vi er interesserede i lette kanter (eng. *light edges*) for en kantmængde K i forskellige snit (eng. *cuts*). Hvilke af følgende udsagn er sande for kantmængden $K = \{\{D, F\}, \{C, E\}, \{A, B\}\}$? Vælg ét eller flere korrekte svar.

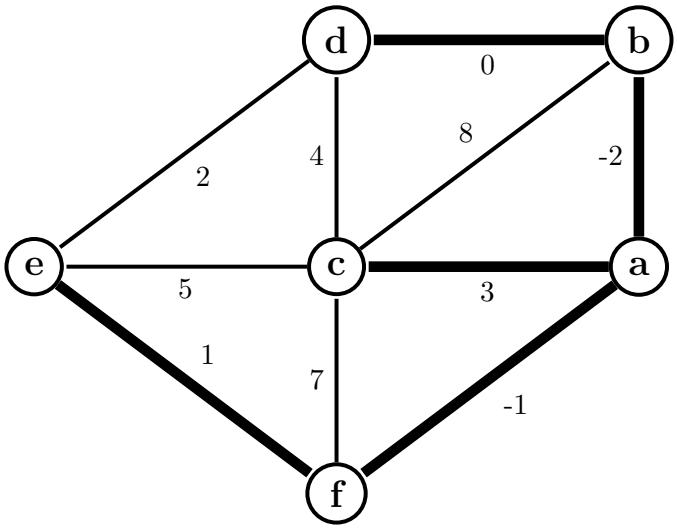


1. Snittet $(S_1, V - S_1)$ respekterer kantmængden K for $S_1 = \{A, B\}$ ✓
2. Snittet $(S_2, V - S_2)$ respekterer kantmængden K for $S_2 = \{C, D, E, F\}$ ✓
3. Snittet $(S_3, V - S_3)$ respekterer kantmængden K for $S_3 = \{A, B, F\}$
4. Snittet $(S_4, V - S_4)$ har $\{E, F\}$ som en let kant for $S_4 = \{A, B, C, E\}$ ✓
5. Snittet $(S_5, V - S_5)$ har $\{B, D\}$ som en let kant for $S_5 = \{C, D, E, F\}$
6. Snittet $(S_6, V - S_6)$ har $\{A, C\}$ som en let kant for $S_6 = \{A, B, D, F\}$ ✓

Svar: Ingen af kanterne incidente med kun én af knuderne A og B tilhører K. Derfor respekterer snittet $(S_1, V - S_1)$ K. Ingen af kanterne incidente med én af knuderne C, D, E, F tilhører K. Derfor respekterer snittet $(S_2, V - S_2)$ K. Kanten $\{D, F\}$ har F i S_3 og D i $V - S_3$. Derfor respekterer snittet $(S_3, V - S_3)$ ikke K. Kanten $\{E, F\}$ har mindste vægt i snittet $(S_4, V - S_4)$ og er derfor en let kant. Kanten $\{B, E\}$ som krydser $(S_5, V - S_5)$ har mindre vægt end kanten $\{B, D\}$ som også krydser samme snit. $\{B, D\}$ er derfor ikke en let kant. Kanten $\{A, C\}$ krydser snittet $(S_6, V - S_6)$ og har mindst vægt i snittet; den er derfor en let kant for snittet.

16 Kruskal's algoritme (4%)

Betragt følgende vægtede, uorienterede graf, hvor kanterne i det mindste udspændende træ (som er unikt) er markeret med fede linjer:



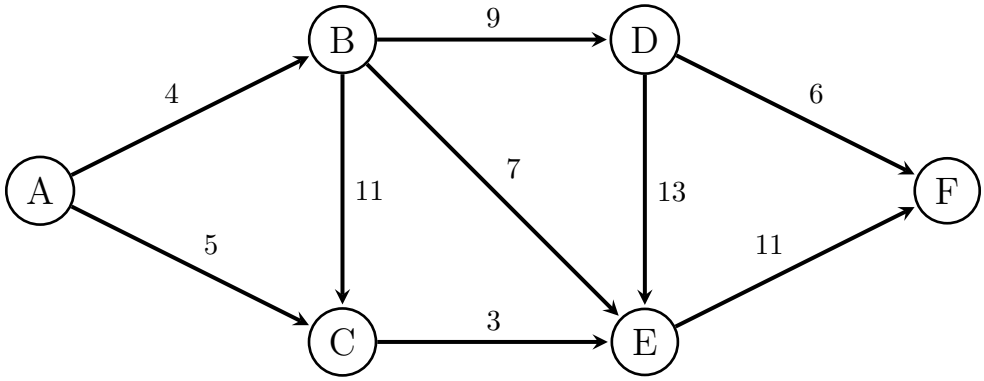
I hvilken rækkefølge bliver kanterne inkluderet i det mindste udspændende træ når Kruskal's algoritme køres på denne graf? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

