Föreläsning 8
Datastrukturer (DAT037)

Fredrik Lindblad[1]

22 november 2017

[1]Slides skapade av Nils Anders Danielsson har använts som utgångspunkt. Se
`http://www.cse.chalmers.se/edu/year/2015/course/DAT037`

Innehåll

- djupet först-sökning
- topologisk sortering
- starkt sammanhängande komponenter

Djupet först-sökning

Djupet först-sökning (DFS)

- Metod för att gå igenom alla noder och kanter.
- Har tidigare sett bredden först-sökning (kortaste vägen för oviktade grafer).
- Fungerar för oriktade och riktade grafer.

Djupet först-sökning: mall

```
visited = new array with indices {0,…,|V|-1}
            and all elements equal to false

// Startar/startar om sökning.
for v ∈ {0,…,|V|-1} do
  if not visited[v] then
    dfs(v)

// Utför sökning.
dfs(v) {
  visited[v] = true

  for w ∈ {w | (v, w) ∈ E} do
    if not visited[w] then
      dfs(w)
}
```

Djupet först-sökning: mall

```
visited = new array with indices {0,…,|V|-1}   Θ(|V|)
            and all elements equal to false

// Startar/startar om sökning.
for v ∈ {0,…,|V|-1} do                          Θ(|V|) ggr
  if not visited[v] then
    dfs(v)

// Utför sökning.
dfs(v) {                                        Θ(|V|) ggr
  visited[v] = true

  for w ∈ {w | (v, w) ∈ E} do                   Θ(|E|) ggr tot
    if not visited[w] then
      dfs(w)
}
```

Djupet först-sökning

Tidskomplexitet: $\Theta(|V| + |E|)$ (linjär).

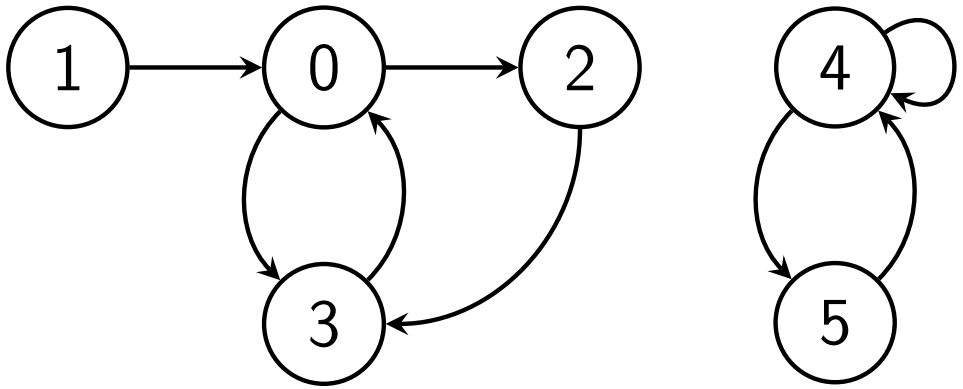
Uppspännande skog

Depth-first spanning forest:

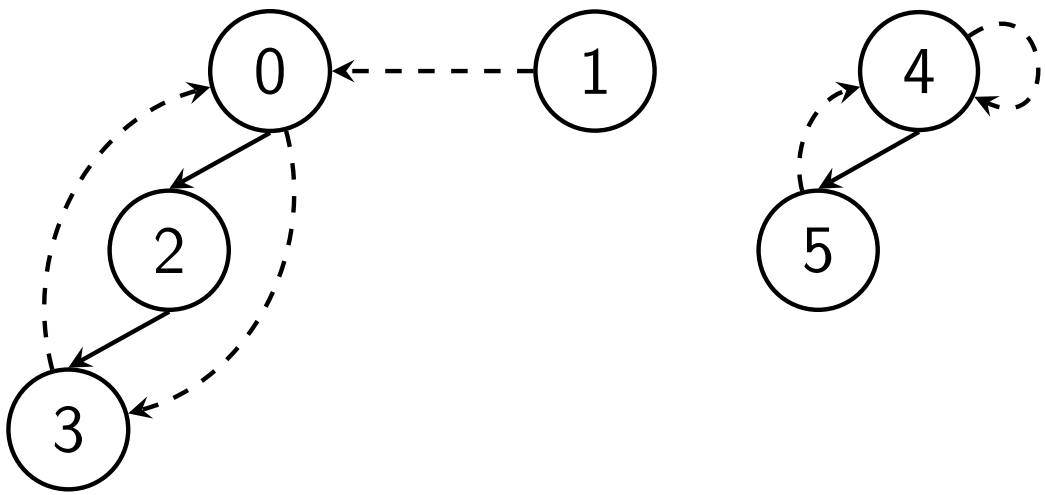
- Skog: Mängd av träd.
- Ett träd för varje toppnivåanrop till `dfs`.
- Rekursiva anrop till `dfs` ger upphov till vanliga trädkanter.
- Om `visited[w]` = `true` får man istället en "speciell" kant:
  - Bakåtkant: Om kanten pekar uppåt (eller på samma nod).
  - Framåtkant: Om kanten pekar nedåt.
  - Tvärkant: I övriga fall ("åt vänster").

