Eksamen i Algoritmer og Datastrukturer (NDAA04010U)

(version med svar og forklaringer)

Københavns Universitet

2023-04-11

Instruktioner

Du har 4 timer til at besvare 22 opgaver, beskrevet nedenfor. Karakteren baseres på det samlede antal point (vægten af hver opgave er angivet) og på en helhedsbedømmelse. Opgavesættet har 17 nummererede sider. CLRS refererer til kursets lærebog, Introduction to Algorithms, 3rd 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 angivet hvorvidt der er præcis én korrekt svarmulighed eller der skal angives en *mængde* af ét eller flere korrekte 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. Hvis du ikke sætter nogen krydser får du 0 point, og sætter du krydser ved alle svarmuligheder får du også 0 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.

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 laves *én side til hvert delspørgmål, i samme rækkefølge som i opgavesættet.* Angiv delspørgsmålets nummer og bogstav (fx 10.a) men kopiér ikke opgaveteksten ind i besvarelsen. Totalt skal Word dokumentet have 8 sider — hvis du ikke svarer på et delspørgsmål, så lav en tom side.

Multiple-choice

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

1 MergeSort og rekursion (4%)

Hvis MergeSort(A, p, r), implementeret som i CLRS sektion 2.3, bliver kaldt på 4 elementer (dvs. $r-p+1 = 4$) laves det første rekursive kald på en liste af 2 elementer, hvorfra det første rekursive kald laves til en liste med 1 element. På det tidspunkt hvor vi når til bunden af rekursionen er der altså 3 kald af MergeSort på rekursionsstakken samtidig. Hvis MergeSort(A, p, r) bliver kaldt på 5 elementer bliver der op til 4 kald samtidig på rekursionsstakken.

Antag nu at MergeSort(A, p, r) bliver kaldt på 42 elementer (dvs. $r - p + 1 = 42$). Hvor stort bliver det største antal samtidige kald til Merge-Sort, dvs. antal kald på rekursionsstakken? Vælg præcis ét svar.

1. 3 samtidige kald.
2. 4 samtidige kald.
3. 5 samtidige kald.
4. 6 samtidige kald.
5. 7 samtidige kald. ✓

Svar: Største antal elementer på et kald i rekursionsdybde 1,2,3,... er henholdsvis 42, 21, 11, 6, 3, 2 og 1. Dvs. der er op til 7 samtidige kald.

2 Korteste-veje træ (4%)

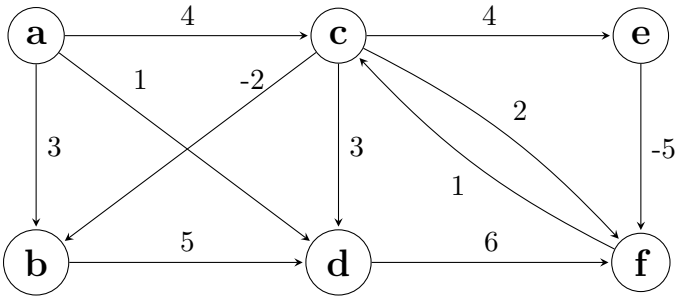
Betragt et korteste veje problem (eng. *single-source shortest-paths problem*) på en orienteret graf $G = (V, E)$ med vægtfunktion $w : E \to \mathbb{R}$ og startknude s. Hvilke af følgende egenskaber har et korteste-veje træ (eng. *shortest-paths tree*) $G' = (V', E')$ altid? Vælg ét eller flere korrekte svar.

1. Enhver sti i G' er en korteste vej i G. ✓
2. Alle korteste veje i G svarer til en sti i G'.
3. G' er et binært træ med s som rod.
4. G' er et mindste udspændende træ (eng. *minimum spanning tree*).
5. Kan beregnes med Dijkstra's algoritme.

Svar: Der kan være flere korteste veje mellem s og en knude v (med samme vægt), og korteste-vej træet indeholder netop én af disse. Korteste-vej træet er ikke nødvendigvis binært, fx kan vi have en graf hvor alle korteste veje fra s består af én kant, dvs. at s har kanter til $|V'|-1$ knuder. Et mindste udspændende træ (MST) minimerer summen af kantvægte og indeholder ikke nødvendigvis de korteste veje fra en bestemt knude s; betragt fx en uorienteret graf med tre knuder og kanter af vægt 2, 2 og 3 hvor et MST vil indeholde de to kanter af vægt 2, men hvor kanten af længde 3 udgør den korteste vej mellem to af knuderne. Dijkstra's algoritme er kun garanteret korrekt når der ikke er negative vægte, så hvis dette ikke er tilfældet kan output være forkert. (Man kan i stedet bruge en anden algoritme, fx Bellman-Ford.)

