Eksamensdisposition - Shortest Path

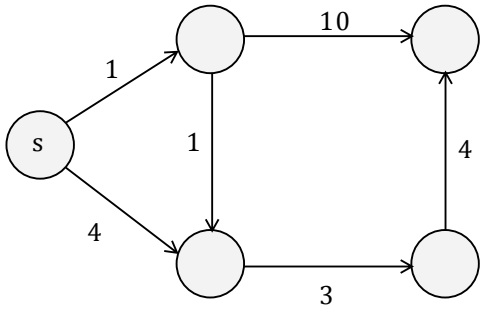
Søren Mulvad, rbn601

3. april 2019

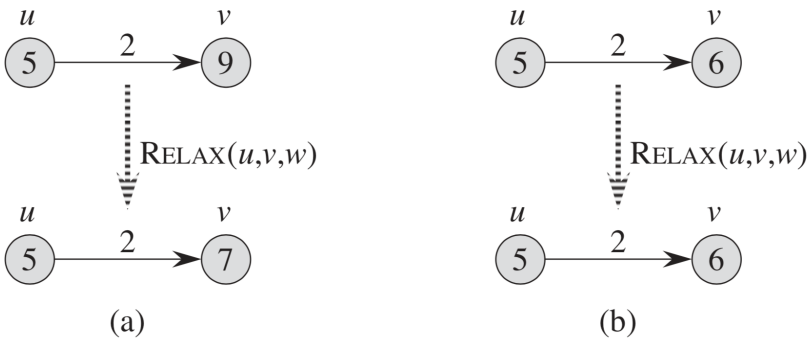
- **Motivation samt beskrivelse af problemet**
- **Init og Relaxation**
- **Bevis på at problemet udviser optimal delstruktur**
- **Håndkørsel af `Bellman-Ford` algoritmen**
- **Bevis af korrekthed af `Bellman-Ford` algoritmen**
  - Lemma 24.15 (Path-relaxation property)
  - Lemma 24.2 ("Kortest-vej-estimatet" er korrekt ved terminering)
  - Theorem 24.4 (Korrekthed)
- **Håndkørsel af `Dijkstra`'s algoritme**

1 Shortest Path

- **Motivation samt beskrivelse af problemet**
  - Åbenlyst utrolig brugbart. Kan bruges både bogstaveligt til korteste vej, men man kan også have en vægtfunktion der er baseret på tid, materialeomkostninger, etc.
    Jeg vil her tale om "Single Source Shortest Path", altså når vi ønsker at finde den korteste vej fra en kildeknude til alle andre knuder i grafen.
  - Givet en graf $G = (V, E)$ og en vægtfunktion $w(E)$, find da vejen p fra f.eks. knude u til v med den laveste samlede vægt $w(p)$ således at $w(p) = \delta(u, v)$.
  - Såfremt der ikke findes en vej fra u til v har vi defineret $\delta(u, v) = \infty$.
  - Tegn denne graf:



- **Init og Relaxation**
  - Vi indfører nu for alle knuder $v \in V$ en attribut $v.d$, som vi kan se som en upper bound for vægten af en kortest vej fra kildeknuden s til v. Denne kaldes "kortest-vej-estimat".
  - Derudover indfører vi attributten $v.\pi$ for alle knuder, som er dens forgænger (altså hvilken knude vi lige kom fra for at opnå den pt. korteste vej fundet).
  - `Init` sætter for alle knuder $v.d = \infty$ og $v.\pi = NIL$.
  - Vi indfører relaxtion for to knuder u og v der virker på den måde, at såfremt vi kan få et mindre "kortest-vej-estimat", så ændrer vi estimatet til dette og sætter v's forgænger til u på følgende måde:



Figur 1: Her relaxer vi kanten (u, v). I (a) kan vi forbedre "kortest-vej-estimateren" og opdaterer derfor værdierne for v, mens vi ikke ændrer noget i (b).

