# Riktade & viktade grafer.

## TDDE22, 725G97: DALG

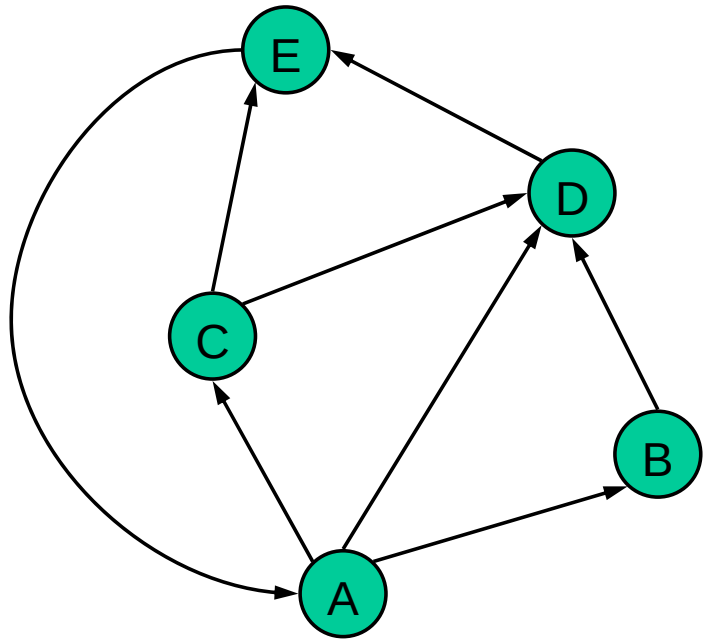
Magnus Nielsen

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

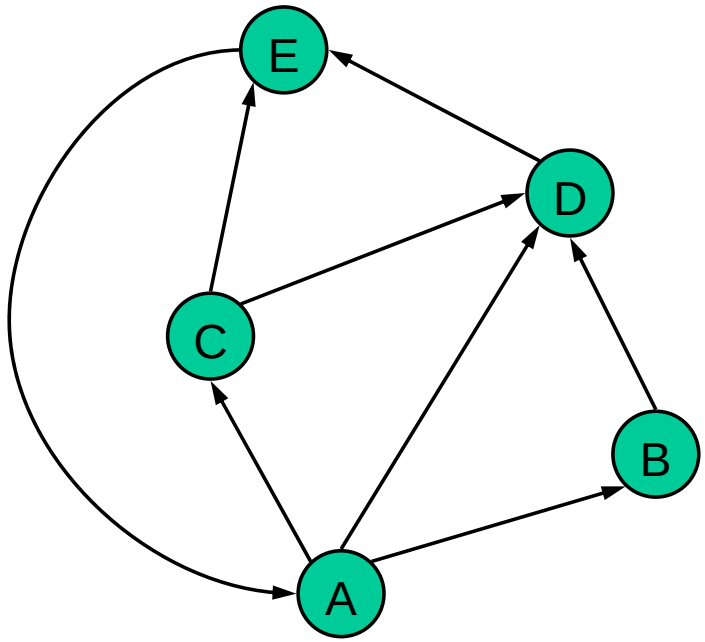
# Introduktion

- I en riktad graf är alla bågar riktade

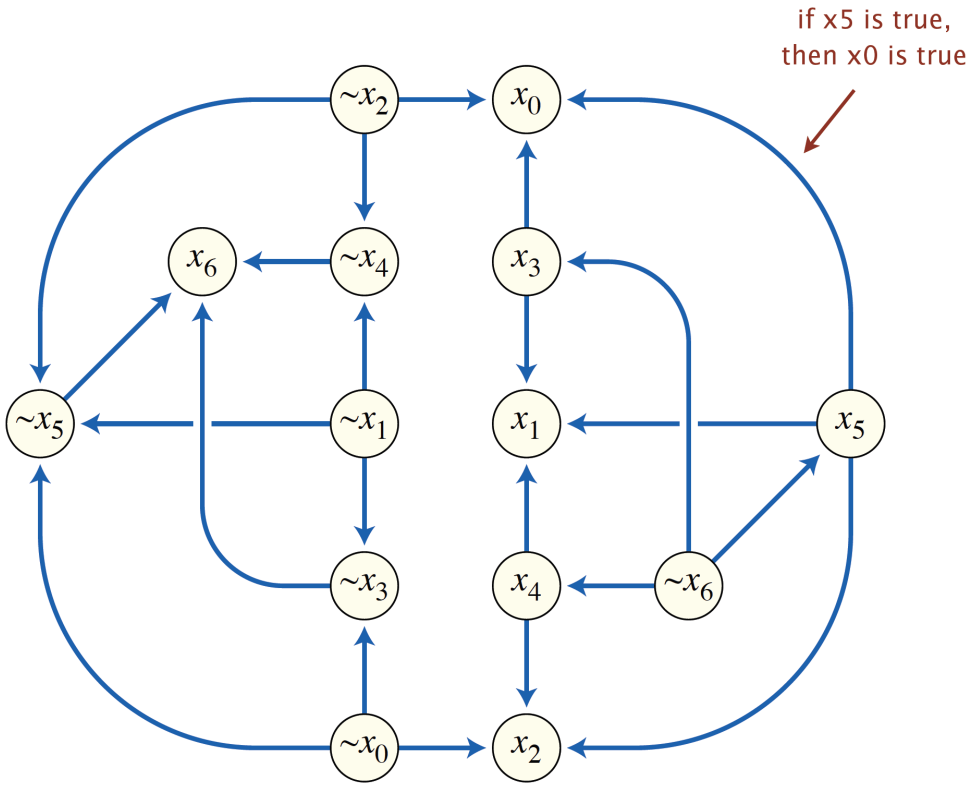


# Egenskaper

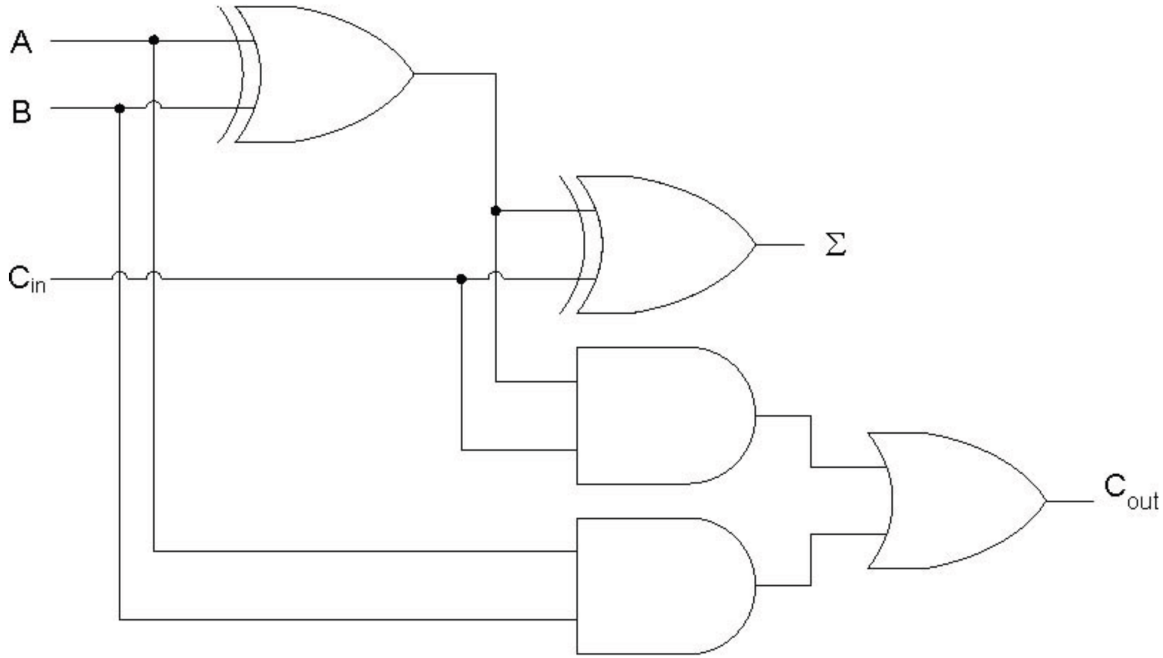
- En graf $G = (V, E)$ sådan att varje båge går i en riktning:
  - Båge $(a, b)$ går från $a$ till $b$ men inte från $b$ till $a$.
- Om $G$ är enkel (inga parallella bågar och inga öglor/loopar) gäller $m \leq n \cdot (n-1)$, d.v.s. $m \in O(n^2)$, där m är antalet bågar och n är antalet noder.



# Implikationsgraf



# Kombinatorisk krets



# Tillämpningar

| riktad graf | nod | riktad båge |
|---|---|---|
| transport | gatukorsning | enkelriktad gata |
| www | hemsida | hyperlänk |
| näringskedja | art | rovdjur-byte-förhållande |
| schemaläggning | uppgift | föregångarvillkor |
| finansiell | bank | transaktion |
| mobiltelefon | person | ringt samtal |
| smittsam sjukdom | person | infektion |
| citeringar | artikel | citering |
| objektgraf | objekt | pekare |
| arvshierarki | klass | ärver från |
| kontrollflöde | kodblock | hopp |

# Några algoritmiska grafproblem

- Stig. Finns det en riktad stig från $s$ till $t$?
- Kortaste väg. Vilken är den kortaste riktade stigen från $s$ till $t$?

# Några algoritmiska grafproblem

- Stig. Finns det en riktad stig från $s$ till $t$?
- Kortaste väg. Vilken är den kortaste riktade stigen från $s$ till $t$?
- Stark konnektivitet. Finns det en riktad stig mellan alla par av noder?

# Några algoritmiska grafproblem

- Stig. Finns det en riktad stig från $s$ till $t$?
- Kortaste väg. Vilken är den kortaste riktade stigen från $s$ till $t$?
- Stark konnektivitet. Finns det en riktad stig mellan alla par av noder?
- Topologisk sorting. Går det att rita den riktade grafen så att alla kanter pekar uppåt?

# Några algoritmiska grafproblem

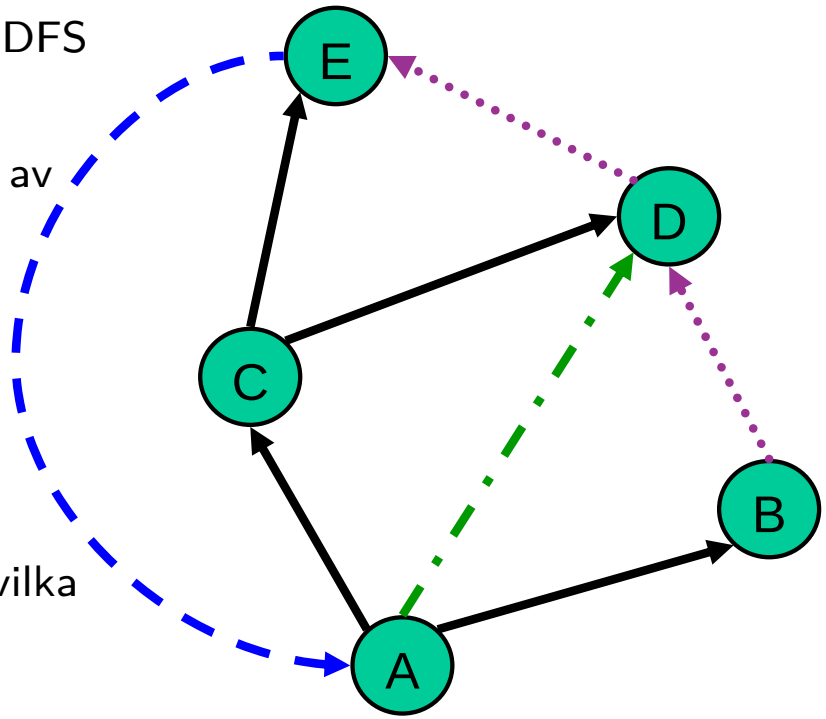
- Stig. Finns det en riktad stig från $s$ till $t$?
- Kortaste väg. Vilken är den kortaste riktade stigen från $s$ till $t$?
- Stark konnektivitet. Finns det en riktad stig mellan alla par av noder?
- Topologisk sorting. Går det att rita den riktade grafen så att alla kanter pekar uppåt?
- Transitivt hölje. För vilka noder $v$ och $w$ finns det en stig från $v$ till $w$?

# Några algoritmiska grafproblem

- Stig. Finns det en riktad stig från $s$ till $t$?
- Kortaste väg. Vilken är den kortaste riktade stigen från $s$ till $t$?
- Stark konnektivitet. Finns det en riktad stig mellan alla par av noder?
- Topologisk sorting. Går det att rita den riktade grafen så att alla kanter pekar uppåt?
- Transitivt hölje. För vilka noder $v$ och $w$ finns det en stig från $v$ till $w$?
- Page Rank. Hur betydelsefull är en webbsida?

