Løsningsforslag for utvalgte oppgaver fra kapittel 9

Innhold

9.2–1 Grafer og minne

Minnebruk	Kant	Node
Liste	8 byte	4 byte
Tabell	4 byte	0 byte

Tabellen over gir oss minnebruk for ulike representasjoner av grafer. Vær imidlertid oppmerksom på at tabellrepresentasjonen alltid har et antall kanter lik kvadratet av antall noder, fordi tabellrepresentasjonen lagrer alle *muligheter* for kanter enten de finnes eller ikke.

Oppg	#noder	#kanter	Minne, tabell	Minne, kantliste
a)	20	350	$20^2 \cdot 4 \approx 1,6\text{kB}$	$20 \cdot 4 + 350 \cdot 8 \approx 2,8\text{kB}$
b)	500	2 500	$500^2 \cdot 4 \approx 977\text{kB}$	$500 \cdot 4 + 2\,500 \cdot 8 \approx 21,5\text{kB}$
c)	50 000	10^6	$50\,000^2 \cdot 4 \approx 9,3\text{GB}$	$50\,000 \cdot 4 + 10^6 \cdot 8 \approx 7,8\text{MB}$

Vi ser at det er stor forskjell på hvor mye plass de ulike grafrepresentasjonene trenger i disse tilfellene!

9.2–4 Omvendt graf, G^T

a) For å transponere en graf på listeform lager vi en ny graf, som inneholder kanter i motsatt retning. Vi lager en løkke som går gjennom alle kantene i den gamle grafen, f.eks. med en dobbeltløkke hvor den ytre løkka tar for seg alle nodene og den indre alle kantene ut fra den enkelte noden. For hver kant lages en kant i den nye grafen, i motsatt retning. Kompleksiteten for dette blir $\Theta(N+K)$.

b) Å transponere en tabell er enkelt. Her er en kodebit (java) som «snur» alle kantene i en graf:

```
//Her bruker vi bare én graf, som transponeres.
for (y = 0; y < G.N; ++y) for (x = 0; x < G.N; ++x) {
   Kanttab tmp = G.kant[x][y];
   G.kant[x][y] = G.kant[y][x];
   G.kant[y][x] = tmp;
}
```

Kompleksiteten for dette blir $\Theta\left(N^2\right)$.

En student har også påpekt at vi egentlig ikke trenger foreta oss noe for å «snu» kantene med tabellrepresentasjonen. Vi kan simpelthen bytte om indeksene ved oppslag i tabellen, noe aksessormetodene til graf-klassen kan gjøre for oss i det skjulte. I så fall finner vi den omvendte grafen uten å gjøre annet enn å gi grafklassen beskjed om å transponere fremtidig tilgang til grafen. Da blir kompleksiteten $\Theta(1)$.

9.2–5 Kompleksitet

a) Finne antall kanter ut fra en node x:

Tabell: $\Theta(N)$, Liste: $\Theta(K_x)$ Minst arbeid med liste.

b) Finne antall kanter ut fra hver node i grafen:

Tabell: $\Theta\left(N^2\right)$, Liste: $\Theta(N+K)$ Minst arbeid med liste.

c) Antall kanter inn til en node x:

Tabell: $\Theta(N)$, Liste: $\Theta(N+K)$ Minst arbeid med tabell.

Opphavsrett: Helge Hafting og stiftelsen TISIP

d) Antall kanter inn til hver node i grafen:

Tabell: $\Theta\left(N^2\right)$, Liste: $\Theta\left(N+K\right)$ Minst arbeid med liste.

Merk at «minst arbeid med en liste» ikke alltid betyr asymptotisk mindre arbeid, fordi $K \in O\left(N^2\right)$. Men husk likevel at for mange praktiske grafer, f.eks. veikart, er K proporsjonal med N, ikke N^2.

9.3–2 BFS Kompleksitet

Hvis vi implementerer BFS på en graf representert som tabell, øker kompleksiteten.Den ytre løkka har fortsatt en kompleksitet på $\Theta\left(N\right)$. Den indre løkka går gjennom én nodes kanter og får nå kompleksiteten $\Theta\left(N\right)$, til sammen blir kompleksiteten $\Theta\left(N^2\right)$.

