Föreläsning 10

Riktade grafer. Viktade grafer.

TDDC91,TDDE22,725G97: DALG

Utskriftsversion av föreläsning i *Datastrukturer och algoritmer* 9 oktober 2018

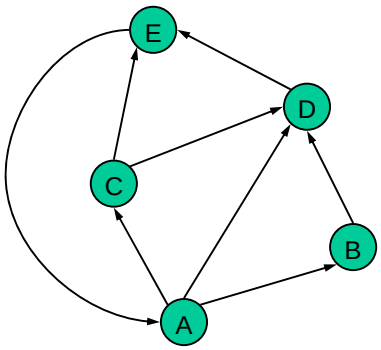
Magnus Nielsen, IDA, Linköpings universitet

10.1

1 Riktade grafer

Introduktion

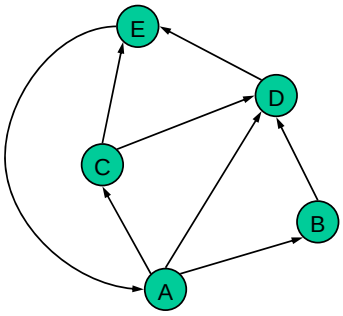
- I en riktad graf är alla bågar riktade



10.2

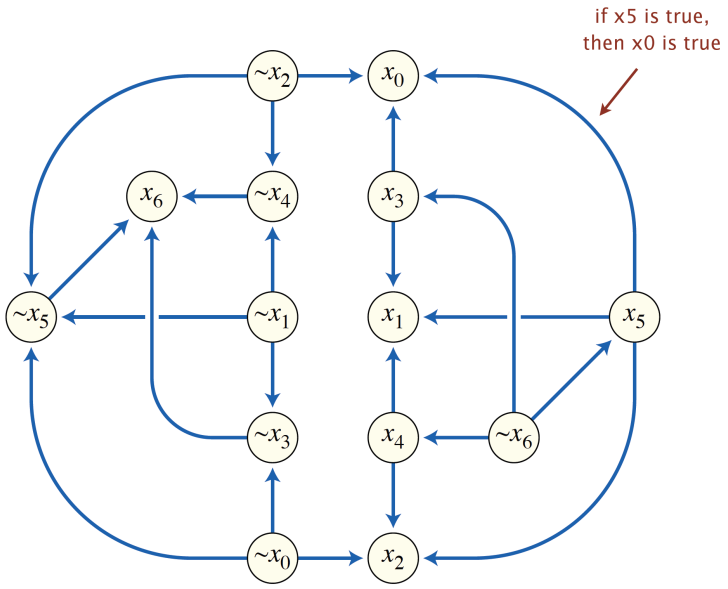
Egenskaper

- 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) 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.



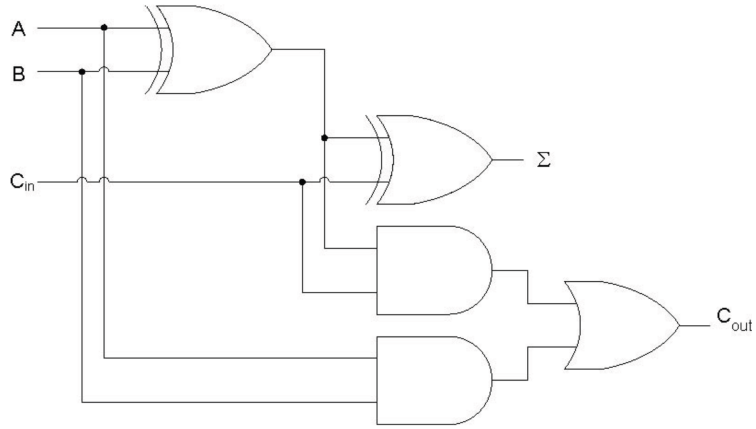
10.3

Implikationsgraf



10.4

Kombinatorisk krets



10.5

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

10.6

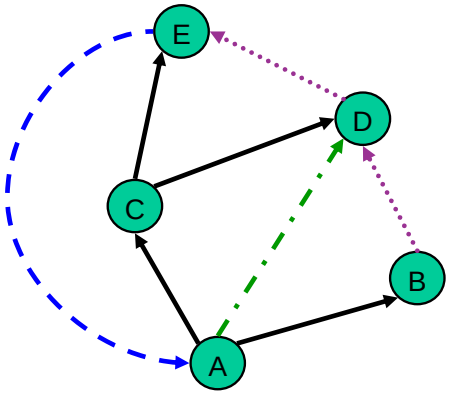
Några algoritmiska grafproblem

- 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?

