Informasjon

Denne øvingen var frivillig og teller ikke inn for å bestå øvingsopplegget.

Finner du feil eller mangler eller har forslag til forbedringer, ta gjerne kontakt!

Oppgave 2

«*Legge til et supersluk bak alle eksamenslokasjonene, og en superkilde bak alle fabrikkene, hvor kantvekten mellom kilde og fabrikk er produksjonskapasiteten, og kapasitet mellom eksamenslokasjonene og sluk er* ∞.» og «*Splitte de antiparallelle kantene som går mellom fabrikkene i to, og legge en node mellom de to nye kantene.*» er riktig.

Kommentar:
Dette er vanlig håndtering av antiparallelle kanter og flere kilder og sluk, som nevnt i forelesning 12, og CLRS kapittel 26.

«*Fjerne syklene som er mellom fabrikkene, siden det da kan hende vi aldri klarer å finne en maksimal flyt.*» - Ingen av algoritmene i pensum har problemer med å finne makismal flyt i grafer med sykler.

«*Splitte fabrikk-nodene i to, hvor kapasiteten mellom de to nodene er lik produksjonskapasiteten, uten å legge til supersluk og superkilde.*» gir mening som en del av løsningen, men siden vi da utelukker å legge til supersluk og superkilde kan det dermed ikke være en del av den totale løsningen.

Oppgave 3

«$c(u, v) = 6$ *og* $f(u, v) = 3$» er riktig

Kommentar:
Notasjonen er på formen $f(u, v)/c(u, v)$ og hvor $c(u, v)$ indikerer kapasiteten til kanten og $f(u, v)$ den nåværende flyten over kanten. En mer detaljert forklaring finnes i CLRS s. 710 og forelesning 12.

Oppgave 4

«*3*» er riktig.

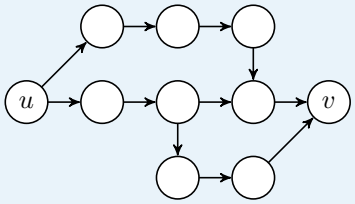
Kommentar:
Det følger av definisjonen av residualkapasitet ligning (26.2), CLRS s. 716. Siden $(u, v) \in$ E får vi at $c_f(u, v) = c(u, v) - f(u, v) = 6 - 3 = 3$

Oppgave 5

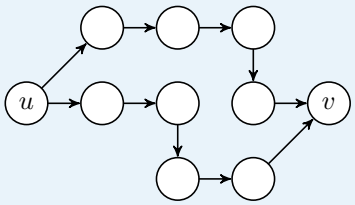
«*Det maksimale antallet stier mellom u og v som kan følges samtidig, uten å dele kanter.*» er riktig.

Kommentar:
Denne oppgaven var inspirert av oppgave g på eksamen 2012S.

«*Det maksimale antallet stier mellom v og u som kan følges samtidig, uten å dele kanter.*» — siden kantene er rettet, vil det ikke nødvendigvis finnes en sti motsatt vei.
«*Korteste vei fra u til v.*» — Den korteste veien mellom u og v er ikke nødvendigvis en del av løsningen for maksimal flyt. Et eksempel på dette er følgende graf:



I denne grafen går den korteste stien fra u til v gjennom de tre midtre nodene, men denne stien kan ikke være med i en maksimal pakking av kanter fra u til v fordi med denne korteste stien er det ikke mulig å finne andre stier mellom u og v som ikke bruker noen av de samme kantene. Derimot er det maksimale antallet med stier mellom u og v som ikke deler kanter, 2. Slik som kan ses i grafen under:



«*Det maksimale antallet stier mellom u og v*» — Antallet mulige stier har kan gjenbruke samme kant uavhengig av kapasiteten, dermed finner vi ikke alle mulige.

Oppgave 6

«*22*» er riktig.

Kommentar:
Vi summerer opp flyt ut fra kilden S, og får $14 + 8 = 22$ i total flyt.

Oppgave 7

«S $\to v_2 \to v_4 \to$ T» er riktig.

