Institut for Matematik og Datalogi
Syddansk Universitet, Odense

27. maj 2020

DM507 — Algoritmer og Datastrukturer

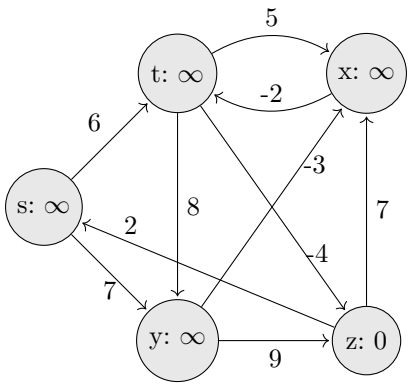
Eksaminatorie-timer uge 21, Forår 2020

Instruktorerne for DM507

Indhold

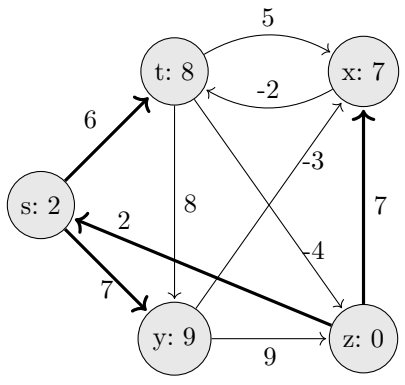
1 Cormen et al. øvelse 24.1-1 (side 654)

Vi skal her kigge på grafen i figur 24.4 fra side 652 i [1], hvor vi skal bruge Bellman-Ford-Moore algoritmen til at finde alle korteste veje fra z til de andre knuder. Algoritmen kalder først `INITIALIZE-SINGLE-SOURCE` og sætter start værdierne som kan ses i figuren nedenunder.



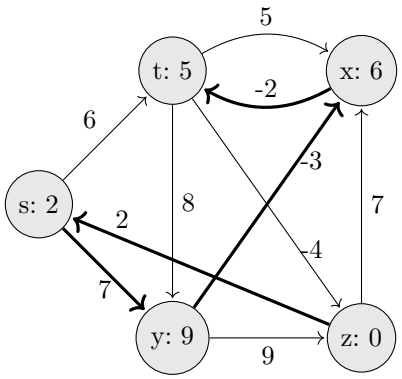
Figur 1: Grafen fra figur 24.4 i Cormen et al.

Herefter skal vi bruge `RELAX` på kantene i samme rækkefølge som de gør i eksemplet tilknyttet grafen i figur 24.4. For en kant (u, v) opdaterer vi $v.d$ og $v.\pi$, hvis $u.d$ + vægten for kanten (u, v) er mindre end den nuværende værdi af $v.d$. Når vi gennemgår kanterne første gang vil der ikke ske noget ved kanterne (t,x), (t,y), (t,z), (x,t), (y,x) og (y,z), da kanterne går fra en knude u med $u.d = \infty$ (vi kan springe disse over da $u.d + w(u, v)$ aldrig vil være mindre end $v.d$). Så kommer vi til kanten (z,x), hvor z.d + kantens vægt ikke er uendelig og vi sætter derfor d værdien for x til $z.d + w(u, v) = 0 + 7 = 7$ samt $x.\pi$ til at være z. Samme proces sker for s med kanten (z,s), hvor $s.d$ sættes til 2 og $s.\pi$ til z. Ved kanten (s,t) kan vi nu nu opdatere t.d til at være 8, da s.d tidligere blev sat til 2. Det samme sker for y.d ved den sidste kant (s,y) som sat til 9. Grafen kommer der for til at se ud som i følgende figur, hvor forgængeren π er vist ved at gøre kanten som førte til ændringen af π ekstra tyk.



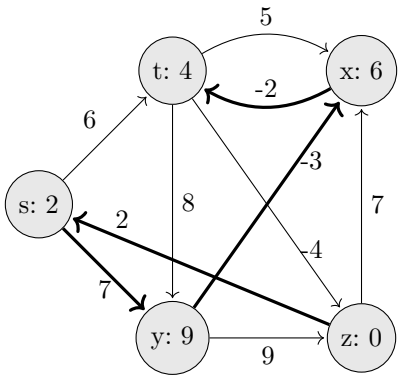
Figur 2: Efter første gennemgang af kanterne

Vi skal nu gennemgå alle kanterne en gang til. Her opdages det at man med kanten (y, x) kan gå til x med en sti af længde $y.d + (-3) = 9 - 3 = 6$ og vi opdaterer x derefter.



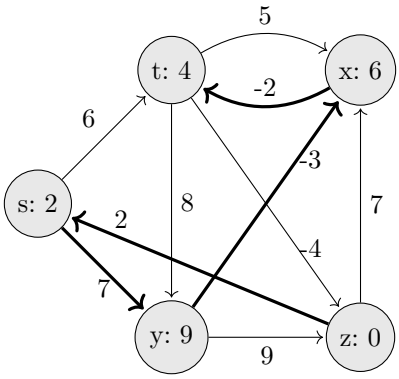
Figur 3: Efter anden gennemgang af kanterne

Vi gennemgår kanterne en tredje gang. Her opdages det, at fordi man opdaterede stiens længde til x i sidste iteration, så kan man med kanten (x, t) gå til t med en sti af længde $x.d + (-2) = 6 - 2 = 4$ og vi opdaterer t derefter.



Figur 4: Efter tredje gennemgang af kanterne

Dette var sidste iteration, da kanterne skal gennemgås $|V| - 1 = 4$ gange (antallet af knuder - 1).



Figur 5: Efter fjerde gennemgang af kanterne

Til sidst skal vi tjekke for hver kant (u, v) at $v.d$ er mindre end eller lig med $u.d$ + vægten for kanten (u, v) for ellers skal vi returnere false. Det kan man verificere holder i figur 5 og algoritmen returnerer derfor true (der er ingen negative kredse i grafen).

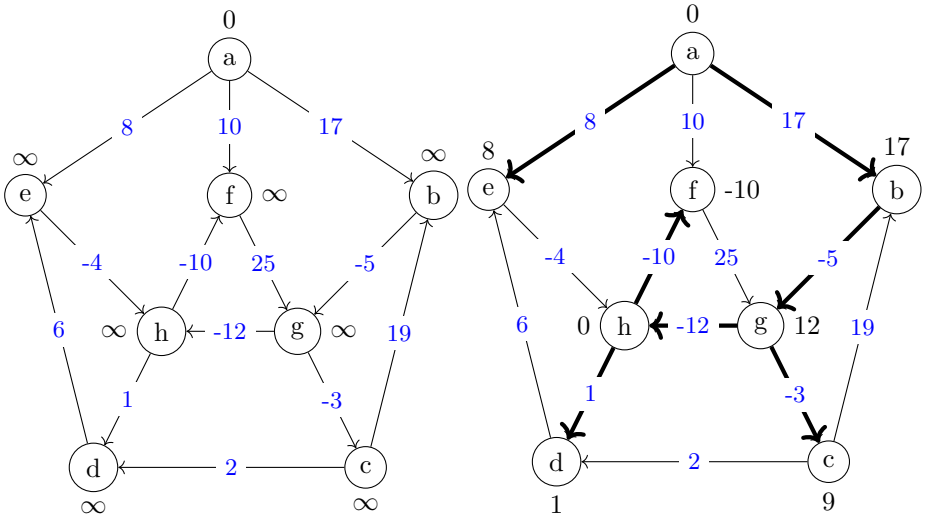
2 Eksamen juni 2012, opgave 4, dog ikke spørgsmål b.

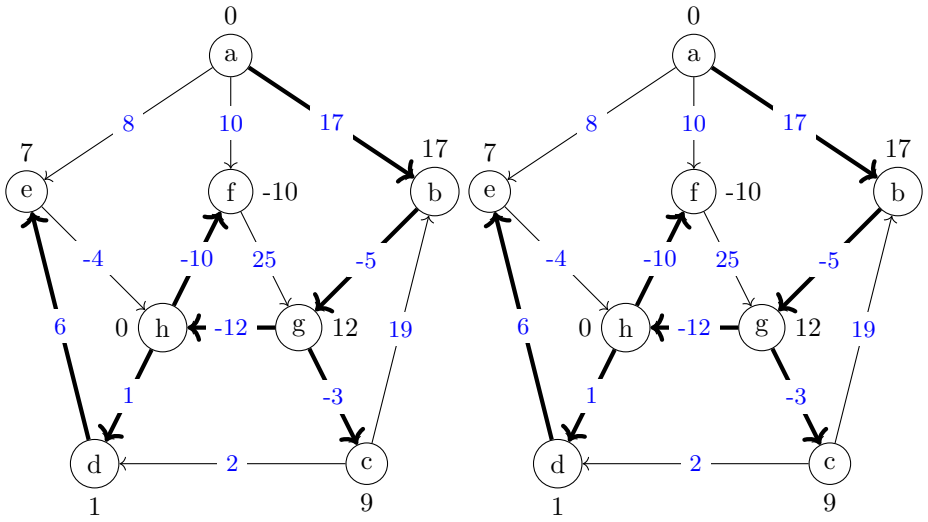
Spørgsmål a

Vi følger pseudokoden som beskrevet på Rolfs slides[3, p. 8]. Følgende relaxeringsrækkefølge er brugt:

$$(a, b), (a, e), (a, f), (b, g), (c, b), (c, d), (d, e), (e, h), (f, g), (g, c), (g, h), (h, d), (h, f)$$

Hver figur er vist før en iteration i den første for-løkke, hvor en knudes distance (d) er skrevet ved siden af den, og en tyk streg kant (u,v) markerer at knude u er v's predecessor (v.π=u):

