NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4120 Algoritmer og datastrukturer

Faglig kontakt under eksamen	Magnus Lie Hetland
Telefon	918 51 949
Eksamensdato	17. desember, 2016
Eksamenstid (fra–til)	09:00–13:00
Hjelpemiddelkode/tillatte hjelpemidler	D
Annen informasjon	Oppgavearkene leveres inn, med svar i svarrute under hver oppgave
Målform/språk	Bokmål
Antall sider (uten forside)	9
Antall sider vedlegg	0

Informasjon om trykking av eksamensoppgave

Originalen er

1-sidig ☑ **2-sidig** ☐

sort/hvit ☑ **i farger** ☐

Skal ha flervalgskjema ☐

Kvalitetssikret av Ole Edsberg

Kontrollert av

Dato Sign

Merk: Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Les dette nøye

(*i*) Det at kjøretiden er oppgitt til å være $O(n)$ – tvinger det bøttestørrelsen til å være konstant? Er de større vil de jo bli færre. Hvor store kan de være, og likevel gi lineær tid, dersom man tar kvadratet av hver enkelt?

(*ii*) Les hele eksamenssettet nøye *før* du begynner!

(*iii*) Faglærer går normalt én runde gjennom lokalet. Ha evt. spørsmål klare!

(*iv*) Skriv svarene dine i svarrutene og levér inn oppgavearket. Bruk gjerne blyant! Evt. kladd på eget ark først for å unngå overstrykninger, og for å få en egen kopi.

(*v*) Ekstra ark kan legges ved om nødvendig, men det er meningen at svarene skal få plass i rutene på oppgavearkene. Lange svar teller ikke positivt.

(*vi*) Eksamen har 20 oppgaver, totalt verdt 120 poeng. Av disse er 20 bonuspoeng, så en poengsum over 100 regnes som 100. Poengverdi er angitt ved hver oppgave.

Oppgaver

(6 p) 1. Du skal sortere n heltall i verdiområdet 0 til k, der $k = O(n)$. Hvilken sorteringsalgoritme i pensum vil det være mest naturlig å bruke?

Svar: Tellesortering (Counting-Sort).

Forklaring: Tellesortering vil i dette tilfellet ha kjøretid $\Theta(n)$. Man kunne også ha brukt Radix-Sort, men det ville være en unødvendig komplikasjon (3 poeng). Bucket-Sort vil ikke være naturlig å bruke, siden vi ikke vet noe om fordelingen til tallene.

Relevant læringsmål: Forstå Counting-Sort.

Relevant pensum: Kap. 8.2.

(6 p) 2. I læreboka er dybde-først-søk (*depth-first search*, DFS) implementert med rekursjon. Det er også mulig å implementere dybde-først-søk *uten* rekursjon. Hvordan da?

Svar: Ved å bruke en stakk (LIFO-kø) til å holde oppdagede, ubesøkte noder.

Sensurering: Det holder at man nevner bruken av en stakk (jfr. læringsmålet som testes). Om man kun svarer «med iterasjon», så gir det 0 poeng. Å kun svare «kø» er upresist (evt. feil, siden dette normalt betyr en FIFO-kø) og «prioritetskø» er en unødvendig komplikasjon; begge disse svarene gir 2 poeng.

Relevant læringsmål: Forstå hvordan DFS kan *implementeres med en stakk*.

Relevant pensum: Lysark om iterativ DFS (forelesning 8, s. 310).

(6 p) 3. Hvilken teknikk – inkrementell design (*incremental design*), splitt og hersk (*divide-and-conquer*), dynamisk programmering (*dynamic programming*), grådighet (*greedy algorithms*) eller amortisert analyse (*amortized analysis*) – er brukt i den vanlige løsningen på 0-1-ryggsekkproblemet (*0-1 knapsack*)?

Svar: Dynamisk programmering.

Relevant læringsmål: Forstå løsningen på *0-1-ryggsekkproblemet*.

Relevant pensum: Kap. 16; oppgave 16.2-2; lysark, forelesning 6 (s. 402).

