Informasjon

Du måtte klare 10 av 18 oppgaver for å få øvingen godkjent.

Finner du feil, mangler eller forbedringer, ta gjerne kontakt!

Oppgave 2

«*Finne stier som minimerer summen av kantvektene.*» er riktig.

Kommentar:
Står på side 643 i læreboka.

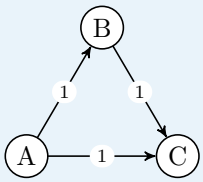
Oppgave 3

Riktige alternativer:

- Ved å snu alle kantene i en graf kan man finn korteste alle-til-én veier, gitt at man vet hvordan man finner korteste en-til-alle veier.
- Selv om man har korteste vei fra node A til B og korteste vei fra node B til C vil man ikke kunne garantere at det finnes en korteste vei mellom A og C som går via B.

Kommentar:
Ved å snu alle kantene i en graf kan man finne korteste alle-til-én veier, gitt at man vet hvordan man finner korteste en-til-alle veier. Dette kan man gjøre, siden etter å snu alle kantene tilbake igjen vil alle stier som gikk fra noden til alle de andre nodene nå gå i motsatt retning. Siden den eneste endringen som er gjort er å snu kantene, så må stiene fortsatt være de korteste.

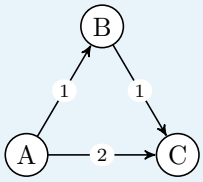
Selv om man har korteste vei fra node A til B og korteste vei fra node B til C vil man ikke kunne garantere at det finnes en korteste vei mellom A og C som går via B. Enkelt eksempel:



Her er korteste vei fra A til B $\{(A, B)\}$ og korteste vei fra B til C $\{(B, C)\}$, men veien $\{(A, B), (B, C)\}$ fra A til C har lengde 2, mens korteste vei $\{(A, C)\}$ har lengde 1.

DFS vil ikke finne korteste vei til alle noder i en graf, siden DFS kan traversere til en node på en vei som ikke er kortest, og kan derfor ikke propagere korteste vei.

Det kan finnes mer enn én korteste vei mellom 2 noder. Enkelt eksempel:



Både veien $\{(A, B), (B, C)\}$ og veien $\{(A, C)\}$ har lengde 2 og er korteste veier fra A til C.

Korteste vei har optimal substruktur om vi har positive kantvekter. Dette er siden en korteste vei fra A til C på formen $A \rightsquigarrow B \rightsquigarrow C$ består av korteste vei fra A til B og korteste vei fra B til C.

Oppgave 4

«*Vi har også korteste vei mellom* A *og* B*, og* B *og* C.» er riktig.

Kommentar:
Hvis dette ikke gjaldt, så hadde det fantes enten en kortere sti mellom A og B eller mellom B og C. Vi kunne da byttet inn denne i stien vår og fått en kortere sti enn den vi allerede hadde funnet som korteste sti. Dette er selvfølgelig ikke mulig når vi allerede har funnet korteste sti. Dette er også lemma 24.1 på side 645 i læreboka.

Oppgave 5

«*Man har ikke algoritmer som i verste tilfelle løser dette problemet raskere enn korteste vei fra én til alle.*» er riktig.

Kommentar:
Se **Single-pair shortest-path problem** på side 644 i læreboka.

«Det er aldri raskere å løse dette enn å løse fra én til alle.» er ikke sant, siden man kan stoppe Dijkstra når man henter målnoden ut av prioritetskøen. Hvis dette er den andre noden man tar ut av køen vil kjøretiden bli $O(V) = o(V \lg V + E)$, som er raskere enn å løse problemet fra én til alle.

«Det er aldri et relevant problem å løse.» er ikke sant heller, siden det er relevant dersom man skal fra et sted til et annet for eksempel i et veinett.

Oppgave 6

Riktige alternativer:

- Vi kan fjerne 0-vektede sykler fra en sti for å lage en ny sti med samme vekt.
- Bellman-Ford finner alltid korteste vei med negative kanter, men ikke negative sykler.

Kommentar:
Vi kan fjerne 0-vektede sykler fra en sti for å lage en ny sti med samme vekt. Dette fordi sykelen har en samlet vekt på 0, dermed vil det å fjerne den ikke påvirke vekten til stien. I tillegg er det en sykel, så man får en gyldig sti etter å ha fjernet denne. Se også side 646 i læreboken.

