Avsluttende eksamen i TDT4125
Algoritmekonstruksjon, videregående kurs (løsningsforslag)

Kontakt under eksamen Magnus Lie Hetland
Tillatte hjelpemidler Alle trykte/håndskrevne; bestemt, enkel kalkulator

Bruk gjerne blyant! Les hele oppgavesettet først, disponer tiden og forbered evt. spørsmål til faglærer kommer til eksamenslokalet. Gjør antagelser der det er nødvendig. Svar konsist, fortrinnsvis i svarskjemaet.

Svarskjema

1a (13 %)
1b (8 %)
2a (11 %)
2b (8 %)
2c (8 %)
3a (9 %)

3b (12 %)

3c (10 %)

4a (12 %)

4b (9 %)

Oppgave 1 (Parallellitet)

Neste to deloppgaver er tatt fra http://faculty.kfupm.edu.sa/ics/malalla/Teaching/ICS556/Exams/FinalExam-556.pdf

La A være en Boolsk tabell (*array*) av lengde n, og la ☆ være en hvilken som helst binær logisk operator.

a. Beskriv en algoritme med kjøretid $\Theta(\lg n)$ som beregner $A_1 \star A_2 \star \ldots \star A_n$ på en EREW PRAM-maskin.

Svar: F.eks. lag et binærtre av prosessorer, der hver utfører én ☆-operasjon. (Kan evt. simuleres med n prosessorer.)

b. Beskriv en annen EREW-algoritme for samme problem som er så effektiv som mulig, der produkt av kjøretid og antall prosessorer er forventet å være $\Theta(n)$.

(Merk: Det er altså snakk om en algoritme som er *work-efficient*.)

Svar: Bruk RANDOMIZED-LIST-PREFIX. Løsninger med kjøretid $\Theta(n)$ gir ingen uttelling (er ikke så effektiv som mulig).

Oppgave 2 (Kompleksitet)

En binærhaug (*binary heap*) er gjerne representert som en tabell (*array*), og har dermed en fast kapasitet.

a. Hvordan kan man øke kapasiteten til en slik haug uten at den amortiserte kjøretiden til de ulike operasjonene påvirkes? Begrunn svaret.

Svar: Bruk tabelldobling. Når haugen er full kopieres alt over i en ny tabell som er dobbelt så stor. Amortisert kostnad for dette er konstant (kjent fra pensum).

Betrakt problemet med å finne billigst maks-flyt (*min-cost max-flow*) fra kilde s til sluk t i en graf der alle kantene har en kapasitet på 1. Grafen har n noder og m kanter og alle nodene kan nås fra s. Anta at ingen kantkostnader er negative.

b. Gi en tett øvre grense (O-notasjon) for kjøretiden (som funksjon av n og m) for Busacker-Gowen på dette problemet, hvis du bruker prising (*pricing*) og Dijkstras algoritme med en binærheap (*binary heap*) for å finne de flytforøkende stiene (*augmenting paths*).

Svar: $O(nm \lg n)$.

Forklaring (kreves ikke av studentene)**:** Flyten kan maksimalt være $O(n)$ (f.eks. antall kanter ut av s). Hver forøkende sti øker denne flyten med 1, opp fra 0, og med DIJKSTRA tar det $O(m \lg n)$ å finne denne stien. Her vil $O(m^2 \lg n)$ gi ca. halv uttelling (begrenser bare flyten til $O(m)$; riktig, men ikke stram nok grense).

La BIPARTITE være problemet å avgjøre hvorvidt en graf er bipartitt.

c. Vis at BIPARTITE $\in$ NL.

Svar: Traverser i en odde sykel, om mulig (svaret er da *nei*; ellers *ja*). Husk bare nåværende posisjon ($\lg n$) og velg riktig startnode og ut-kant ikke-deterministisk. Her kan man også verifisere et sertifikat/en løsning med logaritmisk plassbruk i tillegg til løsningen. Merk at å referere til en node eller en kant krever logaritmisk lagringsplass. Selv om det dermed ikke er korrekt å si at verifiseringen krever konstant plass har det i denne sensuren ikke blitt trukket for dette.

Oppgave 3 (Approksimering)

Basert på oppgave fra MIT, Introduction to Algorithms, 18. mai, 2003

Anta at du skal flytte eiendeler med størrelser $x_1, x_2, \ldots, x_n$ fra sted A til sted B. Størrelsene er reelle tall mellom 0 og 1 ($0 \leq x_i \leq 1$, for alle i), og eiendelene skal flyttes i en bil med lastekapasitet 1. Det gjelder å lage en plan for alle flyttelassene slik at det blir så få kjøreturer (lass) som mulig. Merk at vi skal lage en fullstendig plan før noen kjøringer foretas. Dette problemet er NP-hardt, og vi skal vurdere følgende enkle *first-fit*–approksimeringsalgoritme:

Putt enhet x_1 på det første lasset og deretter, for $i = 2, 3, \ldots, n$, putt x_i i det første av de planlagte lassene der det er stor nok kapasitet.

a. Gi et eksempel med $n = 4$ som viser at first-fit–algoritmen ikke er optimal.

Svar: F.eks. [0.3, 0.8, 0.2, 0.7].

b. Vis at first-fit–algoritmen er en 2-tilnærming (2-*approximation*).

(I begrunnelsen kan du bruke m for antall lass i first-fit-løsningen og m^* for det optimale antallet lass. Du kan anta at det er snakk om minst to eiendeler.)

Svar: Maks ett lass L kan være under halvfullt, og snittet av L og ethvert annet lass (og dermed snittet av alle) vil være mer enn halvfullt – ellers ville L ha vært kombinert med et av dem.

