TDT4120 Algoritmer og datastrukturer

Eksamen, 9. desember 2024, 09:00–13:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	E

Oppgaver

5 % **1** Hva er kjøretiden til KRUSKAL?

Oppgi svaret med O-notasjon.

5 % **2** Hvilke antagelser gjør vi normalt om input til BUCKET-SORT?

5 % **3** Hvordan vurderer du følgende hashfunksjon? Forklar kort.

$h(k) = \min\{m, 2^k\}$

Spørsmålet er altså hvor god eller dårlig hashfunksjonen er, og hvorfor. Her er k et positivt heltall og m er størrelsen på hashtabellen.

5 % **4** Hvilket problem løser denne prosedyren?

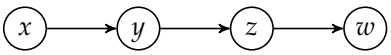
```
1 while x.left ≠ NIL
2       x = x.left
3 return x
```

Dette er altså en prosedyre fra pensum. Vi er ute etter funksjonaliteten – hva den brukes til – og ikke bare hvordan den oppfører seg.

5 % **5** Hvordan kan du gjenskape funksjonaliteten til en stakk (*stack*) ved hjelp av en makshaug (*max-heap*)?

Baser deg på vanlige haugoperasjoner, uten å tenke på hvordan haugen faktisk er implementert. Fokuser på hvordan prioritetene/nøklene skal velges.

5 % **6** Utfør DFS på grafen nedenfor, slik at første kall til DFS-VISIT starter i node y. Hva blir $v.d$ og $v.f$ (*discovery time* og *finish time*) for hver node v?



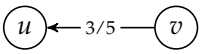
Oppgi svaret som to lister med 4 tall på hver sin linje, som f.eks.:

1, 2, 4, 7
5, 6, 3, 8

Første linje er $x.d$, $y.d$, $z.d$, $w.d$, i rekkefølge, og andre linje er $x.f$, $y.f$, $z.f$, $w.f$.

Hint: Husk at DFS starter en ny dybde-først-traversering med DFS-VISIT fra hver eneste node som ikke allerede er besøkt.

5 % **7** Hva er restkapasiteten (*residual capacity*) fra u til v her?



Det er altså snakk om kapasiteten i restnettet, $c_f(u, v)$.

5 % **8** Du har oppgitt følgende frekvenser for tegnene `a`, `b`, `c` og `d`:

$$\texttt{a}.freq = 8 \qquad \texttt{b}.freq = 4 \qquad \texttt{c}.freq = 2 \qquad \texttt{d}.freq = 4$$

Hvilke av trærne 1–5 i figur 1 er korrekte huffmantrær for disse tegnene, med disse frekvensene? Forklar kort.

Oppgi svaret som en liste av tall, som f.eks.:

1, 2, 3, 4, 5

Med «huffmantre» menes et tre som kan konstrueres av Huffmans algoritme, hvis det er tilfeldig om en barnenode havner til venstre eller høyre. Det er minst ett korrekt huffmantre i figur 1, men det kan være flere.

5 % **9** Hva er kjøretiden til følgende prosedyre, som funksjon av n?

```
1  for i = 1 to Ω(n)
2      for j = 1 to O(n)
3          print "Hello, world!"
```

De asymptotiske operatorene representerer her ukjente men reelle funksjoner, på samme måte som når de f.eks. brukes i aritmetiske uttrykk.

5 % **10** De følgende to trinnene utgjør tilsammen en sammenligningsbasert sorteringsalgoritme (*comparison sort*), som sorterer en tabell A med n elementer:

1 INIT(A, n)
2 MAIN(A, n)

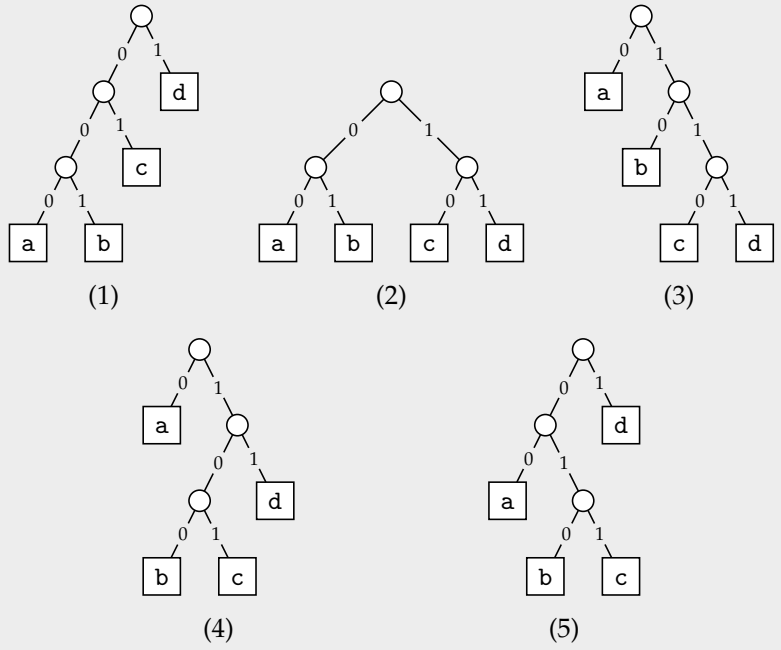
Kjøretiden til MAIN er $O(n)$. Hva er kjøretiden til INIT i verste tilfelle?

Om du har en usortert tabell A av lengde n og først kjører INIT(A, n) og deretter MAIN(A, n), vil altså A ende opp sortert, og både INIT og MAIN baserer seg kun på å sammenligne elementene i A.

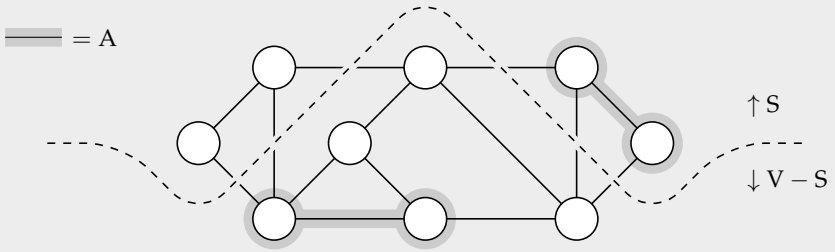
5 % **11** Løs rekurrensen $T(n) = 2T(n/4) + \sqrt{n \lg n}$. Oppgi svaret i Θ-notasjon.

Husk at $\sqrt{x} = x^{1/2}$ og $\sqrt{xy} = \sqrt{x} \cdot \sqrt{y}$.

Figur 1



Figur 2



5 % **12** Du bygger et minimalt spenntre gradvis, ved å legge til kanter i mengden A. Du vurderer å legge til den letteste kanten som krysser snittet $(S, V - S)$.

Hvilket krav stiller vi normalt til forholdet mellom nåværende A og dette snittet for at det skal være trygt å legge til denne kanten?

Med «normalt» menes her altså strategien som beskrives i pensum ifm. GENERIC-MST, og som bl.a. MST-KRUSKAL og MST-PRIM baserer seg på.

En illustrasjon av hvordan den foreløpige kantmengden A og snittet (*cut*) mellom S og $(V - S)$ kan se ut finner du i figur 2.

5 % **13** Din venn Lurvik har støtt på et problem han er usikker på om kan løses i polynomisk tid. Han har tidligere vist at problemet er i NP, men har nå nettopp klart å redusere det til SUBSET-SUM i polynomisk tid, og spør deg om råd. Hva er

din vurdering av situasjonen? Svar kort.

5 % **14** Vi snakker gjerne om tre ulike typer gjennomsnittlig kjøretid. Hvilke?

Merk at det er snakk om ulike typer *gjennomsnittlig* kjøretid (altså ikke f.eks. beste og verste tilfelle).

Her regnes forventning (*expectation*) som en form for gjennomsnitt. Poenget er å forklare hva vi tar gjennomsnitt av, eller finner forventningsverdien over, for hver av de tre typene, heller enn å oppgi hva de heter.

5 % **15** Under ser du matrisene W og $L^{(2)}$ fra en kjøring av SLOW-APSP.

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

W

	1	2	3	4	5
1	0	1	2	4	6
2	∞	0	1	2	4
3	∞	∞	0	1	2
4	∞	∞	∞	0	1
5	∞	∞	∞	∞	0

$L^{(2)}$

Hva blir $l_{1,5}^{(3)}$? Forklar svært kort.

Du skal altså i praksis utføre EXTEND-SHORTEST-PATHS én gang.

Her angir $L^{(r)}$ lengdene til de korteste stiene som inneholder maksimalt r kanter, der $l_{i,j}^{(r)}$ er lengden fra node i til node j, altså cellen i rad i og kolonne j.