3 Korteste veje (4%)

Betragt korteste veje problemet (eng. *single-source shortest-paths problem*) med startknude **a** på følgende graf:



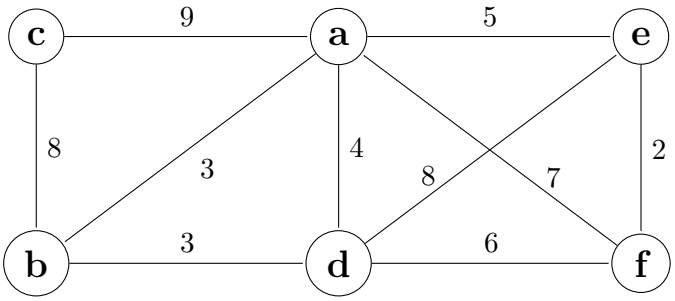
Hvilke kanter er med i korteste-vej træet? Vælg ét eller flere korrekte svar.

1. (a,b)
2. (a,c) ✓
3. (a,d) ✓
4. (b,d)
5. (c,b) ✓
6. (c,d)
7. (c,e) ✓
8. (c,f)
9. (d,f)
10. (e,f) ✓
11. (f,c)

Svar: Da der er negative vægte kan kanterne i træet findes ved at køre Bellman-Ford algoritmen, der giver det angivne output (som i dette tilfælde er unikt).

4 Prim's algoritme (4%)

Betragt følgende vægtede, uorienterede graf:



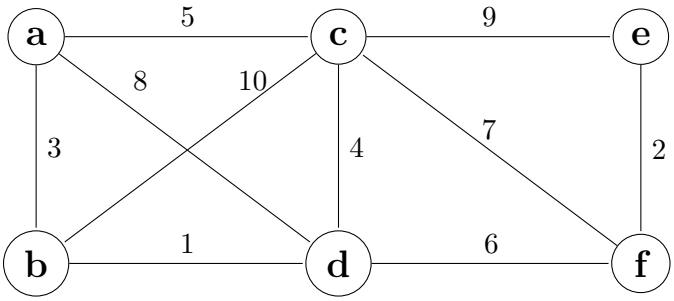
Hvis Prim's algoritme køres på denne graf med start i knude **a**, i hvilken rækkefølge bliver knuderne inkluderet i det mindste udspændende træ? (Med andre ord, i hvilken rækkefølge bliver de taget ud af prioritetskøen?) Vælg præcis ét svar.

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

Svar: Håndkørsel af Prim's algoritme og giver den angivne rækkefølge. Bemærk at rækkefølgen *ikke* er den samme som for Dijkstra's algoritme til at finde korteste veje.

5 Mindste udspændende træ (4%)

Betragt mindste udspændende træ (eng. *minimum spanning tree*) problemet på følgende vægtede, uorienterede graf:



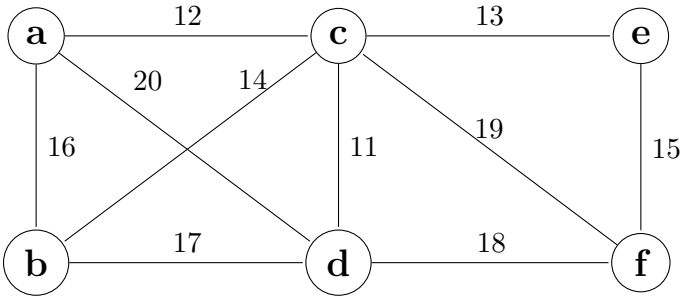
Hvilke kanter er med i et mindste udspændende træ? Vælg ét eller flere korrekte svar.

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

Svar: Mindste udspændende træ kan findes ved at køre fx Kruskál's eller Prim's algoritme og giver det angivne input (som i dette tilfælde er unikt).

6 Snit og sikre kanter (4%)

Betragt følgende uorienterede, vægtede graf med knudemængde $V = \{a, b, c, d, e, f\}$:



Betragt kantmængden $A = \{\{a, c\}, \{c, d\}\}$, som er en delmængde af et minimalt udspændende træ (eng. *minimum spanning tree*). Vi er interesserede i snit (eng. *cuts*), sikre kanter (eng. *safe edges*) for A, og snit der respekterer (eng. *respects*) en kantmængde A. Hvilke af følgende udsagn er sande? Vælg ét eller flere korrekte svar.

1. Lad $S_1 = \{a, b, c, d\}$. Snittet $(S_1, V - S_1)$ respekterer kantmængden A. ✓
2. Lad $S_2 = \{a, c, e\}$. Snittet $(S_2, V - S_2)$ respekterer kantmængden A.
3. Lad $S_3 = \{b, e, f\}$. Snittet $(S_3, V - S_3)$ respekterer kantmængden A. ✓
4. $\{d, f\}$ er en sikker kant for A.

5. $\{c, e\}$ er en sikker kant for A. ✓

6. $\{a, d\}$ er en sikker kant for A.