Vi har nå $m^* \geq \sum x_i > 0.5m$, så $m/m^* < 2$.

c. Hvis first-fit–algoritmen har en kjøretid på $f(n)$, beskriv en randomisert algoritme med en parameter k, som har kjøretid $k \cdot f(n)$, som alltid gir minst like godt svar som first-fit–algoritmen, som alltid har mulighet til å finne den optimale løsningen, og der større verdier for k gir en høyere forventet kvalitet.

Svar: Kjør first-fit–algoritmen k ganger. Randomiser rekkefølgen mellom hver kjøring. (Det er viktig å kjøre normal first-fit først, for å garantere minst like godt svar. Om man randomiserer allerede fra starten vil dette gi noe trekk.)

Oppgave 4 (Kombinatorisk optimering)

Basert på §5.2 i *Integer Programming*, av Laurence A. Wolsey

Betrakt følgende problem **P**: Du skal planlegge produksjon av et produkt for n måneder fremover. For hver måned $i = 1 \ldots n$ har du oppgitt følgende:

p_i	Produksjonskostnad per enhet
h_i	Lagringskostnad per enhet
d_i	Etterspørsel (antall enheter)

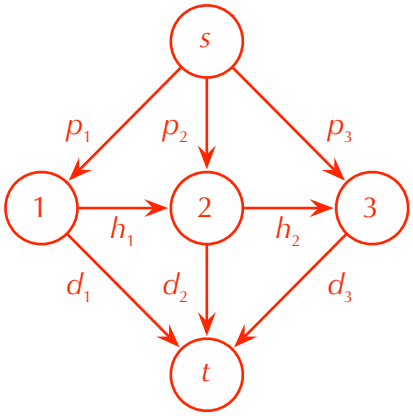
La x_i være antall enheter som produseres i måned i, mens o_i er antall enheter som til nå (ved slutten av måned i) er produsert, men foreløpig ikke solgt. Merk at x_i, o_i og d_i er et ikke-negative heltall for $i = 1 \ldots n$, mens kostnadene er positive rasjonale tall. Målet er å finne en produksjonsplan ($x_1 \ldots x_n$) som tilfredsstiller all etterspørsel, men med lavest mulig totalkostnad. Lagringskostnaden (h_i) betales per enhet på lager ved slutten av måned i (altså o_i). Etter siste måned (n) skal lageret være tømt.

Anta at du har en algoritme **A** tilgjengelig, som kan løse min-kost–max-flyt–problemet (*min-cost max-flow*).

a. Tegn **P** (for $n = 3$) representert som et nettverksproblem som kan løses av **A**. Forklar figuren.

Svar: Kantene har påskrevet kapasitet (d) eller kostnad (p, h). De nederste kantene har kostnad 0; de resterende har ubegrenset kapasitet. Planen ($x_1 \ldots x_3$) er gitt av flyten i de øverste

kantene. (Andre, ekvivalente, nettverk gir naturligvis også full uttelling, så lenge de bare er en konstant faktor større.)



Betrakt nå problem **Q**, som er likt **P**, bortsett fra følgende ekstrakostnad:

f_i Fast kostnad (oppstartskostnad) for produksjon

Denne faste kostnaden betales kun dersom det produseres minst én enhet i måned i. (Merk at dette er en engangskostnad for måned i, og er *ikke* en kostnad per enhet, i motsetning til p_i.) Hvis $f_i = 0$ for $i = 1 \ldots n$ så er **Q** = **P**.

b. Løs problemet **Q** så effektivt som mulig ved hjelp av dynamisk programmering. Du trenger kun finne kostnaden til den optimale planen (dvs., det er ikke nødvendig å finne selve planen).

Merk: En løsning med kjøretid $\Theta(n^2)$ eller $\Theta(n^3)$ gir full uttelling. En kjøretid på $\Theta(n^4)$ gir noe trekk.

Hint: Du kan anta at produksjonen i en måned (hvis den er større enn null) fullstendig dekker behovet et antall påfølgende måneder. Det vil si, hvis $x_i > 0$, så kan du anta at $x_i = d_i + d_{i+1} + \ldots + d_{i+k}$, for en eller annen $k \geq 0$. Du kan også anta at $x_i = 0$ hvis $o_i > 0$. En optimal løsning med disse egenskapene vil alltid eksistere.

Svar: Hovedpoenget er å fylle ut kostnaden $C[k]$ for $k = 1 \ldots n$, der man prøver ut alle mulige siste produksjonsmåneder k. En rett frem løsning gir fire løkker (og total kjøretid $\Theta(n^4)$), der den innerste summerer lagringskostnader for det gjeldende intervallet. (Det er viktig å få med de varierende lagringskostnadene.) Disse kan forhåndsberegnes i kvadratisk tid, noe som gir en kubisk algoritme. Det optimale er en kvadratisk algoritme – her er en måte å få til det på (svaret finnes til slutt i $C[n]$):

```
for k = 1..n:                   # Alle delproblemer
    C[k] = ∞                    # Foreløpig beste C[k]
    m = 0                       # Ant. enheter som må produseres i måned j
    l = 0                       # Lagringskostnad for måned j .. k
    for j = k..1:               # Prøv ulike j som siste produksjonsmåned
        l += m * h[j]           # Lagre alle de resterende også ut måned j
        m += d[j]               # Ekstra produksjon for å dekke behovet
        c = f[j] + m*p[j] + l   # Oppstart, produser alt, lagre det usolgte
        C[k] = min(c, C[k])     # Oppdater C[k] om nødvendig
```