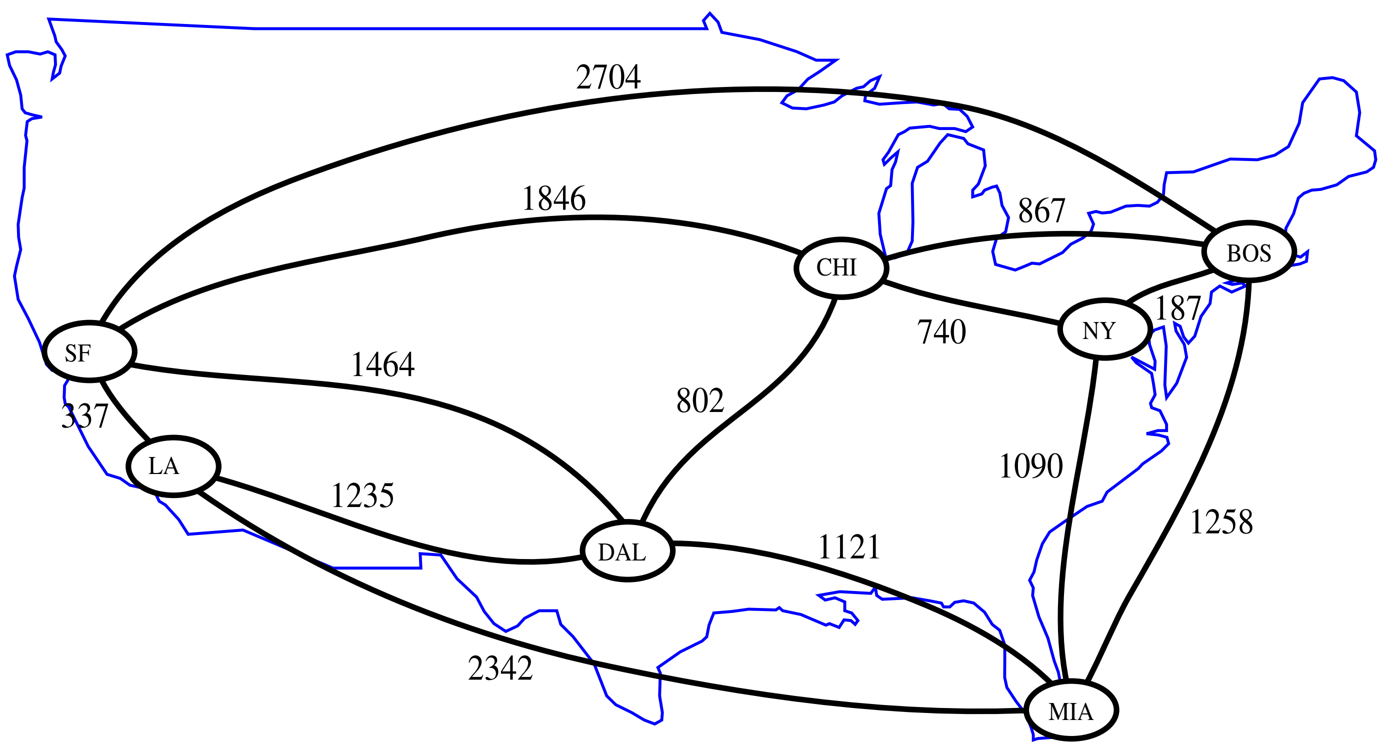
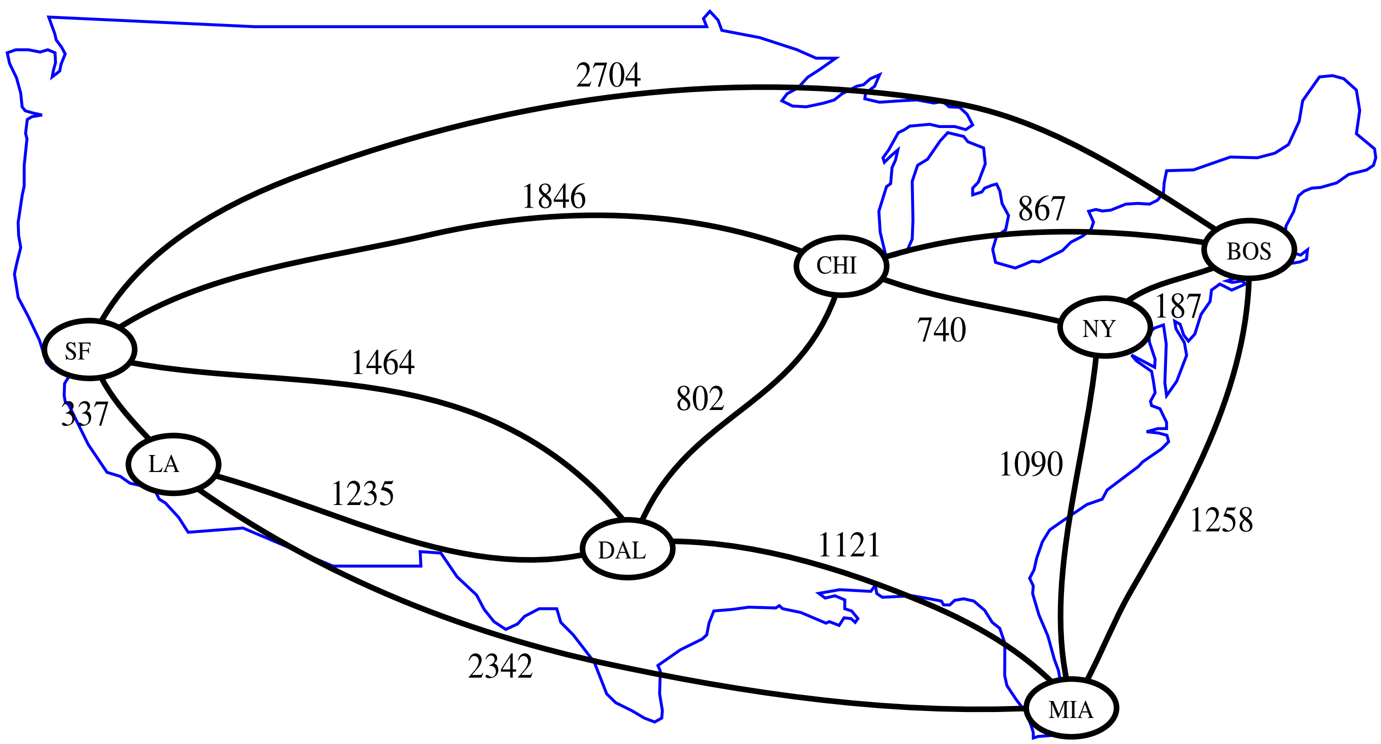
KORTESTE STI