Bellman-Ford finner korteste vei med negative kanter, men ikke negative sykler. At Bellman-Ford finner korteste vei med negative kanter er teorem 24.4 i læreboka. Hvis det er negative sykler i grafen, vil korteste vei potensielt ha lengde $-\infty$ ved å gå den negative sykelen uendelig mange ganger, men siden Bellman-Ford ikke setter verdien til dette, vil den ikke nødvedigvis gi riktig svar om det er sykler i grafen. Det følger av dette at «Bellman-Ford finner korteste vei med både negative kanter og negative sykler.» er feil.

«Det er umulig å detektere om en graf har negative sykler.» er feil siden Bellman-Ford returnerer true om det ikke er noen negative sykler som kan nås fra s og false ellers. Ved å kjøre Bellman-Ford fra alle nodene, kan man dermed detektere negative sykler i grafen dersom det finnes noen. (Man kan også kjøre Bellman-Ford kun én gang, men da må man initialiserere $v.d = 0$ i stedet for $v.d = \infty$ for $v \in \mathrm{V}$, noe som gjør at man ikke får korteste avstander fra s.)

Både «Dijkstra finner korteste vei med både negative kanter og negative sykler.» og «Dijkstra finner korteste vei med negative kanter, men ikke negative sykler.» er feil, siden Dijkstra ikke nødvendigvis gir riktig svar dersom kantene er negative.

Oppgave 7

$u.d = 3, v.d = 6$ er riktig.

Kommentar:
Relax(u,v,w) setter $v.d = min(v.d, u.d + w(u, v)) = min(7, 3 + 3) = 6$, mens $u.d$ forblir uendret.

Oppgave 8

«*Vi kan ikke vite hva* $u.\pi$ *er gitt infomasjonen i oppgaven, mens* $v.\pi = u$.» er riktig.

Kommentar:
Vi har at $u.\pi$ forblir hva enn den var før kallet, noe vi ikke vet hva var. Siden $v.d > u.d + w(u, v)$ før kallet, blir $v.\pi$ satt til u.

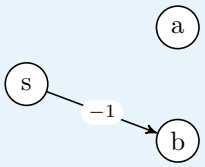
Oppgave 9

Riktige alternativer:

- ∞
- 0

- $\sum_{(u,v)\in \mathrm{P}} w(u,v)$, hvor P er den korteste stien mellom s og a
- -1

Kommentar:
Hvis vi kjører DAG-Shortest-Path på denne grafen fra s:



Vil vi ende opp med $s.d = 0$, $a.d = \infty$ og $b.d = -1$. Videre er -1 lengden på korteste vei fra s til b, så «$\sum_{(u,v)\in \mathrm{P}} w(u,v)$, hvor P er den korteste stien mellom s og a» er et riktig alternativ.

For å se at vi ikke kan ende opp med NIL eller $-\infty$, så må vi se når algoritmene oppdaterer d-attributtene. Dette skjer kun under Initialize-Single-Source og Relax. I Initialize-Single-Source settes attributtene kun til ∞ eller 0. Dette betyr at $u.d \in \mathbb{R} \cup \{\infty\}$ for alle $u \in$ G.V før noen av Relax-kallene. I Relax oppdateres $v.d = min(v.d, u.d + w(u,v))$. Hvis $v.d \in \mathbb{R} \cup \{\infty\}$ og $u.d \in \mathbb{R} \cup \{\infty\}$ før kallet og $w(u,v) \in \mathbb{R}$, vil $v.d = min(v.d, u.d + w(u,v)) \in \mathbb{R} \cup \{\infty\}$ etter kallet, og dermed kan vi gjøre en induktiv slutning om at alle d-attributtene vil forbli i $\mathbb{R} \cup \{\infty\}$. Siden NIL $\notin \mathbb{R} \cup \{\infty\}$ og $-\infty \notin \mathbb{R} \cup \{\infty\}$ kan heller ikke noen av nodene ha det som verdi på d-attributtet når algorimen er ferdig.