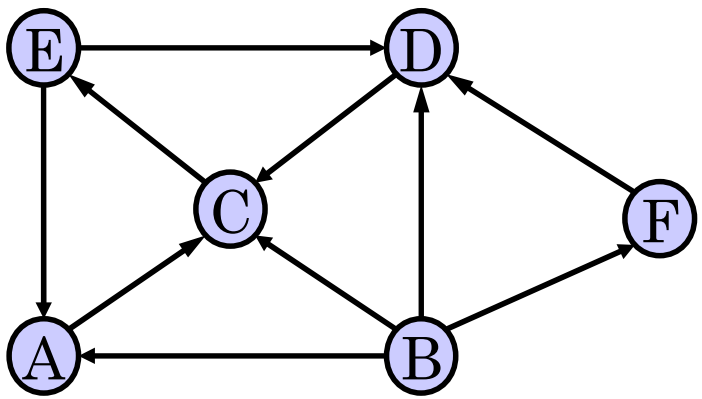
# Riktad DFS

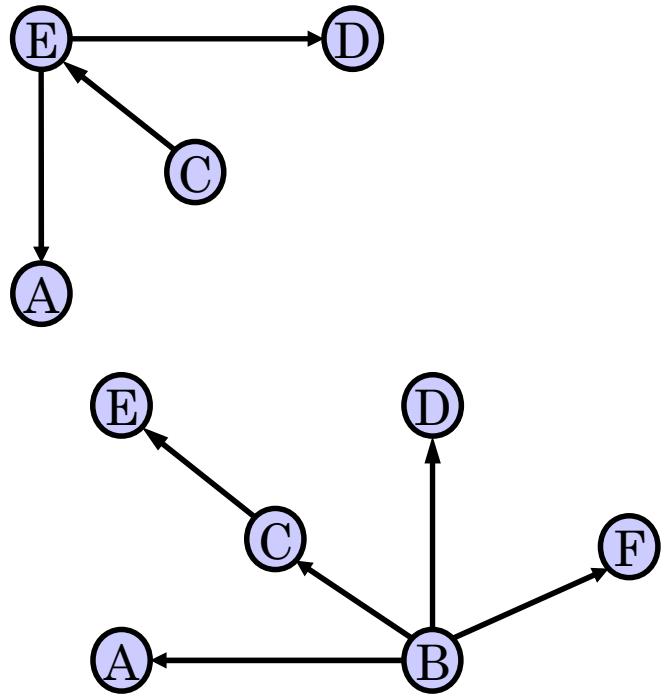
- Vi kan specialisera traverseringsalgoritmerna (DFS och BFS) till riktade grafer
- I den riktade DFS-algoritmen får vi fyra typer av bågar
  - ”discovery”-bågar
  - bakåt-bågar
  - framåt-bågar
  - korsande bågar
- En riktad DFS med start i nod $v$ bestämmer vilka noder som är nåbara från $v$
- Skapar en spännande skog av grafen



# Nåbarhet

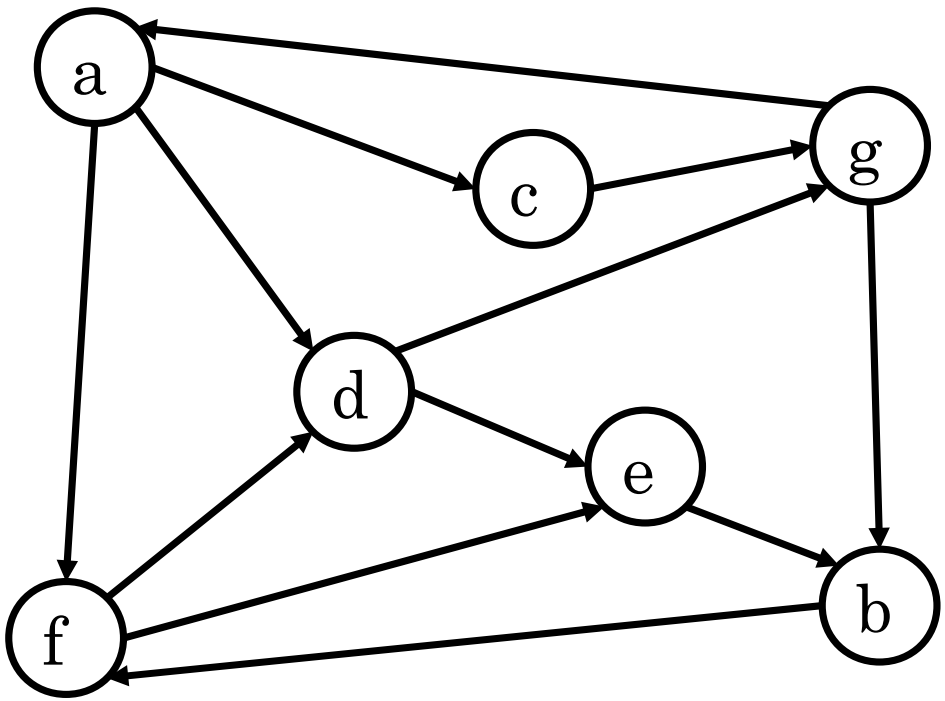
DFS-träd rotat i $v$: noder nåbara från $v$ via riktade stigar



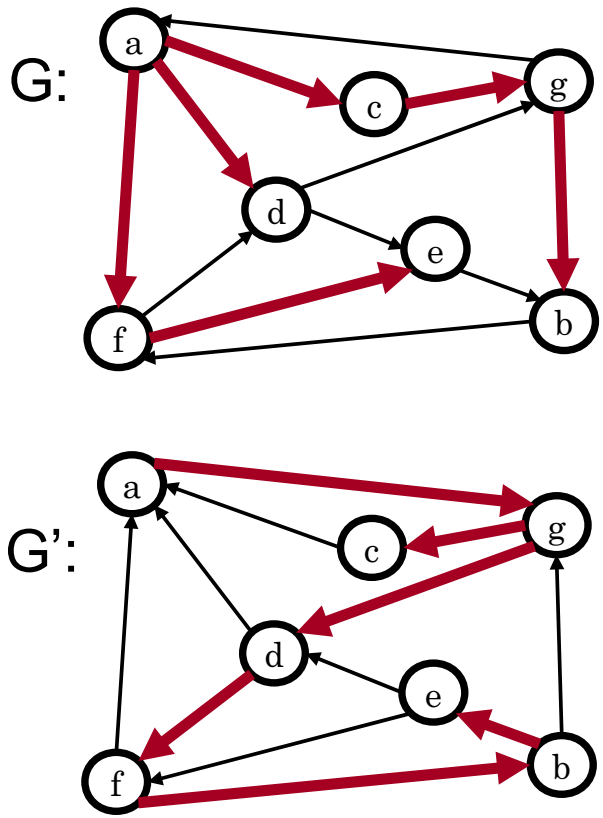
# Starkt sammanhängande

Varje nod är nåbar från alla andra noder



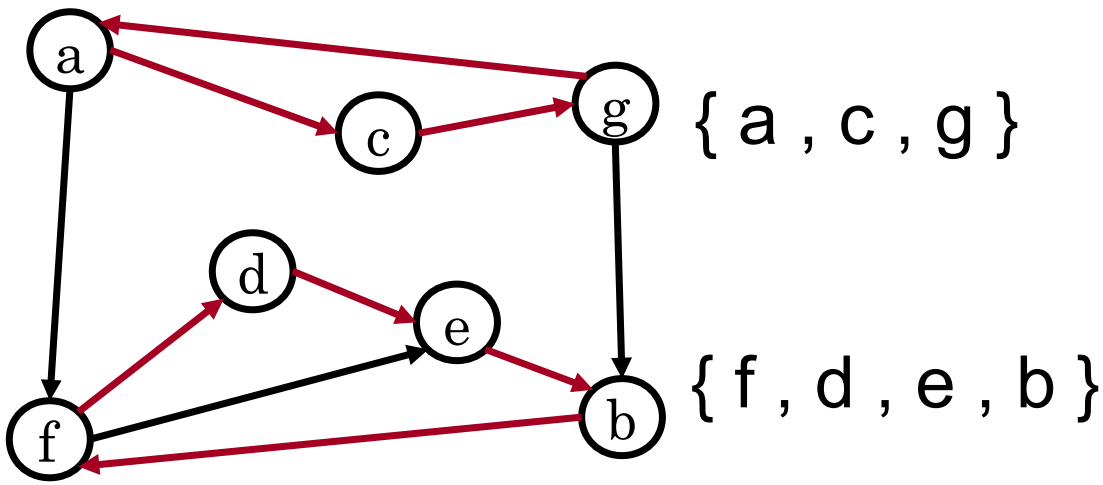
# Algoritm för att avgöra starkt sammanhängande

- Välj en nod $v$ i $G$
- // Kan alla noder nås från $v$?
  Utför DFS från $v$ i $G$
  - Om det finns någon nod $w$ som inte besöks svara "nej"
- Låt $G'$ vara $G$ med riktningen på varje båge omkastad
- // Kan $v$ nås från alla noder?
  Utför DFS från $v$ i $G'$
  - Om det finns någon nod $w$ som inte besöks svara "nej"
  - Annars, svara "ja"
- Körtid: $O(n+m)$



# Starkt sammanhängande komponenter

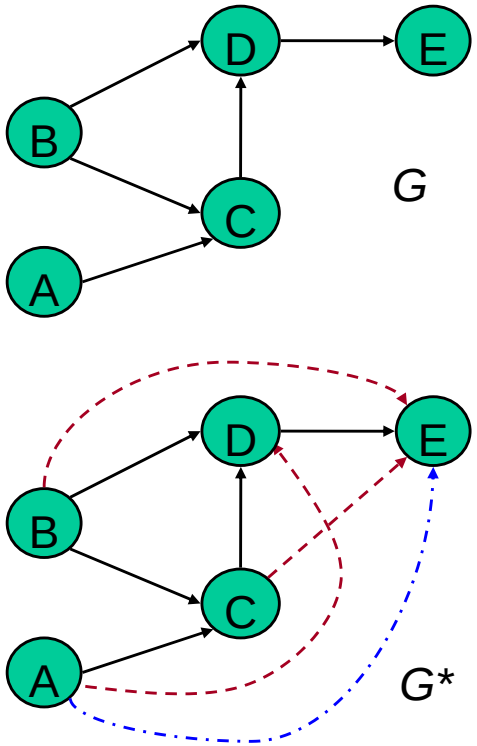
- Maximal delgraf sådan att varje nod kan nå alla andra noder i delgrafen
- Kan också göras i $O(n+m)$ tid genom att använda DFS i flera steg



LINKÖPING UNIVERSITY

# Transitivt hölje

- Givet en riktad graf $G$, låt det transitiva höljet av $G$ vara den riktade grafen $G^*$ sådan att
  - $G^*$ har samma noder som $G$
  - om $G$ har en riktad stig från $u$ till $v$ $(u \neq v)$ så har $G^*$ en riktad båge från $u$ till $v$
- Det transitiva höljet ger information om nåbarheten i en riktad graf



# Beräkning av transitiva höljet

- Vi kan köra DFS med start i varje nod $v_1, \ldots, v_n$, alltså $O(n \cdot (n + m))$
- Floyd-Warshall's algoritm
  Robert Floyd 1962: Hitta kortaste väger i viktade grafer
  Stephen Warshall 1962: Transitivt hölje

  I princip identiska med Robert Roy 1959.

