Dijkstras algoritme

Åsmund Eldhuset

asmunde *at* stud.ntnu.no

folk.ntnu.no/asmunde/algdat/dijkstra.pdf

Korteste sti

- Vi er ofte interessert i å finne korteste, raskeste eller billigste vei mellom to punkter
  - Gods- og persontransport
  - GPS-kartsystemer
  - Ruting av internettrafikk
- Kart og nettverk kan modelleres som vektede grafer, der hver kant har en lengde, pris eller tid
- Gitt en vektet graf og en startnode, finn de korteste stiene fra startnoden til hver av de andre nodene

BFS i uvektede grafer

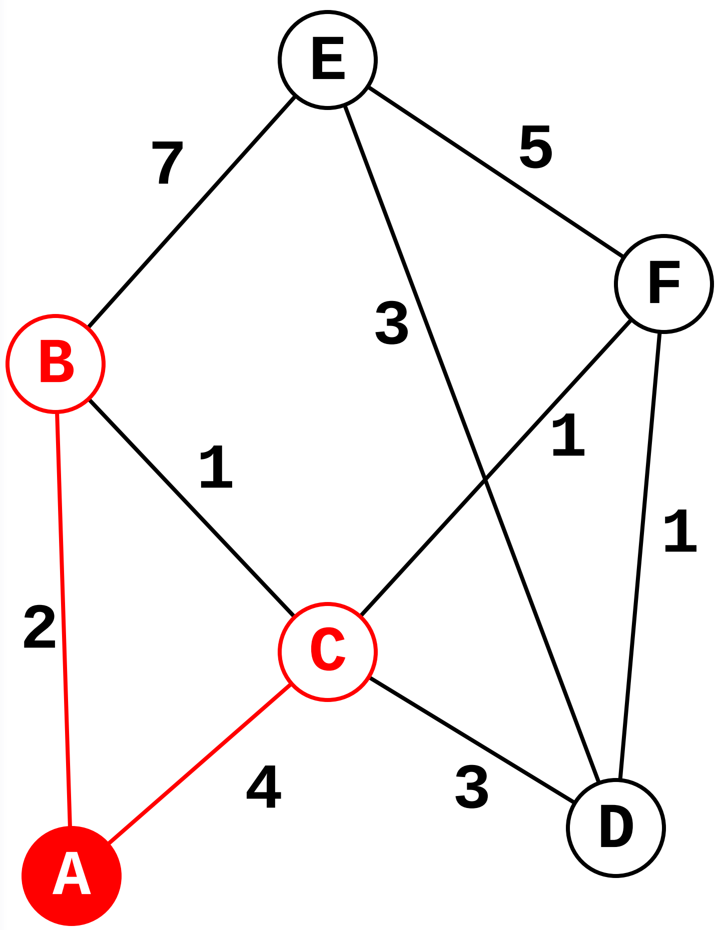
- I uvektede grafer har alle kanter samme lengde
- I BFS vil da noder komme inn i (og ut av) køen i stigende rekkefølge mhp. avstand
  - Startnoden (avstand 0) vil legge til alle nodene med avstand 1
  - Hver node med avstand 1 vil legge til naboer som har avstand 2, osv.
- Derfor fungerer BFS til å finne korteste vei fra ett punkt til alle andre, fordi ALLE noder med avstand d er funnet før fjernere noder

BFS i vektede grafer

- Prøver vi BFS på en vektet graf, får vi problemer
- På grunn av de ulike vektene vil nodene komme inn i køen i feil rekkefølge
- Når vi tar ut en node v og legger til naboene dens, kan det hende at det egentlig fantes en kortere vei til v

BFS i vektede grafer

Kø: A (0), B (2), C (4)



- C ble lagt inn i køen med avstand 4, men B kan tilby en kortere vei
- Når vi behandler C vil D legges til før F, selv om F har den korteste veien

Modifikasjon av BFS

- Problem nr. 1: Vi risikerer å behandle noder som vi ikke har funnet den korteste veien til
- Idé nr. 1: i stedet for å ta ut den første noden fra køen, hva med å ta ut den noden med kortest avstand?

Modifikasjon av BFS

- Problem nr. 2: underveis kan vi finne kortere veier til noen noder, men da er nodene allerede markert som besøkt
- Idé nr. 2: hva med å tillate at man finner bedre avstander underveis?
- Må ikke markere noder som besøkt før man tar dem ut av køen
- Avstandene vi finner underveis, er ikke endelige – de er bare estimater helt frem til vi er sikre på at vi har den korteste stien
- Når kan vi være sikre på at vi har funnet den korteste stien?

Dijkstras algoritme

- Hvis vi modifiserer BFS slik, får vi Dijkstras algoritme
- Vi bruker en prioritetskø slik at vi raskt kan finne den ”korteste” ubesøkte noden (den med kortest estimat)
- En prioritetskø kan lages slik:
  - Et array som man holder sortert når man setter noe inn (treigt å sette inn, raskt å hente ut)
  - Et array som man søker gjennom hver gang man skal ta noe ut (raskt å sette inn, treigt å hente ut)
  - En heap (”haug”) (begge deler går temmelig raskt)

Pseudokode

```
S er en tom liste
Q er en prioritetskø
sett alle estimater til ∞
sett startnodens estimat til 0
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat
            sett u som v sin forgjenger
```

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each vertex $v \in Adj[u]$
      do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each vertex $v \in Adj[u]$
      do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

$\text{INITIALIZE-SINGLE-SOURCE}(G, s)$
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow \text{EXTRACT-MIN}(Q)$
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do $\text{RELAX}(u, v, w)$

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

$\text{INITIALIZE-SINGLE-SOURCE}(G, s)$

$S \leftarrow \emptyset$

$\mathbf{Q \leftarrow V[G]}$

while $Q \neq \emptyset$

do $u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

for each vertex $v \in Adj[u]$

do $\text{RELAX}(u, v, w)$

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \varnothing$
$Q \leftarrow V[G]$
while $Q \neq \varnothing$
    do $u \leftarrow$ **EXTRACT-MIN(Q)**
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $\boldsymbol{S \leftarrow S \cup \{u\}}$
        for each vertex $v \in Adj[u]$
            do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

