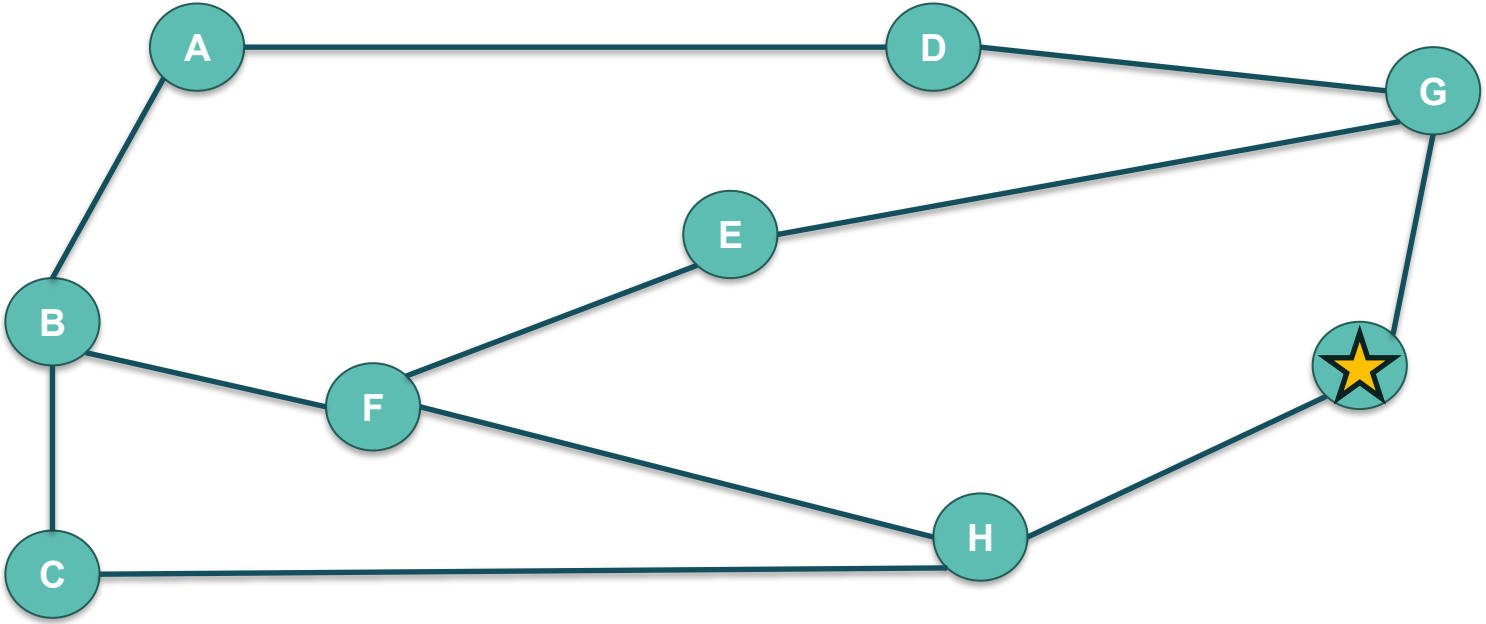
Vektede Grafer

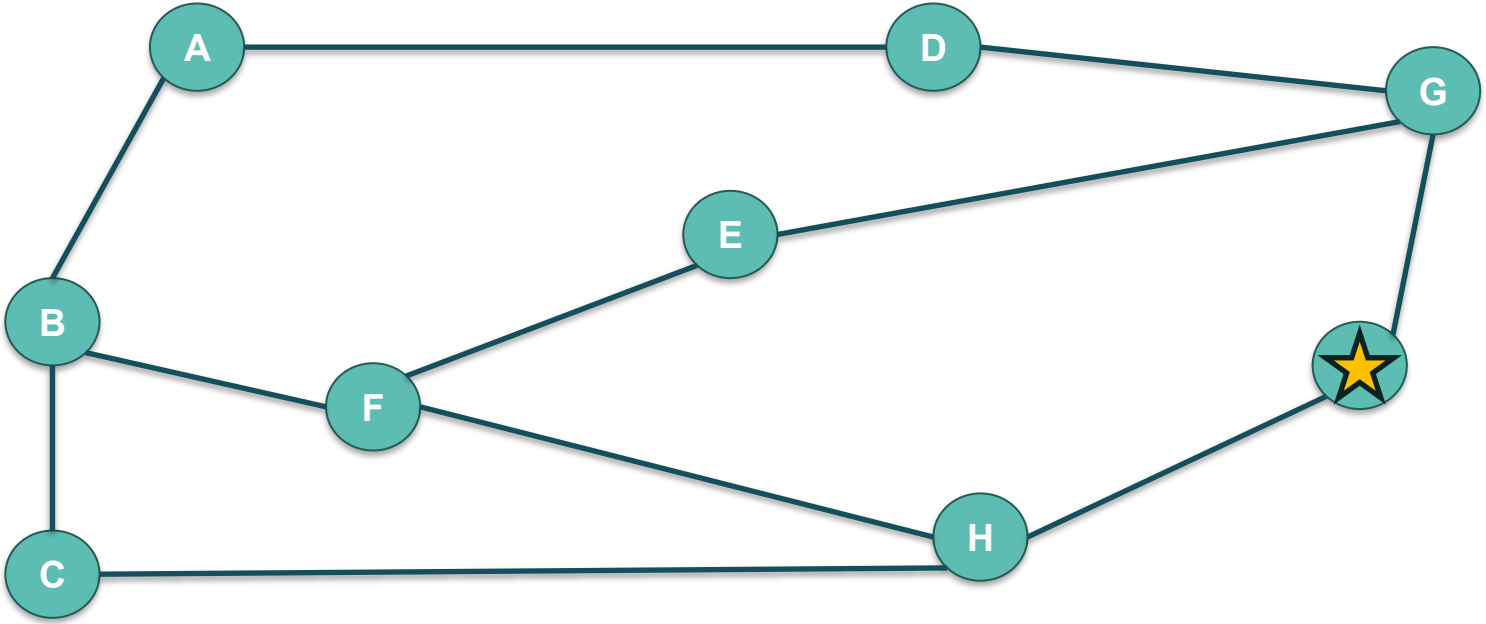


IN2010 – Uke 7

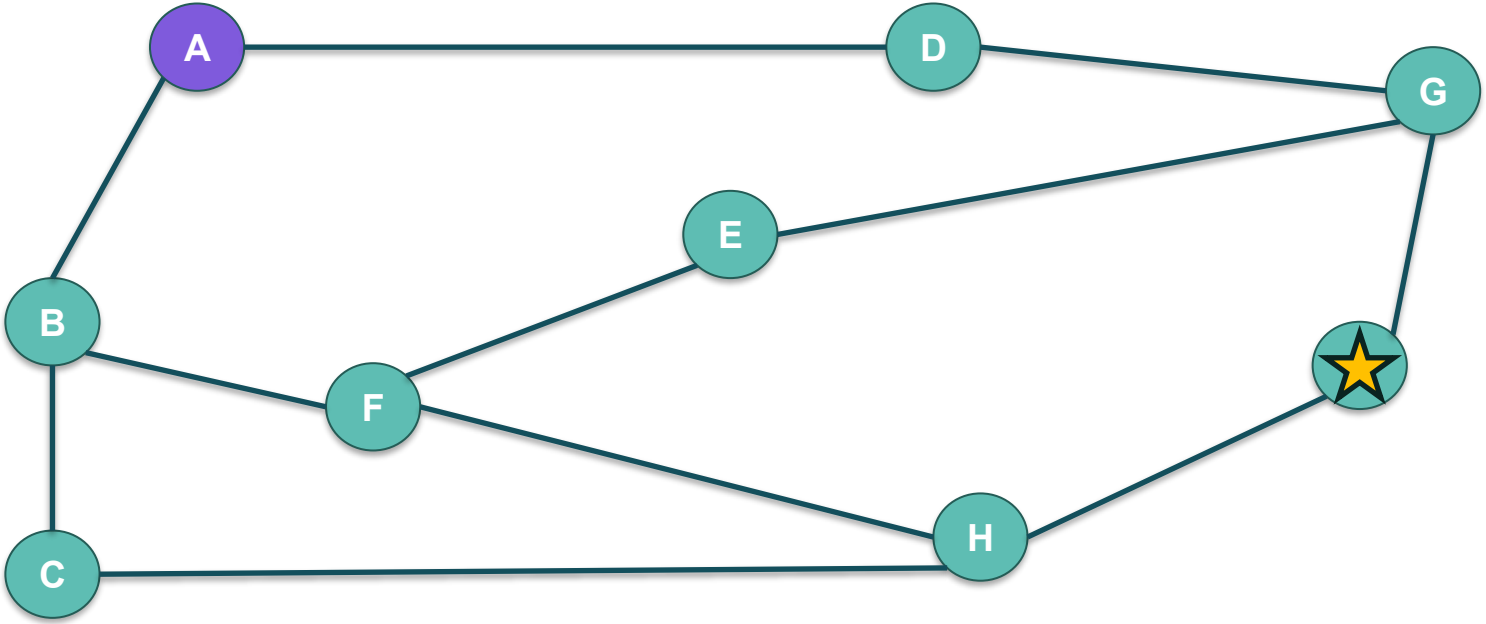
Grafer vs vektede grafer



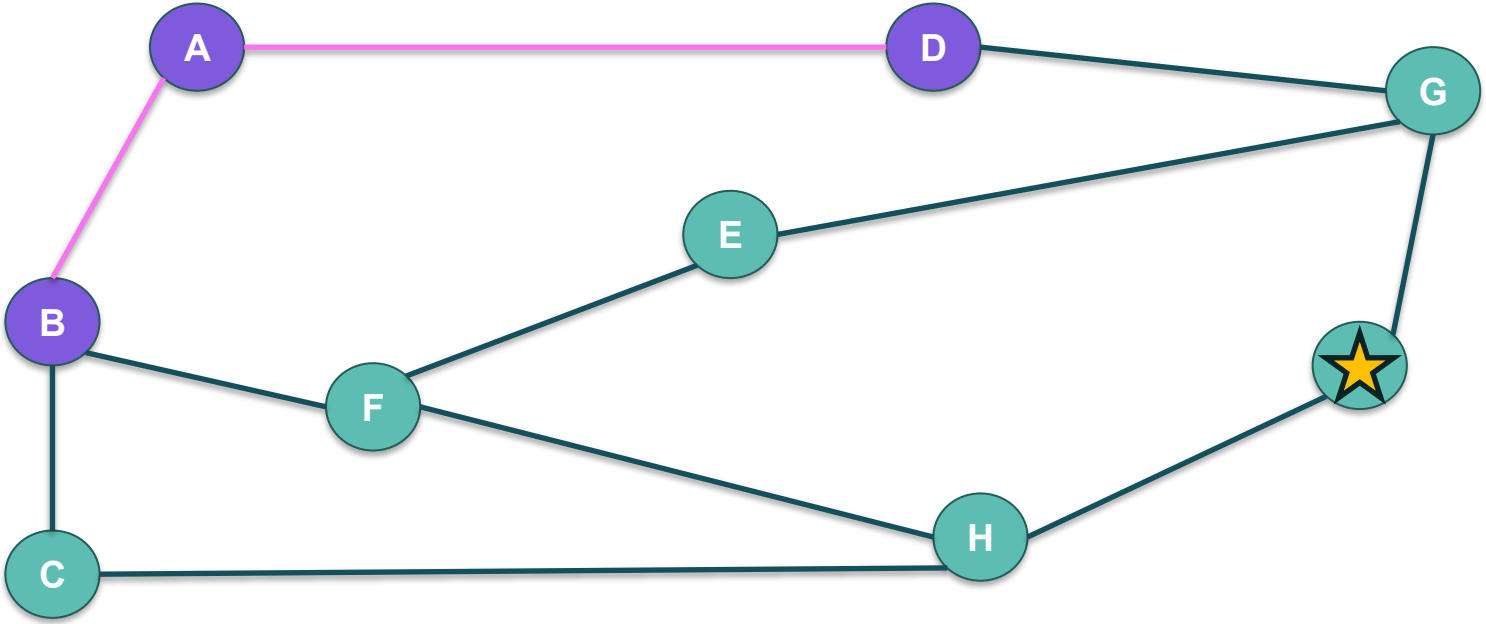
Korteste stien



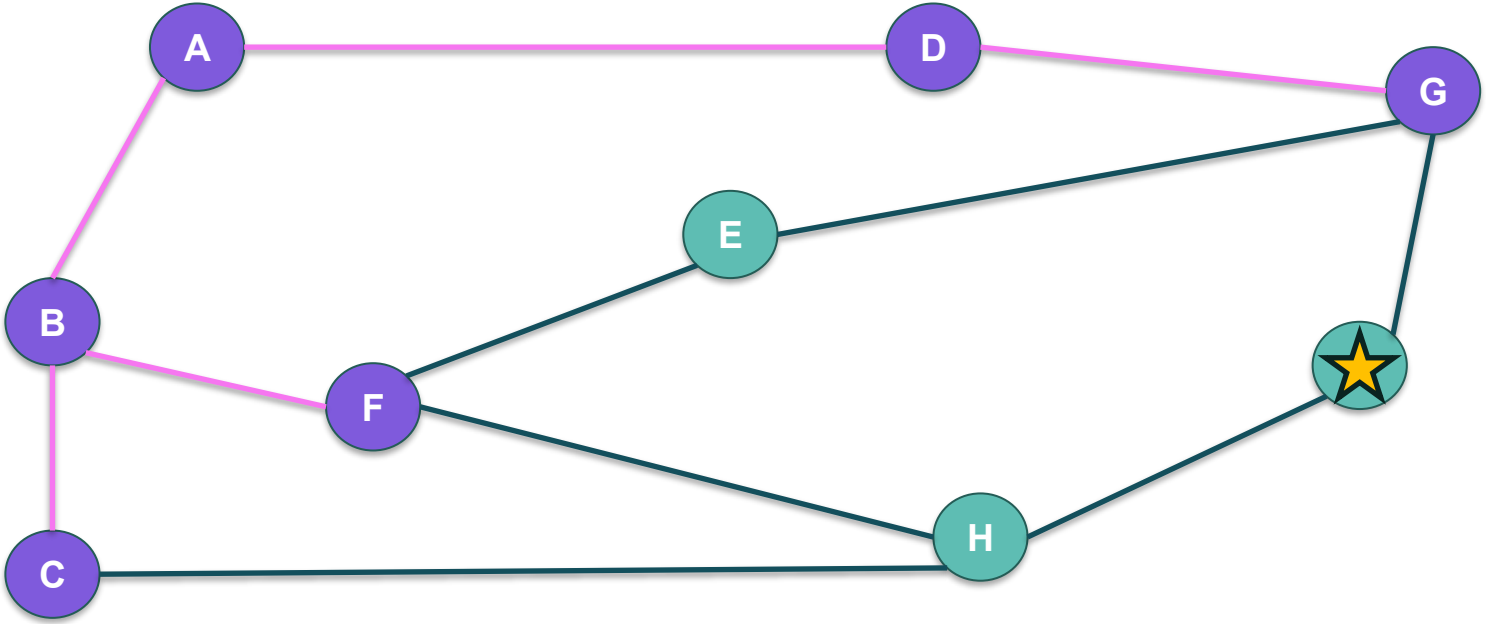
Korteste stien



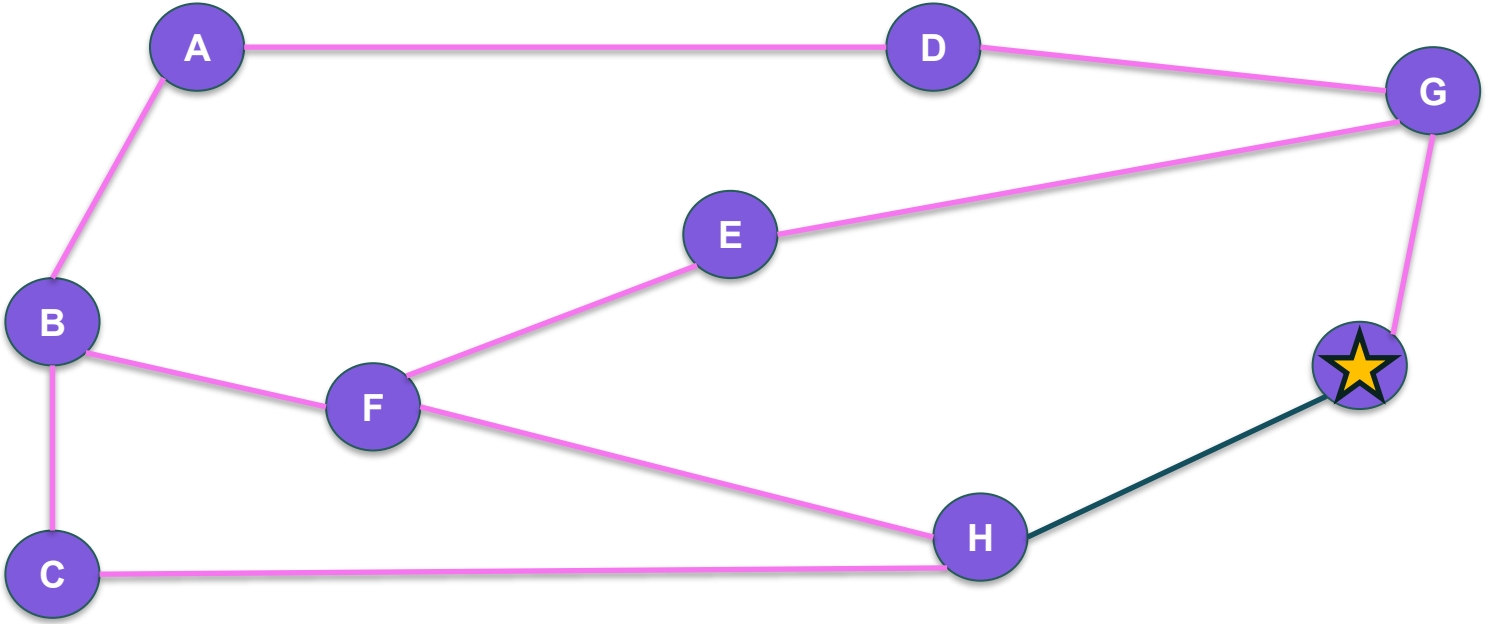
Korteste stien



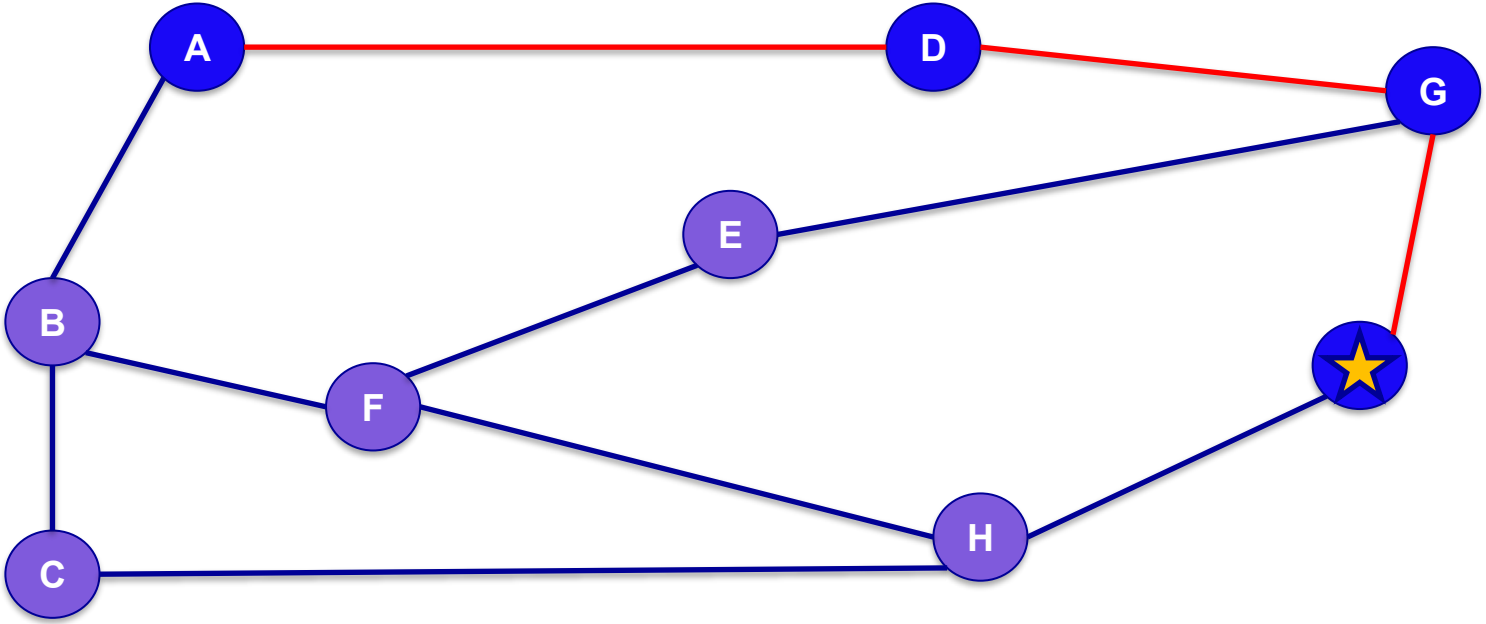
Korteste stien



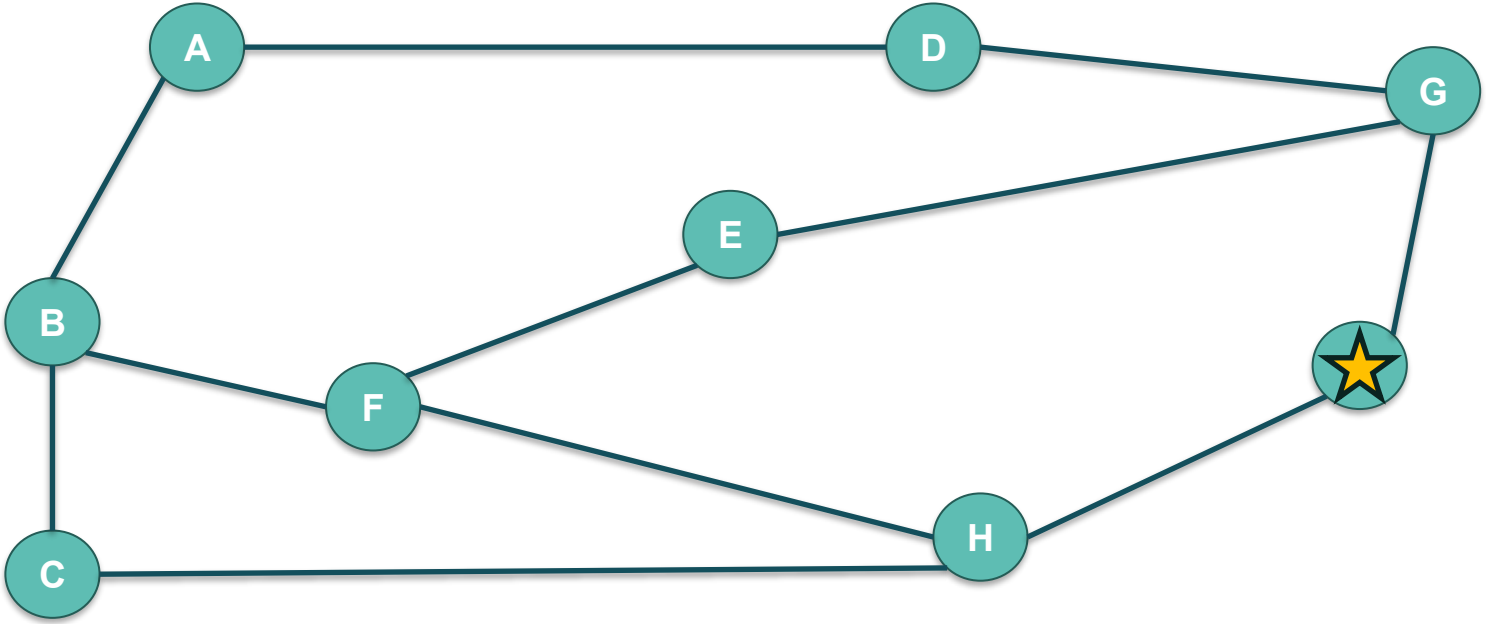
Korteste stien