9.5–1 Topologisk sortering

Her kan det være mange korrekte løsninger. For å se om din alternative løsning er korrekt, kan du tegne inn kantene også og sjekke at ingen piler går oppover i den sorterte grafen.

a) Mikrobrytere
Ledninger
Vifter
RAM
CPU
Hovedkort
Skjermkort
Strømforsyning
Kabinett
Mus
Harddisk
Tastatur
Ferdig pc

b) s, a, b,c, d. Her fins ingen andre løsninger.

9.6–2 Veikart

a) veigrafen er sterkt sammenhengende

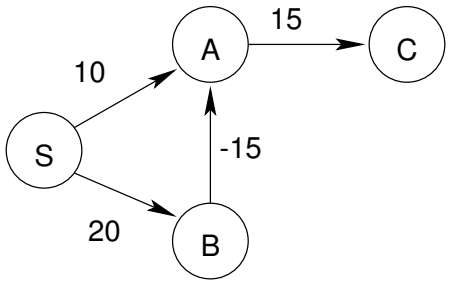
I så fall kan vi komme oss til ethvert sted fra ethvert annet sted uten å bryte trafikkreglene. I praksis er alle byer slik.

Opphavsrett: Helge Hafting og stiftelsen TISIP

b) veigrafen består av flere sterkt sammenhengende komponenter

I så fall kan vi alltid finne veien innenfor hver enkelt sterkt sammenhengende komponent. Men kjører vi ut av en sterkt sammenhengende region kan vi ikke komme tilbake igjen uten å bryte trafikkreglene. Det kan også forekomme minst én sterkt sammenhengende regioner som vi ikke kan komme ut av i det hele tatt, som f.eks. region 0 og region 5 i figur 9.11(a), eller regionen med nodene 1, 2 og 7 i figur 9.11(b) på side 188 i læreboka.

9.8–1 Negativ kant og Dijkstras algoritme



Dijkstras algoritme starter med startnoden S som får distansen 0. Deretter finner algoritmen ut at avstanden til A er 10, og avstanden til B er 20, ved å bruke kantene ut fra S.

Så er algoritmen ferdig med S, og tar for seg den noden som har lavest distanseestimat. Det er A. Ettersom A har avstand 10 og kanten (A,C) har lengde 15, blir avstanden til C satt til $10+15=25$.

Deretter tar algoritmen for seg node B. Dens distanse er 20, og kanten (B,A) har lengden -15. Dermed forandres A sitt lengdeestimat, det blir $20-15=5$. Dermed oppstår problemer, fordi algoritmen nå har forandret på en node som allerede har vært brukt. Det hadde ikke kunnet skje hvis det ikke hadde vært for den negative kanten, uten den kunne ikke $\langle S,B,A\rangle$ vært kortere enn $\langle S,A\rangle$ samtidig som $\langle S,B\rangle$ er lengre enn $\langle S,A\rangle$.

Til slutt tar algoritmen for seg node C som ikke har noen utgående kanter, og blir deretter ferdig. Distansen for C blir stående på 25, men vi ser jo lett at lengden av $\langle S,B,A,C\rangle = 20$. Dijkstras algoritme feilet, fordi forutsetningen om ingen negative kanter ble brutt.

Opphavsrett: Helge Hafting og stiftelsen TISIP

9.8–2 Dijkstras algoritme

Utgangspunkt Trondheim

By	Avstand	Vei
Trondheim	0 km	
Oslo	496 km	Trondheim–Oslo
Bergen	657 km	Trondheim–Bergen
Kristiansand	817 km	Trondheim–Oslo–Kristiansand
Stavanger	835 km	Trondheim–Bergen–Stavanger

Utgangspunkt Stavanger

By	Avstand	Vei
Trondheim	835 km	Stavanger–Bergen–Trondheim
Oslo	565 km	Stavanger–Kristiansand–Oslo
Bergen	178 km	Stavanger–Bergen
Kristiansand	244 km	Stavanger–Kristiansand
Stavanger	0 km	

Det er åtte andre korrekte svar, alt ettersom hvilke byer en velger som startpunkt.

