Eksamensdisposition - Minimum Spanning Trees (MST)

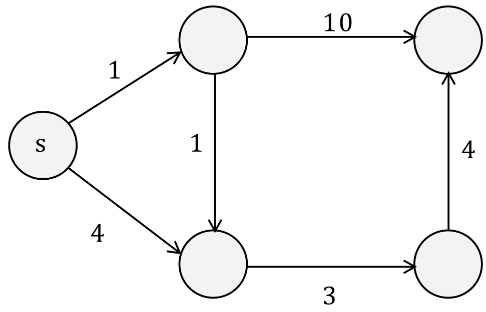
Søren Mulvad, rbn601

3. april 2019

- **Motivation samt beskrivelse af problemet**
- **Generisk løsning**
  - Terminologi: Cut, krydse et cut, respektere et sæt A af kanter, light kant, sikker kant.
- **Bevis af Theorem 23.1 (Genkendelse af sikre kanter)**
- **Håndkørsel af Kruskals algoritme**
- **Håndkørsel af Prims algoritme**

1 Minimum Spanning Trees

- **Motivation samt beskrivelse af problemet**
  - Løser det problem at forbinde en graf af knuder med den minimale omkostning.
  - Bliver ofte benyttet til f.eks. at designe elektriske kredsløb.
  - Kan modelleres som en graf $G = (V, E)$ hvor V er sættet af knuder og E er sættet af kanter. For hver kant $(u, v) \in E$ skal der være en vægtfunktion $w(u, v)$ som siger hvad omkostningen er.
  - Vi vil gerne finde det acyklise subset $T \subseteq E$ som forbinder alle knuderne, og minimerer den totale vægt.
  - Tegn en graf her hvor du beskriver terminologien ud fra, følgende er en god størrelse:



- **Generisk løsning**
  - Terminologi:
    - Cut: En partitionering $(S, V - S)$ af V i grafen $G = (V, E)$. Vil sige at vi deler grafen op i to subsets.
    - Krydse et cut: Når en kant (u, v) kryder cuttet $(S, V - S)$ er det fordi en af endepunkterne er i S og den anden er i $V - S$.
    - Respektere et sæt A af kanter: Når et cut ikke resulterer i, at nogle af kanterne i sættet A krydser cuttet.
    - Light kant: En kant, som både kryder et cut og hvis vægt er minimum af alle de kanter der kryder cuttet (NB: Der kan være flere for en iteration).
    - Sikker kant: En kant vi kan tilføje til subsettet A, som vedligeholder at A stadig er et subset af et Minimum Spanning Tree
  - Algoritme:

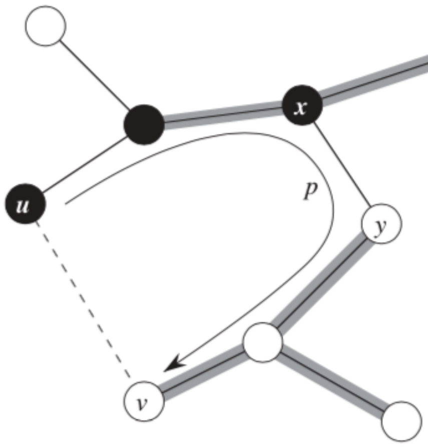
Algorithm 1: Generic-MST

`Generic-MST`(G, w)
    $A = \emptyset$
    while $A \neq$ *et Spanning Tree*
        find en kant (u, v) sikker for A
        $A = A \cup \{(u, v)\}$
    return A

  - Trivielt ser vi, at for hver iteration vil A være et subset af et MST da der kun tilføjes sikre kanter, og når algoritmen slutter er alle kanter tilføjet til A i et MST.

- **Bevis af Theorem 23.1 (Genkendelse af sikre kanter)**
  - *Def:* For grafen $G = (V, E)$ med vægtfunktion w defineret for E, så lad A være et subset af E som er inkluderet i et MST. Lad $(S, V - S)$ være ethvert cut af G som respekterer A, og lad (u, v) være en light kant der krydser dette cut.
    Så er kanten (u, v) sikker for A.

- Lad os nu sige at vi har et Minimum Spanning Tree T, som også inkluderer subsettet A, og antage at T ikke indeholder light kanten (u, v).
- *Mål:* Vi konstruerer nu et nyt MST T' som inkluderer $A \cup \{(u, v)\}$ ved at bruge en cut-and-paste teknik, hvorved vi viser at (u, v) er en sikker kant for A.



Figur 1: Sorte knuder i S, hvide knuder i $V - S$. Kanterne i MST'et T er vist, men ikke alle kanter i grafen G er vist. Kanterne i A er markeret og (u, v) er en light kant der krydser cuttet $(S, V - S)$.