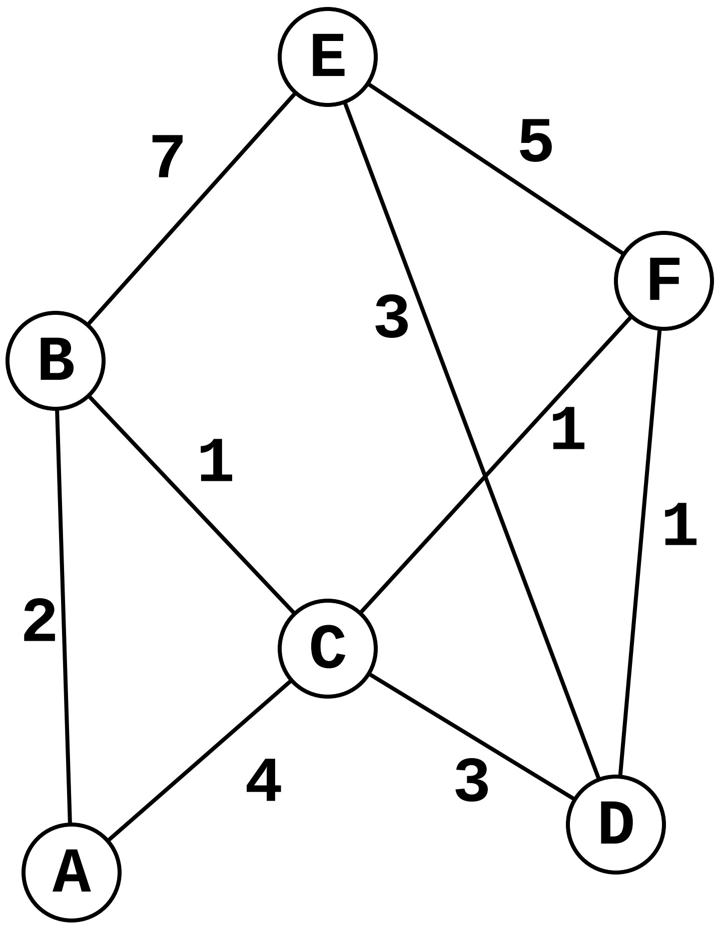
INITIALIZE-SINGLE-SOURCE(G, s)
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each vertex $v \in Adj[u]$
      do RELAX(u, v, w)

Sammenl. med koden i boka

```
sett alle estimater til ∞
sett startnodens estimat til 0
S er en tom liste
Q er en prioritetskø
legg alle noder inn i Q
så lenge Q ikke er tom:
    sett u til den "korteste" noden i Q
    fjern u fra Q
    legg u til de kjente nodene
    for hver nabo v av u:
        hvis u kan tilby en kortere sti til v:
            oppdater v sitt estimat og forgjenger
```

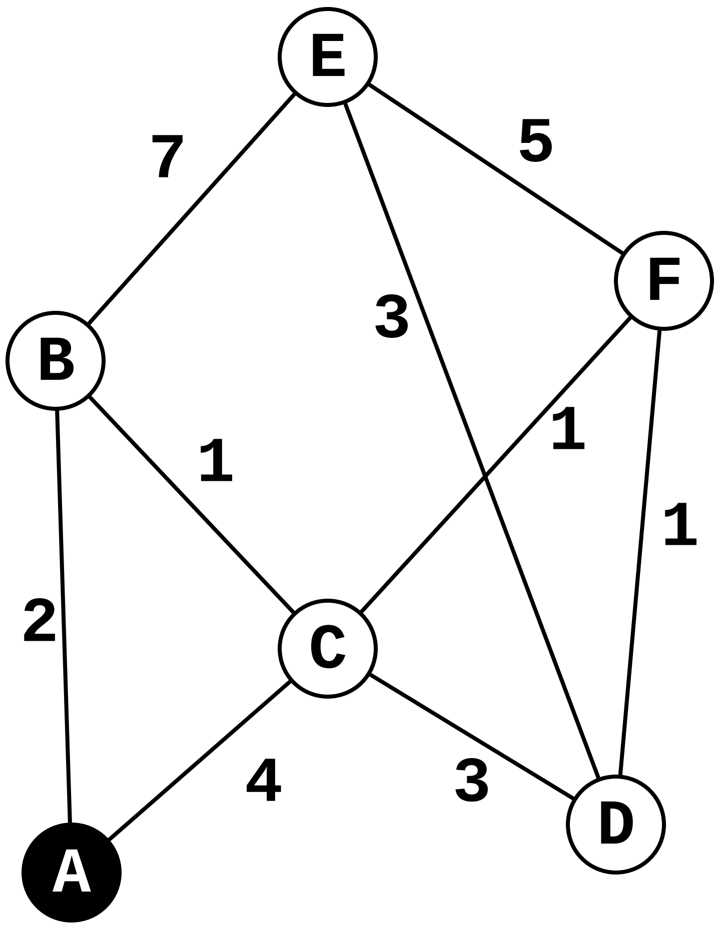
```
INITIALIZE-SINGLE-SOURCE(G, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)
```

Eksempel



Forside
Korteste sti
BFS
Modifikasjon
Dijkstra
:: Eksempel
Korrekthet
Analyse
Øving
Spørsmål