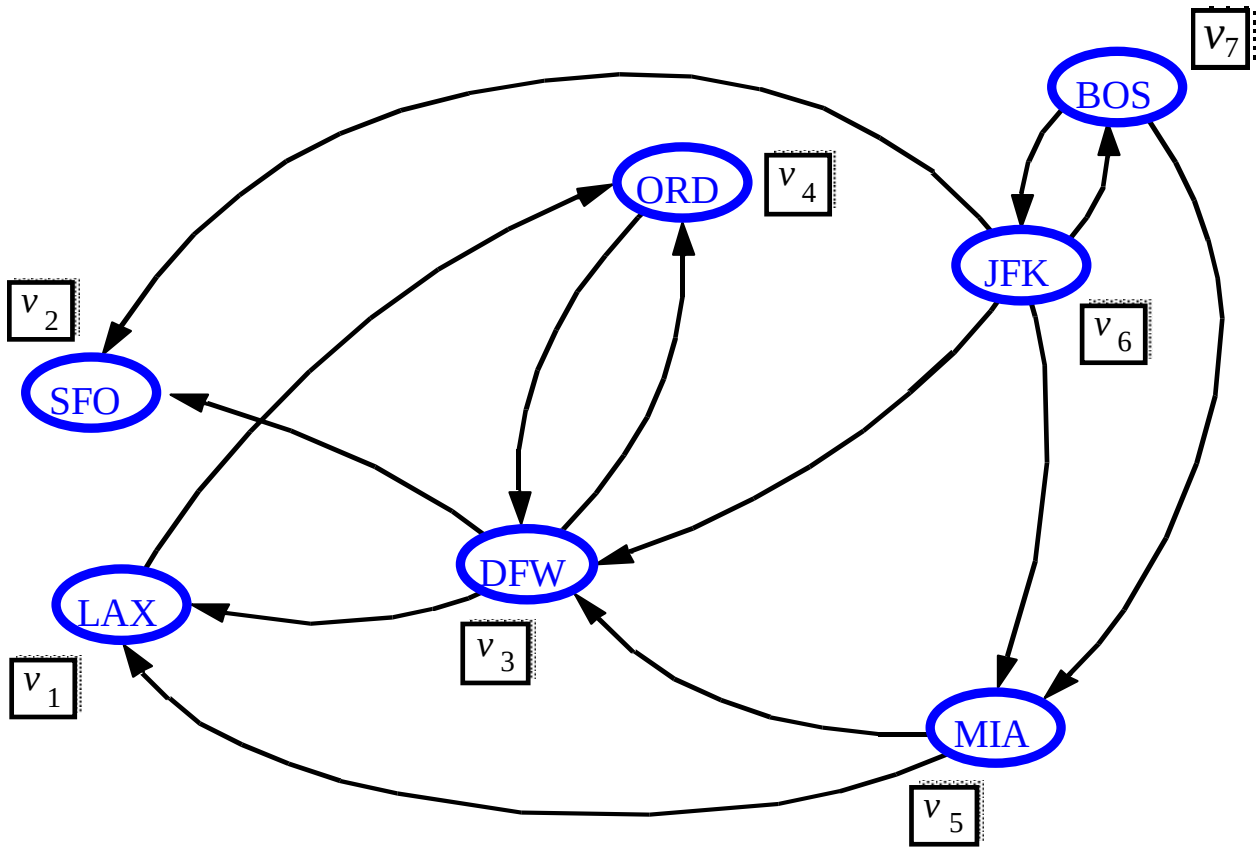
  1. Numrera noderna 1,2,...,n.
  2. I varje fas $k$ beräkna kortaste vägen mellan samtliga noder $i, j$
  3. I fas $k$ tar vi bara hänsyn till stigar som använder noder med nummer 1,2,...,k som mellanliggande noder

  $\Theta(n^3)$

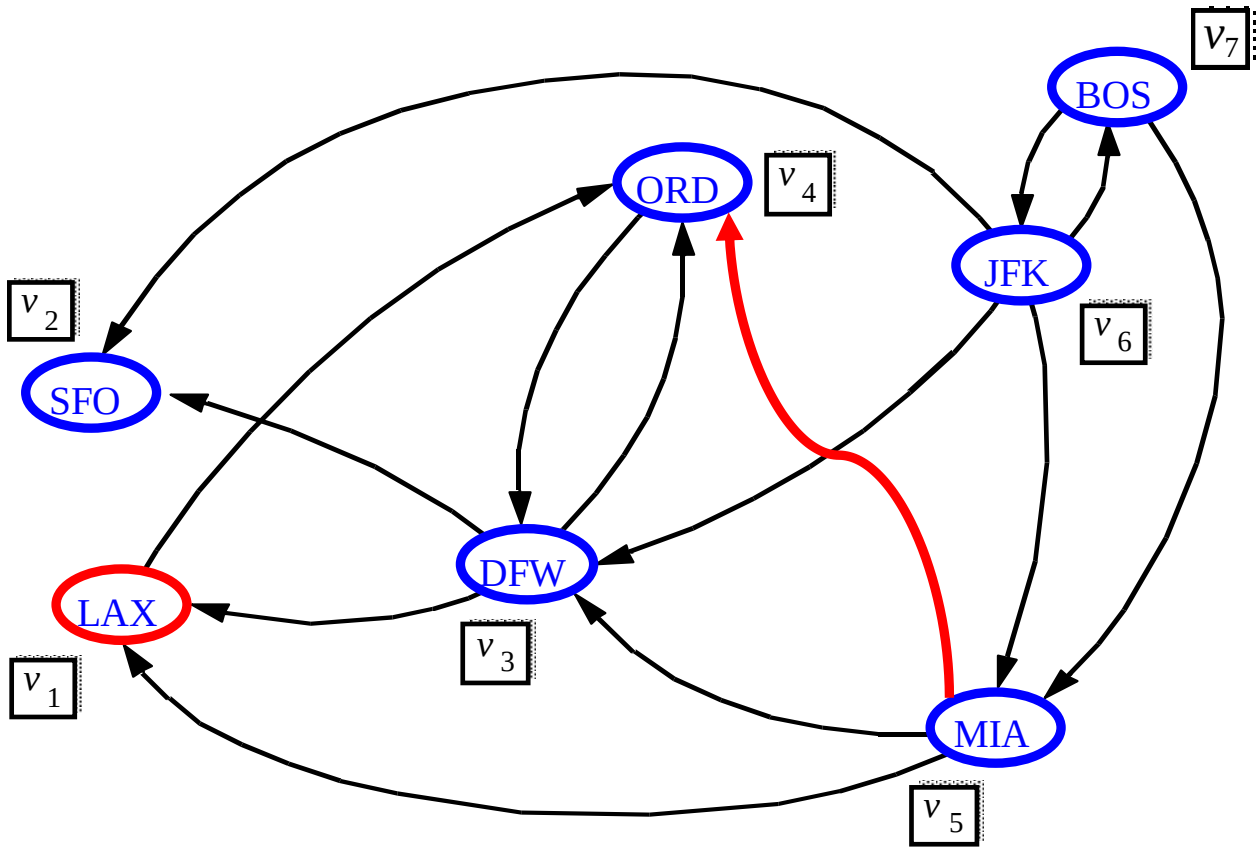
# Floyd-Warshalls algoritm

**function** FLOYDWARSHALL($G$)
    $G_0 \leftarrow G$
    **for** $k \leftarrow 1$ **to** $n$ **do**
        $G_k \leftarrow G_{k-1}$
        **for** $i \leftarrow 1$ **to** $n$ $(i \neq k)$ **do**
            **for** $j \leftarrow 1$ **to** $n$ $(j \neq i, k)$ **do**
                **if** $G_{k-1}$.AREADJACENT$(v_i, v_k)$ **then**
                    **if** $G_{k-1}$.AREADJACENT$(v_k, v_j)$ **then**
                        **if** $\neg G_k$.AREADJACENT$(v_i, v_j)$ **then**
                            $G_k$.INSERTDIRECTEDEDGE$(v_i, v_j, k)$
    **return** $G_n$