10.7

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 s bestämmer vilka noder som är nåbara från s

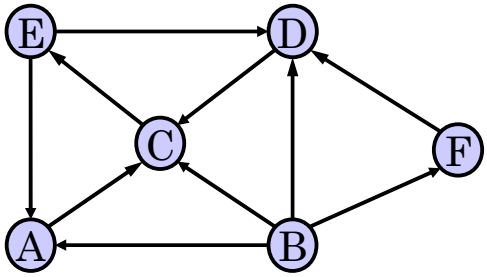


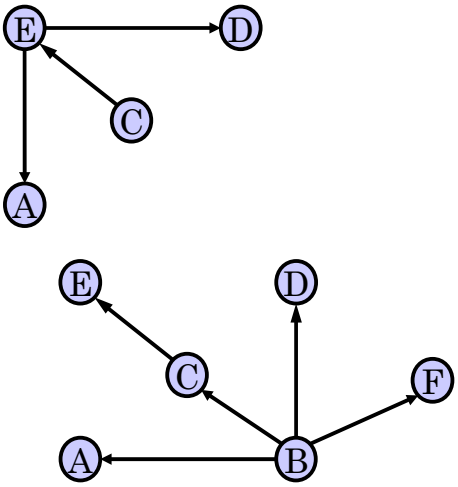
10.8

2 Konnektivitet

Nåbarhet

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

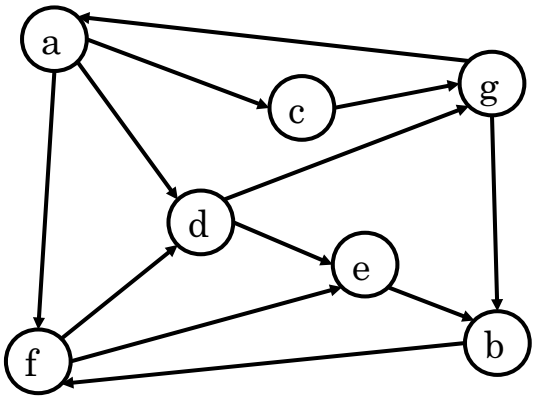




10.9

Starkt sammanhängande

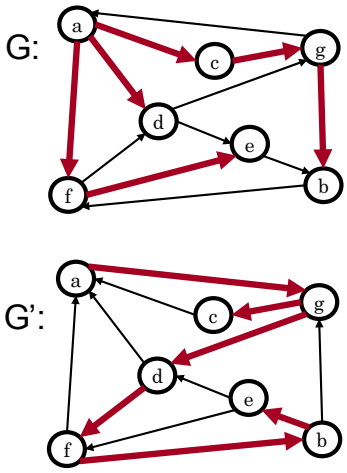
Varje nod är nåbar från alla andra noder



10.10

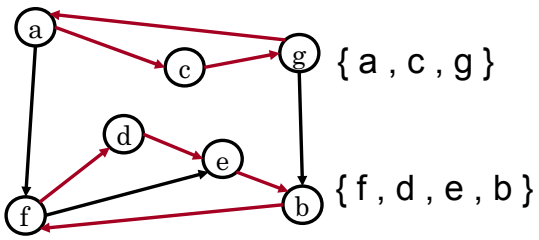
Algoritm för att avgöra starkt sammanhängande

- Välj en nod v i G
- *// Kan alla noder nås från v?*Utför DFS från v i G
  - Om det finns 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 w som inte besöks svara ”nej”
  - Annars, svara ”ja”
- Körtid: $O(n+m)$



Starkt sammanhängande komponenter

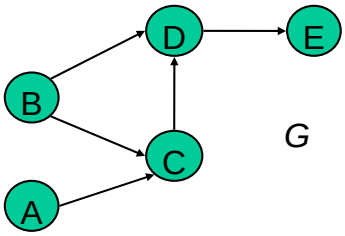
- Maximal delgraf sådan att varje nod kan nå alla andra noder i delgrafen
- Kan också göras i $O(n+m)$ tid genom att använda DFS i flera steg