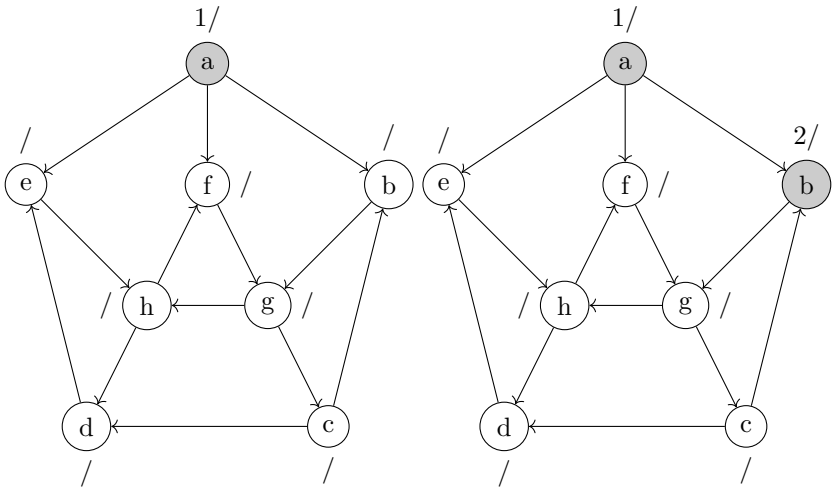


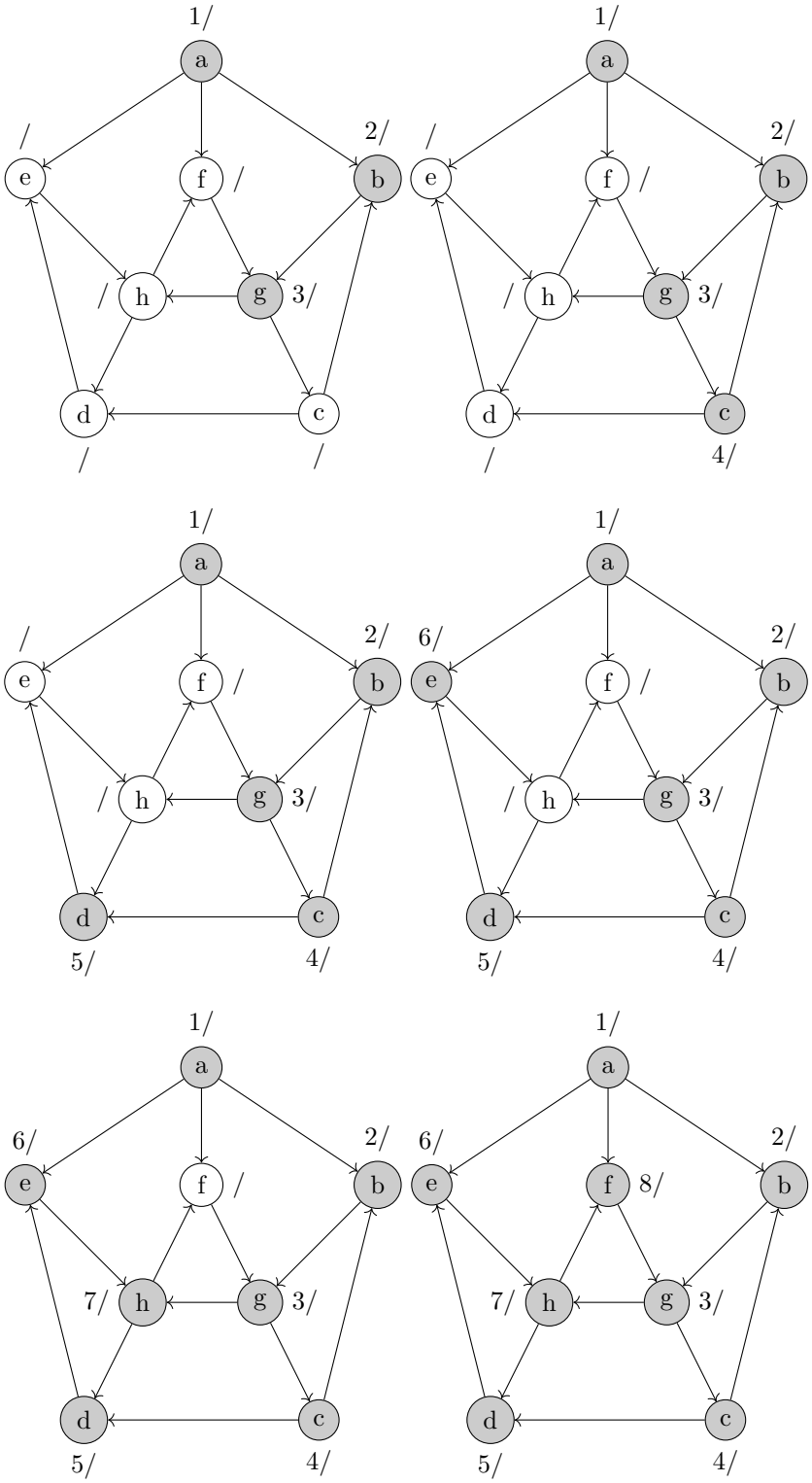


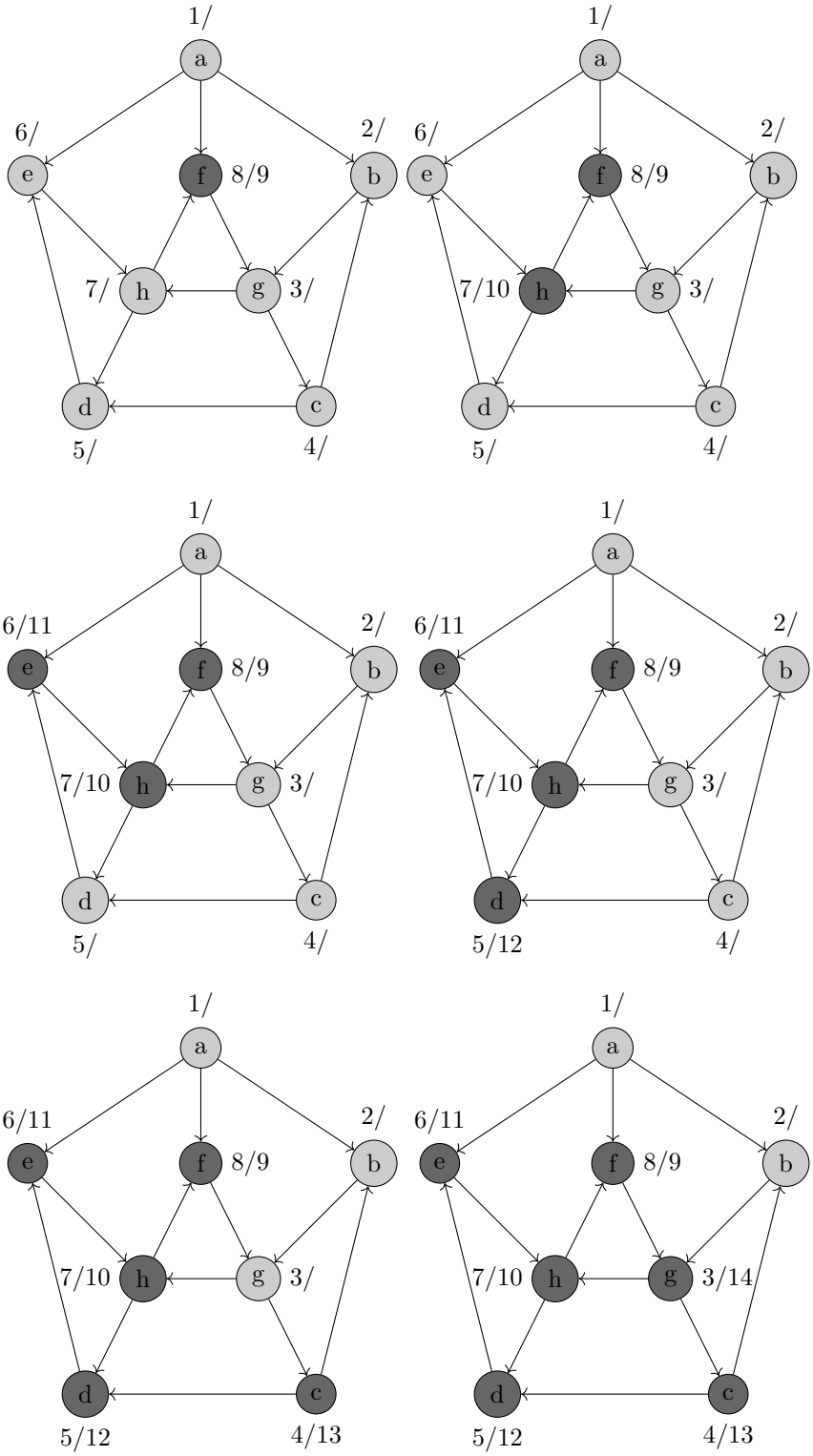
For-løkkens relax afhænger af to ting, knudernes distance (d) og kanternes vægte. Da ingen af disse to ting har ændret sig efter sidste iteration af for-løkken, så kan vi være sikre på, at grafen ikke vil ændre sig under de restende iterationer.