Oppgave 10

```
def dfs_visit(task, visited_tasks, topological_order):
    if task in visited_tasks:
        return
    visited_tasks.add(task)
    for t in task.depends_on:
        dfs_visit(t, visited_tasks, topological_order)
    topological_order.append(task)


# Topologisk sorterer oppgavene slik at oppgaver
# kommer etter oppgavene de avhenger av
def toposort(tasks):
    visited_tasks = set()
    topological_order = []
    for task in tasks:
        dfs_visit(task, visited_tasks, topological_order)
    return topological_order


def building_time(tasks):
    finish_times = {}
    # Topologisk sorterer oppgavene
    topological_order = toposort(tasks)
    for task in topological_order:
```

```
        # Tidligst mulige tidspunkt oppgaven kan starte på, som
        # er etter tidligst mulige tidspunkt der alle oppgavene den
        # er avhengig av er ferdigstilte.
        start_time = max([0] + [finish_times[t] for t in task.depends_on])
        finish_times[task] = task.time + start_time
    return max(finish_times.values())
```

Kommentar:
Her var poenget å finne en kritisk sti, som er den lengste stien i en DAG. Kritisk sti er diskutert i læreboka på side 657. Ved å topologisk sortere oppgavene, og deretter la tidligste ferdigtid på oppgavene være lik tidligste ferdigtid på oppgavene den avhenger av pluss hvor lang tid oppgaven tar, vil vi finne tidligste ferdigtid på alle oppgavene.

Som en implementasjonsdetalj, så legger den topologiske sorteringen her nodene bakerst i listen. Det er fordi «kantene» her går til nodene en node avhenger av heller enn motsatt, men siden vi skal ha nodene en node avhenger av først, må vi snu rekkefølgen på den topologiske sorteringen.

Oppgave 11

Riktige alternativer:

- DAG-Shortest-Path har bedre asymptotisk tidskompleksitet enn Dijkstra i verste tilfelle om $|\mathrm{E}| = o(\mathrm{V} \lg \mathrm{V})$.
- DAG-Shortest-Path fungerer også med negative kantvekter, mens Dijkstra gjør ikke det.

Kommentar:
«DAG-Shortest-Path har bedre asymptotisk tidskompleksitet enn Dijkstra i verste tilfelle uansett hva $|\mathrm{E}|$ er.» er feil siden kjøretiden til DAG-Shortest-Path er $\Theta(\mathrm{V}+\mathrm{E})$, mens kjøretiden til Dijkstra (med Fibonacci-haug) er $\mathrm{O}(\mathrm{V} \lg \mathrm{V} + \mathrm{E})$ i verste tilfelle. Hvis $|\mathrm{E}| = \Omega(\mathrm{V} \lg \mathrm{V})$ vil $\Theta(\mathrm{V}+\mathrm{E}) = \Theta(\mathrm{E})$ og $\mathrm{O}(\mathrm{V} \lg \mathrm{V}+\mathrm{E}) = \mathrm{O}(\mathrm{E})$, og dermed er ikke kjøretiden til DAG-Shortest-Path bedre enn kjøretiden til Dijkstra i verste tilfelle.

«DAG-Shortest-Path har bedre asymptotisk tidskompleksitet enn Dijkstra i verste tilfelle om $|\mathrm{E}| = o(\mathrm{V} \lg \mathrm{V})$.» er riktig. For å bevise dette, skal jeg først bevise at Dijkstra (med Fibonacci-haug) har kjøretid $\Omega(\mathrm{V} \lg \mathrm{V})$ i verste tilfelle.

Si at vi har en tabell med tall $\mathrm{A} = \langle a_1, a_2, \ldots a_n \rangle$ og lag en graf med en startnode s og en node u_i for $1 \leqslant i \leqslant n$. La det så gå en kant fra s til u_i med vekt a_i. Gjør om Dijkstra slik at den tar vare på rekkefølgen den tar ut noder. Vi kjører så Dijkstra fra s. Etter Dijkstra har besøkt s vil Q bestå av u_i for $1 \leqslant i \leqslant n$ med $u_i.d = a_i$. Siden det ikke går noen kanter mellom nodene u_i vil ikke $u_i.d$ endre seg. Rekkefølgen vi tar ut nodene u_i på vil være en gyldig sortering av tabellen A, siden man kjører Extract-Min på nodene basert på attributtet $u_i.d = a_i$. Siden vi ikke har gjort noen antagelser om tallverdiene a_i, vet vi at en slik sortering må ta $\Omega(n \lg n)$ tid i verste tilfelle. Det å ta vare på rekkefølgen man tar ut nodene kan for eksempel gjøres i amortisert konstant tid ved å legge de bakerst i en dynamisk tabell. Dermed må det finnes en A slik at Dijkstra bruker $\Omega(n \lg n) = \Omega(\mathrm{V} \lg \mathrm{V})$ tid, noe som betyr at Dijkstra må bruke $\Omega(\mathrm{V} \lg \mathrm{V})$ tid i verste tilfelle.

