Avsluttende eksamen i TDT4120 Algoritmer og datastrukturer

Løsningsforslag

Eksamensdato	30. november 2009
Eksamenstid	0900–1300
Sensurdato	21. desember
Språk/målform	Bokmål
Kontakt under eksamen	Magnus Lie Hetland (tlf. 91851949)
Tillatte hjelpemidler	Ingen trykte/håndskrevne; bestemt, enkel kalkulator

Vennligst les hele oppgavesettet før du begynner, disponer tiden og forbered evt. spørsmål til faglærer kommer til eksamenslokalet. Gjør antagelser der det er nødvendig. Skriv kort og konsist på angitt sted. Lange forklaringer og utledninger som ikke direkte besvarer oppgaven tillegges liten eller ingen vekt.

Algoritmer kan beskrives med tekst, pseudokode eller programkode, etter eget ønske, så lenge det klart fremgår hvordan den beskrevne algoritmen fungerer. Korte, abstrakte forklaringer kan være vel så gode som utførlig pseudokode, så lenge de er presise nok. Kjøretider oppgis med asymptotisk notasjon, så presist som mulig.

Merk: Flere av forklaringene er her mer utførlige enn det som kreves i en besvarelse.

Oppgave 1 (56%)

a) Hva er forskjellen på et *problem* og en *probleminstans*?

Svar (7%): Et problem er en relasjon mellom input og output, mens en probleminstans er én bestemt input. Andre lignende svar som får frem det viktige skillet mellom det generelle problemet og den mer spesifikke instansen vil gi uttelling.

b) Hvilket krav stiller vi normalt til input til bøttesortering (*bucket sort*)?

Svar (7%): Verdiene må være uniformt tilfeldig fordelte (gjerne i intervallet [0,1)). Svar som «tilfeldig» eller «helt tilfeldig» (altså uten «uniformt») vil gi noe uttelling.

c) Hvis du i et flytnettverk har en kant fra u til v med kapasitet 7 og flyt 3, hvor mye kan du øke flyten med fra v til u? Hva vil flyten fra v til u være da?

Svar (7%): Pga. skifte av lærebok, og endring i fremstilling av flytbegrensningene, vil det her aksepteres flere ulike svar:

Basert på forrige utgave av læreboka:

- Den kan økes fra –3 til 0
- Det kommer an på om det går en kant fra v til u, og hva kapasiteten er

Basert på nåværende utgave av læreboka:

- Det kan ikke gå noen flyt fra v til u
- Man kan øke den fra 0 til 3

De siste to refererer hhv til den opprinnelige grafen og residualnettverket.

d) Å finne korteste vei (*simple, weighted path*) i en vektet graf med negative sykler er NP-hard. Hvordan håndterer BELLMAN–FORD dette?

Svar (7%): Den gir opp hvis den finner en vei kortere enn en som er garantert korrekt.

Her får man også uttelling for bare å si at algoritmen gir opp hvis den oppdager negative sykler, eller noe tilsvarende.

e) Om du står overfor et ukjent problem A i NP, hvordan vil du vise at det er NP-komplett?

Svar (7%): Reduser et kjent NPC-problem til A (i polynomisk tid).

f) Er det mulig å bevise at *noen* NP-komplette problemer kun kan løses av algoritmer med eksponentiell kjøretid uten at man avgjør hvorvidt de andre kan løses i polynomisk tid? Begrunn svaret kort.

Svar (7%): Nei: NPC-problemer kan reduseres til hverandre i polynomisk tid.

Evt.: Hvis ett NPC-problem er i P så vil P = NP. Andre tilsvarende svar gir full uttelling.

g) Gi en nedre og øvre grense for T_P ut fra *work, span* og P på en ideell parallell maskin med grådig arbeidsfordeling (*scheduling*).

Svar (7%): $\max\{ T_1/P, T_\infty \} \leq T_P \leq T_1/P + T_\infty$

Her vil man også få noe uttelling for å ha oppgitt bare T_1/P eller T_∞ som nedre grense. T_1 vil også gi noe uttelling som øvre grense.