Kommentar:
BFS finner den korteste flytforøkende stien. I flytnettverket er det nøyaktig to mulige stier fra s til t, men kun en av disse finnes i residualnettverket.

Oppgave 8

«7» er riktig.

Kommentar:
Kanten (v_4, t) er den som har minst residualkapasitet og begrenser oss til å kunne sende $19 - 12 = 7$ flyt over stien.

Oppgave 9

«33» er riktig.

Kommentar:
Etter å ha brukt den flytforøkende stien som ble funnet i oppgave 7, kan vi forstatt finne en flytforøkende sti. En av disse er stien $s \to v_1 \to v_2 \to v_4 \to v_3 \to t$. Denne gir en økning av flyten på 4 og etter dette kan vi ikke lengre finne noen flytforøkende sti. Det vil si at den totale flyten blir $22 + 7 + 4 = 33$, basert på dette og de tidligere oppgavene.

Det minimale snittet blir $\{\{s, v_1, v_2, v_4\}, \{v_3, t\}\}$.

Oppgave 10

«29» er riktig.

Kommentar:
Flyten over et snitt kan man finne ved å telle hvor mye flyt som går fra det første settet med noder til det andre og trekke av hvor mye flyt som går motsatt vei. Det vil si, man bruker følgende formel.

$$f(\mathrm{S}, \mathrm{T}) = \sum_{u \in \mathrm{S}} \sum_{v \in \mathrm{T}} f(u, v) - \sum_{u \in \mathrm{S}} \sum_{v \in \mathrm{T}} f(v, u)$$

Bruker vi denne og fjerner alle nodepar hvor det ikke er mulig at det går flyt mellom, får vi

$$f(\{s, v_2\}, \{v_1, v_3, v_4, t\}) = f(s, v_1) + f(v_2, v_4) - f(v_1, v_2) - f(v_3, v_2) = 14 + 19 - 4 - 0 = 29$$

Oppgave 11

«Edmonds-Karp *er en implementasjon av* Ford-Fulkerson*, som benytter* BFS *for å finne flytforøkende stier (*augmenting paths*)*» er riktig.

Kommentar:
Se forelesning 12, eller læreboka kapittel 26.

Vi bruker ikke DFS siden det kan føre til at kjøretiden er avhengig av den maksimale flyten, om vi er uheldige. Merk at vi fint kunne benyttet Dijkstra til å finne flytforøkende stier, det ville bare ført til verre kjøretid.

Oppgave 12

«*Fordi vi måler kjøretid basert på input-størrelse, og tallet* $|f^*|$ *kan lagres med* $c \cdot \lg |f^*|$ *bits, hvor* c *er konstant, er kjøretiden eksponentiell.*» og «*Ford-Fulkerson har polynomisk kjøretid, som funksjon av* $|f^*|$*.*» er riktig.

Kommentar:
Vi definerer vanligvis kjøretid som en funksjon av *problemstørrelsen*, som måles i hvor mange bits det kreves for å lagre en full instanse av problemet. Vi kan derimot definere kjøretid som en funksjon av tallet $|f^*|$, og da blir kjøretiden polynomisk. Dette gås også igjennom i forelesning 12. Appendiks D i pensumheftet er også anbefalt, da det beskriver samme situasjon, bare med det binære ryggsekkproblemet.

«*Lurespørsmål!* Ford-Fulkerson *er ikke en algoritme og kan derfor ikke ha en kjøretid.*» — Selv om Ford-Fulkerson ikke spesifiserer nøyaktig hvordan vi finner flytforøkende stier, så kan vi beskrive kjøretiden til den, ved å anslå kjøretiden til prosedyren. Det anbefales å se på s. 725 i CLRS, for en fullforklaring bak kjøretiden.