- Vektet Urettet Graf
- Finn: Korteste Sti fra en Enkel Kilde til Alle Noder. (Engelsk: Single Source Shortest Path - SSSP)
- Dijkstra's SSSP Algoritme
- PQ med UpdateKey-operasjon



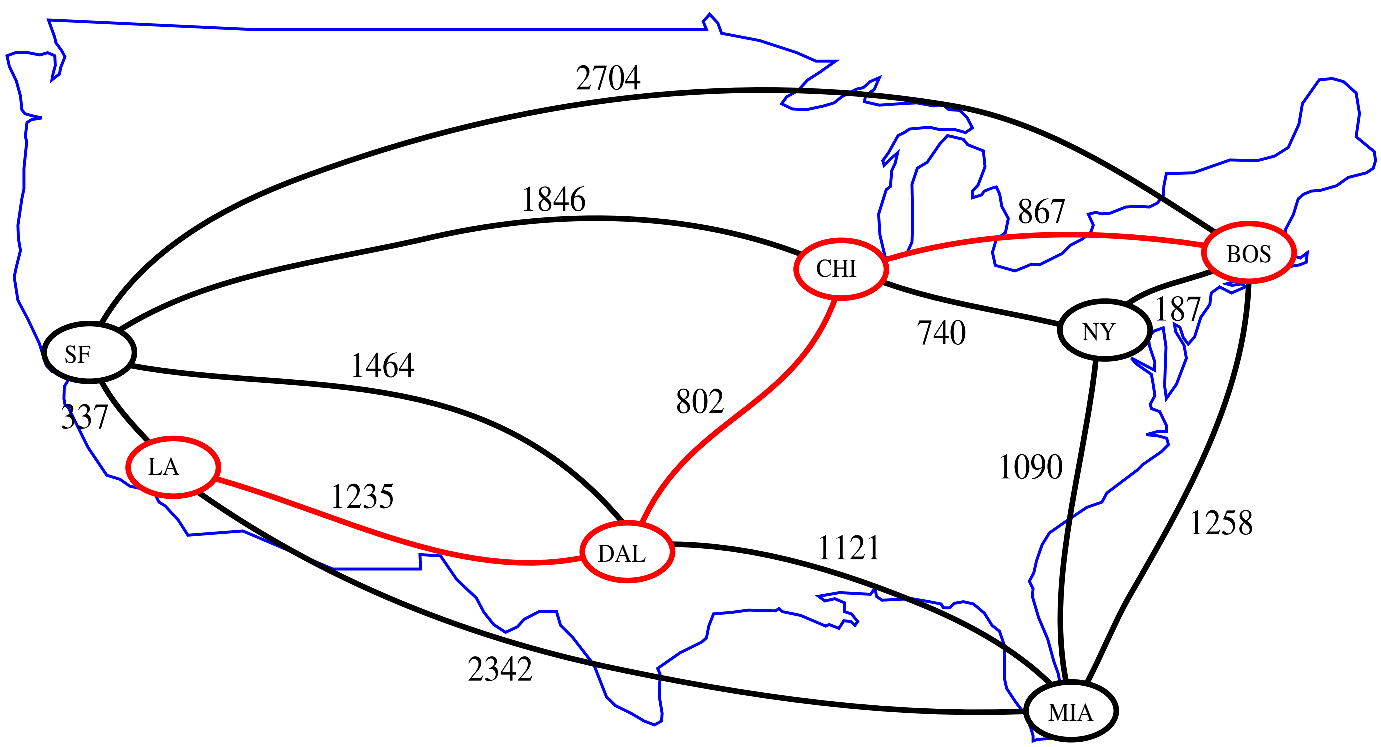
Vektede Grafer

- **vekter** på kanter representerer f.eks. avstand, kostnad, båndbredde...
- Eks: Urettet vektet graf:



Korteste Sti

- Bredde Først Søk finner stier med minst antall kanter fra en start-node.
- Dvs, BFS løser SSSP-problemet dersom alle kanter har samme vekt.
- I mange applikasjoner, f.eks. transport-nettverk, har kantene forskjellig vekt.
- Hvordan skal vi da finne stier med lavest sum av vekt på kanter?
- Eksempel - Boston til Los Angeles:



Dijkstra's Algoritme

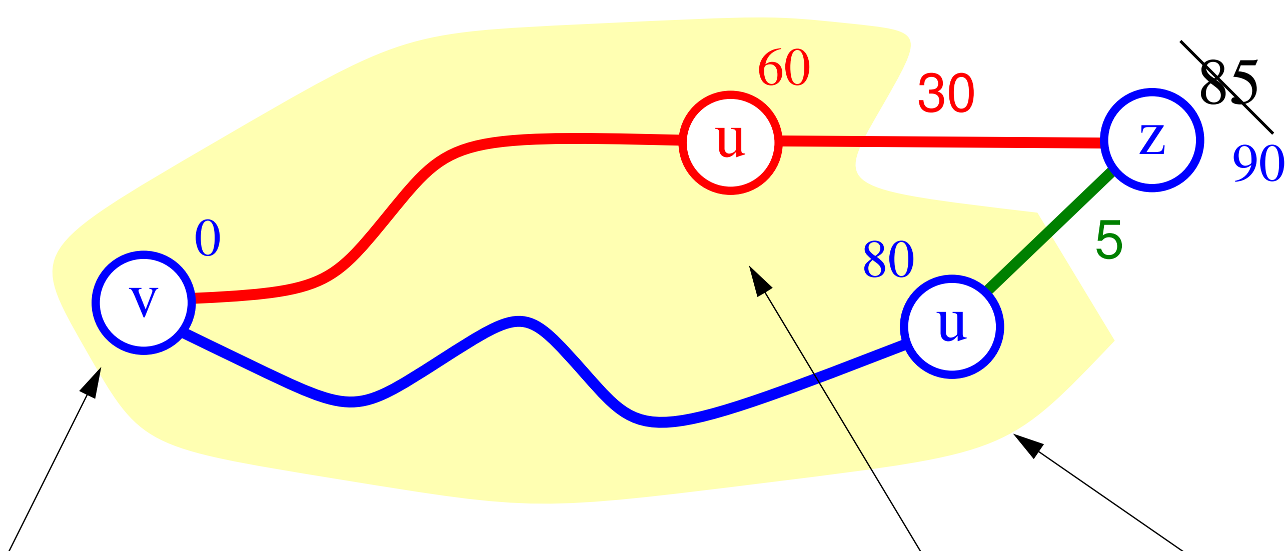
- Dijkstra's algoritme finner korteste sti fra startnode *v* til alle andre noder i en graf med
  - positive kantvekter
- For hver node *u* beregnes **avstand** fra startnode *v* til noden *u*, dvs laveste vektsum over alle stier fra *v* til *u*.
- Algoritmen oppdaterer en sky av de noder for hvilke korrekt avstand allerede er beregnet.
- En foreløpig approksimert avstand fra startnoden lagres i en tabell D, dvs D[*u*] vil inneholde vektsum på korteste sti til node *u* som har blitt funnet hittil.
- Når node *u* legges til skyen, er D[*u*] den faktiske avstand fra startnoden til *u*, dvs vi vet da at ingen kortere sti kan finnes.
- Ved initiering:
  - D[v] = 0 ...avstand fra v til seg selv er 0...
  - D[u] = ∞ for u $\neq$ v ...disse vil endres...