- **Bevis på at problemet udviser optimal delstruktur**
  - Lad $p = \langle v_0, v_1, ..., v_k \rangle$ være en kortest vej fra v_0 til v_k.
    For ethvert i og j således at

$$0 \leq i \leq j \leq k \qquad \text{lad} \qquad p_{ij} = \langle v_i, v_{i+1}, ..., v_j \rangle$$

være en delvej fra v_i til v_j. Så er p_{ij} en kortest vej fra v_i til v_j.

– *Bevis:* Lad os opløse vejen til

$$v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$$

Så har vi, at

$$w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$$

Lad os nu antage, at der findes en vej p'_{ij} fra v_i til v_j med en vægt $w(p'_{ij}) < w(p_{ij})$. Men så ville vi få, at

$$w(p) > w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$$

hvilket er en modstrid, da vi antog at p var den korteste vej fra v_0 til v_k.

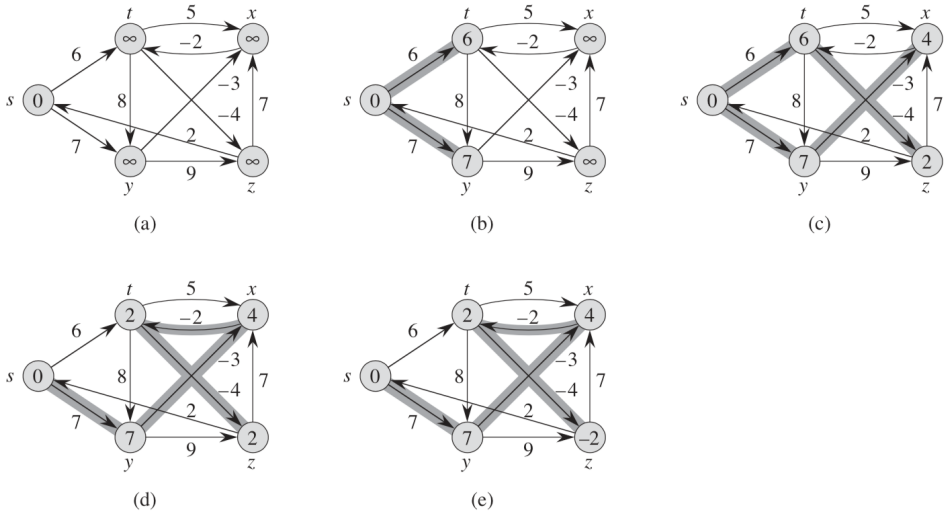
- **Håndkørsel af `Bellman-Ford`**

– Algoritme: *(Ikke skriv op, men for forståelse)*

Algorithm 1: MST-Kruskal

```
1 Bellman-Ford(G, w, s)
2     Initialize-Single-Source(G, s)
3     for i = 1 to |G.V| − 1
4         foreach edge (u, v) ∈ G.E
5             Relax(u, v, w)
6     foreach edge (u, v) ∈ G.E
7         if v.d > u.d + w(u, v)
8             return FALSE
9     return TRUE
```

Altså:
Initialiser alle vægte
Relax alle kanter $|V| - 1$ gange
Test for alle kanter (u, v) om $v.d > u.d + w(u, v)$ - hvis ja, returner `False`
Ellers returner `True`.



Figur 2: Illustration. Vær her opmærksom på, at kanterne relaxes i en bestemt rækkefølge og mellemresultaterne kunne se anderledes ud hvis de blev relaxet i en anden rækkefølge.

– Køretiden for Bellman-Ford er $O(VE)$, da det største bidrag kommer fra vi går igennem alle E kanter $V - 1$ gange.

- **Bevis af korrekthed af `Bellman-Ford`**
  - Tre dele: Path-relaxation property, Lemma 24.2, Theorem 24.4
  - Path-relaxation property
    * *Def:* Givet vi har en kortest vej $p = \langle v_0, v_1, ..., v_k \rangle$ fra $s = v_0$ til v_k. Så vil den sekvens af relaxation steps, som inkluderer i rækkefølge at relaxe