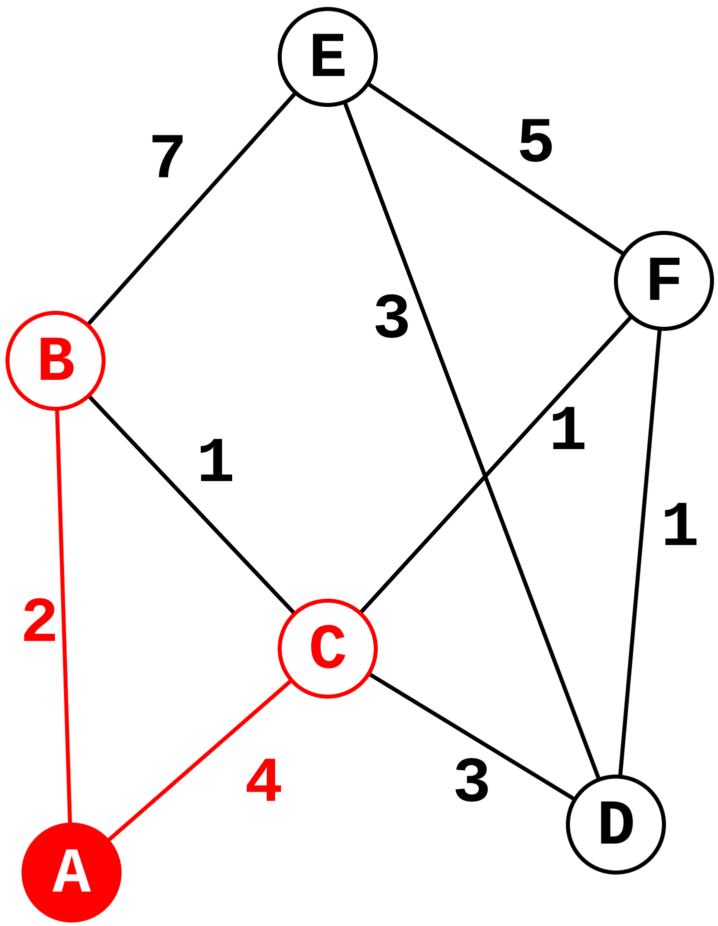
Eksempel



Nodeavstander

<A: 0 (-)>
B: ? (-)
C: ? (-)
D: ? (-)
E: ? (-)
F: ? (-)

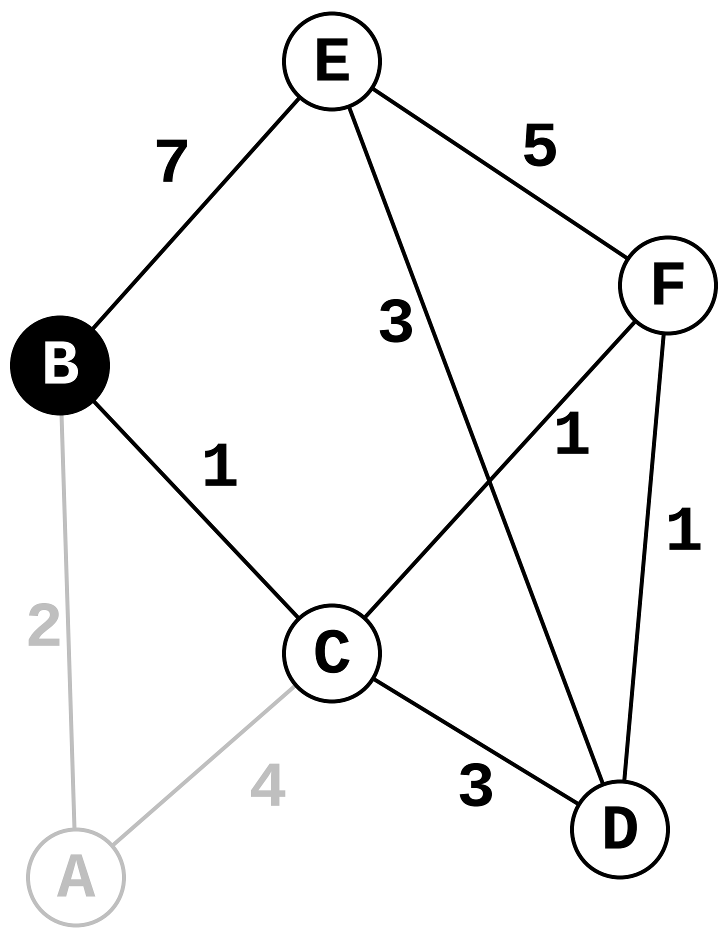
Eksempel



Nodeavstander

```
<A: 0 (-)>
 B: 2 (A)
 C: 4 (A)
 D: ? (-)
 E: ? (-)
 F: ? (-)
```

Eksempel

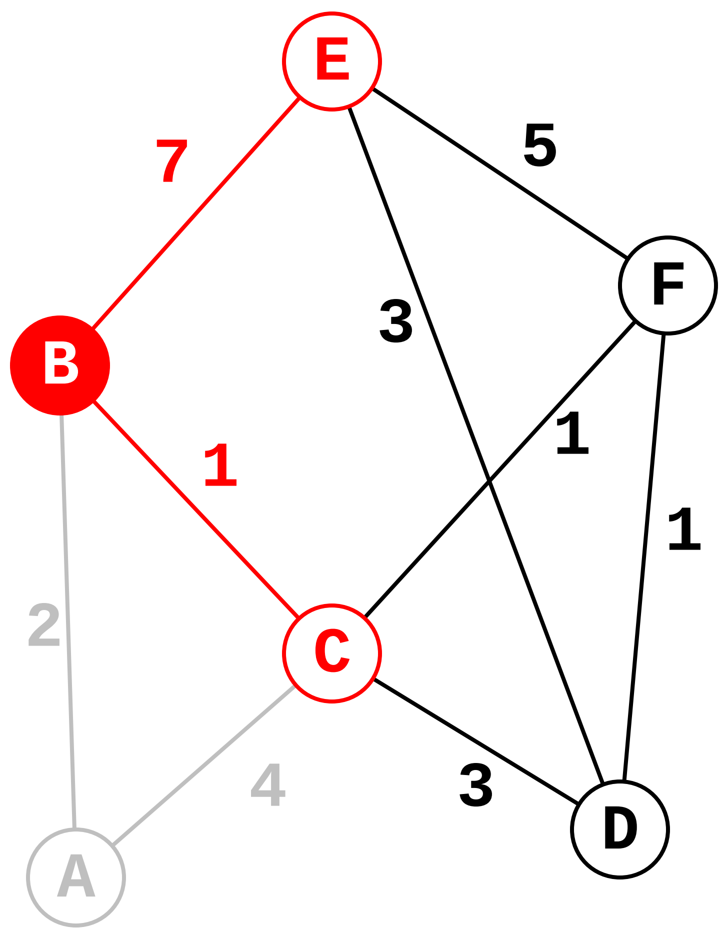


Nodeavstander

```
 A: 0 (-)
&lt;B: 2 (A)&gt;
 C: 4 (A)
 D: ? (-)
 E: ? (-)
 F: ? (-)
```

Forside
Korteste sti
BFS
Modifikasjon
Dijkstra
:: Eksempel
Korrekthet
Analyse
Øving
Spørsmål