Dijsktra's Algoritm: Skyen vokser seg større

- Repeter inntil alle noder ligger i skyen:
  - la u være noden ikke i skyen som har lavest D[u]-verdi (i første runde vil dette være startnoden v)
  - legg u til skyen C
  - så oppdaterer vi approksimerte avstander til naboer av u som følger:

for each vertex z adjacent to u **do**
  if z is not in the cloud C **then**
    if D[u] + weight(u,z) < D[z] **then**
      D[z] = D[u] + weight(u,z)

- steget over kalles **relaksering** av kant (u,z)



v ble lagt i skyen først . Så kom denne u . Så denne u.

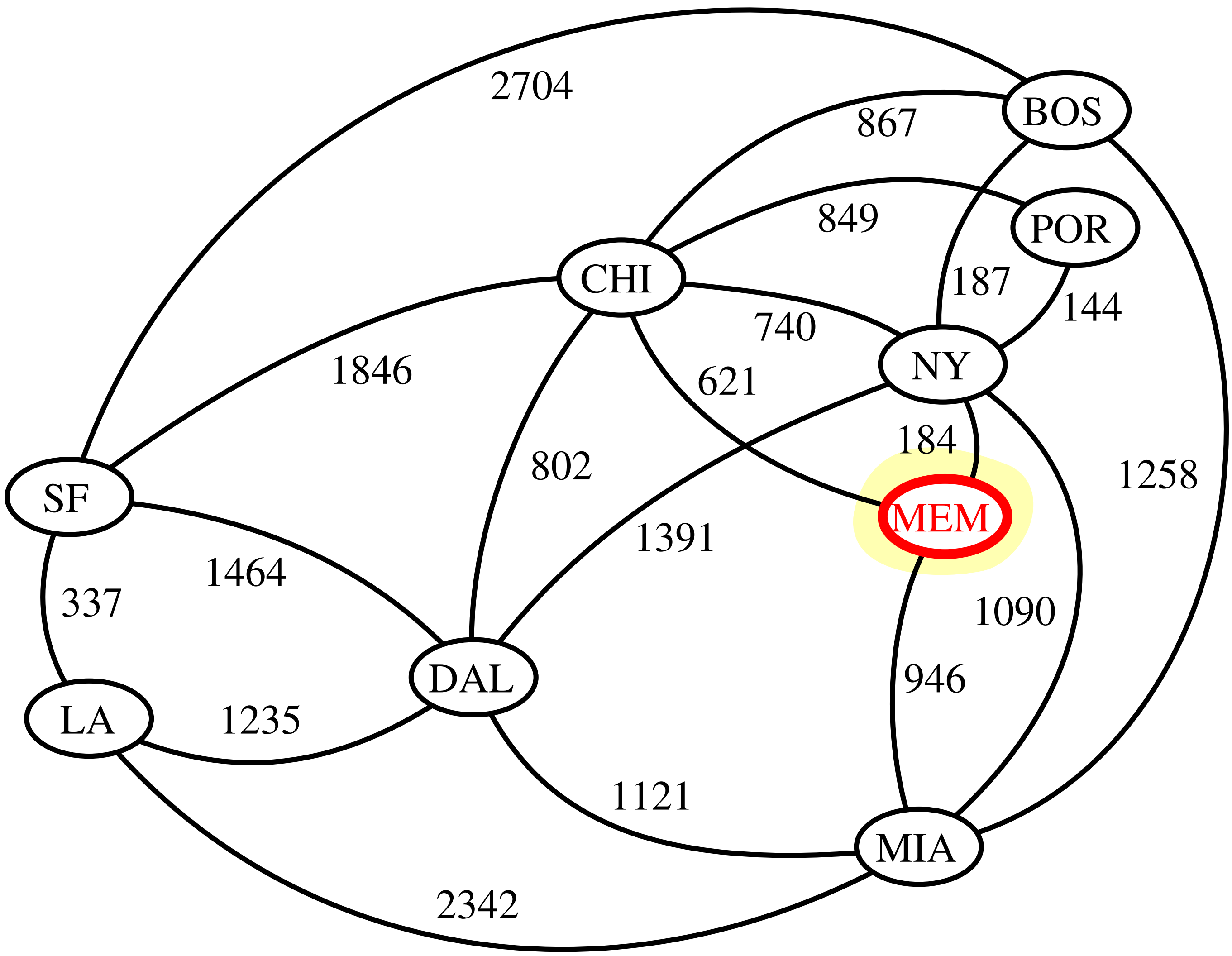
Pseudokode

- Bruker prioritetskø Q for å lagre nodene som ikke er i skyen enda, $D[v]$ er nøkkel for node v i Q

Algorithm ShortestPath(G, v):
  Input: A weighted graph G and a distinguished vertex v of G.
  Output: $D[u]$, for each vertex u of G, such that $D[u]$ is the length of a shortest path from v to u in G.

