TDT4120 Algoritmer og datastrukturer

Eksamen, 28. november 2019, 09:00–13:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	D

Løsningsforslag

Løsningsforslagene i rødt nedenfor er *eksempler* på svar som vil gi full uttelling. Det vil ofte være helt akseptabelt med mange andre, beslektede svar, spesielt der det bes om en forklaring eller lignende. Om du svarte noe litt annet, betyr ikke det nødvendigvis at du svarte feil!

5 % **1** Vurder følgende utsagn om Dijkstra:

> Kantvektene kan være negative, i motsetning til i Bellman-Ford.

Stemmer dette? Svar ja eller nei og forklar kort.

Nei. Det er Bellman-Ford som tillater negative kantvekter.

Om man her forklarer *hvorfor* Dijkstra ikke takler negative kantvekter, så er det bare positivt, men det er ikke nødvendig.

Relevante læringsmål: Forstå Dijkstra; forstå Bellman-Ford

5 % **2** Vurder følgende utsagn om Insertion-Sort:

> Den har kjøretid $\Omega(n \lg n)$ i beste tilfelle (*best-case*).

Stemmer dette? Svar ja eller nei og forklar kort.

Nei. Insertion-Sort har kjøretid $\Theta(n)$ i beste tilfelle.

Da vil ingen av elementene flyttes nedover, og vi ender med å bare gå gjennom tabellen fra start til slutt én gang.

Her er det ikke nødvendigvis meningen at man skal ha *pugget* denne kjøretiden, men at man kan tenke seg frem til den.

Relevante læringsmål: Forstå Insertion-Sort; kunne definere *best-case*, *average-case* og *worst-case*; kunne definere *asymptotisk notasjon*, O, Ω, Θ, o og ω.

5 % **3** Vurder følgende utsagn om splitt-og-hersk (*divide-and-conquer*):

> Metoden bør unngås når vi har overlappende delproblemer.

Stemmer dette? Svar ja eller nei og forklar kort.

Ja. Ellers vil man kunne få eksponentiell kjøretid.

Dette er akkurat poenget med å bruke memoisering (dynamisk programmering).

Relevante læringsmål: Forstå designmetoden *divide-and-conquer* (*splitt og hersk*); forstå hva *overlappende delproblemer* er; forstå løsning ved *memoisering* (*top-down*)

5 % **4** Vurder følgende utsagn om Topological-Sort:

> Nodene sorteres etter synkende starttid (*discover-time*).

Stemmer dette? Svar ja eller nei og forklar kort.

Nei. De sorteres etter synkende sluttid (*finish-time*).

Om man her sier at de sorteres etter *økende* sluttid, gir det også full uttelling. Det er ikke slik det gjøres i pensumalgoritmen, men den eneste forskjellen er at kantene peker i motsatt retning. Om man sorterer etter (økende eller synkende) starttid, derimot, blir ordningen vanligvis gal.

Relevant læringsmål: Forstå Topological-Sort

5 % **5** Vurder følgende utsagn om hauger (*heaps*):

> Build-Max-Heap har kjøretid $\Theta(n \lg n)$.

Stemmer dette? Svar ja eller nei og forklar kort.

Nei. Build-Max-Heap har kjøretid $\Theta(n)$.

Relevante læringsmål: Forstå hvordan *heaps* fungerer, og hvordan de kan brukes som *prioritetskøer* (inkl. operasjonen Build-Max-Heap)

5 % **6** Hvilke algoritmer i pensum finner korteste veier fra én til alle (*single-source shortest paths*) i vektede, rettede grafer?

Ikke inkluder algoritmer som finner korteste veier fra alle til alle.

Bellman-Ford, Dijkstra og DAG-Shortest-Paths.

Her får man ikke trekk for å nevne BFS, selv om den ikke fungerer på vektede grafer generelt. Teknisk sett vil algoritmer som finner korteste veier fra alle til alle også finne korteste veier fra én til alle som en bivirkning; det er derfor de eksplisitt er ekskluderte.

Her er det bare positivt, men ikke nødvendig, om man diskuterer når de ulike algoritmene fungerer. Om man også ekskluderer algoritmer på grunn av spesialkrav som ikke er nevnt i oppgaven, så trekkes det ikke for det. Om man f.eks. sier at Dijkstra ikke løser problemet generelt, fordi den ikke takler negative kanter, så er det helt korrekt; tilsvarende for sykler og negative sykler for DAG-Shortest-Path og Bellman-Ford. Om man påpeker at det generelle problemet er NP-hardt, og at ingen pensumalgoritmer løser det, så er det også korrekt—men da forventes det at man uansett diskuterer noen av de nevnte pensumalgoritmene, som løser de noe begrensede tilfellene.