Svar: Kanten $\{c, d\}$ krydser snittet $(S_2, V - S_2)$, så det respekterer ikke A. Snittene med S_1 og S_3 har ikke nogen kant i A, der krydser, så de respekterer A. Kanten $\{d, f\}$ har ikke mindst vægt i snittet $(\{a, c, d\}, \{b, e, f\})$, så den er ikke sikker. Kanten $\{d, f\}$ har mindst vægt i snittet $(\{a, c, d\}, \{b, e, f\})$, så den er sikker. Kanten $\{a, d\}$ krydser ikke snittet $(\{a, c, d\}, \{b, e, f\})$, så den er ikke sikker.

7 Amortiseret analyse (4%)

I denne opgave betragter vi potentialfunktionen der bruges til at analysere en binær tæller i CLRS sektion 17.3. Lad $\Phi(D_i)$ betegne potentialfunktionens værdi efter i INCREMENT operationer. Hvilke egenskaber har $\Phi(D_i)$? (Læg mærke til at udsagnene ikke er uafhængige — hvis der gælder $f(i) = \Omega(i)$ så gælder også $f(i) = \Omega(\lg i)$, osv.) Vælg ét eller flere korrekte svar.

1. $\Phi(D_i) = \Omega(\lg i)$.

2. $\Phi(D_i) = \Omega(i)$.

3. $\Phi(D_i) = O(1)$.

4. $\Phi(D_i) = O(\lg i)$. ✓

5. $\Phi(D_i) = O(i)$. ✓

Svar: $\Phi(D_i)$ er lig med antal 1er i den binære repræsentation af tallet i (som er mindst 1 for $i > 0$). Hvis i er en potens af 2 er $\Phi(D_i) = 1$, så vi kan ikke begrænse funktionen nedefra med en voksende funktion af i. Hvis $i + 1$ en potens af 2 er $\Phi(D_i) = \lg(i + 1)$, så både $O(\lg i)$ og $O(i)$ er gyldige øvre grænser.

8 Køretid (4%)

Betragt følgende kode i pseudo-kode notationen fra CLRS:

```
LOOPS(n)
1  r = 0
2  for i = 1 to n/2
3        for j = n/2 − i to n
4              r = r + j
5  return r
```

Vi antager at $n/2$ er et heltal således at den yderste **for**-løkke har $n/2$ iterationer. Hvilke af følgende er gyldige udsagn om køretiden $T(n)$ af LOOPS(n)? (Læg mærke til at udsagnene ikke er uafhængige — hvis der gælder $f(n) = O(n)$ så gælder også $f(n) = O(n \lg n)$, osv.) Vælg ét eller flere korrekte svar.

1. $T(n) = O(n)$.
2. $T(n) = O(n \lg n)$.
3. $T(n) = O(n^2)$. ✓
4. $T(n) = \Omega(n)$. ✓
5. $T(n) = \Omega(n \lg n)$. ✓
6. $T(n) = \Omega(n^2)$. ✓

Svar: Der er $n/2$ iterationer af den ydre løkke, der hver tager $O(n)$ tid fordi den indre løkke laver mellem $n/2$ og n iterationer. Dvs. køretiden er $\Theta(n^2)$. Derfor er de to første store-O grænser for lave, mens de øvrige grænser gælder.

9 Rekursionsligninger (4%)

Hvilke af disse rekursionsligninger har løsningen $T(n) = \Theta(n \lg n)$? Antag at $T(n) = 1$ for $n \leq 1$. Vælg ét eller flere korrekte svar.

1. $T(n) = 2T(\lfloor n/2 \rfloor) + n^2$.
2. $T(n) = 2T(\lfloor n/2 \rfloor) + 3n$. ✓
3. $T(n) = 2T(\lfloor n/3 \rfloor) + 2n$.
4. $T(n) = 3T(\lfloor n/2 \rfloor) + n \lg n$.
5. $T(n) = 3T(\lfloor n/3 \rfloor) + n$. ✓
6. $T(n) = T(n-1) + \lg n$. ✓

Svar: De fem første rekursionsligninger kan løses ved hjælp af master theorem, hvor nummer 2 og 5 falder i tilfælde 2 og giver $T(n) = \Theta(n \lg n)$. Den sidste ligning kan løses med substitutionsmetoden, hvor der er n led af størrelse højst $\lg n$, og hvor halvdelen har størrelse mindst $\lg(n/2) = \lg(n) - 1$.

10 Del og hersk (4%)

Antag at du har en rekursiv algoritme A, og lad n betegne størrelsen på algoritmens input X. Hvis $n = 1$ beregnes resultatet $A(X)$ i konstant tid. For $n > 1$ bruger algoritmen først $O(n \lg n)$ tid på at beregne fire inputs X_1, X_2, X_3, X_4, hver af størrelse $\lfloor n/2 \rfloor$, og $A(X_1), A(X_2), A(X_3), A(X_4)$ beregnes rekursivt. Endelig kombineres de rekursive svar til et output $A(X)$ i tid $O(n^2)$.

Hvilken af disse rekursionsligninger beskriver køretiden $T(n)$ af A på input X? Vælg præcis ét svar.