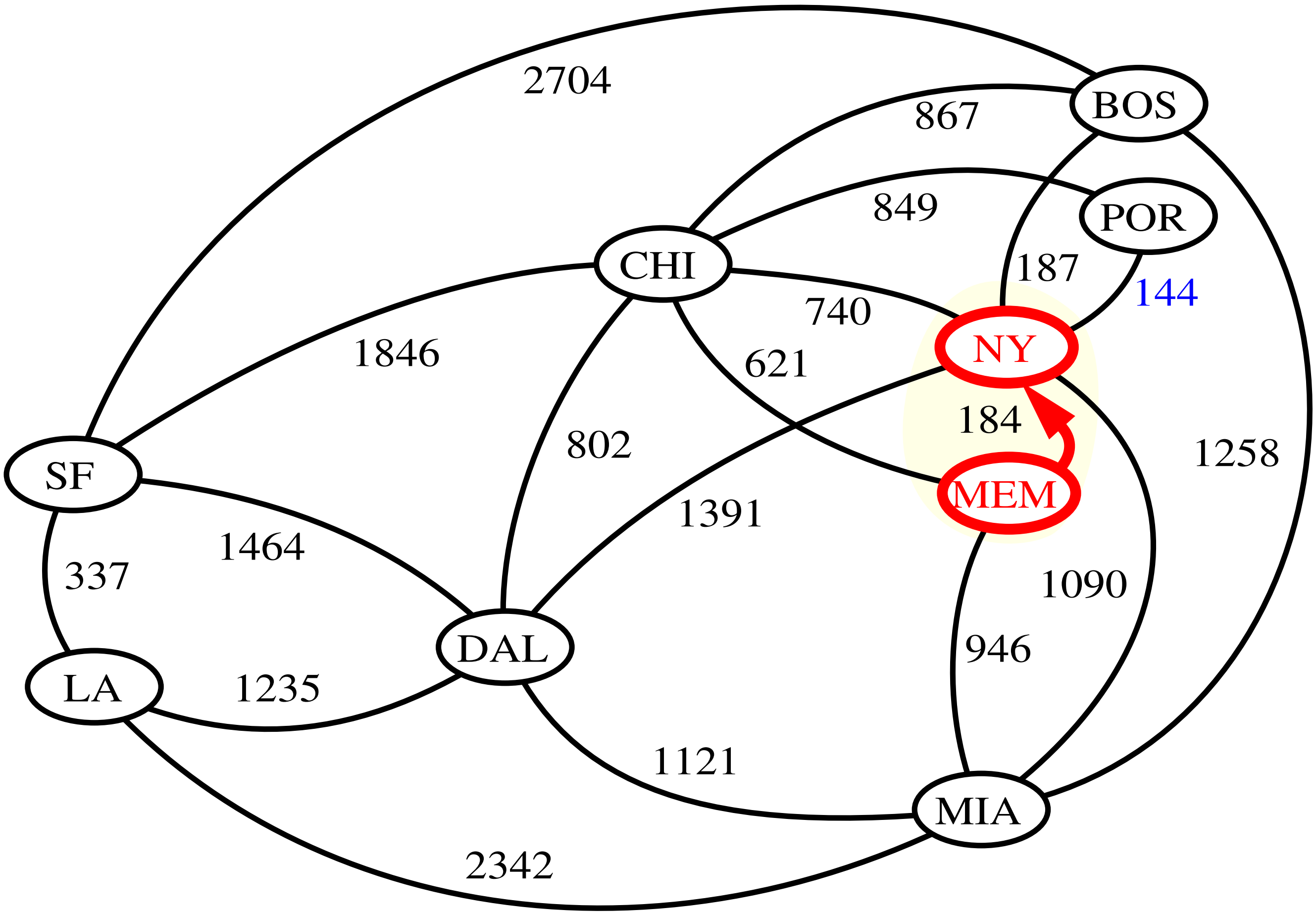
  initialize $D[v] \leftarrow 0$ and $D[u] \leftarrow +\infty$ for each vertex $v \neq u$
  let Q be a priority queue that contains all of the vertices of G using the D labels as keys.
  while $Q \neq \emptyset$ do
    {pull u into the cloud C}
    $u \leftarrow Q$.RemoveMin()
    for each vertex z adjacent to u such that z is in Q do
      {perform the relaxation operation on edge (u, z) }
      if $D[u] + w((u, z)) < D[z]$ then
        $D[z] \leftarrow D[u] + w((u, z))$
        Q.UpdateKey(z,$D[z]$)
  return the label $D[u]$ of each vertex u.

Eksempel: korteste stier fra MEM



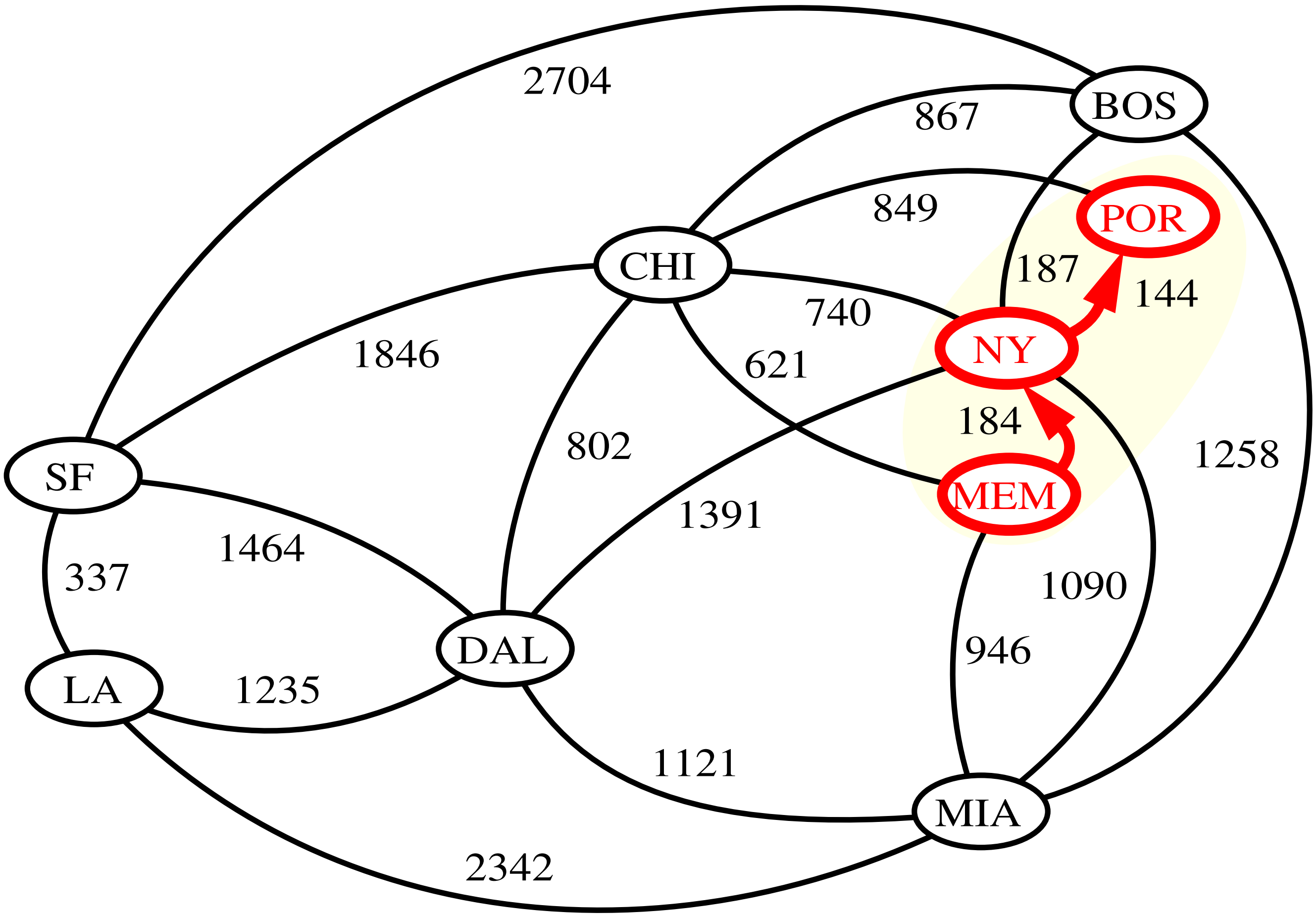
	parent	distance
BOS		∞
MEM		0
DAL		∞
NY	MEM	184
LA		∞
MIA	MEM	946
CHI	MEM	621
POR		∞
SF		∞

- NY er nærmest...