Et k-regulært tre er et tre der alle interne noder har k barn. Det trenger ikke være balansert. Du kan anta at treet har minst én intern node.

h) Hvor mange løvnoder har et k-regulært tre med n interne noder? Sett opp en rekurrensligning og løs den. Oppgi svaret eksakt (det vil si, uten asymptotisk notasjon eller ukjente konstanter), som funksjon av k og n. Vis utregningen.

Svar (7%): Her kan også kortere utregninger godtas.

$L(n) = L(n-1) + k - 1 = (L(n-2) + k - 1) + k - 1 = L(n-2) + 2k - 2 =$
$L(n-i) + ik - i = L(n-(n-1)) + (n-1)k - (n-1) = L(1) + nk - k - n + 1 =$
$k + nk - k - n + 1 = (k-1)n + 1$

Oppgave 2 (20%)

Anta at du har en mengde **U** med bilder og en funksjon $s(\cdot, \cdot)$ som beskriver hvor like to bilder er, i form av et reelt tall. Du kan anta at $s(x, y) = s(y, x)$. Du ønsker nå å utføre en enkel «clustering»: Du vil dele **U** i to delmengder **X** og **Y**, slik at summen av likhetsverdier fra alle objekter i **X** til alle i **Y** blir så liten som mulig. Verken **X** eller **Y** skal være tom.

a) Vis at problemet kan løses polynomisk, *eller* vis at problemet er NP-hard.

Svar (6%): **Merk**: *Denne oppgaven var ment å løses med* positive *likhetsverdier, men dette ble ikke spesifisert i oppgaven. Oppgaven slik den står formulert er vurdert som for vanskelig, og tas dermed ut av sensur der det er til fordel for studenten. Hvis man antar positive likhetsverdier vil følgende svar være korrekt.*

Hvis man vet at et par noder *s* og *t* ligger i hhv **X** og **Y** så er dette MIN-CUT-problemet (urettet versjon), som er i P (løses med f.eks. EDMONDS-KARP). Man kan her prøve alle mulige *s* og *t*, og velge den beste løsningen, og fortsatt ha polynomisk kjøretid.

Du ønsker nå å vise frem bildene i et lysbildeshow, slik at to etterfølgende bilder ligner så mye på hverandre som mulig – og for dramatisk effekt vil du ende opp på det første bildet igjen til slutt. Du ønsker altså å maksimere summen av likhetsverdier mellom par med etterfølgende bilder for hele lysbildeshowet, og du vil vise hvert bilde nøyaktig én gang, bortsett fra første/siste bilde, som altså vises to ganger.

b) Beskriv kort en algoritme som løser problemet effektivt og oppgi kjøretiden, *eller* vis at problemet er NP-hard.

Svar (7%): Man prøver å finne en maksimal tur innom alle nodene i en komplett, vektet graf, som tilsvarer TSP med omvendt fortegn på kantvektene.

Du ønsker å lage en nettside med bildene dine, der man kan navigere til «nabobilder». Du ønsker å gjøre dette slik at hvis du kan navigere direkte fra bilde *x* til *y* kan du også navigere direkte fra bilde *y* til *x*. Det skal være mulig å komme seg fra hvilket som helst bilde til et hvilket som helst annet ved å gå fra nabo til nabo, men det skal ikke være mulig å gå i ring. Du vil legge opp navigeringen slik at summen av likhet over alle par med nabobilder blir størst mulig.

c) Beskriv kort en algoritme som løser problemet effektivt og oppgi kjøretiden, *eller* vis at problemet er NP-hard.

Svar (7%): Siden man ikke kan gå i ring, men det skal finnes en sti mellom alle noder, så er det snakk om å finne et spenntre – i dette tilfelle et maksimalt (ekvivalent med minimalt). Man kan altså bruke PRIM eller KRUSKAL (her $O(V^2 \log V)$).

Noen hadde her forstått det som at hver node maksimalt skulle ha to naboer. Hvis man baserer seg på dette, og viser at resultatet blir NP-hard (maksimal Hamilton-sti, nært beslektet med TSP) så får man full uttelling for det. Begrunnelser med lengste vei vil også gi uttelling.