En enkel tekstlig forklaring er tilstrekkelig. Du trenger ikke bruke matematisk notasjon.

5 % **16** Tabellen $A = \langle a_1, a_2, \ldots, a_n \rangle$ inneholder nøklene fra et komplett (dvs., perfekt balansert) binært søketre uten duplikater. Nøklene er hentet ut fra venstre mot høyre, nivå for nivå fra bunnen, som illustrert i figur 3 (for $n = 15$).

Du skal sortere A ved å utføre alle følgende kall i en eller annen rekkefølge:

INSERTION-SORT(A, n), MERGE-SORT$(A, 1, n)$, QUICKSORT$(A, 1, n)$, REVERSE(A)

Anta at PARTITION er modifisert så den bevarer rekkefølgen på elementene som havner i de to halvdelene, med samme kjøretid som den har originalt. (Dette kan gjøres f.eks. ved hjelp av en lenket liste.)

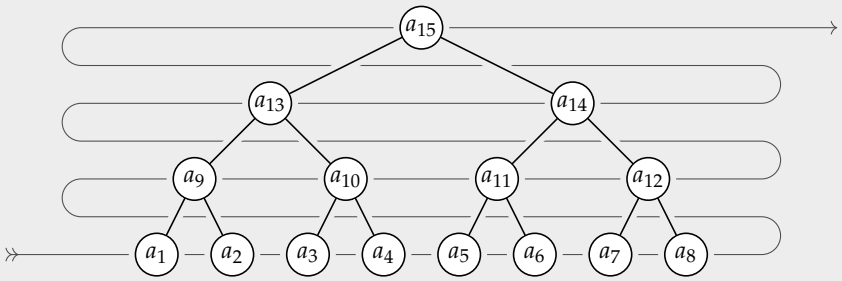
I hvilken rekkefølge vil du utføre prosedyrene? Forklar.

Oppgi svaret ved å liste opp navnene i riktig rekkefølge, som f.eks.:

INSERTION-SORT, MERGE-SORT, QUICKSORT, REVERSE

Prosedyrene utføres etter hverandre, så A endres for hvert kall, og skal ende opp sortert til slutt.

Figur 3



Målet er at hver av prosedyrene *individuelt* skal få så lav asymptotisk kjøretid som mulig. Du bør altså *ikke* bare se på den *totale* asymptotiske kjøretiden.

Merk: Her brukes den deterministiske prosedyren QUICKSORT, som velger siste element som pivot, og ikke RANDOMIZED-QUICKSORT, som velger tilfeldig.

REVERSE reverserer tabellen i lineær tid, så første element blir sist, etc.

5 % **17** Din venn Klokland vil forbedre BELLMAN-FORD ved å holde styr på hvilke avstandsestimater som faktisk endrer seg. Hun vil gjøre dette med en eller annen slags kø, som til å begynne med bare inneholder startnoden. Hun tenker også det er lurt å fokusere på lave estimater, og bruker derfor en prioritetskø.

I hver iterasjon henter hun en node u fra køen og oppdaterer estimatet langs alle kanter (u, v). Hvis $v.d$ endres, og v ikke ligger i køen, legges den inn.

Hva er kjøretiden hvis grafen inneholder negative sykler og hva er den hvis grafen ikke har negative kantvekter? Forklar kort.

Du kan anta (1) at hun bruker en binær min-haug (*binary min-heap*) som prioritetskø, med $v.d$ som prioritet for hver node v, (2) at det tar konstant tid å sjekke om en node ligger i køen og (3) at alle noder kan nås fra startnoden.

For full pseudokode, se algoritme 1, men merk at det er fullt mulig å besvare oppgaven uten å lese eller forstå pseudokoden.

5 % **18** Hvordan vil du modifisere FLOYD-WARSHALL for å finne de m korteste stiene mellom alle par med noder, heller enn bare den korteste? Du kan anta at m er en liten konstant.

Det holder at du finner *lengdene* til de m korteste stiene. Du trenger altså ikke finne de faktiske stiene.

Merk: Her er det *ikke* snakk om å finne m stier fra u til v, der alle har minimal lengde, og følgelig er like lange! Om man sorterer stiene fra u til v etter lengde, er vi ute etter de m første.

