TDT4120 Algoritmer og datastrukturer

Eksamen, 5. august 2024, 09:00–13:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	E

Løsningsforslag

Løsningsforslagene i rødt nedenfor er *eksempler* på svar som vil gi 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!

Merk: Det å forstå hva det spørres etter *er en del av oppgaven*. Enkelte oppgaver kan være konstruert slik at det er nettopp dette som er utfordringen, så om du ikke får dem til, vil de kunne virke uklare eller tvetydige.

1 Det følgende er hentet fra ENQUEUE:

```
1  Q[Q.tail] = x
2  if Q.tail == Q.size
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

Hva skal den sensurerte biten være?

> Se løsning i pseudokoden over.
>
> **Relevant læringsmål:** Forstå hvordan *stakker* og *køer* fungerer (inkl. operasjonene STACK-EMPTY, PUSH, POP, ENQUEUE, DEQUEUE).

2 Anta at du kjører MST-PRIM og MST-KRUSKAL på en usammenhengende graf. Hvilken av algoritmene vil finne et minimalt spenntre for hver av de sammenhengende komponentene i grafen? Forklar kort.

Det vil si, hvilken av dem vil konstruere en usammenhengende løsning som dekker hele grafen?

> MST-KRUSKAL, siden den alltid plukker den billigste gjenværende kanten som ikke danner en sykel, samme hvor i grafen den befinner seg. MST-PRIM, derimot, konstruerer et (sammenhengende) tre som vokser ut fra én

startnode, og vil aldri kunne bevege seg videre til en annen komponent.

Her vil kortere og enklere forklaringer kunne være fullgode.

Relevante læringsmål: Forstå MST-KRUSKAL; forstå MST-PRIM.

3 En av løkkene i COUNTING-SORT går fra n ned til 1 (**for** $j = n$ **downto** 1). Hva er konsekvensen av å skifte retning på løkka (**for** $j = 1$ **to** n)?

Merk: Her kreves ingen forklaring.

Sorteringen blir ustabil.

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

4 Hvis du skal beskrive *best-case*-kjøretiden til en algoritme, hvilken asymptotisk notasjon (av O, Ω eller Θ) er det best å bruke, om mulig?

Θ

Den angir både en øvre og nedre grense for kjøretiden, og gir dermed mest informasjon.

Relevant læringsmål: Forstå at alle av O, Ω, Θ, o og ω kan beskrive *best-*, *worst-* og *average-case*.

5 I en hashtabell med hashfunksjon h, hva vil det si at nøklene k_1 og k_2 kolliderer?

$h(k_1) = h(k_2)$

Dvs. at de har samme hashverdi og mappes til samme posisjon (*slot*). Om vi bruker kjeding (*chaining*), vil de havne i dem samme lenkede listen, på indeks $h(k_1)$ i tabellen.

Relevant læringsmål: Forstå hvordan *direkte adressering* og *hashtabeller* fungerer.

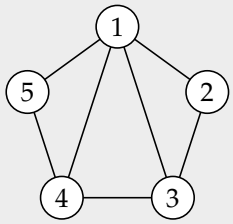
6 Hva er den amortiserte kjøretiden til TABLE-INSERT?

Det er altså snakk om innsetting i en dynamisk tabell, der vi enten kan sette elementet rett inn, om det er plass, eller må allokere en ny og større tabell ellers. Oppgi svaret med Θ-notasjon.

$\Theta(1)$

Relevant læringsmål: Forstå hvordan *dynamiske tabeller* fungerer (inkl. operasjonene TABLE-INSERT).

Figur 1 Graf til oppgave 10



7 Du har en rettet, uvektet graf $G = (V, E)$, og skal finne korteste veier *fra alle noder* i V *til én gitt node t*. Hvordan vil du gå frem?

Snu retningen på alle kantene i G, og kjør BFS fra t.

Relevante læringsmål: Forstå ulike varianter av *korteste-vei-* eller *korteste-sti*-problemet (*Single-source*, *single-destination*, *single-pair*, *all-pairs*); forstå BFS, også for å finne *korteste vei uten vekter*.

8 Du ønsker å finne lengste enkle vei fra node s til node t i en vektet graf. Hvordan kan du gjøre det? Er det tilfeller der metoden ikke vil fungere? Forklar kort.

Gang alle kantvekter med -1 og finn korteste vei med en standard algoritme som f.eks. Bellman-Ford. Det vil ikke fungere (dvs., alle kjente algoritmer vil feile) dersom man kan gå innom en positiv sykel i den opprinnelige grafen (som altså blir en negativ sykel i den nye).

Det skader ikke om man har positive sykler generelt, så lenge ingen av stiene fra s til t kan gå innom noen av dem. Man vil likevel få full uttelling om man sier at metoden ikke vil fungere dersom grafen har positive sykler.

Det vil fortsatt være *mulig* å finne korteste enkle vei (om den eksisterer) uansett. Men om $P \neq NP$, vil det ikke kunne gjøres i polynomisk tid.

Relevant læringsmål: Forstå at *lengste enkle vei* kan løses vha. *korteste enkle vei*; forstå at *lengste-enkle-vei*-problemet er NP-hardt.

9 Løsningen på det binære ryggsekkproblemet (*0-1 knapsack*) har kjøretid $\Theta(nW)$, der n er antall gjenstander og W er kapasiteten til ryggsekken. Er dette en polynomisk algoritme? Forklar kort.

Nei, fordi W vokser eksponentielt med problemstørrelsen.

Relevant læringsmål: Forstå hvorfor løsningen på det binære ryggsekkproblemet *ikke er polynomisk*.

10 Du skal representere grafen i figur 1 som en nabomatrise. Fyll inn 0 og 1 i tabellen under.

	1	2	3	4	5
1		1	1	1	1
2	1		1		
3	1	1		1	
4	1		1		1
5	1			1	

Se tabellen over.

Her er 0-verdier utelatt for økt lesbarhet.

Relevant læringsmål: Forstå hvordan grafer kan implementeres.

11 Hva er et nodedekke (*vertex cover*)?