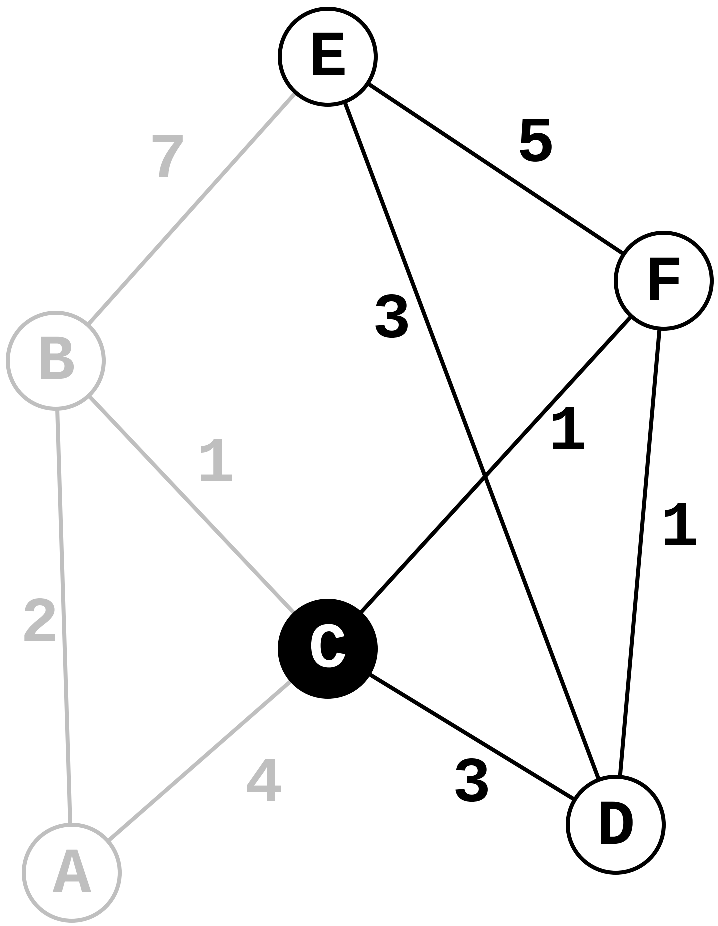
Eksempel



Nodeavstander

A: 0 (-)
<B: 2 (A)>
C: 3 (B)
D: ? (-)
E: 9 (B)
F: ? (-)

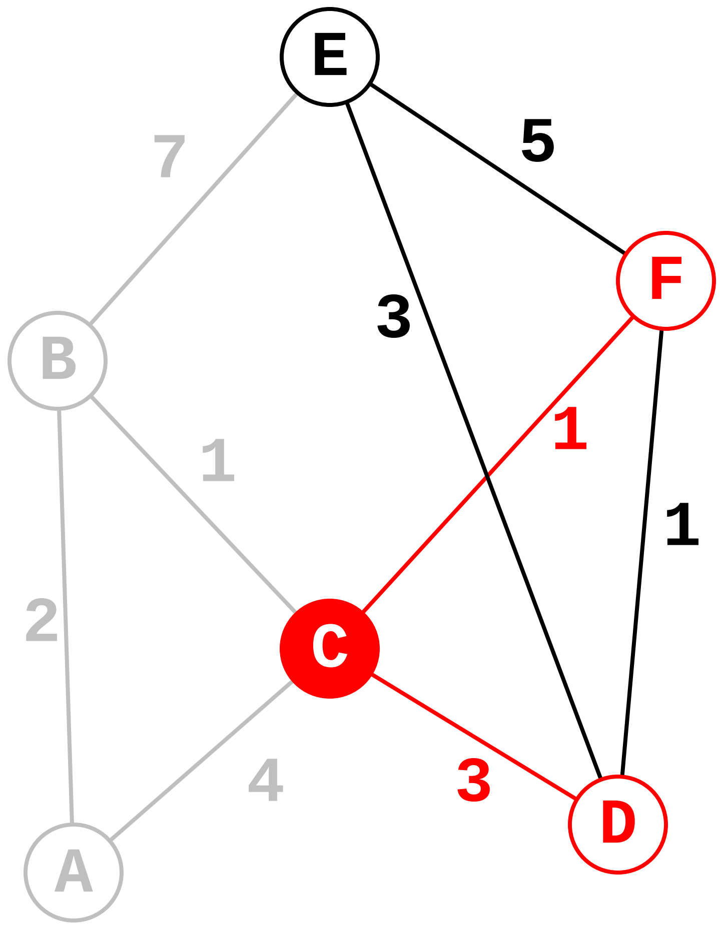
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
<C: 3 (B)>
D: ? (-)
E: 9 (B)
F: ? (-)

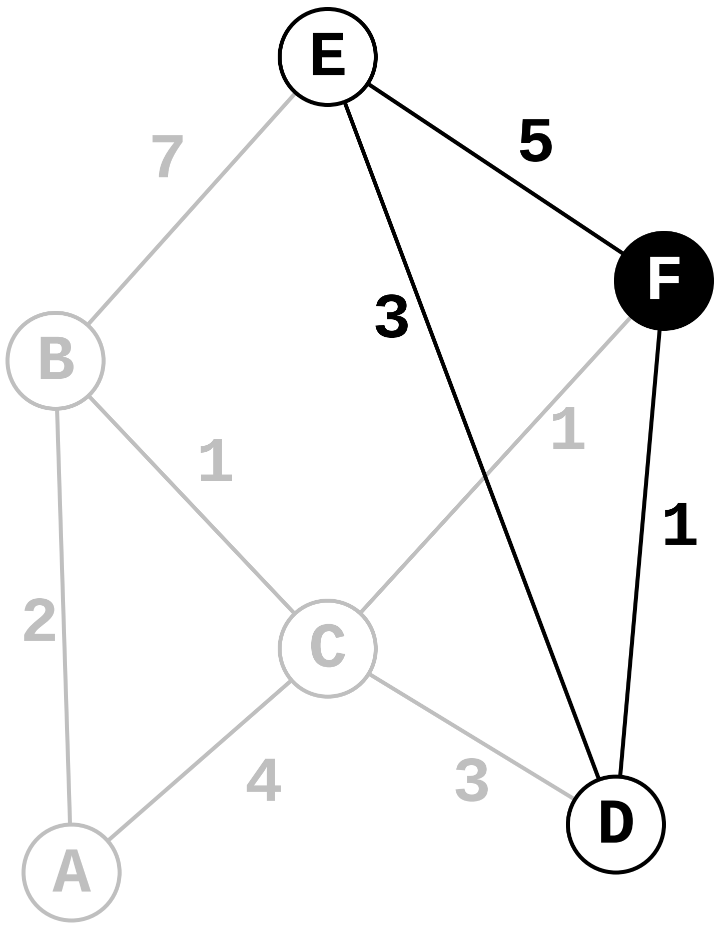
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
<C: 3 (B)>
D: 6 (C)
E: 9 (B)
F: 4 (C)