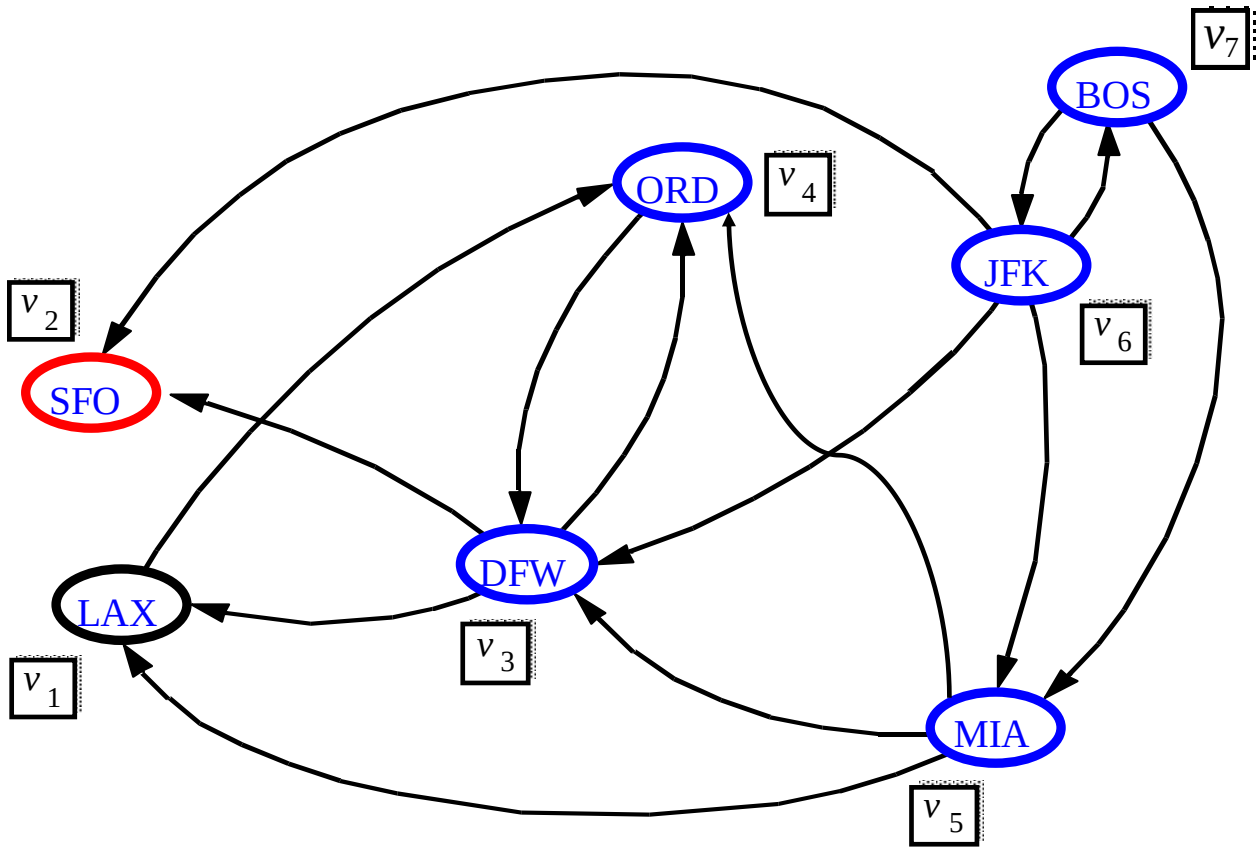
# Exempel: Floyd-Warshall



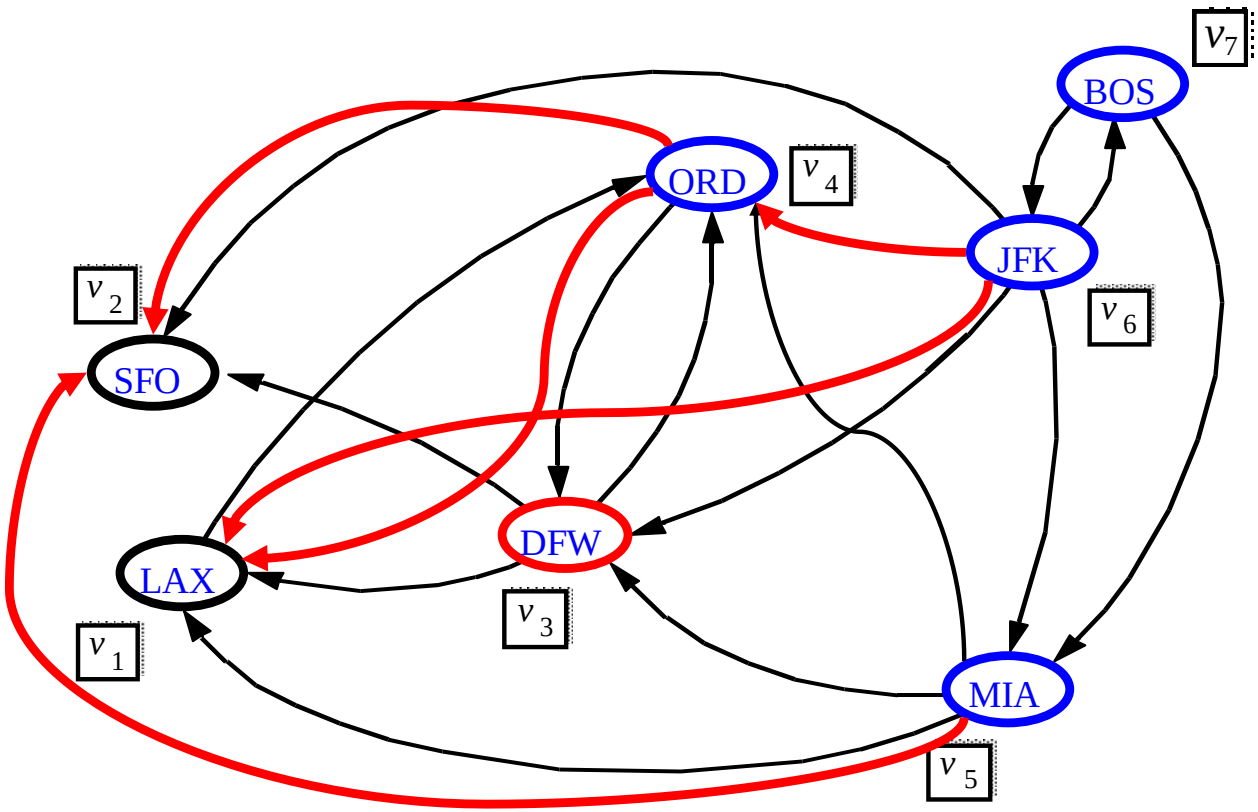
# Floyd-Warshall, iteration 1



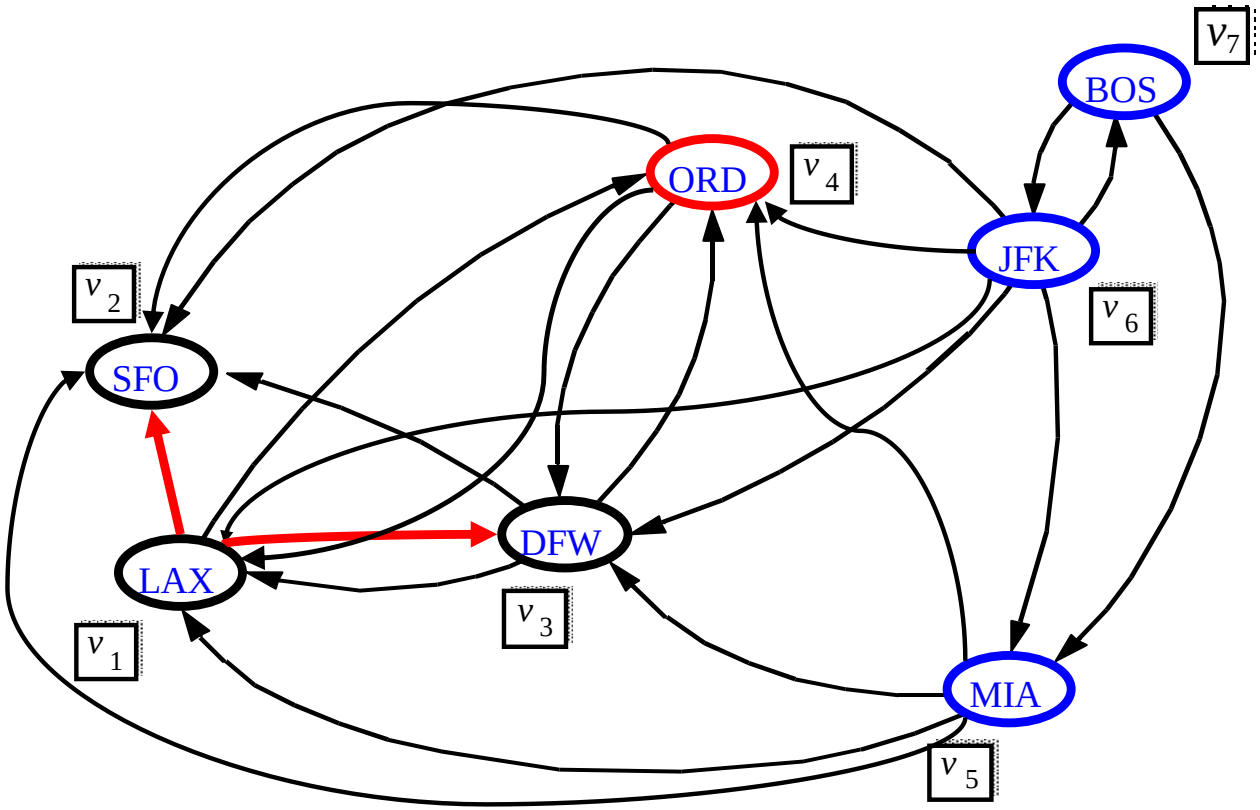
# Floyd-Warshall, iteration 2



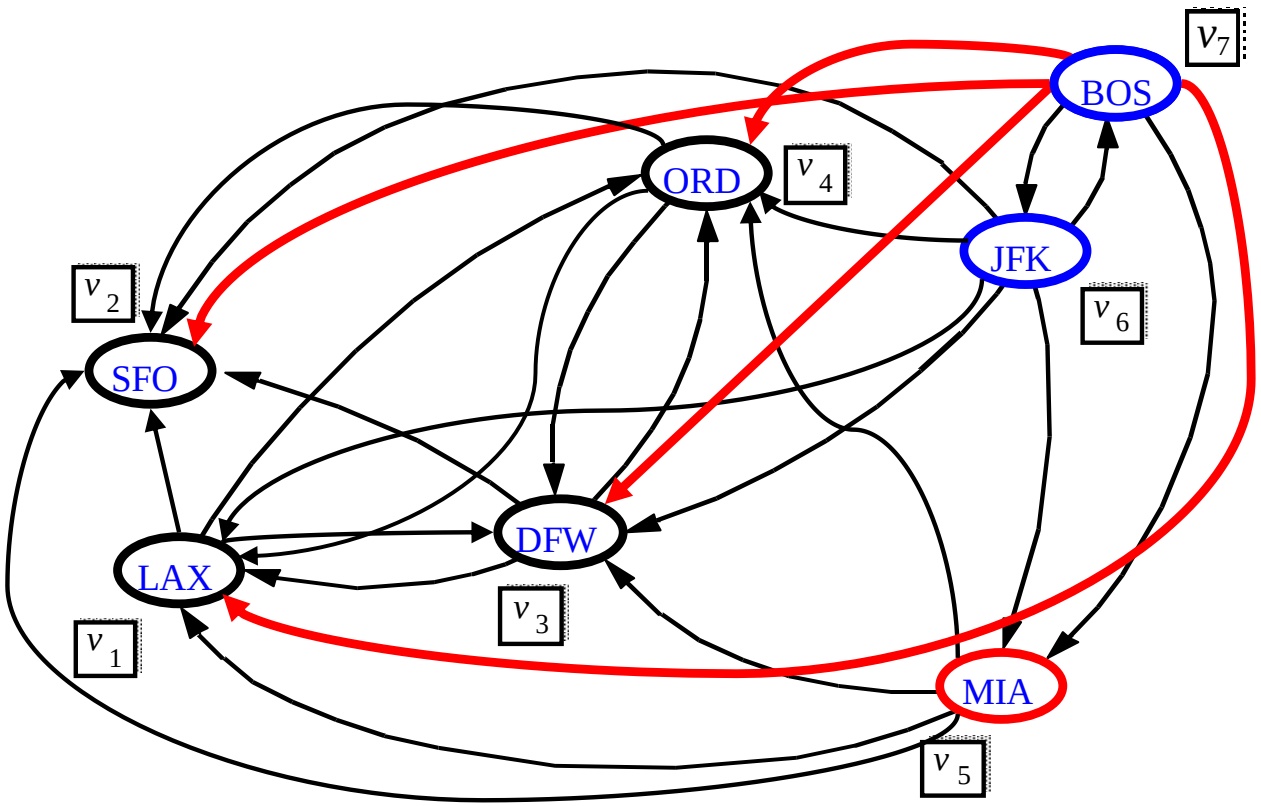
# Floyd-Warshall, iteration 3



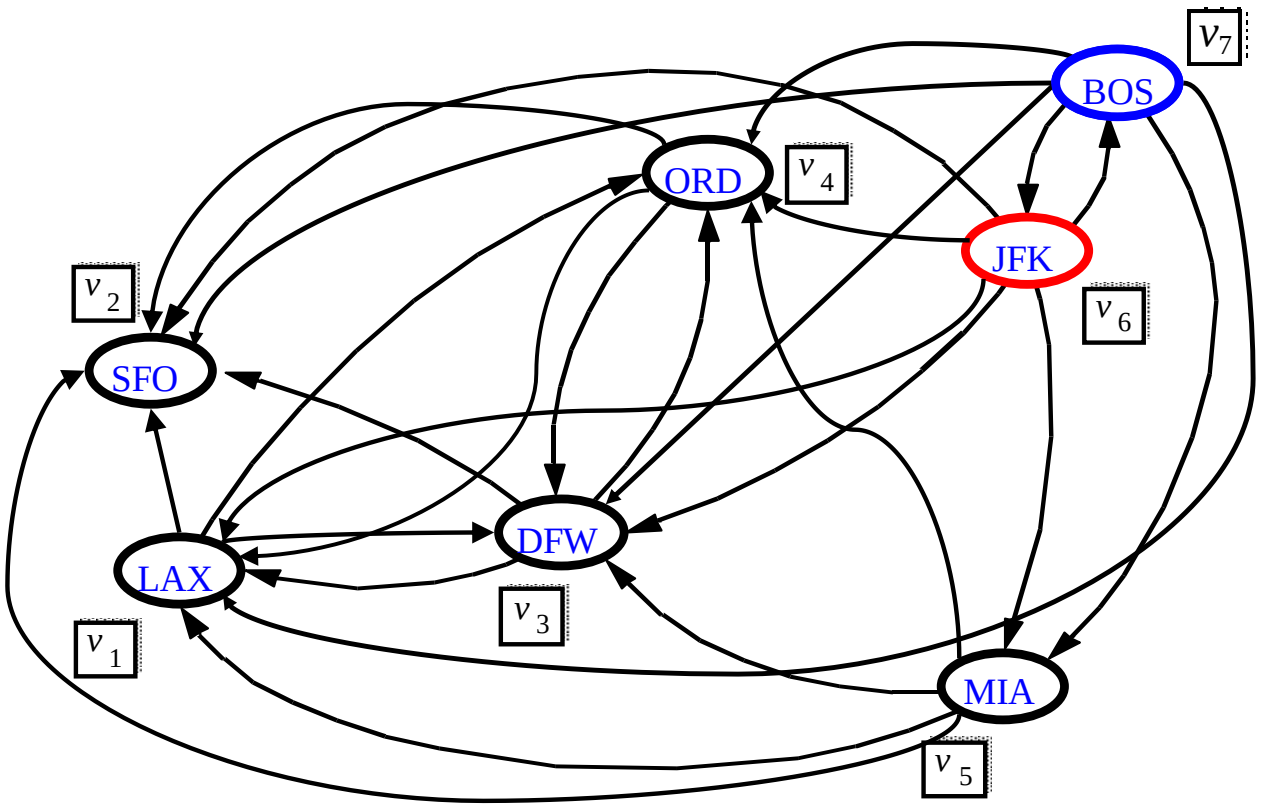
# Floyd-Warshall, iteration 4



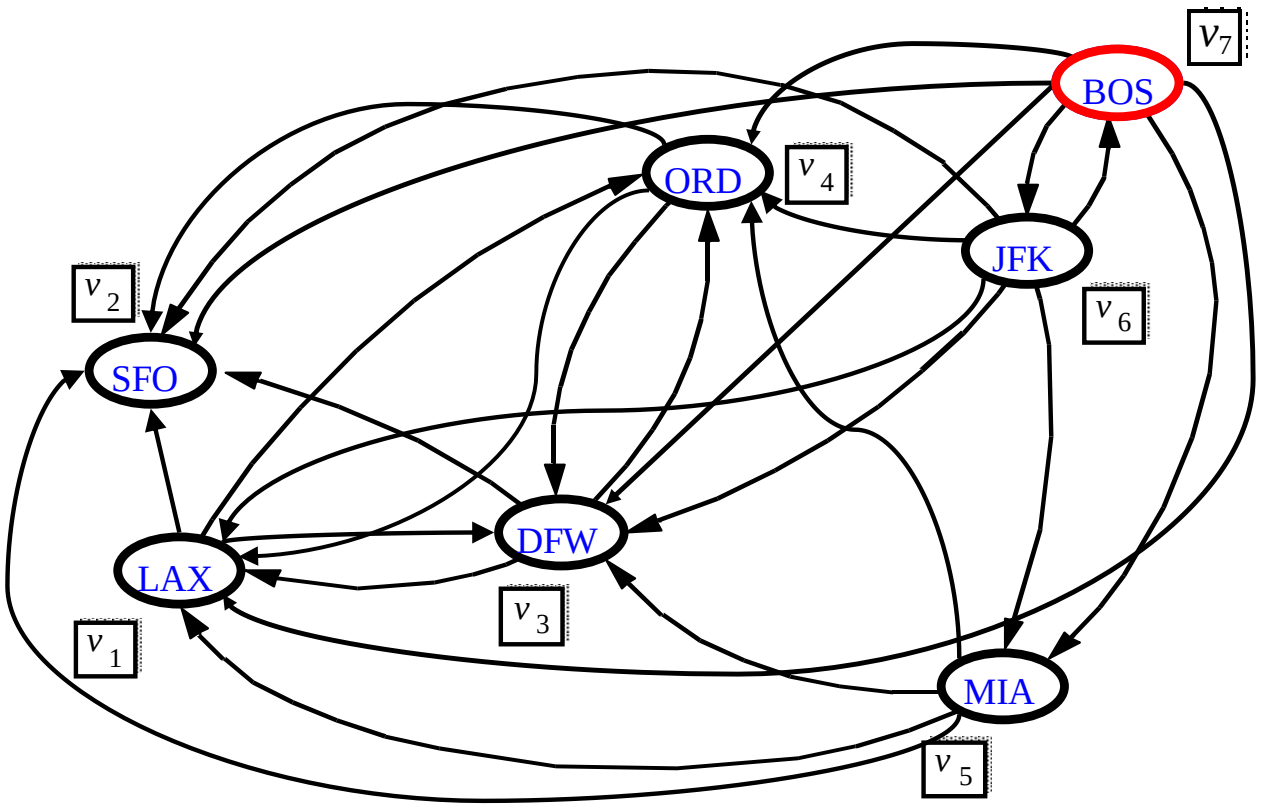
# Floyd-Warshall, iteration 5



# Floyd-Warshall, iteration 6



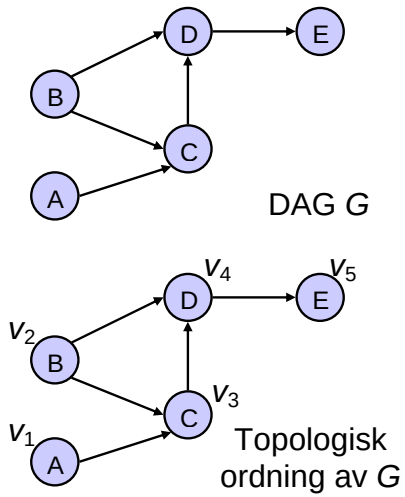
# Floyd-Warshall, slutet



LINKÖPING UNIVERSITY

# Riktade acykliska grafer och topologisk ordning

- En riktad acyklisk graf (DAG) är en riktad graf som inte har några riktade cykler
- En topologisk ordning av en graf är en totalordning $v_1, \ldots, v_n$ av noderna sådan att varje båge $(v_i, v_j)$ uppfyller $i < j$
- Exempel: I en riktad graf som svarar mot en instans av uppgiftsschemaläggning är en topologisk ordning en sekvens av uppgifter som uppfyller kraven på inbördes ordning mellan uppgifterna