Relevante læringsmål: Forstå Bellman-Ford; forstå Dijkstra; forstå DAG-Shortest-Paths

5 % **7** Hvilke algoritmer i pensum finner minimale spenntrær (*minimum spanning trees*)? (Vi antar her vektede, urettede, sammenhengende grafer.)

Prim og Kruskal.

Her vil man ikke få full uttelling om man blander inn f.eks. DFS eller BFS uten begrunnelse, selv om de finner (teknisk sett minimale) spenntrær i det *uvektede* tilfellet.

Om man nevner Borůvka, gir det også full uttelling, selv om Borůvka ikke er pensum.

Relevante læringsmål: Forstå Prim; forstå Kruskal

5 % **8** Følgende delvis sensurerte lemma er tatt fra et delkapittel i læreboka som viser at en pensumalgoritme er korrekt.

> **Lemma 16.2**
>
> Let C be an alphabet in which each character $c \in C$ has frequency $c.freq$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

Hvilken algoritme er det snakk om, og hvilket problem løser den?

HUFFMAN. Den finner optimale prefikskoder (*prefix codes*).

Dette er den eneste algoritmen i pensum som har med frekvenser å gjøre, og kan f.eks. gjenkjennes fra ledetrådene på den måten.

Her får man også uttelling for andre måter å forklare problemet på, som f.eks. at den komprimerer en tekst, e.l.

De sensurerte delene av teksten vises i løsningsforslaget, men det kreves ikke at kandidaten oppgir dem.

Relevant læringsmål: Forstå HUFFMAN og *Huffman-koder*

5 % **9** Din venn Smartnes har gitt deg følgende tabell, som hun påstår er en maks-haug (*max-heap*):

$A = \langle 7, 3, 8, 6, 5, 9, 4, 2 \rangle$

Du er ikke enig i at dette er en haug, men sier deg likevel villig til å utføre HEAP-EXTRACT-MAX på tabellen, *selv om resultatet blir feil*. (Du skal altså bare utføre trinnene i HEAP-EXTRACT-MAX, uten å korrigere A på noe vis.)

Hvordan ser A ut etterpå?

(Oppgi kun de 7 første elementene av A etter operasjonen.)

$\langle 8, 3, 9, 6, 5, 2, 4 \rangle$

Her kan man muligens misforstå «uten å korrigere A», som gis som en tilleggsopplysning, som om man skal endre på HEAP-EXTRACT-MAX på noe vis, f.eks. ved å fjerne linje 6, MAX-HEAPIFY$(A, 1)$, men det er ikke det oppgaven ber om. Man skal utføre HEAP-EXTRACT-MAX, og det er det; dvs., man skal ikke først gjøre om A til en haug (med BUILD-MAX-HEAP).

Relevant læringsmål: Forstå hvordan *heaps* fungerer, og hvordan de kan brukes som *prioritetskøer* (inkl. operasjonen HEAP-EXTRACT-MAX)

5 % **10** Hva er $O(n) + \Omega(n) + \Theta(n) + o(n) + \omega(n)$?

$\omega(n)$

Her vil man få en del uttelling dersom man svarer $\Omega(n)$ og noe uttelling dersom man svarer $\Theta(n)$. Det siste kan virke nærliggende, sidne uttrykket involverer både O og Ω. Men selv om noen av leddene har en øvre grense, så har ikke summen det, siden noen av leddene ($\Omega(n)$ og $\omega(n)$) kan bli vilkårlig store. Poenget er altså at $\omega(n)$ dominerer.

Relevant læringsmål: Kunne definere *asymptotisk notasjon*, O, Ω, Θ, o og ω

5 % **11** Løs følgende rekurrens eksakt, der n er et positivt heltall:

$$\begin{aligned} \mathrm{T}(1) &= 1 \\ \mathrm{T}(n) &= \mathrm{T}(n-1) + 2^{n-1} \qquad \text{(hvis } n > 1\text{)} \end{aligned}$$

Oppgi svaret *uten* bruk av asymptotisk notasjon.