9.8–7 Reiseplanlegging

Her foreslår jeg å bruke en graf, hvor hvert reisemål er en node, og hver direkte reiserute er en kant. For hver kant (reisemulighet) lagres informasjon om hvor lang tid reisen tar, hvor mye den koster, og hva slags transport som brukes.

For å planlegge reiser brukes en lett modifisert Dijkstras algoritme, med utgangspunkt i startstedet. Hvis kunden ønsker raskeste reisemåte, ser algoritmen på «tid» som kantvekt. Hvis kunden derimot ønsker en billigst mulig tur ser algoritmen på «pris» i stedet for tid. Hvis kunden f.eks. ikke vil reise med fly, ser algoritmen bort fra alle kanter av «fly»-typen. Tilsvarende hvis kunden ikke ønsker båt, tog eller buss.

9.9–1 Prims algoritme og kjøretid

De to variantene av Prims algoritme har kjøretidene $O\left(N^2\right)$ og $O\left((N+K)\log N\right)$. For ulike grafer velger vi den varianten som har lavest kjøretid. Hvis grafen er *komplett* eller nesten komplett, har vi $K \in O\left(N^2\right)$. Da er den første varianten asymptotisk raskest, fordi den andre varianten får kompleksiteten $O\left(\left(N+N^2\right)\log N\right) \in O\left(N^2 \log N\right)$. Dette er opplagt verre enn $O\left(N^2\right)$. For en *glissen* graf har vi $K \in O(N)$. Da er det andre alternativet best, fordi det blir til $O\left((N+N)\log N\right) \in O(N \log N)$ som er bedre enn $O\left(N^2\right)$.

Opphavsrett: Helge Hafting og stiftelsen TISIP

9.9–2 Kompleksitetsberegning for Prims algoritme

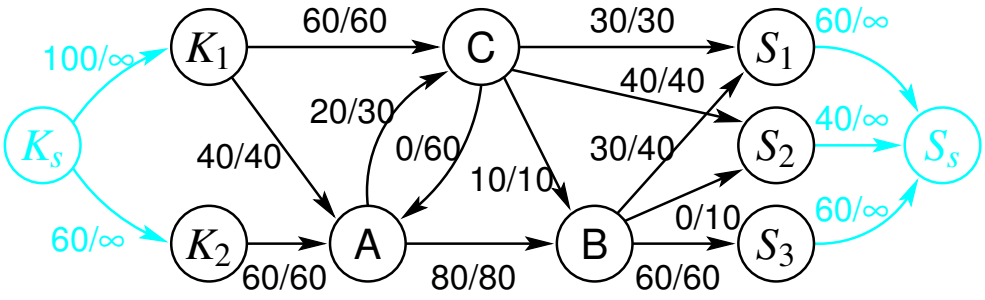
$O((N+K)\log N)$ kan forenkles til $O(K\log N)$ for sammenhengende grafer, fordi $K \geqslant N-1$ i en sammenhengende graf. Det vil si at K enten er proporsjonal med N, eller asymptotisk større. Det kan vi også skrive som $N \in O(K)$, eller at K er en øvre grense for N. Når N ikke kan være asymptotisk større enn K, kan vi som kjent fjerne N fra uttrykket $K+N$.

Strengt tatt *kan* K være mindre enn N i en sammenhengende graf, men altså ikke *asymptotisk* mindre.

9.9–3 Prims og Kruskals algoritmer

Prims og Kruskals algoritme er nødt til å gi samme resultat hvis det bare er ett minimalt spenntre i grafen. Resultatet kan bli ulikt bare når det er flere ulike minimale spenntrær. Enhver urettet graf med bare en eller to noder vil gi samme resultat med begge algoritmer.