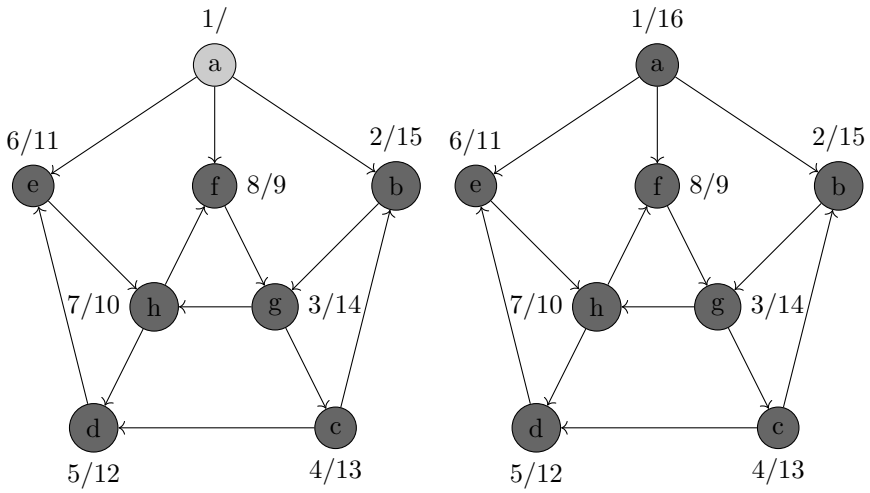
Spørgsmål c

Vi følger pseudokoden som beskrevet på Rolfs slides[2, p. 20]. Hver figur er vist efter en knude farves grå (og får en starttid), eller efter en knude får en sluttid (og farves sort):




1/
a
/
2/
e
f
/
b
/
h
g
3/
d
c
/
/
1/
a
/
2/
e
f
/
b
/
h
g
3/
d
c
/
4/
1/
a
/
2/
e
f
/
b
/
h
g
3/
d
c
5/
4/
1/
a
6/
2/
e
f
/
b
/
h
g
3/
d
c
5/
4/
1/
a
6/
2/
e
f
/
b
7/
h
g
3/
d
c
5/
4/
1/
a
6/
2/
e
f
8/
b
7/
h
g
3/
d
c
5/
4/

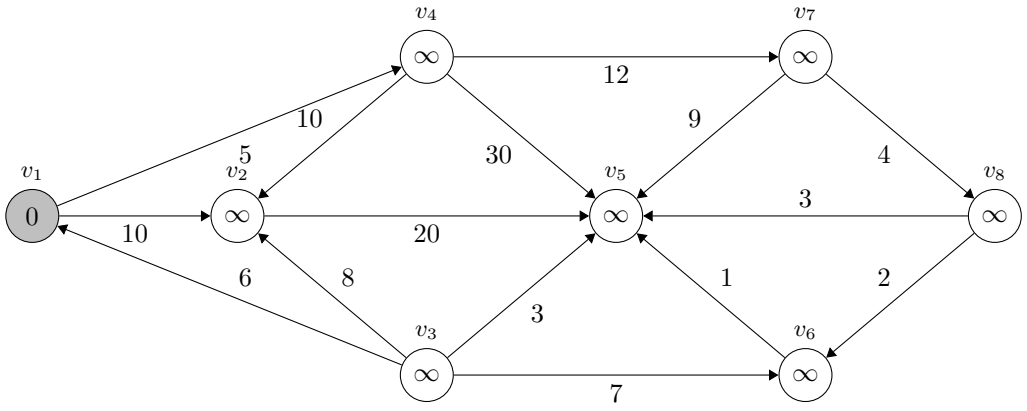

1/
a
6/
e
f
8/9
2/
b
7/
h
g
3/
d
5/
c
4/
1/
a
6/
e
f
8/9
2/
b
7/10
h
g
3/
d
5/
c
4/
1/
a
6/11
e
f
8/9
2/
b
7/10
h
g
3/
d
5/
c
4/
1/
a
6/11
e
f
8/9
2/
b
7/10
h
g
3/
d
5/12
c
4/
1/
a
6/11
e
f
8/9
2/
b
7/10
h
g
3/
d
5/12
c
4/13
1/
a
6/11
e
f
8/9
2/
b
7/10
h
g
3/14
d
5/12
c
4/13

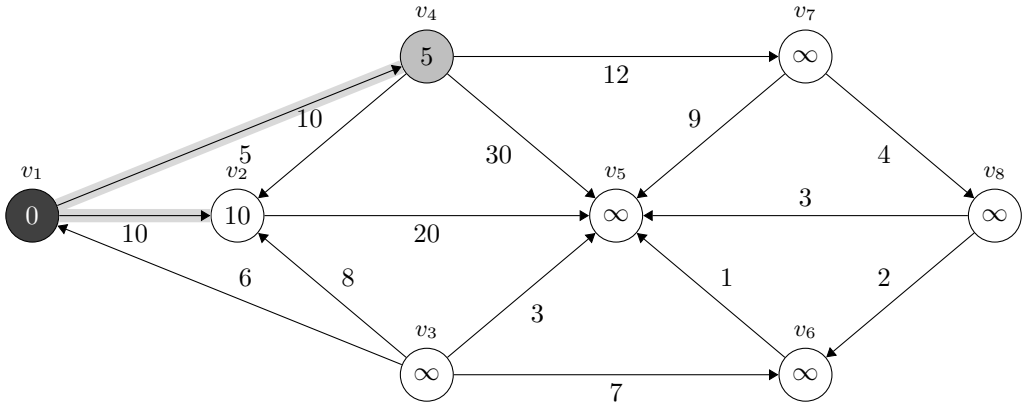


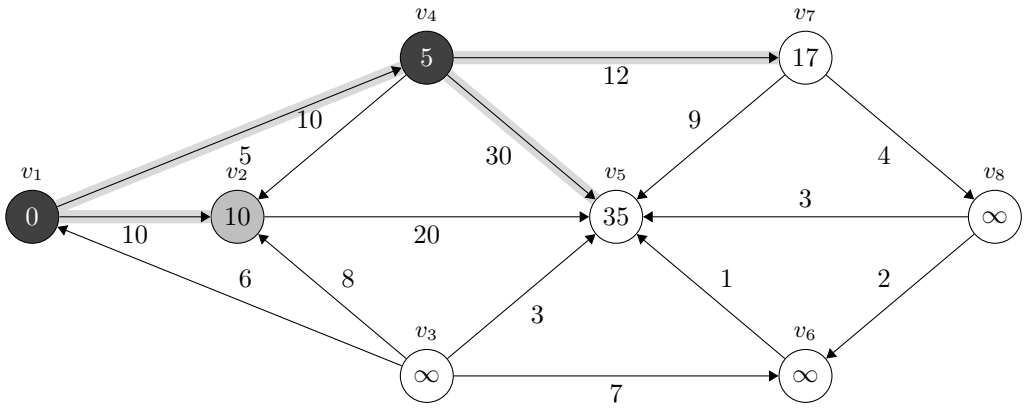
3 Eksamen juni 2010, opgave 2, spørgsmål c.

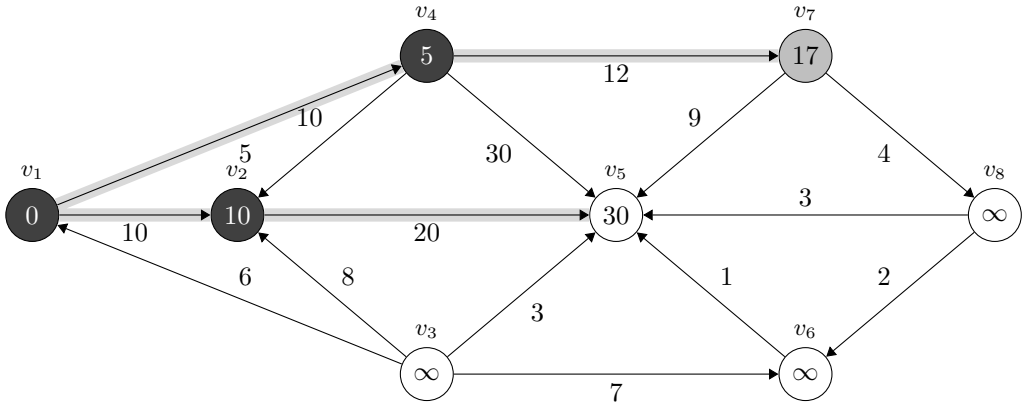
I følgende opgave anvendes Dijkstras algoritme på nedenstående graf $G = (V, E)$, startende i knuden $v_1 \in V$. De følgende figure illustrerer forløbet af Dijkstras algoritme og følger beskrivelsen givet [1, s. 659], Figure 24.6, dvs. vi har at:

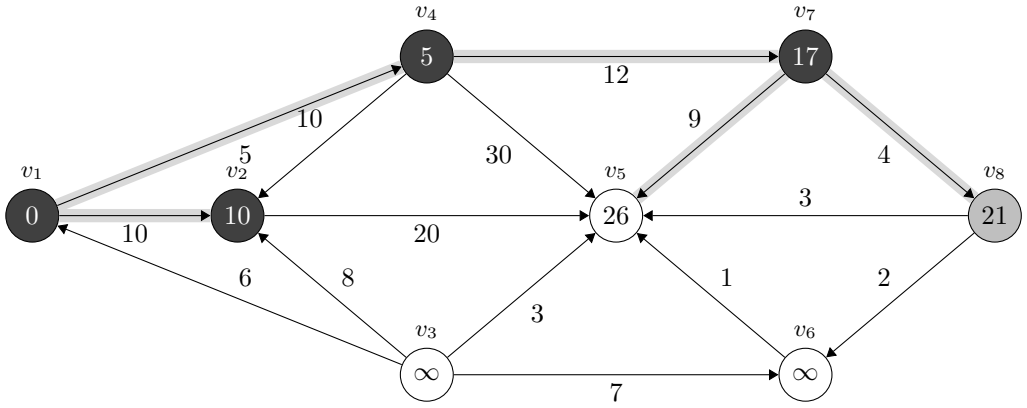
- De korteste sti-estimater forekommer inde i knuderne.
- De grå-skraverede kanter indikerer forgængere (predecessor values).
- De sorte knuder indikerer knuder hvis endelige korteste sti-vægte fra knuden v_1 er bestemt. Disse knuder findes i mængden S, beskrevet i Dijkstras algoritme i sektion 24.3, Cormen et al.
- De hvide knuder indikerer knuder som stadig befinder sig i prioritetskøen $Q = S - V$.
- Den grå-skraverede knude indikerer den knude der på nuværende tidspunkt har den mindste vægt og som udtages af prioritetskøen Q.