$\mathrm{T}(n) = 2^n - 1$

Poenget er at dette gir summen $1 + 2 + 4 + 8 + \cdots + 2^{n-1} = 2^n - 1$. Man vil kunne få noe uttelling også om man har gjort mindre regnefeil, her, og f.eks. endt opp med $\mathrm{T}(n) = 2^n$ eller $\mathrm{T}(n) = 2^n + 1$.

Relevant læringsmål: Kunne løse rekurrenser med *iterasjonsmetoden*

5 % **12** Din venn Klokland har laget en versjon av MERGE med kjøretid $\Theta(n^2)$. Om du bruker Kloklands versjon i stedet for den vanlige, hva blir kjøretiden til MERGE-SORT? Oppgi svaret i Θ-notasjon. (Evt. forklar kort.)

$\Theta(n^2)$

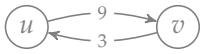
Her kreves ingen forklaring, men man kan f.eks. sette opp rekurrensen for den nye versjonen, som blir $\mathrm{T}(n) = 2\mathrm{T}(n/2) + \Theta(n^2)$, og denne kan man løse med masterteoremet, der $\log_b a = 1$ og $f(n) = \Theta(n^2) = \Omega(n^{1+\epsilon})$, så vi får $\mathrm{T}(n) = \Theta(n^2)$.

Relevante læringsmål: Forstå MERGE-SORT; kunne løse rekurrenser med *substitusjon*, *rekursjonstrær* og *masterteoremet*

5 % **13** I et flytnett, hvis du har restkapasiteter (*residual capacities*) $c_f(u, v) = 9$ og $c_f(v, u) = 3$, og $f(u, v) > 0$, hva er flyten $f(u, v)$ og kapasiteten $c(u, v)$? Oppgi verdiene som normalt, som to tall med skråstrek imellom, f.eks. 1/2 hvis du mener $f(u, v) = 1$ og $c(u, v) = 2$.

3/12

Her representerer c_f restnettkanter som følger:



Dette kan tilsvare én av de to følgende flytnettkantene:

(a) (b)

Siden vi vet at $f(u,v) > 0$, er det bare (a) som kan være riktig.

Relevante læringsmål: Kunne definere *flytnett*, *flyt* og *maks-flyt-problemet*; kunne definere *restnettet* til et flytnett med en gitt flyt

5 % **14** Vurder følgende utsagn om Floyd-Warshall:

> Hvis $d_{ij}^{(k)}$ settes til $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ så settes $\pi_{ij}^{(k)}$ til $\pi_{ik}^{(k-1)}$.

Stemmer dette? Svar ja eller nei og forklar kort.

Nei. I den nye stien $i \rightsquigarrow k \rightsquigarrow j$ kommer j sin forgjenger $\pi_{ij}^{(k)}$ fra den gamle stien $k \rightsquigarrow j$ og er altså $\pi_{kj}^{(k-1)}$.

Relevante læringsmål: Forstå Floyd-Warshall; forstå *forgjengerstrukturen* for *alle-til-alle*-varianten av korteste vei-problemet (inkl. operasjonene Print-All-Pairs-Shortest-Path)

5 % **15** Vurder følgende utsagn om minimale spenntrær (*minimum spanning trees*) i en vektet, urettet graf $G = (V, E)$ der alle kantene har forskjellig vekt:

> Hvis nodene V kan deles i mengdene X og Y (uten overlapp) så vil det minimale spenntreet nødvendigvis inneholde kanten mellom X og Y med lavest vekt.

Stemmer dette? Svar ja eller nei og forklar kort.

(Her holder det ikke nødvendigvis å beskrive hva en pensumalgoritme gjør.)

Ja. Ellers kunne man finne en sti i treet mellom kantens endepunkter, og bytte ut dens tyngste kant med den nye kanten og få en bedre løsning.

Siden alle kantvektene er forskjellige, vil en slik endring alltid være mulig.

Her menes altså det det minimale spenntreet, *om det eksisterer*, ikke at grafen nødvendigvis har et minimalt spenntre (dvs., at den er sammenhengende). Om man problematiserer dette er det bare positivt, men om man ikke får med koblingen til trygge kanter, vil det ikke gi full uttelling.