5 % **19** Du har oppgitt et flytnett (*flow network*) $G = (V, E)$, med følgende endringer:

Algoritme 1

```
1  for each vertex u ∈ G.V − {s}
2      u.d = ∞
3  s.d = 0
4  Q = ∅
5  INSERT(Q, s)
6  while Q ≠ ∅
7      u = EXTRACT-MIN(Q)
8      for each vertex v in G.Adj[u]
9          if v.d > u.d + w(u, v)                  // tilsvarer RELAX
10             v.d = u.d + w(u, v)
11             if v is in Q                        // tar konstant tid
12                 DECREASE-KEY(Q, v, v.d)
13             else INSERT(Q, v)
```

- Kilde (*source*) s og sluk (*sink*) t er fjernet.
- I tillegg til kapasitet, har hver kant (u, v) fått en *nedre grense* $b(u, v)$.

I stedet for $f(u, v) \geqslant 0$, krever vi nå $f(u, v) \geqslant b(u, v)$ for alle $u, v \in \mathrm{V}$.

Som for kapasiteter, sier vi at $b(u, v) = 0$ hvis $(u, v) \notin \mathrm{E}$.

Målet er ikke å maksimere noe, men å finne en hvilken som helst flyt som tilfredsstiller både kapasitetene og de nedre grensene, der flyten inn er lik flyten ut i alle noder – en såkalt *sirkulasjon*.

Hvordan kan du redusere dette til et vanlig flytproblem? Forklar kort.

Hint: Hvis $f'(u, v) = f(u, v) - b(u, v)$ har vi $f'(u, v) \geqslant 0$ og, for alle $u \in \mathrm{V}$,

$$\sum_{v \in \mathrm{V}} f'(u, v) + \sum_{v \in \mathrm{V}} b(u, v) = \sum_{v \in \mathrm{V}} f'(v, u) + \sum_{v \in \mathrm{V}} b(v, u)\,.$$

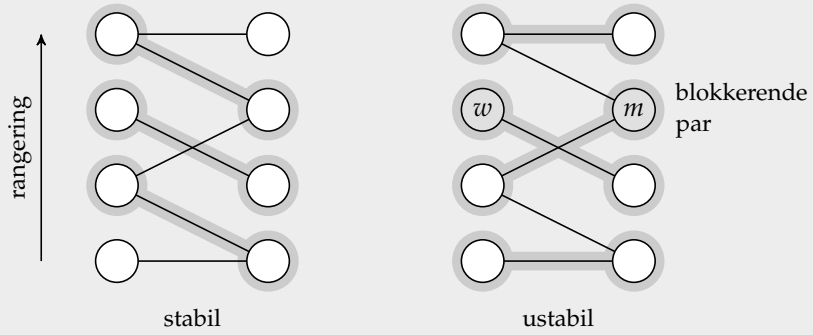
Merk: Det er ikke sikkert at en gyldig sirkulasjon eksisterer!

5 % **20** Din venn Smartnes jobber med å matche n organdonorer og n resipienter, med prioriterte ventelister. Han modellerer dette som en modifisert utgave av stabil matching (*the stable marriage problem*), med donorer som «kvinner» og resipienter som «menn»:

- Ventelistene implementeres ved at alle har like preferanser. Det vil si, alle kvinner rangerer mennene likt, og alle menn rangerer kvinnene likt.
- Ikke alle er kompatible, så den bipartitte grafen er ikke nødvendigvis komplett. Du kan altså ikke alltid matche enhver kvinne med enhver mann.
- Som i vanlig bipartitt matching (*maximum bipartite matching*) er målet å matche flest mulig.

Smartnes er ikke sikker på om det helt gir mening, men han beholder det opprinnelige kravet til stabilitet:

Figur 4



- Du kan ikke ha en kvinne og en mann som heller vil ha hverandre enn partnerne sine (et såkalt *blokkerende par*, som w og m i figur 4).

Han har prøvd å løse problemet ved å kombinere Ford-Fulkerson (for *størst mulig* matching) og Gale-Shapley (for *stabil* matching), men har måttet gi opp. Nå lurer han på om du har noen ideer.

Hvordan ville du ha løst problemet til Smartnes? Forklar kort.

Dersom det er enklere, så holder det at du finner *størrelsen* på den største stabile matchingen, uten å faktisk finne selve matchingen.

Merk: Under Smartnes sin definisjon, kan en kvinne og mann altså utgjøre et blokkerende par selv om de ikke er kompatible med hverandre!

Hint: På grunn av stabilitetskravet, vil matchede kvinner og menn rangeres i samme rekkefølge. Den første matchede kvinnen har den første matchede mannen som partner, etc.