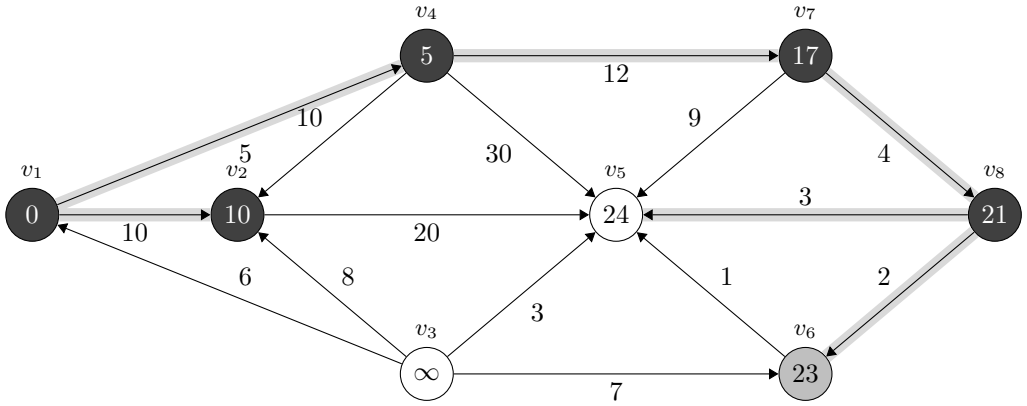


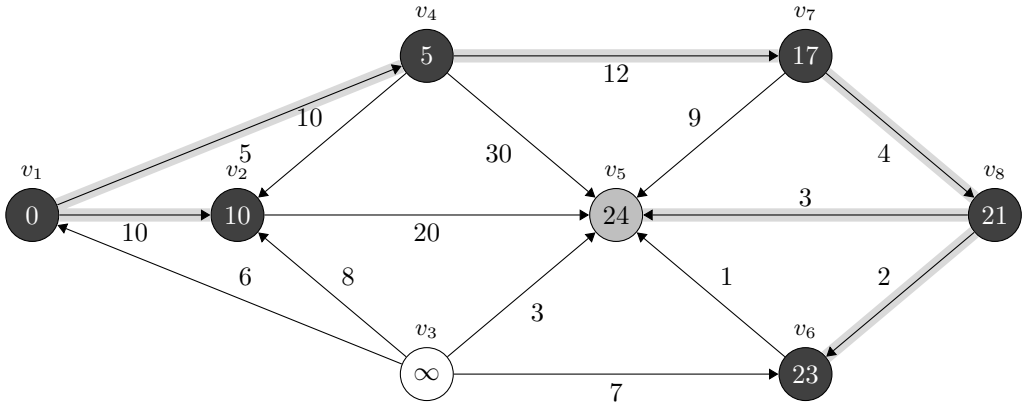

v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
∞
∞
∞
∞
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7

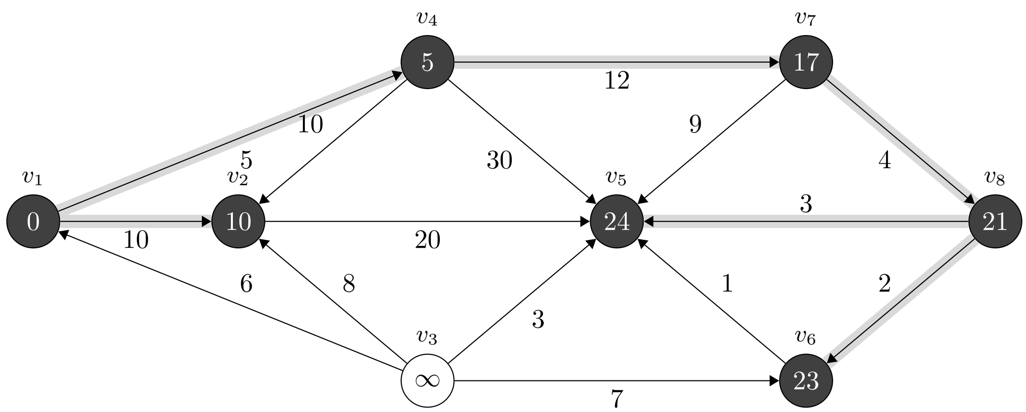

v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
35
∞
17
∞
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7

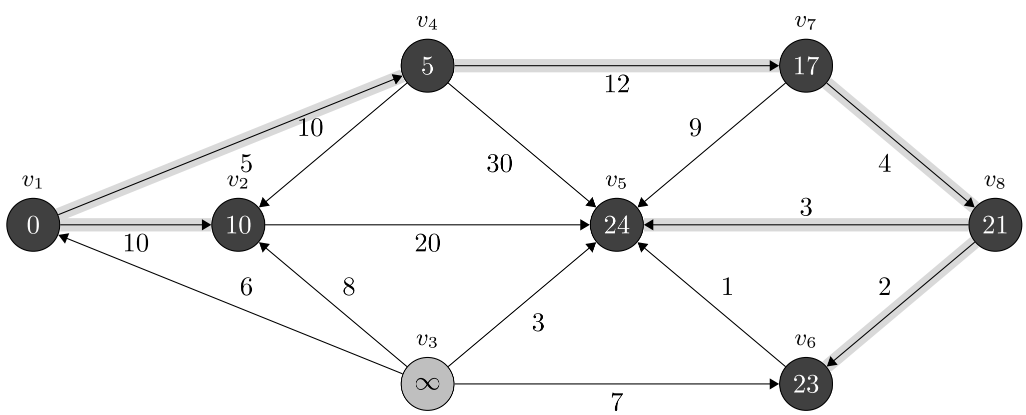

v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
30
∞
17
∞
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7

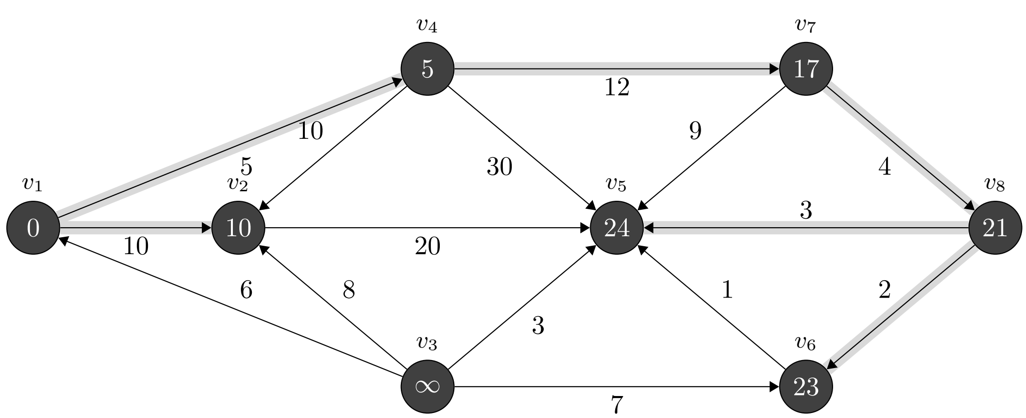

v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
26
∞
17
21
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7


v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
24
23
17
21
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7


v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
0
10
∞
5
24
23
17
21
10
5
30
12
9
4
3
10
20
6
8
3
1
2
7





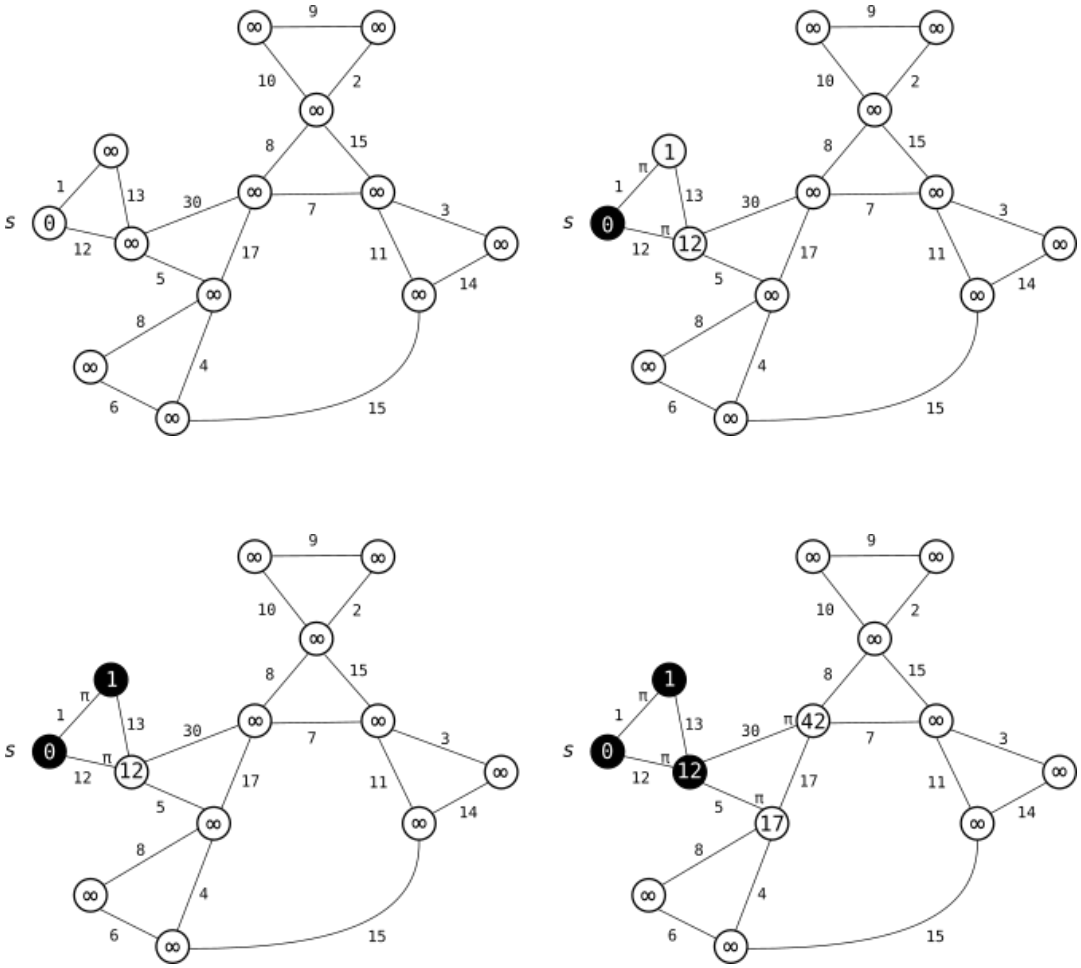


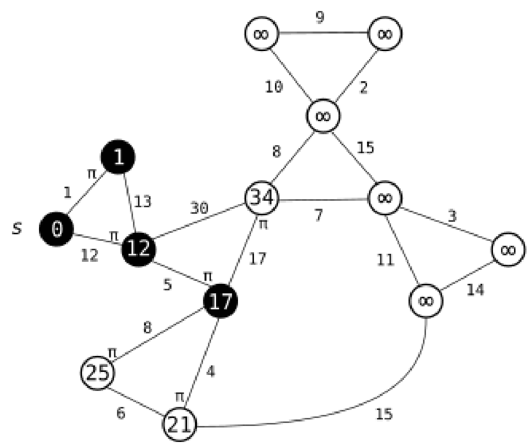
4 Eksamen januar 2008, opgave 2, spørgsmål c

