TDT4120 Algoritmer og datastrukturer

Eksamen, 7. august 2019, 09:00–13:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	D

Oppgaver

6 % **1** Din venn Klokland mener hun har funnet en topp-hemmelig algoritme, der deler av pseudokoden er sensurert (se nedenfor). Du gjenkjenner den straks som en pensumalgoritme. Hvilken?

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

> PARTITION
>
> Merk: Man trenger ikke oppgi de sensurerte kodelinjene i svaret.
>
> **Relevant læringsmål:** Forstå QUICKSORT og RANDOMIZED-QUICKSORT

6 % **2** Klokland har også funnet en sensurert beskrivelse av *deler* av en annen algoritme, som du gjenkjenner som MST-PRIM (se nedenfor). Hvordan skal linje 7 egentlig se ut, uten sensurering? (Her holder det med en kort forklaring av hva linjen skal gjøre, om du ikke husker eksakt notasjon fra pensum.)

```
6  while Q ≠ ∅
7       u = EXTRACT-MIN(Q)
8       for each v ∈ G.Adj[u]
9            if v ∈ Q and w(u, v) < v.key
10                v.π = u
11                v.key = w(u, v)
```

6 % **3** Klokland roter frem en sensurert beskrivelse av deler av *enda* en algoritme, som du gjenkjenner som FLOYD-WARSHALL (se nedenfor). Hvordan skal linje 7 egentlig se ut, uten sensurering? (Her holder det med en kort forklaring av hva linjen skal gjøre, om du ikke husker eksakt notasjon fra pensum.)

```
5    for i = 1 to n
6        for j = 1 to n
7            d_ij^(k) = min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
```