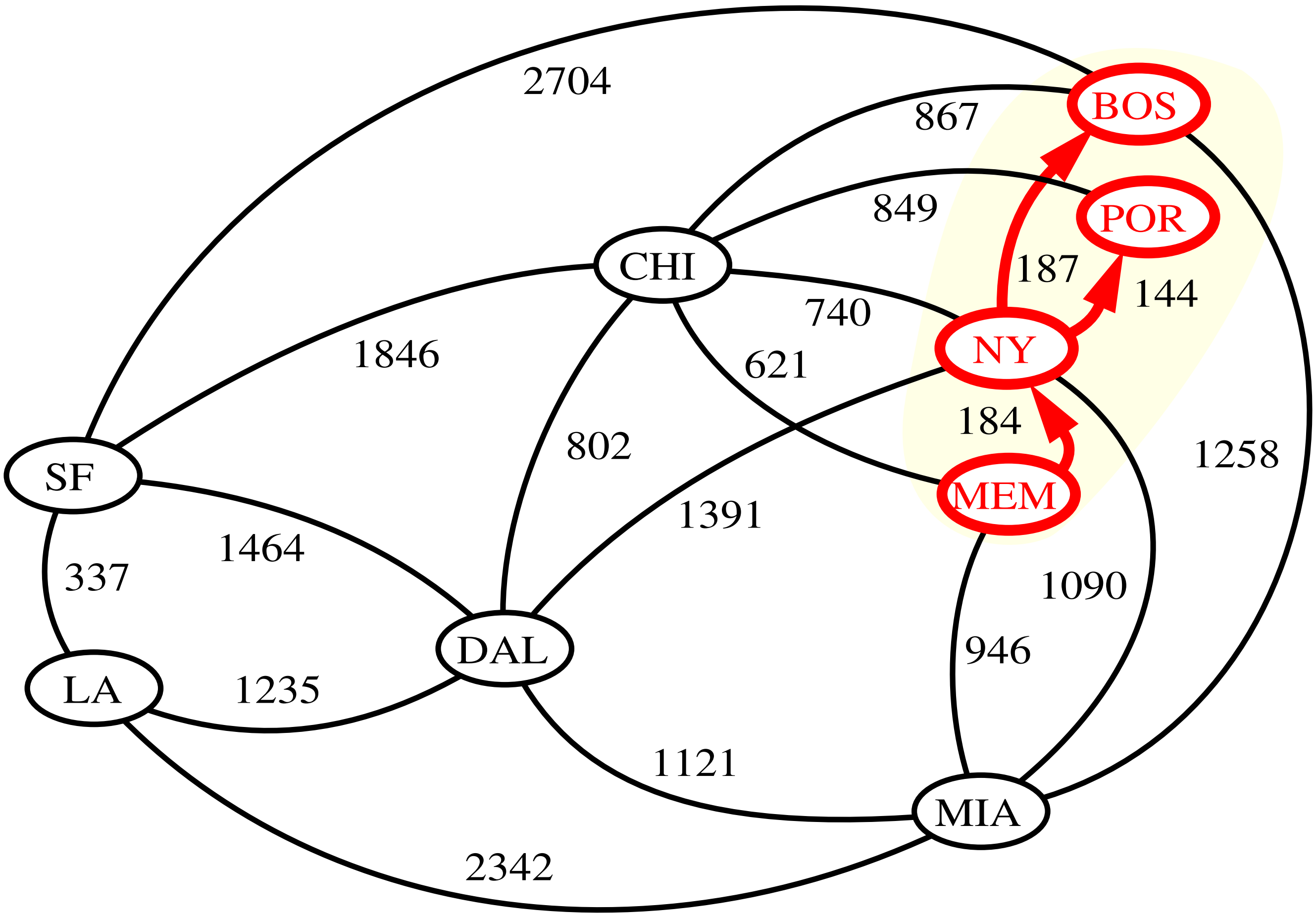
	parent	distance
BOS	NY	371
MEM		0
DAL	NY	1575
NY	MEM	184
LA		∞
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF		∞

- deretter POR.



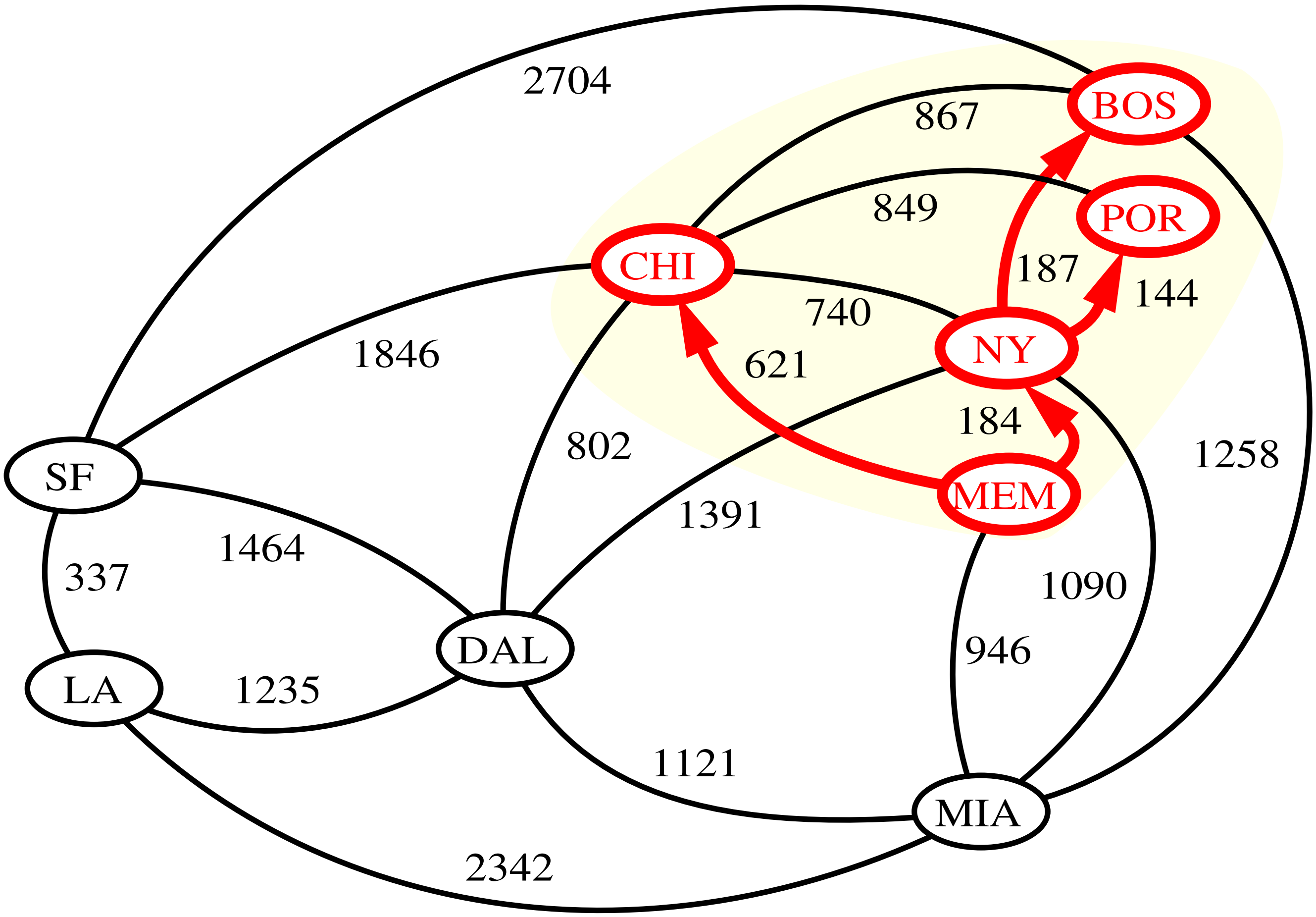
	parent	distance
BOS	NY	371
MEM		**0**
DAL	NY	1575
NY	**MEM**	**184**
LA		∞
MIA	MEM	946
CHI	MEM	621
POR	**NY**	**328**
SF		∞