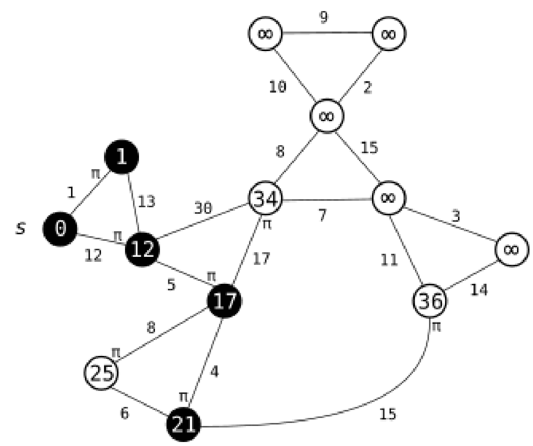
Tegn et korteste-vej træ med rod i den sorte knude s. Dvs. tegn de korteste veje fra s til hver af de andre knuder i grafen. Giv også $v.d$-værdierne. Husk at argumentere for, at dit resultat

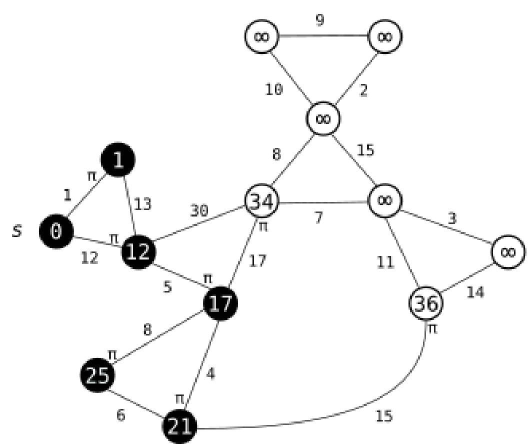
er rigtigt.

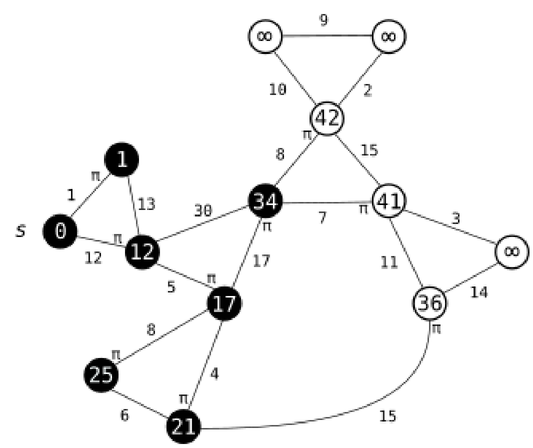
Vi bruger Dijkstras algoritme til at bestemme *single-source shortest-path* fra knuden s til alle andre knuder. Udførslen af Dijkstras algoritme er angivet i figurene nedenunder, hvor en knude er sort hvis den er i mængden S, som vedligeholdes under algoritmens udførsel. Derudover, for en knude $v \in G.V$ er $v.d$ angivet inden i knuden og $v.\pi$ er angivet med π ved kanten, der er incident med v og $v.\pi$.

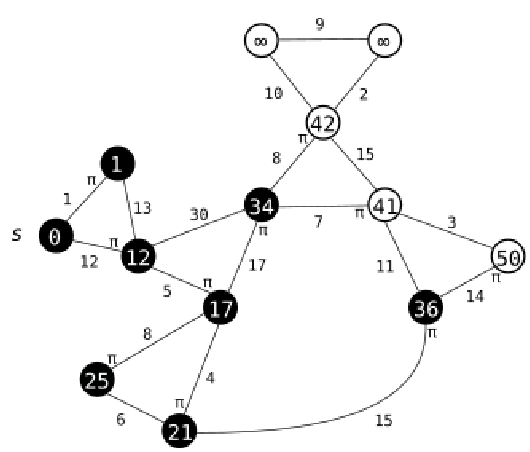


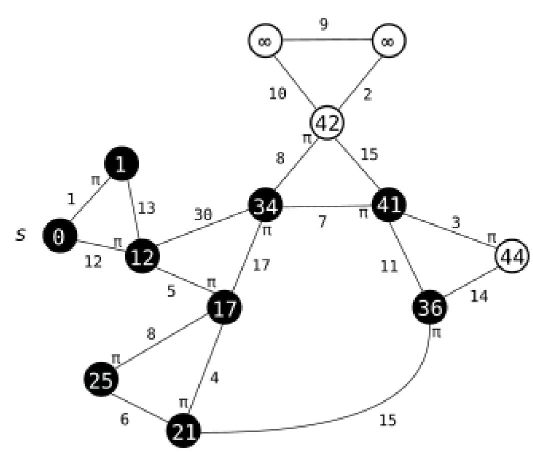

9
∞ ∞
10 2
∞
1 8 15
π 13 30
s 0 12 34 7 ∞ 3 ∞
π π
12 5 π 17 11 14
17 ∞
8
π 4
25
6 π 21 15

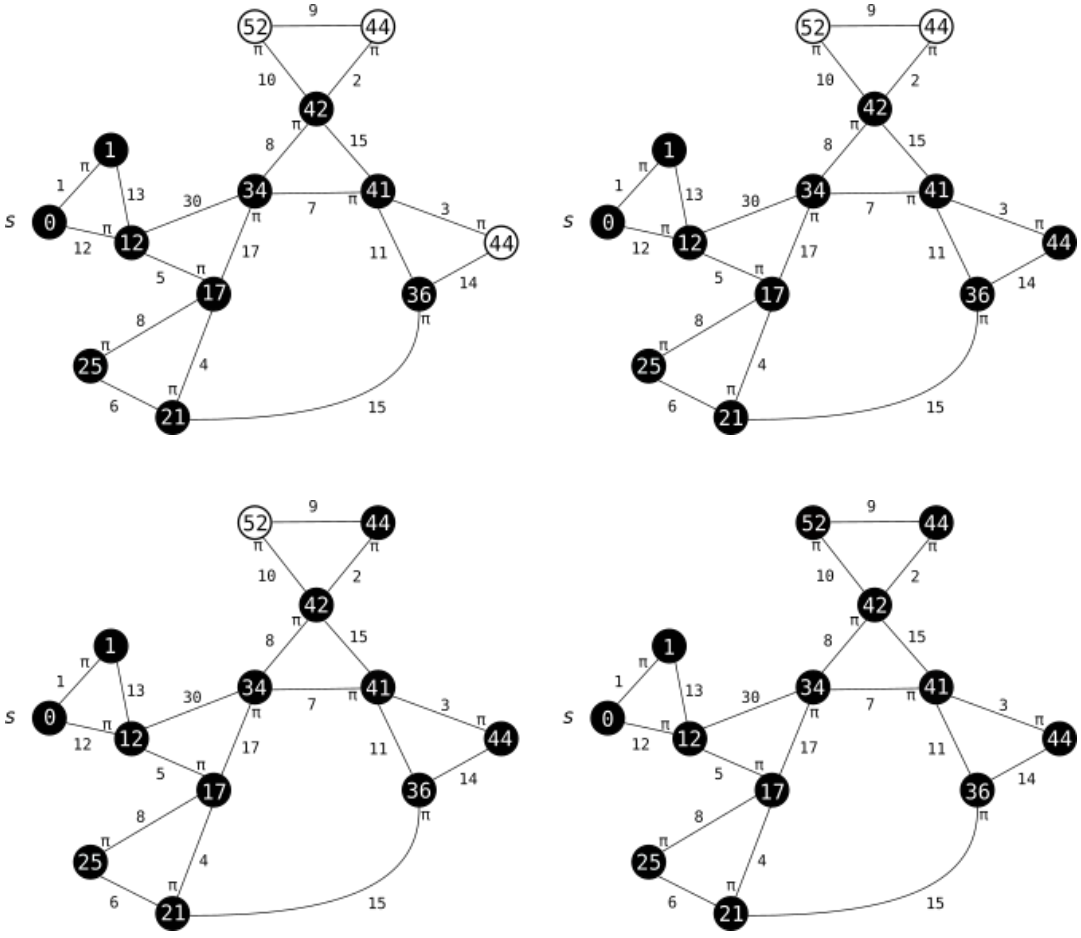

9
∞ ∞
10 2
∞
1 8 15
π 13 30
s 0 12 34 7 ∞ 3 ∞
π π
12 5 π 17 11 14
17 36
8 π
π 4
25
6 π 21 15


9
∞ ∞
10 2
∞
1 8 15
π 13 30
s 0 12 34 7 ∞ 3 ∞
π π
12 5 π 17 11 14
17 36
8 π
π 4
25
6 π 21 15

