

# Grafer och grafsökning

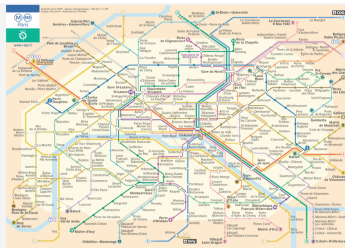
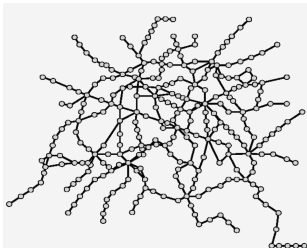
TDDE22, 725G97: DALG

Magnus Nielsen

- 1 Grafer
  - Introduktion
  - ADT graf
  - Datastrukturer
- 2 Sökning i oriktade grafer
  - DFS
  - BFS
  - DFS vs BFS

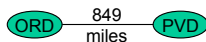
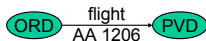
# Definition

- En graf är ett par  $(V, E)$ , där
  - $V$  är en mängd noder (eller hörn)
  - $E$  är en mängd av par av noder kallade bågar (eller kanter)
  - Noder och bågar är positioner och kan lagras element
  - Eng. *vertices*, *edges*



# Bågtyper

- **Riktad** båge
  - ordnat par av noder  $(u, v)$
  - $u$  är startnoden,  $v$  är slutnoden
- **Oriktad** båge
  - oordnat par av noder  $\{u, v\}$
- I en riktad graf är alla bågar riktade
- I en oriktad graf är alla bågar oriktade



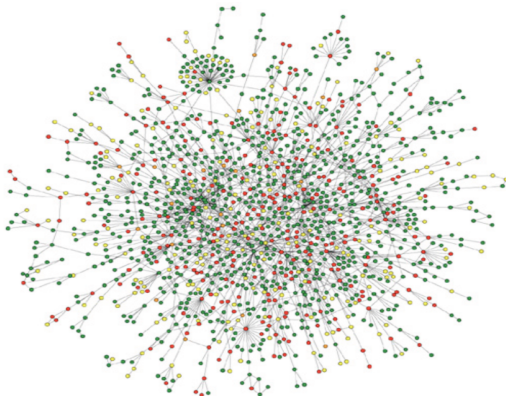
# Varför skall man studera grafalgoritmer?

- Tusentals praktiska tillämpningar
- Hundratals kända grafalgoritmer
- Intressant abstraktion med stor tillämpbarhet
- Gren av datalogi och diskret matematik med många utmaningar

## Exempel på praktiska tillämpningar

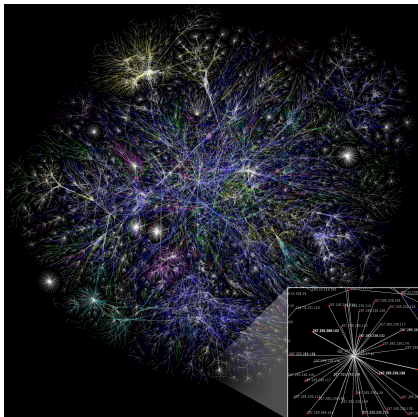
- Flödesproblem (schemaläggning, ex: flyg/tåg/busstrafik ...)
- Kortaste väg (ex: kartor, spel (AI), ...)
- Graffärgning, lösa pussel (ex: Sudoku)
- PageRank (Sökmotorer)
- ...