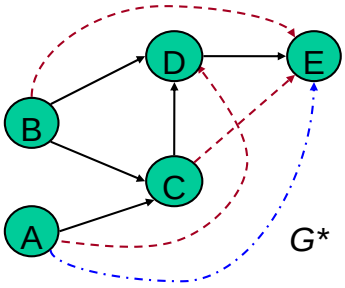


3 Transitivt hölje

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





10.13

Beräkning av transitiva höljet

- Vi kan köra DFS med start i varje nod $v_1, \ldots, v_n$, alltså $O(n \cdot (n+m))$

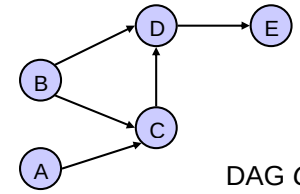
10.14

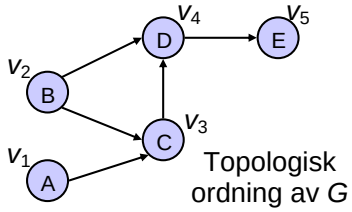
4 Topologisk sortering

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

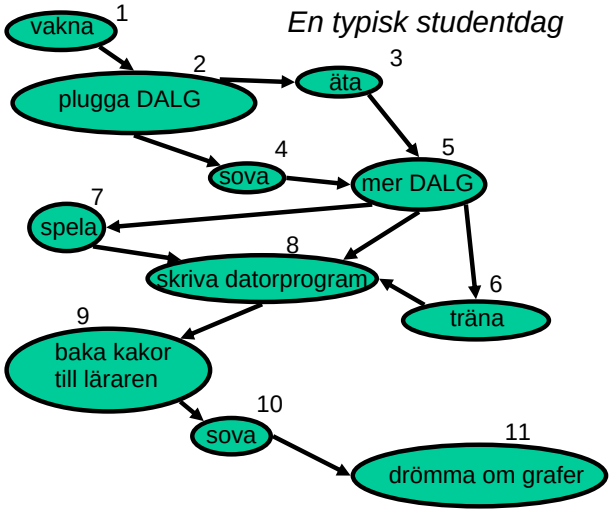




10.15

Topologisk sortering

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



10.16

Algoritm för topologisk sortering

procedure TOPOLOGICALSORT(G)
  $S \leftarrow$ ny tom stack
  for all $u \in G$.VERTICES() **do**
    låt INCOUNTER(u) vara ingraden för u
    if INCOUNTER(u) $= 0$ **then**
      S.PUSH(u)
  $i \leftarrow 1$
  while $\neg S$.ISEMPTY() **do**
    $u \leftarrow S$.POP()
    låt u få nummer i i den topologiska ordningen
    $i \leftarrow i+1$
    for all utgående kanter (u,w) från u **do**
      INCOUNTER(w) $\leftarrow$ INCOUNTER(w) -1
      if INCOUNTER(w) $= 0$ **then**
        S.PUSH(w)

Körtid: $O(n+m)$.

10.17

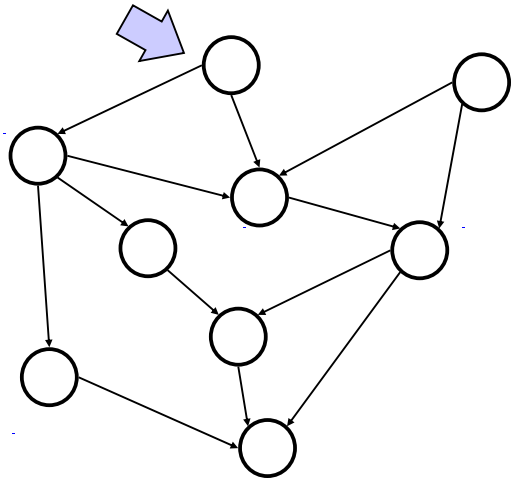
Alternativ algoritm för topologisk sortering

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...?