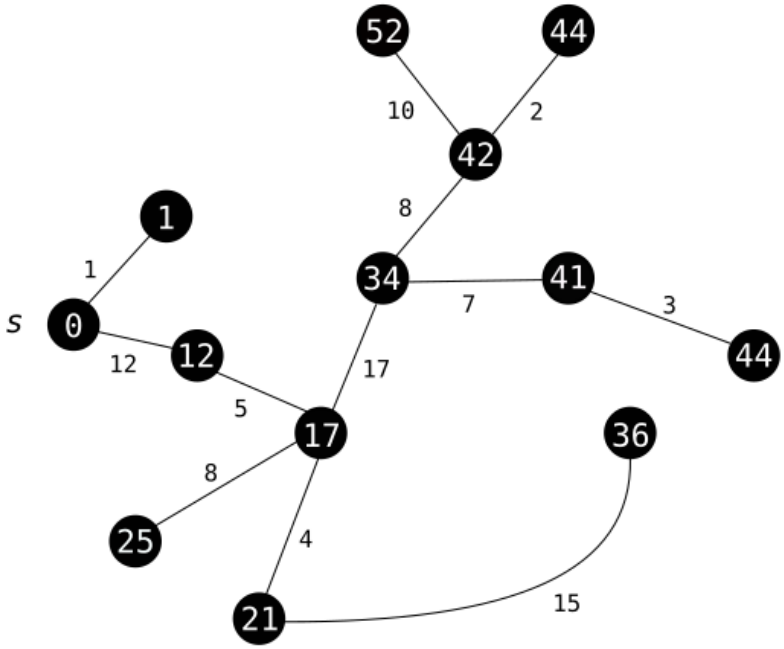

9
∞ ∞
10 2
42
π
1 8 15
π 13 30
s 0 12 34 7 π 41 3 ∞
π π
12 5 π 17 11 14
17 36
8 π
π 4
25
6 π 21 15


9
∞ ∞
10 2
42
π
1 8 15
π 13 30
s 0 12 34 7 π 41 3 50
π π
12 5 π 17 11 π 14
17 36
8 π
π 4
25
6 π 21 15


9
∞ ∞
10 2
42
π
1 8 15
π 13 30
s 0 12 34 7 π 41 3 π 44
π π
12 5 π 17 11 14
17 36
8 π
π 4
25
6 π 21 15



Ud fra $v.\pi$ kan vi nu bestemme korteste-vej-træet ved at beholde de kanter, som indgår i en korteste vej fra startknuden s til en anden knude. Dette træ er angivet nedenunder:



Korrektheden følger af korrektheden af Dijkstras algoritme.

5 Cormen et al. øvelse 24.3-6 (side 663)

I denne opgave skal vi finde den mest holdbare sti mellem to knuder i en orienteret graf $G = (V, E)$. På kanterne i grafen er der vægte angivet som reelle tal mellem 0 og 1, $0 \leq w(u, v) \leq 1$. Disse vægte er et udtryk for, hvor stabil forbindelsen imellem knuderne u og v er, hvor 1 betyder 100 procent stabil og 0 betyder 0 procent stabil.

Hint: Benyt at dette gælder for $\log(x)$:

1. $\log(x) \leq 0$ hvis $0 < x \leq 1$
2. $-\log(x) \geq 0$ hvis $0 < x \leq 1$
3. $\log(x \cdot y) = \log(x) + \log(y)$

Hvis vi observerer $-\log(x)$ når vi ændrer værdien for x imellem 0 og 1 vil vi se at jo tættere x er på 1 jo tættere kommer $-\log(x)$ på 0. Modsat, jo tættere x kommer på 0 jo større bliver $-\log(x)$.

Idé: Modificer vægtningen af kanter og brug Dijkstras algoritme til at finde den mest stabile sti mellem to knuder.

Da vægtningen af kanter er i procent betyder det at vægten for en hel sti er produktet af kanterne på stien. En sti over kanterne $(x, p), (p, z)$ med vægte $w(x, p) = 0.8, w(p, z) = 0.3$ vil altså have samlet vægt $0.8 \cdot 0.3 = 0.24$. En anden sti med kanter $(x, y), (y, z)$ og vægte $w(x, y) = 0.5, w(y, z) = 0.5$ vil have samlet vægt $0.5 \cdot 0.5 = 0.25$. Stien over x, y, z er mest

stabil. Hvis vi brugte produktet af vægtene ville Dijkstras algoritme vælge stien x, p, z fordi den har lavest vægt, dette er ikke den egenskab vi ønsker. Her kommer egenskab nr. 3 til hjælp, $\log(x \cdot y) = \log(x) + \log(y)$. Hvis vi kigger på eksemplet fra før:

$$-\log(0.8 \cdot 0.3) = (-\log(0.8)) + (-\log(0.3)) \approx 1.42712$$

$$-\log(0.5 \cdot 0.5) = (-\log(0.5)) + (-\log(0.5)) \approx 1.38629$$

Dijkstras algoritme ville nu vælge stien x, y, z da $1.42712 > 1.38629$, præcis som vi ønsker. Ved at modificere vægtene med $-\log(w(u, v))$ får vi at lavere stabilitet giver højere vægt, højere stabilitet giver lavere vægt og at vægten for en sti afspejler produktet af stabiliteterne for kanterne i stien.

Vi skal dog være forsigtige da $-\log(0)$ ikke er defineret, vi ved dog at:

$$\lim_{x \to 0} -\log(x) = \infty$$

Dvs. $-\log(x)$ går mod ∞ når x går mod 0. For os betyder det at en kant med vægt 0 tæller som vægt ∞. Det passer også fint med det vi har fået at vide da en kant med vægt 0 vil være 0 procent stabil - altså konstant være ustabil, det er ikke en kant vi har lyst til at bruge.

Herunder er givet pseudokode til modificeringer af Dijkstras algoritme og algoritmen Relax.

```
1  Modified-Dijkstra(G,w,s):
2      Initialize-Single-Source(G,s)
3      S = ∅
4      Q = G.V
5      while Q ≠ ∅:
6          u = Extract-Min(Q)
7          S = S ∪ {u}
8          for each vertex v ∈ G.Adj[u]:
9              Modified-Relax(u,v,w)
```

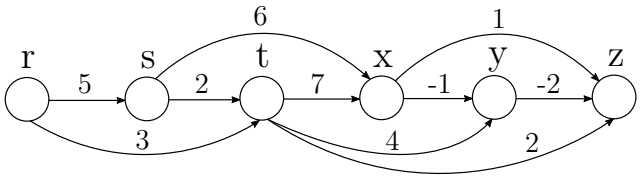
```
1  Modified-Relax(u,v,w):
2      if w(u,v) == 0:
3          return
4      if v.d > u.d + (-log(w(u,v))):
5          v.d = u.d + (-log(w(u,v)))
6          v.π = u
```

Den modificerede version af Dijkstras algoritme fungerer som normalt, vægtene bliver udregnet anderledes i den modificerede version af Relax, og vi tager højde for hvis $w(u, v) = 0$.

Hvis det ønskes at kende den mest stabile sti fra u til v kan denne modificerede version køres med s sat til u. Når algoritmen er færdig vil man kende de mest stabile stier fra u til alle andre knuder, og altså også v.

6 Cormen et al. øvelse 24.2-1 (side 657)

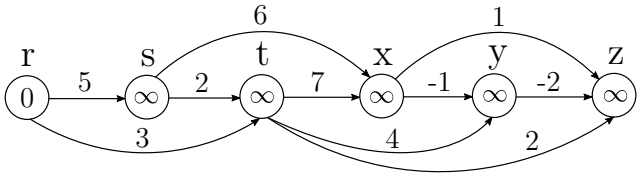
Vi skal udføre `Dag-Shortest-Paths` med knuden r som source på følgende graf:



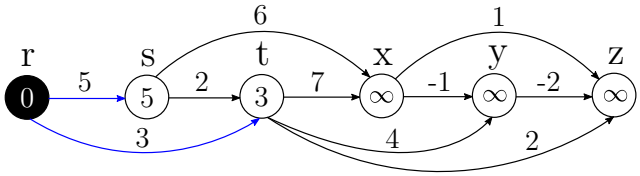
Knuderne i grafen er allerede topologisk sorteret: r, s, t, x, y, z.

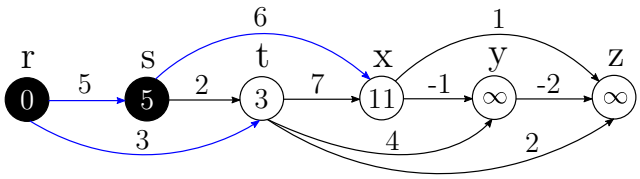
Vi skal derfor blot kalde `Init-Single-Source` med r, og så betragte knuderne efter rækkefølgen i den topologiske sortering, og for hver knude, kalde `Relax` med knuden og hver af dens naboer.

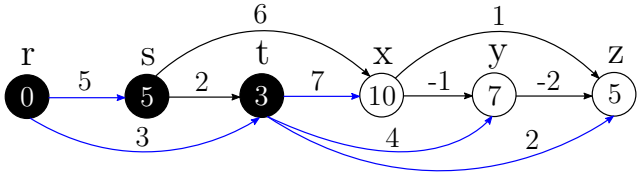
Grafen efter `Init-Single-Source`:

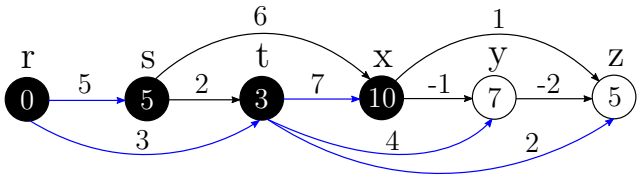


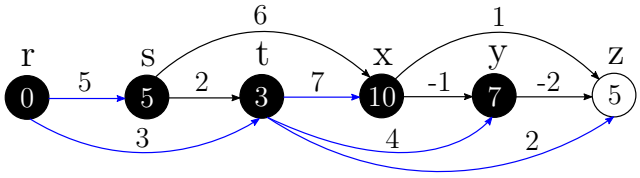
I følgende vises udseendet af grafen efter hver knude er blevet behandlet. De blå kanter indikerer, hvor en knude har fået dens distance fra (dvs. den indikerer π), og en sort knude indikerer, at knuden er blevet behandlet.