Kjøretiden til DAG-Shortest-Path er $\Theta(V + E)$, mens kjøretiden til Dijkstra (med Fibonacci-haug) er $\Omega(V \lg V)$ i verste tilfelle. Hvis $|E| = o(V \lg V)$ vil $\Theta(V + E) = o(V \lg V)$, og dermed er kjøretiden til DAG-Shortest-Path bedre enn kjøretiden til Dijkstra i verste tilfelle.

«DAG-Shortest-Path fungerer også med negative kantvekter, mens Dijkstra gjør ikke det.» er riktig. Teorem 24.5 på side 656 i læreboka setter ingen begrensninger på kantvektene for at DAG-Shortest-Path skal fungere. Dijkstra, derimot, fungerer ikke dersom kantvektene er negative.

«Dijkstra fungerer ikke på rettede asykliske grafer, selv om de har positive kantvekter.» er feil. Dijkstra fungerer på rettede grafer (inkludert asykliske grafer) så lenge kantvektene er positive. Dette er teorem 24.6 i læreboka (side 659).

Oppgave 12

«Decrease-Key» er riktig.

Kommentar:
Amortiserte tidskompleksiteter ved verste tilfelle:

	Binærhaug	Fibonacci-haug
Decrease-Key	$\Theta(\lg n)$	$O(1)$
Extract-Min	$\Theta(\lg n)$	$\Theta(\lg n)$
Minimum	$O(1)$	$O(1)$*

(* = Ikke pensum)
Flere av disse står ved O-notasjon i læreboka, men det finnes måter å vise at grensen er stram i verste tilfelle (lage eksempler der det er tilfelle eller reduksjon fra sammenligningsbasert sortering).

Oppgave 13

```
def least_energy(reactions, start, goal, laws_of_thermodynamics=True):
    # En mengde med alle de ulike stoffene
    substances = set([r[i] for i in [0, 1] for r in reactions])
    # Korteste vei (i energi brukt) til stoffet
    distances = {s: float("inf") for s in substances}
    # Forgjenger langs korteste vei
    predecessor = {s: None for s in substances}
    distances[start] = 0
    # Relakserer kantene |V| ganger
    for i in range(len(substances)):
        for a, b, e in reactions:
            if distances[b] > distances[a] + e:
                distances[b] = distances[a] + e
                predecessor[b] = a
    path = []
    # Henter ut korteste vei til "goal" ved hjelp av forgjengere
    p = goal
    path.append(p)
    while predecessor[p] is not None:
```

```
            p = predecessor[p]
            path.append(p)
        return path[::-1]
```

Kommentar:
Klassisk Bellman-Ford-implementasjon.

For å komme på highscore-listen måtte du her løse problemet med korteste enkle sti, som er NP-hardt om man tillater negative sykler. Dette betyr igjen at dere måtte komme med heuristikker eller approksimasjonsalgoritmer. En mulig løsning er en Dijkstra-versjon der man kun relakserer kanter til noder man ikke har besøkt enda.

Oppgave 14

```
from heapq import heappush, heappop


def earliest_arrival(timetable, start, goal):
    # En mengde med de ulike byene
    cities = set([t[0] for t in timetable] + [t[1] for t in timetable])
    # Tidligste tidspunkt man kan komme til en by
    earliest = {i: float("inf") for i in cities}
    earliest[start] = 0
    # Byer vi allerede har besøkt i traverseringen
    visited = set()
    # Lager nabolister av togtabellene
    departures = {i: [] for i in cities}
    for a, b, t_0, t_1 in timetable:
        departures[a].append((b, t_0, t_1))
    heap = []
    heappush(heap, (earliest[start], start))
    while len(heap):
        time, city = heappop(heap)
        if city in visited:
            continue
        visited.add(city)
        # Relakserer langs tog som går etter vi har kommet til byen
        for b, t_0, t_1 in departures[city]:
            if time <= t_0 and earliest[b] > t_1:
                earliest[b] = t_1
                heappush(heap, (t_1, b))
    return earliest[goal]
```

Kommentar:
Dette er en slags Dijkstra-implementasjon. Vi relakserer langs et tog dersom vi kan komme til startbyen før toget har avreise.

