

# Minimale spenntrær, SCC, topologisk sortering.

**TDT4120, Algoritmer og datastrukturer**

**Institutt for Datateknikk og Informasjonsvitenskap**

**Simon Jonassen**

# Topologisk sortering

- DAG (*directed acyclic graph*) - rettet asyklisk graf. En rettet graf er asyklisk hvis og bare hvis den har ingen tilbakeførende kanter (back edges).
- Topologisk sortering – en (partiell) ordning av alle noder i en DAG, slik at alle noder som starter en kant listes opp før alle noder som avslutter den.

# Topologisk sortering

- En “naiv” algoritme,  $O(V^2)$ :
  - Så lenge det er flere noder, finn en node uten inngående kanter, fjern den, fjern alle utgående kanter.
- En bedre algoritme,  $O(V+E)$ :
  - Kjører DFS på alle noder i den reverserte grafen og legger dem i en liste når vi blir ferdige med dem.
- En DAG kan ha flere topologiske sorteringer.
- Numerisk sortering er en spesiell versjon av topologisk sortering med  $E=V^2/2$ .

# “Strongly Connected Components”

- SCC av en graf  $G = (V, E)$  er en maksimal mengde av alle hjørner,  $C \subseteq E$ , slik at alle hjørner er oppnåelige fra hverandre).
- Algoritme:
  - Kjør DFS på  $G$  og finn alle endetider.
  - Kjør DFS på  $G_t$ . Prioriter etter synkende endetid.
  - Resulterende dybde-først-skog separerer SCC'ene.
- ( $G_t$  er den transponerte/reverserte graf av  $G$ )

# Minimale spenntre

- Gitt en graf  $G = (V, E)$ . Et minimalt spenntre er en asyklisk delmengde av alle elementer  $(u, v) \in E$  som sammenkobler alle noder og har summen av alle kantvektene minimal.
- Veldig generelt:
  - Generisk-MST: Vi bygger ut en løsning ved å legge til “trygge” kanter.
  - Prim: Vi starter med en node og bygger ut løsningen ved å legge til “lette” kanter vi oppdager etter hvert.
  - Kruskal: Vi bygger ut løsningen ved å legge til kanter etter økende vekt uten å danne sykler.

# Minimale spenntre

- Prim( $G, w$ ):
  - La  $G=(E, V)$  være en graf, hvor  $E$  er alle kanter og  $V$  er alle noder i  $G$ . La  $w$  være en vektfunksjon.
  - La  $C$  være en tom delmengde av  $V$ , legg en vilkårlig node inn i  $C$ .
  - La  $A$  være en tom delmengde av  $E$ .
  - Fortsett inntil  $C=V$ :
    - Velg en kant  $e'=(u', v')$  av alle  $e=(u, v)$  hvor  $u \in C$  og  $v \notin C$  med  $w(e')$  minimal. Legg  $e'$  inn i  $A$ . Legg  $u'$  inn i  $C$ .
  - $A$  er et MST til  $G$ .
  
- *Det finnes flere litt forskjellige formuleringer til algoritmen (f.eks. s.572 i Cormen). Overnevnte er litt bedre mht. Teorem 23.1 i Cormen.*

# Minimale spenntrær

- Kruskal( $G, w$ ):
  - La  $G=(E, V)$  være en graf, hvor  $E$  er alle kanter og  $V$  er alle noder i  $G$ .
  - La  $A$  være en tom delmengde av  $E$ .
  - La  $K$  være en kopi av  $E$ .
  - Fortsett inntil  $K$  er tom:
    - Fjern en kant  $e$  med lavest vekt fra  $K$ .
    - Dersom  $e$  ikke danner sykler med alle elementer i  $A$ , legg  $e$  inn i  $A$
  - $A$  er et MST til  $G$ .
  
- *Det finnes flere litt forskjellige formuleringer til algoritmen (f.eks. s.569 i Cormen). Overnevnte er litt bedre mht. Teorem 23.1 i Cormen.*

# Minimale spenntrær

- Generic-MST( $G, w$ )
  - La  $G=(E, V)$  være en graf, hvor  $E$  er alle kanter og  $V$  er alle noder i  $G$ . La  $w$  være en vektfunksjon.
  - La  $A$  være en tom delmengde av  $E$ .
  - Fortsett inntil  $A$  er et spenntre til  $G$ .
    - finn en kant som er “*trygg*” for  $A$  og legg den inn i  $A$ .
  - $A$  er et MST til  $G$ .
  
- For hvert steg er  $A$  en delmengde av et MST til  $G$ . En kant er “*trygg*” hvis den kan legges til uten å bryte dette.

# Minimale spenntrær

- Prim( $G, w$ ), observasjoner:
  - under alle steg er  $A$  en delmengde av et MST.
  - under alle steg velger vi den korteste kanten av de som starter i  $C$  og slutter utenfor  $C$ .
- Kruskal( $G, w$ ), observasjoner:
  - under alle steg er  $A$  en delmengde av et MST.
  - under alle steg velger vi den korteste kanten som kobler sammen to ulike komponenter (deltrær) i  $A$ .
- Ifølge teoremene 23.1 og 23.2 er disse kantene “trygge”. Dermed velger begge algoritmene alltid en trygg kant som utvider den eksisterende løsning. Kruskal og Prim er i så fall bare to spesielle versjoner av Generisk MST.

# Kjøretidsanalyse, Prim og Kruskal

- Prim:
  - Oppsett (heap og “peker”-tabell):
    - Totalt:  $O(V)$
  - Ekstrahering av hjørner fra en heap:
    - Kompleksitet:  $O(\log V)$
    - Antall ganger:  $O(V)$
    - Totalt:  $O(V \log V)$
  - Nabo-oppslag, oppdatering av heap-innhold:
    - Kompleksitet:  $O(\log V)$
    - Antall ganger:  $O(E)$
    - Totalt:  $O(E \log V)$
  - Totalt:  $O(V + V \log V + E \log V) = O(V \log V + E \log V)$
- Forbedringer:
  - Bruk av Fibonacci-heap er  $O(1)$  ved oppdatering. Den totale kjøretiden blir dermed  $O(E + V \log V)$
  - Bruk av et usortert array istedet for heap gir kjøretiden på  $O(V^2)$

# Kjøretidsanalyse, Prim og Kruskal

- Kruskal:
  - Sortering:
    - Totalt:  $O(E \log E)$
  - Sykkeldeteksjon:
    - ... se kap 21.4 og 23.2 i Cormen.
    - Totalt:  $O((V+E) a V)$
  - Totalt:  $O(E \log V + (V + E) a V) = O(E \log V^2 + E \log V)$   
 $= O(E \log V)$