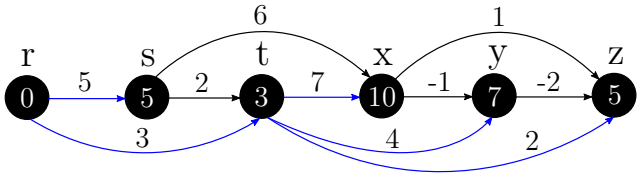












7 Cormen et al. øvelse 25.2-1 (side 699)

Vi får givet en graf i figur 25.2 på side 691 i Cormen et al[1] som vi skal udføre Floyd-Warshall algoritmen på. Før vi kan gøre dette skal vi først lave grafen om til en adjacency-matrix repræsentation. Det gør vi ved at lave en 6×6 matrix og lade plads i, j indeholde vægten for kanten fra knude i til knude j. Er der ikke kant mellem i, j lader vi pladsen indeholde ∞ og hvis $i = j$ lader vi pladsen være 0. Det giver følgende matrix for grafen fra figur 25.2 som også er D^0:

$$D^0 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

I den yderste for-løkke skal vi i hver iteration lave en nye matrix som vi udfylder med de næste 2 indre for-løkker. I den første iteration af den ydre for-løkke, hvor $k = 1$ laver vi en ny matrix D^1. Herefter gennemgås hver plads i matricen og vi skrive den mindste af værdien på korresponderende plads i D^0 (dvs. d^0_{ij}) og $d^0_{ik} + d^0_{kj}$. De fleste pladser kommer til at have de samme værdier som i D^0. Det første sted hvor dette ikke er tilfældet er i række 2 kolonne 5 ($i = 2, j = 5$). Her er den korresponderende plads i D^0 lig med $d^0_{2,5} = \infty$, hvilket er større end $d^0_{2,1} + d^0_{1,5} = 1 + (-1) = 0$. Vi lader derfor pladsen $d^1_{2,5}$ være 0. Den samme proces sker i række 4 kolonne 5 ($i = 4, j = 5$), hvor $d^0_{4,5} = 3$ og $d^0_{4,1} + d^0_{1,5} = (-4) + (-1) = -5$. Den

resulterende fulde matrix D^1 findes nedenunder.

$$D^1 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

I anden iteration laver vi en matrix D^2 og udfylder hvert felt ligesom ved D^1 dog med udgangspunkt i D^1 (da vi lavede D^1 tog vi udgangspunkt i D^0). Her er specielt række 3 og 6 forskellige fra de korresponderende rækker i D^1.

$$D^2 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

I tredje iteration, hvor $k = 3$ laves matrixen D^3, men denne ender med at være identisk med D^2. Da d^2_{ij} var mindst for alle i, j.

$$D^3 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

I de efterfølgende matricer kan man finde de sidste 3 iterationer af den yderste for-løkke. Den samme metode som ved D^1 bliver brugt (dog ændres flere pladser end 2).

$$D^4 = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$D^5 = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

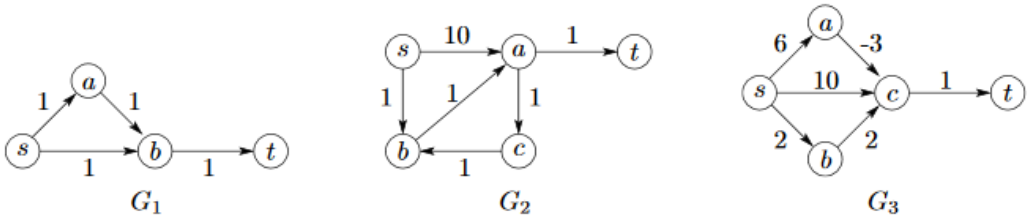
I den sidste iteration laves $D^n = D^6$ som kommer til at indeholde længde på alle de korteste stier mellem alle par af knuder. For at finde længden på den korsteste sti fra knude i til knude j kan man se værdien på plads d^6_{ij}. F.eks. for at finde længden på den korteste sti fra knude

3 til knude 5 kigger vi i række 3 kolonne 5. Her vi kan aflæse at stien har længden -6. Dette svarer til stien 3-6-2-4-1-5 som man kan verificere er den mindste.

$$D^6 = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

8 Eksamen juni 2011, opgave 4

I denne opgave ser vi på tre forskellige algoritmer anvendt på hver af de tre vægtede grafer i Figur 6. Som sædvanligt er længden af en vej i en vægtet graf summen af vægtene på kanterne i vejen.



Figur 6: Tree vægtede grafer

Spørgsmål a: Man kan kun bruge BFS når kantvægtene udelukkende består af 1. Hvis man dog kun er interesseret i at finde de korteste veje, men ikke nødvendigvis de korrekte længder, så er det nok at alle vægte er ens. For G_1 kan man altså benytte BFS, men det kan man ikke for G_2 (BFS giver længden 2 fremfor længden 3) og G_3 (BFS giver længden 2 fremfor længden 4).

Spørgsmål b: Man kan ikke benytte `DAG-SHORTEST-PATH` på G_2, da den indeholder en kreds. Hvis man prøver alligevel, så vil man, hvis man antager nabolisterne er sorteret, få følgende rækkefølge: s, a, t, c, b. Med denne rækkefølge vil algoritmen bestemme længden mellem s og t til at være 11 fremfor den korrekte længde 3. G_1 og G_3 indeholder ingen kredse, så derfor kan man benytte `DAG-SHORTEST-PATH` på disse.

Spørgsmål c: Man kan ikke benytte *Dijkstras algoritme* på G_3, da den indeholder en negative vægt. Forsøger man alligevel, så vil man få længden af den korteste vej fra s til t til at være 5 fremfor den korrekte længde 4. Man kan benytte *Dijkstras Algoritme* på G_1 og G_2.

9 Eksamen juni 2014, opgave 10

Delopgave a

I denne delopgave skal vi udtrykke antallet af knuder og kanter i en kvadratgraf, som funktion af k der angiver hvor mange rækker og kolonner kvadratgrafen har.

Vi lader n og m betegne henholdsvis antallet af knuder og antallet af kanter i en kvadratgraf. Funktioner der udtrykker n og m som en funktion af k kan bestemmes ved at lave følgende observationer. Vi kigger på hvordan n og m variere som k øges:

```
For k = 1:      For k = 2:      For k = 3:      For k = 4:
o               o← →o           o← →o← →o       o← →o← →o← →o
                ↑   ↑           ↑   ↑   ↑       ↑   ↑   ↑   ↑
                o← →o           o← →o← →o       o← →o← →o← →o
                                ↑   ↑   ↑       ↑   ↑   ↑   ↑
                                o← →o← →o       o← →o← →o← →o
                                                ↑   ↑   ↑   ↑
                                                o← →o← →o← →o
n = 1 knude     n = 4 knuder    n = 9 knuder    n = 16 knuder
m = 0 kanter    m = 6 kanter    m = 18 kanter   m = 36 kanter
```

I forhold til antallet af knuder ser vi følgende sammenhæng:

$$\begin{aligned}
\text{knuder } n = 1 &= k^2, \text{ for } k = 1 \\
\text{knuder } n = 4 &= k^2, \text{ for } k = 2 \\
\text{knuder } n = 9 &= k^2, \text{ for } k = 3 \\
\text{knuder } n = 16 &= k^2, \text{ for } k = 4 \\
&\vdots
\end{aligned}$$

Vi ser altså at antallet af knuder må være givet ved: $n = k^2$. Dette kan evt. bevises ved et induktionsbevis.

I forhold til antallet af kanter ser vi at antallet af kanter cirka følger udviklingen:

$$\begin{aligned}
k \cdot (k-1) &= 0, \text{ for } k = 1 \\
k \cdot (k-1) &= 2, \text{ for } k = 2 \\
k \cdot (k-1) &= 6, \text{ for } k = 3 \\
k \cdot (k-1) &= 12, \text{ for } k = 4 \\
&\vdots
\end{aligned}$$

Hvis vi justerer med en faktor 3, så får vi samme udvikling, som observeret i ovenstående

illustration:

$$\begin{aligned}
&\text{kanter } m = 0 \ = 3k \cdot (k-1), \text{ for } k = 1 \\
&\text{kanter } m = 6 \ = 3k \cdot (k-1), \text{ for } k = 2 \\
&\text{kanter } m = 18 = 3k \cdot (k-1), \text{ for } k = 3 \\
&\text{kanter } m = 36 = 3k \cdot (k-1), \text{ for } k = 4 \\
&\qquad\qquad \vdots
\end{aligned}$$

Vi ser altså at antallet af kanter må være givet ved: $m = 3k \cdot (k-1) = 3k^2 - 3k$.

Delopgave b

I denne delopgave angiver vi tiden af Dijkstras algoritme, når denne udføres på en kvadratgraf. Da alle knuder kan nås fra startknuden $v_{1,1}$, så kører Dijkstras algoritme i $O(m \log(n))$ tid. Dijktras algoritme vil derfor køre i $O\left(k^2 \log(k)\right)$ tid på kvadratgrafen da $m = 3k^2 - 3k$ og $n = k^2$.

Delopgave c

I denne opgave skal vi konstruere en algoritme som finder længden af de korteste stier fra knuden $v_{1,1}$ til alle øvrige knuder i en kvadratgraf. Vi argumentere endvidere for algoritmens køretid og korrekthed.

