Föreläsning 7
Datastrukturer (DAT037)

Fredrik Lindblad[1]

20 november 2017

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

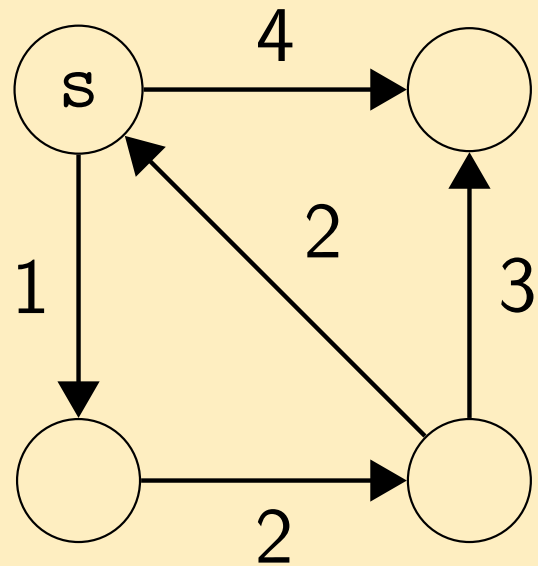
Innehåll

- Kortaste vägen – Dijkstras algoritm
- Minimala uppspännande träd – Prims och Kruskals algoritmer

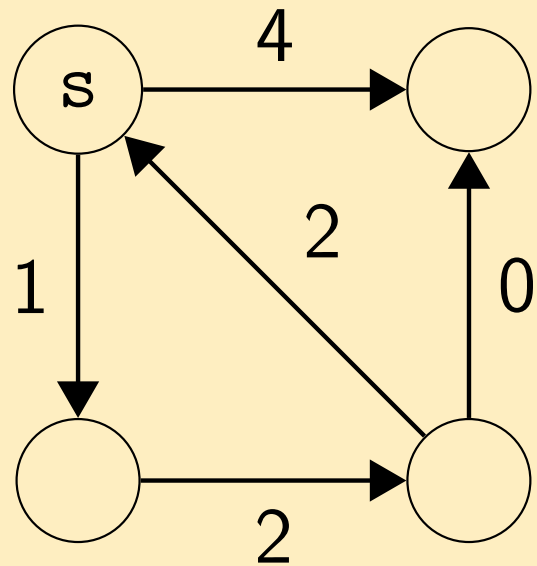
Kortaste vägen

Vi såg att bredden först-sökning löser kortaste vägen-problemet för oviktade grafer. Fungerar algoritmen för viktade grafer (om `+ 1` byts ut mot `+ vikten av kanten från v till v'`)? Testa!

A:

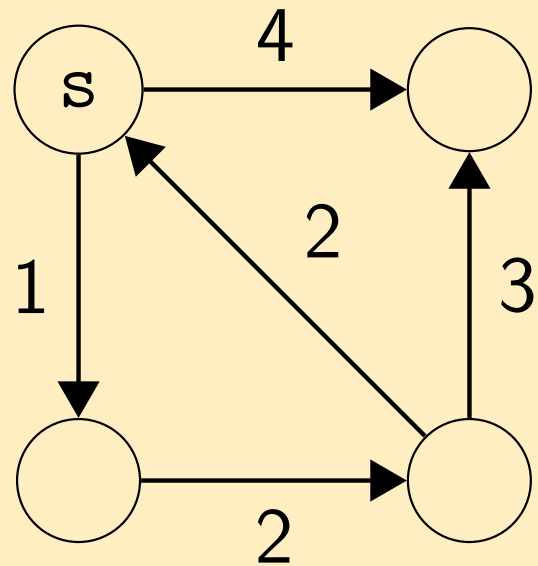


B:

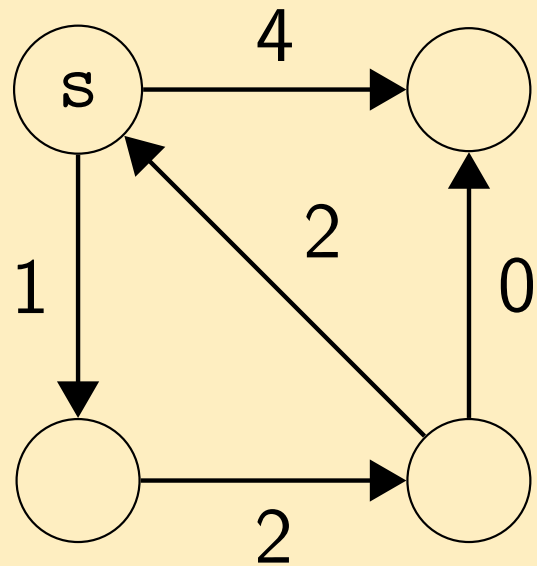


Vi såg att bredden först-sökning löser kortaste vägen-problemet för oviktade grafer. Fungerar algoritmen för viktade grafer (om `+ 1` byts ut mot `+ vikten av kanten från v till v'`)? Testa!

A:



B:



Svar: A: Ja, B: Nej

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:
  $O(|E| + |V|^2)$
- För icke multigrafer gäller $|E| = O(|V|^2)$ så då gäller även:
  $O(|V|^2)$

(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

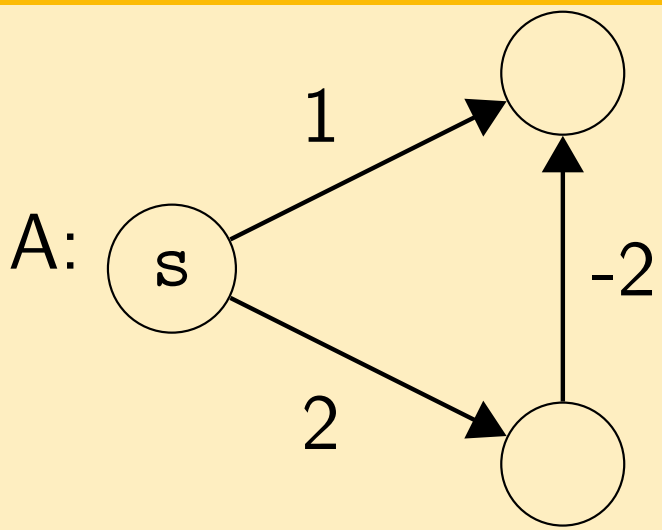
En variant är att, när en kortare väg till en nod hittas, ta bort det gamla kö-elementet för noden eller ändra dess prioritet.

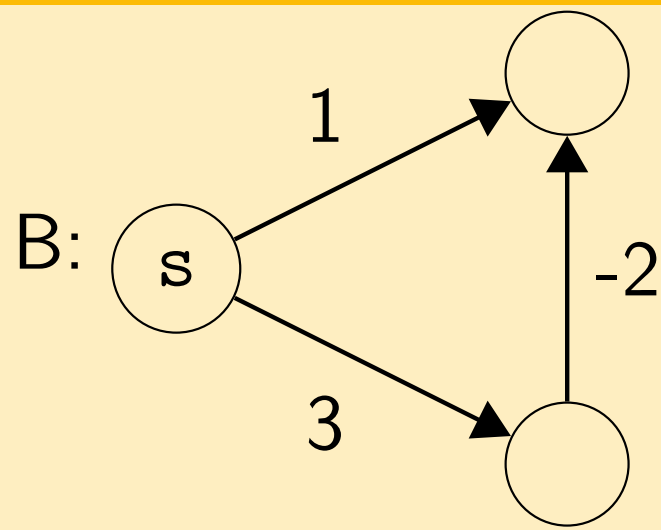
- Dubbletter i kön undviks.
- Direkt analys av algoritmen ger då: $O((|V| + |E|) \log |V|)$
- Eftersom algoritmen inte når fler än $|E| + 1$ noder så gäller även: $O(|E| \log |V|)$