- BOS er neste.



	parent	distance
BOS	NY	371
MEM		0
DAL	NY	1575
NY	MEM	184
LA		∞
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	BOS	3075

- CHI: Chicago kommer så.

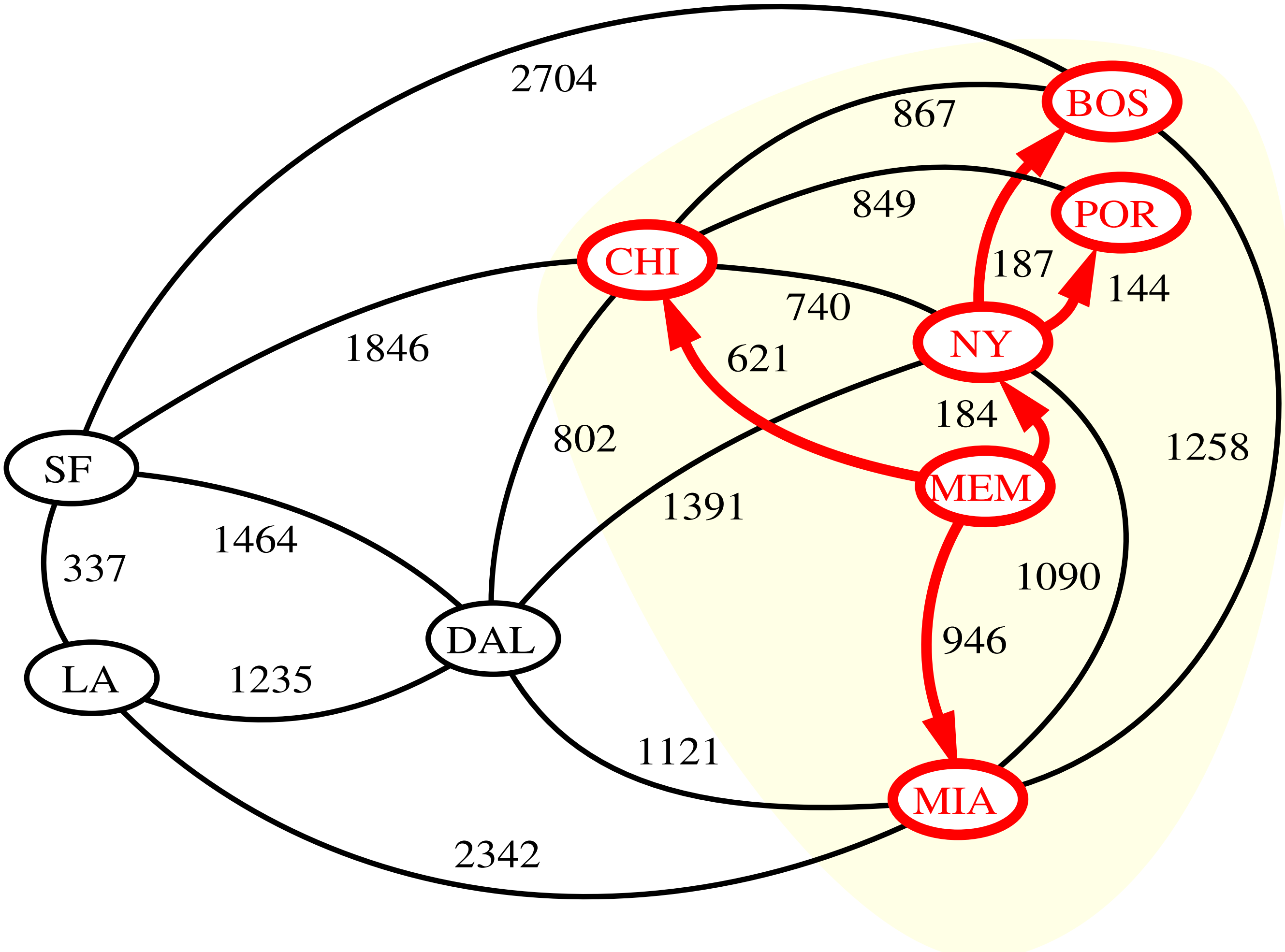


	parent	distance
BOS	NY	371
MEM		0
DAL	CHI	1423
NY	MEM	184
LA		∞
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	CHI	2467