(6 p) 4. La tabellen $S[1..10]$ være $\langle 50, 70, 45, 15, 72, 41, 61, 22, 26, 64\rangle$ og la $S.top = 5$. Utfør så følgende utsagn, i rekkefølge:

```
1  x = POP(S)
2  y = POP(S)
3  PUSH(S, x)
4  PUSH(S, y)
```

Hva er innholdet i tabellen S nå?

Merk: Det spørres her om innholdet i *hele* tabellen, ikke bare i stakken.

Svar: $S = \langle 50, 70, 45, 72, 15, 41, 61, 22, 26, 64 \rangle$

Forklaring: Her er $x = 72$ og $y = 15$. POP returnerer elementet $S[S.top]$ og dekrementerer $S.top$, men endrer ikke innholdet i S. PUSH inkrementerer først $S.top$, og setter så inn den angitte verdien som nytt element $S[S.top]$. Effekten her er altså at elementene på posisjon 4 og 5 bytter plass.

Sensurering: Om man her har byttet om på element 5 og 6 og fått $\langle 50, 70, 45, 15, 41, 72, 61, 22, 26, 64 \rangle$, så tyder det på at man antatt at stakken vokser nedover mot 1. Det er ikke slik prosedyrene POP og PUSH er implementert, og det er heller ikke en vanlig eller naturlig måte å gjøre det å (siden man da blir nøtt til å endre $S.top$ ved reallokering når stakken blir full), men det viser at man har forstått mye av virkemåten. Det gis derfor 4 poeng.

Relevant læringsmål: Forstå hvordan *stakker* fungerer. Forstå algoritmene for de ulike operasjonene på strukturen.

Relevant pensum: Side 232–233.

(6 p) 5. Etter at vi har utført en algoritme for å finne korteste vei fra én node s til alle andre i en graf (*the single-source shortest path problem*) har vi $v.\pi = \text{NIL}$ for en gitt node v. Hva er da verdien til $v.d$?

Svar: $v.d = 0$ eller $v.d = \infty$

Forklaring: Her var det egentlig tenkt at $v \neq s$. I så fall, hvis $v.\pi = \text{NIL}$, betyr det at v ikke har noen forgjenger i kortestei-vei-treet, altså at vi ikke har funnet noen sti fra s til v. Men siden dette ikke ble presisert godtas både «0» (dvs., når $s = v$), «∞» og «0 eller ∞» som svar.

Relevant læringsmål: Forstå hvordan man kan representere et *korteste-vei-tre*.

Relevant pensum: Kap. 24 (s. 647).

(6 p) 6. Hvis du har et (ikke nødvendigvis balansert) binært søketre med n noder og høyde h, hvor lang tid tar det å finne minste element (TREE-MINIMUM)? Uttrykk svaret med O-notasjon.

Svar: $O(h)$

Sensurering: Her vil svarene $O(\lg n)$ og $O(n)$ gi 1 poeng. Om man skriver $O(h)$, men så forklarer at dette er $O(\lg n)$ eller $O(n)$ så gir det også 1 poeng, med mindre man har spesifisert at det kun gjelder *best-case* eller *worst-case*.

Relevante læringsmål: Forstå flere ulike operasjoner på binære søketrær, ut over bare søk.

Relevant pensum: Kap. 12 (s. 291).

(6 p) 7. Hva er den amortiserte kjøretiden for innsetting i en dynamisk tabell (TABLE-INSERT)? Oppgi svaret i O-notasjon.

Svar: $O(1)$

Sensurering: Evt. ekstra kommentarer om å sjekke hvovidt elementet er i tabellen fra før vitner om misforståelse av oppgaven (knyttet til hashing), og gir dermed 0 poeng.

Relevante læringsmål: Kunne definere *amortisert analyse*; forstå hvordan *dynamiske tabeller* fungerer.

Relevant pensum: Kap. 17.4 (s. 463).

(6 p) 8. Om man bruker BUCKET-SORT på n uavhengig uniformt fordelte tall i området $[0, 1)$, så får man en kjøretid på $O(n)$. Underveis, som en del av BUCKET-SORT, kalles INSERTION-SORT flere ganger. Hva er den forventede kjøretiden til *hvert enkelt* av disse kallene til INSERTION-SORT? (Det er her altså snakk om en forventningsverdi.) Bruk O-notasjon og uttrykk svaret som funksjon av n.