Giriga algoritmer

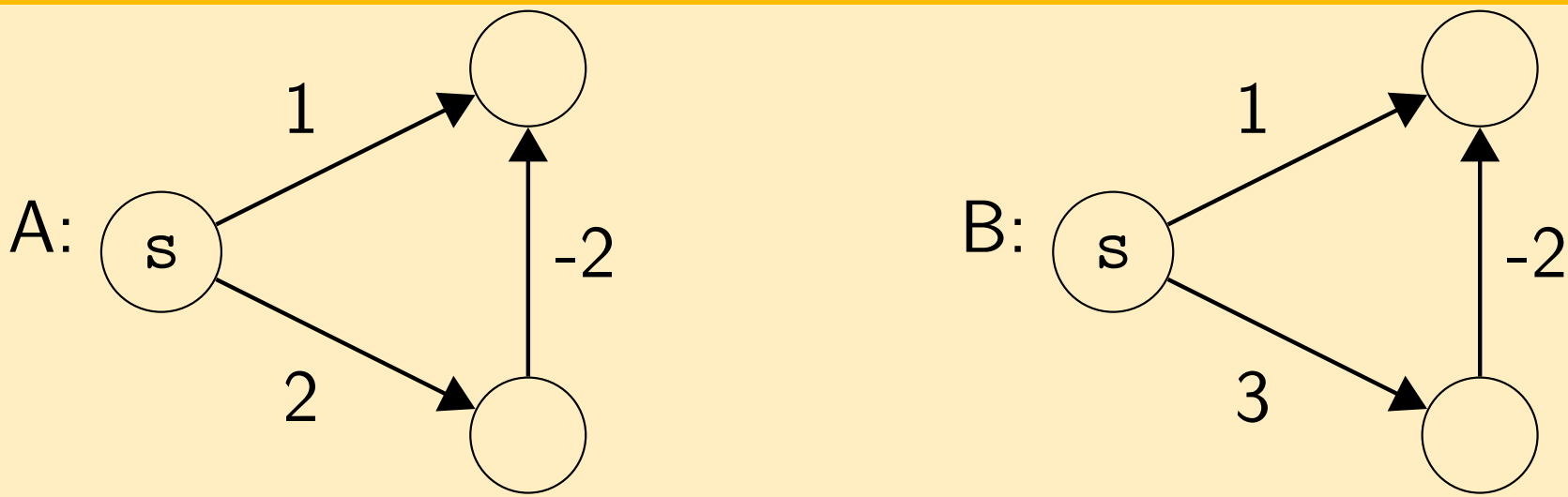
- Girig algoritm: Varje steg baserat på det som verkar bäst ”just nu”.

Ger Dijkstras algoritm rätt svar för följande grafer?





Ger Dijkstras algoritm rätt svar för följande grafer?



Svar: A: Nej, B: Ja

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:

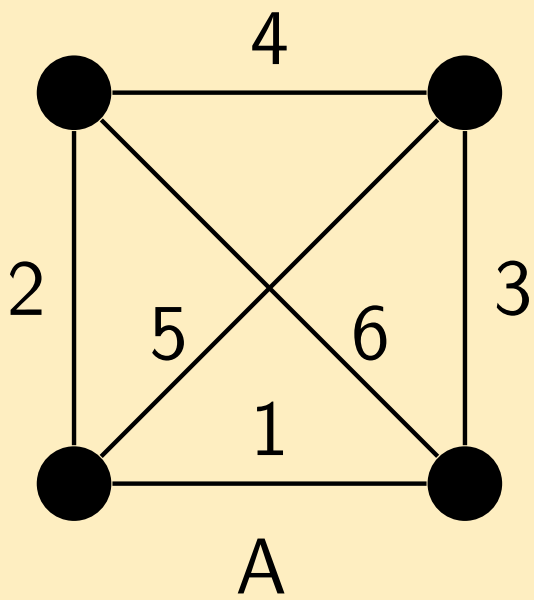
- Konstruktion av nätverk.
- Klusteranalys.
- Bildsegmentering.
- Och mycket annat.

