Föreläsning 7
Datastrukturer (DAT037)

Fredrik Lindblad[1]

2016-11-21

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

Förra gången:

- hashtabeller
- grafer
  - definitioner
  - representationer
  - kortaste väg för oviktad graf

Idag:

- kortaste vägen (Dijkstras algoritm)
- minimala uppspännande träd (Prims och Kruskals algoritm)
- djupet-först-genomlöpning

Kortaste vägen

Kortaste vägen-problem:

- Givet två noder u och v,
  hitta en kortaste väg från u till v.
- Givet nod u, för varje nod v,
  hitta en kortaste väg från u till v.
- Hitta kortaste vägen
  från varje nod till varje annan.

Visar sig naturligt att lösa den mittersta varianten ovan.

Oviktade grafer: bredden först-sökning

I algoritmen på nästa slide:

- `d` lagrar kortaste kända vägen till alla noder.
- `p` lagrar föregående nod för kortaste vägen.
- `q` är en kö som bestämmer besöksordningen av noder.

Oviktade grafer: bredden först-sökning

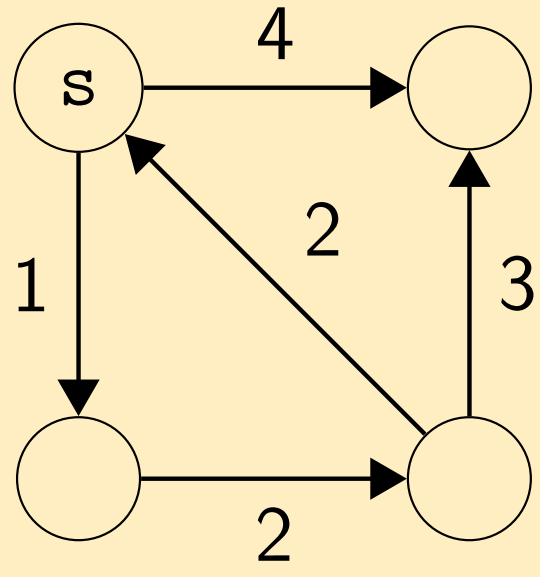
```
d = new array of size |V|, initialised to ∞
p = new array of size |V|, initialised to null
q = new empty queue

q.enqueue(s)
d[s] = 0

while q is non-empty do
  v = q.dequeue()
  for each direct successor v' of v do
    if d[v'] = ∞ then
      d[v'] = d[v] + 1
      p[v'] = v
      q.enqueue(v')

return (d, p)
```

Oviktade grafer: bredden först-sökning

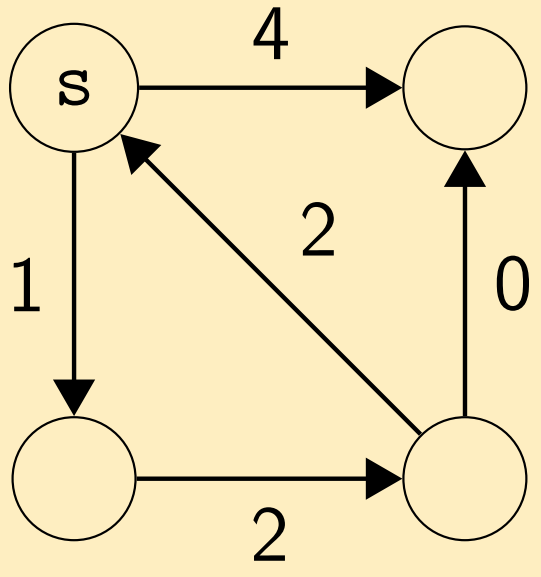
```
d = new array of size |V|, initialised to ∞      O(|V|)
p = new array of size |V|, initialised to null   O(|V|)
q = new empty queue

q.enqueue(s)
d[s] = 0

while q is non-empty do                          O(|V|) ggr
  v = q.dequeue()
  for each direct successor v' of v do           O(|E|) ggr
    if d[v'] = ∞ then
      d[v'] = d[v] + 1
      p[v'] = v
      q.enqueue(v')

return (d, p)
```

Fungerar algoritmen för viktade grafer (om `+ 1` byts ut mot `+ vikten av kanten från v till v'`)? Testa!

A:



B:



Viktade grafer

Viktade grafer: Dijkstras algoritm

- Observation: Kan behöva uppdatera kostnader flera gånger.
- Antagande: Inga negativa vikter.
- Grundidé:
  - Anta att vi redan känner till kortaste vägen till vissa noder.
  - Beräkna avståndet till alla de här nodernas direkta efterföljare (utom noderna själva).
  - Det kortaste av de här avstånden måste vara korrekt.