# Protein till protein-interaktionsnätverk



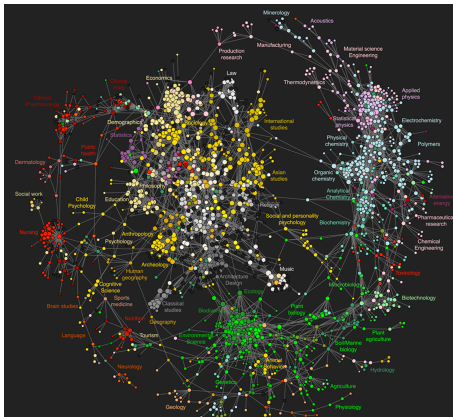
Jeong et al. Nature Review | Genetics

# Internet kartlagt av Opte-projektet





# Vetenskapliga klickströmmar

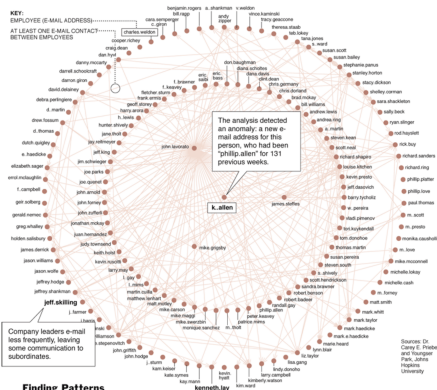


# 10 miljoner Facebook-vänner

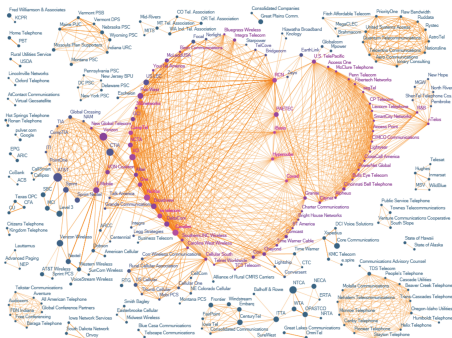


"Visualizing Friendship" av Paul Butler

# En veckas mejl inom Enron



# Utvecklingen hos lobbyingsamarbeten inom FCC



"The Evolution of FCC Lobbying Coalitions" av Pierre de Vries från JoSS Visualization Symposium 2010

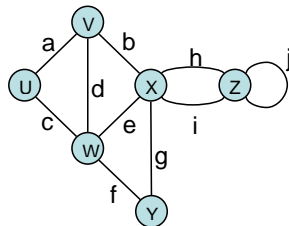
# Tillämpningar

graf	nod	båge
kommunikation	telefon, dator	fiberoptisk kabel
krets	grind, register, processor	koppling
mekanisk	led	stag, bjälke, fjäder
finansiell	aktie, valuta	transaktion
transport	gatukorsning, flygplats	väg, flygrutt
internet	klass C-nät	förbindelse
brädspel	pjäsernas positioner	giltigt drag
socialt nätverk	person, aktör	vänskap, relation
neuralt nätverk	neuron	synaps
proteinnätverk	proteïn	protein till protein-interaktion
kemisk sammansättning	molekyl	bindning

# Terminologi

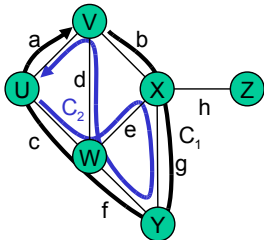
- En båge har **ändpunkter** ( $a$  har ändpunkterna  $U$  och  $V$ )
- Bågar som slutar i en nod  $n$  sägs vara **incidenta** ( $a$ ,  $d$  och  $b$  är incidenta till  $V$ )
- Noder kan vara **grannar** ( $U$  och  $V$  är grannar)
- Noder har **grad** ( $X$  har grad 5)
- **Parallella** bågar ( $h$  och  $i$  är parallella bågar)
- **Öglor** ( $j$  är en ögla)

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



## Mer terminologi

- En **cykel** är en cirkulär sekvens av alternerande noder och bågar. Varje båge föregås och efterföljs av sina ändpunkter.
- En **enkel cykel** är en cykel sådan att alla dess noder och bågar är distinkta.
- $C_1 = (V, b, X, g, Y, f, W, c, U, a, V)$  är en enkel cykel.
- $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a, U)$  är *inte* en enkel cykel.



Eng. *cycle, simple cycle*

# Egenskaper

## Egenskap 1

$$\sum_v \text{deg}(v) = 2m$$

Bevis: Varje båge räknas två gånger

## Egenskap 2

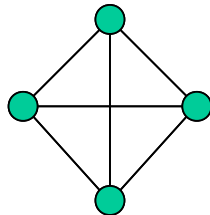
I en oriktad graf utan öglor och parallella bågar gäller

$$m \leq n(n-1)/2$$

Bevis: Varje nod har max grad  $(n-1)$

## Notation

- $n$  antalet noder
- $m$  antalet bågar
- $\text{deg}(v)$  är nod  $v$ 's grad



Exempel  
 $n = 4$   
 $m = 6$   
 $\text{deg}(v) = 3$



# Några algoritmiska grafproblem

- **Stig**. Finns det en stig mellan  $s$  och  $t$ ?
- **Kortaste väg**. Vilken är den kortaste stigen mellan  $s$  och  $t$ ?