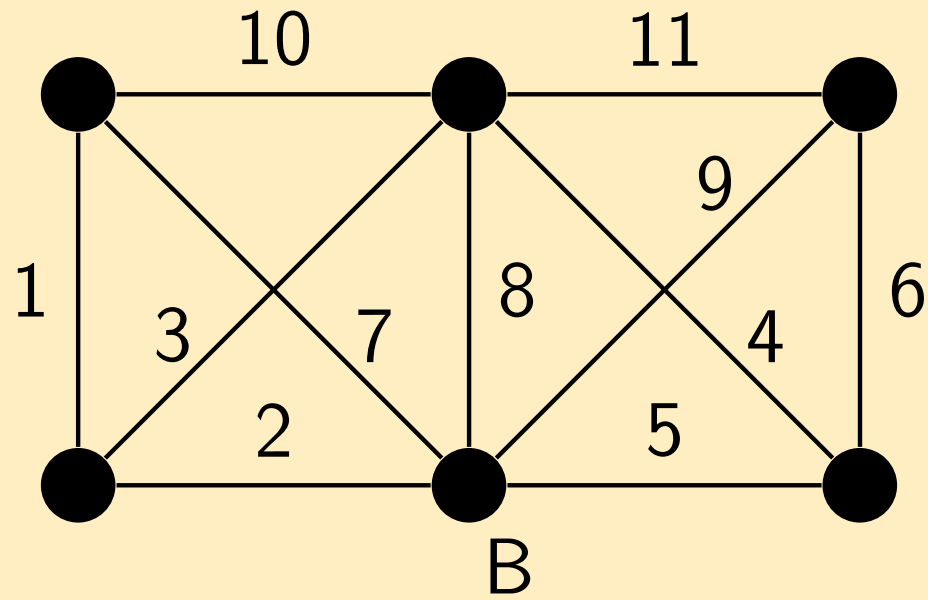
Minsta uppspännande träd

Några egenskaper:

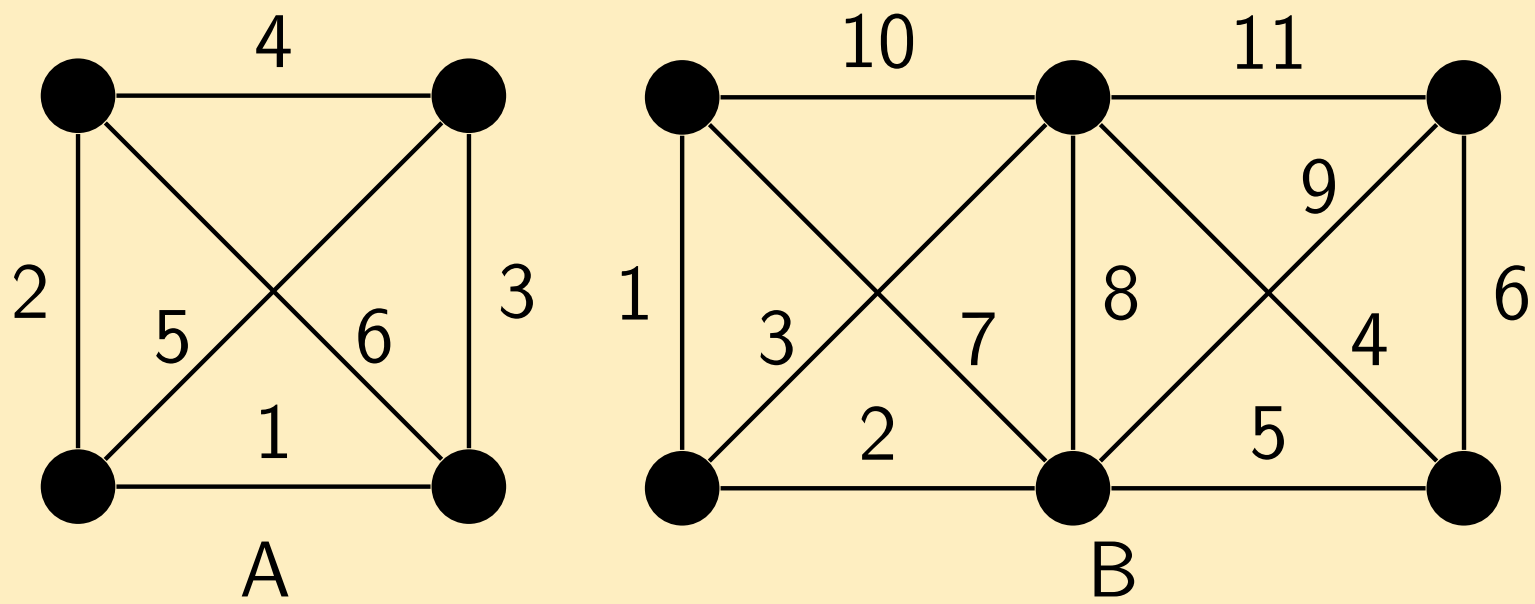
- Existerar omm grafen är sammanhängande.
- Lägger man till en kant får man en cyklisk graf med exakt en cykel.
- Tar man sedan bort en kant från cykeln får man ett uppspännande träd.