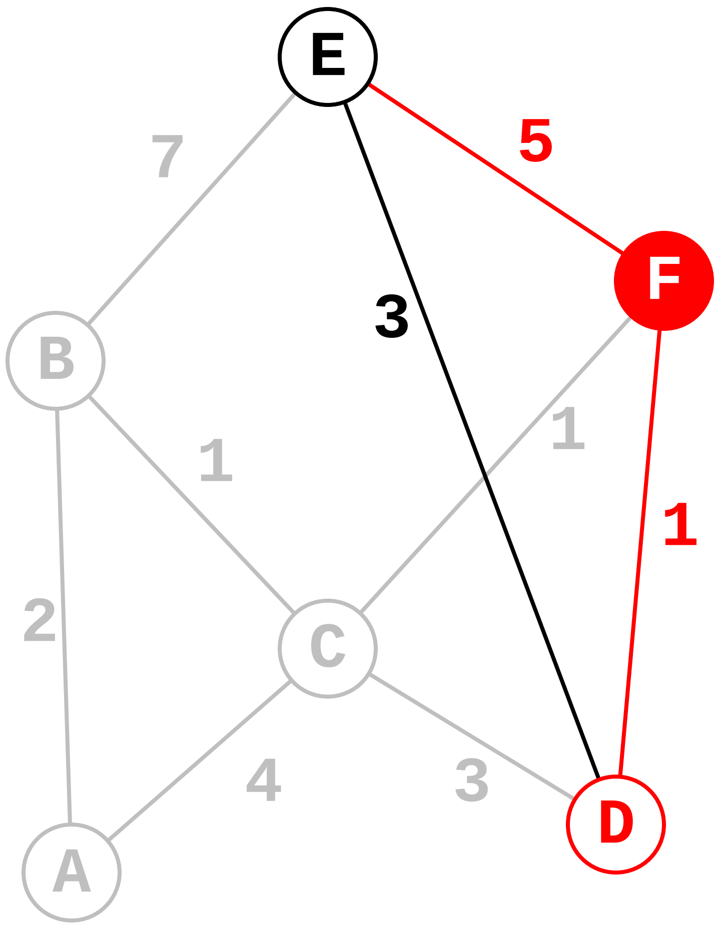
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
C: 3 (B)
D: 6 (C)
E: 9 (B)
<F: 4 (C)>

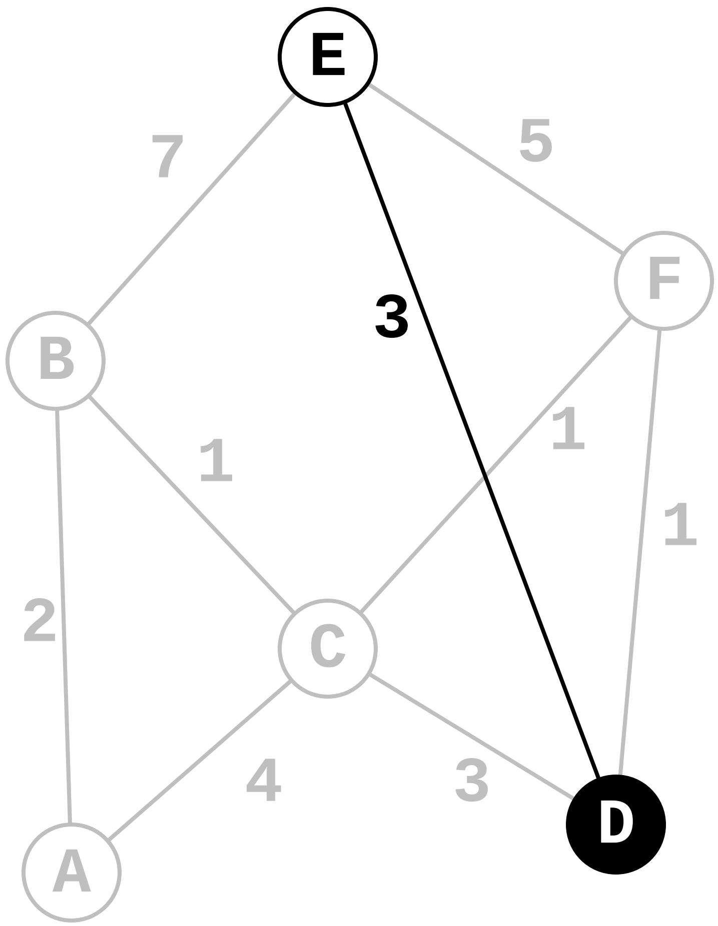
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
C: 3 (B)
D: 5 (F)
E: 9 (F)
<F: 4 (C)>

Eksempel

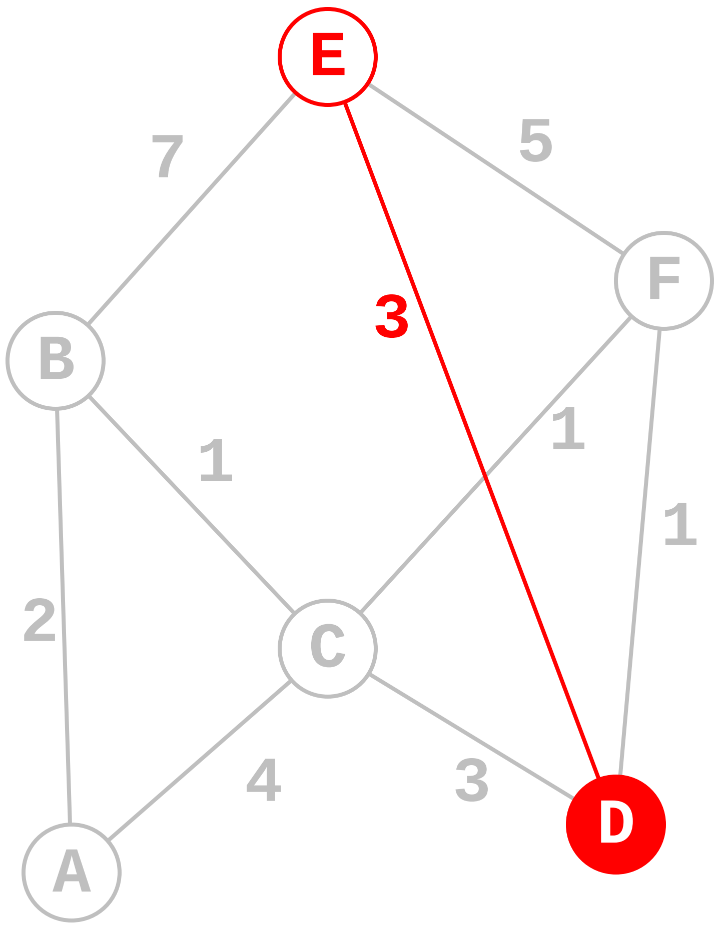


Nodeavstander

```
 A: 0 (-)
 B: 2 (A)
 C: 3 (B)
<D: 5 (F)>
 E: 9 (F)
 F: 4 (C)
```