1. {a,b}, {a,c}, {a,f}, {b,d}, {e,f}
2. {a,b}, {a,f}, {b,d}, {e,f}, {a,c} ✓
3. {b,d}, {a,f}, {e,f}, {a,b}, {a,c}
4. {b,d}, {a,c}, {e,f}, {a,b}, {a,f}
5. {b,d}, {a,f}, {e,f}, {a,b}, {a,c}

Svar: Håndkørsel af Kruskal's algoritme og giver den angivne rækkefølge.

17 Korteste veje (4%)

Betragt følgende orienterede, vægtede graf $G = (V, E)$:



Hvilke kanter er med i en korteste vej (eng. *shortest path*) fra knude A til *mindst* én af knuderne {B,C,D,E,F}? Vælg ét eller flere korrekte svar.

1. {A,B} ✓
2. {A,C} ✓
3. {B,C}
4. {B,D} ✓
5. {B,E}
6. {C,E} ✓
7. {D,E}
8. {D,F} ✓
9. {E,F} ✓

Svar: De fleste kanter på en korteste vej er i korteste-vej træet, der fx kan findes ved at håndkøre Dijkstra's algoritme. I det sidste skridt er der dog to mulige kanter til F der begge resulterer i en korteste vej, så de skal begge inkluderes.

18 Valg af algoritmer og datastrukturer (4%)

Antag at du skal løse k-DIFFERENCE problemet, defineret således: Givet en parameter k og et array A med n heltal, afgør om der findes to tal i A med en difference på k.

Eksempel. Hvis $k = 4$ og $A = \langle 2, 7, 13, 3, 11, 17 \rangle$ er svaret "ja", fordi vi har $A[2] - A[4] = 7 - 3 = 4$. Hvis $k = 3$ og A er uændret er svaret "nej" da der ikke findes to tal i A med en difference på 3.

Hvilken af følgende algoritmer og datastrukturer er mest relevant som en del af en simpel og effektiv algoritme, der løser k-DIFFERENCE? Der er præcis ét korrekt svar, men det er tilladt at vælge flere svar hvis du er i tvivl.

1. Dijkstra's algoritme
2. Disjunkte mængder
3. Dynamiske tabeller
4. Merge-sort ✓
5. Prim's algoritme

Svar: Ved først at sortere A kan problemet løses i yderligere $O(n)$ tid ved at lave MERGE af $\langle A[1], \ldots, A[n] \rangle$ og $\langle A[1]+k, \ldots, A[n]+k \rangle$ og bemærke om der findes identiske elementer i de to lister. (Alternativt kan der laves n binære søgninger efter et element med værdi $A[i]-k$, for $i = 1, \ldots, n$.) I begge tilfælde tager algoritmen totalt tid $O(n \lg n)$. De øvrige nævnte algoritmer og datastrukturer kan ikke umiddelbart bruges som en del af en simpel og effektiv algoritme.

Skriftlige opgaver

19 Induktionsbeviser (7%)

Betragt følgende funktion, i pseudo-kode notationen fra CLRS, hvor input A er et array af heltal, n er længden af A, og b er en heltalsparameter.

HeavyHitter(A, n, b)

```
1  let r[1 : b] be a new array
2  for i = 1 to b
3        r[i] = 0
4  for j = 1 to n
5        if A[j] ≥ 1 and A[j] ≤ b
6              r[A[j]] = r[A[j]] + 1
7  return max_{1≤ℓ≤b}(r[ℓ])
```

Eksempel: For $n = 6$, $b = 10$, $A = \langle 5, 1, 1, 7, 1, 3\rangle$ returnerer HeavyHitter værdien 3.

a) Forklar i ord hvad HeavyHitter beregner. Skriv dit svar på side 2 i Word dokumentet.

Svar: HeavyHitter beregner hvor mange gange det hyppigste element i A forekommer, hvor elementer i A der ikke ligger i intervallet $[1, \ldots, b]$ ignoreres.

b) Formulér en løkkeinvariant $I(j)$ der beskriver indholdet af r efter iteration j af **for**-løkken i linje 4–6. Argumentér ved induktion for at invarianten gælder efter j iterationer, for $j \geq 0$. Invarianten $I(n)$, der gælder når løkken afsluttes skal underbygge dit svar i spørgsmål **a)**. Skriv dit svar på side 3 i Word dokumentet.