NB:
D for DAL
oppdatert

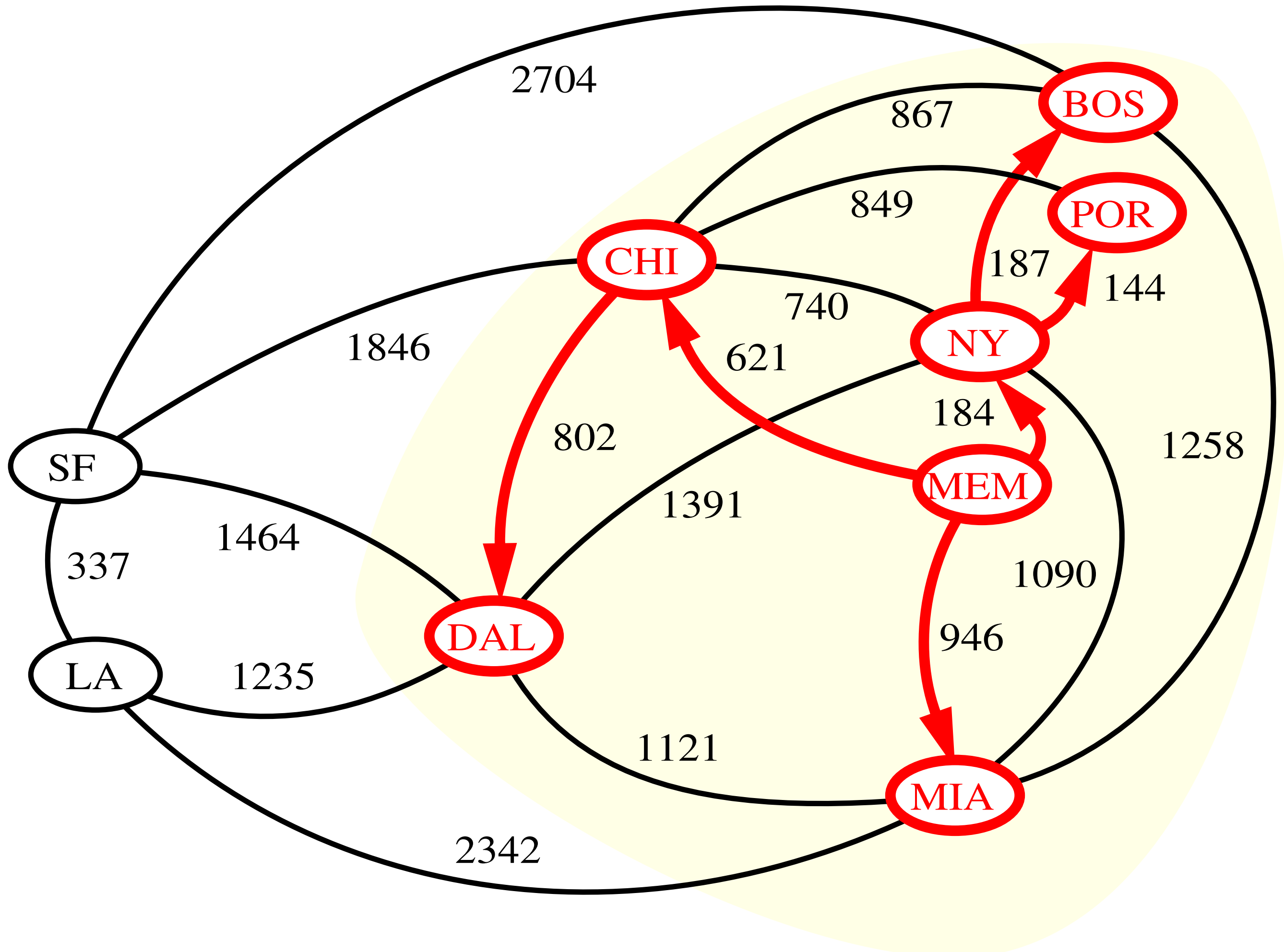
også for
SF

- MIA neste.



	parent	distance
BOS	NY	371
MEM		0
DAL	CHI	1423
NY	MEM	184
LA	MIA	3288
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	CHI	2467

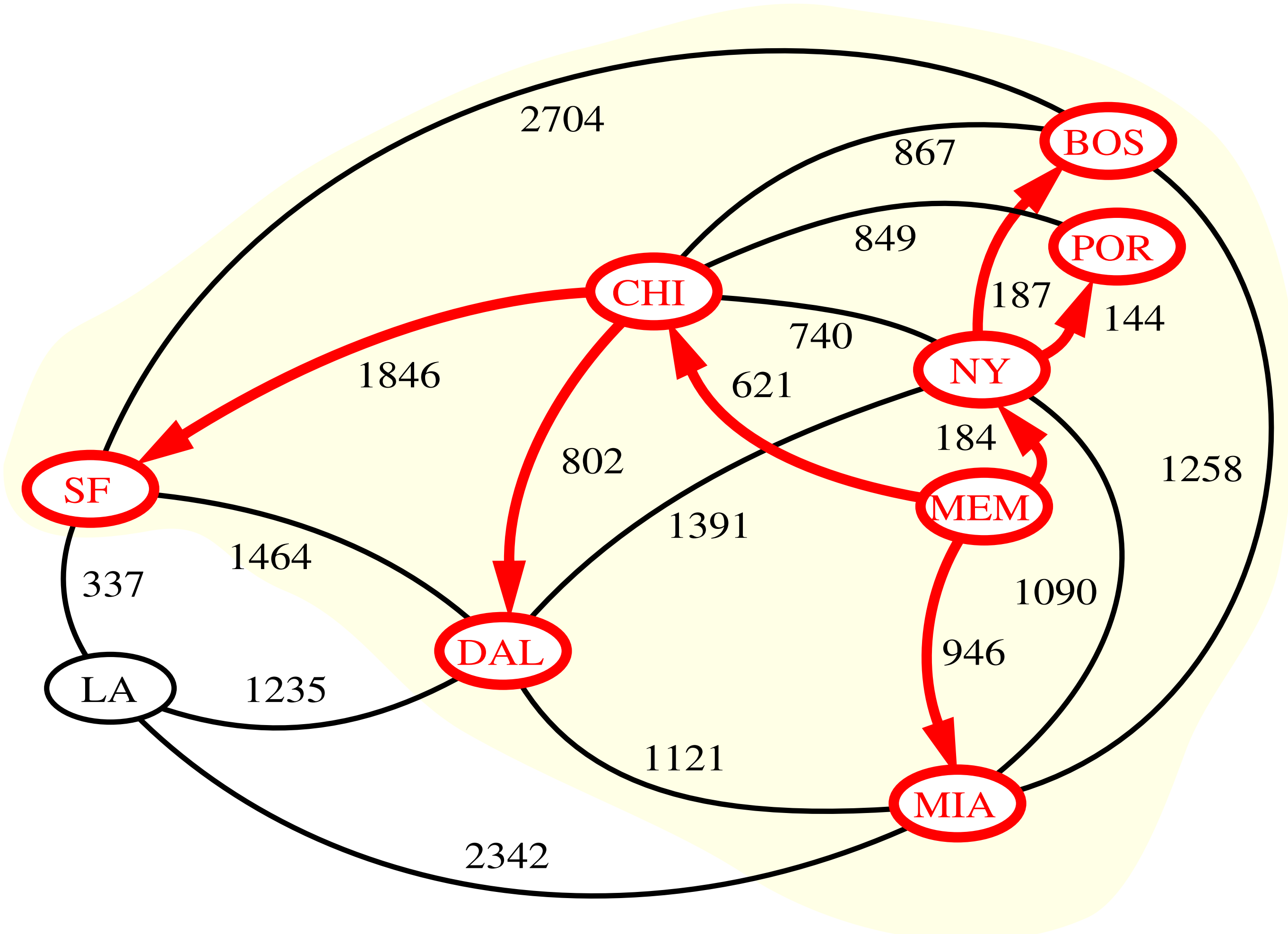
- DAL deretter.



	parent	distance
BOS	NY	371
MEM		0
DAL	CHI	1423
NY	MEM	184
LA	DAL	2658
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	CHI	2467