Oppgave 3 (24%)

Anta følgende: For en problemstørrelse på n kan vi behandle tall med $O(\log n)$ siffer i konstant tid, og vi kan bruke dem som indekser i tabeller, med konstant oppslagstid. (Dette er den normale antagelsen i læreboka.)

Anta også at du har en urettet graf med n noder og m kanter, representert ved hjelp av nabolister lagret i en tabell indeksert med nodenummer (standard nabolisterepresentasjon). Du ønsker nå å sortere nabolistene slik at nabolisten for hver node inneholder naboene i stigende rekkefølge.

a) Forklar hvordan du kan sortere alle nabolistene som beskrevet med total kjøretid $O(m+n)$.

Svar (8%): Her er det flere måter å løse problemet på, ved bruk av telling (som i tellesortering). Det enkleste er kanskje å lage en global kantliste, og sortere denne med RADIX-SORT. Vi vil da ha m elementer, med to siffer i verdiområdet $[0, n)$. Nabolistene kan så hentes ut av den sorterte kantlisten i lineær tid. Mer spesialiserte algoritmer godkjennes naturligvis også. (Algoritmer der hver kantliste sorteres for seg vil gi for høy kjøretid, men kan gi noe uttelling.)

Et alternativ er å lage en ny graf. Gå gjennom tabellen – altså fra-noder i sortert rekkefølge. For en gitt fra-node går man gjennom til-nodene. For hver til-node legger man til fra-noden som en til-node i den nye grafen. (Alle kanter vil da referseres. Det kan man korrigere, men siden grafen er urettet spiller det ingen rolle.)

Merk: Det er viktig her at man skiller mellom nabolister og nabomatriser.

Anta at du har en algoritme A som finner en sirkulasjon med minimal kostnad i en rettet graf $G = (V, E, w, b, u)$ med noder V, kanter E, kant-vekter w og nedre og øvre kapasiteter hhv b og u, eller som sier at ingen sirkulasjon eksisterer.

b) Vis kort hvordan du kan bruke A for å avgjøre om det finnes en sti fra en node u til en node v i en urettet graf $G' = (V', E')$.

Svar (8%): **Merk**: *Siden sirkulasjonsproblemet har vært marginalt i årets undervisningsopplegg vil denne oppgaven tas ut av sensur der det er til fordel for kandidaten.*

Lag rettede kanter begge veier for hver kant. Vekt 1, nedre og øvre kap. hhv 0 og 1. Kant fra v til u med nedre kap. 1.

(Forklaring: Vi simulerer på sett og vis en urettet maks-flyt fra u til v med kapasiteter på f.eks. 1. Finner vi en flyt så vil det måtte finnes en sti fra u til v.)

La $T = (V, E)$ være et urettet tre, og la $h(u) = \max \{ d(u, v) \mid v \in V \}$, der $d(u, v)$ er lengden på stien fra u til v, målt i antall kanter. Definer et midtpunkt i T til å være en node $u \in V$ slik at $h(u) = \min \{ h(v) \mid v \in V \}$.

c) Forklar (uten detaljert kode) hvordan du kan finne alle midtpunktene i T ved hjelp av topologisk sortering.

Svar (8%): Tilsvarer oppgave 3 fra eksamen i august, 2005.

Det er altså snakk om å finne den/de noden(e) som ligger mest mulig «sentralt» (det vil være enten én eller to): Den maksimale veien man kan gå fra noden skal være minst mulig. Her kan man tenke seg at man lager retning på kantene for å vise hvilke noder som er «mest sentrale», dvs. at man har en kant (u, v) hvis v er mer sentral enn u. Retningen på kantene vil da gå innover fra løvnodene, og midtpunktene vil havne sist i en topologisk sortering.

Om man lager retning på kantene kan man bruke f.eks. den DFS-baserte algoritmen for å finne den topologiske sorteringen, men når man har kommet så langt at kantretningene er på plass gir svaret seg egentlig selv. Den enkleste måten å finne svaret på her er å bruke den algoritmen for topologisk sortering der man gjentatte ganger «klipper av» noder uten innkanter – i dette tilfellet løvnoder. De nodene vi sitter igjen med til slutt (én eller to) vil være midtpunktene.

Andre lignende løsninger vil også kunne gi full uttelling.