Merk: Du skal *ikke* oppgi svaret som funksjon av input-størrelsen til INSERTION-SORT, men som funksjon av input-størrelsen til BUCKET-SORT.

Svar: $O(1)$.

Forklaring: INSERTION-SORT brukes for å sortere hver liste, og hver av disse forventes å ha konstant lengde. Eventuelt kan man her resonnere seg baklengs: Siden det er oppgitt at kjøretiden til BUCKET-SORT er lineær, hver av de n bøttene sorteres med INSERTION-SORT, og hvert kall har samme forventede kjøretid, så må hvert kall ta konstant tid.

Sensurering: Her godtas også $O(2 - 1/n)$.

Flere har svart $O((n/k)^2)$, der k er antall bøtter. Det er unaturlig å oppgi antall bøtter separat her, siden kjøretiden er oppgitt, og vi får $k \cdot O((n/k)^2) = O(n)$, som gir oss $O(k) = O(n)$. Likevel, siden dette svaret viser stor grad av forståelse for det oppgaven er ment å teste, gis det 4 poeng.

Merk at svaret ikke er helt rett uansett, *selv om* man ønsker å oppgi kjøretiden som funksjon også av antall bøtter. Om man regner ut $E[n_i]^2$, kommer man lett frem til $O((n/k)^2)$, men det vi er ute etter er $E[n_i^2]$, som i dette tilfellet blir $(n(n+k-1))/k^2 = O((n/k)^2 + nk/k^2)$.

Relevant læringsmål: Forstå BUCKET-SORT; kjenne kjøretidene under ulike omstendigheter, og forstå utregningen.

Relevant pensum: Kap. 8.4 (s. 202–203).

(6 p) 9. $A[1..n]$ er en tabell med heltall. Du har nesten skrevet ferdig en algoritme basert på designmetoden *splitt og hersk* (*divide-and-conquer*) for å finne det minste elementet (eller ett av de minste elementene) i A. Det du har skrevet så langt ser slik ut:

```
MINIMUM(A, p, r)
1  if p == r
2       return A[p]
3  else m = ⌊(p + r)/2⌋
4       x = [ MINIMUM(A, p, m)          ]
5       y = [ MINIMUM(A, m + 1, r)      ]
6       if x < y
7            return x
8       else return y
```

For å finne minimum i A bruker du MINIMUM$(A, 1, n)$. Fyll ut det som mangler i de to rutene i pseudokoden over.

Forklaring: Standard *divide-and-conquer*-løsning. Se f.eks. MERGE-SORT for en algoritme med lignende rekursive kall (s. 24).

Sensurering: Her er det naturligvis helt greit å bruke ekvivalente grenser, som for eksempel $(p, r - m + p - 1)$ og $(r - m + p, r)$.

Hovedpoenget er å teste grunnleggende forståelse av designmetoden. Svar som har små indekseringsfeil, eller som ikke har fått med A, vil også gi noe uttelling. (Flere varianter her, men typisk 2 poeng trekk per gale indeks eller manglende A, eller en uttelling på 3 poeng for én indekseringsfeil og en manglende A).

Relevant læringsmål: Forstå ideen bak *divide-and-conquer*.

Relevant pensum: Kap. 4.

(6 p) 10. Løs følgende rekurrens, der $n \geq 0$ er et heltall:

$$T(n) = \begin{cases} 0 & \text{hvis } n = 1\,, \\ 10\,000\,T(n/10) + n^4 + n^2 & \text{hvis } n > 1\,. \end{cases}$$

Oppgi svaret i Θ-notasjon.

Svar: $\Theta(n^4 \lg n)$

Forklaring: Kan for eksempel løses med masterteoremet (tilfelle 2), der $a = 10\,000$, $b = 10$, $\lg_b a = 4$ og $f(n) = n^4 + n^2 = \Theta(n^4)$.

Relevant læringsmål: Kunne løse rekurrenser med *substitusjon*, *rekursjonstrær* og *masterteoremet*.

Relevant pensum: Kap. 4.3–4.5 (f.eks. s. 94).