## Några algoritmiska grafproblem

- **Stig**. Finns det en stig mellan  $s$  och  $t$ ?
- **Kortaste väg**. Vilken är den kortaste stigen mellan  $s$  och  $t$ ?
- **Cykel**. Finns det en cykel i grafen?
- **Eulertur**. Finns det en cykel som använder varje båge exakt en gång?
- **Hamiltoncykel**. Finns det en cykel som använder varje nod exakt en gång?

## Några algoritmiska grafproblem

- **Stig**. Finns det en stig mellan  $s$  och  $t$ ?
- **Kortaste väg**. Vilken är den kortaste stigen mellan  $s$  och  $t$ ?
- **Cykel**. Finns det en cykel i grafen?
- **Eulertur**. Finns det en cykel som använder varje båge exakt en gång?
- **Hamiltoncykel**. Finns det en cykel som använder varje nod exakt en gång?
- **Konnektivitet**. Finns det en förbindelse mellan alla noder?
- **Bikonnektivitet**. Finns det en nod som gör att grafen inte hänger samman om man tar bort den?

## Några algoritmiska grafproblem

- **Stig**. Finns det en stig mellan  $s$  och  $t$ ?
- **Kortaste väg**. Vilken är den kortaste stigen mellan  $s$  och  $t$ ?
- **Cykel**. Finns det en cykel i grafen?
- **Eulertur**. Finns det en cykel som använder varje båge exakt en gång?
- **Hamiltoncykel**. Finns det en cykel som använder varje nod exakt en gång?
- **Konnektivitet**. Finns det en förbindelse mellan alla noder?
- **Bikonnektivitet**. Finns det en nod som gör att grafen inte hänger samman om man tar bort den?
- **Planaritet**. Går det att rita grafen utan att några bågar korsar varandra?
- **Grafisomorfi**. Är två grafer identiska bortsett från namnen på noderna?

## Några algoritmiska grafproblem

- **Stig**. Finns det en stig mellan  $s$  och  $t$ ?
- **Kortaste väg**. Vilken är den kortaste stigen mellan  $s$  och  $t$ ?
- **Cykel**. Finns det en cykel i grafen?
- **Eulertur**. Finns det en cykel som använder varje båge exakt en gång?
- **Hamiltoncykel**. Finns det en cykel som använder varje nod exakt en gång?
- **Konnektivitet**. Finns det en förbindelse mellan alla noder?
- **Bikonnektivitet**. Finns det en nod som gör att grafen inte hänger samman om man tar bort den?
- **Planaritet**. Går det att rita grafen utan att några bågar korsar varandra?
- **Grafisomorfi**. Är två grafer identiska bortsett från namnen på noderna?

**Utmaning**. Vilka av problemen ovan är enkla? Svåra? Omöjliga att lösa?

# De viktigaste metoderna för oriktade grafer

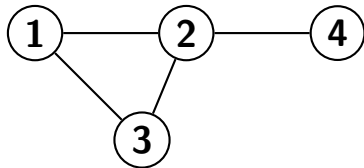
- Noder och bågar
  - är positioner
  - lagrar element
- Åtkomstmetoder
  - `endVertices( $e$ )`: en array med  $e$ 's två ändpunkter
  - `opposite( $v, e$ )`: noden motsatt  $v$  längs  $e$
  - `areAdjacent( $v, w$ )`: **true** om  $v$  och  $w$  är grannar
  - `replace( $v, x$ )`: ersätt elementet i nod  $v$  med  $x$
  - `replace( $e, x$ )`: ersätt elementet i båge  $e$  med  $x$

# De viktigaste metoderna för oriktade grafer

- Uppdateringsmetoder
  - `insertVertex(o)`: sätt in en nod som lagrar elementet  $o$
  - `insertEdge(v, w, o)`: sätt in en kant  $(v, w)$  som lagrar elementet  $o$
  - `removeVertex(v)`: ta bort nod  $v$  (och dess incidenta bågar)
  - `removeEdge(e)`: ta bort båge  $e$
- Iteratormetoder
  - `incidentEdges(v)`: bågarna incidenta till  $v$
  - `vertices()`: alla noder i grafen
  - `edges()`: alla bågar i grafen