      $$(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$$

      medføre at $v_k.d = \delta(s, v_k)$. Dette gælder uanset om vi relaxer andre knuder indimellem.
    * *Bevis:* Induktionsbevis, hvor vi viser, at efter den i'te kant er relaxet har vi, at $v_i.d = \delta(s, v_i)$.
    * For vores basis $i = 0$ ser vi det er sandt, da $v_0.d = s.d = 0 = \delta(s, s)$.
    * For det induktive step antager vi, at $v_{i-1}.d = \delta(s, v_{i-1})$ og ser på hver der sker når vi relaxer kanten (v_{i-1}, v_i). Her har vi en konvergensegenskab jeg ikke vil bevise som da siger, at $v_i.d = \delta(s, v_i)$. Denne lighed er vedligeholdt for hele tiden fremover. Hermed er det bevist.
  - Lemma 24.2 ("Kortest-vej-estimatet" er korrekt ved terminering)
    * *Def:* Efter de $|V| - 1$ iterationer af for-loopet har vi, at $v.d = \delta(s, v)$ for alle knuder der kan nås fra kildeknuden (Vi antager at der ikke er nogle negative cykler).
    * *Bevis:* Tag udgangspunkt i enhver knude v som kan nås fra s, og lad $p = \langle v_0, v_1, ..., v_k \rangle$ hvor $v_0 = s$ og $v_k = v$ være en kortest vej fra s til v. Fordi korteste veje er simple, så har vi at p højest har $|V| - 1$ kanter, så:

      $$k \leq |V| - 1$$

      Vi relaxer netop alle kanter så mange gange, heriblandt kanten (v_{i-1}, v_i) i den i'te iteration. Ved at tage udgangspunkt i path-relaxation property har vi da ved terminering, at:

      $$v.d = \delta(s, v)$$
  - Theorem 24.4 (Korrekthed af `Bellman-Ford`)
    * *Def:* Hvis en graf G ikke har nogle negative cykler, så returnerer algoritmen `True` og har beregnet en korteste vej korrekt til alle knuder fra kildeknuden.
      Hvis G indeholder negative cykler, så returnerer algoritmen `False`.
    * *Bevis:* Først beviser vi det tilfælde, når G ikke har negative cykler. Da har vi lige bevist Lemma 24.2, og har altså derfor for alle knuder at $v.d = \delta(s, v)$. Vi har en "predecessor-subgraph property" som jeg ikke vil bevise, men af den følger, at så vil deres forgænger-attributer være sat korrekt og vi herved får den korteste vej. Hvis knuden ikke kan nås fra kildeknuden, så har vi en anden egenskab, "no path property", som viser at $v.d = \infty$ hvilket er korrekt jf. vores definition.
    * Herefter skal vi vise at den returnerer `True`.
      Når den terminerer har vi for alle knuder, at:

      $$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \qquad \text{(pga. trekantsuligheden)} \\ &= u.d + w(u, v) \end{aligned}$$

      Ingen af testene vil således returnere `False`, og derfor vil den returnere `True`.
    * Nu beviser vi, at den returnerer `False` når der er en negativ cyklus i G. Lad os kalde denne cyklus $c = \langle v_0, v_1, ..., v_k \rangle$, hvor $v_0 = v_k$. Så har vi at den totale vægt af c er mindre end 0 pr. definition.
      Antag nu for modstrid, at Bellman-Ford returnerer `True`. Altså vil

      $$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) \qquad \text{(for } i = 1, 2, ..., k)$$

Ved at summere ulighederne for cyklerne får vi:

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) \tag{1}$$

$$= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{2}$$

Men da vi har at $v_0 = v_k$ vil hver knude i c indgå præcis én gang i både hvad der er på venstresiden i Eq. (1) og venstre led i Eq. (2), og disse to udtryk er derfor lig hinanden. Derved får vi, at

$$0 \leq \sum_{i=1}^{k} w(v_{i-1,v_i})$$

Men vi antog netop, at vores cyklus var negativ, hvilket er en modstrid. Hermed er det bevist at algoritmen korrekt vil returnere `False` såfremt der er en negativ cyklus.