(6 p) 11. Din venn Lurvik mener følgende rekurrens har løsning $T(n) = n^3 - n^2$:

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 8T(n/2) + n^2 & \text{if } n > 1, \end{cases}$$

der $n = 2^k$, for et positivt heltall k. Vis at hun har rett.

Svar: Vises for eksempel ved induksjon (substitusjonsmetoden).

Grunntilfelle: $T(1) = 1^3 - 1^2 = 0$.

Induktivt trinn: Antar $T(n/2) = (n/2)^3 - (n/2)^2 = n^3/8 - n^2/4$.

Substituerer dette inn i rekurrensen: $T(n) = 8(n^3/2^3 - n^2/2^2) + n^2 = n^3 - 2n^2 + n^2 = n^3 - n^2$.

Forklaring: Svaret holder for grunntilfellet og videreføres i det induktive trinnet, og holder dermed for alle naturlige tall n. Den induktive antagelsen er at $T(m) = m^3 - m^2$, for alle positive heltall $m < n$. Mer spesifikt antar vi altså dette for $m = n/2$.

Sensurering: Det står i oppgaven at k er positiv. Det burde ha stått «ikke-negativ», for om vi krever $n \geq 2$, så er ikke T definert. Dersom man har tolket det til å bety at man skal vise at *løsningen holder* for $n \geq 2$, og har grunntilfelle $T(2)$, men likevel bruker det faktum at $T(1) = 0$, så gir dette full uttelling.

Enkelte har prøvd å vise korrekthet ved å sette inn $n = 1, 2, \ldots$. Dette vil naturligvis ikke vise at svaret er korrekt (bare gi eksempler på verdier der det er korrekt), men det viser likevel en viss forståelse for rekurrenser, og gir dermed 2 poeng.

Relevant læringsmål: Kunne løse rekurrenser med *substitusjon*.

Relevant pensum: Kap. 4.3.

(6 p) 12. Du har en urettet graf $G = (V, E)$, der hver kant $(u, v) \in E$ har en vekt $w(u, v) = -1$. Du har oppgitt to noder s og t, og ønsker å finne den lengste (dvs. tyngste) stien fra s til t, altså den med størst vektsum. Hvordan ville du løse problemet?

Svar: Bruk BFS.

Forklaring: Siden alle kantene har vekt -1, tilsvarer dette å finne korteste vei i samme graf, men med vekter 1, eller, ekvivalent, i den uvektede grafen. Dette kan løses med bredde-først-søk (BFS).

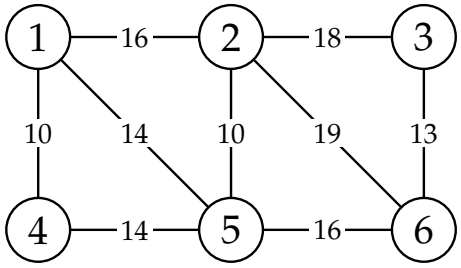
Sensurering: Her kan man naturligvis også bruke andre korteste-vei-algoritmer (etter å ha skiftet fortegn på vektene), og få dårligere kjøretid. Om man bruker Dijkstra gir det 2 poeng. Om man ikke spesifiserer algoritme, gir det 1 poeng. Om man bruker Bellman-Ford, gir det 0 poeng. Om man ikke spesifiserer hvordan vektene endres, gir det 0 poeng.

Relevant læringsmål: Forstå BFS, også for å finne *korteste vei uten vekter*.

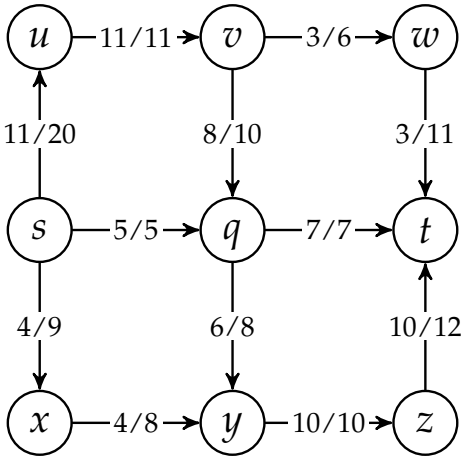
Relevant pensum: Kap. 22.2 (s. 594).