Forside
Korteste sti
BFS
Modifikasjon
Dijkstra
:: Eksempel
Korrekthet
Analyse
Øving
Spørsmål

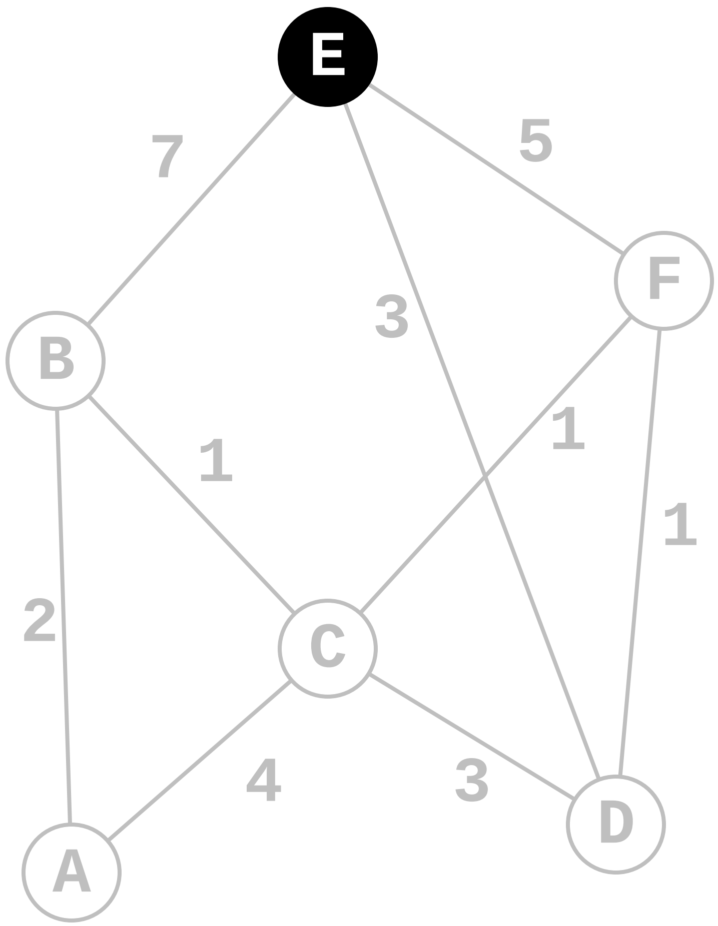
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
C: 3 (B)
<D: 5 (F)>
E: 8 (D)
F: 4 (C)

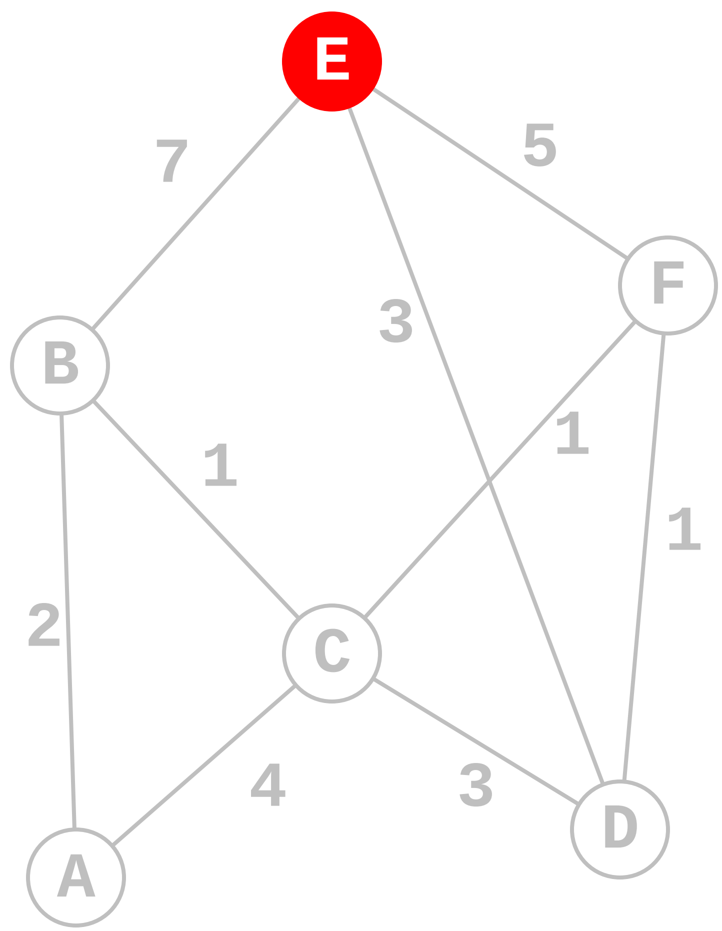
Eksempel



Nodeavstander

```
 A: 0 (-)
 B: 2 (A)
 C: 3 (B)
 D: 5 (F)
<E: 8 (D)>
 F: 4 (C)
```

