Avsluttende eksamen i TDT4125
Algoritmekonstruksjon, videregående kurs (Bokmål)

Kontakt under eksamen Magnus Lie Hetland (tlf. 91851949)
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 (9%)
1b (9%)
1c (9%)
2a (9%)
2b (9%)

3a (9%)

3b (9%)

4a (9%)

4b (9%)

5a (9%)

5b (10%)

Merk: Svarene i løsningsforslaget er til tider mer utførlige enn det som kreves av studentene.

Oppgave 1

a. Du ønsker å avgjøre hvorvidt to logiske formler kan ha samme verdi under de samme omstendighetene (dvs., evt. felles variable har samme verdi). Vis kort at dette er et NP-komplett problem.

Svar: Reduser fra SAT ved å sette den ene formelen til *true* (dvs. en konstant).

b. Du ønsker nå å avgjøre hvorvidt to logiske formler alltid vil ha samme sannhetsverdi under samme omstendigheter (samme verdier for felles variable). Vis kort at dette er et co-NP-komplett problem.

Svar: Et problem er co-NP-komplett dersom komplementet av problemet er NP-komplett. Med andre ord kan vi vise at problemet med å vise at to formler *ikke* alltid har samme sannhetsverdi er NP-komplett. Her kan vi igjen gjøre samme reduksjon som i **a**, men la den andre formelen være *false*.

Følgende (formodentlig gale) utsagn er tatt fra Wikipedia:[1]

> A problem is said to be strongly NP-hard if a strongly NP-complete problem has a polynomial reduction to it.[2]

(Merk at sterk NP-kompletthet er et spesialtilfelle av NP-kompletthet.)

c. Vis at utsagnet implisitt besvarer spørsmålet om hvorvidt P = NP.

Svar: Et sterkt NP-komplett problem er i NP og kan dermed reduseres til ethvert NP-komplett problem. Disse vil da (i følge utsagnet) alle være sterkt NP-harde. Samtidig finnes det pseudopolynomiske løsninger for flere av disse (f.eks. SUBSET-SUM), og det kan ikke finnes en pseudopolynomisk løsning for et sterkt NP-hardt problem med mindre P = NP.

Oppgave 2

a. Beskriv hvordan Floyd-Warshall kan tilpasses en PRAM-maskin med n^2 prosessorer slik at du får en kjøretid på $\Theta(n)$, der n er antall noder. Hvilken PRAM-modell trenger du?

(Du kan gjenbruker/skrive over avstandsmatrisen, så du unngår kubisk allokeringstid.)

Svar: Hver prosessor representerer én rute i tabellen for en gitt k. Hver av disse trenger kun å lese 3 verdier fra forrige iterasjon og skrive resultatet i sin egen rute. Det er n iterasjoner.

Hvert element leses av mange prosessorer (ikke et konstant antall), men hver celle skrive stil av kun én prosessor, så vi trenger CREW. (CRCW vil også fungere, men er strengere enn nødvendig.)

Kjernen i Floyd-Warshall kan uttrykkes rekursivt som følger:

D(u, v, k):
if $k = 0$
    return $W[u, v]$
return min { D(u, v, k–1), D(u, k, k–1) + D(k, v, k–1) }

---

[1] http://en.wikipedia.org/wiki/Strongly_NP-complete

[2] På norsk: «Et problem sies å være sterkt NP-hardt dersom et sterkt NP-komplett problem har en polynomisk reduksjon til det.»

b. Hvis du antar at hvert rekursive kall utføres i parallell (med **spawn**) Hva er parallellitetsgraden (*parallelism*) til prosedyren D, som funksjon av k? Vis kort utregning.

Svar: Her må man løse to rekurrenser, én for arbeidet (*work*) og én for spennet (*span*). Arbeidet kan uttrykkes slik:

$$T(k) = 3T(k-1) + 1;\ T(0) = 1$$

Dette gir T(k) = $\Theta(3^k)$ (mer spesifikt, $T(k) = (3^{k+1} - 1)/2$); det kan regnes ut med standardmetoder for rekurrensregning. (Om man setter T(1) = 1, f.eks., får man bare andre konstantledd.)

Den andre rekurrensen er slik:

$$T(k) = T(k-1) + 1 \text{ som er } \Theta(k).$$

Parallellitetsgraden er altså $\Theta(3^k/k)$.

Oppgave 3

Anta at du har et sett med n punkter i planet. Hvert punkt har én av k farger (at alle fargene er representert. Du ønsker å velge ut k punkter, ett av hver farge, slik at omkretsen av det konvekse skallet (*convex hull*) rundt de k punktene blir så liten som mulig.