(6 p) 13. I Transitive-Closure brukes den binære variabelen $t_{ij}^{(k)}$ til å indikere om det går en sti fra i til j hvis alle noder på veien mellom dem *må* ligge i mengden $\{1, 2, \ldots, k\}$. For eksempel er $t_{ij}^{(0)} = 1$ hvis og bare hvis $(i, j) \in E$. Hva er uttrykket for $t_{ij}^{(k)}$, når $k > 0$?

Svar: $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$



Figur 1: En vektet, urettet graf til bruk i oppgave 14



Figur 2: Flytnettverk brukt i oppgave 15

Sensurering: Om man svarer $t_{ij}^{(k)} = t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}$, så gir det 2 poeng.

Relevant læringsmål: Forstå TRANSITIVE-CLOSURE.

Relevant pensum: Delavsnitt på side 697 (ligning 25.8).

(6 p) 14. Hvis du utfører MST-KRUSKAL på grafen i figur 1, hvilken kant vil velges som den femte i rekken? Det vil si, hvilken kant vil være den femte som legges til i løsningen?

Oppgi kanten på formen (i, j), der $i < j$.

Svar: $(5, 6)$.

Forklaring: MST-KRUSKAL velger hele tiden den letteste kanten som ikke danner en sykel i løsningen. Her vil den først velge $(1, 4)$ og $(2, 5)$ i en udefinert rekkefølge og deretter $(3, 6)$, og så enten $(1, 5)$ eller $(4, 5)$, men ikke begge, siden det ville danne en sykel. Etter dette vil $(1, 2)$ danne en sykel, så den neste kanten må bli $(5, 6)$.

Relevant læringsmål: Forstå MST-KRUSKAL.

Relevant pensum: Kap. 23.2 (s. 631).

(6 p) 15. Figur 2 viser flytnettverket G, med kilde s, sluk t og flyt f. Er flyten maksimal? Svar ja eller nei.

Hvis ja, oppgi også mengden av noder som kan nås (dvs., som det finnes stier til) fra s i G_f.

Hvis nei, oppgi også nodene i en flytforøkende sti (*augmenting path*), i rekkefølge.

Svar: Nei, flyten er ikke maksimal. Den eneste flytforøkende stien er $\langle s, x, y, q, v, w, t \rangle$.

Forklaring: Her opphever vi flyt som i utgangspunktet går gjennom kantene (q, y) og (v, q). Flaskehalsen blir kanten (v, w), der vi kun kan øke med 3 enheter, så vi kan totalt øke flyten med 3.

Relevant læringsmål: Forstå FORD-FULKERSON.

Relevant pensum: 26.2.

(6 p) 16. Under en kjøring av FLOYD-WARSHALL er $D^{(2)}$ og $\Pi^{(2)}$ som angitt i figur 3. Hva blir $D^{(3)}$ og $\Pi^{(3)}$? Fyll ut tabellene nedenfor.

Merk: Vi antar her en implementasjon som i læreboka, det vil si at vi i hver iterasjon k lager nye tabeller $D^{(k)}$ og $\Pi^{(k)}$, heller enn en mer plass-effektiv variant som overskriver tabellene.

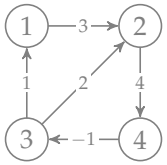
$D^{(3)}$

	1	2	3	4
1	0	3	∞	7
2	∞	0	∞	4
3	1	2	0	6
4	0	1	-1	0

$\Pi^{(3)}$

	1	2	3	4
1	NIL	1	NIL	2
2	NIL	NIL	NIL	2
3	3	3	NIL	2
4	3	3	4	NIL

Forklaring: Grafen ser ut som følger:



Fra $k = 0$ til $k = 1$ er ingenting endret, så D og Π er gitt direkte av grafen. For $k = 2$ får vi $d_{1,4} = 7$ og $d_{3,4} = 6$, ved å gå innom node 2. Vi har da situasjonen i figur 3. (Merk at man ikke trenger vite hvordan vi kom frem hit.) Fra $k = 2$ til $k = 3$, som er iterasjonen det spørres om, får vi $d_{4,1} = 0$ og $d_{4,2} = 1$ ved å gå via node 3.