Uppspännande skog

Graf:



Skog
(om vi söker från 0, 1 och 4, och väljer 2 före 3):



Djupet först-sökning

- DFS kan användas för att implementera andra algoritmer.
- Exempel:
  Är en oriktad graf sammanhängande?
  - Testa om DFS från godtycklig nod besöker alla noder (utan omstart).

Sammanhängande?

```
if |V| = 0 then return true

visited = new array with indices {0,…,|V|-1}
            and all elements equal to false
dfs(0)
for v ∈ {1,…,|V|-1} do
  if not visited[v] then return false
return true

dfs(v) {
  visited[v] = true
  for w ∈ {w | {v, w} ∈ E} do
    if not visited[w] then
      dfs(w)
}
```

Topologisk sortering

Topologisk sortering

Definition:

- Total ordning av V.
- Om det finns en väg från v_1 till v_2 så är $v_1 < v_2$.

Exempel:

- Förkunskapskrav $\Rightarrow$ giltig ordning av kurser.

Topologisk sortering

- Ointressant för oriktade grafer.
- Cykel $\Rightarrow$ ingen topologisk sortering.
- DAGs (riktade acykliska grafer) kan alltid sorteras topologiskt.
- Tillräckligt villkor för att vara cyklisk: Grafen innehåller minst en nod, och alla noder har ingrad > 0.

Topologisk sortering: enkel algoritm

```
r = new empty list

while V ≠ ∅ do
  if any v ∈ V with indegree(v) = 0 then
    r.add-last(v)
    remove v from G
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Topologisk sortering: enkel algoritm

```
r = new empty list

while V ≠ ∅ do
  if any v ∈ V with indegree(v) = 0 then
    r.add-last(v)
    remove v from G
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Kan vi undvika radering?

Topologisk sortering: enkel algoritm (2)

```
r = new empty list
d = map from vertices to their indegrees
    // null for nodes in r.

repeat |V| times
  if d[v] == 0 for some v then
    r.add-last(v)
    d[v] = null
    for each direct successor v' of v do
      decrease d[v'] by 1
  else
    raise error: cycle found

return r  // Nodes, topologically sorted.
```

Grafrepresentation

Pseudokod: Behöver mer information för tidskomplexitetsanalys.

Grafrepresentation (den här gången):

- Noder numrerade $0, 1, ..., |V| - 1$.
- Array `adjacent` med $|V|$ positioner.
- `adjacent[i]` innehåller grannlista (länkad lista) för nod `i`.

`r`: dynamisk array, `d`: array.

Pseudokod

Exempel:

```
// d är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
d = new array of size |V|

// Initialisera d.
for i in [0,...,|V|-1] do
  d[i] = 0
for i in [0,...,|V|-1] do
  for each direct successor j of i do
    d[j]++
```

Pseudokod

Exempel:

```
// d är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
d = new array of size |V|

// Initialisera d.
for i in [0,...,|V|-1] do                    O(|V|) ggr
  d[i] = 0                                   O(1)
for i in [0,...,|V|-1] do
  for each direct successor j of i do
    d[j]++
```

Pseudokod

Exempel:

```
// d är en map från nodindex till
// /virtuella/ ingrader, de ingrader
// respektive nod skulle ha om alla noder i
// r togs bort från grafen. Den virtuella
// ingraden för noder i r är null.
d = new array of size |V|

// Initialisera d.
for i in [0,...,|V|-1] do                    O(|V|) ggr
  d[i] = 0                                   O(1)
for i in [0,...,|V|-1] do                    O(|V|)
  for each direct successor j of i do        O(|E|) ggr tot
    d[j]++                                   O(1)
```

Topologisk sortering: enkel algoritm (2)

```
r = new empty list                                  O(1)
d = initiera d enligt koden på föreg. s.            O(|V| + |E|)
    // null for nodes in r.

repeat |V| times                                    O(|V|) ggr
 if d[v] == 0 for some v then                        O(|V|)
  r.add-last(v)                                     O(1)
  d[v] = null                                       O(1)
  for each direct successor v' of v do              O(|E|) ggr tot
   decrease d[v'] by 1                              O(1)
 else
  raise error: cycle found                          O(1)

return r  // Nodes, topologically sorted.           O(1)
```

Totalt: $O(|V|^2 + |E|) = O(|V|^2)$.