Oppgave 15

Du kan oppdatere kantvektene til $w'(u,v) = w(u,v) + \frac{1}{|\mathrm{V}|}$.

Kommentar:
Denne oppgaven er inspirert av oppgave 4a i høsteksamenen 2008. Oppgave 4b i samme eksamenssett er relatert, men litt annerledes. Det anbefales å gjøre denne om du ønsker å trene mer på disse konseptene.

Kantvektene vil fortsatt være positive så DIJKSTRA vil finne korteste vei med de nye kantvektene. La $|\mathrm{P}|$ være antall kanter langs en vei P. Det som så må vises er:

1. At om en vei P var kortere enn en annen vei P' i med de opprinnelige kantvektene, vil P være kortere enn P' med de nye kantvektene. Altså:

$$\sum_{(u,v)\in \mathrm{P}} w(u,v) < \sum_{(u,v)\in \mathrm{P}'} w(u,v) \implies \sum_{(u,v)\in \mathrm{P}} w'(u,v) < \sum_{(u,v)\in \mathrm{P}'} w'(u,v)$$

2. At dersom to veier P og P' var like lange med de opprinnelige kantvektene, men P har færre kanter, må P være kortere enn P' med de nye kantvektene. Dette for at DIJKSTRA må velge P foran P'. Altså:

$$\left(\sum_{(u,v)\in \mathrm{P}} w(u,v) = \sum_{(u,v)\in \mathrm{P}'} w(u,v) \ \wedge \ |\mathrm{P}| < |\mathrm{P}'|\right) \implies \sum_{(u,v)\in \mathrm{P}} w'(u,v) < \sum_{(u,v)\in \mathrm{P}'} w'(u,v)$$

Etter definisjonen av $w'(u,v)$ så

$$\sum_{(u,v)\in \mathrm{P}} w'(u,v) = \sum_{(u,v)\in \mathrm{P}} \left(w(u,v) + \frac{1}{|\mathrm{V}|}\right) = \left(\sum_{(u,v)\in \mathrm{P}} w(u,v)\right) + \frac{|\mathrm{P}|}{|\mathrm{V}|}$$

Å bevise (2) kan vi nå gjøre slik:

$$\begin{aligned}
&\left(\sum_{(u,v)\in \mathrm{P}} w(u,v) = \sum_{(u,v)\in \mathrm{P}'} w(u,v)\right) \ \wedge \ |\mathrm{P}| < |\mathrm{P}'| \\
\implies &\left(\sum_{(u,v)\in \mathrm{P}} w(u,v)\right) + \frac{|\mathrm{P}|}{|\mathrm{V}|} < \left(\sum_{(u,v)\in \mathrm{P}'} w(u,v)\right) + \frac{|\mathrm{P}'|}{|\mathrm{V}|} \\
\implies &\sum_{(u,v)\in \mathrm{P}} w'(u,v) < \sum_{(u,v)\in \mathrm{P}'} w'(u,v)
\end{aligned}$$

For å bevise (1) må vi bruke at vi har heltallsvekter, så

$$\begin{aligned}
&\sum_{(u,v)\in \mathrm{P}} w(u,v) < \sum_{(u,v)\in \mathrm{P}'} w(u,v) \\
\implies &\sum_{(u,v)\in \mathrm{P}'} w(u,v) - \sum_{(u,v)\in \mathrm{P}} w(u,v) \geqslant 1
\end{aligned}$$

Videre kan ikke en korteste vei ha mer enn $|\mathrm{V}| - 1$ kanter, så

$$|\mathrm{P}| \leqslant |\mathrm{V}| - 1 < |\mathrm{V}| \implies |\mathrm{P}| - |\mathrm{P}'| < |\mathrm{V}| \implies \frac{|\mathrm{P}| - |\mathrm{P}'|}{|\mathrm{V}|} < 1$$