Vad är vikterna av följande två grafers minsta uppspännande träd?





Vad är vikterna av följande två grafers minsta uppspännande träd?



Svar: Vänstra: $1 + 2 + 3 = 6$,
Högra: $1 + 2 + 3 + 4 + 6 = 16$

Prims algoritm

Väldigt lik Dijkstras algoritm.

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

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 $k = (\mathrm{u}, \mathrm{v})$, en kant från `Done` till `ToDo` minst minsta vikt:
  $\mathtt{T} \cup \{ k \}$ 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).

Prims algoritm

Kan implementera Prims algoritm
på ungefär samma sätt som Dijkstras.
(Glöm inte kontrollera att
grafen är sammanhängande.)

Tidskomplexitet
(om viktoperationer tar konstant tid):

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

Kruskals algoritm

- För osammanhängande grafer är en minimal uppspännande skog en uppsättning fria träd som innehåller alla noder i grafen och och har minimal total vikt.
- Prims algoritm hanterar inte osammanhängande grafer.
- Istället Kruskals algoritm

Kruskals algoritm

Idé:

- Ha en partition av noderna i grafen och en MST för varje delmängd.
- Lägg till nästa minsta båge som går mellan två delmängder (inte inom en delmängd).
- Slå samman dessa två delmängder.
- Korrektheten följer ett liknande resonemang som för Prims.

Kruskals algoritm

```
 1  T = tom mängd av kanter
 2  p = avbildning från noder till mängder av noder
 3  för varje nod v låt p[v] = {v}
 4  för varje kant (u, v) ordnade i ökad vikt-ordning
 5    om p[u] != p[v]
 6      lägg till (u, v) i T
 7      låt l vara den mindre mängden av p[u] och p[v]
 8      och s den större
 9      låt s = unionen av s och l
10      för varje nod w i l låt p[w] = s
```

Kruskals algoritm – komplexitet

- Räkna upp kanterna i viktordning: $O(e \log e)$.
- Avgöra om mängderna lika: $O(1)$ (räcker att jämföra pekare)
- Innehållet i if-satsen utförs max $v - 1$ gånger.
- Raderna 6-8 tar konstant tid för varje iteration.
- Raderna 9,10 tar totalt $O(v \log v)$ eftersom noderna från minsta mängden flyttas till den största. (Detta är inte uppenbart.)
- Totalt får vi: $O(e \log e + v \log v)$
- För icke multi-graf: $O((e + v) \log v)$
- Man kan skriva om komplexiteten ytterligare.