Topologisk sortering med kö

```
r = new empty list
d = map from vertices to their indegrees
q = queue with all nodes of indegree 0

while q is non-empty do
  v = q.dequeue()
  r.add-last(v)
  for each direct successor v' of v do
    decrease d[v'] by 1
    if d[v'] = 0 then
      q.enqueue(v')

if r.length() < |V| then
  raise error: cycle found

return r  // Nodes, topologically sorted.
```

Analysera värstafallstidskomplexiteten.

- $\Theta(|V|)$.
- $\Theta(|E|)$.
- $\Theta(|V| + |E|)$.
- $\Theta(|V|^2)$.

Bonusövning

Vad händer om man använder en stack istället för en kö?

Analysera värstafallstidskomplexiteten.

- $\Theta(|V|)$.
- $\Theta(|E|)$.
- $\Theta(|V| + |E|)$.
- $\Theta(|V|^2)$.

Bonusövning

Vad händer om man använder en stack istället för en kö?

Svar: $O(|V| + |E|)$. Bonusövning: Fungerar också, samma tidskomplexitet, ev annan ordning.

Topologisk sortering med kö

```
r = new empty list                                  Θ(1)
d = map from vertices to their indegrees            Θ(|V| + |E|)
q = queue with all nodes of indegree 0              O(|V|)

while q is non-empty do                             O(|V|) ggr
 v = q.dequeue()                                     Θ(1)
 r.add-last(v)                                       Θ(1)
 for each direct successor v' of v do              O(|E|) ggr tot
  decrease d[v'] by 1                                Θ(1)
  if d[v'] = 0 then                                  Θ(1)
   q.enqueue(v')                                     Θ(1)

if r.length() < |V| then                            Θ(1)
 raise error: cycle found                           Θ(1)

return r  // Nodes, topologically sorted.           Θ(1)
```

Topologisk sortering

Kan sortera DAG topologiskt genom att:

- Utföra DFS.
- Gå igenom träden i uppspännande skogen i preordning, med skillnaden att (barn)träden gås igenom från höger till vänster.
- Om grafen kan vara cyklisk så är det inte så lätt att avgöra det. Måste hålla koll på om uppåt- eller tvärkanter uppstår.

Starkt sammanhängande komponenter

Starkt sammanhängande komponenter

För riktade grafer:

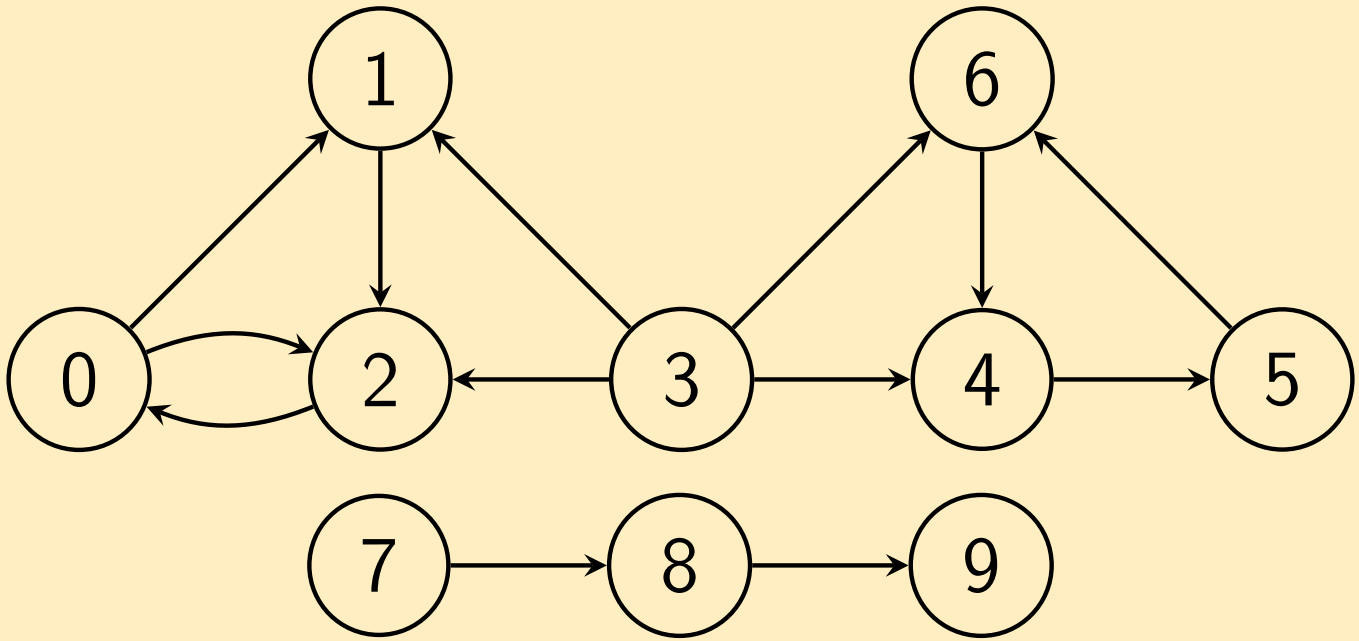
- Starkt sammanhängande graf:
  Om $u, v \in V$ så finns det en väg från u till v
  (och vice versa).
- Starkt sammanhängande komponent (SCC):
  Maximal starkt sammanhängande delgraf.
- Om varje SCC byts ut mot en ny nod
  (en per SCC) får vi en acyklisk multigraf.