Dette gir

$$\begin{aligned}
&\sum_{(u,v)\in \mathrm{P}} w(u,v) < \sum_{(u,v)\in \mathrm{P}'} w(u,v) \\
\implies &\sum_{(u,v)\in \mathrm{P}'} w(u,v) - \sum_{(u,v)\in \mathrm{P}} w(u,v) \geqslant 1 > \frac{|\mathrm{P}| - |\mathrm{P}'|}{|\mathrm{V}|} \\
\implies &\left(\sum_{(u,v)\in \mathrm{P}'} w(u,v)\right) + \frac{|\mathrm{P}'|}{|\mathrm{V}|} - \left(\left(\sum_{(u,v)\in \mathrm{P}} w(u,v)\right) + \frac{|\mathrm{P}|}{|\mathrm{V}|}\right) > 0 \\
\implies &\left(\sum_{(u,v)\in \mathrm{P}} w(u,v)\right) + \frac{|\mathrm{P}|}{|\mathrm{V}|} < \left(\sum_{(u,v)\in \mathrm{P}'} w(u,v)\right) + \frac{|\mathrm{P}'|}{|\mathrm{V}|} \\
\implies &\sum_{(u,v)\in \mathrm{P}} w'(u,v) < \sum_{(u,v)\in \mathrm{P}'} w'(u,v)
\end{aligned}$$

Merk: Det er flere måter å løse denne oppgaven på. En mulighet er å sette oppdaterte kantvekter til $w'(u,v) = w(u,v) \cdot |\mathrm{V}| + 1$.

Oppgave 16

«*Ja*» er riktig.

Kommentar:
Vi må vise at for en vilkårlig korteste vei fra s til t kalt P på formen $\langle (s,u_1),(u_1,u_2),\ldots(u_k,t)\rangle$, vil alle kantene $(u,v) \in \mathrm{P}$ relakseres etter at u har blitt gitt riktig avstandsestimat fra s. Dette kan vises ved at alle kantene fra s relakseres når vi starter, samt at når en kant $(u,v) \in \mathrm{P}$ relakseres vil enten v allerede ha riktig avstandsestimat og ha blitt puttet i køen med det estimatet tidligere, ellers vil den få riktig avstandsestimat nå og så legges i køen. Dermed er vi sikret at kanten som kommer etter (u,v) i P vil relakseres etter at v har fått riktig avstandsestimat. Dermed vil avstandsestimatene bli riktige etter lemma 24.15 på side 650 i læreboka. Videre siden forgjengernoden settes samtidig som avstandsestimatet, vil forgjengernodene representere en korteste vei fra s.

Oppgave 17

Se kommentaren på oppgaven over.

Oppgave 18

«$\Theta(V)$» er riktig.

Kommentar:
INITIALIZE-SINGLE-SOURCE tar $\Theta(V)$ tid. Hvis det ikke går noen kanter ut fra s vil ingen andre noder legges inn i køen, noe som betyr at kodelinjer 7-16 vil kjøre maksimalt én gang. I tillegg kjører alle operajoner på køen i $O(1)$. Derfor blir kjøretiden $\Theta(V)$ i beste tilfelle.

Oppgave 19

«$\Theta(VE)$» er riktig.

Kommentar:
Først skal vi vise at NEW-SSSP er $O(VE)$. Vi har INITIALIZE-SINGLE-SOURCE som tar $\Theta(V) = O(VE)$ tid. Videre har vi at noder kun legges i køen når kanter relakseres. Så om vi har $O(VE)$ kantrelakseringer vet vi at algoritmen har en kjøretid på $O(VE)$. Vi vet at en kant (u, v) relakseres etter at u har fått riktig avstandsestimat. Så langs en korteste vei fra s til en node $v \neq s$ på formen $s \rightsquigarrow u_1 \rightsquigarrow u_2 \rightsquigarrow \cdots \rightsquigarrow u_k \rightsquigarrow v$ vil s relakseres, og deretter legge u_1 i køen. Køen kan ha $O(V)$ noder, så $O(V)$ noder besøkes før kanten (u_1, u_2) relakseres. Hvis ikke u_2 har fått riktig avstandsestimat tidligere og blitt lagt inn i køen da, vil u_2 legges inn i køen nå. Køen vil fortsatt ha $O(V)$ noder. Slik fortsetter det helt til v har riktig avstandsestimat. Siden $O(V)$ distinkte noder besøkes mellom hver gang en node langs korteste vei til v får riktig avstandsestimat, relakseres $O(E)$ kanter mellom hver gang en node langs korteste vei til v får riktig avstandsestimat. Siden lengden på en korteste vei er maksimalt $|V| - 1 = O(V)$ får vi $O(VE)$ kantbesøk, som blir kjøretiden på algoritmen.