Svar: Invariant $I(j)$: For $i \in \{1, \ldots, b\}$ gælder $r[i] = |\{k \mid 1 \leq k \leq j \wedge A[k] = i\}|$. Invarianten gælder for $j = 0$ da $r[i] = 0$ (pga. initialiseringen i linje 2-3). I induktionsskridtet antager vi $I(j-1)$ og vil vise at $I(j)$ gælder efter iteration j. Værdien af $r[i]$ øges med 1 i iteration j præcis hvis $A[j] = i$, og det samme gælder for $|\{k \mid 1 \leq k \leq j \wedge A[k] = i\}|$.

20 Så længe som muligt (12%)

Vi betragter følgende variant af problemet aktivitetsudvælgelse (eng. *activity selection*) fra CLRS sektion 15.1. I stedet for at maksimere antallet af aktiviteter som i CLRS, vil vi vælge en mængde af aktiviteter, der samlet varer *så lang tid som muligt*. Vi er altså givet en mængde $S = \{a_1, a_2, \ldots, a_n\}$ af n elementer, hvor hver aktivitet a_i har en givet starttid s_i og en sluttid f_i, så $0 \leq s_i < f_i < \infty$. Antag at aktiviterne er sorteret efter sluttider, dvs. $f_1 \leq f_2 \leq \cdots \leq f_n$. Vi ønsker at vælge en mængde $A \subseteq S$ af ikke-overlappende aktiviteter som maksimerer $\sum_{a_i \in A}(f_i - s_i)$. At aktiviteterne i A er ikke-overlappende vil

sige at hvis $a_i, a_j \in A$ hvor $i \neq j$, så gælder $[s_i, f_i) \cap [s_j, f_j) = \emptyset$, dvs. det er tilladt at $f_i = s_j$ eller $f_j = s_i$.

a) Antag at aktiviteterne er $S = \{a_1, a_2, a_3, a_4\}$ med starttider $s = \langle 0, 3, 1, 6\rangle$ og sluttider $f = \langle 2, 4, 5, 8\rangle$. Angiv en optimal løsning A. Skriv dit svar på side 4 i Word dokumentet.

Svar: Den optimale løsning er $\{a_3, a_4\}$, som har samlet varighed 6.

For hvert $i \in \{0, 1, \ldots, n\}$, lad $S_i = \{a_1, a_2, \ldots, a_i\}$ være mængden bestående af de i første aktiviteter, dvs. $S_0 = \emptyset$, $S_1 = \{a_1\}$, $S_2 = \{a_1, a_2\}$, osv. Definér t_i til at være den samlede tid for aktiviteterne i en optimal løsning til S_i, hvor specielt $t_0 = 0$.

b) Betragt et fast indeks $i \in \{1, \ldots, n\}$ og lad $j \in \{0, 1, \ldots, i-1\}$ være det største indeks hvor $f_j \leq s_i$ (her definerer vi $f_0 = 0$). Argumentér for at

$$t_i = \max\{t_{i-1}, t_j + f_i - s_i\}.$$

Skriv dit svar på side 5 i Word dokumentet.

Svar: Enten inkluderer den optimale løsning til S_i aktiviteten a_i, eller også gør den ikke. I det første tilfælde skal vi udover a_i vælge en optimal løsning blandt aktiviteter der slutter senest når a_i starter, altså blandt S_j, hvor j er defineret som i opgaveteksten. I det andet tilfælde må løsningen være den samme som den optimale til S_{i-1}, altså t_{i-1}. Dette viser at $t_i \leq \max\{t_{i-1}, t_j + f_i - s_i\}$. Begge de to kandidater er gyldige løsninger til S_i, så vi har også $t_i \geq \max\{t_{i-1}, t_j + f_i - s_i\}$. Derfor er $t_i = \max\{t_{i-1}, t_j + f_i - s_i\}$.

c) Skriv pseudokode til en algoritme som returnerer den samlede tid t_n af aktiviteterne i den optimale løsning til hele mængden S. Signaturen på din pseudokode skal være

$$\textsc{Activity-Selection-Max-Span}(s, f, n),$$

hvor s og f er arrays af længde n med hhv. start- og sluttider. For at få fuldt point skal algoritmen have køretid $O(n^2)$, men det er ikke nødvendigt at bevise grænsen for køretiden. Skriv dit svar på side 6 i Word dokumentet.