Starkt sammanhängande komponenter

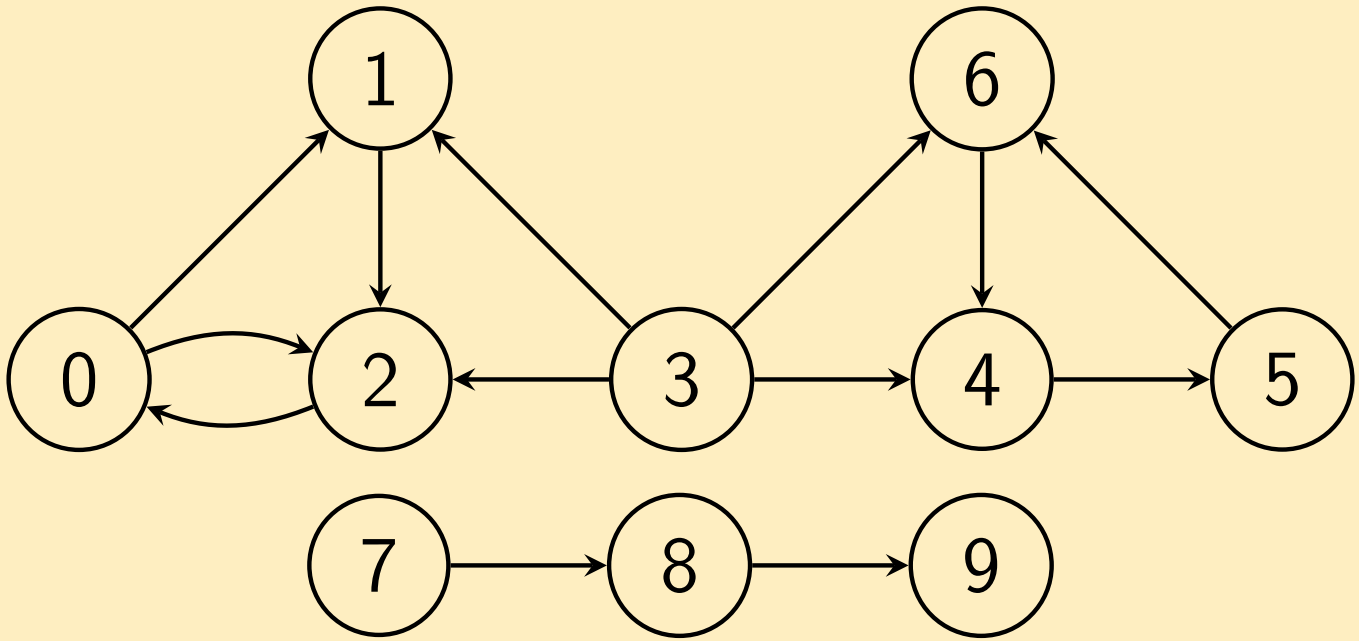
Exempel:

- Noder: Funktioner.
- Kanter: Anrop från en funktion till en annan.
- Funktioner i en SCC är ömsesidigt rekursiva.

Hur många starkt sammanhängande komponenter innehåller följande graf?



Hur många starkt sammanhängande komponenter innehåller följande graf?



Svar: 6

SCC-algoritm

Kan vi hitta alla SCCer?

- Kan hitta alla noder i ”löv-SCC” genom DFS (utan omstart) från någon av dem.
- Kan sedan ta bort (ignorera) löv-SCCn och fortsätta med annan löv-SCC.
- Men hur hittar man löv-SCCer?

SCC-algoritm

Utför DFS, gå igenom uppspännande skogen i preordning, från höger till vänster, som i topologisk sortering:

- Första noden (om någon)
  måste höra till en "rot-SCC".
- Tar man bort alla noder som hör till den SCCn
  så hör nästa nod (om någon)
  återigen till en rot-SCC.
- Och så vidare...

För löv-SCC:
Utför DFS *baklänges* (på *reverserad* graf).

SCC-algoritm

1. Utför DFS baklänges.
2. Skapa en nodlista med `preorderRL`.
3. Utför DFS framlänges,
   börja hela tiden med
   första obesökta listnoden.
   Varje sökning ger en SCC.

Tidskomplexitet: $\Theta(|V| + |E|)$.