1. $T(n) = 4T(\lfloor n/2 \rfloor) + O(n \lg n)$.
2. $T(n) = 2T(\lfloor n/4 \rfloor) + O(n \lg n)$.
3. $T(n) = 4T(\lfloor n/2 \rfloor) + O(n^2)$. ✓
4. $T(n) = 2T(\lfloor n/4 \rfloor) + O(n^2)$.

Svar: De fire rekursive delproblemer af størrelse højst $\lceil n/2 \rceil$ tager total tid $4T(\lfloor n/2 \rfloor)$. Derudover bruges der totalt tid $O(n^2)$ på at beregne $A(X)$.

11 Fibonacci hobe (4%)

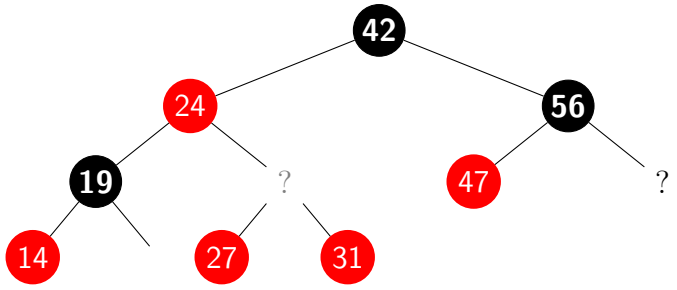
Betragt en Fibonacci hob H (eng. *Fibonacci heap*), som beskrevet i CLRS kapitel 19, hvorpå der udføres n_1 INSERT operationer, n_2 EXTRACT-MIN operationer, og n_3 DECREASE-KEY operationer. Det totale antal operationer er $n = n_1 + n_2 + n_3$. Hvilke af følgende udsagn er altid sande? Vælg ét eller flere korrekte svar.

1. Den samlede worst case tid for operationerne er $\Omega(n \lg n)$.
2. Den samlede worst case tid for operationerne er $O(n \lg n)$. ✓
3. Den samlede worst case tid for operationerne er $O(n_1 + n_3 + n_2 \lg n)$. ✓
4. H har $n_1 - n_2 - n_3$ knuder.
5. H har $m(H) = n_1 - n_2$ markerede knuder.

Svar: Den amortiserede analyse giver en øvre grænse på $O(n_1 + n_3 + n_2 \lg n)$ for operationerne. Da $n = n_1 + n_2 + n_3$ er det specielt $O(n \lg n)$. Hvis $n_1 + n_3 > n_2 \lg n$ bliver tiden $O(n)$, så tiden er ikke nødvendigvis $\Omega(n \lg n)$. Antal knuder i H er antal indsættelser minus antal sletninger, dvs. $n_1 - n_2$ som er forskellig fra $n_1 - n_2 - n_3$ hvis $n_3 > 0$. Antal markerede knuder kan ikke udtrykkes ved hjælp af n_1 og n_2 — for eksempel ændrer INSERT ikke antallet af markerede knuder.

12 Binære 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 to indre knuder er uspecificerede:



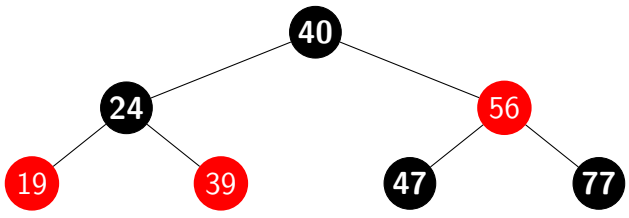
Hvilke knuder er mulige på mindst én af positionerne markeret med ?, hvis træet skal overholde invarianterne for et rød-sort binært søgetræ? Vælg ét eller flere korrekte svar.

1. En rød knude med nøglen 28.
2. En rød knude med nøglen 44.
3. En rød knude med nøglen 58. ✓
4. En sort knude med nøglen 25.
5. En sort knude med nøglen 30. ✓
6. En sort knude med nøglen 66.

Svar: Knuden til venstre skal være sort (fordi røde knuder har sorte børn) og ligge mellem 27 og 31. Derfor er kun en sort knude med nøglen 30 en mulighed hér. Knuden til højre skal være rød (fordi alle stier fra roden skal have samme antal sorte knuder) og større end 56, så kun en rød knude med nøglen 58 er mulig hér.

13 Binære søgetræer (4%)

Betragt dette rød-sorte binære søgetræ (igen med bladene udeladt):



Antag at vi bruger indsættelsesalgoritmen RB-INSERT fra CLRS kapitel 13 til at indsætte nøglen 42. Hvad sker med træet? Vælg præcis ét svar.

1. 42 indsættes som en sort knude under knuden 47.
2. 42 indsættes som en rød knude under knuden 47. ✓
3. 42 indsættes som ny sort rodknude og 40 bliver barn til knuden 39.
4. 42 indsættes som ny rød rodknude og 40 bliver barn til knuden 39.
5. Ingen af ovenstående.