I algoritmen på nästa slide:

- `d` lagrar hittills kortaste vägen till alla noder.
- `p` lagrar föregående nod för hittills kortaste vägen.
- `k` lagrar om kortaste vägen till noden är känd.

```
d    = new array of size |V|, initialised to ϖ
p    = new array of size |V|, initialised to null
k    = new array of size |V|, initialised to false
d[s] = 0

repeat until no unknown node v' satisfies d[v'] < ϖ

  v = one of the unknown nodes v' with smallest d[v']
  k[v] = true

  for each direct successor v' of v do
    if (not k[v']) and d[v'] > d[v] + c(v,v') then
       d[v'] = d[v] + c(v,v')
       p[v'] = v

return (d, p)
```

Viktade grafer: Dijkstras algoritm

- Enkel implementation:
  $O(|E| + |V|^2) = O(|V|^2)$.

(Antagande: Viktoperationer tar konstant tid.)

```
d    = empty map from node indices (by default ∞)
p    = empty map from node indices
k    = empty set of node indices
q    = new empty priority queue
d[s] = 0
q.insert(s, 0)

while q is non-empty do
  v = q.delete-min()
  if v not in k then
    insert v into k
    for each direct successor v' of v do
      if (v' not in k) and d[v'] > d[v] + c(v,v') then
         d[v'] = d[v] + c(v,v')
         p[v'] = v
         q.insert(v', d[v'])

return (d, p)
```

Viktade grafer: Dijkstras algoritm

Med prioritetskö (binär heap eller leftistheap):

- Om `d`, `p` och `k` är arrayer:

$$O(|V| + 2|E| \log |E|) = O(|V| + |E| \log |V|).$$

- Med vissa andra avbildnings- och mängddatastrukturer:

$$O(|E| \log |V|).$$

- För täta grafer med $|E| = \Theta(|V|^2)$:

$$O(|V|^2 \log |V|).$$

Viktade grafer: Dijkstras algoritm

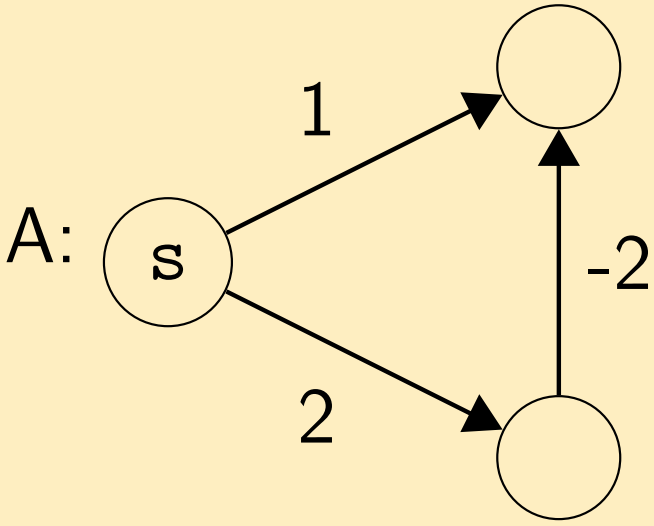
Kan också använda `decrease-key`:

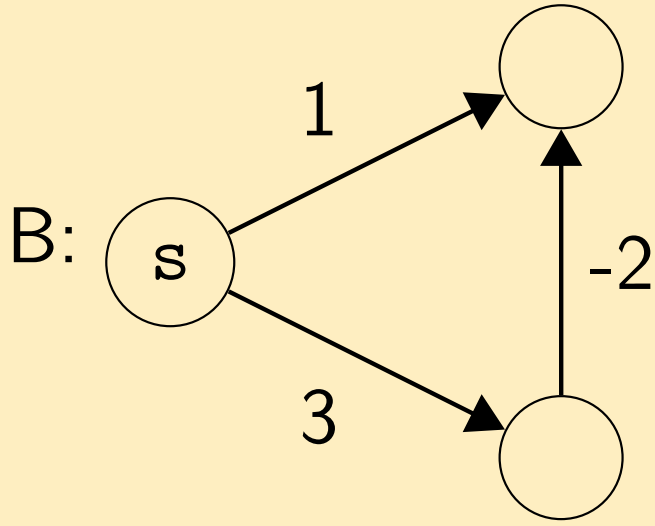
- Dubbletter i kön undviks.
- $O(|E| \log |V|)$.

Giriga algoritmer

- Girig algoritm: Varje steg baserat på det som verkar bäst ”just nu”.

Ger Dijkstras algoritm rätt svar för följande grafer?





Minsta upp-spännande träd

Fritt träd (träd utan rot)

Sammanhängande, acyklisk, oriktad graf.

Uppspännande träd

Träd som är delgraf och innehåller alla noder.

Minsta uppspännande träd (MST)

Uppspännande träd vars totala (kant)vikt är $\leq$ vikten av alla andra uppspännande träd.

Minsta uppspännande träd

Några tillämpningar:

- Konstruktion av nätverk.
- Klusteranalys.
- Bildsegmentering.
- Och mycket annat.