## Båglista (Edge list)

- Båglistan representerar alla kanter i grafen
- Nodernas ordning är inte relevant i *oriktade* grafer
- *Riktade* grafer: nodernas ordning dikterar kantens riktning
- Kan innehålla ytterligare information



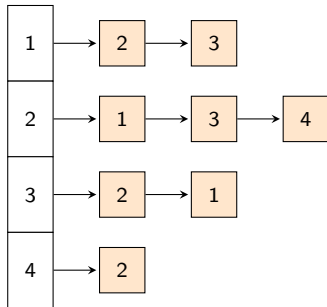
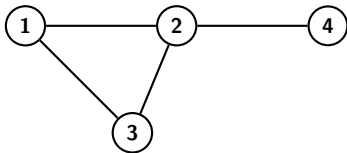
Exempel på arrayrepresentation:

edgeList =  $\left[ \left[ 1, 2 \right], \left[ 1, 3 \right], \left[ 2, 3 \right], \left[ 2, 4 \right] \right]$

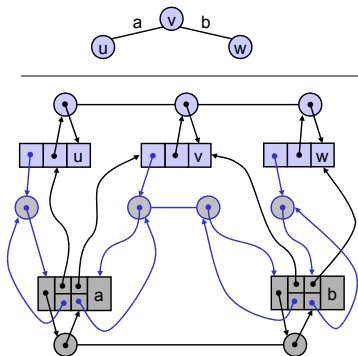


## Grannlista (Adjacency list)

- Lägger till extra struktur till båglistan
- Utöka nodobjekten med heltalsnycklar som mappas mot index i listan
- Varje nod har en lista med grann-noder

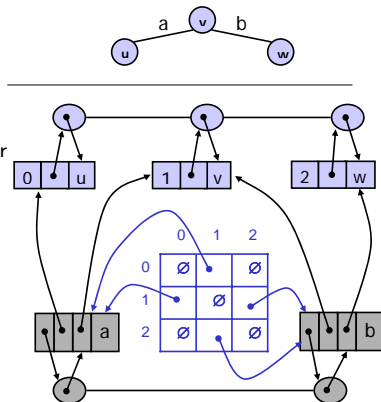


# Grannlista (Adjacency list)



# Grannmatrix (Adjacency matrix)

- Lägger till extra struktur till båglistan
- Utöka nodobjekten med heltalsnycklar som mappas mot index i matrisen (ett heltal rækker)
- Tvådimensionell grannarray
  - Referens till bågobjekt för noder som är grannar
  - **null** för noder som inte är grannar



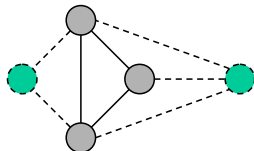
# Asymptotisk prestanda

$n$ noder, $m$ bågar inga parallella kanter inga öglor	Båglista	Grannlista	Grann- matris
minne	$O(n + m)$	$O(n + m)$	$O(n^2)$
incidentEdges( $v$ )	$O(m)$	$O(\text{deg}(v))$	$O(n)$
areAdjacent( $v, w$ )	$O(m)$	$O(\min(\text{deg}(v), \text{deg}(w)))$	$O(1)$
insertVertex( $o$ )	$O(1)$	$O(1)$	$O(n^2)$
insertEdge( $v, w, o$ )	$O(1)$	$O(1)$	$O(1)$
removeVertex( $v$ )	$O(m)$	$O(\text{deg}(v))$	$O(n^2)$
removeEdge( $e$ )	$O(1)$	$O(1)$	$O(1)$

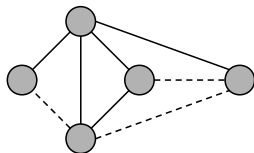
- 1 Grafer
  - Introduktion
  - ADT graf
  - Datastrukturer
- 2 **Sökning i oriktade grafer**
  - DFS
  - BFS
  - DFS vs BFS

# Delgrafer

- En **delgraf**  $S$  av en graf  $G$  är en graf sådan att
  - Noderna i  $S$  är en delmängd av noderna i  $G$
  - Bågarna i  $S$  är en delmängd av bågarna i  $G$
- En **spännande delgraf** av  $G$  är en delgraf som innehåller alla noder i  $G$