Svar: RB-INSERT finder det rette sted i træet til nøglen 42, som det venstre barn under knude 47. Knuden farves rød og derefter kaldes RB-INSERT-FIXUP, men den terminerer uden at ændre noget fordi forældreknuden 47 er sort.

14 Disjunkte mængder (4%)

Vi betragter trærepræsentationer af disjunkte mængder (eng. *disjoint sets*), som beskrevet i CLRS sektion 21.3, ved brug af heuristikkerne *union by rank* og *path compression*. Antag at MAKE-SET er kørt n gange og lad k være en variabel. Hvilke af følgende udsagn er altid sande? Vælg ét eller flere korrekte svar.

1. Tiden for et kald til UNION(x, y) er $O(\lg n)$. ✓
2. k kald til FIND-SET(42) lige efter hinanden tager tid $O(k + \lg n)$. ✓
3. Kaldene MAKE-SET(x_1),...,MAKE-SET(x_k) tager total tid $O(k)$. ✓
4. En LINK(x, y) operation øger den maksimale værdi af *rank* i træerne for x og y med 1.
5. I værste fald tager en FIND-SET operation $\Omega(n)$ tid.
6. UNION(x, x) tager konstant tid.

Svar: Dybden af trærepræsentationen er $O(\lg n)$, så det gælder også tiden for FIND-SET og dermed UNION. Kalder vi FIND-SET(42) igen efter at den lige er blevet kaldt vil operationen tage konstant tid, dvs. tiden for k kald bliver $O(k + \lg n)$. En LINK operation giver kun en større *rank* hvis røddernes rank er identisk. Med den givne implementation af UNION(x, x) laves der to kald til FIND-SET, der ikke nødvendigvis tager konstant tid.

15 Invarianter (4%)

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

```
SECOND-SMALLEST(A, n)
1  m = A[1]
2  r = ∞
3  for i = 2 to n
4       if A[i] < m
5            r = m
6            m = A[i]
7       elseif A[i] < r
8            r = A[i]
9  return r
```

Hvilke udsagn gyldige løkke-invarianter ved starten af hver iteration i **for**-løkken? Vælg ét eller flere korrekte svar.

1. $m = \min(A[1], \ldots, A[i-1])$. ✓
2. $m = \min(A[1], \ldots, A[n])$.
3. $r \geq m$. ✓
4. $m \geq 0$.
5. Mindst 1 element i $A[1], \ldots, A[i-1]$ er mindre end eller lig med r. ✓
6. Højst 2 elementer i $A[1], \ldots, A[i-1]$ er mindre end eller lig med r.

Svar: Udsagn 1, 3 og 5 gælder for $i = 2$ pga. initialiseringen af m og r, og kan ses at gælde efter hver iteration ved induktion. Udsagn 2 gælder ikke hvis minimum i A ikke er lig med $A[1]$. Udsagn 4 gælder ikke hvis $A[1] < 0$. Udsagn 6 gælder ikke hvis der er 3 værdier i A, der har den samme, mindste værdi.

16 Valg af datastruktur (4%)

Antag at vi gerne vil skabe en datastruktur VENTETID, der indeholder en mængde af n afgangstidspunkter (ugedag og klokkeslet, fx for en bus). For enkelheds skyld skal der blot holdes styr på én rute og ét stoppested, så den eneste information er afgangstidspunkter. Datastrukturen skal understøtte to operationer: INSERT(x), der indsætter et nyt afgangstidspunkt, og HVORLÆNGE(x) der returnerer antal minutter fra tidspunkt x til den næste afgang.

Hvad er det mest passende valg af datastruktur til løsning af dette problem blandt nedenstående, og hvilken worst-case (amortiseret) køretid for HvorLænge(x) bliver resultatet af dette valg? Udtryk svaret som funktion af n. Vælg præcis ét svar.

1. Disjunkte mængder (eng. *disjoint sets*), tid $O(1)$.
2. Disjunkte mængder (eng. *disjoint sets*), tid $O(\lg n)$.
3. Fibonacci hob (eng. *Fibonacci heap*), tid $O(\lg n)$.
4. Fibonacci hob (eng. *Fibonacci heap*), tid $O(1)$.
5. Rød-sort binært søgetræ (eng. *red-black binary search tree*), tid $O(\lg n)$. ✓
6. Rød-sort binært søgetræ (eng. *red-black binary search tree*), tid $O(1)$.

Svar: Hvis vi indsætter afgangstiderne i et rød-sort søgetræ så kan HvorLænge(x) implementeres som et kald til Successor(x). Hvis der ikke findes nogen efterfølger til x (fordi der ikke er nogen senere afgang i samme uge), returneres minimum, dvs. den første afgang i næste uge. I begge tilfælde tager det tid $O(\log n)$.