En mulig algoritme der kører i $O(m)$ tid og finder længden af de korteste stier fra knuden $v_{1,1}$ og alle øvrige knuder, er beskrevet i det følgende. I en sammenhængende graf $G = (V, E)$ findes der altid en korteste sti fra en startknude til en anden knude, som er simpel dvs. der er ikke nogen gentagne knuder på stien. Dette er tilfældet, da en gentagen knude i en sti betyder at der er en kreds i stien, og da alle kantvægte er ikke-negative, kan sådanne kredse blot skæres væk uden at gøre stien længere.

I vores tilfælde vil en simpel sti fra startknuden $v_{1,1}$ til en anden vilkårlig knude v i V, have følgende form:

```
0 eller flere kanter mod højre
1 skridt op
0 eller flere kanter mod højre ELLER 0 eller flere kanter mod venstre
1 skridt op
0 eller flere kanter mod højre ELLER 0 eller flere kanter mod venstre
.
.
.
```

Fra Cormen et al. Lemma 24.15 ved vi at hvis kanterne på en korteste sti fra startknuden til en anden knude v relakseres i stiens rækkefølge, så vil distancen $v.d$ være være sat korrekt, da v's afstand fra startknuden ikke ændres via yderligere relakseringer.

Hvis vi kombinere dette med ovenstående viden om udseendet af simple stier i kvadratgrafen, så ser vi at følgende algoritme ved dens afslutning vil have sat $v.d$ korrekt for alle knuder, eftersom den for alle knuder relakserer kanterne på en korteste sti til knuden i denne stis rækkefølge:

```
1  Initialize(G, k):
2      # Lad G = (V, E) og v{1, 1},..., v{k, k} være i V
3      # Initialiser distancer for alle knuder i grafen
4      for i = 1 to k:
5          for j = 1 to k:
6              v.d = inf
7          v_{1, 1}.d = 0
8
9  Relax-Edges(G, k):
10     # Lad G = (V, E) og v{1, 1},..., v{k, k} være i V
11     # Relakser højrevendte kanter i lag 1, fra venstre mod højre
12     for j = 1 to k-1:
13         relax(v{1, j}, v_{1, j+1})
14     # For alle lag i >= 2
15     for i = 2 to k:
16         # Relakser opvendte kanter fra lag i-1 til lag i
17         for j = 1 to k:
18             relax(v{i-1, j}, v_{i, j})
19         # Relakser højrevendte kanter i lag i, fra venstre mod højre
20         for j = 1 to k-1:
21             relax(v{i, j}, v_{i, j+1})
22         # Relakser venstrevendte kanter i lag i, fra højre mod venstre
23         for j = 1 to k-1:
24             relax(v{i, k-j+1}, v_{i, k-j})
```

Ovenstående algoritme relakserer højst hver kant en gang, hvilket tager $O(m)$ tid. Derudover bruger den $O(n)$ tid på initialisering. Da n er $O(m)$ for kvadratgrafer, kører den i $O(m)$ tid.

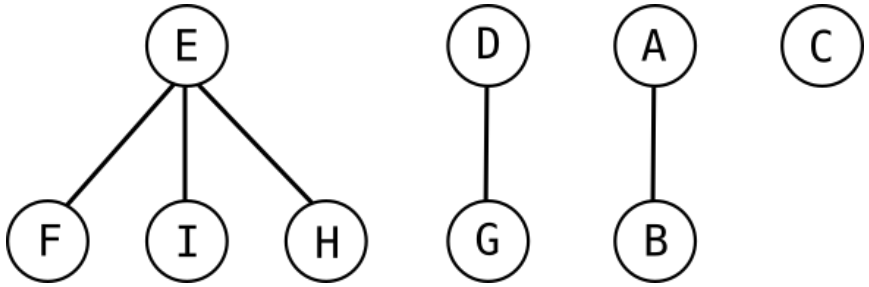
Vi ser i denne forbindelse at denne algoritme kører i $O(k^2)$ tid, hvilket er en faktor $\log(k)$ hurtigere end Dijkstras algoritme.

10 Eksamen januar 2006, opgave 3, spørgsmål a og c.

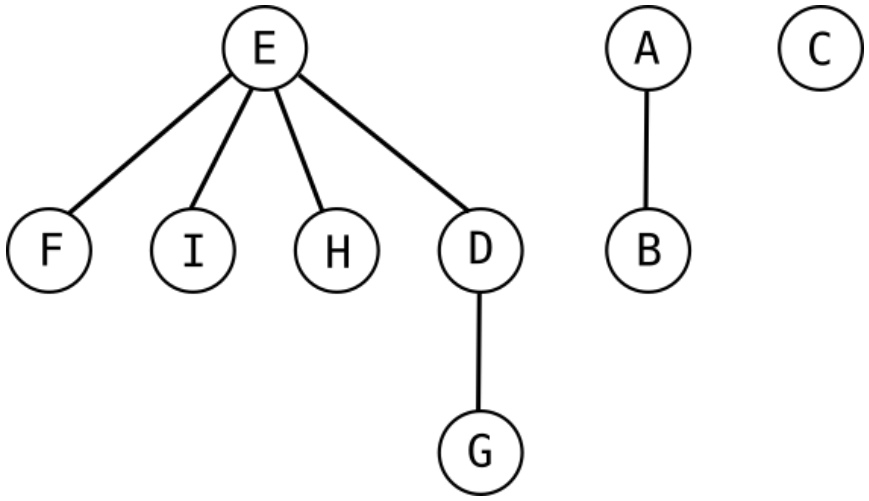
Spørgsmål a: Hvilken kant er den næste, som Kruskals algoritme inkluderer i det udspændende træ? Vis skoven, som repræsenterer de disjunkte mængder, efter at denne sjette kant er tilføjet. Husk at bruge *union by rank* og *path compression*.

Den næste kant som Kruskals algoritme betragter må enten være (F, I) eller (D, H), da begge har vægt 4, og der er ingen kanter med lavere vægt, som ikke allerede er valgt. Dog kan (F, I) ikke vælges, da F og I er i den samme mængde (enderpunkter er i samme sammenhængskomponent). Selv hvis Kruskals algoritme betragter (F, I) først, så vil *path compression* ikke ændre *disjoint-set* skoven.

Knuderne D og H er ikke i samme mængde, så derfor vil Kruskals algoritme vælge (D, H). For at afgøre om D og H er i forskellige mængder evalueres `FIND-SET(D)` $\neq$ `FIND-SET(H)`, hvilket giver anledning til følgnede ændring af *disjoint-set* skoven pga. *path compression*:



Derefter udføres `UNION(H, D)`. Da roden i træet for den mængde H er i har større rank end roden i træet for den mængde D er i, så opdateres *disjoint-set* skoven på følgende måde pga. *union by rank*:



Spørgsmål c: Lad G være en vægtet graf, og lad $\{e_1, \ldots, e_k\}$ være kanter i G. Forklar, hvordan man kan undersøge, om der findes et MST for G, som indeholder disse k kanter. Hvad er køretiden af din algoritme?

En mulig algoritme vil være følgende som består af to dele:

Del 1: Kør en modificeret udgave af Kruskals algoritme, hvor kanterne $\{e_1, \ldots, e_k\}$ tages først (i en vilkårlig rækkefølge). Hvis vi under denne del finder at to kanter tilhører den samme sammenhængskomponent (de er i samme mængde), så stopper vi og returnerer FALSK, da nogle af kanterne $\{e_1, \ldots, e_k\}$ danner en kreds (der kan ikke være kredse i et MST). Derefter tages de restende kanter i sorteret orden og to cases kan opstå:

- Case 1: De indtil nu valgte kanter $\{e_1, \ldots, e_k\}$ kan udvides til et MST. Algoritmen vil fortsætte med at vedligeholde invarianten, at de indtil nu valgte kanter kan udvides til et MST, og derfor ender man med et MST (argumentet for dette er identisk med argumentet i [4, s. 19]).
- Case 2: De indtil nu valgte kanter $\{e_1, \ldots, e_k\}$ kan ikke udvides til et MST.

Den endelige mængde af kanter som returneres indgår altså ikke i et MST. Observer, at algoritmen stadig aldrig vil introducere en kreds, og da den oprindelige graf er sammenhængen, så vil den endelig mængde af kanter (sammen med den oprindelige mængde knuder) danne et træ.

Eftersom både Case 1 og Case 2 returnerer en mængde af kanter, der danner et træ, så skal vi bare afgøre om værdien af dette træ er lig værdien af et MST for G. Dette afgør vi i Del 2.

Del 2: Kør Kruskals algoritme (eller Prim-Jarniks algoritme) på G. Hvis værdien af dette MST er lig værdien af det returnerede træ fra Del 1, så returneres SANDT (træet med $\{e_1, \ldots, e_k\}$ har samme værdi som et MST). Ellers returneres FALSK (træet med $\{e_1, \ldots, e_k\}$ har ikke samme værdi som et MST).

Den modificerede udgave af Kruskals algoritme har samme køretid som Kruskals algoritme. Asymptotisk set er det sorteringen af kanterne der dominerer, og i worst-case er der ingen kanter givet som input, hvilket vil sige vi sorterer alle m kanter - derfor er køretiden fortsat $O(m \log m)$. Bruges Kruskals algoritme i Del 2, så tager dette $O(m \log m)$ tid. Køretiden for algoritmen er altså $O(m \log m)$.

NB: Opgaven siger ikke noget om G er sammenhængende eller ej, så man skal i princippet først sikre sig G er sammenhængende. For at tjekke om G er sammenhængende kan man f.eks. bruge BFS for at sikre sig alle knuder farves sorte - dette tjek vil ikke ødelægge $O(m \log m)$ køretiden.

Litteratur

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[2] Rolf Fagerberg. Grafer og graf-gennemløb. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/graphTraversalSlides.pdf`, 2020.

[3] Rolf Fagerberg. Korteste veje. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/shortestPathsSlides.pdf`, 2020.

[4] Rolf Fagerberg. Minimum udspændende træer. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/mstSlides.pdf`, 2020.