Betrakt følgende grådige approksimeringsalgoritme for dette problemet: Velg et punkt p, og for hver farge c forskjellig fra p sin, velg et punkt av farge c som er nærmest p. Gjenta denne prosedyren for ethvert punkt p, og velg til slutt den av de n punktmengdene som har et konvekst skall med minst diameter. (Diameteren er avstanden mellom de to punktene som er lengst fra hverandre.)

a. Vis at denne algoritmen har en approksimerings-ratio (*approximation ratio*) på $\pi/2$.

Hint: Anta at det finnes en gyldig løsning med diameter d. Hva kan du da si om diameteren til den optimale løsningen? Hva med den grådige?

Merknad: Denne oppgaven er galt stilt (algoritmen har ikke en approximation ratio på $\pi/2$). Om studenten har argumentert for dette, eller gitt et moteksempel som viser at $\pi/2$ ikke er riktig gir det full uttelling. Dersom det er til studentens fordel tas oppgaven ut av sensur, og den resterende vektingen justeres tilsvarende.

Det kan vises at den har en ratio på π som følger: (1) Anta at diameteren på den optimale løsningen er d. Den største avstanden mellom punktene i denne mengden er da altså d. (2) Når den grådige algoritmen velger et av punktene i den optimale løsningen som utgangspunkt vil den ikke plukke noen punkter med større avstand enn d. (3) For en vilkårlig mengde der største avstand mellom punkter er r vil ikke diameteren kunne overskride $2r$. Av punktene 1, 2 og 3 kan vi slutte at den grådige algoritmen vil plukke en mengde med diameter mindre enn eller lik $2d$, og vi får en ratio på π etter samme argument som i deloppgave **b**.

Betrakt nå en billigere variant av den grådige algoritmen. I stedet for å velge løsningen med minst diameter, velg løsningen med minst omkrets (perimeter).

b. Vis at denne algoritmen har en approksimerings-ratio på π.

Hint: Anta at den optimale løsningen har diameter d. Da finnes det et punkt p som har punkter av alle de andre fargene innen en avstand på d. Dette punktet vil naturligvis vurderes av den grådige algoritmen.

Svar: Anta at den optimale løsningen har diameter d og dermed en omkrets på minst $2d$. Da vil den grådige algoritmen finne minst ett punktsett med en omkrets på maksimalt $2\pi d$ (som

gir oss ratioen vi ønsker). Denne mengden kan vi konstruere som følger: La p være ett av punktene i den optimale løsningen. Da vet vi at alle de andre punktene har en maks-avstand på d til p. Det verste tilfellet for den grådige algoritmen i dette punktet er om alle punktene har avstand d, og er spredd utover i en sirkel rundt p, med diameter $2d$.

Merknad: En observasjon om at denne løsningen ikke kan være dårligere enn løsningen i deloppgave **a** vil gi uttelling. Om man i **a** har argumentert for en ratio på π vil en referanse til det delsvaret her gi full uttelling.

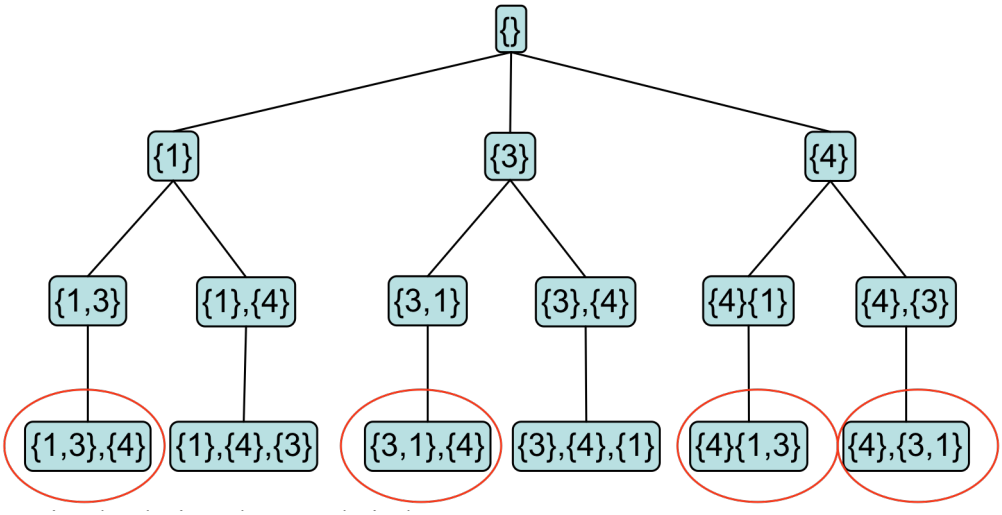
Oppgave 4

I *bin-packing*-problemet har du en mengde objekter av ulike størrelser som skal plasseres i så få båser (*bins*) som mulig, der båsene har fast størrelse. Mer spesifikt, gitt en mengde av elementer med størrelser $S = \{s_1, \ldots, s_n\}$ og en bås-størrelse l ($l \geq s_i$ for alle s_i i S) er målet å finne en B-partisjonen $S_1 \cup \ldots \cup S_B$ av $\{1, \ldots, n\}$ som er slik at $\sum_{i \in Sk} s_i \leq l$ for alle $k = 1, \ldots, B$ and slik at B er minimal. For eksempel, for $S = \{1, 3, 4, 2, 2\}$ and $l = 6$ vil en slik optimal B-partition være $\{1, 2, 4\} \cup \{3, 5\}$; det vil si, vi plasserer 1,3, and 2 i én bås og 4 and 2 i en annen bås.