Et nodedekke for en graf $G = (V, E)$ er en delmengde $V' \subseteq V$ slik at hvis $(u, v) \in E$, så er minst én av u og v i V'.

Relevant læringsmål: Kjenne det NP-komplette problemet VERTEX-COVER.

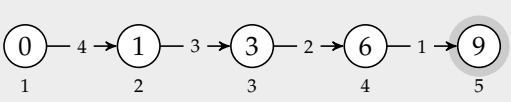
12 I pensumdefinisjonen av flytnett (*flow networks*) tillates ikke antiparallelle kanter, altså at vi har en kant både fra v_1 til v_2 og fra v_2 til v_1. Dersom vi likevel har slike kanter, hvordan kan vi håndtere det?

Vi kan splitte én av dem ved å sette inn en ny node.

Vi kan f.eks. erstatte (v_1, v_2) med (v_1, v') og (v', v_2), der v' er en ny node. De to nye kantene får samme kapasitet som (v_1, v_2).

Relevant læringsmål: Kunne håndtere *antiparallelle kanter*.

13 På tilsvaraende måte som i BELLMAN-FORD, skal du utføre RELAX på alle kantene i den følgende grafen én gang. Rekkefølgen er ikke gitt.



Når du er ferdig, hva er den minste og største verdien 5.d kan ha (altså $v.d$ for node 5, uthevet)? Oppgi svaret som to tall, adskilt med komma.

Hver nodes d-verdi før du starter er angitt i noden i figuren, så f.eks. $4.d = 6$.

Du skal altså *ikke* utføre hele BELLMAN-FORD, men oppdatere estimatet én gang langs hver kant, i en eller annen rekkefølge.

6, 7

Relevant læringsmål: Forstå *kant-slakking* (*edge relaxation*) og RELAX.

14 Løs følgende rekurrens:

$T(n) = T(n-1) \cdot 2^{2^n} \qquad (n \geqslant 1)$

$T(0) = 2$

Oppgi svaret eksakt.

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

$$\begin{aligned} T(n) &= T(n-1) \cdot 2^{2^n} \\ &= T(n-2) \cdot 2^{2^{n-1}} \cdot 2^{2^n} \\ &= T(n-3) \cdot 2^{2^{n-2}} \cdot 2^{2^{n-1}} \cdot 2^{2^n} \\ &\vdots \\ &= 2^{2^0} \cdot 2^{2^1} \cdots 2^{2^{n-1}} \cdot 2^{2^n} \\ &= 2^{2^0+2^1+2^{n-1}+2^n} \\ &= 2^{2^{n+1}-1} \end{aligned}$$

Relevant læringsmål: Kunne løse rekurrenser med *iterasjonsmetoden*; ha noe kjennskap til rekkesummer (inkl. $0 + 1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$).

15 I TRANSITIVE-CLOSURE angir t_{ij} om det finnes en sti fra i til j. Anta nå at den rettede grafen du får som input er asyklisk. Hvordan kan du endre algoritmen så t_{ij} blir *antall* stier fra i til j?

Du kan ev. beskrive løsningen din som en endring av FLOYD-WARSHALL.

I iterasjon k, la $t_{ij} = t_{ij} + t_{ik} \cdot t_{kj}$.

Når vi får lov til å gå innom k, tell alle stier vi har som ikke går innom k, og så tell alle kombinasjoner av stier fra i til k og stier fra k til j.

Siden vi ikke har sykler, vet vi at stiene fra i til k ikke kan dele noder med stiene fra k til j, så antallet kombinasjoner blir $t_{ik} \cdot t_{kj}$.

Mer drastiske endringer, som å bytte ut hele algoritmen med mer direkte dynamisk programmering (nært beslektet med DAG-SHORTEST-PATHS) gir liten eller ingen uttelling.

Relevant læringsmål: Forstå FLOYD-WARSHALL og TRANSITIVE-CLOSURE.

16 Et problem med QUICKSORT er at kjøretiden blir dårlig om pivotelementet er dårlig. Kan man velge pivot slik at kjøretiden garantert blir $\Theta(n \lg n)$? Forklar.

Her er det snakk om å kun modifisere hvordan man velger pivot; resten av QUICKSORT skal utføres som normalt. Hvert rekursive kall skal også utføres på samme måte, så man kan ikke f.eks. bruke MERGE-SORT til å begynne med for å «jukse seg til» riktig kjøretid.

> Ja. Bruk SELECT til å finne medianen i lineær tid.
>
> Merk at dette vil gi en mye høyere konstantfaktor enn vanlig QUICKSORT, og vil trolig gi atskillig dårligere kjøretid i praksis.
>
> **Relevante læringsmål:** Forstå QUICKSORT; kjenne til SELECT.

17 Du har n gjenstander og skal gi én til hver av n personer. Personene kan foretrekke ulike gjenstander. Helst vil du at ingen skal misunne noen andre, men du innser at det neppe er mulig.

I stedet lager du et lotteri for hver gjenstand. Heller enn å gi ut gjenstandene direkte, får hver person en tilfeldig *prioritet* for hver gjenstand. Målet ditt er at ingen skal misunne noen som har lavere prioritet.

Vil det alltid være mulig å fordele gjenstandene slik? Hvordan?

Om du har fått gjenstand x, så skal jeg altså ikke misunne deg, med mindre jeg har lavere prioritet for gjenstand x.

> Ja. Man kan bruke GALE-SHAPLEY, som gir en stabil matching, som tilsvarer den typen fordeling vi er ute etter.
>
> Relevante læringsmål: Forstå hva stabil matching (*stable matching*) er; forstå GALE-SHAPLEY.

18 Din venn Lurvik studerer to beslutningsproblemer, A og B, der han har en eksponentiell algoritme for A og en polynomisk algoritme for B. Han har vist at A ikke kan løses raskere enn eksponentielt.

Lurvik har også funnet reduksjoner fra A til B og fra B til A. Hva kan du si om kjøretiden til hver av disse reduksjonene? Forklar kort.

Om vi ser på A og B som formelle språk, har Lurvik altså funnet to reduksjonsfunksjoner f og g, der

$x \in \text{A}$ hvis og bare hvis $f(x) \in \text{B}$ og

$x \in \text{B}$ hvis og bare hvis $g(x) \in \text{A}$.

Spørsmålet er hva du kan si om kjøretiden som kreves for å beregne reduksjonsfunksjonene f og g.

I pensum antas en reduksjon generelt å ha polynomisk kjøretid, men her kan du se bort fra det.

Algoritme 1 Inversen av ZIP