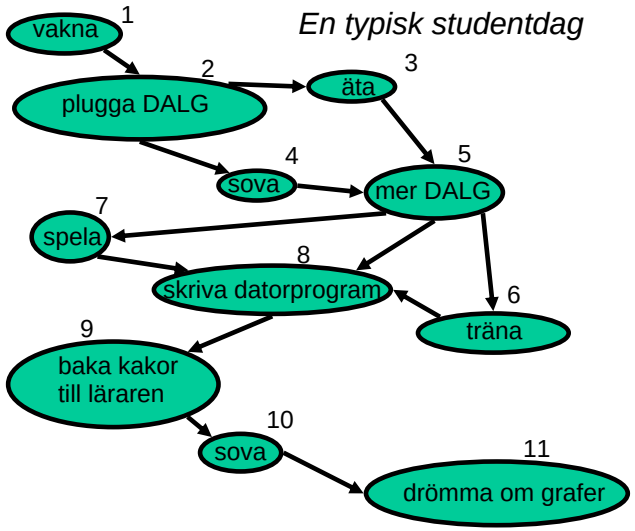

**Proposition 1** *En riktad graf går att ordna topologiskt omm grafen är en DAG*

# Varför topologisk ordning?

- Schemaläggning av arbeten
  - noder motsvarar arbeten
  - bågar motsvarar beroenden
- Utvärdering av celler vid beräkning i, exempelvis, excel-ark
- Logiksyntetisering
- Symbolberoenden i länkare (kompilatorer)
- Serialisering av data
- ...

# Topologisk sortering

Numrera noderna, så att $(u, v) \in E \Rightarrow u < v$



# Algoritm för topologisk sortering

```
procedure TopologicalSort(G)
    S ← ny tom stack
    for all u ∈ G.vertices() do
        låt incounter(u) vara ingraden för u
        if incounter(u) = 0 then
            S.push(u)
    i ← 1
    while ¬S.isEmpty() do
        u ← S.pop()
        låt u få nummer i i den topologiska ordningen
        i ← i + 1
        for all utgående kanter (u, w) från u do
            incounter(w) ← incounter(w) − 1
            if incounter(w) = 0 then
                S.push(w)
```

# Alternativ algoritm för topologisk sortering

**procedure** TopologicalSort($G$)
  $H \leftarrow G$ ▷ temporär kopia av $G$
  $n \leftarrow G$.numVertices
  **while** $H$ är icke-tom **do**
    låt $v$ vara en nod utan utgående bågar
    märk $v$ med $n$
    $n \leftarrow n - 1$
    ta bort $v$ från $H$

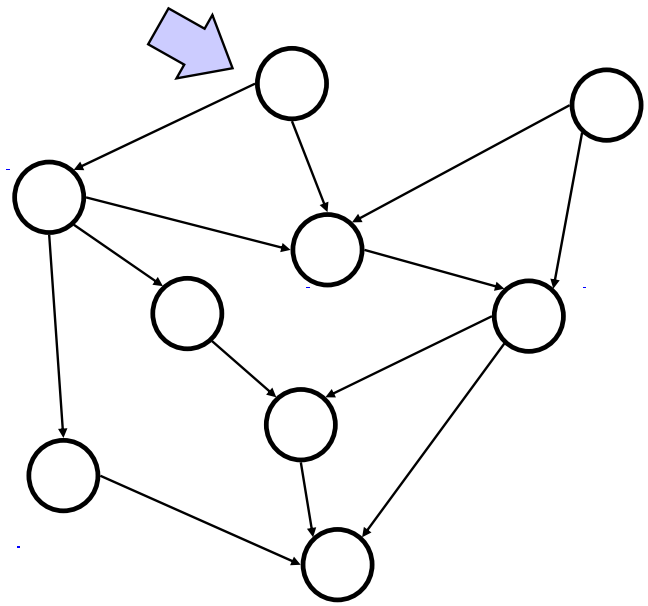
Körtid: $O(n + m)$. Hur då...?

# Algoritm för topologisk sortering via DFS

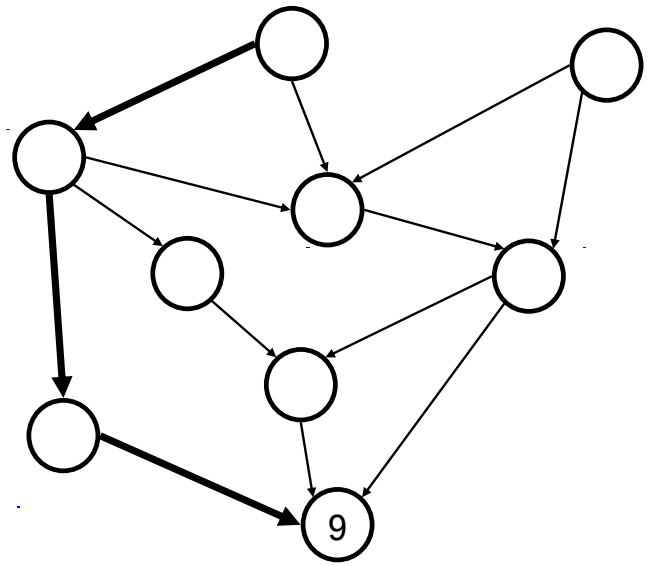
**procedure** TOPOLOGICALDFS($G$)
  $n \leftarrow G$.NUMVERTICES
  sätt alla noder och bågar till $UNEXPLORED$ som i DFS
  **for all** $v \in G$.VERTICES() **do**
    **if** GETLABEL($v$) $= UNEXPLORED$ **then**
      TOPOLOGICALDFS($G, v$)

**procedure** TOPOLOGICALDFS($G, v$)
  SETLABEL($v, VISITED$)
  **for all** $e \in G$.INCIDENTEDGES($v$) **do**
    **if** GETLABEL($e$) $= UNEXPLORED$ **then**
      $w \leftarrow$OPPOSITE($v, e$)
      **if** GETLABEL($w$) $= UNEXPLORED$ **then**
        SETLABEL($e, DISCOVERY$)
        TOPOLOGICALDFS($G, w$)
      **else**
        $e$ är korsande båge eller framåt-båge
  märk $v$ med topologiskt nummer $n$
  $n \leftarrow n - 1$