Relevante læringsmål: Vite hva *spenntrær* og *minimale spenntrær* er; forstå MST-KRUSKAL; forstå hvorfor *lette kanter* er *trygge kanter*

5 % **16** Din venn Lurvik påstår han har laget en algoritme basert på dynamisk programmering, som setter sammen det kuleste antrekket basert på klærne i garderoben hans. Når du ser på algoritmen, ser du at den kun beregner kulhetsgraden til det kuleste antrekket, *uten å faktisk si noe om hvilke klær som inngår*! Han mener det er en triviell forskjell, og at det vil være enkelt å legge til den funksjonaliteten. Hva tror du? Forklar kort.

Her har trolig Lurvik rett. Optimering basert på dynamisk programmering er gjerne basert på en serie beslutninger. Om vi for hver delløsning lagrer hvilken beslutning som tas, kan løsningen lett rekonstrueres.

Her kan problemstillingen virke uklar, og det kan være vanskelig å skjønne hva det spørres etter, men det er en del av oppgaven. Om man i tilstrekkerlig grad har forstått at det i dynamisk programmering ofte holder å lage optimeringsalgoritme som finner optimal kostnad, og deretter lagre beslutningene som tas (som f.eks. i *0-1-ryggsekkproblemet*), så vil man kjenne igjen strategien.

Relevante læringsmål: Forstå designmetoden *dynamisk programmering*; forstå løsningen på *0-1-ryggsekkproblemet*; forstå hvordan man *rekonstruerer* en løsning fra lagrede beslutninger

5 % **17** Din venn Gløgsund vil vite om en graf har et minimalt spenntre med vekt mindre eller lik k, og mener hun har klart å redusere problemet i polynomisk tid til handelsreiseproblemet (*the traveling-salesman problem*). Din andre venn Klokland klarer ikke helt å følge beviset hennes, men mener det umulig kan stemme. Tror du det er mulig? Forklar kort.

Dette må være mulig, siden spenntreproblemet er i NP (kan verifiseres i polynomisk tid) og TSP er NP-komplett (så alt i NP kan reduseres til det).

Her er noe av poenget å teste at man har skjønt hvilken retning man må redusere for å vise hva. Om man antyder at en reduksjon *til* TSP gjør et problem NP-komplett, vil man dermed ikke få noen uttelling. Om man derimot indikerer at det ikke er noe problem å redusere til TSP, uten at man eksplisitt nevner at alle problemer i NP kan reduseres til TSP, så vil man likevel få en del uttelling.

Relevante læringsmål: Forstå definisjonen av NP-*kompletthet*; forstå den konvensjonelle hypotesen om forholdet mellom P, NP og NPC; kjenne det NP-komplette problemet TSP; være i stand til å konstruere enkle NP-kompletthetsbevis

5 % **18** Anta at du har en rettet graf $G = (V, E)$ med positive heltalls-kantvekter, der $s, t \in V$. Denne grafen kan ha flere enkle stier fra s til t med minimal lengde, det vil si flere «korteste» veier. Hvordan vil du finne den av dem som består av *flest mulig* kanter?

For eksempel bytt ut $w(u, v)$ med $|V| \cdot w(u, v) - 1$.

Flere kanter vil da lønne seg, men aldri så mye at det lønner seg å velge en sti som er lengre med de opprinnelige vektene. En forskjell i kantvekt uansett utgjøre mer enn det største antall kanter i en sti.

Her vil man naturligvis også få full uttelling dersom man konstruerer finner en annen effektiv, korrekt måte å løse problemet på, f.eks. ved å modifisere en algoritme heller enn å redusere til korteste vei.

Merk: Her har man naturligvis ikke *oppgitt* de korteste veiene; dem vil det jo gjerne være eksponentielt mange av. Bruk av BFS her vil kunne gi noe uttelling, siden det vitner om forståelse av at BFS finner korteste veier, målt i antall kanter, men det løser ikke problemet, så uttellingen vil være begrenset.