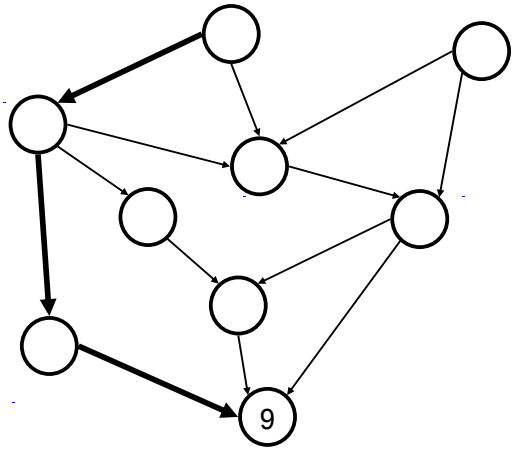
10.18

Exempel: Topologisk sortering



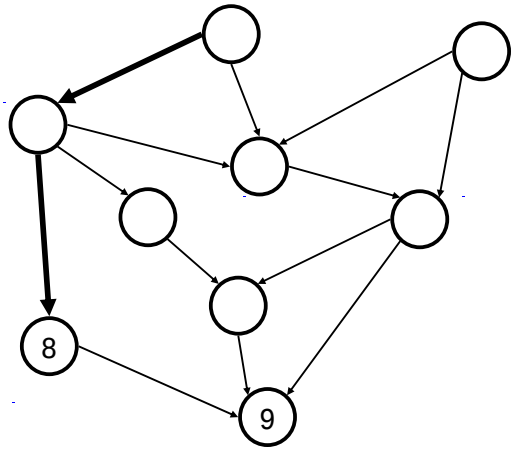
10.19

Exempel: Topologisk sortering



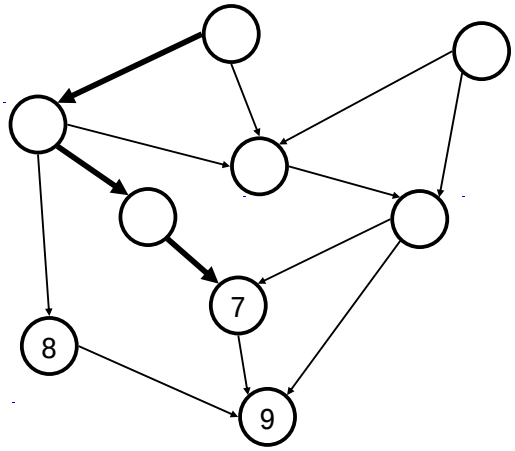
10.20

Exempel: Topologisk sortering



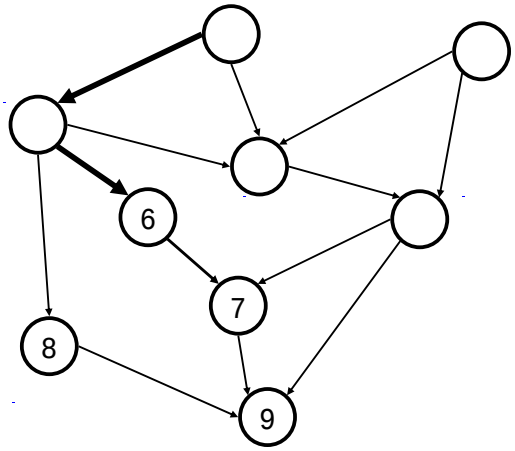
10.21

Exempel: Topologisk sortering

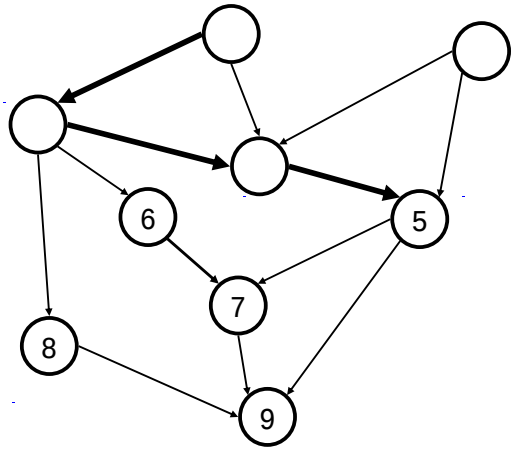


10.22

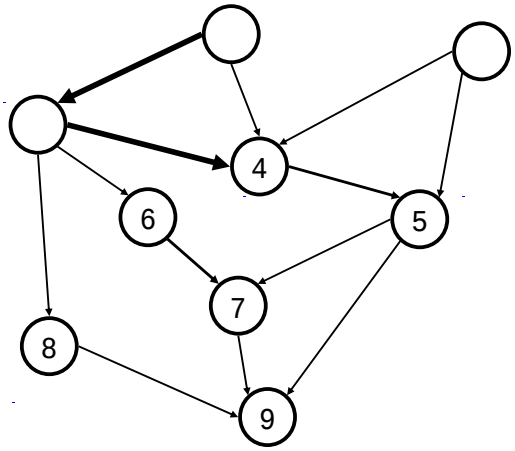