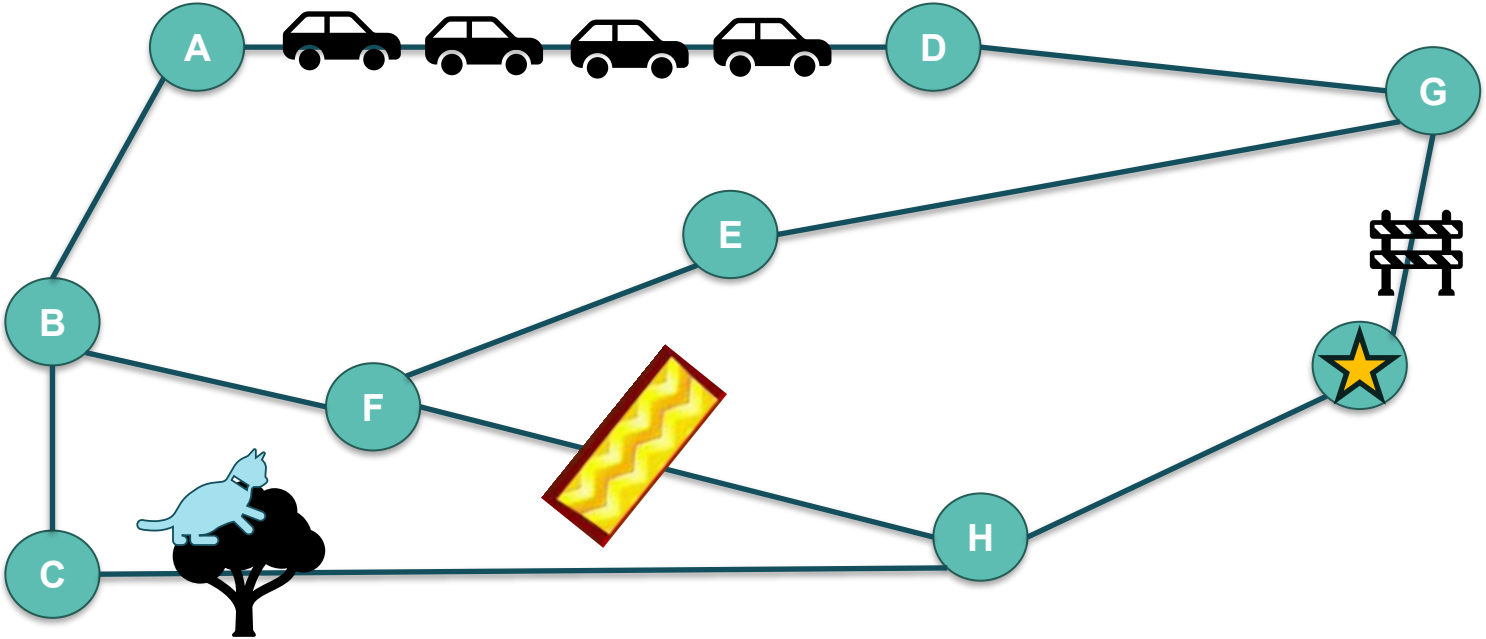


Korteste stien

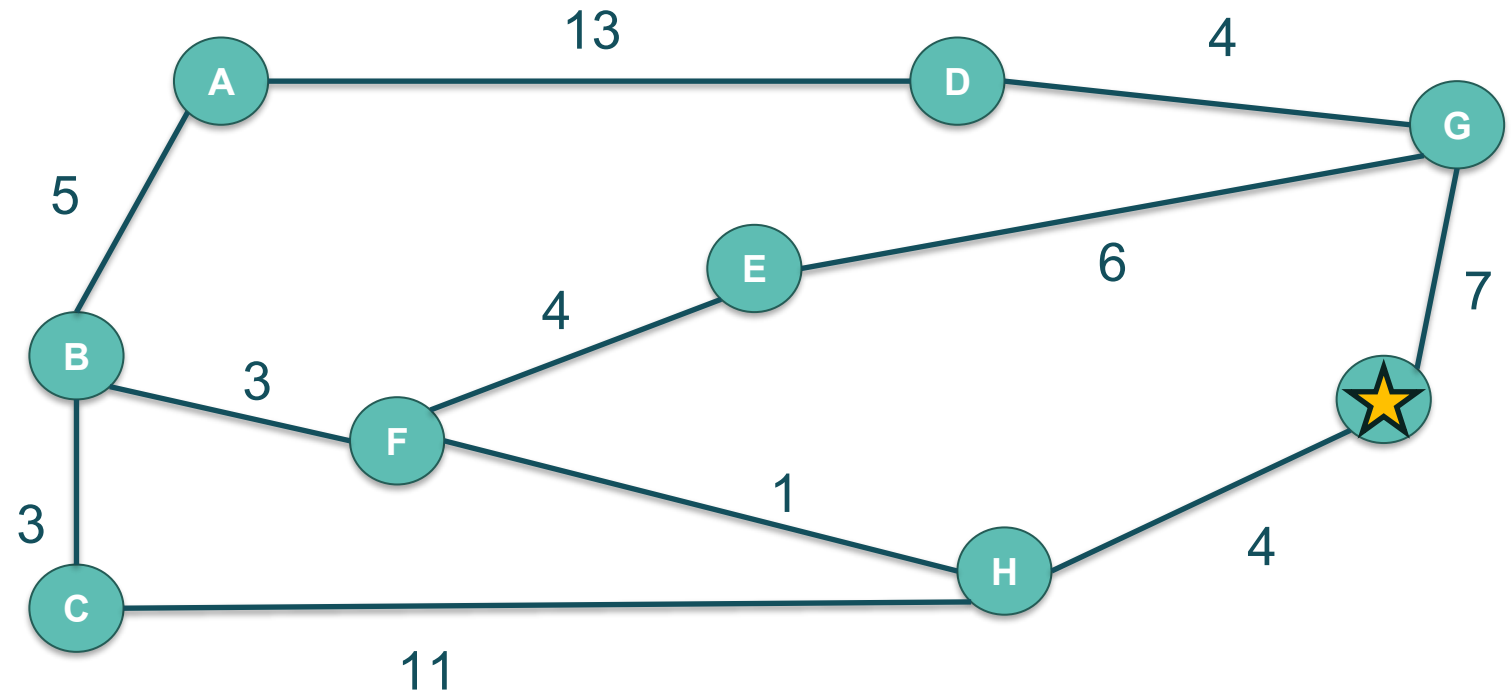


Men hva om den er vektet?




A
D
G
E
B
F
H
C

Hva blir den korteste stien nå? (3-4 min)



Løsningen? Dijkstra!!!

Ukas video

God forklaring av Dijkstra (OBS: Litt annerledes implementasjon enn pseudokoden i forelesning)



https://youtu.be/GazC3A4OQTE

Dijkstra i praksis

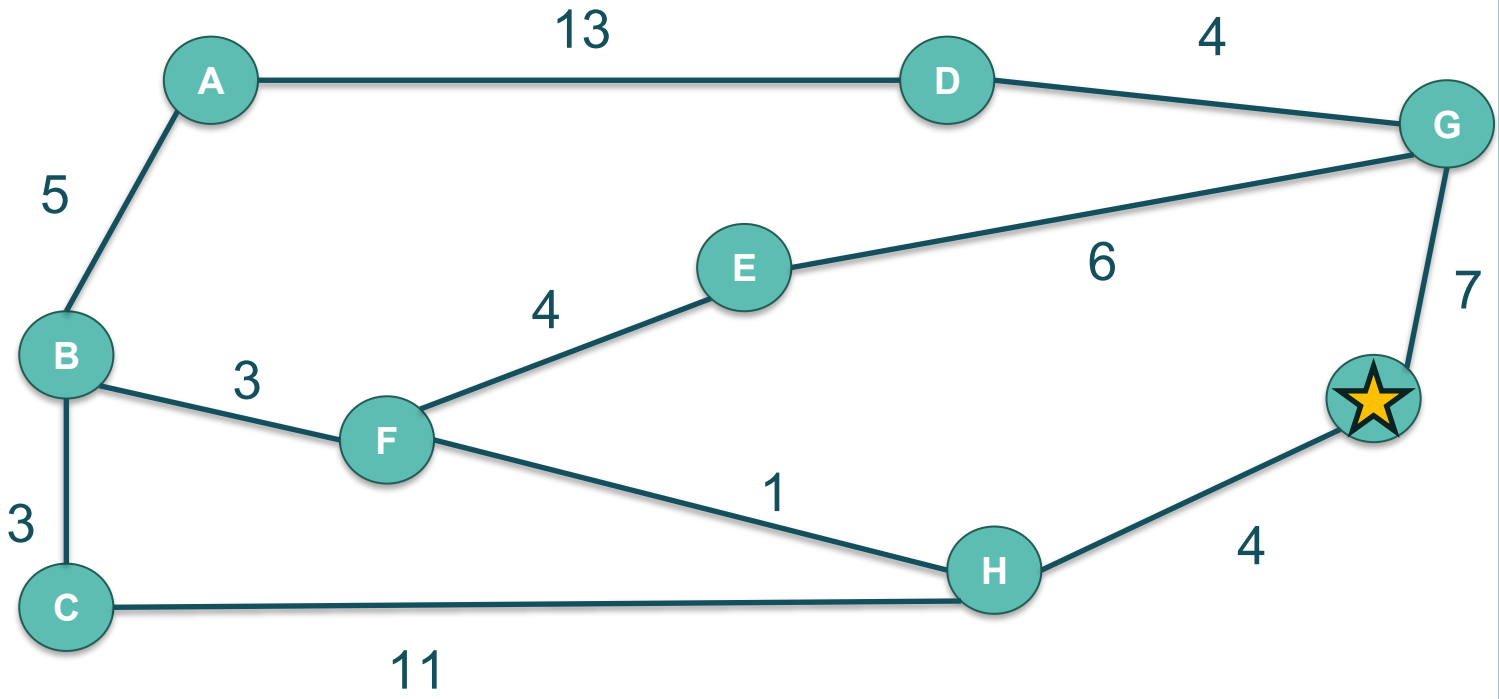
Input:
Vektet graf G = (V,E)
En vektfunksjon w: w(u,v) = vekt
Startnode s