Relevant læringsmål: Forstå FLOYD-WARSHALL

6 % **4** Klokland har et siste sensurert algoritmefragment, denne gangen slutten av tellesortering (COUNTING-SORT, se nedenfor). Hva betyr det at tellesortering er *stabil*? Hva er det man må passe på i linje 10 for at algoritmen skal bli stabil? Forklar kort hvorfor.

```
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Stabilitet betyr at den ikke bytter rekkefølge på like elementer. Man må passe på å telle *nedover*. Det er fordi C er en kumulativ telling av forekomster, så $C[A[j]]$ angir *siste* posisjon i B til tegnet $A[j]$. For å sørge for at $A[j]$ er siste forekomst også i B, må vi telle nedover.

Relevant læringsmål: Forstå COUNTING-SORT, og hvorfor den er stabil

6 % **5** Gi en svært kort beskrivelse av hvordan binærsøk (BISECT) fungerer. Hva er kjøretiden? (Du trenger ikke bruke kode eller pseudokode. Det holder med en kort tekstlig forklaring.)

BISECT Leter etter et v element i en sortert tabell A. Sjekker først om v er lik midterste element. Dersom det er større, søk rekursivt i høyre halvdel; ellers, søk i venstre. Kjøretiden er logaritmisk.

Relevant læringsmål: Forstå BISECT og BISECT′

5 % **6** La W, nedenfor, være vektmatrisen til en urettet graf G.

W

	1	2	3	4	5
1	0	4	5	∞	1
2	4	0	∞	2	4
3	5	∞	0	7	8
4	∞	2	7	0	3
5	1	4	8	3	0

Utfør Kruskals algoritme (MST-KRUSKAL) på grafen G. List opp kantene i den rekkefølgen de legges til i løsningen. (Dersom $i < j$, oppgi kanten mellom i og j som (i, j). Skriv én kant per linje i svaret.)

(1, 5)
(2, 4)
(4, 5)
(1, 3)

Relevante læringsmål: Forstå MST-KRUSKAL; vite hvordan den oppfører seg; kunne utføre algoritmen, trinn for trinn

6 % **7** La oss si du skulle løse rekurrensen $T(n) = 3T(n/4) + n$ med iterasjonsmetoden (*repeated substitution*), der du ekspanderer det rekursive leddet gjentatte ganger, og hver ekspandering foregår på en ny linje. Hvor mange linjer hadde du trengt? (Oppgi svaret i Θ-notasjon. Merk: Du skal *ikke* løse rekurrensen.)

For hver linje vil parameteren til det rekursive kallet deles på 4, så vi får $\Theta(\log_4 n) = \Theta(\lg n)$ linjer.

Relevant læringsmål: Kunne løse rekurrenser med *iterasjonsmetoden*

6 % **8** Du sorterer n tall ved å sette dem inn (i tilfeldig rekkefølge) i et binært søketre, for så å kjøre INORDER-TREE-WALK. Hva blir den forventede totale kjøretiden for innsetting med påfølgende traversering? Forklar kort hvordan du kommer frem til svaret.

Den forventede høyten til treet er $\Theta(\lg n)$, så hver av de n innsettingene tar logaritmisk tid. Traverseringen til slutt tar lineær tid. Totalt blir dette $\Theta(n \lg n)$.

Relevant læringsmål: Forstå hvordan *binære søketrær* fungerer (inkl. operasjoner INORDER-TREE-WALK og TREE-INSERT); vite at forventet høyde for et tilfeldig binært søketre er $\Theta(\lg n)$

6 % **9** Du skal velge ut videokanaler til forsiden på et videonettsted. Du har et sett med kanaler som hver har en estimert målgruppe, oppgitt som et aldersintervall (f.eks. *fra 10 til 25 år*). Du skal velge ut flest mulig kanaler, men får ikke velge noen som har overlappende målgrupper. Hvordan vil du gå frem? Forklar kort.

Sorter etter høyeste alder og plukk én og én kanal fra starten, der du hopper over dem der målgruppene overlapper. Dette er det samme som aktivitets-utvelgelses-problemet i pensum.

Relevant læringsmål: Forstå eksemplene *aktivitet-utvelgelse* og *det fraksjonelle ryggsekkproblemet*

6 % **10** Dine venner Lurvik og Smartnes diskuterer forholdet mellom abstrakte beslutningsproblemer, konkrete beslutningsproblemer og formelle språk. Begge er enige om at beslutningsproblemer kan representeres som formelle språk, men mens Lurvik mener at vi da omtaler dem som abstrakte, så mener Smartnes at det er dette vi omtaler som konkrete problemer. Hvem har rett?

Dette er *konkrete* problemer.

Relevant læringsmål: Forstå forskjellen på *konkrete* og *abstrakte* problemer

6 % **11** Hvis kanten (u, v) i et flytnettverk har flyt 4 og kapasitet 11, hvilke kanter vil finnes mellom u og v i restnettet, og hvilke kapasiteter vil de ha?

Vi vil ha $c_f(u, v) = 7$ og $c_f(v, u) = 4$.

Relevant læringsmål: Kunne definere *restnettet* til et flytnett med en gitt flyt

6 % **12** Din venn Klokland har laget følgende hashfunksjon $h : \mathrm{U} \to \{0, \ldots, m-1\}$ til bruk i hashtabellen $\mathrm{T}[0\,..\,m-1]$, der universet U er alle positive heltall:

$$h(k) = \min\big(\lfloor m(k\mathrm{A} \bmod 1)\rfloor + 2^k, m\big) - 1$$

Her er $\mathrm{A} \approx (\sqrt{5}-1)/2$. Er h en god hashfunksjon? Diskutér kort.

Nei. Dersom $k > \log_2 m$ så vil $h(k) = m$, så alle slike verdier vil kollidere.

Vi har ingen grunn til å tro at k skal være spesielt liten. Dersom vi likevel kunne anta det, så vil vi bare kunne ha $\mathrm{O}(\lg m)$ ulike verdier, og dermed $\mathrm{O}(\lg m)$ elementer, som betyr at vi trenger en eksponentielt stor hashtabell.