Relevante læringsmål: Forstå at *korteste enkle vei* kan løses vha. *lengste enkle vei* og omvendt; være i stand til å bruke eksisterende algoritmer på nye problemer; kunne konstruere nye effektive algoritmer

5 % **19** Du har oppgitt tre sekvenser $A = \langle a_1, \ldots, a_m \rangle$, $B = \langle b_1, \ldots, b_n \rangle$ og $X = \langle x_1, \ldots, x_{m+n} \rangle$ og ønsker å avgjøre om X er en sammenfletting av A og B, dvs., at X består av elementene til A og B, i sin opprinnelige rekkefølge, men flettet i hverandre, så f.eks.

$$X = \langle a_1, a_2, b_1, a_3, b_2, b_3, a_4, \ldots, b_n, a_{m-1}, a_m \rangle .$$

Du kan tenke på en slik sammenfletting som å først sette sammen A og B til $\langle a_1, \ldots, a_m, b_1, \ldots, b_n \rangle$ og så (kanskje) endre rekkefølgen på elementene (uten å legge til eller fjerne noen elementer, og der a_i fortsatt kommer før a_{i+1} og b_i fortsatt kommer før b_{i+1}). Det er godt mulig at A og B inneholder noen like verdier, og at disse forekommer flere ganger.

Beskriv en algoritme som løser problemet.

La $T[i, j, k]$ være svaret for $A[1..i]$, $B[1..j]$, $X[1..k]$. Anta at alle elementer i T er FALSE til å begynne med. La $T[0, 0, 0] =$ TRUE. For $i, j, k \geqslant 1$:

```
1  if X[k] == A[i]
2        T[i, j, k] = T[i, j, k] ∨ T[i − 1, j, k − 1]
3  if X[k] == B[j]
4        T[i, j, k] = T[i, j, k] ∨ T[i, j − 1, k − 1]
```

Her kan man naturligvis forklare ting helt annerledes, med kode, pseudokode eller prosa. Man kan også lage en rekursiv, memoisert løsning, om man foretrekker det. Det vil også gi full uttelling.

Om man antar at alle elementene i X er forskjellige, så kan problemet løses i lineær tid med en prosedyre som er omtrent motsatt av MERGE, dvs., man går gjennom X og flytter en indeks i enten A eller B, avhengig av hva man finner. Straks man finner et element i X som stemmer med elementet i både A og B, bryter denne metoden sammen. Dette svaret vil gi noe, men ikke full uttelling.

Relevante læringsmål: Forstå designmetoden *dynamisk programmering*; forstå eksemplet *LCS*; være i stand til å bruke eksisterende algoritmer på nye problemer; kunne konstruere nye effektive algoritmer

5 % **20** Du har et spillbrett, representert ved en rettet graf $G = (V, E)$, med et sett med k startposisjoner $A \subset V$ og et sett med k sluttposisjoner $Z \subset V$ oppgitt. Du har også k brikker som til å begynne med står i startposisjonene A (én brikke i hver posisjon). Du skal flytte brikkene langs kantene i G slik at det til slutt står én brikke i hver sluttposisjon i Z. (Det spiller ingen rolle hvilken brikke som havner i hvilken sluttposisjon.)

Du skal utføre flyttingen som en serie med maks n trekk. I hvert trekk kan du flytte så mange brikker du vil (samtidig), men de kan maks flytte seg ett hakk hver (altså langs én kant), og du kan ikke flytte to brikker til samme posisjon (node) samtidig. Du kan anta at $k \geqslant 1$ og at det er mulig å finne en løsning.

Beskriv en algoritme som løser problemet.

Lag en ny graf $G' = (V', E')$, der V' består av $n + 1$ kopier av V, $V_1, \ldots, V_{n+1}$, der $v \in V$ kopieres til $v_i \in V_i$. Legg til kanter (v_i, v_{i+1}) for $i = 1, \ldots, n$ og (u_i, v_{i+1}) for $(u, v) \in E$. Finn så stier fra startposisjonene til sluttposisjonene som ikke krysser hverandre vha. flyt med kant- og nodekapasiteter på 1.

V_i er posisjonene i runde i; kant (v_i, v_{i+1}) representerer en brikke som står stille, mens (u_i, v_{i+1}) representerer en brikke som flyttes langs kanten (u, v) i trekk nr i. En sti fra en kilde til et sluk blir et gyldig sett med trekk for én brikke, og så lenge de ikke krysser hverandre, vil aldri noen brikker så på samme node til samme tid.

Nodekapasiteter innfører vi som vanlig vha. splitting og flere kilder/sluk implementeres som normalt, ved å legge til superkilde og supersluk.

Relevante læringsmål: Kunne definere *flytnett*, *flyt* og *maks-flyt-problemet*; kunne håndtere *flere kilder og sluk*; forstå *heltallsteoremet* (*integrality theorem*); være i stand til å bruke eksisterende algoritmer på nye problemer; kunne konstruere nye effektive algoritmer