«Ford-Fulkerson *får dårlig kjøretid siden den finner den korteste flytforøkende stien, som kan medføre at vi kun øker flyten i nettverket med en flytenhet for hver iterasjon av for-løkken.*» — For det første, så finner ikke Ford-Fulkerson nødvendigvis den korteste flytforøkende stien. For det andre, så er det å finne den korteste flytforøkende stien som *hindrer* at vi ender opp i situasjoner hvor vi kun øker med en flytenhet for hver itersjon.

Oppgave 13

Kort fortalt: vi kan vise at avstander i residualnettverket aldri synker når vi bruker BFS. Basert på det, så kan vi vise at en kant kun kan være en flaskehals maksimalt $|V|/2$ ganger, altså en kant some begrenser flytøkningen i en sti. Og det kan skje for hver kant i E, så vi får O(VE) iterasjoner med flytforøkning, som hver tar O(E).
Se forøvrig Hvorfor har Edmonds-Karp kjøretid $O(VE^2)$, forelesning 12, og CLRS s. 727–730 dersom du ennå er usikker.

Oppgave 14

```python
def max_flow(
    source: int, sink: int, nodes: int, capacities: List[List[int]]
) -> Tuple[List[List[int]], int]:
    # initialiserer flyt-nettverket
    flows = [[0] * nodes for _ in range(nodes)]
    total_flow = 0

    path = find_augmenting_path(source, sink, nodes, flows, capacities)

    while path != None:
        flow = max_path_flow(path, flows, capacities)
        send_flow(path, flow, flows)
        total_flow += flow
        path = find_augmenting_path(source, sink, nodes, flows, capacities)

    return flows, total_flow
```

Kommentar:
Merk at grafene i testene er relativt tette (*dense*), og det derfor kan være nyttig å prøve å implementere en annen algoritme, f.eks. Push-Relabel, slik at antallet kanter har mindre å si for kjøretiden.

Oppgave 15

«*En delmengde av alle kanter, der hver node er tilknyttet maks en kant fra delmengden.*» er riktig.

Kommentar:
Merk at de andre alternativene beskriver andre kjente problemer:

«*En delmengde av alle kanter som er slik at alle noder har minst en tilknyttet kant.*» — Beskriver Edge-Cover

«*En delmengde av alle noder som er slik at alle kanter har minst en tilknyttet node.*» — Beskriver Vertex-Cover

Oppgave 16

Konstruer en bipartitt graf med noder som korresponderer til hvert filnavn f_i for $1 \leqslant i \leqslant n$, og en samling med akseptable fortkortelser for hvert filnavn $f_{i1}, \ldots, f_{ik}$. Legg til en kant mellom hvert originale filnavn, og den akseptable forkortelsen. Da må vi finne en mengde med n kanter, som har ingen felles noder, slik at hvert filnavn endres til en unik akseptabel forkortelse. Tilpass grafen ved å legge til et supersluk og en superkilde, og bruk en algoritme som løser maksimal-flyt problemet. Matchingen du får er løsningen på problemet.

Kommentar:
Denne oppgaven er hentet fra andre utgave av *The Algorithm Design Manual* av Steven S. Skiena.

Oppgave 17

Riktige svar er:

- «*Dersom man kan løse ett problem i NPC i polynomisk tid så kan man løse alle problemer i NP i polynomisk tid.*»
- «*Alle problem i NPC kan verifiseres i polynomisk tid.*»

Kommentar:
«Dersom man kan løse ett problem i NPC i polynomisk tid så kan man løse alle problemer i NP i polynomisk tid» er riktig ettersom man kan redusere alle problemer i NP til et gitt problem i NPC polynomisk. Siden man har en polynomisk løsning på et NPC-problem vil man da ha en polynomisk løsning på alle NP-problemer ved å kombinere en polynomisk reduksjon av NP-problemet til det aktuelle NPC-problemet, for så å kjøre den revolusjonerende algoritmen som også tar polynomisk tid.

«Alle problem i NPC kan verifiseres i polynomisk tid.» stemmer fordi alle problemet i NPC er også i NP, og NP består av alle problemer som kan verifiseres polynomisk.