```
UNZIP(x)
 1  if x == NULL
 2      let L, R be new lists
 3  else allocate new nodes y and z
 4      y.key = x.key[1]
 5      z.key = x.key[2]
 6      L = UNZIP(x.next)[1]
 7      R = UNZIP(x.next)[2]
 8      LIST-PREPEND(L, y)
 9      LIST-PREPEND(R, z)
10  return ⟨L, R⟩
```

For å redusere fra A til B kreves eksponentiell kjøretid. Kunne vi gjøre det raskere, kunne vi løse A raskere, og det har Lurvik bevist er umulig.

Vi kan ikke si noe om kjøretiden til reduksjonen i motsatt retning.

Det eksisterer reduksjoner fra B til A med polynomisk kjøretid, som bare løser B og så velger én av to mulige instanser for å A, for å få riktig svar. Men hvis B inneholder store nok instanser, kan reduksjonsfunksjonen også mappe fra instanser av størrelse n til instanser av størrelse 2^n eller $n!$ eller verre, og vil da kunne kreve vilkårlig kjøretid.

Her gis det også full uttelling om man beskriver hva som er mulig, heller enn hva vi kan si om Lurviks reduksjoner, det vil si, om man sier at vi kan redusere i polynomisk tid fra B til A.

Relevant læringsmål: Forstå *redusibilitets-relasjonen* $\leqslant_{\mathrm{P}}$.

19 I mange programmeringsspråk har man en funksjon som heter ZIP, som tar inn to lister, og returnerer en liste av par, der par i består av element i fra hver av de to listene.

Prosedyren UNZIP (algoritme 1) gjør det motsatte. Den tar inn hodet til en lenket liste (*linked list*) av par (tabeller av lengde 2) og fordeler dem i to lister L og R.

Hvordan ville du ha endret algoritmen for å forbedre kjøretiden? Hva blir kjøretiden før og etter forbedringen din? Forklar.

En enkel løsning er å bytte ut linje 6 og 7 med noe som dette:

```
6       L, R = UNZIP(x.next)
```

Da reduseres antallet rekursive kall fra 2 til 1, og man endrer rekurrensen for kjøretiden fra $T(n) = 2T(n-1) + \Theta(1)$ til $T(n) = T(n-1) + \Theta(1)$, og man går fra eksponentiell til lineær kjøretid, altså fra $\Theta(2^n)$ til $\Theta(n)$.

Man kan selvfølgelig beskrive løsningen på flere måter, som f.eks.:

```
6       U = UNZIP(x.next)
7       L = U[1]
8       R = U[2]
```

20 COUNTING-SORT(A, n, k) tar inn en tabell $A[1:n]$ med heltall i området $0, \ldots, k$ og fyller en tabell $C[0:k]$ med antall forekomster i A av hver mulige verdi, og bruker $\Theta(n+k)$ operasjoner på dette.

Du skal nå gjøre en lignende telling, der A allerede er sortert. Du kan anta at du også får inn C som parameter, og at C er initialisert, så $C[i] = 0$ for $i = 0, \ldots, k$.

Bruk metoden *splitt og hersk* til å konstruere en algoritme som løser problemet med kjøretid $O(n)$ generelt, men som er raskere enn dette når A inneholder mange duplikater.

Om første og siste element er forskjellige, løs rekursivt for hver halvdel. Ellers øk $C[i]$ med lengden av intervallet, der i er første element.

Pseudokode kreves ikke, men inkluderes her for å vise detaljer:

```
FREQ(A, C, p, r)
1  if A[p] == A[r]
2       C[A[p]] = C[A[p]] + r - p + 1
3  else q = ⌊(p + r)/2⌋
4       FREQ(A, C, p, q)
5       FREQ(A, C, q + 1, r)
```

Prosedyren kalles initielt med FREQ$(A, C, 1, n)$, der det antas at $n \geqslant 1$.

At kjøretiden er $O(n)$ følger av rekurrensen $T(n) \leqslant 2T(n/2) + \Theta(1)$. Jo flere ganger linje 1 slår inn, jo lavere vil kjøretiden være.

Om man bruker binærsøk separat for å finne starten og slutten på forekomstene av hver verdi, er ikke kjøretiden $O(n)$. Det vil likevel gi 4 poeng.

Relevant læringsmål: Forstå designmetoden *divide-and-conquer* (*splitt og hersk*).