Sensurering: Om $D^{(3)}$ er korrekt, mens $\Pi^{(3)}$ er gal, gis det 3 poeng. Noen har fullført hele algoritmen (dvs., gått videre til $k = 4$). Det gir 4 poeng. Man vil da få:

$D^{(4)}$

	1	2	3	4
1	0	3	6	7
2	4	0	3	4
3	1	2	0	6
4	0	1	-1	0

$\Pi^{(4)}$

	1	2	3	4
1	NIL	1	4	2
2	3	NIL	4	2
3	3	3	NIL	2
4	3	3	4	NIL

Relevant læringsmål: Forstå FLOYD-WARSHALL; kunne utføre algoritmen, trinn for trinn.

Relevant pensum: Kap. 25.2 (s. 695–697).

$D^{(2)}$

	1	2	3	4
1	0	3	∞	7
2	∞	0	∞	4
3	1	2	0	6
4	∞	∞	-1	0

$\Pi^{(2)}$

	1	2	3	4
1	NIL	1	NIL	2
2	NIL	NIL	NIL	2
3	3	3	NIL	2
4	NIL	NIL	4	NIL

Figur 3: Forrige tilstand i utførelsen av FLOYD-WARSHALL, brukt i oppgave 16

(6 p) 17. Et flytnettverk er en rettet graf $G = (V, E)$ der hver kant $(u, v) \in E$ har en *kapasitet* $c(u, v) \geq 0$. Hvis $(u, v) \notin E$, lar vi $c(u, v) = 0$. En *flyt* i G er en reell funksjon $f : V \times V \to \mathbb{R}$ som tilfredsstiller to egenskaper. Hvilke egenskaper er dette?

Egenskapene uttrykkes fortrinnsvis med matematisk notasjon. En kort tekstlig beskrivelse kan gi opptil 3 poeng. Det holder *ikke* å oppgi navnene på dem. (Det er heller ikke *nødvendig* å oppgi navnene deres.)

Svar: Flyten må være ikke-negativ og ikke overgå kapasiteten, og summen av flyt inn i en node (utenom kilde og sluk) må være lik summen av flyt ut.

Mer presist:

(*i*) For alle $u, v \in V$ krever vi at $0 \leq f(u, v) \leq c(u, v)$.

(*ii*) For alle $u \in V - \{s, t\}$ krever vi at $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$.

Sensurering: Her gis det også full uttelling om man ikke oppgir kravet $f(u, v) \geq 0$.

Relevant læringsmål: Kunne definere *flytnettverk*, *flyt* og *maks-flyt-problemet*.

Relevant pensum: Kap. 26.1 (s. 709).

(6 p) 18. Er følgende utsagn riktig?

> «For ethvert språk L, hvis L er i NP, så er L ikke i co-NP.»

Svar ja eller nei, og forklar svært kort.

Svar: Nei. For eksempel er ethvert problem i P både i NP og co-NP.

Forklaring: NP er mengden av språk som kan verifiseres i polynomisk tid. Uformelt betyr det at vi kan sjekke sertifikater for ja-svar i polynomisk tid. co-NP er mengden av språk der *komplementet* kan verifiseres i polynomisk tid, altså at vi kan sjekke sertifikater for nei-svar i polynomisk tid. For eksempel for problemer i P gjelder begge deler.

Sensurering: Her vil også svar av typen «Det er ukjent hvorvidt NP = co-NP» aksepteres. Å kun svare «Det er ukjent hvorvidt P = NP» uten samtidig å påpeke at det er ukjent hvorvidt $\text{P} = \text{NP} \cap \text{co-NP}$ gis ikke full uttelling.

Relevant læringsmål: Forstå definisjonen av klassene NP og co-NP.

Relevant pensum: Kap. 34.2, s. 1064.

(6 p) 19. En *uavhengig mengde* (*independent set*) i en graf $G = (V, E)$ er en delmengde $U \subseteq V$ av nodene som er slik at hver kant i E er tilkoblet maksimalt én node i U (dvs., ingen av nodene i U er naboer).