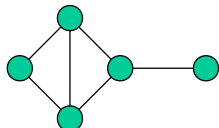
Delgraf



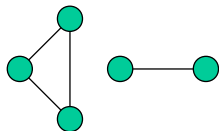
Spännande delgraf

# Konnektivitet

- En graf är **sammanhängande** om det finns en stig mellan varje par av noder
- En **sammanhängande komponent** i en graf  $G$  är en maximal sammanhängande delgraf av  $G$



Sammanhängande graf

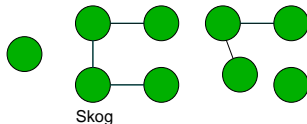
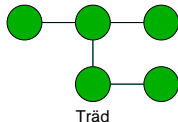


Ej sammanhängande graf med två sammanhängande komponenter

Eng. *connected*, *connected component*

# Träd och skogar

- Ett (fritt) **träd** är en oriktad graf  $T$  sådan att
  - $T$  är sammanhängande
  - $T$  inte har några cykler
  - Den här definitionen av träd skiljer sig från den för rotade träd
- En **skog** är en oriktad graf utan cykler
- De sammanhängande komponenterna i en skog är träd

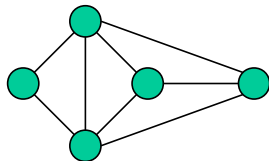


Eng. *tree, forest*

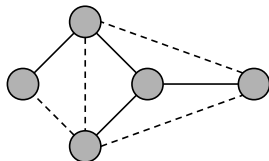


# Spännande träd och skogar

- Ett **spännande träd** till en sammanhängande graf är en spännande delgraf som är ett träd
- Ett spännande träd är inte unikt om inte ursprungsgrafen är ett träd
- En **spännande skog** till en graf är en spännande delgraf som är en skog



Graf



Spännande träd

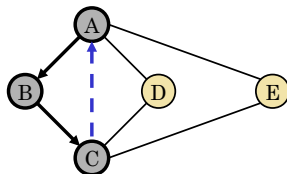
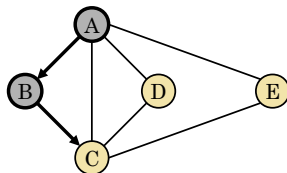
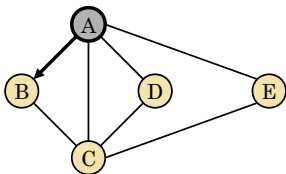
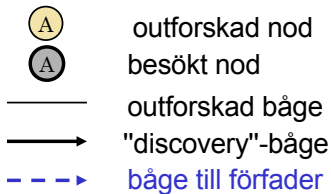
# Djupetförstsökning

- Djupetförstsökning (DFS) är en allmän teknik för att traversera en graf
- DFS i en graf  $G$ 
  - Besöker alla noder och bågar i  $G$
  - Avgör om  $G$  är sammanhängande
  - Beräknar de sammanhängande komponenterna i  $G$
  - Beräknar en spännande skog till  $G$
- DFS på en graf med  $n$  noder och  $m$  bågar tar  $O(n + m)$  tid
- DFS kan utökas för att lösa andra grafproblem
  - Hitta och beskriv en stig mellan två givna noder i en graf
  - Hitta en cykel i en graf

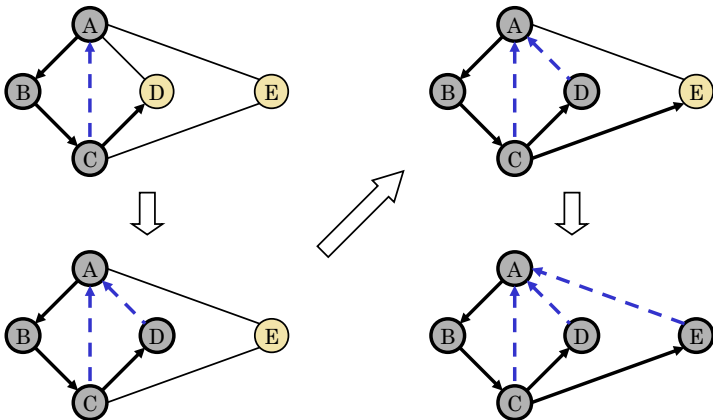
# Algorithm für DFS

```
procedure DFS(G)  
  for all u ∈ G.VERTICES() do  
    SETLABEL(u, UNEXPLORED)  
  for all e ∈ G.EDGES() do  
    SETLABEL(e, UNEXPLORED)  
  for all v ∈ G.VERTICES() do  
    if GETLABEL(v) = UNEXPLORED then  
      DFS(G, v)  
  
procedure DFS(G, v)  
  SETLABEL(v, VISITED)  
  for all e ∈ G.INCIDENTEDGES(v) do  
    if GETLABEL(e) = UNEXPLORED then  
      w ← OPPOSITE(v, e)  
      if GETLABEL(w) = UNEXPLORED then  
        SETLABEL(e, DISCOVERY)  
        DFS(G, w)  
    else  
      SETLABEL(e, BACK)
```