Output:
Dist (distances): Et map med korteste vei fra s til alle noder i G

ALTSÅ: Om du vil finne korteste veien fra s til en annen node, må du enten sjekke det ferdige dist-mappet

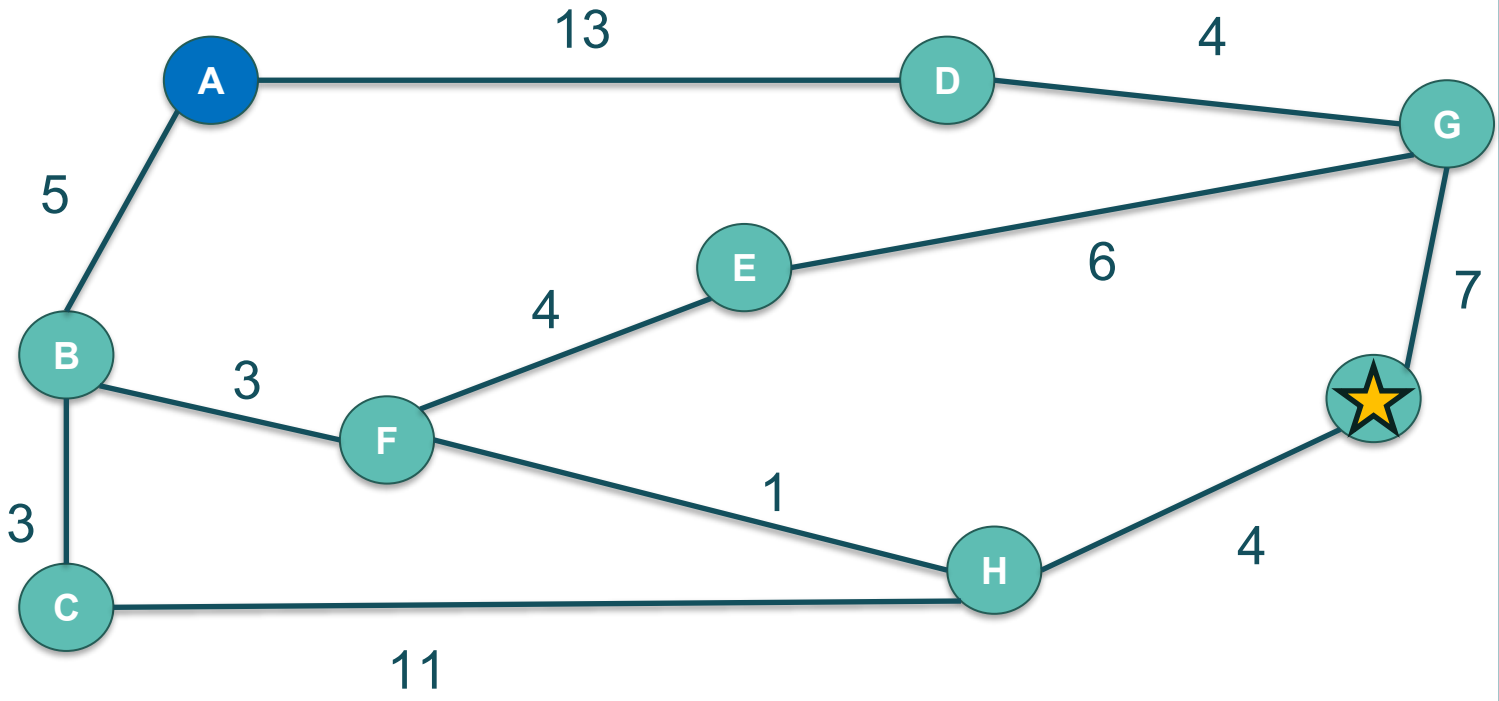
ELLER
Prøve å lagre den korteste veien underveis (<-Relevant til obligen)

Queue:



Queue:

A	0
B	∞
C	∞
D	∞
…	…
★	∞



Queue:

B | ∞

C | ∞

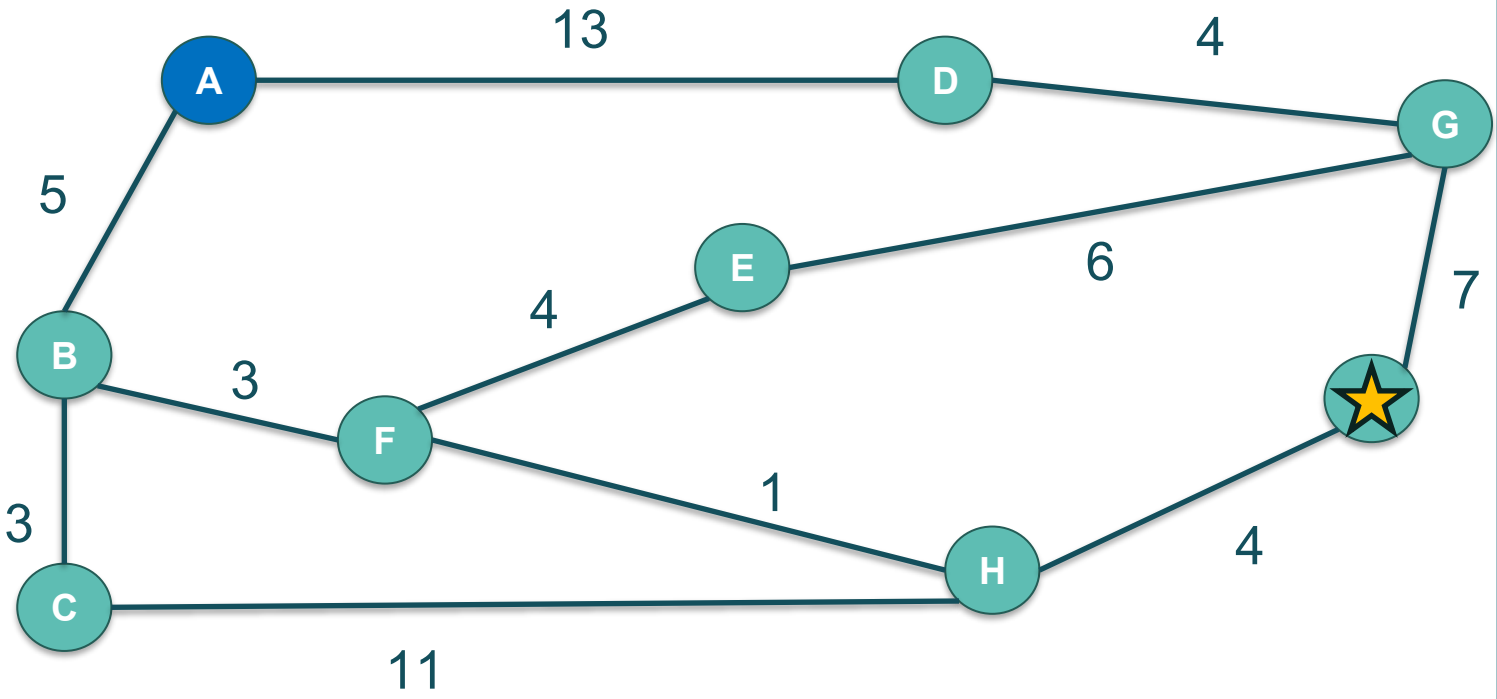
D | ∞

… | …

★ | ∞

A | 0

Hvor kan vi reise fra ‘A’?



Queue:

B | ∞

C | ∞

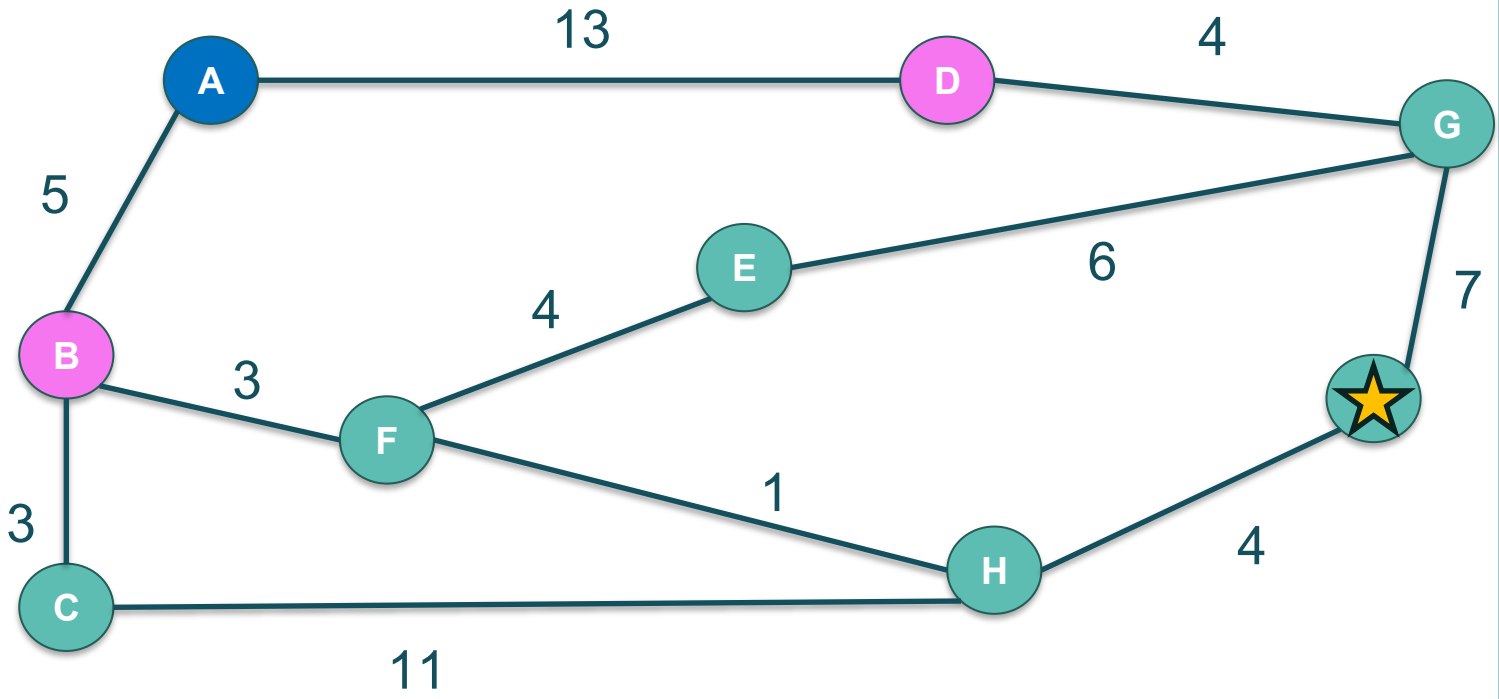
D | ∞

… | …

★ | ∞

A | 0

Hvor kan vi reise fra ‘A’?



Queue:

B | 5

C | ∞

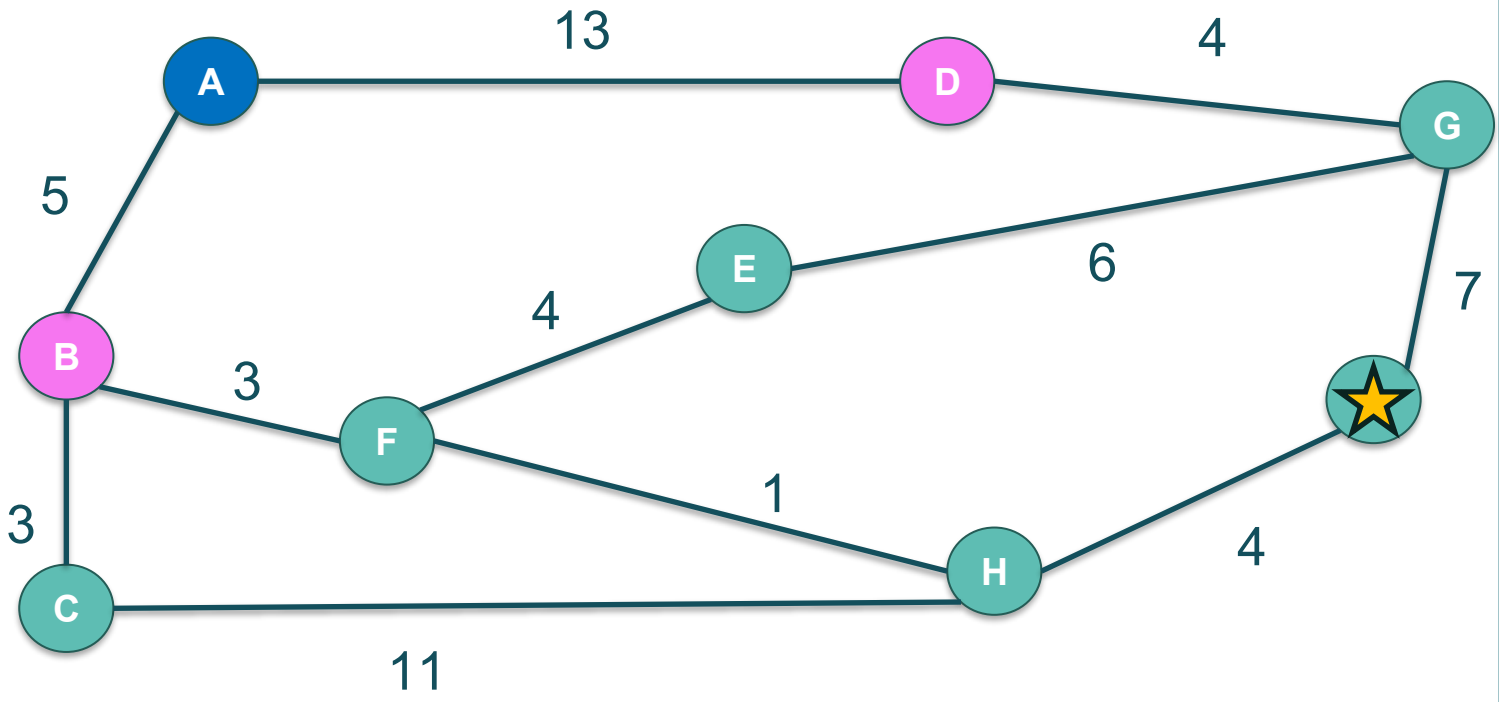
D | 13

… | …

★ | ∞

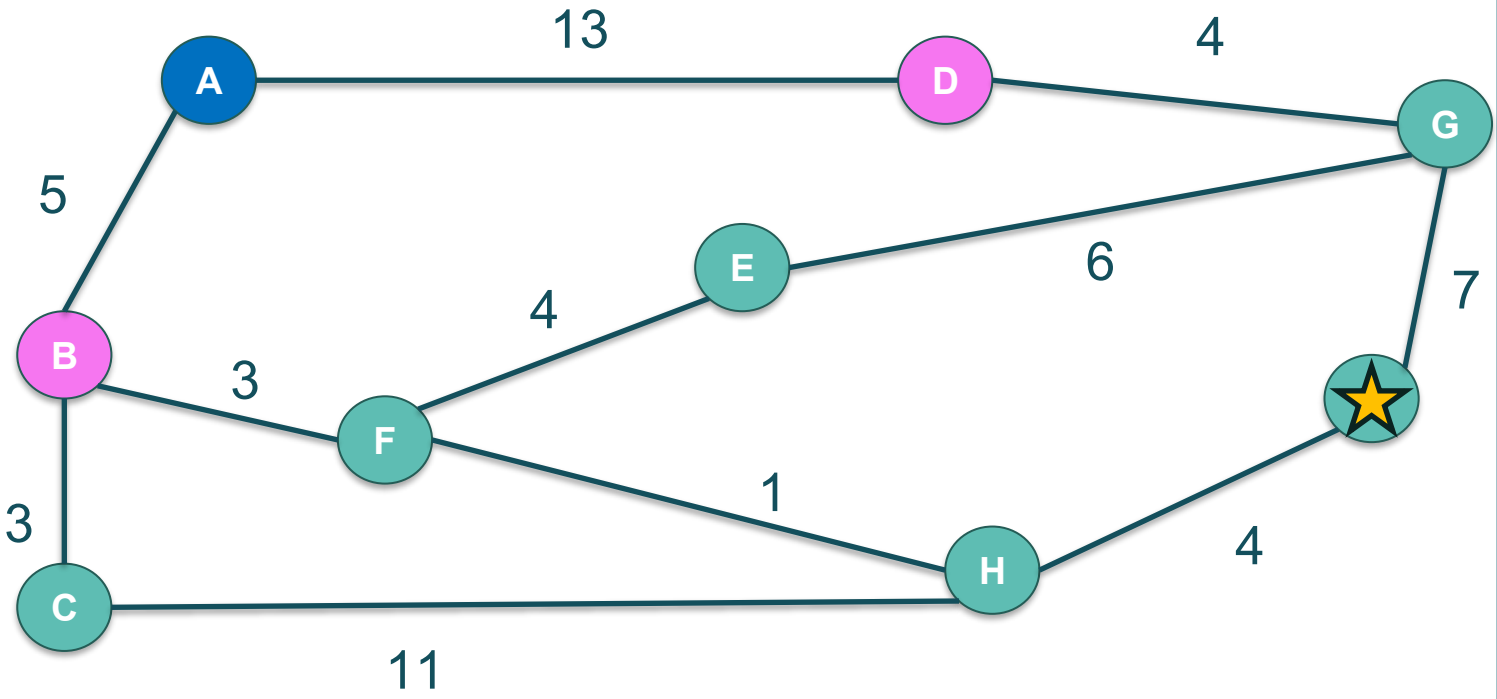
A | 0

Hvor kan vi reise fra ‘A’?