Et *fritt tre* (*free tree*) er en sammenhengende, asyklisk urettet graf (dvs., et tre uten noen angitt rot).

Du skal løse følgende problem.

> **Input:** Et fritt tre $T = (V, E)$ der hver node $v \in V$ har en vekt $w(v) \neq 0$.
> **Output:** En uavhengig mengde U i T med maksimal vektsum $\sum_{v \in U} w(v)$.

Se figur 4 for et eksempel. Merk at nodevektene *kan være negative*.

For enkelhets skyld trenger du *ikke* beskrive hvordan du vil finne selve mengden U; det holder at du finner korrekt vektsum. Beskriv konsist en algoritme som løser problemet effektivt. Oppgi kjøretiden så presist som mulig, i asymptotisk notasjon.

Svar: Velg en vilkårlig rot, så du får et rotfast tre. For hvert deltre med rot u skal u enten være med eller ikke. Prøv begge deler, og velg beste lovlige løsning for hvert barn v: Om u er med, kan ikke v være med; ellers kan vi velge fritt. Kan enten løses rekursivt, med memoisering, eller iterativt, f.eks. etter høyden til noden u. Kjøretiden blir $\Theta(V)$.

Forklaring: Hovedpoenget her er å bruke dynamisk programmering, der et delproblem er spesifisert av hvorvidt en gitt node skal være med eller ikke. Om noden er med, tvinger det naboene

til å ikke være med, mens om noden ikke er med, så står naboene fritt til å være med eller ikke, uavhengig av hverandre. For å kunne løse problemet med denne fremgangsmåten, må man finne en fornuftig rekkefølge på disse delproblemene, for eksempel ved å gjøre treet rotfast, og så løse problemet for barn før foreldre. Arbeidet som gjøres i en node vil variere med antall barn, men arbeidet per barn (evt., per kant, der $|E| = |V| - 1$ er konstant. Derfor blir kjøretiden lineær som funksjon av antall noder.

Merk at det som står om at man ikke eksplisitt trenger å beskrive hvordan man finner mengden U tilsvarer at man dropper trinn 4 og kun trenger trinn 1–3 i DP-løsningen (se s. 359). Dvs., man skal beregne *verdien* til den optimale løsningen, men trenger ikke faktisk finne den optimale løsningen. (Det betyr naturligvis ikke at man har den optimale løsningen oppgitt, eller at man har tilgjengelig en funksjon som kan finne den.)

Relevant læringsmål: Forstå designmetoden *dynamisk programmering*.

Relevant pensum: Kap. 15. (App. B.4.)

(6 p) 20. En *dominerende mengde* (*dominating set*) i en urettet graf er en delmengde av nodene som tilsammen er naboer med alle de andre nodene. Det vil si, en dominerende mengde for en urettet graf $G = (V, E)$ er en mengde $U \subseteq V$ som er slik at for enhver node $v \in V - U$ så finnes det minst én node $u \in U$ der $(u, v) \in E$. (Se figur 5 for et eksempel.)

Det såkalte *dominating set problem* handler om å finne en dominerende mengde av minimal størrelse (dvs., med så få noder som mulig) i en gitt graf. Uttrykt som et beslutningsproblem, går det ut på å avgjøre om grafen har en dominerende mengde av en gitt størrelse k. Som et språk, definerer vi

$$\text{DOMINATING-SET} = \{\langle G, k\rangle : \text{grafen } G \text{ har en dominerende mengde med } k \text{ noder}\}.$$

Anta at du allerede vet at språket VERTEX-COVER er NP-komplett. Bruk denne kunnskapen til å vise at DOMINATING-SET er NP-komplett.

Merk: For full uttelling må alle bestanddelene i et NP-kompletthetsbevis være med.

Hint: Anta at noden w kun har u og v som naboer, og at u og v er naboer med hverandre, som i figur 6. Dersom w er med i løsningen vår, kan vi alltid bytte den ut med enten u eller v, og fortsatt ha en løsning (en dominerende mengde) med samme antall noder.

Svar: Vi kan bruke den dominerende mengden selv som sertifikat, og det kan enkelt verifiseres i polynomisk tid. Dermed vet vi at DOMINATING-SET er i NP.