Exempel: Topologisk sortering



Exempel: Topologisk sortering

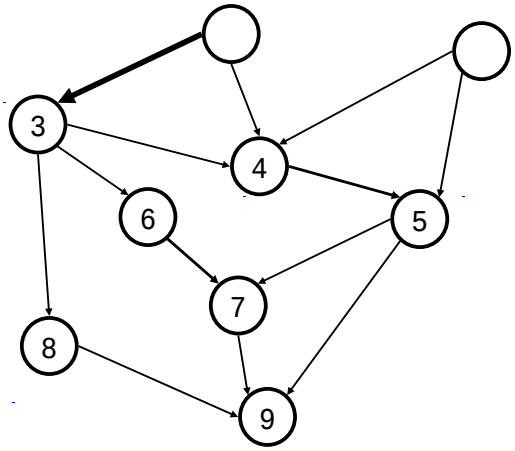


Exempel: Topologisk sortering



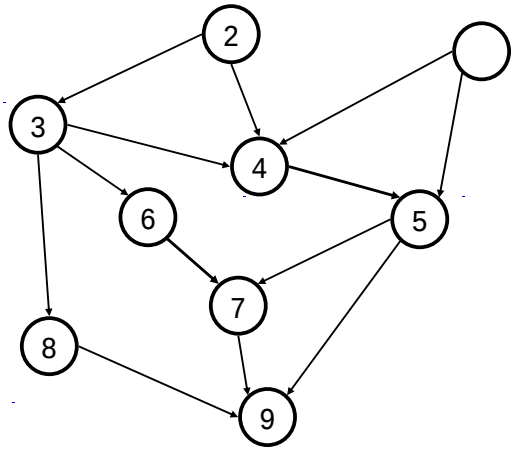
10.25

Exempel: Topologisk sortering



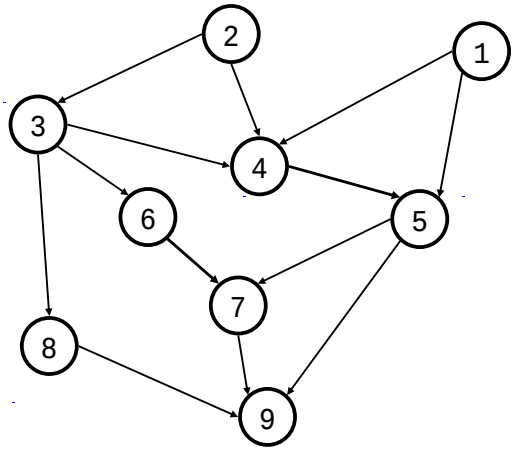
10.26

Exempel: Topologisk sortering



10.27

Exempel: Topologisk sortering



10.28

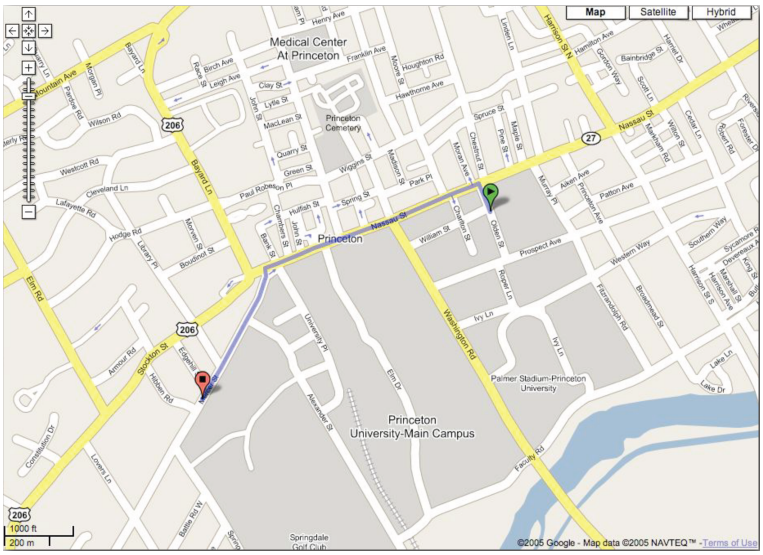