- Vi laver nu et cut $(S, V - S)$ som respekter subsettet A.
- Kanten (u, v) danner en cyklus med kanterne på den simple vej p fra u til v i T, som set på Figur 1.
- Siden u og v er på modsatte sider af cuttet $(S, V - S)$, så må minimum en af kanterne i T være på den simple vej p og samtidig krydse cuttet. Lad os kalde den kant (x, y).
- Kanten (x, y) er ikke i A, siden cuttet respekterer A.
- Da vi har at (x, y) er en del af den unikke simple vej fra u til v, så vil det at fjerne den skære T over i to komponenter.
- Tilføjer vi nu (u, v) genforbinder vi komponenterne og får et nyt spanning tree

$$T' = T + \{(u, v)\} - \{(x, y)\}$$

- Herefter skal vi vise at T' er et Minimum Spanning Tree. Siden (u, v) er en light kant der kryder $(S, V - S)$ og (x, y) også krydser dette cut, så får vi:

$$\begin{aligned} w(T') &= w(T) + \overbrace{w(u, v) - w(x, y)}^{\leq 0 \text{ da } w(u,v) \leq w(x,y)} \\ &\leq w(T) \end{aligned}$$

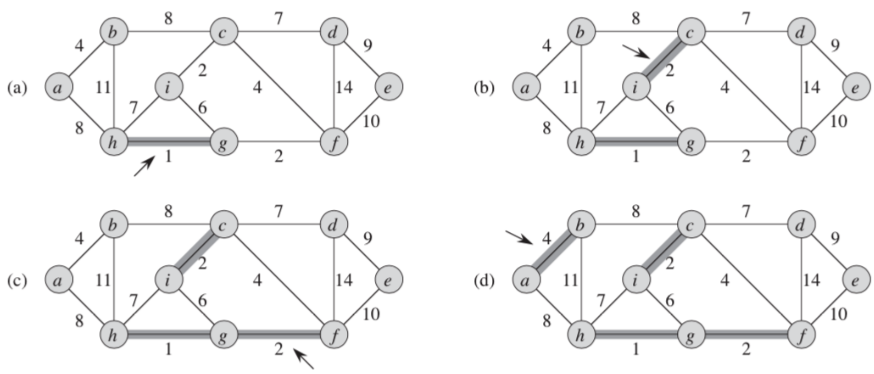
  Men vi havde før defineret T til at være et minimum spanning tree, så vi har også $w(T) \leq w(T')$, altså må T' også være et MSP.
- Vi skal stadig vise at (u, v) faktisk er en sikker kant for A.
  Da $A \subseteq T$ og $(x, y) \notin A$ må vi have at $A \subseteq T'$. Derfor må også $A \cup \{(u, v)\} \subseteq T'$.
  Siden T' er et minimum spanning tree må der gælde, at (u, v) er en sikker kant for A.

- **Håndkørsel af Kruskals algoritme**
  - Grådig algoritme *(Skriv ikke op, men for forståelse under forberedelse)*:

Algorithm 2: MST-Kruskal

```
MST-Kruskal(G, w)
    A = ∅
    foreach vertex v ∈ G.V
        Make-Set(v)
    sort the edges of G.E into increasing order by weight w
    foreach edge (u, v) ∈ G.E in sorted order
        if Find-Set(u) ≠ Find-Set(v)
            A = A ∪ {(u, v)}
            Union(u, v)
    return A
```

  - Altså:
    Start med et tomt sæt A
    Lav et nyt sæt for hver knude
    Sorter alle kanter efter stigende vægt w
    For hver eneste kant i sorteret rækkefølge, hvis de to knuder forbundet af kanten ikke er en del af samme sæt, så tilføj kanten til A og foren de to knuder.
  - Illustration:



Figur 2: Kruskals algoritme *(fortsætter, kun første fire steps)*

  - Køretiden afhænger af hvor hurtigt vi kan tjekke om to elementer er en del af samme sæt samt forene dem. Vi bruger disjoint set forests.