Vi reduserer fra VERTEX-COVER ved å innføre en ny node w for hver kant (u, v), som i figur 6. Det kan enkelt gjøres i polynomisk tid. La den originale grafen være G, og den nye grafen G'. Et nodedekke (*vertex cover*) for G vil enten inneholde u eller v for enhver kant (u, v), og er dermed også en dominerende mengde for G. Denne mengden vil inneholde minst én nabo for hver av de nye nodene i G'. Altså, om G har et nodedekke av størrelse k, så har G' en dominerende mengde av størrelse k.

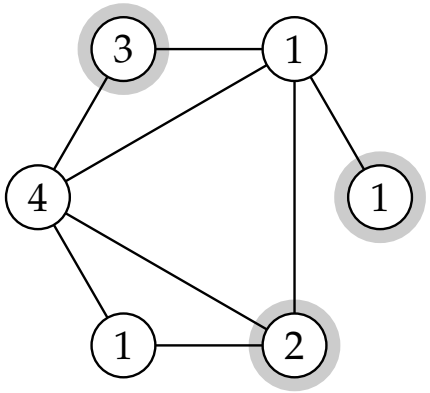
For ethvert trippel u, v, w som i figur 6, må enten u, v eller w være med i en dominerende mengde. Som forklart i hintet, kan vi alltid bytte ut w med u eller v, og disse dekker begge kanten (u, v). Altså, dersom G' har en dominerende mengde av størrelse k, så har G et nodedekke av størrelse k.

Sensurering: Her er det viktig at reduksjonen går rett vei, altså *fra* VERTEX-COVER. Det er også viktig at man viser at implikasjonen går begge veier (dvs. $x \in$ VERTEX-COVER $\iff f(x) \in$ DOMINATING-SET.) For full uttelling må man også vise medlemsskap i NP og at reduksjonen tar polynomisk tid.

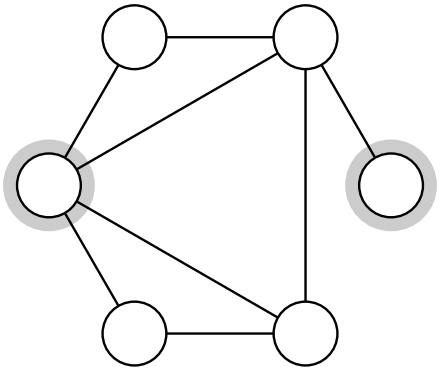
Merk at dersom an *sier* at man må redusere fra VERTEX-COVER, men så tar utgangspunktet i input-instanser for DOMINATING-SET, eller løsninger for VERTEX-COVER (og altså faktisk reduserer feil vei), gis det ingen poeng.

Relevant læringsmål: Være i stand til å konstruere enkle NP-kompletthetsbevis.

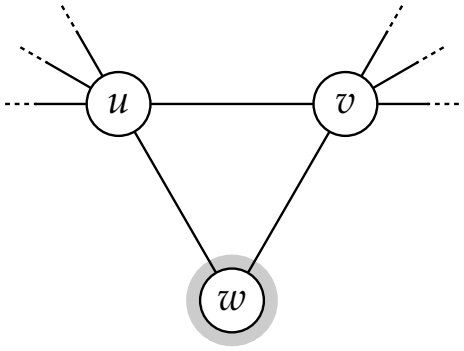
Relevant pensum: Kap. 34.4, 34.5.2.



Figur 4: Eksempel på en maksimal, vektet uavhengig mengde (se oppgave 19). De tre uthevede nodene er med i den uavhengige mengden U, som har vektsum $\sum_{v \in U} w(u) = 1 + 2 + 3 = 6$



Figur 5: En dominerende mengde (se oppgave 20) med størrelse 2. De uthevede nodene er med i den dominerende mengden, og hver av de andre nodene har en kant til minst én av dem



Figur 6: Nodene u og v kan være koblet til andre noder i grafen, men w er kun koblet til u og v. Dersom w er med i en dominerende mengde av størrelse k, så kan vi bytte ut w med enten u eller v, og fortsatt ha en dominerende mengde av størrelse k