5 Viktade grafer

Viktade grafer

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

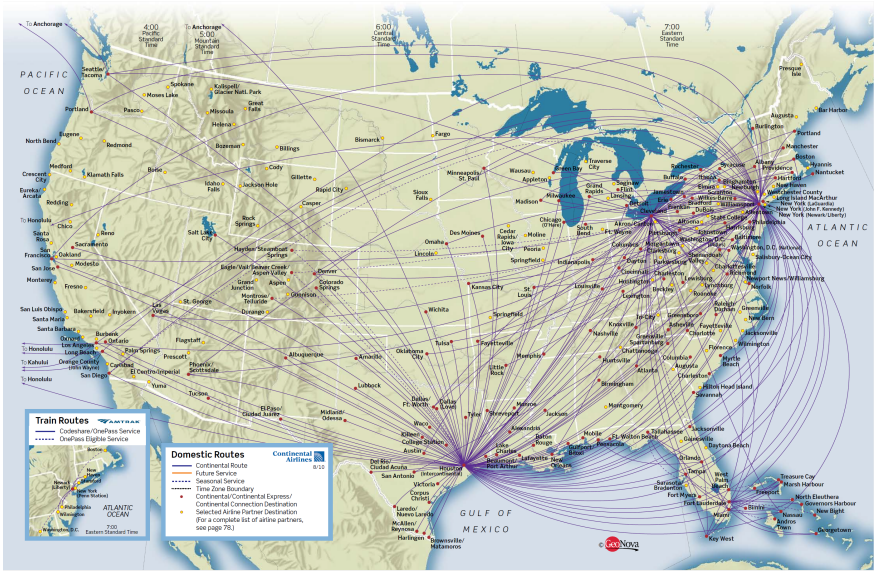
10.29

Google maps



10.30

Bolaget Continentals flygrutter i USA (augusti 2010)



10.31

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

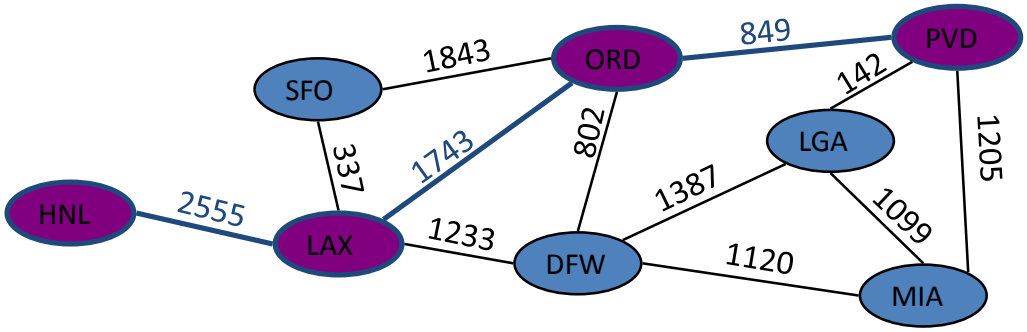
Eng. *endpoints, incident, adjacent, degree, parallel, loops* 10.32