«Om en algoritme løser et problem i NP, så kan den også løse alle NP-harde problemer.» Stemmer ikke, ellers kunne f.eks. PRIM's algoritme løse alle NP-harde problemer.

«For at et problem skal være i NP må det kunne løses i polynomisk tid.» Stemmer ikke, det må kunne verifiseres i polynomisk tid.

Oppgave 18

Riktige svar er:

- «*Klassen co-NP består av språkene L der komplementet av L kan verifiseres i polynomisk tid.*»
- «*En instans av et konkret problem (*concrete problem*) er en binær streng.*»

Kommentar:
«Klassen co-NP består av språkene L der komplementet av L kan verifiseres i polynomisk tid.» er definisjonen på co-NP.

«En instans av et konkret problem (*concrete problem*) er en binær streng.» Stemmer siden et konkret problem er et problem som har mengden av binærstrenger som instanse mengde.

«Et optimeringsproblem er som regel enklere enn et tilhørende bestemmelsesproblem (*decision

problem*).» stemmer ikke - optimeringsproblemer er som regel vanskeligere enn bestemmelsesproblemer.

«Kodingen av en probleminstans har ingen ting å si for kompleksiteten av å løse problemet.» stemmer ikke - kompleksiteten av en algoritme avhenger av størrelsen på inputen som igjen avhenger av kodingen av probleminstansen.

«En algoritme A sies å akseptere et språk L dersom $\mathrm{A}(x) = 1$ hvis $x \in \mathrm{L}$ og $\mathrm{A}(x) = 0$ hvis $x \notin \mathrm{L}$.» stemmer ikke - hvis dette gjelder for en algoritme A og et språk L, aksepterer algoritmen språket, men den siste delen av utsagnet er ikke fullstending nødvendig. For at et språk skal aksepteres krever vi kun at L er settet $\{x \in \Sigma^* | \mathrm{A}(x) = 1\}$ det vil si alle strenger x som er slik at $\mathrm{A}(x) = 1$. For de resterende strengene i $x \in \Sigma^* \setminus \mathrm{L}$ så trenger vi ikke nødvendigvis at $\mathrm{A}(x) = 0$. For eksempel er det helt greit at A aldri stopper å kjøre for en slik x.

Oppgave 19

«P ⊆ NP», «P ⊆ co-NP» og «NPC ⊆ NP-hard» er riktig.

Kommentar:
P ⊆ NP er riktig siden alle problemer som kan løses polynomisk også kan verifiseres polynomisk.

P ⊆ co-NP er riktig. Siden P kan bestemmes i polynomisk tid kan også komplementet bestemmes i polynomisk tid, og dermed også verifiseres i polynomisk tid.

NPC ⊆ NP-hard er riktig siden NP-harde problemer defineres av én av de to betingelsene som definerer NP-komplette problemer (de trenger ikke være polynomisk verifiserbare).

P = NP er et fundamentalt spørsmål som enda ikke er bevist eller motbevist.

NP-hard ⊆ NP stemmer ikke, NP-harde problemer trenger ikke å være polynomisk verifiserbare.

NP ⊆ co-NP Forholdet mellom NP of co-NP er usikkert.

Oppgave 20

«*Kjøretiden er avhengig av antall bits w en trenger for å representere* W*, kjøretiden blir derfor* $\mathrm{O}(n2^w)$» er riktig.

Oppgave 21

Riktige svar er:

- «*Finne ut om en boolsk 2-CNF-formel er oppfyllbar.*»
- «*Finne ut om en graf har en sti mellom to noder med lengde* $\leqslant k$.»

Kommentar:
Merk at «Sortere en liste med elementer.» ikke er med i P ettersom det ikke er et bestemmelsesproblem.

Oppgave 22

Riktige svar er:

- «*Finne ut om en digital krets kan gi 1 som output.*»
- «*Finne ut om en graf har en hamiltonsykel.*»
- «*Finne ut om en boolsk 3-CNF-formel er oppfyllbar.*»
- «*Finne ut om en boolsk 2-CNF-formel er oppfyllbar.*»
- «*Finne ut om en graf har en sti mellom to noder med lengde* $\leqslant k$.»
- «*Finne ut om en graf har en sti mellom to noder av lengde* $\geqslant k$.»