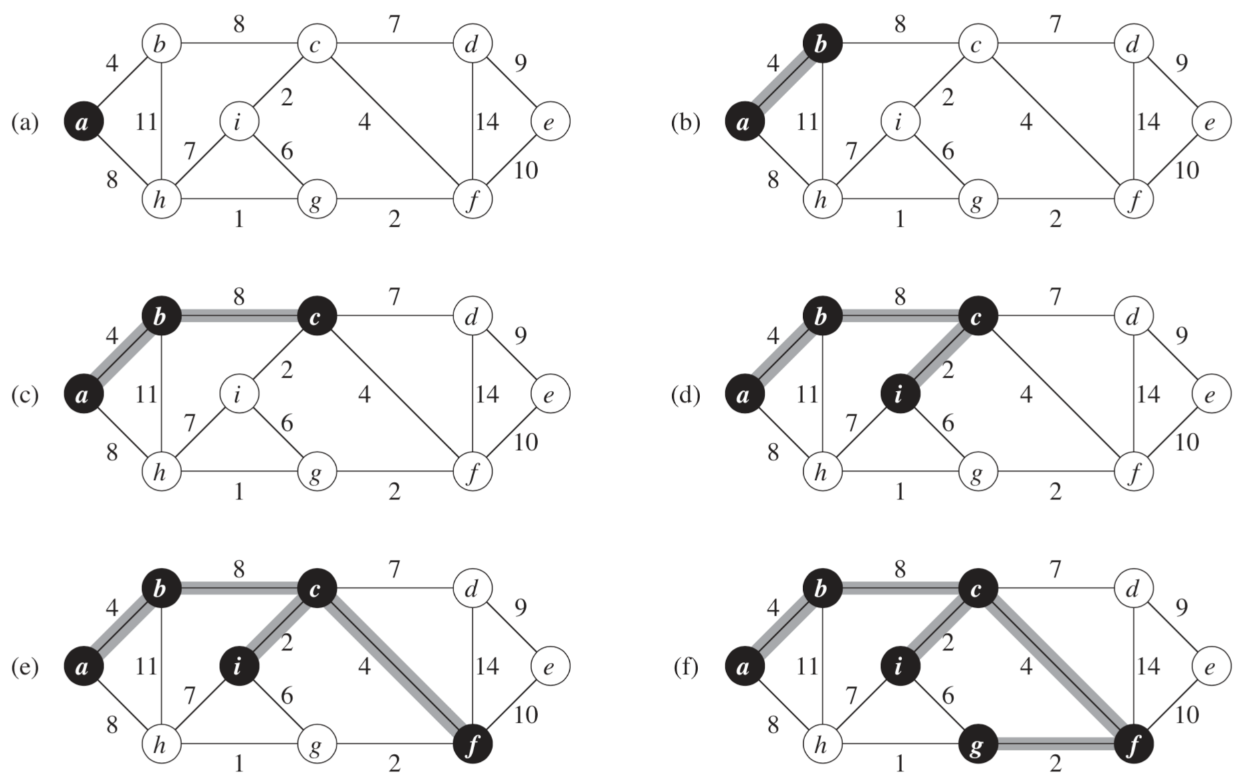
    Sortering af kanterne tager $O(E \lg E)$ tid. Efter det udfører vi $O(E)$ `Find-Set` og `Union`, som sammen med $|V|$ `Make-Set` operationer tager $O\Big((V+E)\ \alpha(V)\Big)$-tid.
    $\alpha(V)$ er en funktion der vokser *meget* langsomt

    Da vi antager at grafen er forbundet har vi $|E| \geq |V| - 1$, og derfor kan vi omskrive ovenstående til $O(E\alpha(V))$ tid.

    Vi har, at $\alpha(|V|) = O(\lg V) = O(\lg E)$, og derfor kan vi skrive køretiden fra det som $O(E \lg E) = O(E \lg V)$. Vi ser derfor, at algoritmen umiddelbart er upper-boundet af sorteringen. Da det er et MST vi skal finde, kan vi dog også antage, at det er en forbundet graf, hvorved $|E| \leq |V|^2$, så $\lg E = \lg V^2 = 2 \lg V = O(\lg V)$. Derfor kan vi skrive den endelige køretid som:

$$O(E \lg E) = O(E \lg V)$$

- **Håndkørsel af Prims algoritme**
  - Grådig algoritme
  - Vi starter ved en rodknude r og tilføjer herefter altid en light kant der krydser cuttet $(S, V - S)$. Har den effekt at vores subset A hele tiden er et sammenhængende træ.
  - Virker ved at holde styr på en $v.key$ og $v.\pi$ (parent-pointer) for alle knuder $v \in V$, som til at starte med sættes til henholdsvis ∞ og `NIL`. Herefter opdateres de omkringliggende kanter til den knude vi pt. kigger på såfremt værdierne er bedre.
  - Illustration:



Figur 3: Prims algoritme *(fortsætter, kun første seks steps)*.
Bemærk at vi i step (b)-(c) både kunne have valgt enten kanten (b, c) eller kanten (a, h).

  - Køretiden afhænger af hvor hurtigt vi kan køre `Extract-Min` og `Decrease-Key`. Derfor er det optimalt at bruge Fibonacci Heaps. Der er $|V|$ elementer at holde styr på, så hver `Extract-Min` operation vil tage $O(\lg n)$ amortiseret tid. Derudover udfører vi $2|E|$ `Decrease-Key` operationer, som hver amortiseret tager $O(1)$ tid. Altså vil den totale køretid blive:

$$O(V \lg V + E)$$