a. Gitt S = {1, 3, 4} and $l = 4$, tegn et søketre som representerer de mulige løsningene til *bin-packing*-problemet. Hvor mange optimale løsninger inneholder søketreet ditt?

Svar: Se figur nedenfor for én mulighet. Her er alle permutasjonre av 1, 3 og 4 undersøkt, og etterfølgende verdier som kan være i samme bås er slått sammen. Dette eliminerer en del suboptimale løsninger man kan få om man prøver alle partisjoner, men et slikt mer ekstensivt tre gir også uttelling. (Det er også mulig å ha et mye mer konsist tre; her har flere løsninger blitt godkjent.) Om man filtrerer ut mange løsninger uten å forklare hvordan så kan det trekke ned.

De optimale løsningene er markert med røde ovaler. (Her er løsningene oppgitt ved selve verdiene; det er også OK å bruke indeksene 1, 2 og 3.)



b. Anta at du har en partiell løsning P, der et element s_i har blitt plassert i en bås. Beskriv en algoritme for å beregne et underestimat $g(P)$ for den beste mulige løsningen du kan få (antall båser), gitt din partielle løsning P. Hva er kjøretiden på algoritmen din?

Svar: En mulig heuristikk: $g(P) = 1 + \text{floor}((\sum_{k \neq i} s_k)/L)$. Kjøretid: $\Theta(n)$.

(Merk at her ble det ikke spurt om et over- eller under-estimat, så det har blitt gitt uttelling for alle rimelige estimater.)

Oppgave 5

Du skal planlegge bruken av piloter for et flyselskap. Du har oppgitt en liste med byer, tiden det tar å fly fra hver by til hver av de andre (der det går fly) og en liste med flyvninger. For hver flyvning vet du byen den starter i, byen den lander i, samt tiden den starter. Du har n piloter som skal fordeles på flyvningene.

a. Beskriv en effektiv algoritme som løser problemet. (Bruk gjerne en figur.)

Svar: Her bruker vi flyt med tilbud/etterspørsel og nedre beskrankninger. (Kan lett justeres for direkte løsning med maks-flyt.)

Vi har en kilde s med tilbud n og et sluk t med etterspørsel n. For hver flyvning i, legg til to noder u_i og v_i, en kant (u_i, v_i) med kapasitet og nedre beskrankning 1, en kant (s, u_i) og en kant (v_i, t), begge med kapasitet 1. Hvis man fra en flyvning i kan nå en flyvning j (i ender i samme by som j starter i, og i lander før j tar av), legg til en kant (v_i, u_j) med kapasitet 1.

En flyt kan inneholde sykler – dvs. vi kan ha en sykel i flytnettverket der alle kantene har tilordnet flyt større enn 0. I mange anvendelser vil dette representere bortkastet arbeid, der vi transporterer en ressurs i ring, heller enn å få den dit den skal. En *effektiv* flyt er en flyt uten sykler.

b. Skisser en algoritme for å finne en effektiv maksimal flyt for et gitt flytnettverk. Forklar kort hvorfor algoritmen er korrekt.

Svar: Her kan man bruke for eksempel Busacker-Gowen eller en tilsvarende algoritme for å finne billigst flyt (*min-cost flow*). Trinn 1: Finn maksimal flyt. Trinn 2: Finn billigst flyt med med maksimal flytverdi, der hver kant har en kostnad på 1. Det er ikke nok å svare at Busacker-Gowen egner seg til å finne effektiv maks flyt, for eksempel.

Det at vi ikke kan ha sykler vises f.eks. ved selvmotsigelse: Dersom vi hadde en sykel kunne vi redusere flyten i denne sykelen (ved å «dytte» den mot sykelretningen), og dermed få samme flytverdi, men lavere kostnad.

Det holder ikke å skrive at man først kan finne maks-flyt og så fjerne alle sykler ved å senke flyten i dem med flaskehals-flyten. Grunnen til det er at det er NP-hardt å finne alle sykler i en graf. (Mengden av alle sykler kan brukes til å finne f.eks. en Hamilton-sykel.) En slik løsning vil likevel kunne gi noe uttelling. (Det å sette flyten eller kapasiteten til 0, uansett flaskehals, er galt, og gir ingen uttelling.)