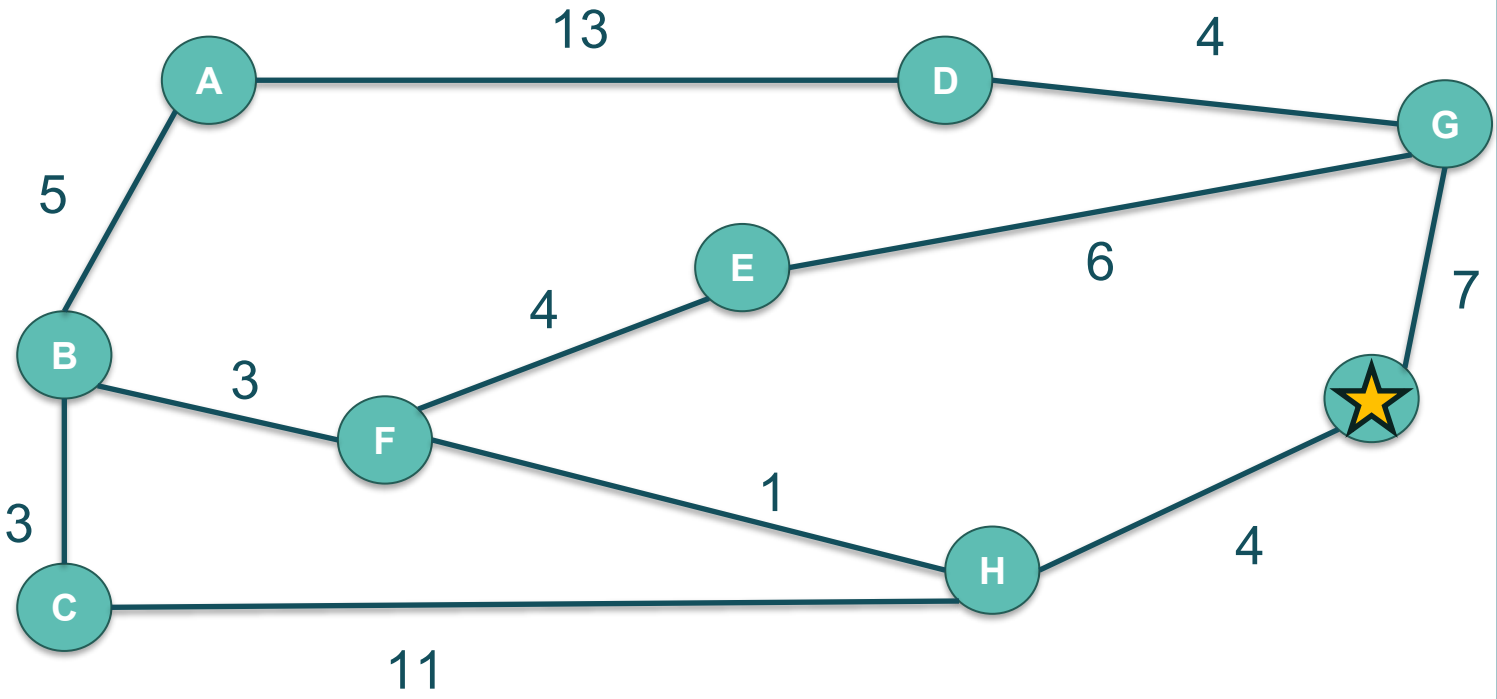
Queue:

B	5
D	13
C	∞
...	...
★	∞



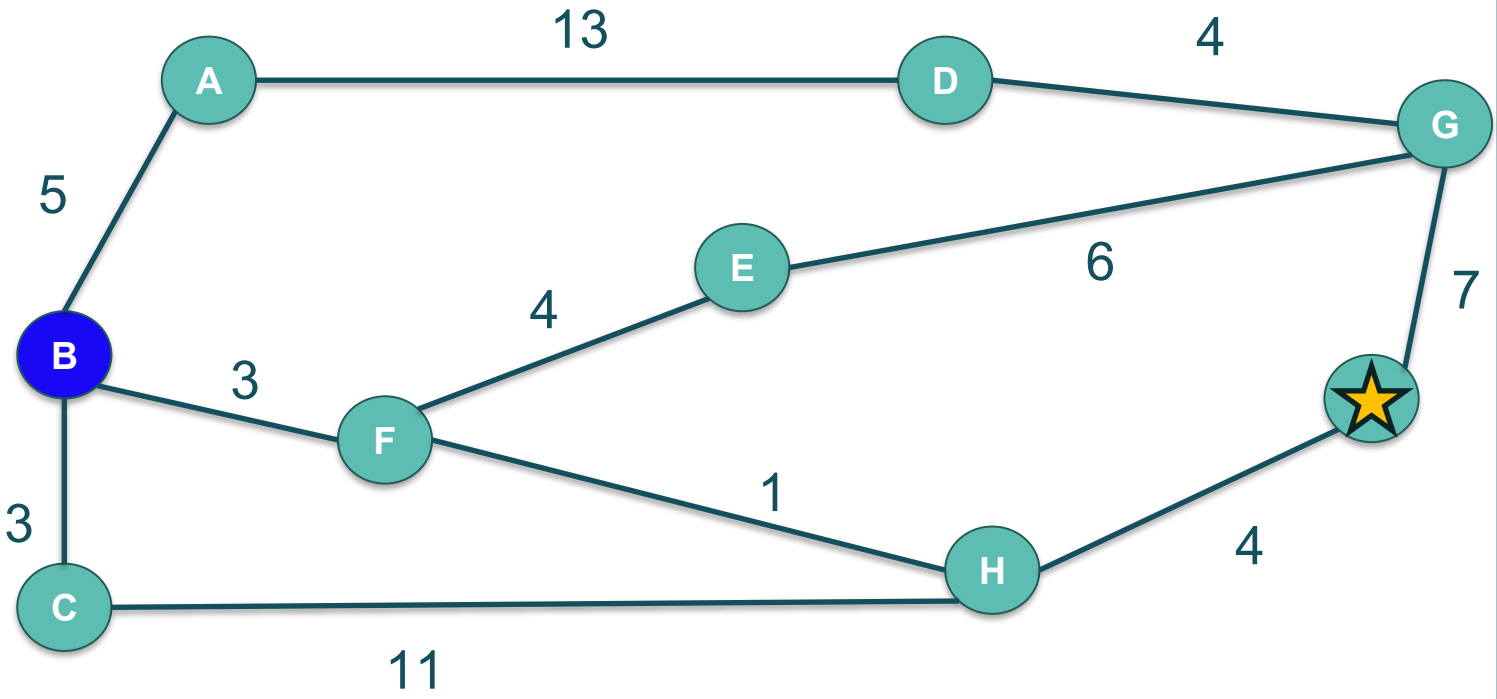
Queue:

B	5
D	13
C	∞
…	…
★	∞



Queue:

B	5
D	13
C	∞
…	…
★	∞



Queue:

D | 13

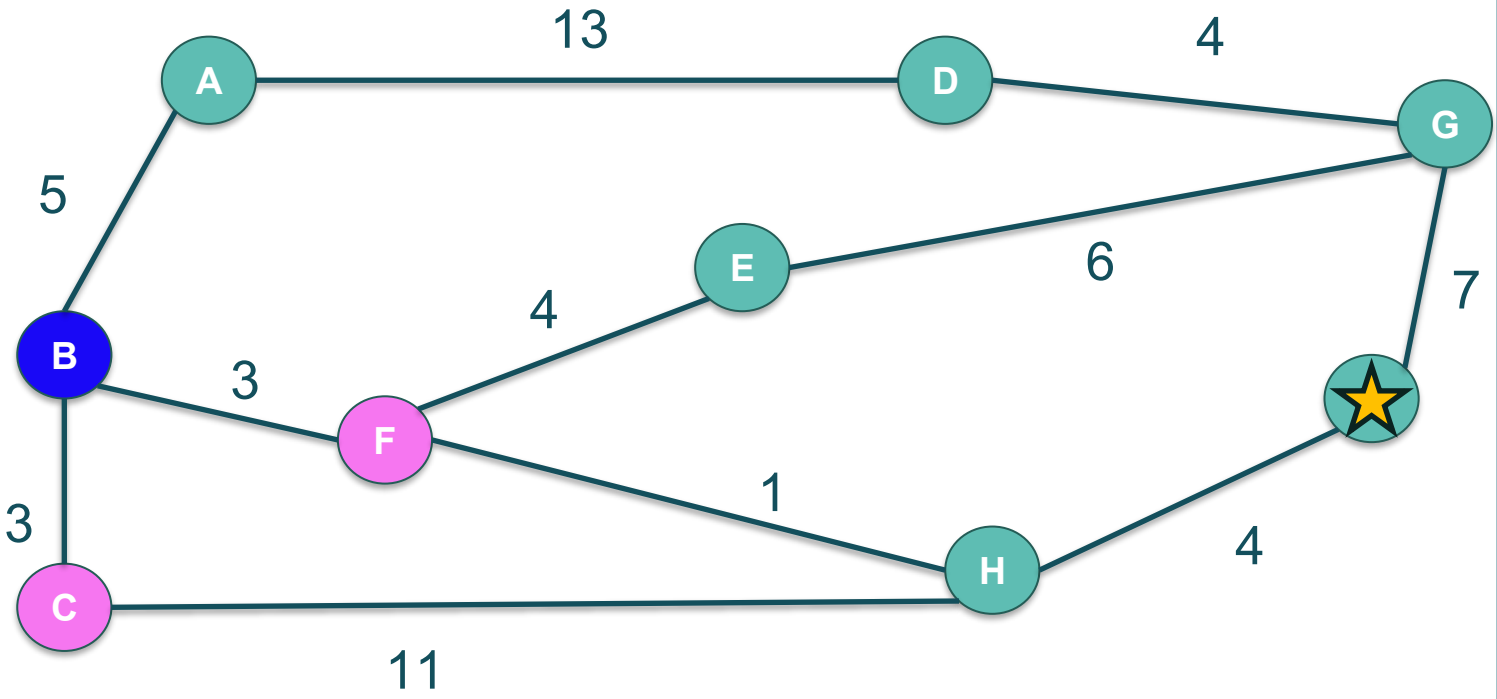
C | ∞

F | ∞

... | ...

★ | ∞

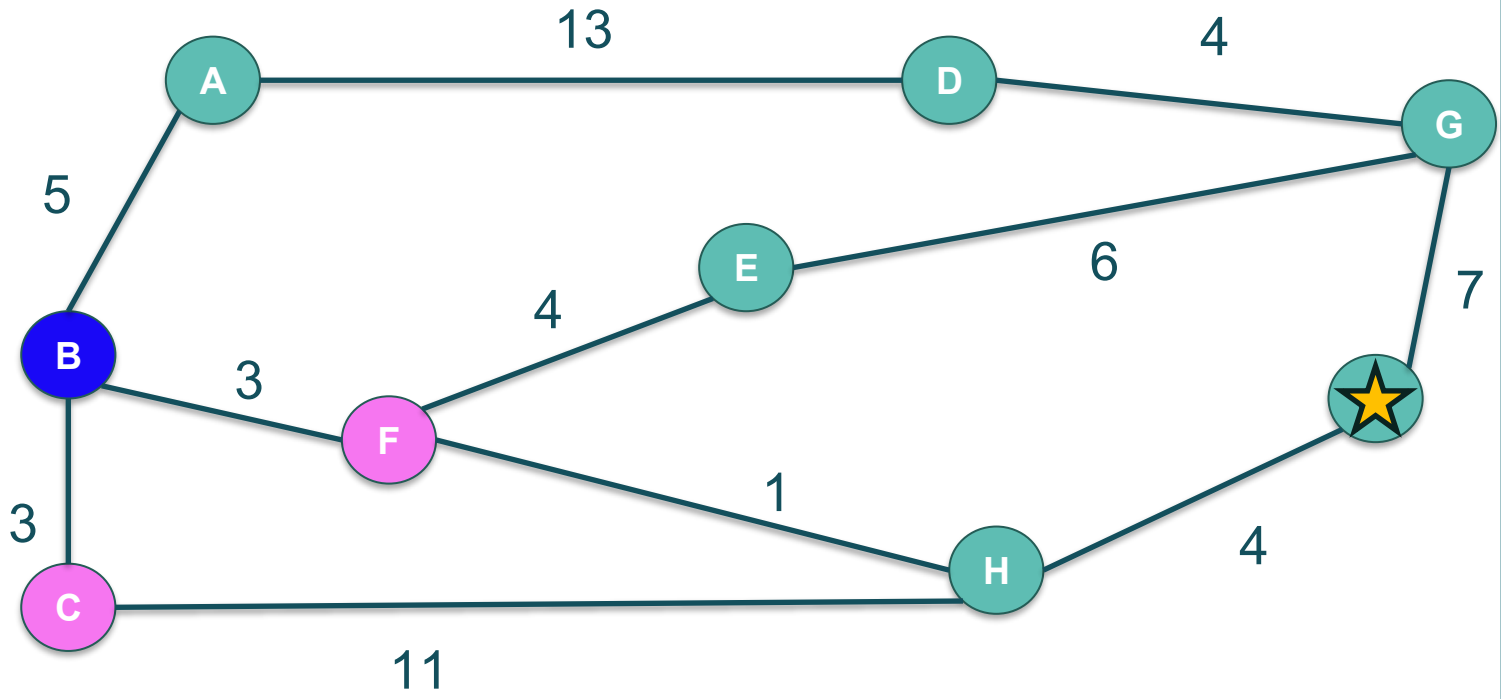
B | 5



Queue:

D	13
C	8
F	8
...	...
★	∞

B | 5



Queue:

B | 5

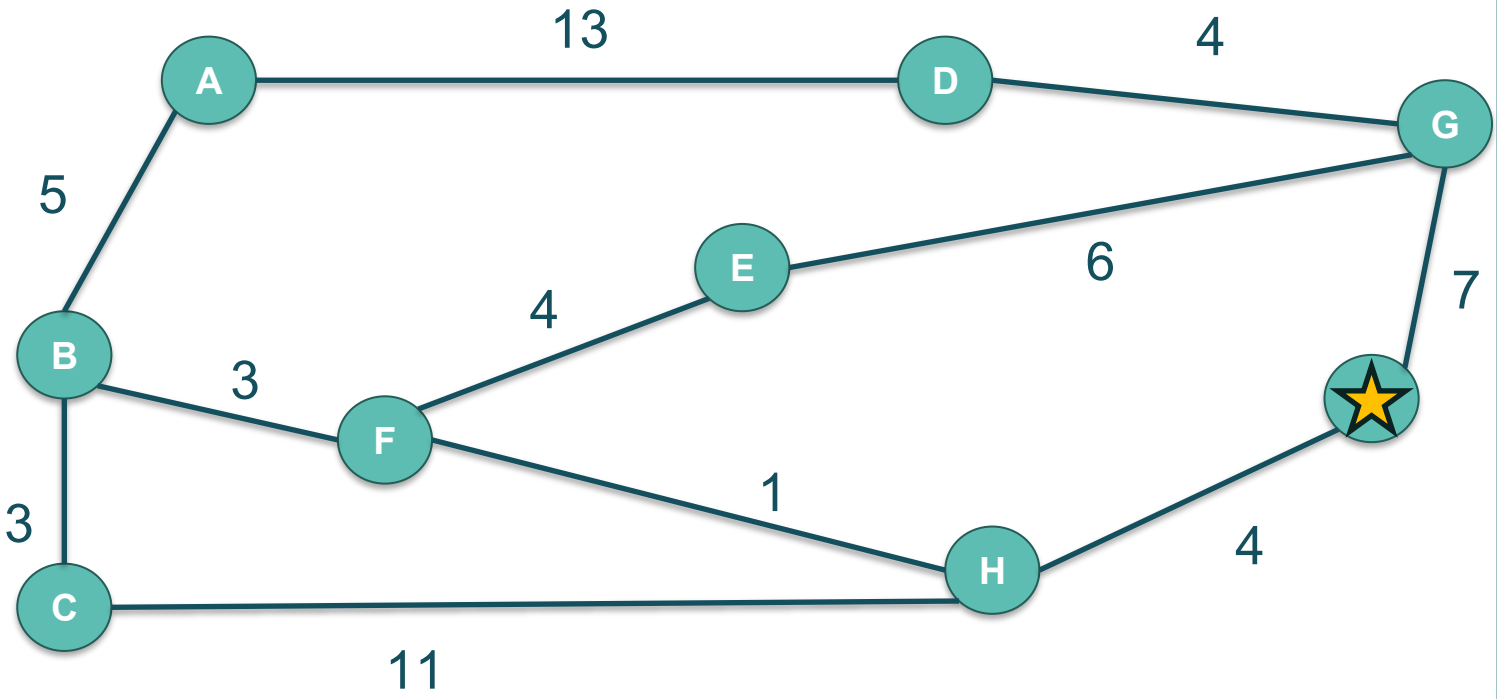
D | 13

C | 8

F | 8

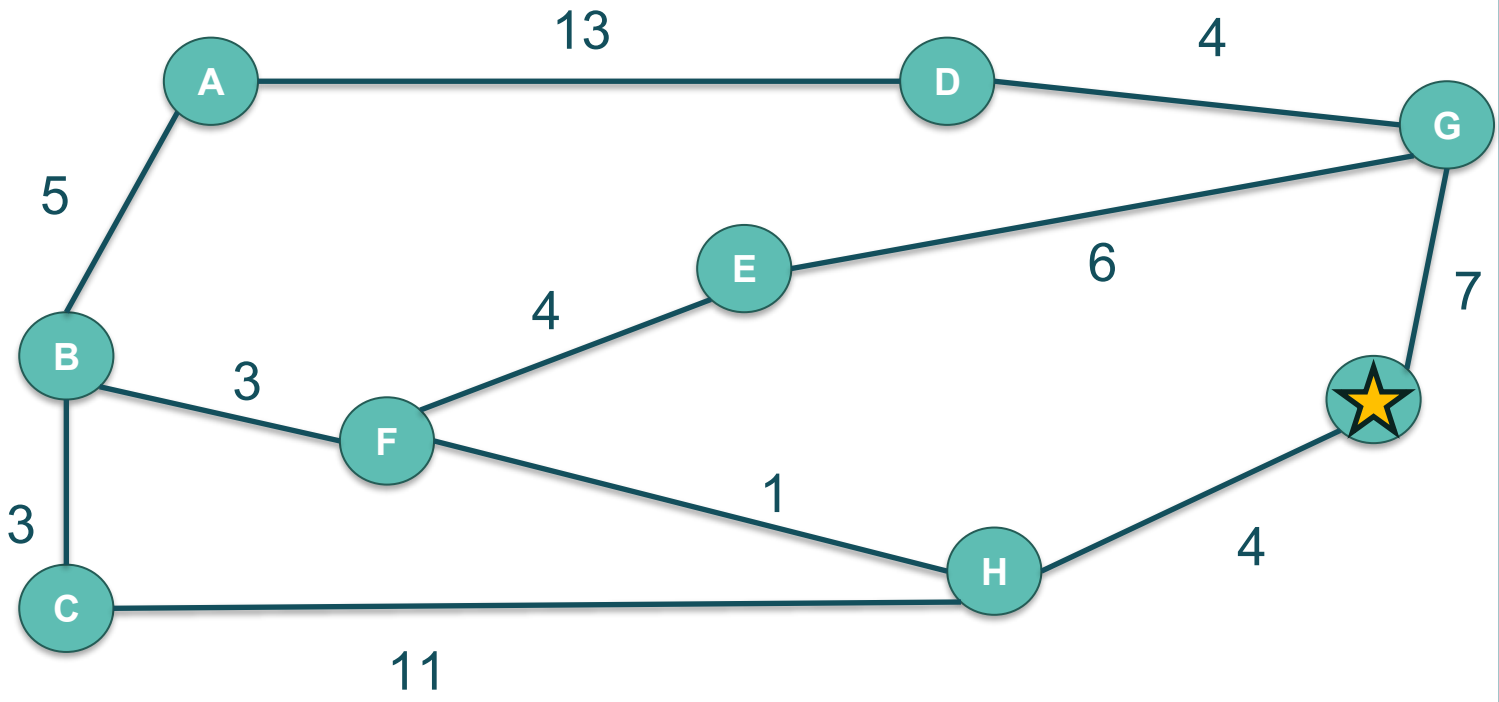
... | ...

★ | ∞



Queue:

C	8
F	8
D	13
…	…
★	∞



Queue:

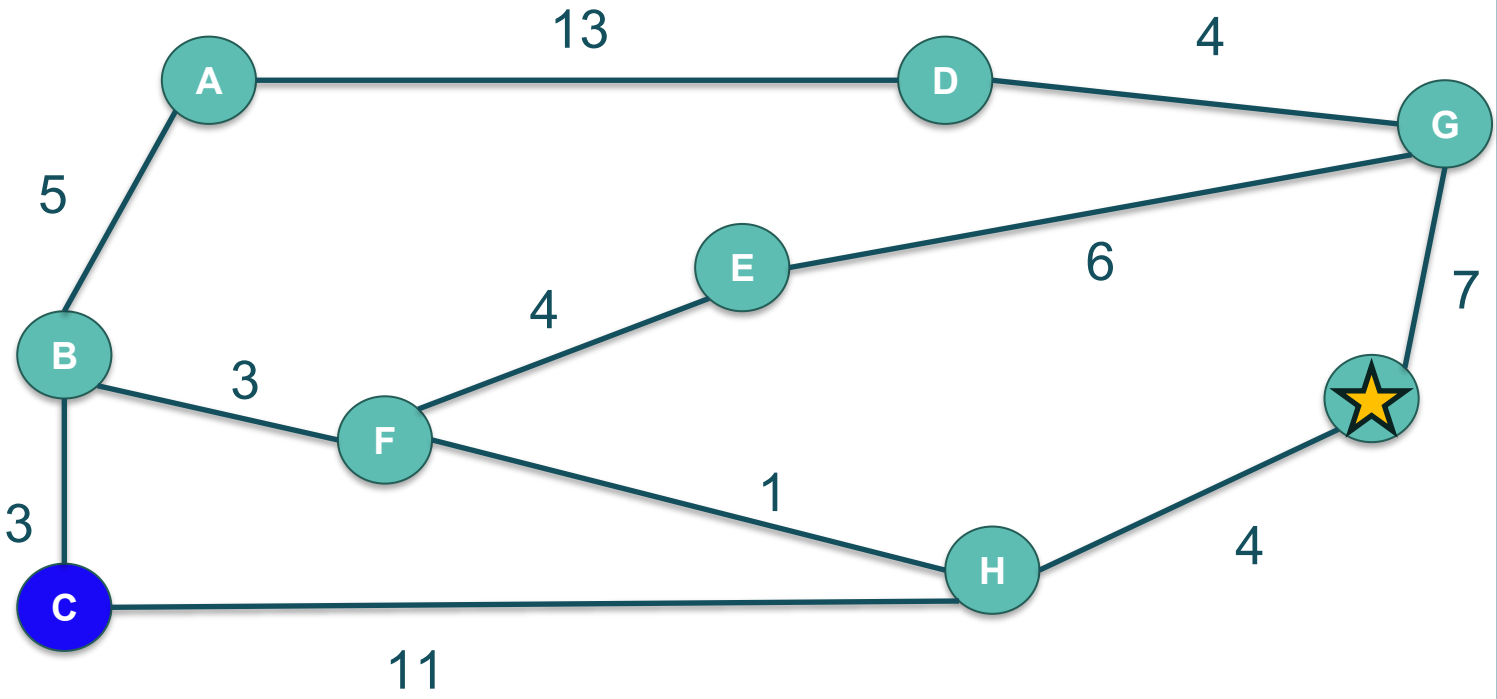
F | 8

D | 13

... | ...

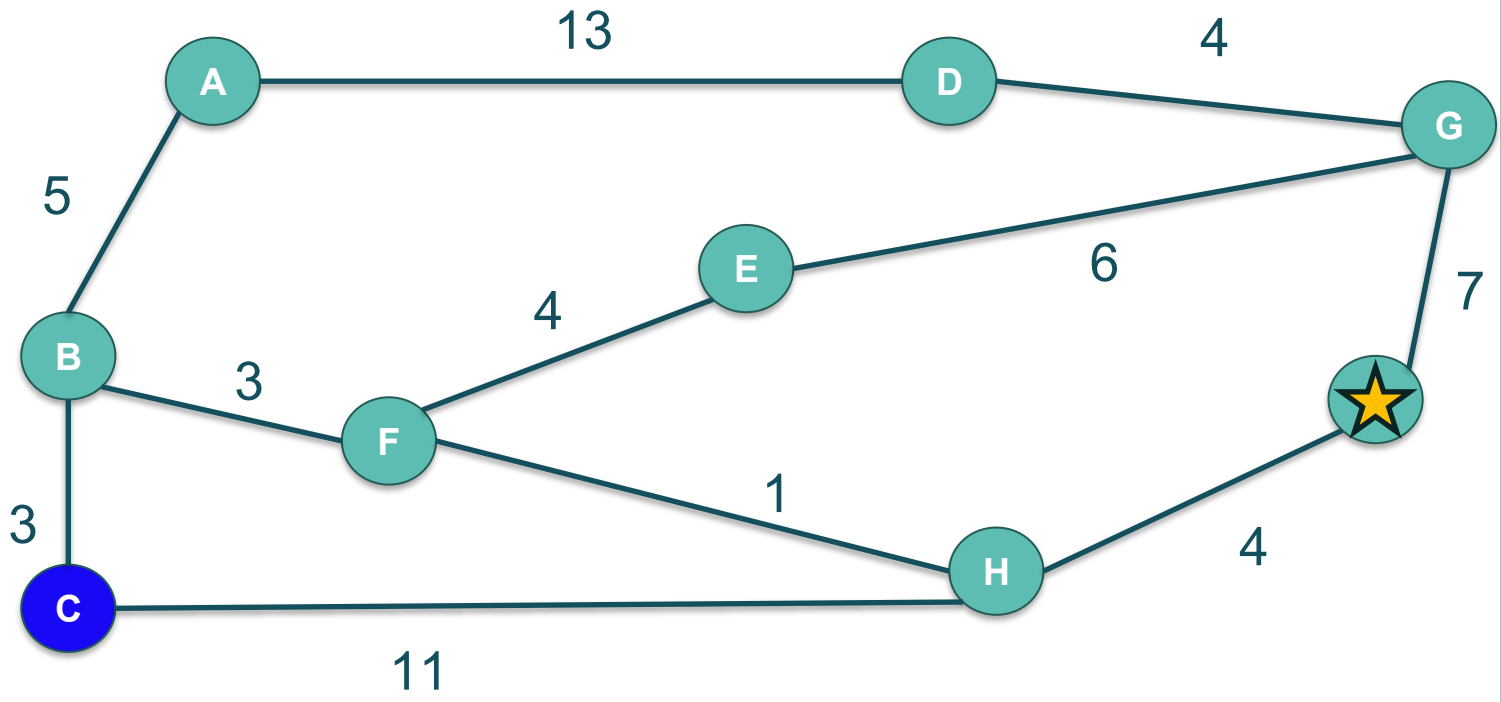
★ | ∞

C | 8



Queue:

F	8
D	13
H	19
…	…
★	∞



Queue:

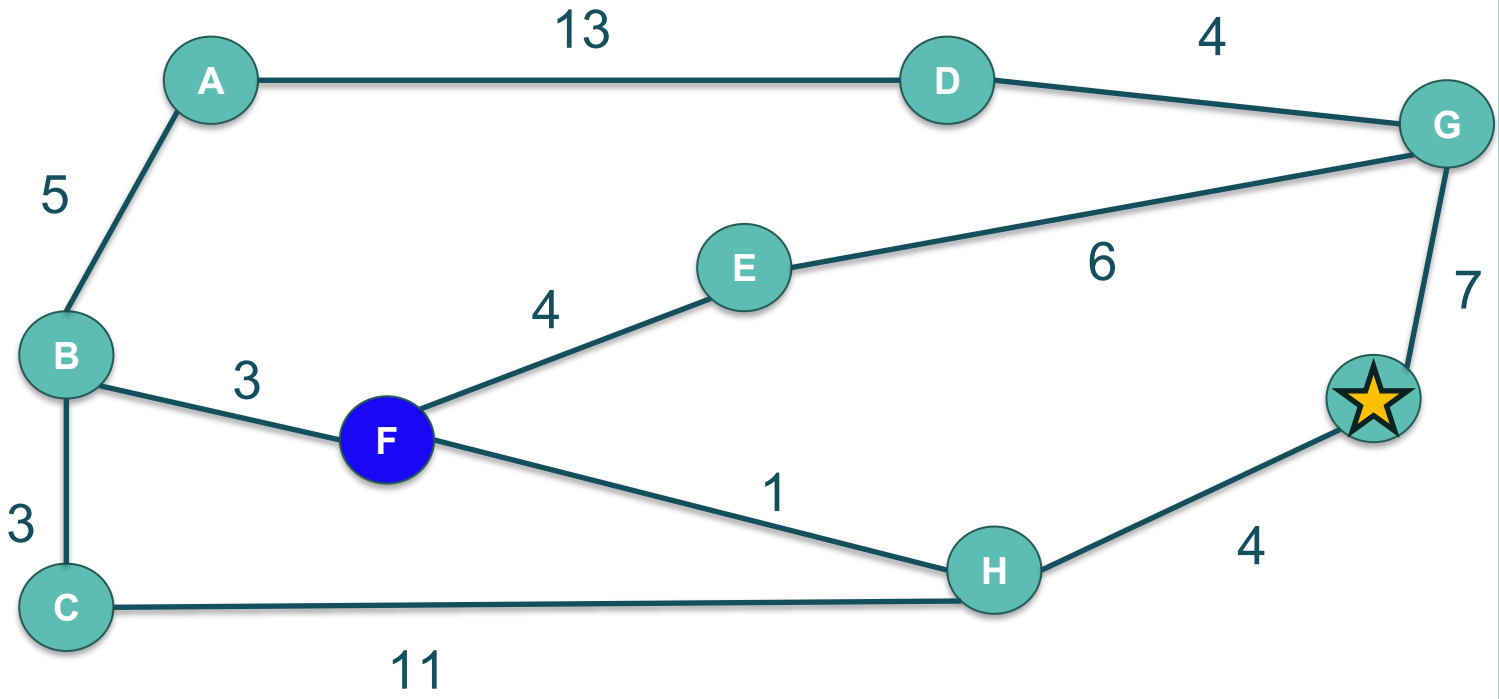
D	13

H	19

…	…

★	∞

F	8



Queue:

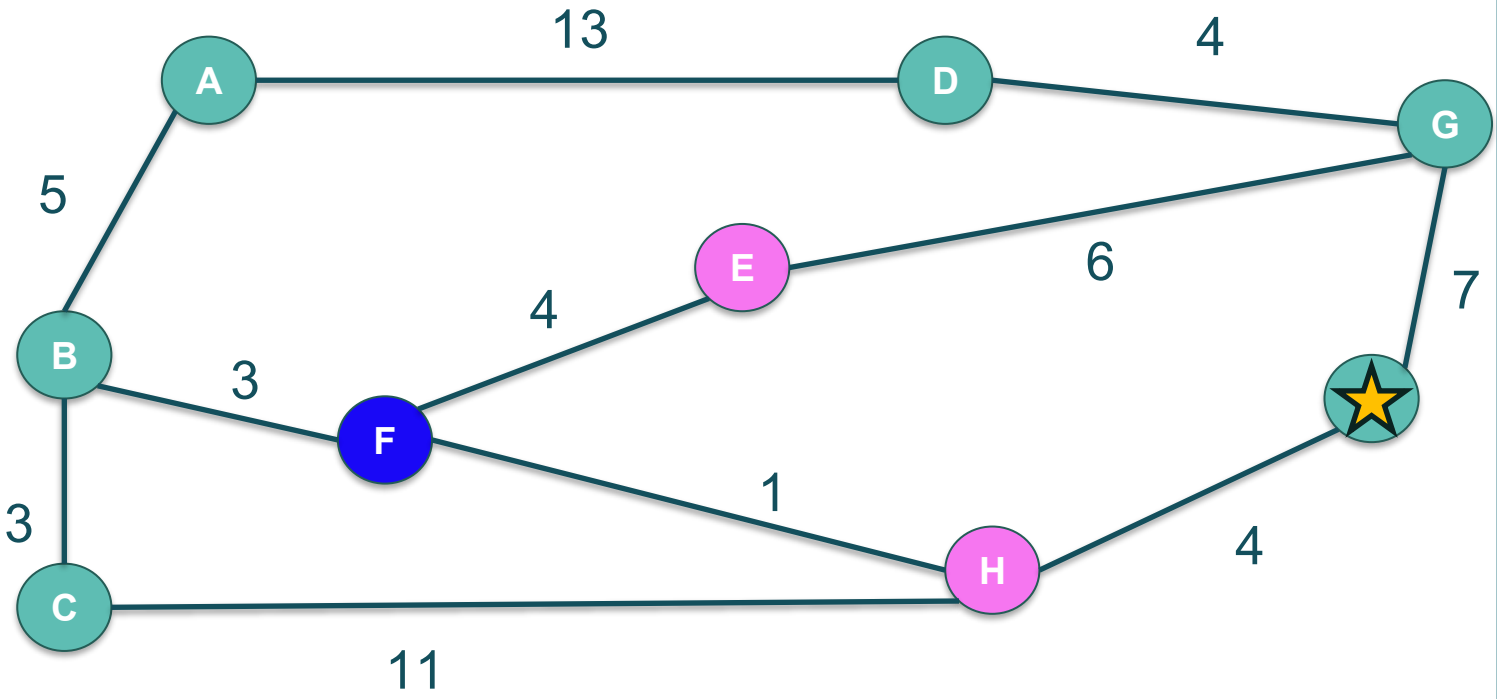
D | 13

H | 19

… | …

★ | ∞

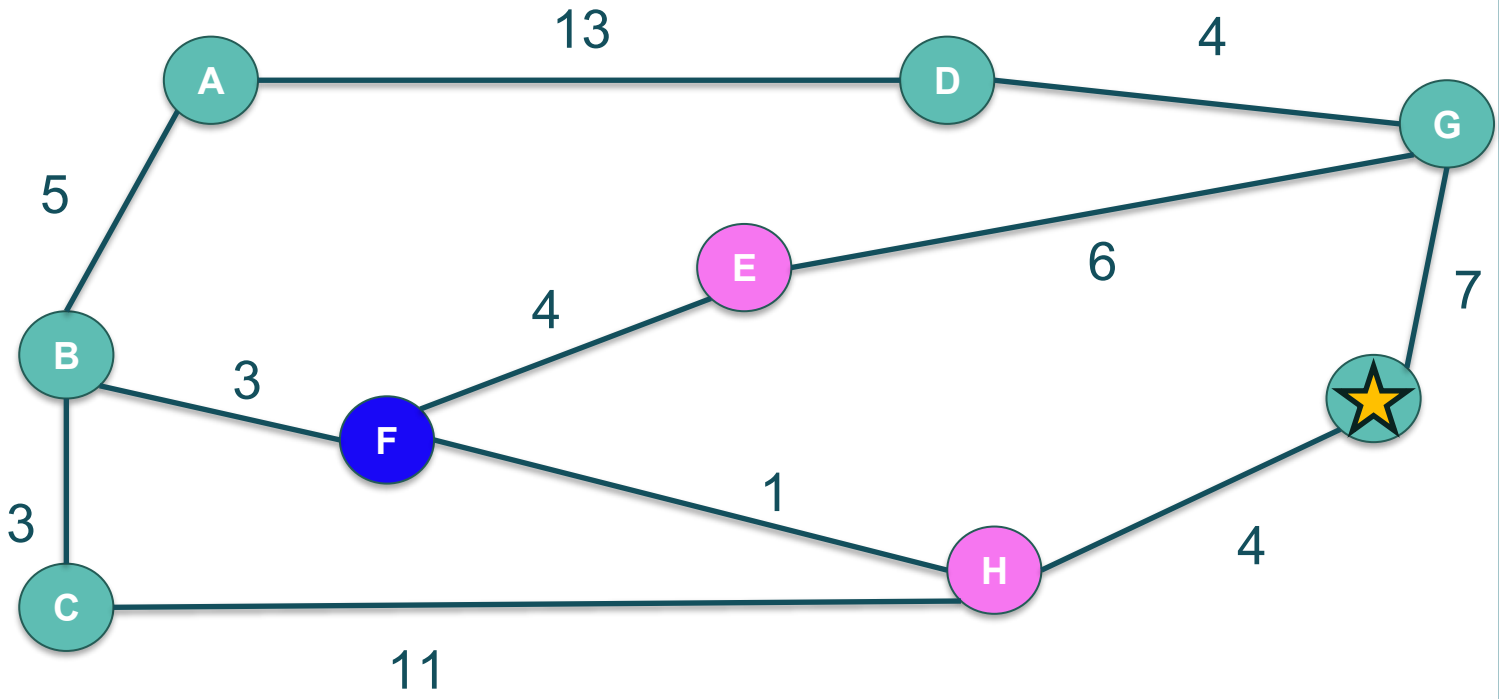
F | 8



Queue:

D	13
H	19
E	∞
…	…
★	∞

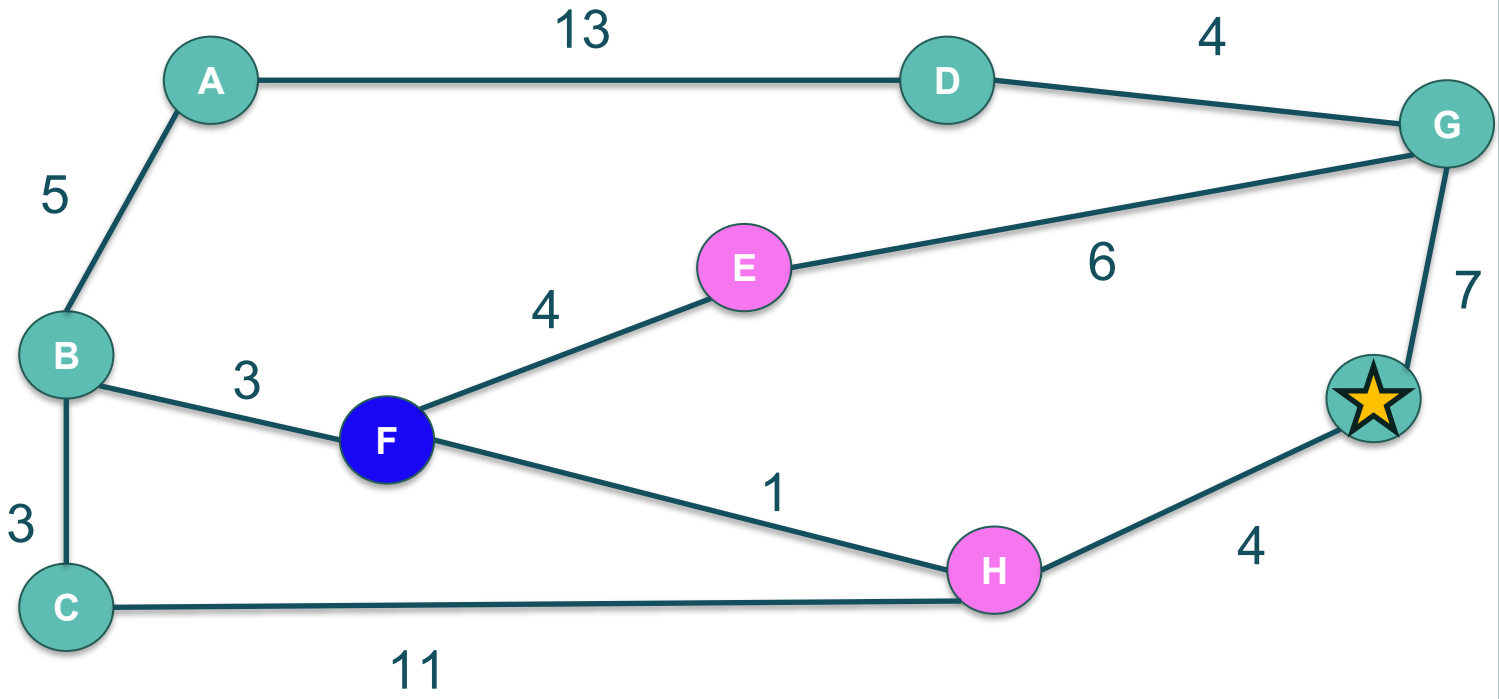
F | 8



Queue:

D	13
H	19
E	12
…	…
★	∞

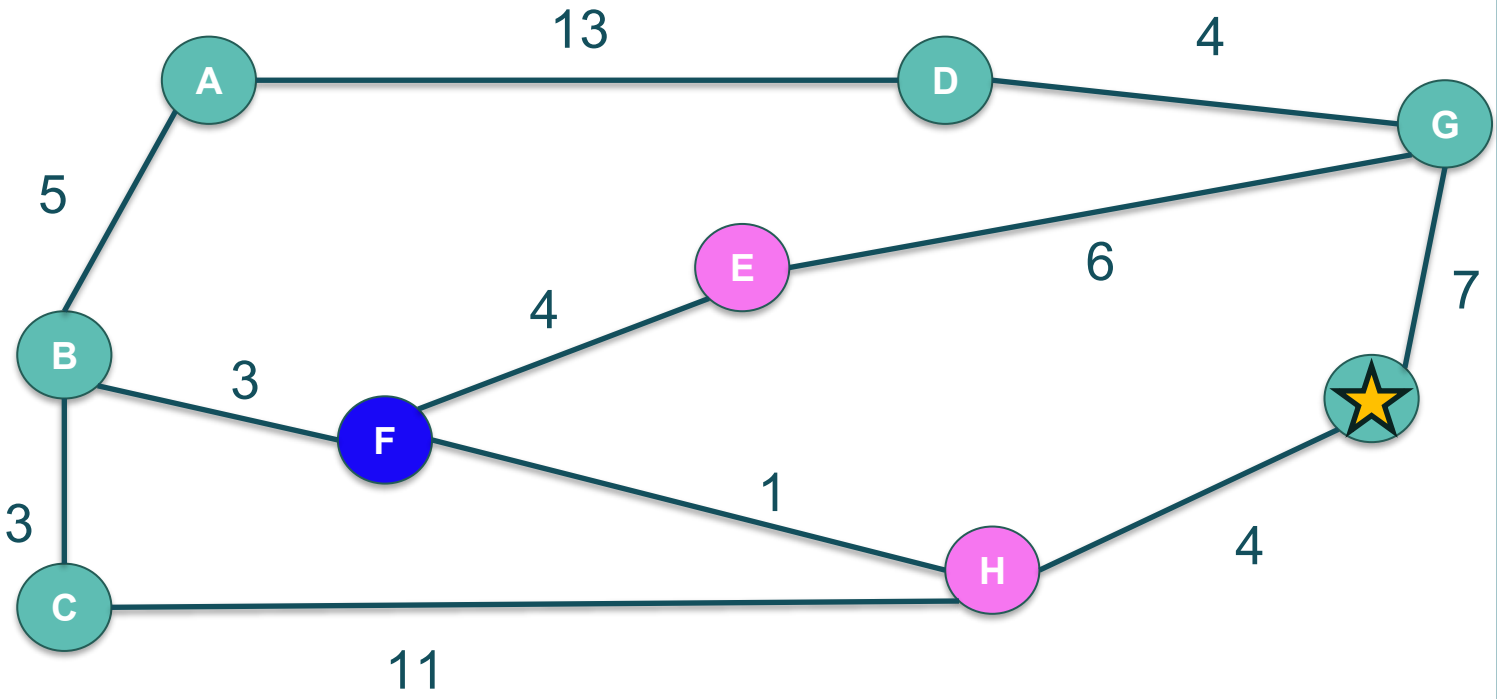
F | 8



Queue:

D	13
H	?
E	12
…	…
★	∞

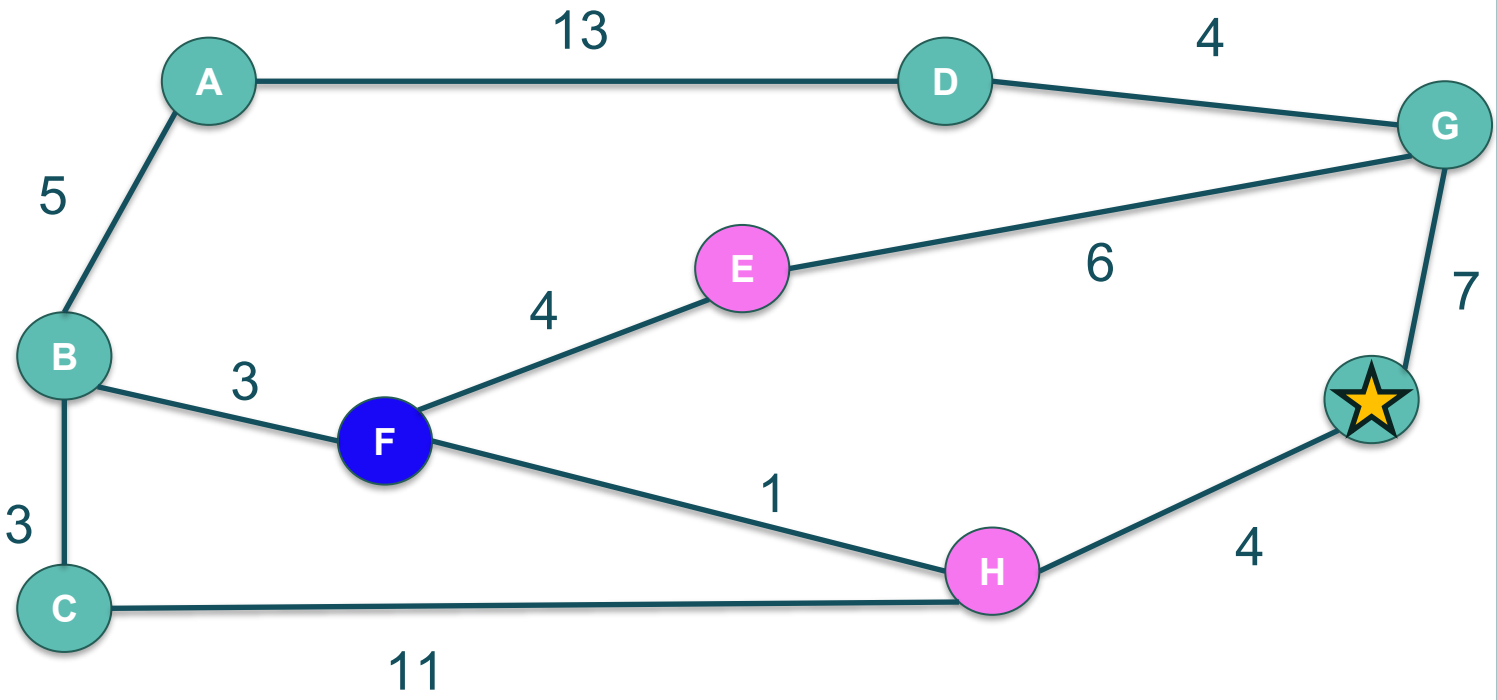
F | 8 → H | ? = 9



Queue:

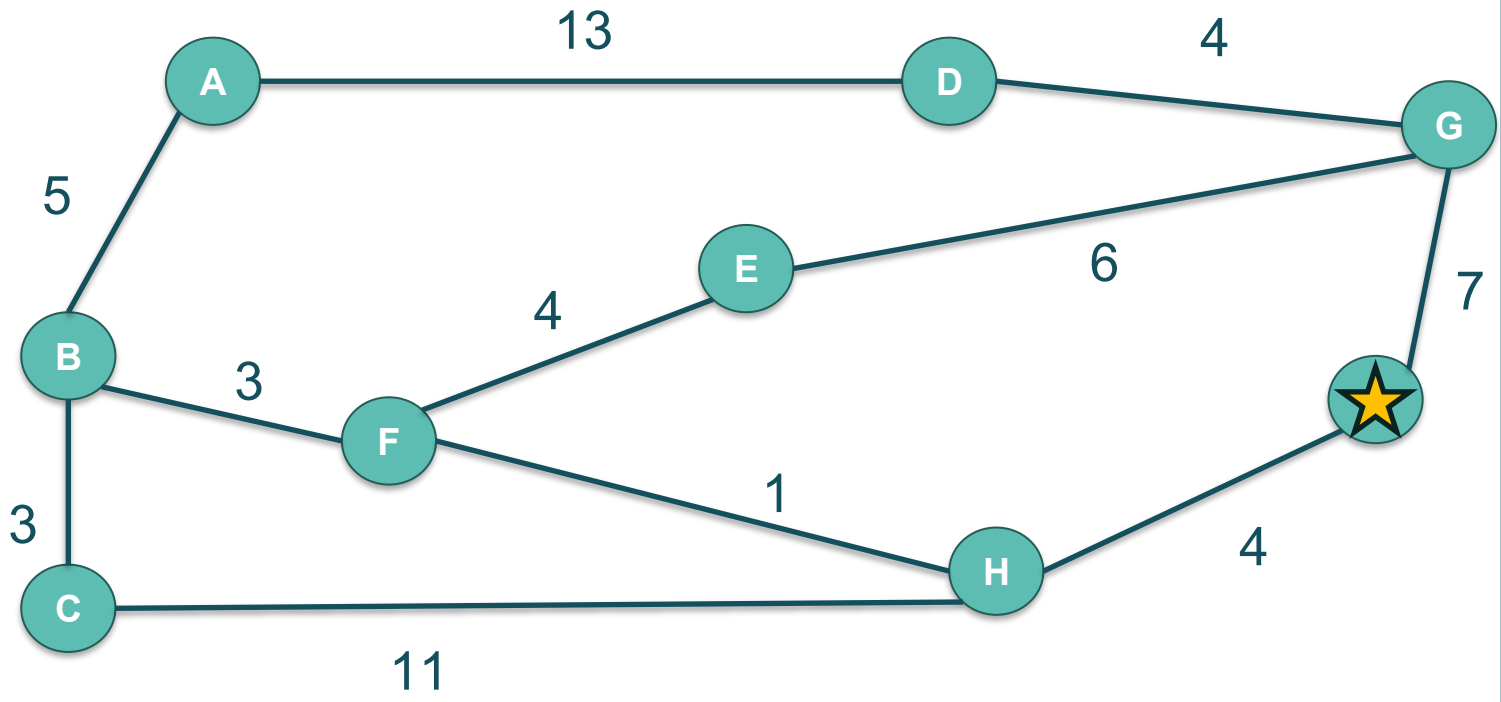
D	13
H	9
E	12
…	…
★	∞

F | 8 → H | 9



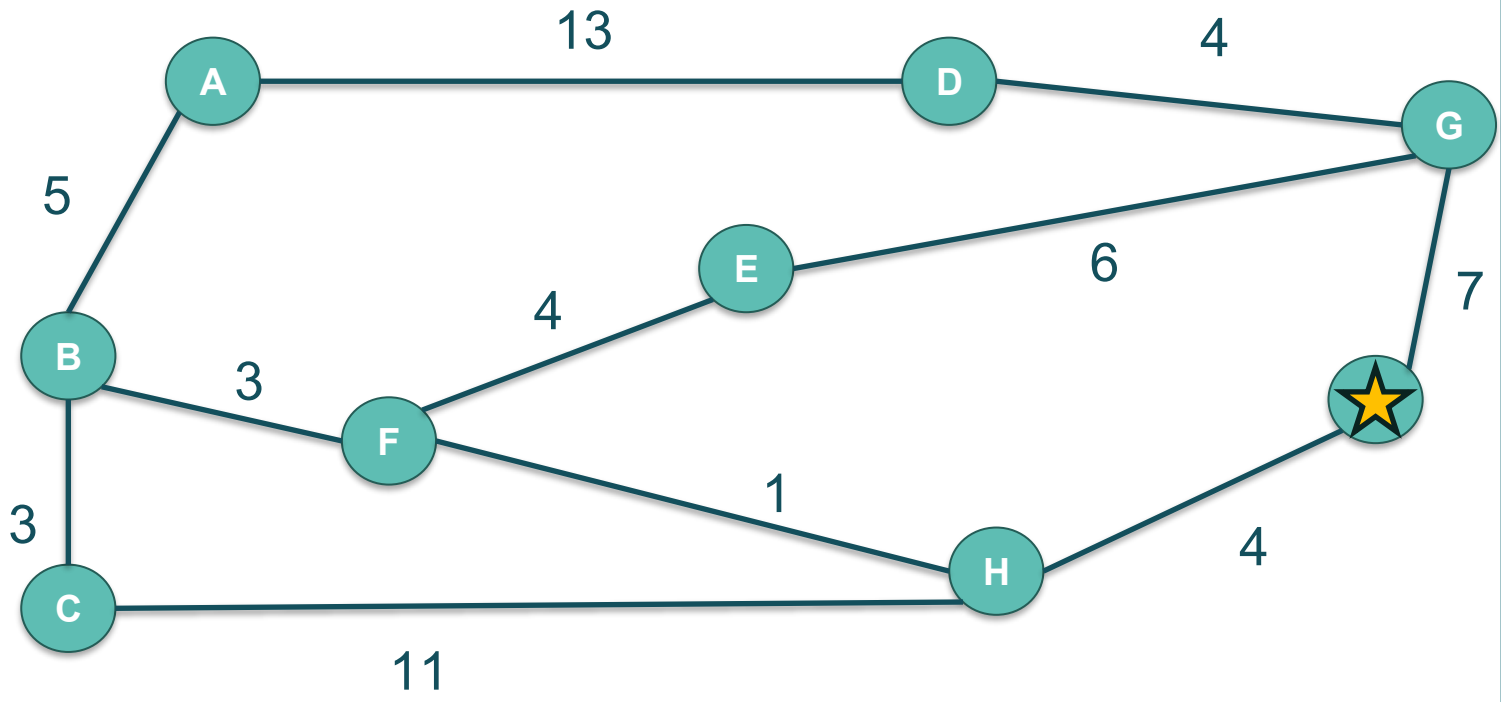
Queue:

D	13
H	9
E	12
…	…
★	∞



Queue:

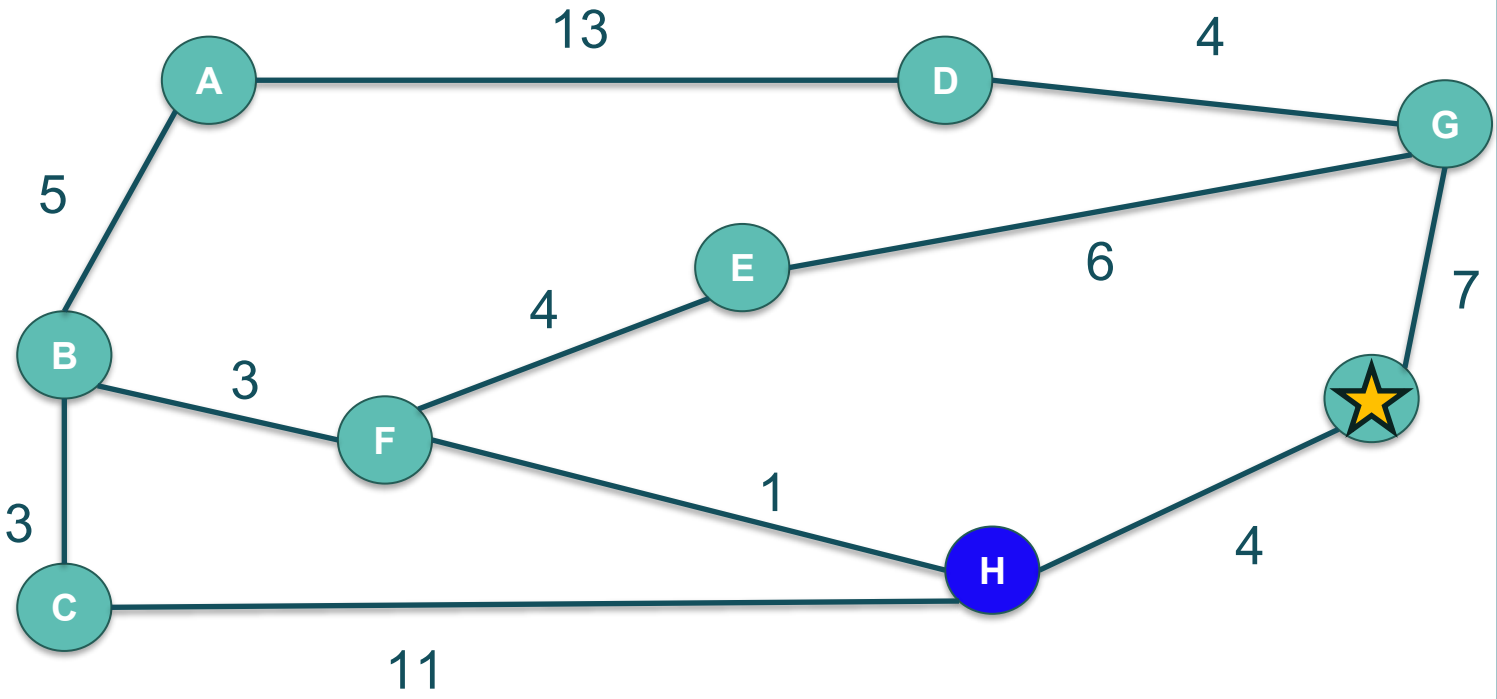
H	9
E	12
D	13
…	…
★	∞



Queue:

E	12
D	13
…	…
★	∞

H | 9



Queue:

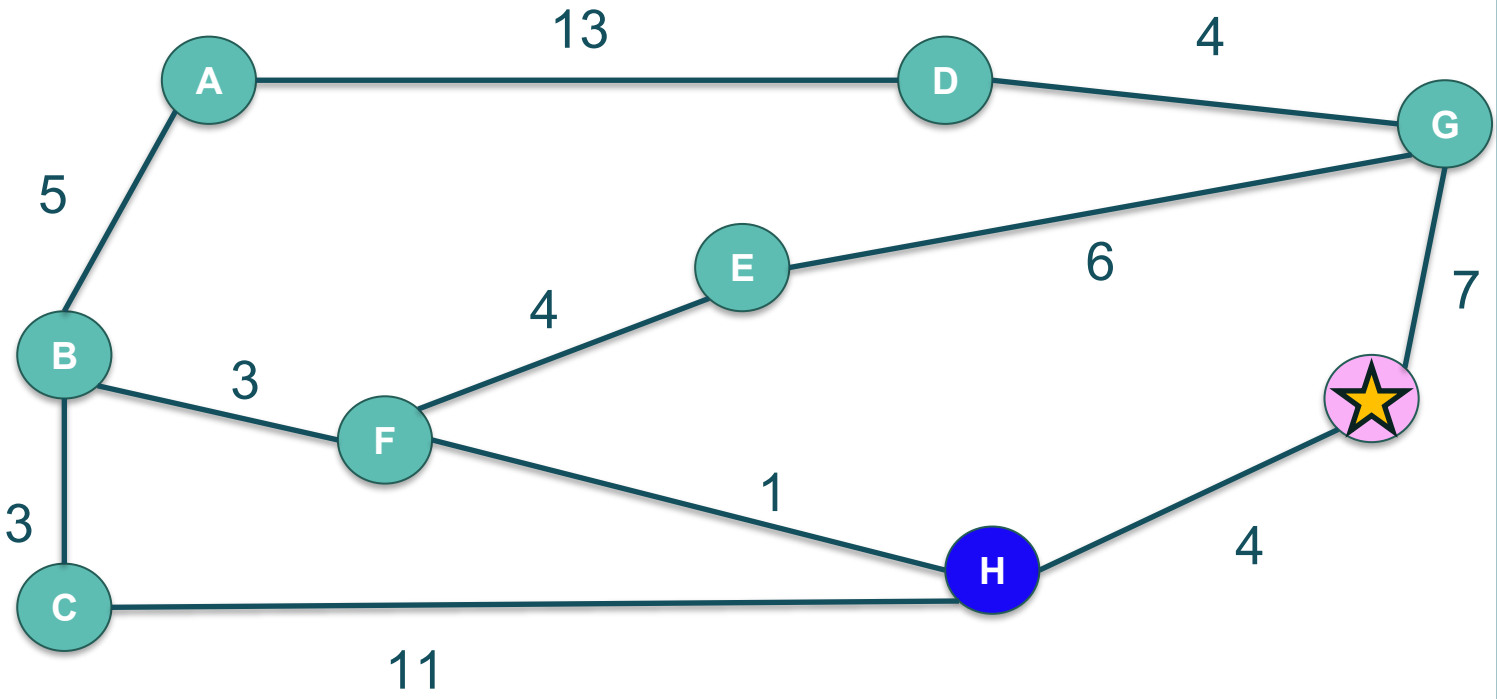
E	12

D	13

…	…

★	∞

H	9



Queue:

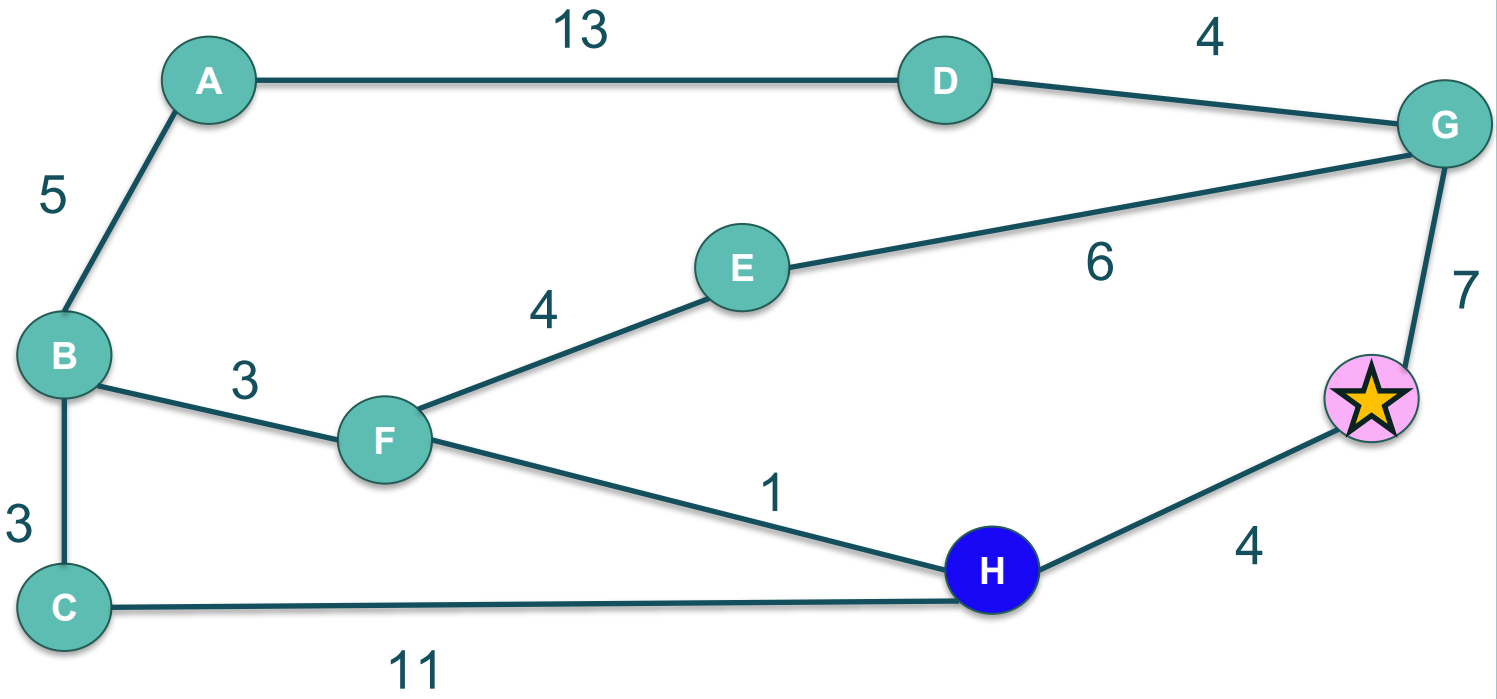
E | 12

D | 13

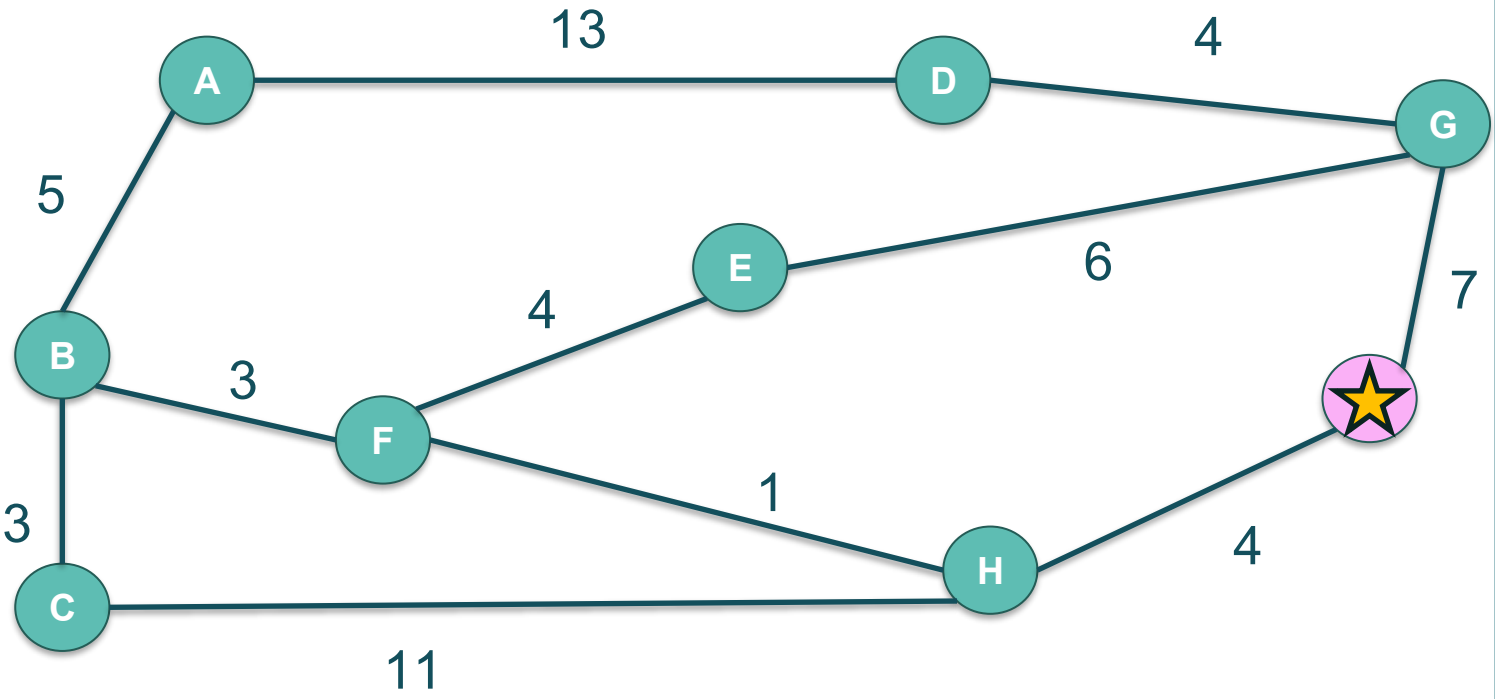
… | …

★ | 13

H | 9




A
B
F
H


A
D
G
B
E
F
C
H
13
4
5
6
7
4
3
3
1
4
11

MEN!