17 Korrekthedsargumenter (4%)

Betragt følgende problem, kaldet `MaxProduct`:

Givet en liste af tal $x_1, \ldots, x_n \in \mathbf{R}$ (kan være både positive og negative), find en delmængde $I^* \subseteq \{1, \ldots, n\}$ hvor produktet $\prod_{i \in I^*} x_i$ er så stort som muligt, dvs. for alle mængder $I \subseteq \{1, \ldots, n\}$ skal gælde at $\prod_{i \in I} x_i \leq \prod_{i \in I^*} x_i$.

Eksempel. På input $x_1 = -4$, $x_2 = -0.5$, $x_3 = 0.5$, $x_4 = 0.5$, $x_5 = 1.5$, $x_6 = 6$ kan vi vælge $I = \{5, 6\}$ og få et produkt på $1.5 \cdot 6 = 9$, men et bedre valg er $I = \{1, 2, 5, 6\}$ som giver produktet $(-4) \cdot (-0.5) \cdot 1.5 \cdot 6 = 18$. Den tomme mængde $I = \emptyset$ giver per definition $\prod_{i \in \emptyset} x_i = 1$.

Hvad gælder altid for en optimal løsning I^*? Vælg ét eller flere korrekte svar.

1. I^* indeholder *ikke* noget indeks i hvor x_i er mellem 0 og 1 ($x_i \in (0, 1)$). ✓
2. I^* indeholder *alle* indekser i hvor $x_i < 0$.
3. I^* indeholder *alle* indekser i hvor $x_i > 1$. ✓
4. I^* indeholder *alle* indekser i hvor $|x_i| > 1$ (i.e., absolutværdien af x_i er større end 1).

Svar: Den optimale løsning har altid produkt mindst 1 fordi vi kan vælge $I = \emptyset$. Hvis I^* indeholdt i hvor $x_i \in (0, 1)$ så kunne man opnå et større produkt ved at vælge $I = I^* \backslash \{i\}$, hvilket er en modstrid. Hvis antallet af negative indeks er ulige kan vi ikke have dem alle med i en optimal løsning, da produktet så ville blive negativt. Hvis vi udelader et indeks med $x_i > 1$, så $i \notin I^*$ kan vi opnå et større produkt ved at vælge $I = I^* \cup \{i\}$, hvilket er en modstrid. Hvis antallet af indeks i med $x_i < -1$ er ulige kan vi ikke have dem alle med i en optimal løsning, da produktet så ville blive negativt.

18 Dynamisk programmering (4%)

En *stigende delsekvens* af en tabel A, der indeholder heltal $A[1], \ldots, A[n]$, er en delmængde af indekser $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ således at $A[i_1] < A[i_2] < \cdots < A[i_k]$. Længden af delsekvensen er antal indekser, k. En *ikke-aftagende delsekvens* defineres på samme måde bortset fra at kravet er $A[i_1] \leq A[i_2] \leq \cdots \leq A[i_k]$. Betragt følgende funktion, i pseudo-kode notationen fra CLRS, hvor input A er en tabel af heltal.

```
MaximumMystery(A, n)
1  let r[1...n] be a new array
2  r[1] = 1
3  for j = 2 to n
4       q = 1
5       for i = 1 to j − 1
6            if not A[j] < A[i]
7                 q = max(q, r[i] + 1)
8       r[j] = q
9  return max_{1≤ℓ≤n} r[ℓ]
```

Hvad beregner proceduren MaximumMystery? Vælg præcis ét svar.

1. Indekserne i en længste stigende delsekvens.
2. Indekserne i en længste ikke-aftagende delsekvens.
3. Længden af en længste stigende delsekvens.
4. Længden af en længste ikke-aftagende delsekvens. ✓

Svar: En invariant for den ydre løkke er at efter hver iteration gælder for $\ell = 1 \ldots, j$ at $r[\ell]$ er længden af den længste ikke-aftagende delsekvens, der indeholder indekset ℓ. Den indre løkke beregner $r[j]$ ved rekursionsformlen $r[j] = 1 + \max_{i<j,\, A[i] \leq A[j]} r[i]$, hvor maximum af den tomme mængde defineres som 0. Hvis $A[j]$ er mindre end alle tidligere værdier i A beholdes initialiseringsværdien $q = 1$.

Skriftlige opgaver

19 Længste fælles delsekvens (4%)

Vi betragter algoritmen til længste fælles delsekvens (eng. *longest common subsequence*) i CLRS sektion 15.4.

a) Tegn tabellen c som algoritmen udfylder på følgende input:

$$x = \langle N, A, A, N \rangle \text{ og } y = \langle A, N, N, A \rangle.$$

Tilføj pile ($\uparrow$, $\nwarrow$, $\leftarrow$), der forklarer tallene i tabellen, som i CLRS Figur 15.8 og angiv den længste fælles delsekvens. Skriv dit svar på side 2 i Word dokumentet.

Svar: For overskuelighedens skyld inkluderer vi de to strenge i kanten af tabellen c, ligesom i CLRS Figur 15.8.