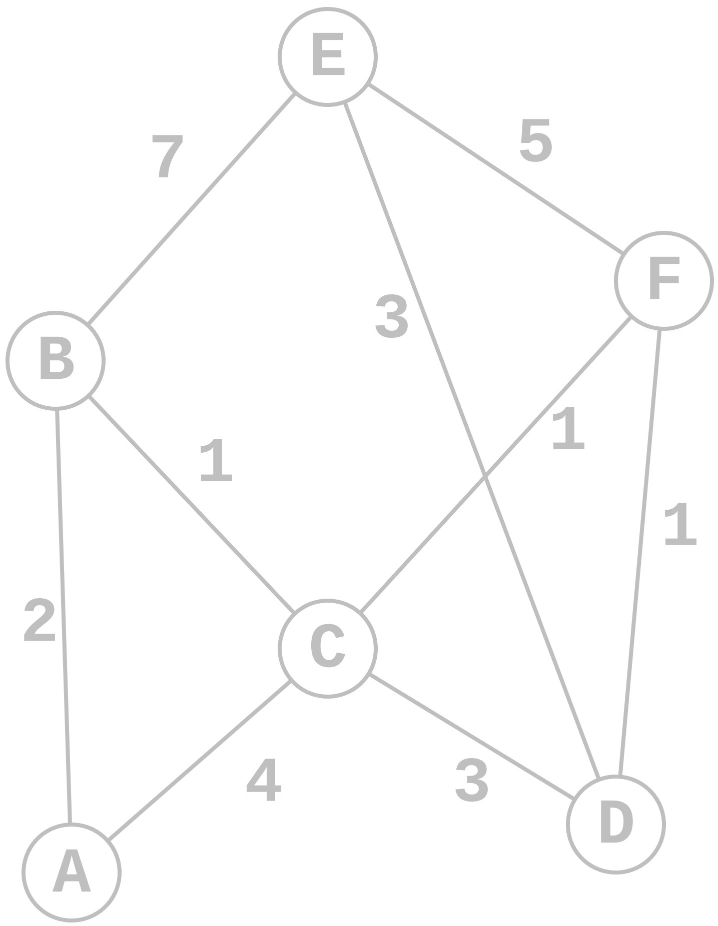
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
C: 3 (B)
D: 5 (F)
<E: 8 (D)>
F: 4 (C)

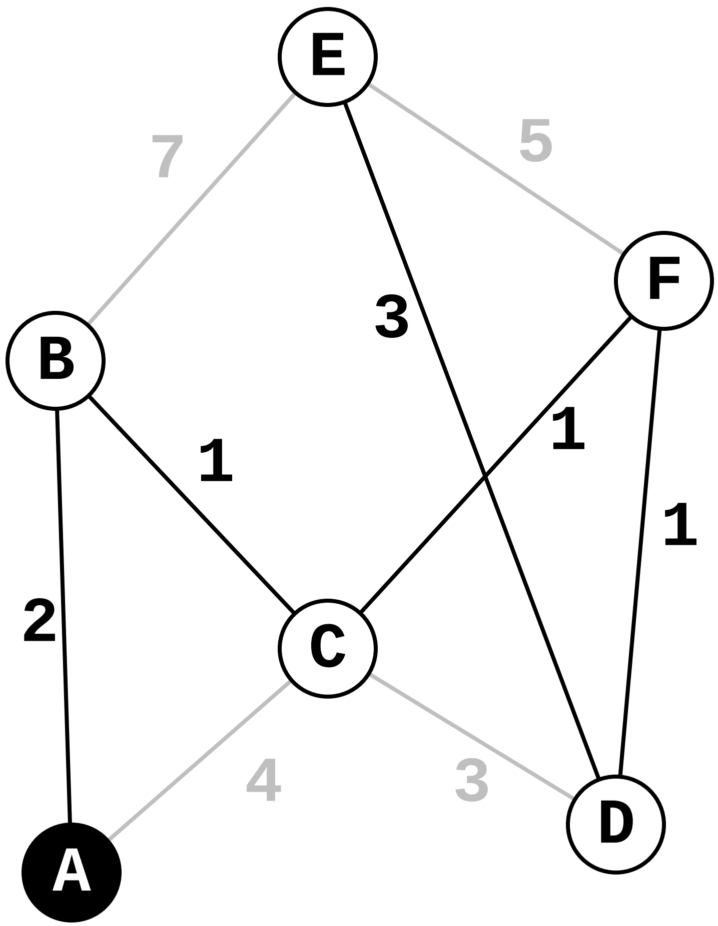
Eksempel



Nodeavstander

A: 0 (-)
B: 2 (A)
C: 3 (B)
D: 5 (F)
E: 8 (D)
F: 4 (C)