9.10–1 Maksimal flyt

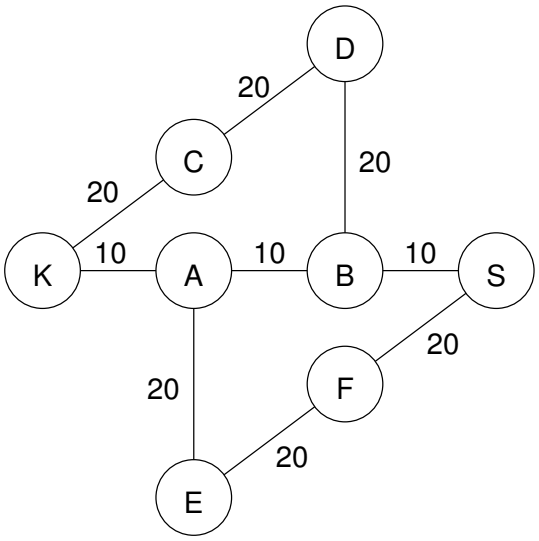


Som vi ser er den maksimale flyten lik 160.

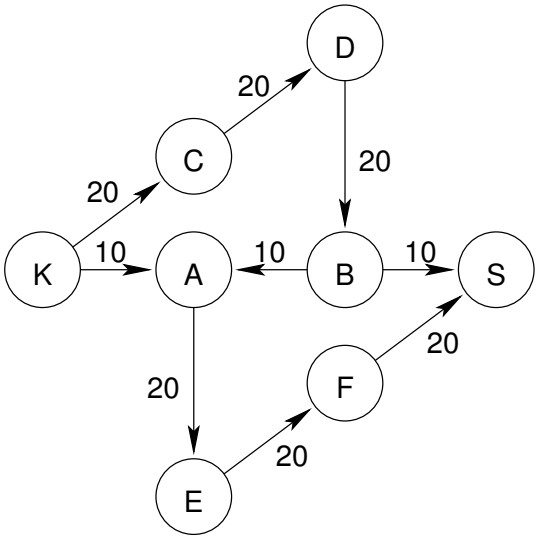
9.10–2 Edmonds-Karp

For å løse denne oppgaven må vi lage en graf der en kant ligger på to flytøkende veier med ulik lengde. Dessuten må kanten forekomme i ulik retning i de to veiene. En kant som skal brukes i to retninger kan ikke forbindes direkte med kilde eller sluk, så denne korteste flytøkende veien må inneholde minst to noder i tillegg til kilde og sluk. Her er en løsning:

Opphavsrett: Helge Hafting og stiftelsen TISIP



Edmonds Karp-algoritmen bruker de korteste flytøkende veiene først. Den korteste veien fra kilde til sluk her er $\langle K,A,B,S\rangle$ som har kapasitet 10. Etter at vi har brukt denne veien forsvinner kantene (K,A), (A,B) og (B,S) fra restnettet, fordi all kapasiteten er brukt opp. Men kantene (A,K), (B,A) og (S,B) finnes fortsatt, og har nå kapasitet 20 fordi det går an å kansellere flyt. Det er mulig å finne én ny flytøkende vei gjennom restnettet, og det er $\langle K,C,D,B,A,E,F,S\rangle$. Denne veien har kapasitet 20, fordi alle kantene i den, inkludert (B,A) har kapasitet 20. Når vi øker flyten langs denne veien med 20 får vi en total flyt på 30 gjennom grafen. Legg merke til at flyten gjennom (B,A) blir snudd. flyten går ikke over kapasiteten på 10, men *endringen* er på 20 fordi vi går fra en flyt på -10 til $+10$. Om noen er i tvil, slik blir flyten til slutt:



Opphavsrett: Helge Hafting og stiftelsen TISIP

9–4 Edmonds-Karp og Dijkstra

Dette var et lurespørsmål. Dijkstras algoritmen kan naturligvis ikke brukes på flytgrafer, av den enkle grunn at kantene ikke har noen *lengde*. Kantvekten i slike grafer er *kapasitet*. Dijkstras algoritme kunne forsåvidt funnet veien med lavest sum av kapasiteter for oss, men det er helt uinteressant og gjør ihvertfall ikke algoritmen noe raskere.

Opphavsrett: Helge Hafting og stiftelsen TISIP