- Algoritmen stopper ikke før den har funnet den korteste stien fra s til ALLE noder

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (TRADISJONELL)

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnode s
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

Dijkstra - Kompleksitet

- RemoveMin: Logaritmisk O(log(n))
- DecreasePriority: Logaritmisk O(log(n))

- Det er |V| på prioritetskøen
- Så vi gjør RemoveMin |V| ganger
- O(|V| * log(|V|))

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (T

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnc
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

Dijkstra - Kompleksitet

- Det er |V| på prioritetskøen
- Så vi gjør RemoveMin |V| ganger
- O(|V| * log(|V|))

- Vi besøker alle kanter (E) èn gang
- Så vi gjør DecreasePriority |E| ganger
- O(|E| * log(|V|))

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (T

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnc
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

Dijkstra - Kompleksitet

- Det er |V| på prioritetskøen
- Så vi gjør RemoveMin |V| ganger
- **O(|V| * log(|V|))**

- Vi besøker alle kanter (E) èn gang
- Så vi gjør DecreasePriority |E| ganger
- **O(|E| * log(|V|))**

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (T

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnc
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

Dijkstra - Kompleksitet

O(|V| * log(|V|))

+

O(|E| * log(|V|))

=

O(|V|*log(|V|) + |E|*log(|V|))

=

O((|V| + |E) * log(|V|))

ALGORITHM: DIJKSTRAS ALGORITME FOR KORTESTE STIER (T

Input: En vektet graf $G = (V, E)$ med vektfunksjon w og en startnc
Output: Et map `dist` som angir korteste vei fra s til alle noder i G

```
1  Procedure Dijkstra(G, s)
2      queue ← empty priority queue
3      dist ← empty map
4      for v ∈ V do
5          dist[v] ← ∞
6          Insert(queue, v) with priority ∞
7      dist[s] ← 0
8      DecreasePriority(queue, s, 0)
9
10     while queue is not empty do
11         u ← RemoveMin(queue)
12         for (u, v) ∈ E do
13             c ← dist[u] + w(u, v)
14             if c < dist[v] then
15                 dist[v] ← c
16                 DecreasePriority(queue, v, c)
17     return dist
```

Minimale spenntrær

Minimale spenntrær

Graf -> Tre

Spenntrær

- Trær er grafer
- Men det finnes grafer som ikke er trær

- Et spenntre av en graf G skal bestå av akkurat de samme nodene som G, men kun et utvalg (En submengde) av kantene

Minimale spenntrær

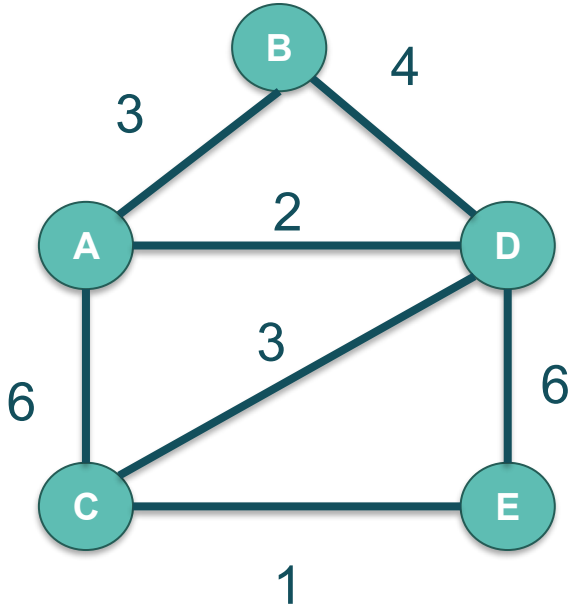
- Èn graf kan ha flere mulige spenntrær
- Et minimalt spenntre er det spenntreet til graf G som har minst vekt til sammen

- Kan lages med Prims algoritme, Kruskals algoritme og Boruvkas.

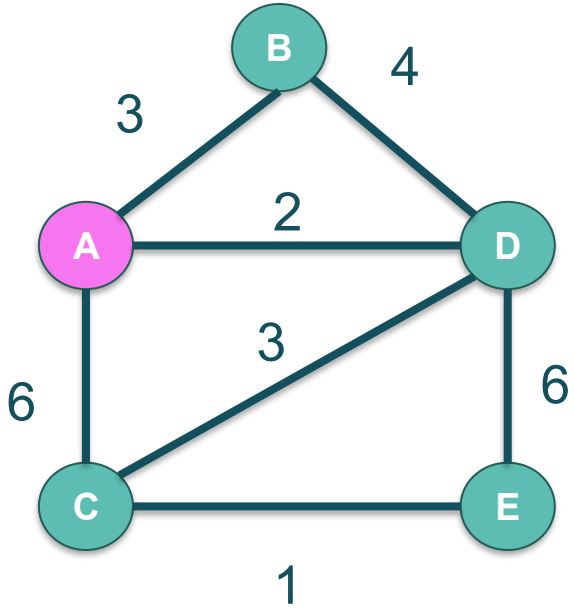
Prim

1) Velg en vilkårlig node (Spiller ingen rolle hvilken), U
2) Hva er den ‘korteste’/minste kanten til U som går til èn node vi ikke har besøkt før? Denne kanten blir en del av spenntreet!
3) Så går vi til den ubesøkte noden (via kanten vi akkurat valgte) og starter på trinn 2 igjen

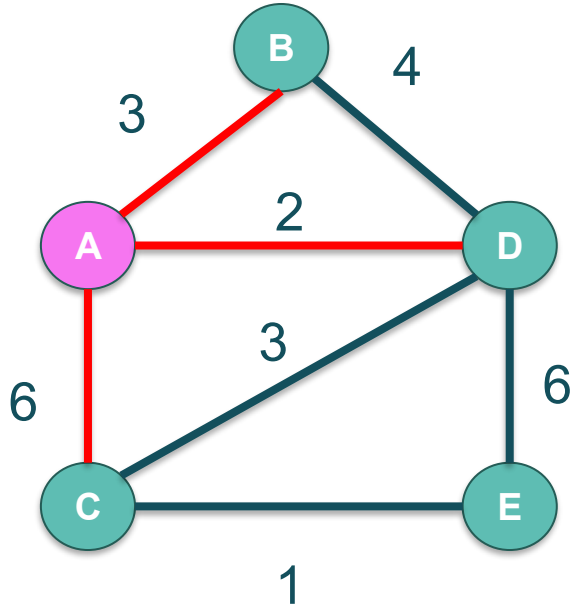
Prim



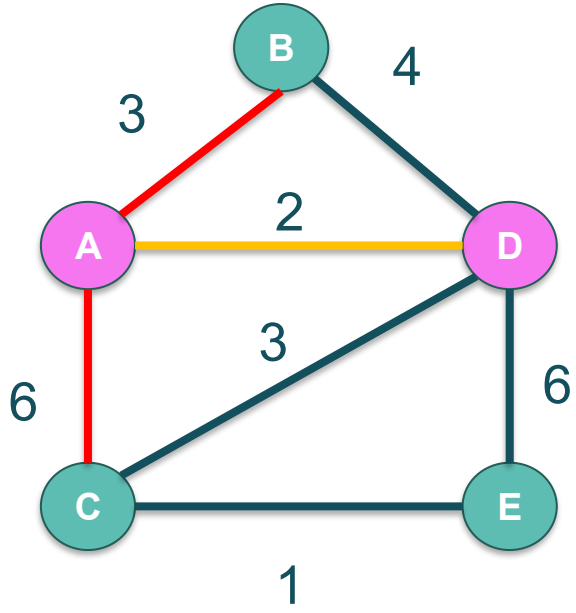
Prim



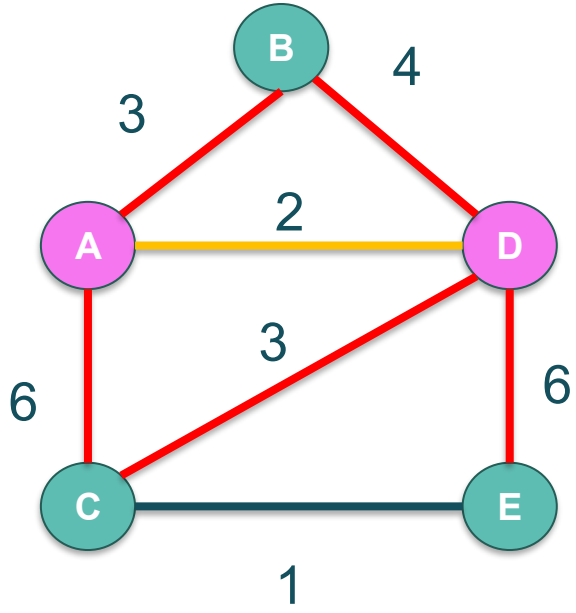
Prim



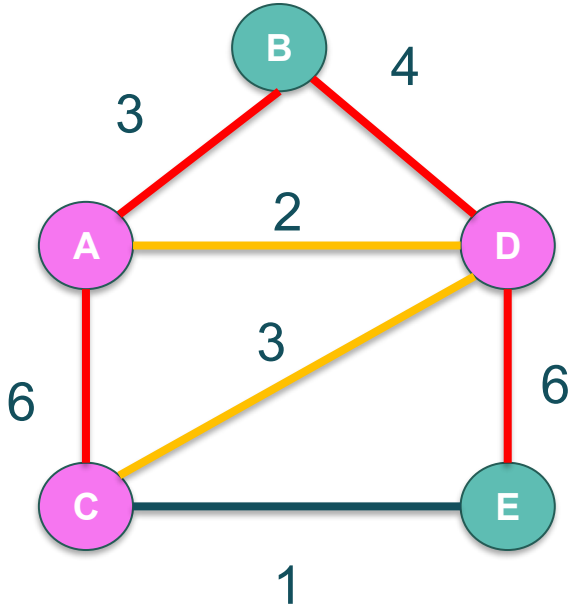
Prim



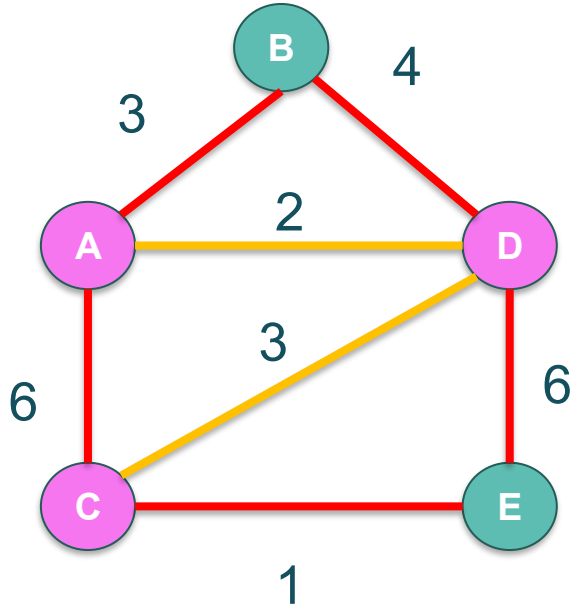
Prim



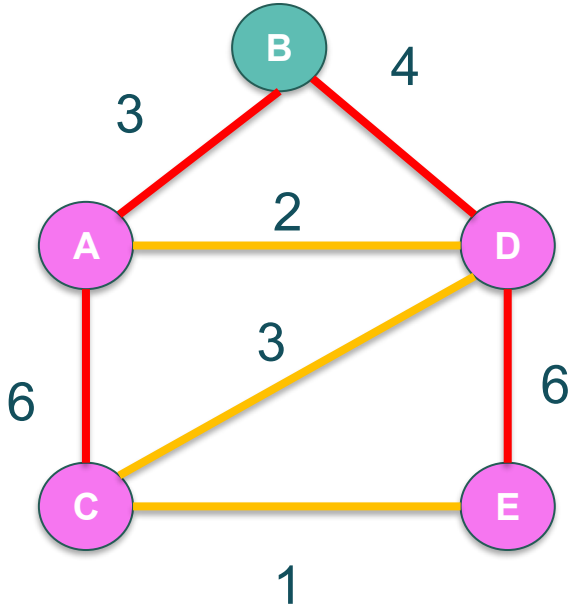
Prim



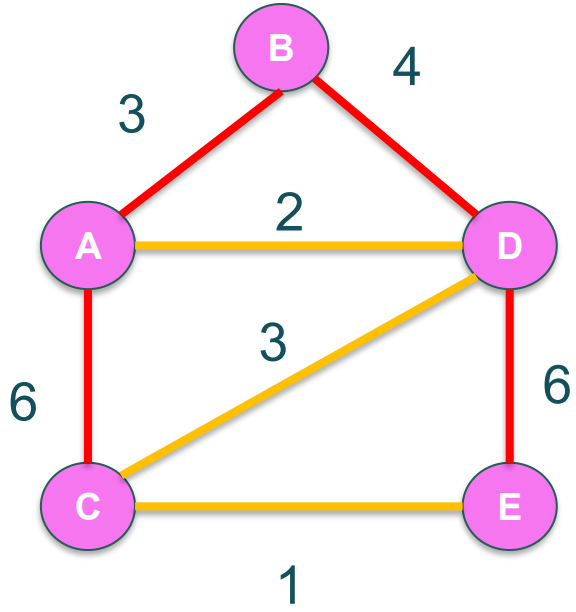
Prim



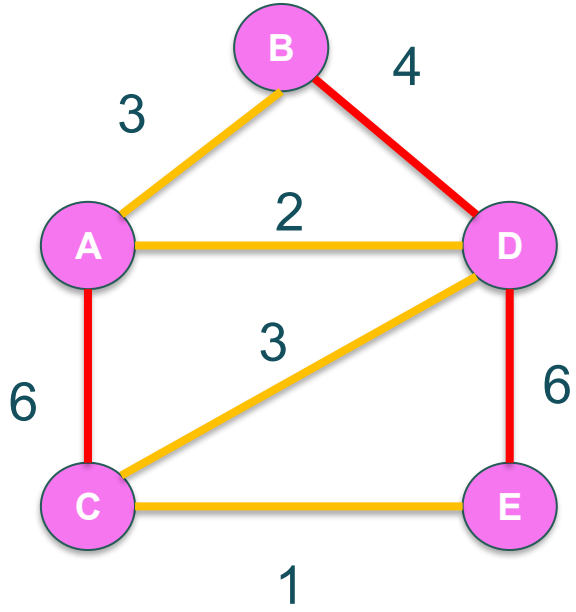
Prim



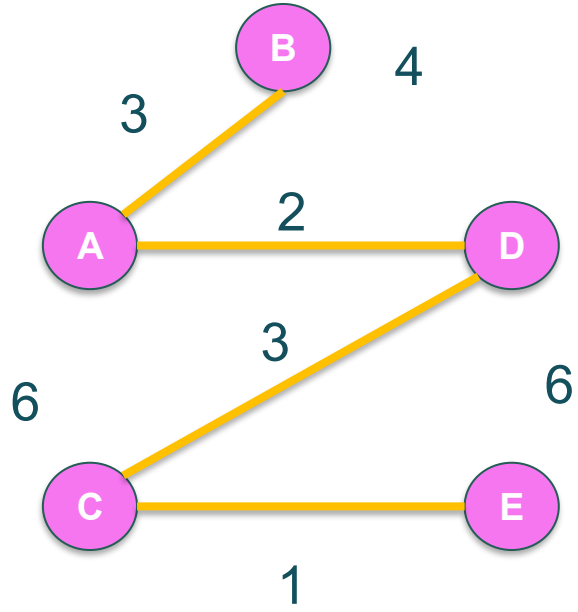
Prim



Prim



Prim



Oppgaver