6 Kortaste vägar

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

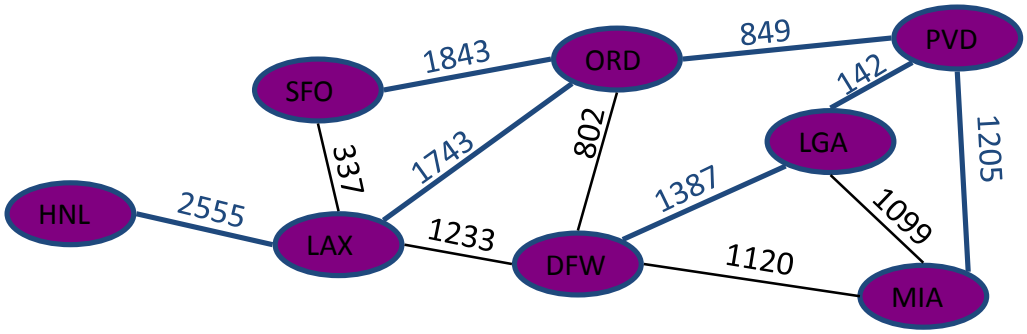


10.33

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



10.34

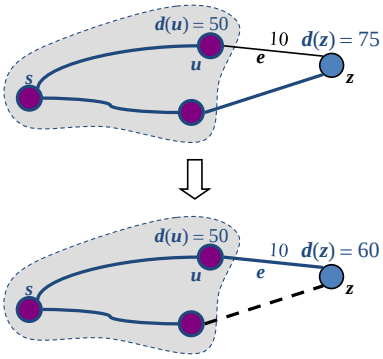
Dijkstras algoritm

- 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

10.35

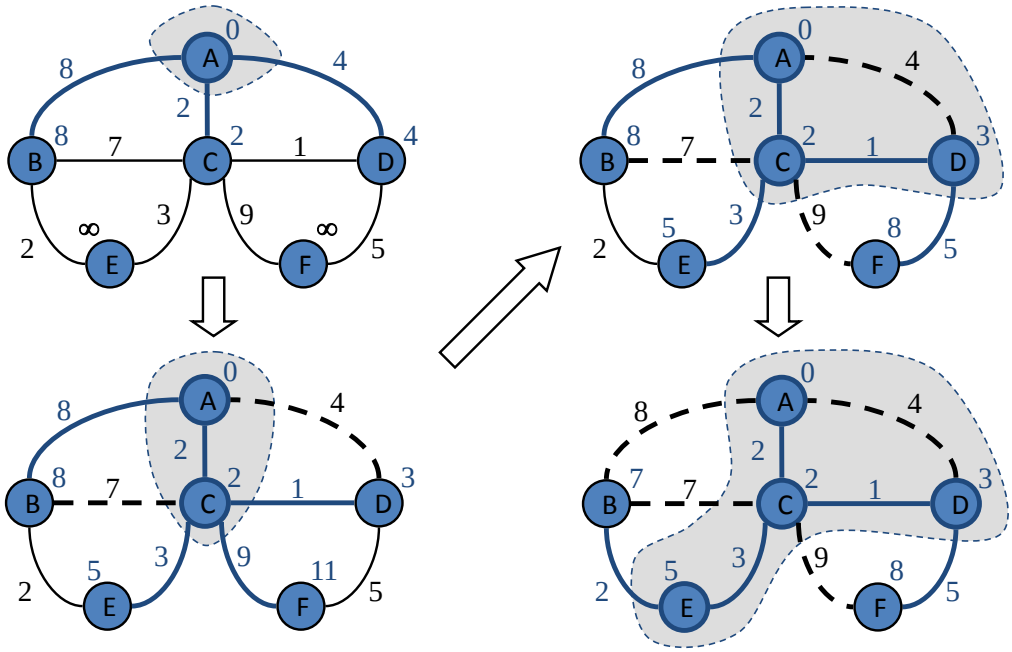
Utökningssteget

- 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:
  - $d(z) \leftarrow \min\{d(z), d(u) + weight(e)\}$



