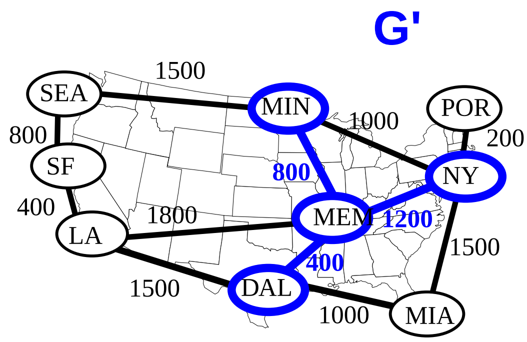
MINIMUM UTSPENNENDE TRE

- Engelsk: Minimum Spanning Tree (MST)
- Prim's algoritme for MST
- Sammenlikne Prim's MST og Dijkstra's SSSP
- Endring i pensum:
  - 10.2.1 Kruskal's MST algoritme går ut
  - 10.2.2 Prim's MST algoritme går inn

Vektede Grafer

(vekt av en delgraf **G'**) =
(sum av alle kantvekter i **G'**)

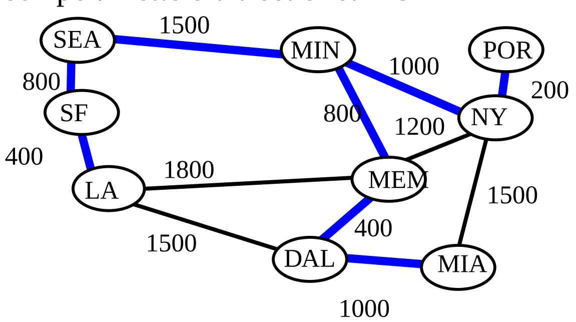
$$\textbf{vekt}(G') = \sum_{(e \in G')} \textbf{vekt}(e)$$



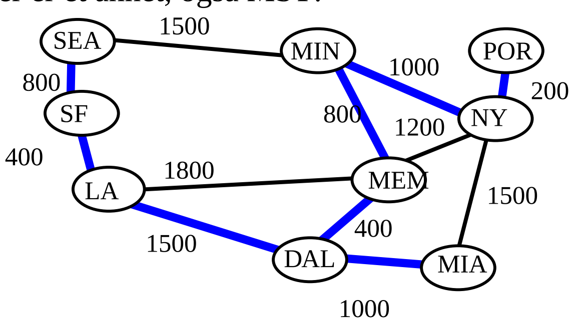
vekt(G') = 800 + 400 + 1200
= 2400

Minimum utSpennende Tre

- Finn et utspennende tre med minste vekt
- f.eks. koble sammen alle datamaskiner i en bygning med minst kabelmetere
- eksempel: Dette blå treet er et MST



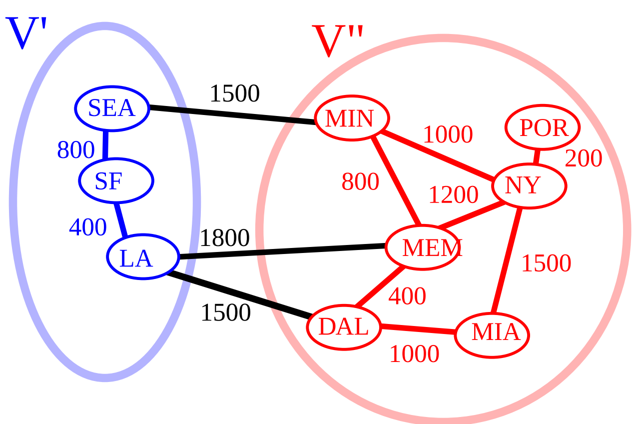
- Her er et annet, også MST!



Hvilke kanter kan være med i et MST?