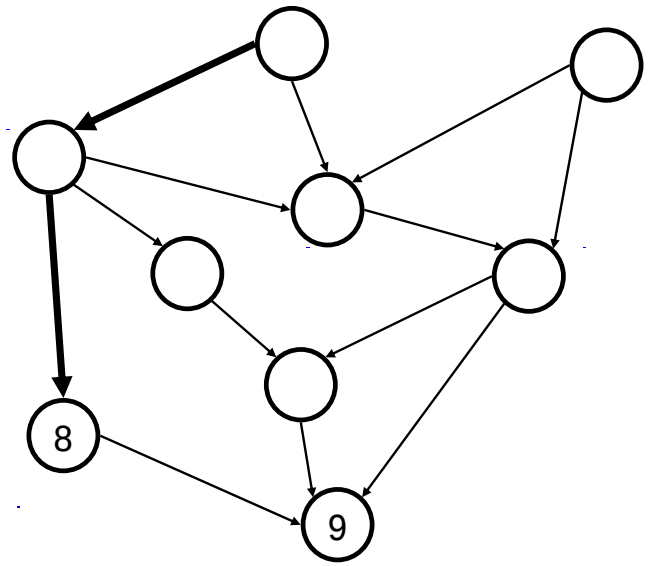
# Exempel: Topologisk sortering



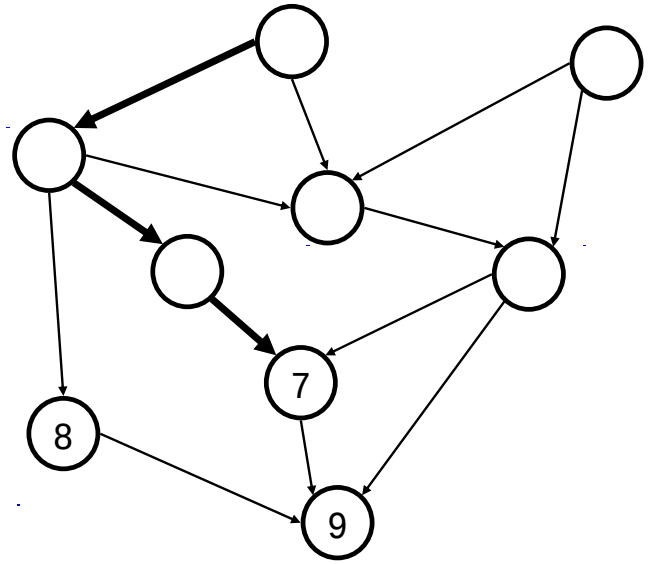
# Exempel: Topologisk sortering



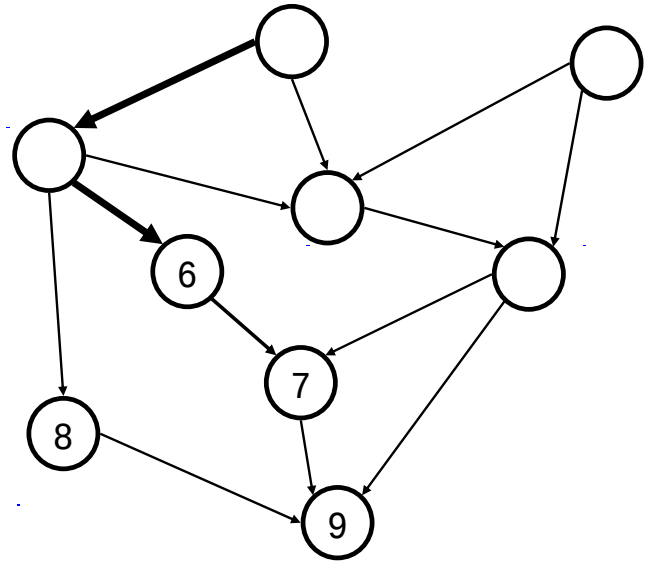
# Exempel: Topologisk sortering



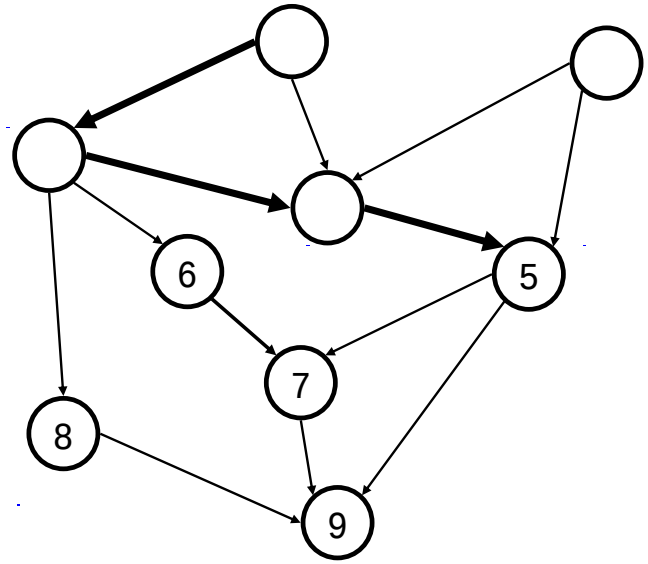
# Exempel: Topologisk sortering



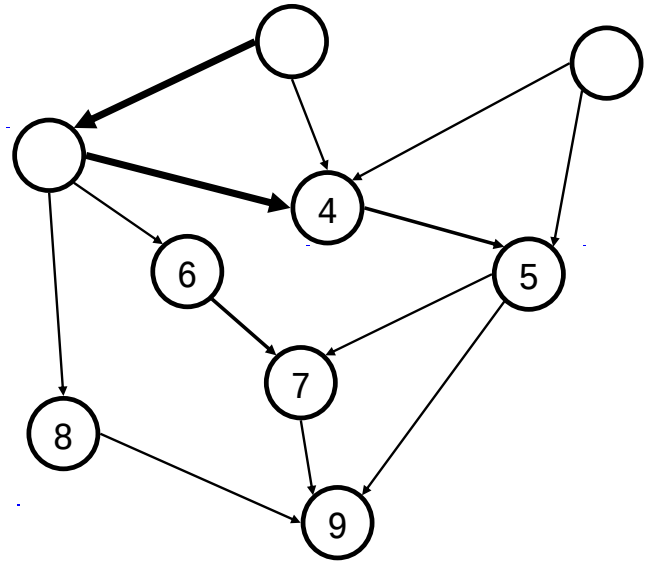
# Exempel: Topologisk sortering



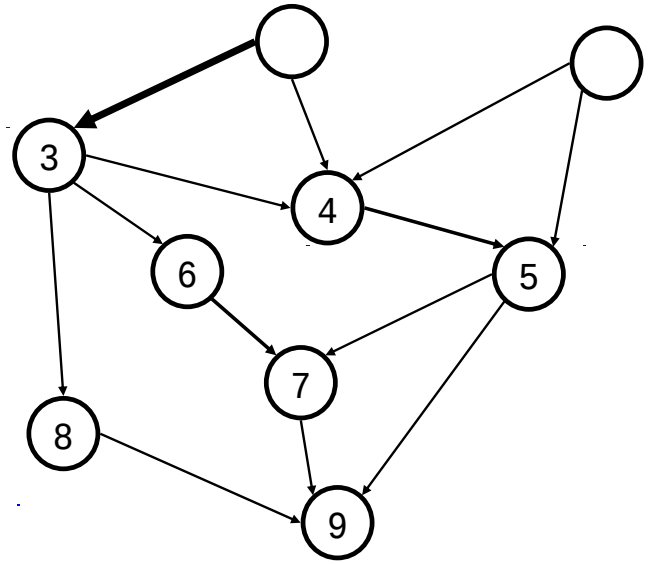
# Exempel: Topologisk sortering



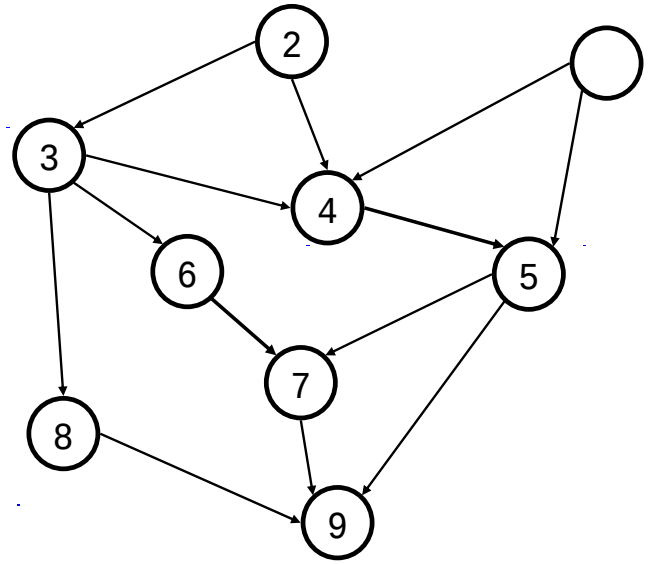
# Exempel: Topologisk sortering



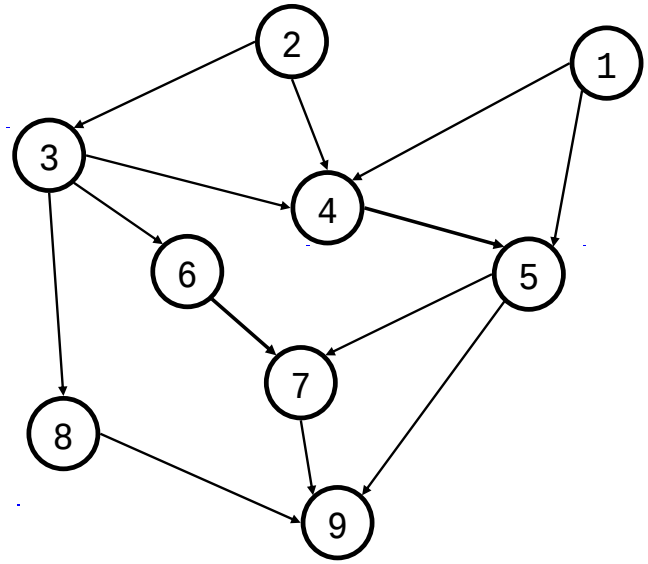
# Exempel: Topologisk sortering



# Exempel: Topologisk sortering



# Exempel: Topologisk sortering

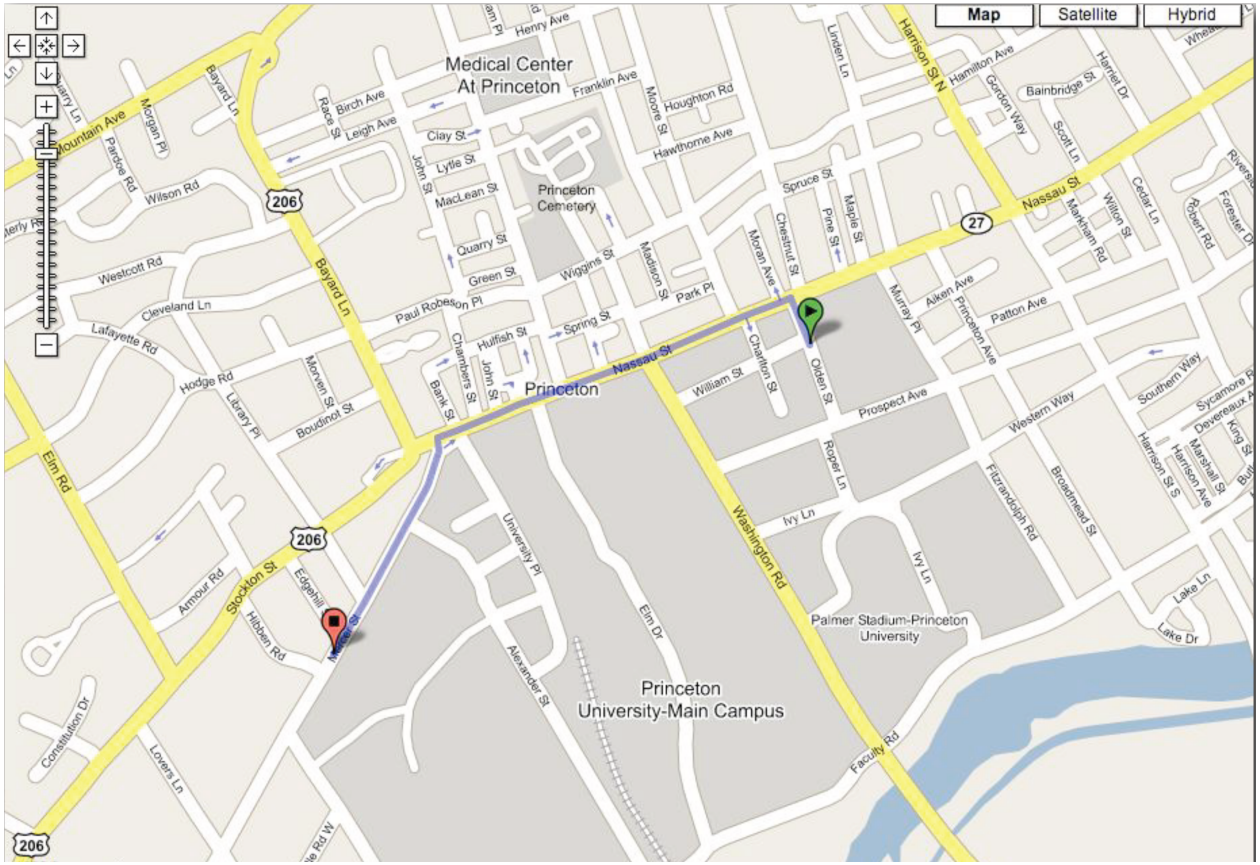


LINKÖPING UNIVERSITY

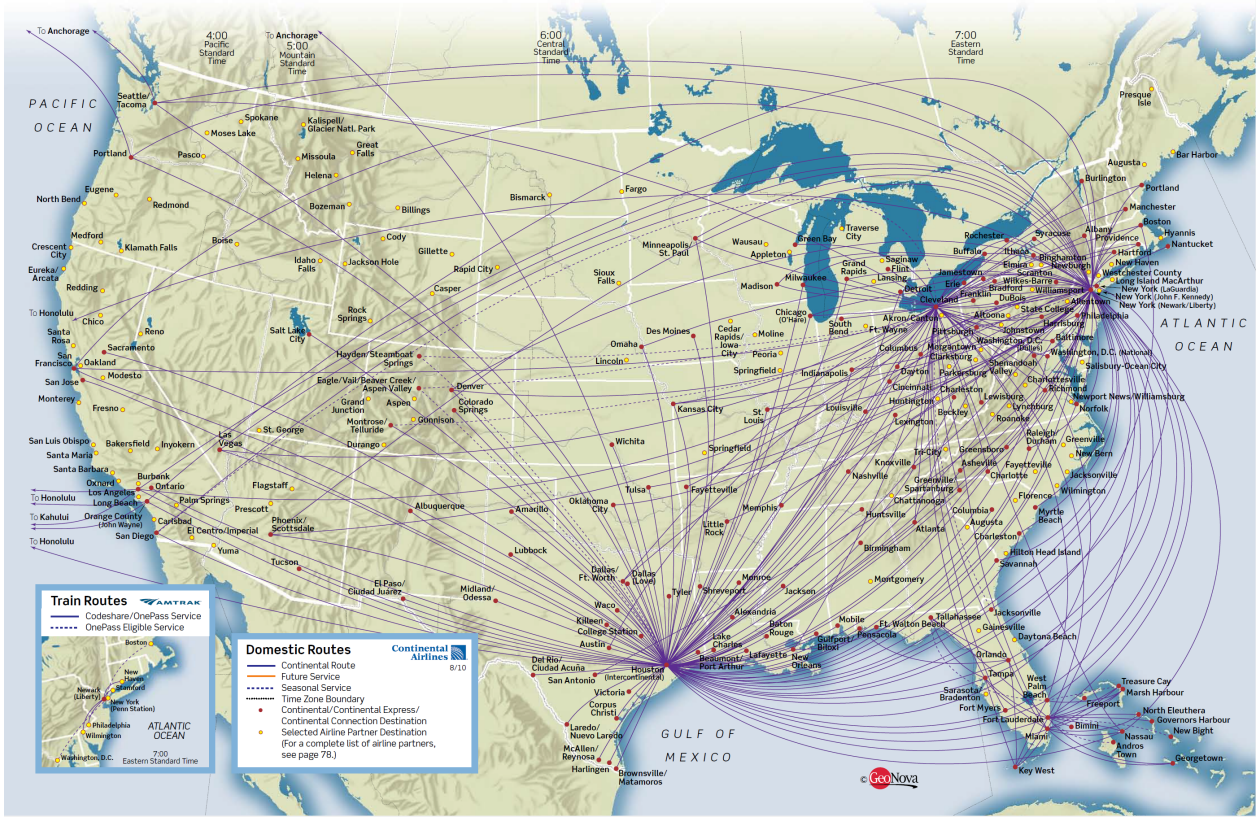
# Viktade grafer

- I en viktad graf är varje båge associerad med ett numeriskt värde kallat bågens *vikt*.
- Bågvikter kan representera avstånd, kostnader, etc.