Svar: Ifølge del b) kan vi bruge dynamisk programmering. Her er en mulig løsning:

```
ACTIVITY-SELECTION-MAX-SPAN(s, f, n)
  Let t[0 : n] be a new array.
  t[0] = 0
  for i = 1 to n
    Use binary search to find the max. j ∈ {0, 1, ..., i − 1} such that j == 0 or f[j] ≤ s[i].
    t[i] = max{t[i − 1], t[j] + f[i] − s[i]}
  return t[n]
```

d) Forklar hvordan man kan lave en algoritme som løser problemet i $O(n \lg n)$ tid. Hvis din algoritme fra opgave c) allerede har denne køretid så argumentér for det. Skriv dit svar på side 7 i Word dokumentet.

Svar: Algoritmen fra spørgsmål c) har køretid $O(n \lg n)$ fordi for-løkken itererer over n værdier og vi bruger $O(\lg n)$ tid på binær søgning i hver iteration.

21 Så mange som muligt (9%)

Vi betragter et array af n heltal $X = \langle x_1, x_2, \ldots, x_n \rangle$, hvor x_i kan være positivt, negativt eller 0. Lad $S = \sum_{i=1}^{n} x_i$ være tallenes sum. Vi ønsker at løse LONGEST-SUBSET-SUM problemet: Find størrelsen på den største indeksmængde $I \subseteq \{1, 2, \ldots, n\}$ hvor $\sum_{i \in I} x_i \leq S/2$.

Eksempel. For $X = \langle 1, 4, 2, 8 \rangle$, der har $S = 15$, kan vi finde en mængde I med 3 elementer, nemlig $I = \{1, 2, 3\}$. Dette er den størst mulige mængde der overholder $\sum_{i \in I} x_i \leq S/2$ for ingen løsning kan indeholde alle fire elementer.

a) Giv et eksempel på et array X hvor løsningen ikke er entydig, dvs. hvor der findes to *forskellige* indeksmængder I_1 og I_2 af højst mulige størrelse hvor $\sum_{i \in I_1} x_i \leq S/2$ og $\sum_{i \in I_2} x_i \leq S/2$. Skriv dit svar på side 8 i Word dokumentet.

Svar: Eksempel: $X = \langle 1, 1 \rangle$ har to optimale løsninger: $I = \{1\}$ og $I = \{2\}$, begge af størrelse 1.

Vi ser nu på følgende tilgang til at løse LONGEST-SUBSET-SUM: Start med summen af alle tal i X og fjern ét tal ad gangen i omvendt sorteret rækkefølge så længe summen er højere end $S/2$. Returnér antallet af elementer, der ikke er fjernet fra X.

Eksempel. For $X = \langle 4, -2, -3, 2, 5, 5 \rangle$, der har sum $S = 11$, fjerner vi de to største elementer 5 og 5, hvorefter summen af de fire resterende tal er 1, hvilket er mindre end eller lig med $S/2$. Derfor returneres 4.

b) Bevis at en optimal løsning til LONGEST-SUBSET-SUM kan konstrueres som beskrevet ovenfor. Skriv dit svar på side 9 i Word dokumentet.

Svar: Antag at X er sorteret i ikke-faldende rækkefølge, så den foreslåede algoritme vil fjerne tallene fra summen i rækkefølgen $x_n, x_{n-1}, \ldots, x_1$. Betragt en vilkårlig indeksmængde $I^* \subseteq \{1, 2, \ldots, n\}$ som giver en optimal løsning, og lad $k = \mid I^* \mid$ være antallet af tal i I^* og lad $I = \{1, 2, \ldots, k\}$. Hvis ikke $I^* = I$, så giver I en sum som er højst så stor som den for I^*, så I er også en optimal indeksmængde. Siden I er optimal, har vi $\sum_{i=1}^{k'} a_i > S/2$ for ethvert $k' \in \{k+1, \ldots, n\}$. Derfor finder den foreslåede algoritme det optimale antal k.

c) Skriv pseudokode til en algoritme der løser LONGEST-SUBSET-SUM. Signaturen til din algoritme skal være

$$\text{LONGEST-SUBSET-SUM}(x, n),$$

hvor x er et array af længde n. Skriv dit svar på side 10 i Word dokumentet.

Svar: Ifølge del b) giver en simpel grådig algoritme en optimal løsning. Pseudo-koden kan fx se således ud:

```
LONGEST-SUBSET-SUM(x, n)
  Sort x in non-decreasing order, for instance using MERGE-SORT.
  Let S be the sum of the numbers in x.
  R = S
  i = n
  while R > S/2
    R = R − x[i]
    i = i − 1
  return i
```