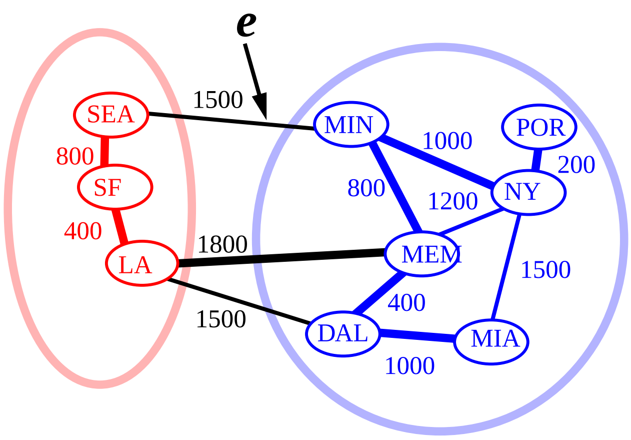
La (V',V") være partisjon av noder i G.

La ***e*** = (v', v") være en kant med minste vekt av alle de som krysser partisjonen, dvs v' ∈ V' and v" ∈ V".
Da finnes et MST med denne kanten.



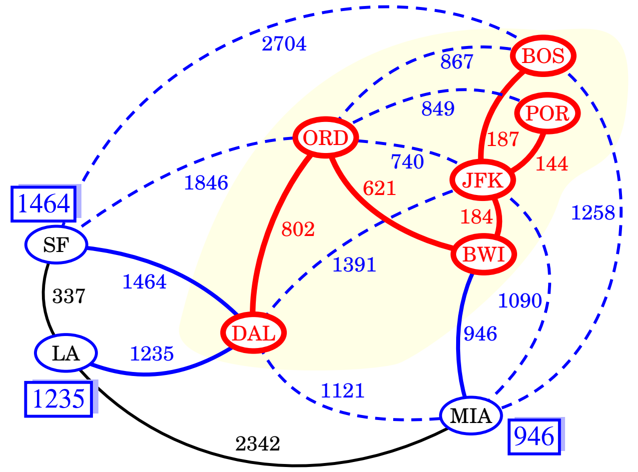
Bevis for dette:

Anta G har et MST T som IKKE inneholder ***e***. Da konstruerer vi et annet MST T’, ikke høyere vekt, som inneholder ***e***. Hvordan?



Prim’s Algoritme for å finne et MST

- Vi bygger opp et MST T som vokser med en ny node av gangen.
- Opprettholder ***en sky*** av noder som dekker delen av T beregnet hittil
- Tabeller D[u] og E[u] indeksert ved noder u, hvor
  - E[u] er kant med lavest vekt som knytter u til T
  - D[u] (avstand til skyen) er vekten til E[u]



Sammenlikning av Prim’s MST og Dijkstra’s SSSP

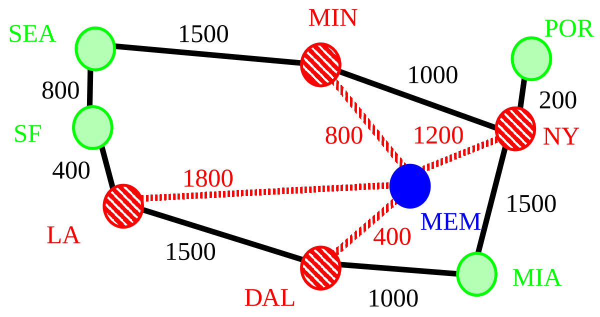
- Prim’s beregner **MST**, Dijkstra’s SSSP
- For enhver node u, er **D[u] vekten på den minste enkelt-kant som knytter u til skyen** (Dijkstra: ‘vektsum for den minste sti fra startnoden som knytter u til skyen’)
- Som Dijkstra brukes en prioritetskø Q hvor nøkler er vekter, men i Prim’s er **elementene i Q node-kant par**.
- **Enhver** node v kan være **startnode**.
- Vi initialiserer D[u] verdier til INFINITE, men også **initialiserer vi E[u] (kant for u) til null-verdi.**

Vi kan gjenbruke endel av koden fra Dijkstra’s algoritme.