Nå skal vi vise at algoritmen er $\Theta(VE)$. Dette gjøres ved å lage et eksempel der $|E|$ er på sitt meste, nemlig $|E| = \Theta(V^2)$ og at det har kjøretid $\Theta(VE)$. La oss ha en graf $G = (V, E)$ der $V = \langle a_1, a_2, \ldots a_n \rangle$. La videre $E = \{(a_i, a_j) | 1 \leqslant i < j \leqslant n\}$, og

$$w(a_i, a_j) = \begin{cases} -1 & i + 1 = j \\ 0 & i + 1 \neq j \end{cases}$$

Da er korteste vei fra a_1 til en node a_i på formen $a_1 \rightsquigarrow a_2 \rightsquigarrow \cdots \rightsquigarrow a_{i-1} \rightsquigarrow a_i$. Vi lar nå nabolistene til en node a_i være sortert i motsatt rekkefølge. Med det menes $\langle a_n, a_{n-1}, \ldots a_{i+2}, a_{i+1} \rangle$.

La så NEW-SSSP kjøre fra a_1. Da vil Q bli $\langle a_n, a_{n-1}, \ldots a_3, a_2 \rangle$. Når vi så skal besøke a_2 vil alle de andre nodene være besøkt én gang. Videre vil hver av nodene a_i ha relaksert a_{i+1} for $2 \leqslant i < n$, så etter vi har besøkt a_2 er $Q = \langle a_n, a_{n-1}, \ldots a_4, a_3 \rangle$. Når a_3 så besøkes neste gang vil alle nodene a_i for $4 \leqslant i < n$ være besøkt, og etter a_3 besøkes vil $Q = \langle a_n, a_{n-1}, \ldots a_5, a_4 \rangle$. Slik fortsetter det.

Her kan man se at en node a_i vil dermed besøkes

$$\begin{cases} 1 & i = 1 \\ i - 1 & i \neq 1 \end{cases}$$

ganger. Videre har en node a_i $(n - i)$ kanter som går ut av noden. Summen av kantbesøk blir

dermed:

$$(n-1)+\sum_{i=2}^{n}(i-1)\cdot(n-i)=(n-1)+\sum_{i=1}^{n}(i-1)\cdot(n-i)=(n-1)+\sum_{i=1}^{n}(ni-i^2-n+i)$$

$$=(n-1)+(n+1)\sum_{i=1}^{n}i-\sum_{i=1}^{n}i^2-n\sum_{i=1}^{n}1=(n-1)+\frac{n(n+1)^2}{2}-\frac{n(n+1)(2n+1)}{6}-n^2$$

$$=n-1+\frac{n^3}{2}+n^2+\frac{n}{2}-\frac{n^3}{3}-\frac{n^2}{2}-\frac{n}{6}-n^2=\Theta(n^3)$$

Dermed må $\Theta(\text{VE})$ være en tett grense for verste tilfelle for NEW-SSSP.

Oppgave 20

«*Algoritmen vil ikke terminere dersom en av de negative syklene er mulig å nå fra s, ellers vil den garentert gi riktig svar.*» er riktig.

Kommentar:
Alle noder u som ikke er mulig å nå fra s vil ha $u.d=\infty$ siden det aldri vil relakseres noen kanter inn til dem. Siden ingen negative sykler er mulig å nå fra s, vil vi få samme resultat hvis vi fjerner kantene i negative sykler. Nå er vi garantert algoritmens korrekthet, siden det ikke er negative sykler i grafen. Dermed vil algoritmen gi riktig svar dersom grafen har negative sykler som ikke er mulig å nå fra s.

Jeg skal nå bevise at algoritmen ikke terminerer om en negativ sykel er mulig å nå fra s. La sykelen være $\text{S}=\langle a_0,a_1,\dots a_{k-1}\rangle$. Vi vet at alle nodene vil bli besøkt fordi NEW-SSSP er en slags utvidelse av en generell traversering, siden når vi treffer på en node u vi ikke har truffet på før vil denne noden ha $u.d=\infty$, og vil legges i køen. Derfor vil vi ha et tidspunkt i algoritmen da alle nodene i S er besøkt og har et avstandsestimat $u.d\neq\infty$. Når en node oppdaterer sitt avstandsestimat, legges den inn i køen. Om algoritmen terminerer, må vi ha $u_i\leqslant u_{i+1 \bmod k}+w(u_i,u_{i+1 \bmod k})$, for om $u_i>u_{i+1 \bmod k}+w(u_i,u_{i+1 \bmod k})$ og algoritmen terminerte, kan ikke kanten $w(u_i,u_{i+1 \bmod k})$ ha blitt relaksert etter avstandsestmatet til u_i oppdaterte seg, noe som ikke går. Vi har altså ulikhetene:

$$u_0.d\leqslant u_1.d+w(u_0,u_1)$$
$$u_1.d\leqslant u_2.d+w(u_1,u_2)$$
$$\vdots$$
$$u_{k-1}.d\leqslant u_0.d+w(u_{k-1},u_0)$$