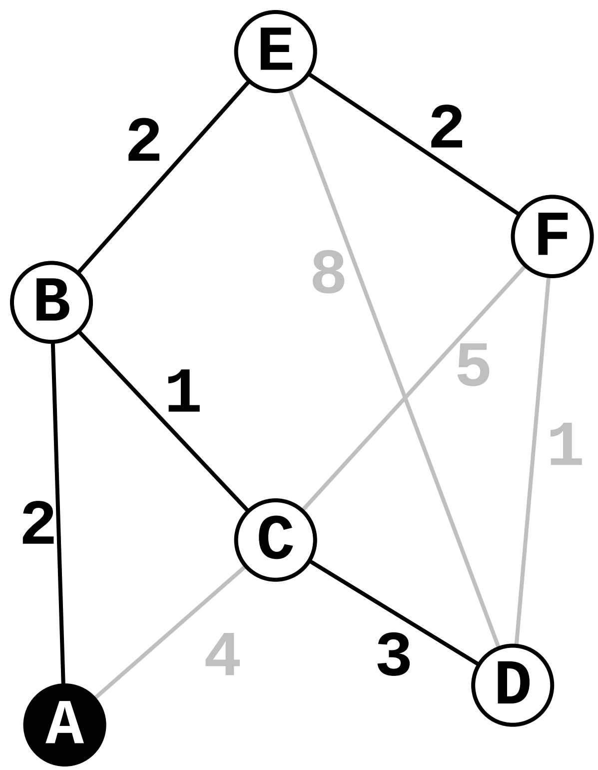
Eksempel



Nodeavstander

```
A: 0 (-)
B: 2 (A)
C: 3 (B)
D: 5 (F)
E: 8 (D)
F: 4 (C)
```

Korteste sti-tre



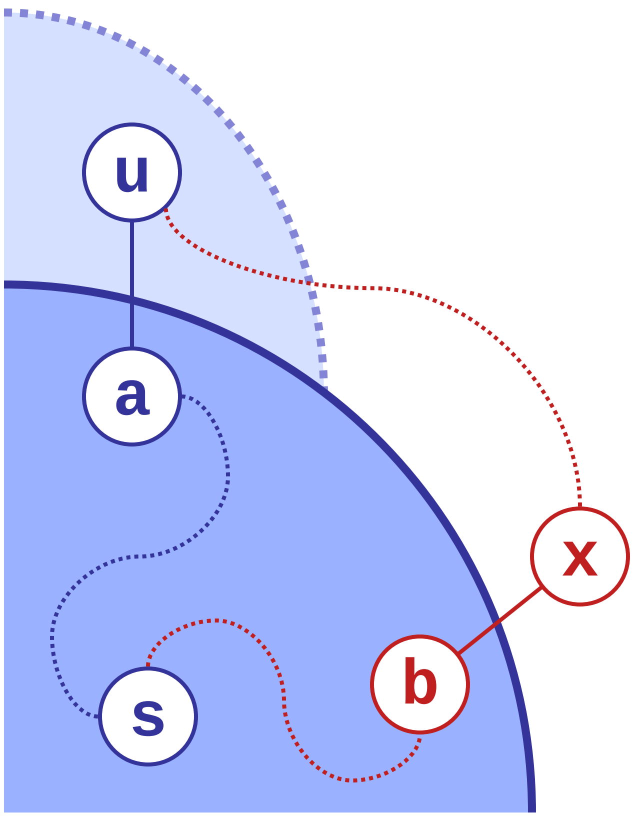
- I forrige eksempel lå alle nodene etter hverandre på den korteste stien
- Som regel vil man heller få et tre av korteste stier
- Et eksempel vises til høyre (grafen har andre vekter enn i sted)

- Merk at vi nå også har funnet korteste stier mellom alle noder i samme gren (B ~> E, B ~> F, E ~> F, B ~> C, B ~> D og C ~> D), men ikke mellom noder i forskjellige greiner (f.eks. E ~> C)

Korrekthet

- Vi må vise at når vi tar ut en node fra prioritetskøen, er estimatet til noden faktisk den korteste distansen
- Er det mulig å finne kortere veier?
- Når vi tar ut en node u fra prioritetskøen, vet vi at vi har undersøkt alle stier som har kortere lengde enn u sitt estimat
  - Dersom det fantes kortere stier, måtte de ha begynt fra en node x med kortere distanse
  - Men hvis x hadde kortere distanse enn u, ville x ha blitt tatt ut av prioritetskøen før u, og da ville den kortere veien ha blitt funnet

Illustrasjon av korrekthet



La oss tenke oss at Dijkstra gjør en tabbe på et tidspunkt, ved å legge en node u til de kjente nodene uten at estimatet til u faktisk er kortest. La oss se på situasjonen der vi tar ut u fra prioritetskøen og skal til å legge den til de kjente nodene. Vi vil bevise at det faktisk ikke er mulig at det finnes en annen vei som er kortere.

- Dersom det finnes en kortere vei fra s til u, kan vi spore den veien til det punktet hvor den går fra en kjent node b til en ukjent node x

- x ligger i prioritetskøen siden den er ukjent

- b er en kjent node, med korrekt korteste avstand (siden u er den første noden der vi gjorde en tabbe)

- Da b ble lagt til, oppdaterte vi estimatet til alle nabonodene, deriblant x. Dermed har x også riktig estimat, siden b -> x ligger på en korteste vei

- **Dersom alle kantvektene er positive, må x ha et estimat som er kortere enn u sitt**

- Men da skulle x ha blitt tatt ut fra prioritetskøen FØR u!

- Vi har altså en selvmotsigelse, så det er ikke mulig at det finnes kortere stier når vi tar ut en node

Negative kantvekter

- Merk argumentet på forrige slide:
  - "Dersom alle kantvektene er positive…"
  - Essensen er at noder på den korteste veien til u må ha kortere estimat enn u
- Dersom det finnes negative kanter, gjelder ikke beviset lenger!
  - Det blir da mulig å gå fra en node med estimat d, følge en negativ kant, og ende opp med en distanse som er mindre enn d
- Konklusjon: Dijkstras algoritme fungerer *ikke* når grafen inneholder negative kanter!

Forside
Korteste sti
BFS
Modifikasjon
Dijkstra
Eksempel
Korrekthet
:: Analyse
Øving
Spørsmål

Analyse

```
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 S ← ∅
3 Q ← V[G]
4 while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for each vertex v ∈ Adj[u]
8            do RELAX(u, v, w)
```

- Konstruksjon av Q med $|V|$ elementer (3)
- While-løkka kjøres $|V|$ ganger
  - Ett uttak fra Q hver gang (5)
- For-løkka kjøres totalt $|E|$ ganger (7)
  - I verste fall én oppdatering i Q hver gang (8)

Forside
Korteste sti
BFS
Modifikasjon
Dijkstra
Eksempel
Korrekthet
:: Analyse
Øving
Spørsmål

Valg av prioritetskø-struktur

- Med n elementer i prioritetskøen har vi følgende kompleksiteter:
- Usortert array
  - Konstruksjon: $O(V)$
  - Uttak: $O(n)$
  - Oppdatering: $O(1)$
- Heap
  - Konstruksjon: $O(V)$
  - Uttak: $O(\lg n)$
  - Oppdatering: $O(\lg n)$

Valg av prioritetskø-struktur

- Vi hadde altså konstruksjon, V uttak og E oppdateringer
- Usortert array: totalt $O(V + V^2 + E) = O(V^2)$
  - Konstruksjon: $O(V)$
  - Uttak: $O(V^2)$
  - Oppdatering: $O(E)$
- Heap: totalt $O(V + V \lg V + E \lg V) = O(E \lg V)$
  - Konstruksjon: $O(V)$
  - Uttak: $O(V \lg V)$
  - Oppdatering: $O(E \lg V)$

Valg av prioritetskø-struktur

- Dersom heap skal lønne seg, må $E \lg V = o(V^2)$ (altså $E \lg V < V^2$)
- Generelt sett kan vi ikke si noe annet enn at $E = O(V^2)$ (maksimalt én kant mellom hvert par av noder)
- Hvis vi vet hvor mange noder og kanter en bestemt graf har, kan vi finne tettheten (formelen fra øving 5): $T = E / V^2$
- Dersom $T < 1 / \lg V$, lønner det seg å bruke heap

Øving 8: Mumien

- Her går det ut på å finne "mest sannsynlige vei"
- Hver node har en viss sannsynlighet
- Hint:
  - Sannsynligheten til en kant er lik sannsynligheten til noden kanten går til
  - Problem: Finne en vei der *produktet* av sannsynlighetene er *størst* mulig
  - Klarer vi å transformere dette til et problem om å finne en vei der en *sum* skal være *minst* mulig?

Spørsmål?