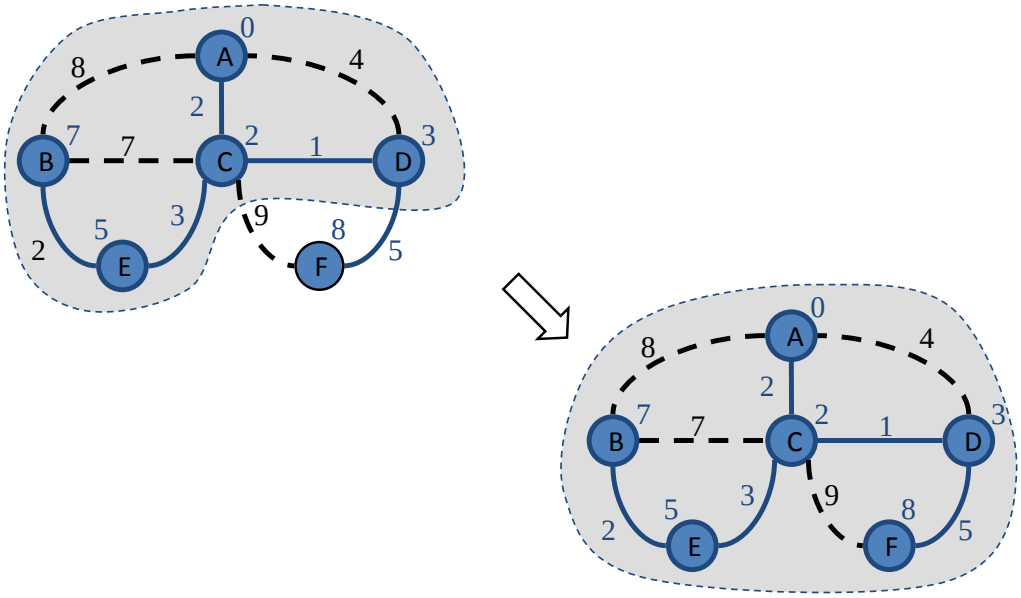
10.36

Exempel



10.37

Fortsättning på exemplet



10.38

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)
  $Q \leftarrow$ ny tom heapbaserad prio-kö
  for all $v \in G$.VERTICES() **do**
    if $v = s$ **then**
      SETDISTANCE$(v,0)$
    else
      SETDISTANCE(v,∞)
    $l \leftarrow Q$.INSERT(GETDISTANCE$(v),v)$
    SETLOCATOR(v,l)
  while $\neg Q$.ISEMPTY() **do**
    $u \leftarrow Q$.REMOVEMIN()
    for all $e \in G$.INCIDENTEDGES(u) **do**
      $z \leftarrow G$.OPPOSITE(u,e)
      $r \leftarrow$GETDISTANCE$(u)+$WEIGHT(e)
      if $r <$GETDISTANCE(z) **then**
        SETDISTANCE(z,r)
        Q.REPLACEKEY(GETLOCATOR$(z),r)$

10.39

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

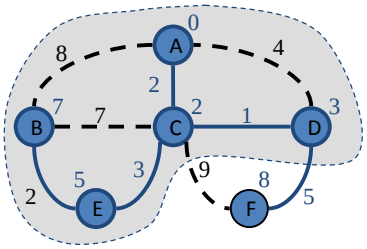
  - Varje nod sätts in en gång och tas ut en gång från prio-kön, där varje insättning och borttagning tar tid $O(\log n)$
  - En nods nyckel i prio-kön ändras som mest $deg(w)$ gånger, där varje nyckeländring tar tid $O(\log n)$
- Dijkstras algoritm har exekveringstid $O((n+m)\log n)$ givet att grafen 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

10.40

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.

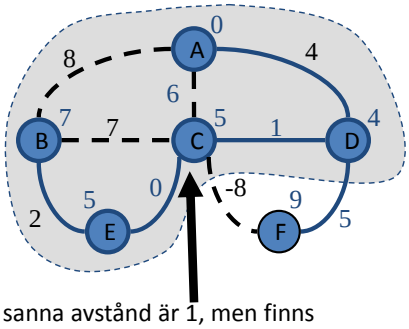


10.41

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!

10.42

Bellman-Fords algoritm (finns inte i kursboken)

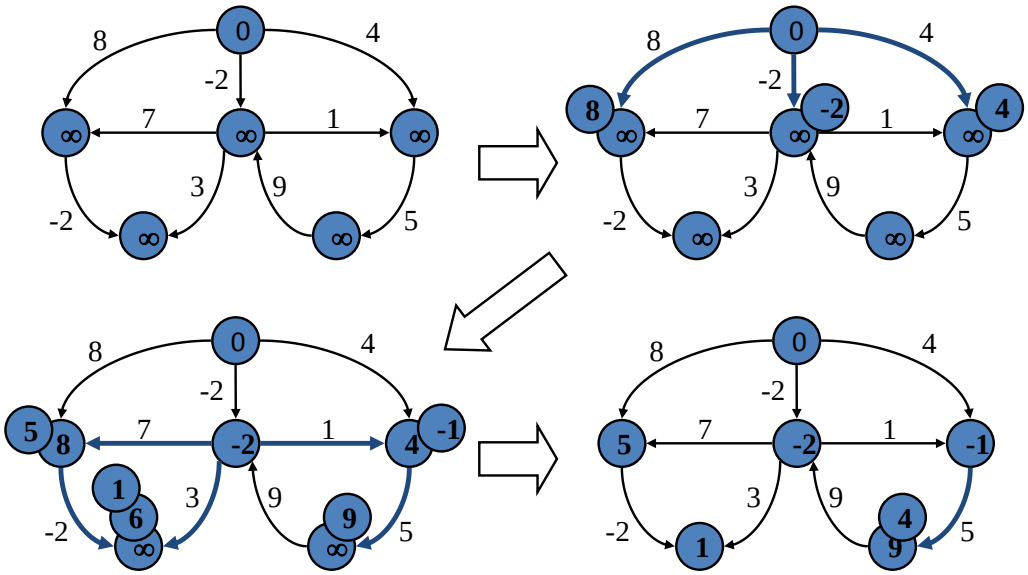
- Fungerar även med negativa bågvikter
- Måste anta riktade bågar (annars kan det finnas cykler med negativ vikt)
- Iteration i hittar alla kortaste vägar som använder i bågar
- Exekveringstid: $O(nm)$
- Kan utökas till att detektera cykler med negativ vikt

```
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)
```

10.43

Exempel

Noderna är märkta med resp d(v)-värde



10.44