Hvis vi summerer opp venstre og høyre side av ulikhetene får vi:

$$\sum_{i=0}^{k-1}u_i.d\leqslant\sum_{i=0}^{k-1}u_i.d\ +\ \sum_{i=0}^{k-1}w(u_i,u_{i+1 \bmod k})$$
$$\implies\qquad 0\leqslant\sum_{i=0}^{k-1}w(u_i,u_{i+1 \bmod k})$$

Siden $\sum_{i=0}^{k-1} w(u_i, u_{i+1 \bmod k})$ representerer summen av kantvektene i sykelen S, betyr det at sykelen S ikke er negativ, noe som strider med antagelsen om at S er negativ. Derfor vil ikke algoritmen terminere i dette tilfellet.

Oppgave 21

Hvis vi setter alle kantvektene til -1 vil en hamiltonsti være en enkel sti med lengde $-(|\mathrm{V}|-1)$. Likedan vil en enkel sti i grafen med lengde $-(|\mathrm{V}|-1)$ være en hamiltonsti. Videre kan ingen enkle stier i grafen ha lengde mindre enn $-(|\mathrm{V}|-1)$, siden da må nødvendigvis minst en node bli besøkt mer enn en gang.

En måte å finne ut om grafen har en hamiltonsti er dermed å kjøre SHORTEST-SIMPLE-PATH mellom alle noder i grafen, og se om funksjonen returnerer $-(|\mathrm{V}|-1)$ på minst ett av kallene. Hvis SHORTEST-SIMPLE-PATH(G, w, a, b) returnerer $-(|\mathrm{V}|-1)$ vet vi at det finnes en hamiltonsti i grafen som starter i a og slutter i b.

Hvis vi kun ønsker å kjøre SHORTEST-SIMPLE-PATH én gang, kan vi lage en ny graf $\mathrm{G}' = (\mathrm{V}', \mathrm{E}')$, der $\mathrm{V}' = \mathrm{V} \cup \{u, v\}$ og $\mathrm{E}' = \mathrm{E} \cup \{(u, a) | a \in \mathrm{V}\} \cup \{(a, v) | a \in \mathrm{V}\}$. Alle kantvekter er fortsatt -1. Hvis SHORTEST-SIMPLE-PATH(G', w, u, v) returnerer $-(|\mathrm{V}'|-1)$ betyr det at det finnes en enkel sti på formen $u \rightsquigarrow a \rightsquigarrow \cdots \rightsquigarrow b \rightsquigarrow v$, der den enkle stien $a \rightsquigarrow \cdots \rightsquigarrow b$ må ha en lengde på $-(|\mathrm{V}|-1)$, og dermed være en hamiltonsti i G. Likedan, dersom det er en hamiltonsti i G kalt P fra a til b vil $u \rightsquigarrow a \overset{\mathrm{P}}{\rightsquigarrow} b \rightsquigarrow v$ være en sti i G' med lengde $-(|\mathrm{V}'|-1)$. Igjen kan ingen enkle stier i grafen ha lengde mindre enn $-(|\mathrm{V}'|-1)$, siden da må nødvendigvis minst en node bli besøkt mer enn en gang.

Dette er et frempek til NP-hardhetsbevis som er pensum i forelesning 13. Litt forenklet kan man si at beviset går slik:

1. Vi vet at problemet om en graf har en hamiltonsti er NP-hardt.
2. Vi ser at hvis vi har en graf vi lurer på om har en hamiltonsti, kan vi endre problemet slik at problemet heller blir å finne korteste enkle sti i en graf mellom to noder. Siden vi kan finne en hamiltonsti hvis vi kan finne korteste enkle sti, må problemet med å finne korteste enkle sti være minst like vanskelig som å finne ut om en graf har en hamiltonsti.
3. Siden problemet å finne korteste enkle sti er minst like vanskelig som å finne ut om en graf har en hamiltonsti, og det å finne ut om en graf har hamiltonsti er NP-hardt, må også problemet å finne korteste enkle sti være NP-hardt.

Oppgave 22

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.