		A	N	N	A
	0	0	0	0	0
N	0	$\uparrow 0$	$\nwarrow 1$	$\nwarrow 1$	$\leftarrow 1$
A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$
A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$
N	0	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 2$	$\uparrow 2$

(I nogle tilfælde er der et valg mellem $\uparrow$ og $\leftarrow$, det lægger vi ikke vægt på når der gives point.) Længste fælles delsekvens er NA.

20 Dynamisk programmering (10%)

Vi betragter følgende variant af længste fælles delsekvens (eng. *longest common subsequence*) problemet fra CLRS sektion 15.4. Input er to vektorer $x = \langle x_1, \ldots, x_m \rangle \in \Sigma^m$ og $y = \langle y_1, \ldots, y_n \rangle \in \Sigma^n$. I lighed med længste fælles delsekvens er målet at finde en optimal vektor $z = \langle z_1, \ldots, z_k \rangle$, men i stedet for at maksimere længden af z skal *vægten* af tegnene i z maksimeres. Vi antager at vægtene er ikke-negative. Vægten er givet ved en datastruktur w, således at vægten af $z_i \in \Sigma$, der betegnes med $w[z_i]$, kan findes i konstant tid. Vægten af z er summen af vægte, $\sum_{i=1}^{k} w[z_i]$, og målet er et finde en fælles delsekvens med størst mulig vægt.

Eksempel. Antag at vi har $x = \langle A, H, A, A \rangle$, $y = \langle A, A, R, G, H \rangle$, og $w[A] = 2$, $w[G] = 8$, $w[R] = 10$, $w[H] = 5$. Delsekvensen $\langle A, A \rangle$ med vægt $2 + 2 = 4$ findes i både x og y. Delsekvensen $\langle A, H \rangle$ med vægt $2 + 4 = 7$ findes også i både x og y, og er den delsekvens, der har størst vægt.

a) Skriv en rekursionsligning for den optimale vægt af en fælles delsekvens af x og y. Skriv dit svar på side 3 i Word dokumentet.

Svar: I forhold til det almindelige længste fælles delsekvens problem skal vi ændre omkostningen i det tilfælde hvor to karakterer matcher så det afspejler vægten:

$$c[i,j] = \begin{cases} 0 & \text{hvis } i = 0 \text{ eller } j = 0 \\ c[i-1, j-1] + w[\alpha] & \text{hvis } i,j > 0 \text{ og } x_i = y_j = \alpha \\ \max(c[i-1,j], c[i,j-1]) & \text{hvis } i,j > 0 \text{ og } x_i \neq y_j \end{cases}$$

b) Lav en analyse af køretiden af den algoritme, der bruger rekursionsligningen og dynamisk programmering til at beregne den optimale vægt. Analysen skal give en øvre grænse på køretiden som funktion af n og m i store-O notation, der er så præcis som mulig. Skriv dit svar på side 4 i Word dokumentet.

Svar: Analysen for længste fælles delsekvens kan genbruges, da rekursionsligningen har samme struktur. Ændringen påvirker kun værdierne og tiden for hvert skridt øges med højst en konstant faktor (da en vægt kan findes i konstant tid). Derfor er køretiden $O(nm)$.

c) Beskriv hvordan algoritmen kan modificeres til at håndtere det tilfælde hvor vægte kan være negative uden at påvirke den asymptotiske køretid. Argumentér for køretid og korrekthed af din løsning. Skriv dit svar på side 5 i Word dokumentet.

Svar: Tegn med negative vægte vil aldrig være del af en optimal løsning, hvilket kan ses med et modstridsargument: Hvis z indeholder et tegn med negativ vægt, så har delsekvensen der udelader dette element højere vægt. Tegn med negativ vægt kan derfor fjernes fra input i tid $O(n+m)$, hvorefter det resterende problem er uden negative vægte. Den samlede tid er $O(n+m+nm)$, hvilket er $O(nm)$ som inden.

21 Induktionsbeviser (6%)

I denne opgave antager vi at T er en sorteret tabel med $n > 1$ elementer således at for $i < j$ gælder $T[i] \leq T[j]$. Antag at $1 \leq p \leq r \leq n$. Betragt følgende funktion, i pseudo-kode notationen fra CLRS:

WhileMystery(x, T, p, r)

```
1  low = p
2  high = max(p, r + 1)
3  while low < high
4        mid = ⌊(low + high)/2⌋
5        if x ≤ T[mid]
6              high = mid
7        else
8              low = mid + 1
9  return high
```

a) Lad $high_i$ og low_i betegne værdien af variablene *high* og *low* efter i iterationer af **while** løkken. Bevis ved induktion at $high_i - low_i \leq n/2^i$. (Du skal bruge antagelsen $1 \leq p \leq r \leq n$.) Der lægges vægt på, at argumentationen er klar og koncis. Skriv dit svar på side 6 i Word dokumentet.