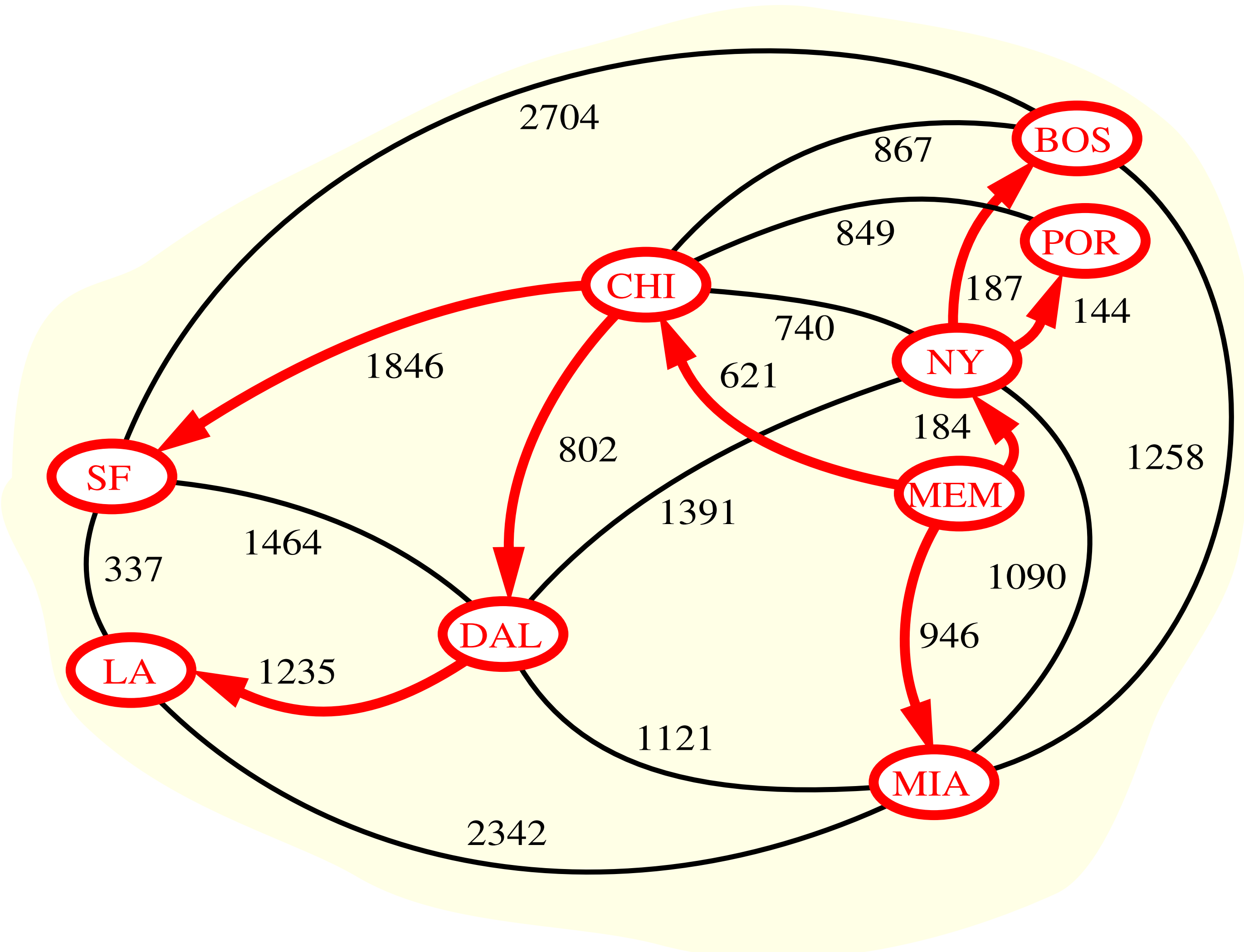
og D for LA blir oppdatert

- SF neste.



	parent	distance
BOS	NY	371
MEM		0
DAL	CHI	1423
NY	MEM	184
LA	DAL	2658
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	CHI	2467

- LA siste stopp.



	parent	distance
BOS	NY	371
MEM		0
DAL	CHI	1423
NY	MEM	184
LA	DAL	2658
MIA	MEM	946
CHI	MEM	621
POR	NY	328
SF	CHI	2467

Dijkstra-VersteFall Kjøretid

- Må raskt gå gjennom alle kanter utfra en node. Dvs vi representer grafen G vha naboliste. Anta G har n noder og m kanter.
- Antall PQ-operasjoner er:
  - n RemoveMin (en for hver node)
  - opptil m UpdateKey-operasjoner, siden hver kant-relaksering kan føre til UpdateKey.
- **Heap**-Implementasjon av prioritetskø Q:
  - Hver FjernMin er **O**(log n) vha Downheap.
  - Hver UpdateKey er O(log n) vha enten Upheap eller Downheap dersom vi bruker locators for å finne elementer i Q i tid O(1).
  - Total kjøretid blir $\mathbf{O}((n+m)\log n)$ eller $\mathbf{O}(n^2 \log n)$.
- **Usortert sekvens-**Implementasjon av Q:
  - Hver FjernMin tar $\mathbf{O}(n)$.
  - Hver UpdateKey er $\mathbf{O}(1)$.
  - Kjøretid blir $\mathbf{O}(n^2 + m) = O(n^2)$
- Dvs heap er best for grafer med 'få' kanter, dvs om m < $n^2 / \log n$, og usortert sekvens for grafer med 'mange' kanter, dvs m > $n^2 / \log n$.

Dijkstra- Forventet Kjøretid

- ***Forventet kjøretid*** blir noe annet:
  - Hvor mange UpdateKey kan vi forvente? UpdateKey for hver eneste kant-relaksering er usannsynlig.
  - Dersom vi anvender **random kant-gjennomgang** så kan vi forvente O(log n) UpdateKey per node.
  - Dvs heap-implementasjon har forventet kjøretid $\mathbf{O}(n\log n+m)$, mens usortert-sekvens-implementasjon har forventet kjøretid $\mathbf{O}(n^2)$.
  - Heap-implementasjon er dermed å foretrekke i nesten alle situasjoner.

Dijkstra's Algoritme, flere anliggender...

- I eksempelet er **kantvekt** geografisk distanse. Men det kunne også vært flytid eller billettpris.
- Det er enkelt å modifisere Dijkstra's algoritme for forskjellige situasjoner:
- Hva om vi kun vil ha vektsum av korteste sti til en enkelt node x? Svar: Stopp såsnart x er i skyen. Bruk locator for elementer i Q slik at x kan finne sin nøkkelverdi=avstand.
- Hva om vi vil ha også nodene på denne korteste stien? Svar: Lagre SSSP-treet (rødt i eksempelet) vha forelder-pekere, og så traversere fra x via forelderpekere til startnoden.