# Google maps


Map
Satellite
Hybrid
Medical Center
At Princeton
Princeton
Cemetery
Princeton
University-Main Campus
Palmer Stadium-Princeton
University
Nassau St
Washington Rd
Bayard Ln
Elm Rd
Stockton St
Harrison St N
206
27

# Bolaget Continentals flygrutter i USA (augusti 2010)



# Tillämpningar

- PERT/CPM
- Kartapplikationer
- Seam carving
- Robotnavigering
- Texture mapping
- Typsättning i TeX
- Trafikplanering i stadsmiljö
- Optimal pipelining för VLSI-chip
- Schemaläggning av telemarketingförsäljare
- Routing av meddelanden inom telekom.
- Routingprotokoll för nätverk (OSPF, BGP, RIP)
- Utnyttja gynsamma situationer vid valutahandel
- Optimal ruttplanering för lastbilar givet trafikstockningsmönster

http://en.wikipedia.org/wiki/Seam_carving

Eng. *endpoints, incident, adjacent, degree, parallel, loops*

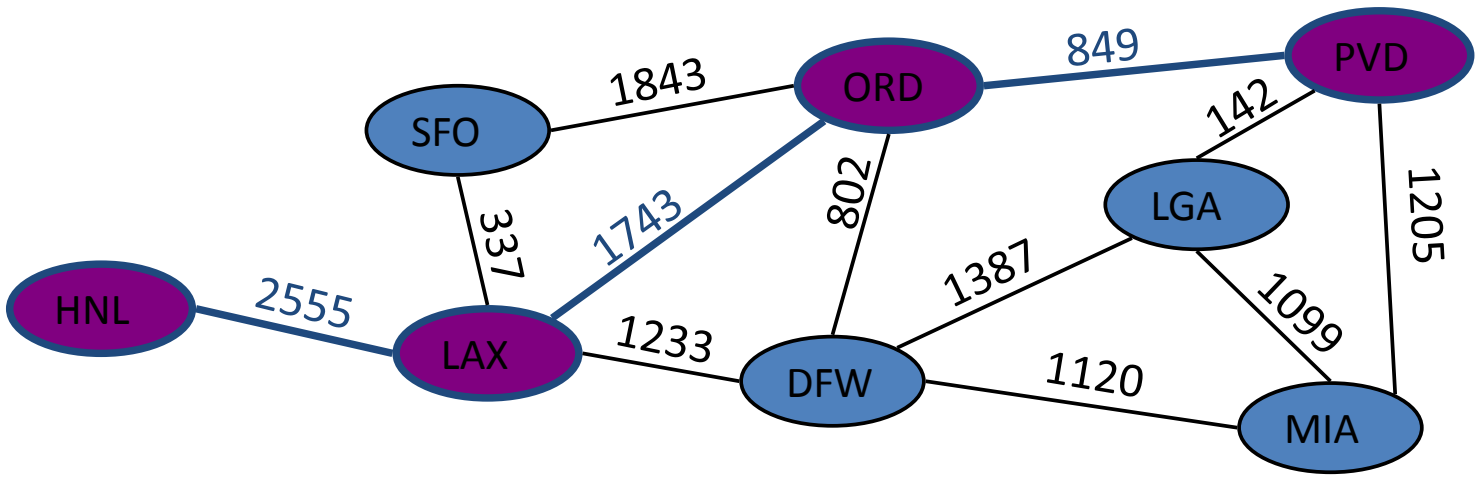
LINKÖPING UNIVERSITY

# Problemet kortaste väg

- Givet en viktad graf och två noder $u$ och $v$ vill vi hitta en stig mellan $u$ och $v$ med minimal total vikt.
  - Längden av en stig är summan av vikterna på stigens bågar

## Exempel

Kortaste vägen mellan Providence och Honolulu

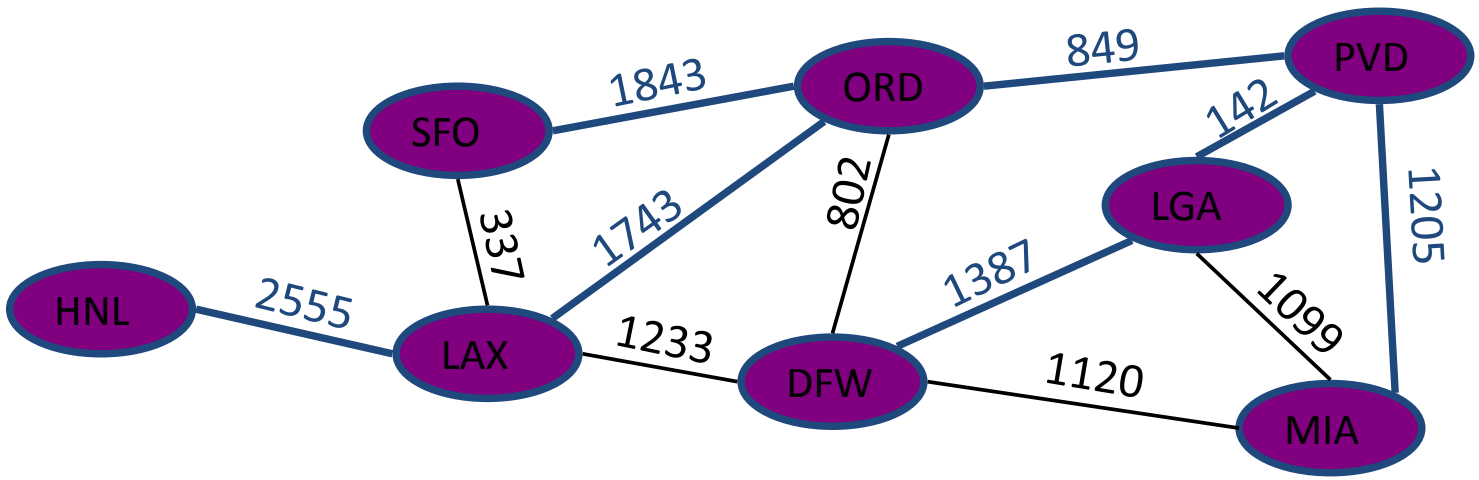


# Egenskaper hos kortaste vägar

- En delväg av en kortaste väg är också en kortaste väg
- Det finns ett träd av kortaste vägar från en startnod till alla andra noder

## Exempel

Ett träd av kortaste vägar från Providence

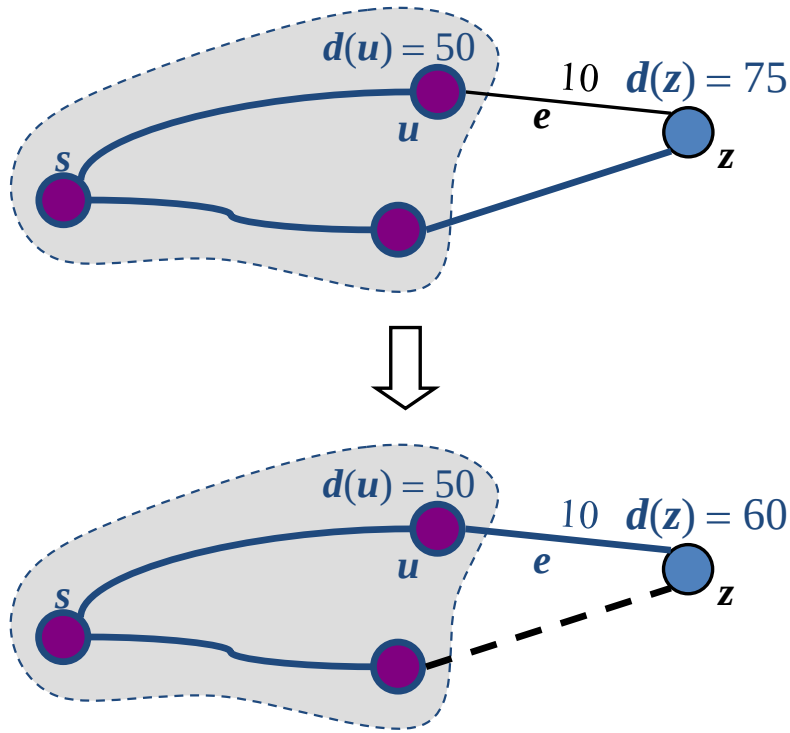


# Dijkstras algoritm

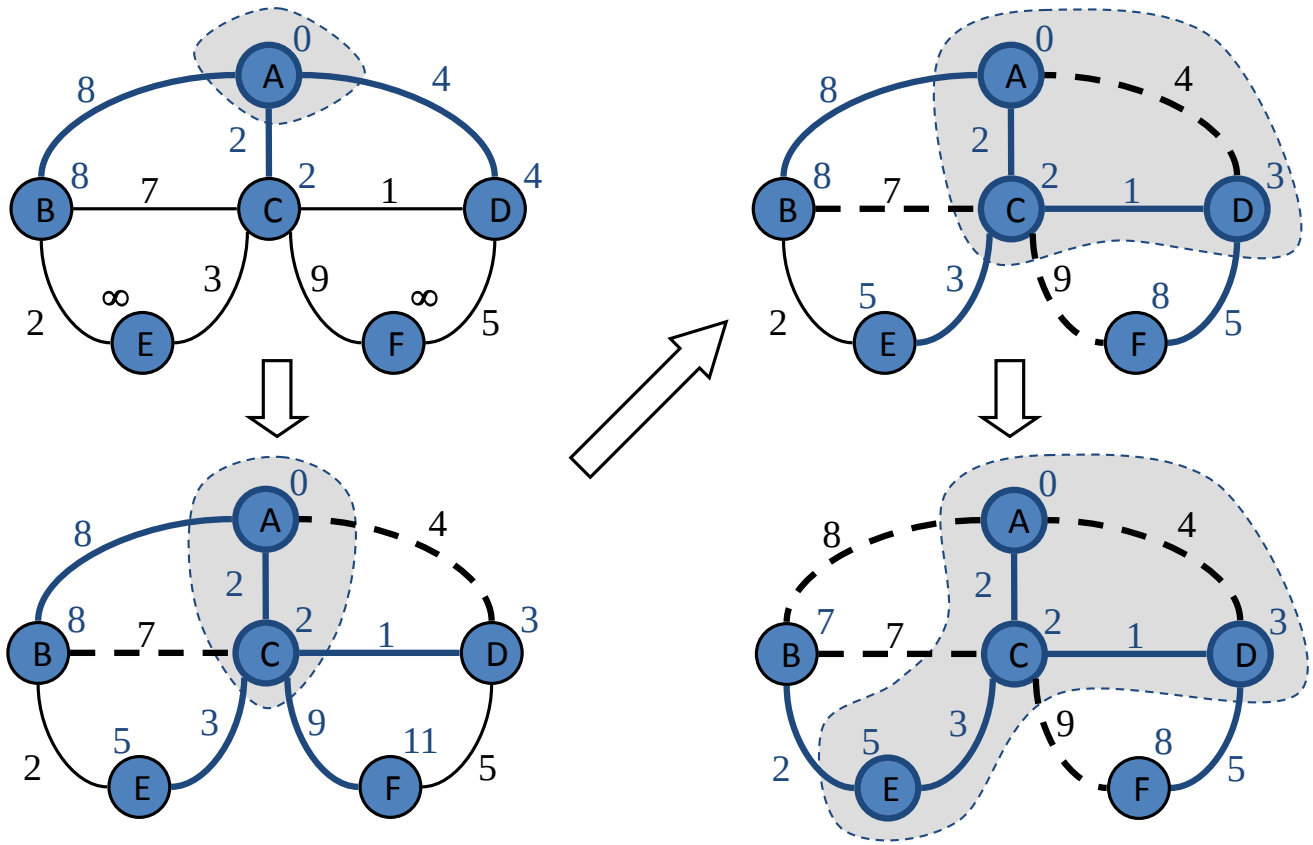
- Avståndet från en nod $v$ till en nod $s$ är längden av kortaste vägen mellan $s$ och $v$
- Dijkstras algoritm beräknar avstånden från en given startnod $s$ till alla noder $v$ i grafen
- Antaganden:
  - grafen är sammanhängande
  - bågarna är oriktade
  - grafen har inga öglor eller parallella bågar
  - bågvikterna är ickenegativa
- Vi bygger ett "moln" av noder med start i $s$, som till slut täcker alla noder
- Vi märker varje nod $v$ med $d(v)$, vilket betecknar avståndet mellan $v$ och $s$ i delgrafen bestående av molnet och noderna som är grannar till molnet
- I varje steg
  - lägger vi till den nod $u$ utanför molnet som har minst avståndsmärkning $d(u)$
  - uppdaterar vi märkningen av noderna som är grannar till $u$