Minsta uppspännande träd

Några egenskaper:

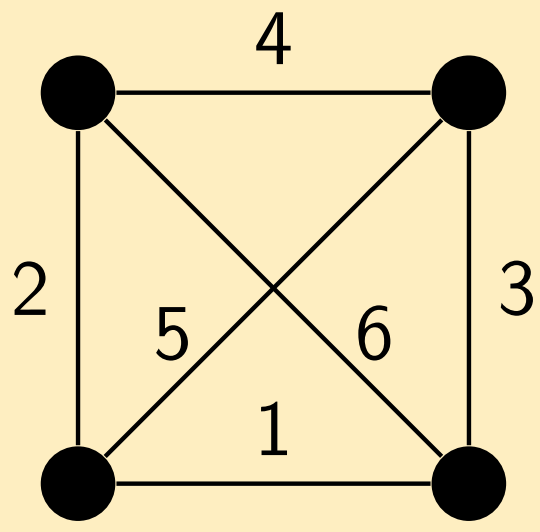
- Existerar omm grafen är sammanhängande.
- Lägger man till en kant får man en cyklisk graf med exakt en cykel.
- Tar man sedan bort en kant från cykeln får man ett uppspännande träd.

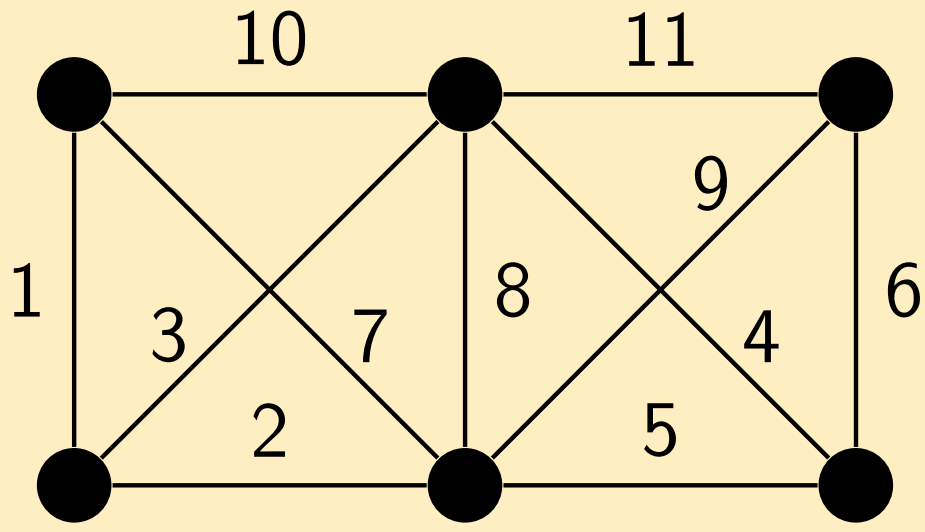
Prims algoritm

Väldigt lik Dijkstras algoritm.

- Båda *giriga algoritmer*.
- Lägger till billigaste nya noden.
- Dijkstra: Minsta avståndet från startnoden.
- Prim: Minsta avståndet från gammal nod.

Vad är den totala vikten av följande två grafers minsta uppspännande träd?





Prims algoritm (för icketom graf)

```
Done = new set containing arbitrary node s
ToDo = V \ { s }
T    = new empty set  // MST för noder i Done.

while ToDo is non-empty do
  if no edge connects Done and ToDo then
    raise error: graph not connected

  (u,v) = cheapest edge connecting Done and ToDo
          (u ∈ Done, v ∈ ToDo)

  Done = Done ∪ { v }
  ToDo = ToDo \ { v }
  T    = T    ∪ { (u,v) }

return T  // Bara kanterna.
```

Prims algoritm: korrekthet

Algoritmen ger ett uppspännande träd eller,
om grafen ej är sammanhängande,
ett felmeddelande:

- Invariant: `T` är ett uppspännande träd för noderna i `Done`.
- Invarianten håller i början:
  Trädet med noden `s` och inga kanter spänner upp $\{$ `s` $\}$.
- Invarianten bevaras:
  Lägger alltid till ny nod, aldrig cykel.
  (Om felmeddelande: ej sammanhängande.)
- När vi kör `return T` så är `Done = V`.

Prims algoritm: korrekthet

Algoritmen ger ett MST (om det finns något):

- Invariant: $\mathtt{T}$ är ett delträd av något MST.
- Invarianten håller i början:
  Det tomma trädet är ett delträd av alla MST.
- Invarianten bevaras:
  Antagande: $\mathtt{T}$ är ett delträd av något MST.
  Visa (för $k = (\mathrm{u}, \mathrm{v})$):
  $\mathtt{T} \cup \{ k \}$ är ett delträd av något MST.

Prims algoritm: korrekthet

- Antagande: `T` delträd av MST M.
- Visa: $\texttt{T} \cup \{ k \}$ delträd av *något* MST.
- Klara om $k \in M$. Annars finns cykel C med $k \in C$ och $C \setminus \{ k \} \subseteq M$.
- Betrakta mängden $K = C \setminus (\texttt{T} \cup \{ k \})$.
- k förbinder `Done` och `ToDo`, `T` gör det inte, C är en cykel $\Rightarrow$ en kant $k' \in K$ förbinder `Done` och `ToDo`.
- k' är minst lika dyr som k.
- $\texttt{T} \cup \{ k \}$ delträd av $(M \setminus \{ k' \}) \cup \{ k \}$ (som är ett MST).