Relevant læringsmål: Kjenne til grunnleggende *hashfunksjoner*

7 % **13** Din venn Gløgsund grubler over korteste-vei-problemet og negative sykler, og hun har klart å forvirre seg selv grundig. Er problemet NP-hardt eller er det bare meningsløst? Og hvilken rolle spiller enkle stier (*simple paths*) oppi det hele? Gi en kort oppsummering.

Noen punkter:

- Korteste vei med negative sykler: Meningsløst dersom noen av løsningene kan inneholde en negativ sykel, siden ingen sti vil være kortest.
- Korteste vei uten negative sykler: Kan løses i polynomisk tid. (Merk at vi fortsatt ikke vet om det er NP-hardt – men det er bare tilfelle dersom P = NP.)
- Korteste *enkle* vei med negative sykler: Gir mening, siden løsningen uansett ikke kan inneholde en sykel, men er NP-hardt.
- Korteste *enkle* vei *uten* negative sykler: Samme problem som korteste vei uten negative sykler generelt, siden løsningen alltid vil være en enkel sti.

Relevant læringsmål: Forstå at negative sykler gir mening for korteste *enkle vei* (*simple path*)

7 % **14** Din venn Gløgsund mener hun har funnet en verifikasjonsalgoritme med polynomisk kjøretid for komplementet til HAM-CYCLE. Om hun har rett, hvilke konsekvenser har det for forholdet mellom klassene P, NP og co-NP? Forklar.

Om hun kan verifisere komplementet til HAM-CYCLE i polynomisk tid, betyr det at HAM-CYCLE er i co-NP. Alle problemer i NP kan da få sine komplement verifisert ved å redusere problemene til HAM-CYCLE, så NP $\subseteq$ co-NP.

Dersom ethvert problem i NP også er i co-NP, så må *komplementet* av ethvert problem i NP også være komplementet til et problem i co-NP, så co-NP $\subseteq$ NP, og dermed NP $=$ co-NP.

(Vi kan ikke konkludere noe spesielt om P.)

Relevant læringsmål: Forstå representasjonen av beslutningsproblemer som *formelle språk*; forstå definisjonen av klassene P, NP og co-NP; kjenne det NP-komplette problemet HAM-CYCLE

7 % **15** Du skal løse et problem i to faser. I første fase får du oppgitt en serie med m ligninger av typen $x_i = x_j$, der i og j er heltall i området $1, \ldots, n$. I andre fase får du oppgitt en serie ulikheter av typen $x_i \neq x_j$, og du skal da avgjøre om disse kan være sanne, gitt den første serien med ligninger. Anta at du vil behandle både ligningene og ulikhetene så effektivt som mulig, i verste tilfelle. Hvordan vil du gå frem? Hvordan ville du gå frem om du fikk bruke vilkårlig lang tid i første fase (dvs., behandle ligningene som et *statisk datasett*)?

En mulig løsning er å representere ligningene som kanter E i en graf (V, E), der $V = \{1, \ldots, n\}$, og så bruke skog-implementasjonen av disjunkte mengder, der ligningen $x_i = x_j$ behandles med operasjonen LINK(i, j) og ulikheten $x_i \neq x_j$ avgjøres ved FIND-SET(i) == FIND-SET(j). (Her foreldrepekere implementeres som en tabell $p[1\,..\,n]$, der $i.p = p[i]$.)

Denne løsningen vil i praksis gi nesten konstant kjøretid for ulikhetene, men om vi kan se på ligningene som et statisk datasett, kan vi konstruere en hashtabell T der HASH-SEARCH(T, i) og HASH-SEARCH(T, j) gir samme (vilkårlige) verdi dersom vi har oppgitt at $x_i = x_j$.

Relevante læringsmål: Forstå *skog*-implementasjonen av *disjunkte mengder* (inkl. operasjonene CONNECTED-COMPONENTS, SAME-COMPONENT, MAKE-SET, UNION, LINK, FIND-SET); vite at man for *statiske datasett* kan ha *worst-case* $O(1)$ for søk

7 % **16** Du har oppgitt to sekvenser $X = \langle x_1, \ldots, x_m \rangle$ og $Y = \langle y_1, \ldots, y_n \rangle$ og ønsker å lage to sekvenser $A = \langle a_1, \ldots, a_k \rangle$ og $B = \langle b_1, \ldots, b_k \rangle$ som er så lange som mulig (dvs., k er så stor som mulig), og der følgende holder:

1. $a_i \in \{1, \ldots, m\}$ og $b_i \in \{1, \ldots, n\}$ (for $i = 1, \ldots, k$)
2. $a_i < a_{i+1}$ og $b_i < b_{i+1}$ (for $i = 1, \ldots, k-1$)
3. $x_{a_i} < y_{b_i}$ (for $i = 1, \ldots, k$)

Beskriv en algoritme som løser problemet. Hva blir kjøretiden?

(En stor del av oppgaven her er å forstå oppgaveteksten.)

Dette er svært likt problemet *lengste felles subsekvens* (LCS); forskjellen ligger i at det tredje punktet krever $x_{a_i} < y_{b_i}$ heller enn $x_{a_i} = y_{b_i}$, som i LCS. Vi kan bruke samme løsning, og bare bytte ut dette kriteriet. Dvs., vi får vi følgende rekursiv løsning:

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

Kjøretiden blir $\Theta(nm)$.

Merk at samme resonnement gjelder: Det skader aldri å bruke det midterste tilfellet dersom $x_i < y_j$. Dersom vi har en gyldig løsning som ender med tidligere elementer i X og/eller Y, så har løsningen samme lengde dersom vi bruker x_i og y_j i stedet.

Relevante læringsmål: Forstå designmetoden *dynamisk programmering*; forstå eksemplene *stavkutting* og *LCS*