Svar: Inden den første iteration har vi $high_0 - low_0 = \max(p, r+1) - p \leq n = n/2^0$. Det viser basistilfældet $i = 0$. I induktionsskridtet antager vi, at efter $i-1$ iterationer gælder $high_{i-1} - low_{i-1} \leq n/2^{i-1}$. Efter iteration i er vi i ét at to tilfælde afhængigt af udfaldes af sammenligningen i linje 5: $high_i = \lfloor (low_{i-1} + high_{i-1})/2 \rfloor$ og $low_i = low_{i-1}$ eller $low_i = \lfloor (low_{i-1} + high_{i-1})/2 \rfloor + 1$ og $high_i = high_{i-1}$. I det første tilfælde har vi, via induktionshypotesen:

$$high_i - low_i = \lfloor (low_{i-1} + high_{i-1})/2 \rfloor - low_{i-1} \leq (high_{i-1} - low_{i-1})/2 \leq (n/2^{i-1})/2 = n/2^i \ .$$

I det andet tilfælde har vi:

$$high_i - low_i = high_{i-1} - (\lfloor (low_{i-1} + high_{i-1})/2 \rfloor + 1) \leq (high_{i-1} - low_{i-1})/2 \leq (n/2^{i-1})/2 = n/2^i \ .$$

22 Grådige algoritmer (8%)

Vi betragter igen problemet fra opgave 17, gengivet hér:

Givet en liste af tal $x_1, \ldots, x_n \in \mathbf{R}$ (kan være både positive og negative), find en delmængde $I^* \subseteq \{1, \ldots, n\}$ hvor produktet $\prod_{i \in I^*} x_i$ er så stort som muligt, dvs. for alle mængder $I \subseteq \{1, \ldots, n\}$ skal gælde at $\prod_{i \in I} x_i \leq \prod_{i \in I^*} x_i$.

Eksempel. På input $x_1 = -4$, $x_2 = -0.5$, $x_3 = 0.5$, $x_4 = 0.5$, $x_5 = 1.5$, $x_6 = 6$ kan vi vælge $I = \{5, 6\}$ og få et produkt på $1.5 \cdot 6 = 9$, men et bedre valg er $I = \{1, 2, 5, 6\}$ som giver produktet $(-4) \cdot (-0.5) \cdot 1.5 \cdot 6 = 18$. Den tomme mængde $I = \emptyset$ giver per definition $\prod_{i \in \emptyset} x_i = 1$.

Et mulig tilgang til at løse `MaxProdukt` er at grådigt føje elementer til I, ét ad gangen. Antag at input er sorteret således at $x_1 \leq x_2 \leq \cdots \leq x_n$.

a) Argumentér for at for en optimal løsning I^* vil der altid vil være et lige antal elementer i mængden $I^*_- = \{i \in I^* \mid x_i < 0\}$, dvs. $|I^*_-|$ er deleligt med 2. Skriv dit svar på side 7 i Word dokumentet.

Svar: Vi ved at $\prod_{i \in I^*} x_i \geq 1$ da den tomme mængde er et muligt valg. For at få et produkt, der er positivt, er antal negative elementer i produktet nødt til at være lige.

b) Foreslå en effektiv algoritme til at finde en optimal løsning I^*. Argumentér for korrekthed og køretid. Der lægges vægt på, at argumentationen er klar og koncis. (Det er *ikke* et krav at argumentere via "greedy choice property" og "optimal substructure".) Skriv dit svar på side 8 i Word dokumentet.

Svar: Problemet kan deles i to uafhængige problemer, der kan løses separat. Antag at k er indekset på det største negative input, dvs. $x_1, \ldots, x_k < 0$ og $x_{k+1}, \ldots, x_n \geq 0$. Vi ønsker at finde $I^*_- \subseteq \{1, \ldots, k\}$ og $I^*_+ \subseteq \{1, \ldots, k\}$ således at $I^* = I^*_- \cup I^*_+$ er optimal. Vi ved at alle produkter der kan opnås med indekser i I^*_+ er positive, så den optimale løsning kan ikke indeholde et negativt produkt over indekser i I^*_-. Derfor er det optimalt at maksimere de to delproblemer separat. Et optimalt valg for de positive tal er $I^*_+ = \{i \mid x_i > 1\}$ hvilket kan ses ved et modstridsargument. Definér $q = |\{i \mid x_i < -1\}|$. For de negative tal er $I = \{1, \ldots, q\}$ ikke nødvendigt optimalt, da det kan indeholde et ulige antal elementer q (i modstrid med spørgsmål b). Hvis q er ulige er der tre kandidater til en optimal løsning I^*_-: $\emptyset$, $\{1, \ldots, q-1\}$ og $\{1, \ldots, q+1\}$. Argumentet for at det ikke er muligt at finde en bedre løsning er at betragte $I_- \subseteq \{1, \ldots, k\}$ og vise at mindst én af disse tre mulige løsninger der lige så god eller bedre. Hvis input er sorteret tager det lineær tid at finde I^*_- og I^*_+.