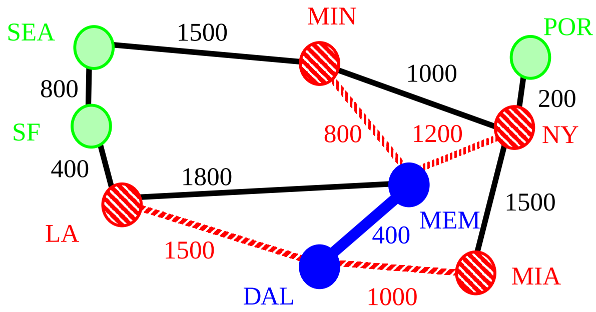
Pseudokode

Algorithm Prim(G):
  Input: A weighted graph G.
  Output: A minimum spanning tree T for G.
pick any vertex v of G
{grow the tree starting with vertex v}
T ← {v}
  $D[u] \leftarrow 0$
  $E[u] \leftarrow \emptyset$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
let Q be a priority queue that contains
    vertices, using the D labels as keys
while $Q \neq \emptyset$ **do**
    {pull u into the cloud C}
    $u \leftarrow$ Q.removeMinElement()
    add vertex u and edge $E[u]$ to T
    for each vertex z adjacent to u **do**
      if z is in Q
        {perform the ***reLAation operation*** on edge (u, z) }
        if weight$(u, z) < D[z]$ **then**
          $D[z] \leftarrow$ weight(u, z)
          $E[z] \leftarrow (u, z)$
          change the key of z in Q to $D[z]$
  return tree T

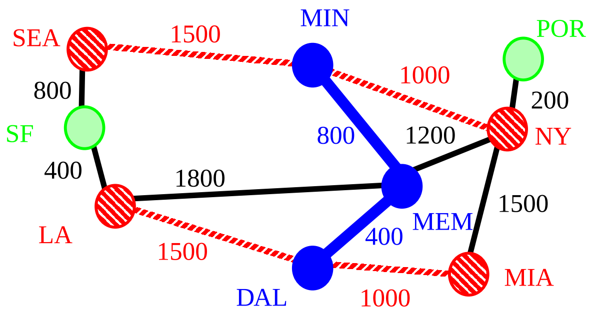
Et eksempel



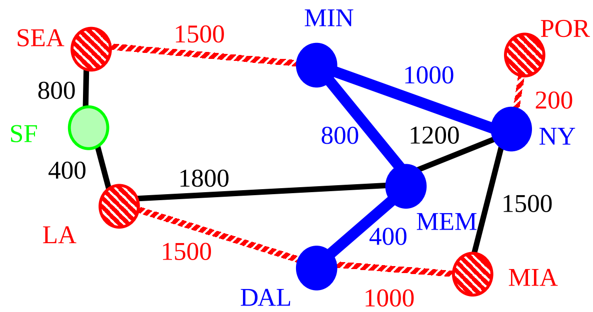
	neighbor	D[u]
DAL	MEM	400
LA	MEM	1800
NY	MEM	1200
MIA		
MIN	MEM	800
POR		
SEA		
SF		
MEM		



	neighbor	D[u]
DAL		
LA	DAL	1500
NY	MEM	1200
MIA	DAL	1000
MIN	MEM	800
POR		
SEA		
SF		
MEM		



	neighbor	D[u]
DAL		
LA	DAL	1500
NY	MIN	1000
MIA	DAL	1000
MIN		
POR		
SEA	MIN	1500
SF		
MEM		



	neighbor	D[u]
DAL		
LA	DAL	1500
NY		
MIA	DAL	1000
MIN		
POR	NY	200
SEA	MIN	1500
SF		
MEM		

Kjøretid

$T \leftarrow \{v\}$
  $D[u] \leftarrow 0$
  $E[u] \leftarrow \emptyset$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
let Q be a priority queue that contains all the
    vertices using the D labels as keys
while $Q \neq \emptyset$ **do**
    $u \leftarrow Q$.removeMinElement()
    add vertex u and edge $E[u]$ to T
    for each vertex z adjacent to u **do**
      if z is in Q
        if weight$(u, z) < D[z]$ **then**
          $D[z] \leftarrow$ weight(u, z)
          $E[z] \leftarrow (u, z)$
          Q.UpdateKey(z,D[z])
return tree T

$O((n+m) \log n)$

hvor n = antall noder, m=antall kanter,

og Q implementert som heap.