# Exempel



# Exempel





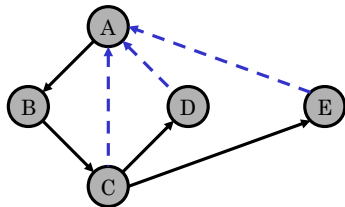
# Egenskaper

## Egenskap 1

$DFS(G, v)$  besöker alla noder och bågar i den sammanhängande delen av  $G$  som  $v$  ingår i

## Egenskap 2

"discovery"-bågarna  $DFS(G, v)$  märker upp utgör ett spännande träd till den sammanhängande komponenten av  $G$  som  $v$  ingår i



# Analys av DFS

- Märka/hämta märkning av nod/båge tar  $O(1)$  tid
- Varje nod märks två ggr
  - en gång som *UNEXPLORED*
  - en gång som *VISITED*
- Varje båge märks två ggr
  - en gång som *UNEXPLORED*
  - en gång som *DISCOVERY* eller *BACK*
- Metoden *incidentEdges* anropas en gång för varje nod
- DFS körs i tid  $O(n + m)$  givet att grafen är representerad med en grannlista
  - kom ihåg att  $\sum_v \text{deg}(v) = 2m$



# Hitta stigar

- Vi kan specialisera DFS-algoritmen till att hitta en stig mellan två givna noder  $v$  och  $z$
- Vi anropar  $\text{DFS}(G, v)$  med  $v$  som startnod
- Vi använder en stack  $S$  för att hålla reda på vägen från startnoden till aktuell nod
- Så snart vi stöter på målnoden  $z$  returnerar vi innehållet på stacken som den sökta stigen

```
procedure PATHDFS( $G, v, z$ )  
  SETLABEL( $v, VISITED$ )  
   $S$ .PUSH( $v$ )  
  if  $v = z$  then  
    skriv ut elementen i  $S$   
  return  
  for all  $e \in G$ .INCIDENTEDGES( $v$ ) do  
    if GETLABEL( $e$ ) = UNEXPLORED then  
       $w \leftarrow$  OPPOSITE( $v, e$ )  
      if GETLABEL( $w$ ) = UNEXPLORED then  
        SETLABEL( $e, DISCOVERY$ )  
         $S$ .PUSH( $e$ )  
        PATHDFS( $G, w, z$ )  
         $S$ .POP() //  $e$   
      else  
        SETLABEL( $e, BACK$ )  
     $S$ .POP() //  $v$ 
```

# Hitta cykler

- Vi kan specialisera DFS-algoritmen till att hitta en enkel cykel
- Vi använder en stack  $S$  för att hålla reda på vägen från startnoden till aktuell nod
- Så snart vi stöter på en kant  $(v, w)$  som leder till en förfader returnerar vi cykeln som innehållet på stacken från toppen till noden  $w$

```
procedure CYCLEDFS( $G, v, z$ )  
  SETLABEL( $v, VISITED$ )  
   $S.PUSH(v)$   
  for all  $e \in G.INCIDENTEDGES(v)$  do  
    if GETLABEL( $e$ ) = UNEXPLORED then  
       $w \leftarrow \text{OPPOSITE}(v, e)$   
       $S.PUSH(e)$   
      if GETLABEL( $w$ ) = UNEXPLORED then  
        SETLABEL( $e, DISCOVERY$ )  
        CYCLEDFS( $G, w$ )  
         $S.POP()$  //  $e$   
      else // hittat cykel  
        repeat  
           $o \leftarrow S.POP()$   
          skriv ut  $o$   
        until  $o = w$   
      return  
   $S.POP()$  //  $v$ 
```