Kommentar:
Merk at «Sortere en liste med elementer.» og klikk-problemet ikke er med i NP ettersom de ikke er bestemmelsesproblemer.
«Finne ut om et program kommer til å stoppe eller kjøre for alltid.» er Turing's Halting problem, og er som kjent ikke polynomisk verifiserbart.

Oppgave 23

Riktige svar er:

- «*Finne ut om en digital krets kan gi 1 som output.*»
- «*Finne ut om en graf har en hamiltonsykel.*»
- «*Finne ut om en boolsk 3-CNF-formel er oppfyllbar.*»
- «*Finne ut om en graf har en sti mellom to noder av lengde* $\geqslant k$.»

Oppgave 24

Riktige svar er:

- «*Finne ut om en digital krets kan gi 1 som output.*»
- «*Finne ut om en graf har en hamiltonsykel.*»
- «*Finne ut om en boolsk 3-CNF-formel er oppfyllbar.*»
- «*Finne ut om en graf har en sti mellom to noder av lengde* $\geqslant k$.»

- «*Finne den største mulige klikken i en graf.*»
- «*Finne ut om et program kommer til å stoppe eller kjøre for alltid.*»

Oppgave 25

Riktige svar er:

- «*Hvis man finner en polynomisk reduksjon fra B til A har man vist at A er i NPC.*»
- «*B er i NPC hvis man kan verifisere B polynomisk.*»

Oppgave 26

Først må vi bevise at problemet er i NP, altså at det er polynomisk verifiserbart. Et sertifikat på dette problemet vil være en mengde noder V′. For å verifisere at svaret er riktig må vi sjekke at mengden består av k noder, og at mengden er stabil. Det første er trivielt. For å bekrefte at mengden er stabil kan vi iterere over alle kantene E i grafen og sjekke at maksimalt én av de to nodene den er forbundet med er med i den stabile mengden. Dette vil ha polynomisk tidskompleksitet (på O(EV′))

Så må vi vise at problemet er NP-hardt. Dette gjør vi ved å redusere fra CLIQUE. Man kan se at en stabil mengde i en graf G tilsvarer en klikk i den komplementære grafen G'. Dermed kan man redusere et CLIQUE problem til et stabilt mengde problem ved å «flippe» alle kantene i grafen. Denne prosedyren har tidskompleksitet på $O(V(V-1))$ ettersom man for hver node $v \in V$ må flippe$|V|-1$ kanter. Dette er polynomisk, og dermed har vi vist at problemet er i NPC.

Oppgave 27

```python
def verify_tsp(path, max_weight, weight_matrix):
    # Stien må ha lengde på |V| + 1
    if len(path) != len(weight_matrix) + 1:
        return False

    # Stien må starte og slutte i samme node
    if path[0] != path[-1]:
        return False

    # Stien må bestå av unike noder, med unntak av
    # start og sluttnoden
    if len(set(path)) != len(path) - 1:
        return False

    weight = 0
    for i in range(len(path) - 1):
        weight += weight_matrix[path[i]][path[i+1]]

    # Stien må ha vekt lavere eller lik den oppgitte vekten
    return weight <= max_weight
```

Oppgave 28

For eksamensoppgavene, se løsningsforslaget til den gitte eksamenen.

For oppgaver i CLRS, så finnes det mange ressurser på nettet som har fullstendige eller nesten fullstendige løsningsforslag på alle oppgavene i boken. Det er ikke garantert at disse er 100% korrekte, men de kan gi en god indikasjon på om du har fått til oppgaven. Det er selvfølgelig også mulig å spørre studassene om hjelp med disse oppgavene.

Et eksempel på et ganske greit løsningsforslag på CLRS, laget av en tidligere doktorgradsstudent ved Rutgers, finnes her.