# Utökningssteget

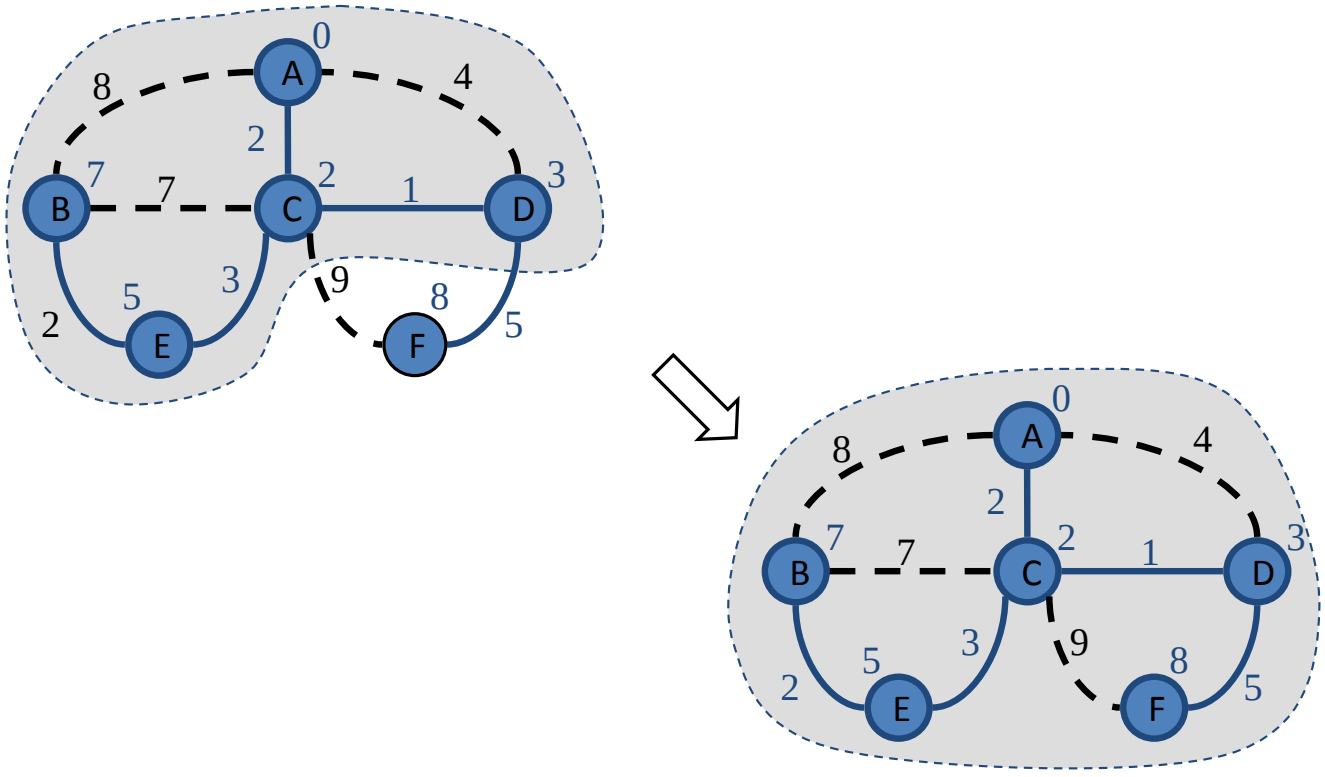
- Betrakta en båge $e = (u, z)$ sådan att
  - $u$ är noden vi nyligen lagt till i molnet
  - $z$ inte är med i molnet
- Relaxeringen av bågen $e$ uppdaterar $d(z)$ enligt:
  - $d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$



# Exempel



# Fortsättning på exemplet



# Dijkstras algoritm

- En prio-kö lagrar noderna utanför molnet
  - Nyckel: avstånd
  - Värde: nod
- Lokator-baserade metoder
  - insert$(k, v)$ returnerar en lokator
  - replaceKey$(l, k)$ ändrar en posts nyckelvärde
- Vi lagrar två saker i varje nod:
  - avstånd ($d(v)$)
  - lokator i prio-kön

```
procedure DIJKSTRA(G, s)
    Q ← ny tom heapbaserad prio-kö
    for all v ∈ G.VERTICES() do
        if v = s then
            SETDISTANCE(v, 0)
        else
            SETDISTANCE(v, ∞)
        l ← Q.INSERT(GETDISTANCE(v), v)
        SETLOCATOR(v, l)
    while ¬Q.ISEMPTY() do
        u ← Q.REMOVEMIN()
        for all e ∈ G.INCIDENTEDGES(u) do
            z ← G.OPPOSITE(u, e)
            r ←GETDISTANCE(u)+WEIGHT(e)
            if r <GETDISTANCE(z) then
                SETDISTANCE(z, r)
                Q.REPLACEKEY(GETLOCATOR(z), r)
```

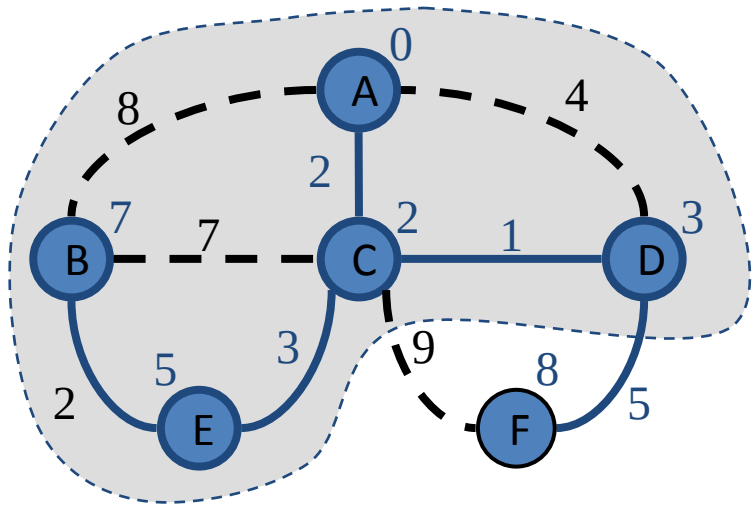
# Analys av Dijkstras algoritm

- Grafoperationer
  - Vi anropar incidentEdges en gång för varje nod
- Märkningsoperationer
  - Vi hämtar/sätter avstånd och lokator för nod $z$ $O(deg(z))$ gånger
  - Att sätta/hämta en märkning tar tid $O(1)$
- Prio-köoperationer
  - Varje nod sätts in en gång och tas ut en gång från prio-kön, där varje insättning och borttagning tar tid $O(\log n)$
  - En nods nyckel i prio-kön ändras som mest $deg(w)$ gånger, där varje nyckeländring tar tid $O(\log n)$
- Dijkstras algoritm har exekveringstid $O((n+m)\log n)$ givet att grafen representeras med en grannlista
  - Kom ihåg att $\sum_v deg(v) = 2m$
- Exekveringstiden kan också uttryckas som $O(m \log n)$ eftersom vi antagit att grafen är sammanhängande

# Varför Dijkstras algoritm fungerar

Dijkstras algoritm är baserad på den giriga metoden. Algoritmen lägger till noderna efter ökande avstånd.

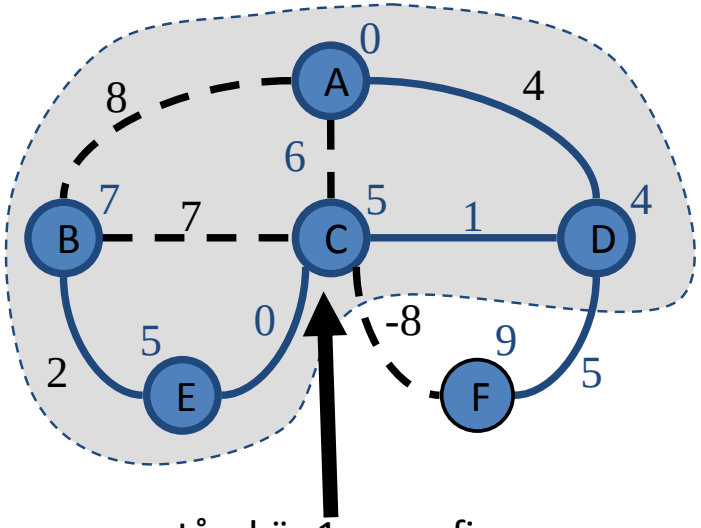
- Antag att algoritmen inte hittade alla kortaste avstånd. Låt $F$ vara den första felaktiga noden som behandlas.
- När den föregående noden, $D$, längs den sanna kortaste vägen behandlades var dess avstånd korrekt.
- Men bågen $(D, F)$ relaxerades i det steget!
- Mao, så länge $d(F) \geq d(D)$ kan inte avståndet till $F$ bli fel. Dvs, ingen nod får fel avstånd.



# Varför Dijkstras algoritm inte fungerar med negativa bågvikter

Dijkstras algoritm är baserad på den giriga metoden. Algoritmen lägger till noderna efter ökande avstånd.

- Om en nod med en negativ incident båge skulle läggas till sent i molnet skulle den förstöra avståndet till noder som redan finns i molnet.




C's sanna avstånd är 1, men finns redan i molnet med d(C)=5!

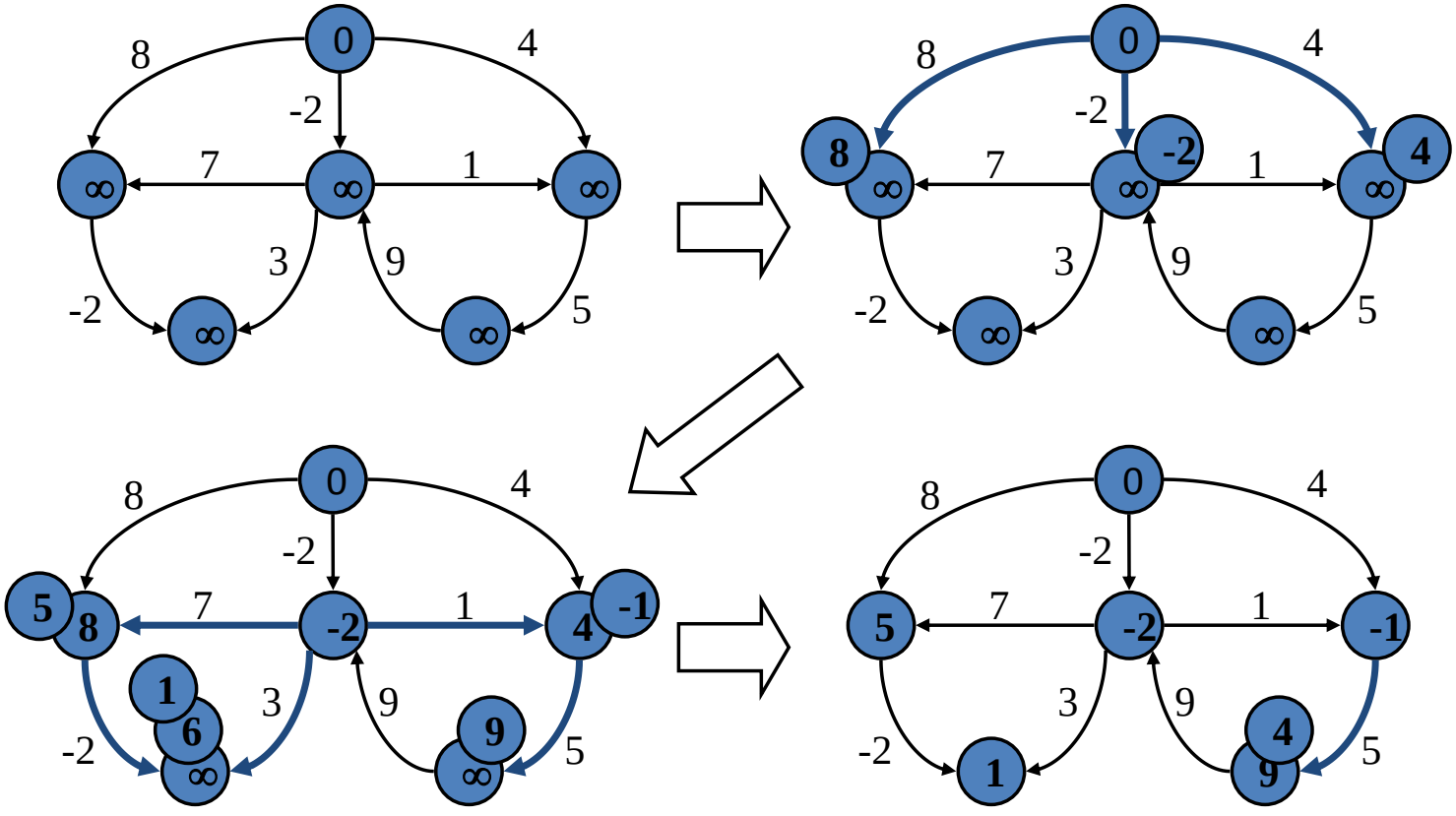
# Bellman-Fords algoritm (finns inte i kursboken)

- Fungerar även med negativa bågvikter
- Måste anta riktade bågar (annars kan det finnas cykler med negativ vikt)
- Iteration $i$ hittar alla kortaste vägar som använder $i$ bågar
- Exekveringstid: $O(nm)$
- Kan utökas till att detektera cykler med negativ vikt

```
procedure BellmanFord(G, s)
    for v ∈ G.vertices() do
        if v = s then
            setDistance(v, 0)
        else
            setDistance(v, ∞)
    for i ← 1 to n − 1 do
        for each e ∈ G.edges() do
            u ← G.origin(e)
            z ← G.opposite(u, e)
            r ←getDistance(u)+weight(e)
            if r <getDistance(z) then
                setDistance(z, r)
```

# Exempel

Noderna är märkta med resp d(v)-värde



# Nästa gång:
# Tentaföreläsning

www.liu.se

LINKÖPING UNIVERSITY