# Breddenförstsökning

- Breddenförstsökning (BFS) är en allmän teknik för att traversera en graf
- BFS i en graf  $G$ 
  - Besöker alla noder och bågar i  $G$
  - Avgör om  $G$  är sammanhängande
  - Beräknar de sammanhängande komponenterna i  $G$
  - Beräknar en spännande skog till  $G$
- BFS på en graf med  $n$  noder och  $m$  bågar tar  $O(n + m)$  tid
- BFS kan utökas för att lösa andra grafproblem
  - Hitta och beskriv en kortaste stig mellan två givna noder i en graf
  - Hitta en enkel cykel i en graf, om det finns någon

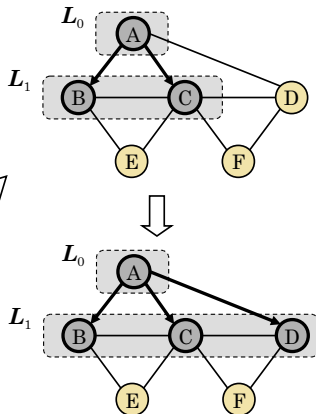
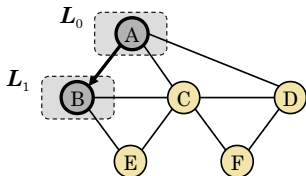
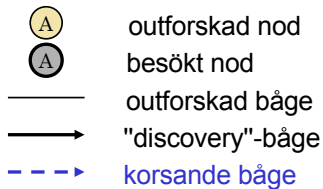
# Algoritm för BFS

```
procedure BFS( $G$ )
  märk alla noder/bågar med UNEXPLORED som i DFS
  for all  $v \in G.VERTICES()$  do
    if GETLABEL( $v$ ) = UNEXPLORED then BFS( $G, v$ )

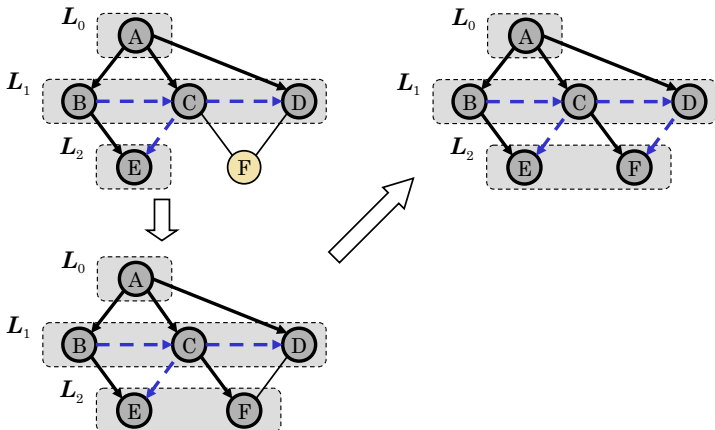
procedure BFS( $G, s$ )
   $L_0 \leftarrow$  ny tom sekvens;  $L_0.INSERTLAST(s)$ ;
  SETLABEL( $s, VISITED$ );  $i \leftarrow 0$ 
  while  $\neg L_i.ISEMPTY()$  do
     $L_{i+1} \leftarrow$  ny tom sekvens
    for all  $v \in L_i.ELEMENTS()$  do
      for all  $e \in G.INCIDENTEDGES(v)$  do
        if GETLABEL( $e$ ) = UNEXPLORED then
           $w \leftarrow$  OPPOSITE( $v, e$ )
          if GETLABEL( $w$ ) = UNEXPLORED then
            SETLABEL( $e, DISCOVERY$ )
            SETLABEL( $w, VISITED$ )
             $L_{i+1}.INSERTLAST(w)$ 
          else
            SETLABEL( $e, CROSS$ )

   $i \leftarrow i + 1$ 
```

# Exempel



## Exempel



# Egenskaper

Låt  $G_s$  beteckna den sammanhängande delen av  $G$  som  $s$  ingår i

## Egenskap 1

$BFS(G, s)$  besöker alla noder och bågar i  $G_s$

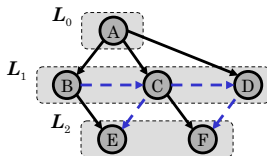
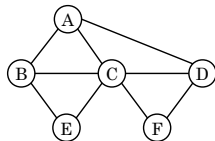
## Egenskap 2

"discovery"-bågarna  $BFS(G, s)$  märker upp utgör ett spännande träd  $T_s$  till  $G_s$

## Egenskap 3

För varje nod  $v$  i  $L_i$  gäller

- Stigen i  $T_s$  från  $s$  till  $v$  har  $i$  bågar



# Analys av BFS

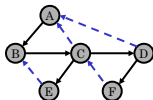
- Märka/hämta märkning av nod/båge tar  $O(1)$  tid
- Varje nod märks två ggr
  - en gång som *UNEXPLORED*
  - en gång som *VISITED*
- Varje båge märks två ggr
  - en gång som *UNEXPLORED*
  - en gång som *DISCOVERY* eller *CROSS*
- Varje nod sätts in en gång i en sekvens  $L_i$
- Metoden *incidentEdges* anropas en gång för varje nod
- BFS körs i tid  $O(n + m)$  givet att grafen är representerad med en grannlista
  - kom ihåg att  $\sum_v \text{deg}(v) = 2m$



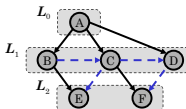
# Tillämpningar

Tillämpningar	DFS	BFS
Spännande träd, sammanhängande komponenter, stigar, cykler	✓	✓
Kortaste stigar		✓
2-sammanhängande komponenter	✓	

DFS



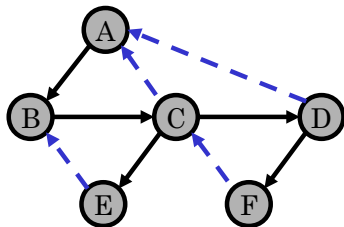
BFS



## Bågar som leder till redan besökta noder

### båge till förfader

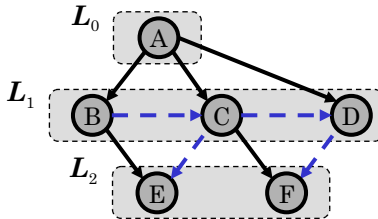
- $w$  är en förfader till  $v$  i trädet av "discovery"-bågar



DFS

### korsande båge

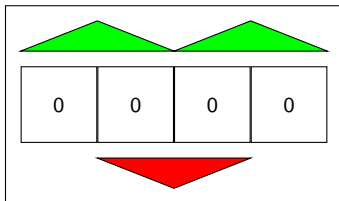
- $w$  finns i samma nivå som  $v$  eller i nästa nivå i trädet av "discovery"-bågar



BFS

## Problem

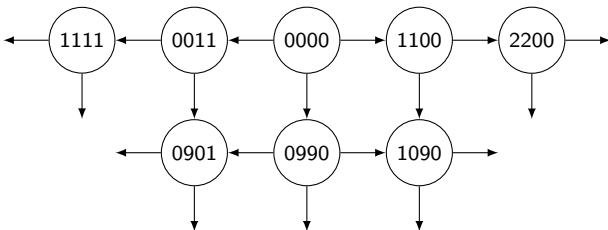
Du har hittat ett gammalt kodlås som du vill låsa upp. Du vet att koden är 3146, och just nu visar siffrorna på låset 0000. Låset har knappar för att öka eller minska de olika siffrorna, dock är många av knapparna trasiga. Hur skriver du in koden med så få knapptryck som möjligt? Låset ser ut som följer:



# Lösningssidé

Representera problemet som en riktad graf:

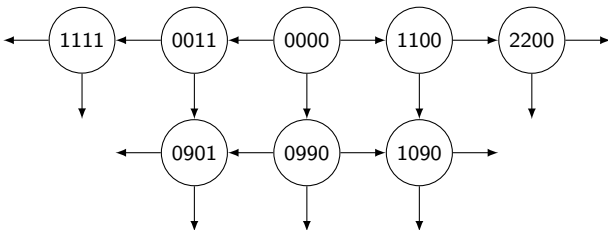
- En nod för varje kombination
- Varje båge motsvarar ett knapptryck



# Lösningssidé

Representera problemet som en riktad graf:

- Oviktad graf  $\Rightarrow$  BFS
- Varje väg motsvarar en lösning!



Nästa gång:  
Riktade och viktade grafer

[www.liu.se](http://www.liu.se)