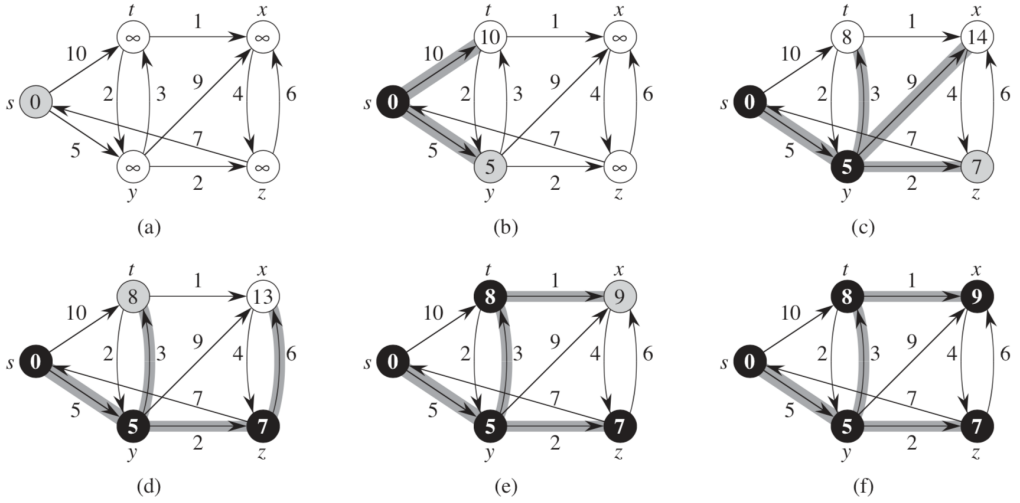
- **Håndkørsel af `Dijkstra`'s algoritme**
  - Antager alle kanter har en ikke-negativ vægt.
  - Algoritme: *(Ikke skriv op, men for forståelse)*

Algorithm 2: Dijkstra

```
1 Dijkstra(G, w, s)
2     Initialize-Single-Source(G, s)
3     S = ∅
4     Q = G.V
5     while Q ≠ ∅
6         u = Extract-Min(Q)
7         S = S ∪ {u}
8         foreach vertex v ∈ G.Adj[u]
9             Relax(u, v, w)
```

Altså:
Initialiser alle vægte
Løbende find knuden med lavest "kortest-vej-estimat" og relax alle dens kanter til omkringliggende knuder.

  - Illustration:



Figur 3: Dijkstras algoritme

– Køretiden afhænger af hvilken underliggende datastruktur vi bruger, hvor Fibonacci Heaps er mest optimalt. Vi udfører

  1. $|V|$ `Extract-Min` operationer som hver tager $O(\lg V)$ tid.
  2. $|E|$ `Decrease-Key` operationer (da vi kigger på alle kanter præcis én gang) som hver amortiseret tager $O(1)$ tid.

  Herved fås vi altså en samlet køretid:

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

- **Korrekthed af `Dijkstra`'s algoritme**
  - Det vi skal vise for at Dijkstra er korrekt, så skal vi vise, at når man tilføjer en knude til løsningsmængden S, så er det fordi vi allerede har fundet den korteste vej. Vi skriver løkkeinvarianten:

    Claim: Når u tilføjes til S har vi at $u.d = \delta(s, u)$.

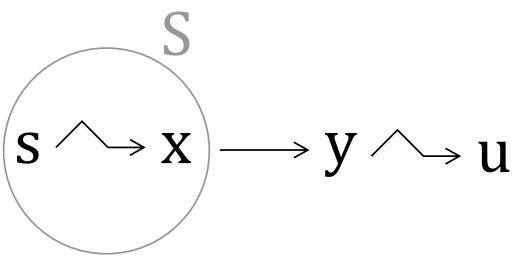
    Base case:

    $$s.d = \delta(s, s) = 0$$

    Induktionsstep:
    Vi bruger stærkt induktion, hvor vi antager at alle de knuder vi allerede har tilføjet til S er den korteste vej.
    Antag nu for modstrid at vi tilføjer knuden u, men $u.d \neq \delta(s, u)$ - hvorved vi får følgende pr. upper bound property:

    $$u.d > \delta(s, u)$$

    Der må findes en vej fra s til u, da hvis der ikke gjorde, så var den korteste vej uendelig pr. upper bound property (da hvis der ikke gjorde, så kunne den aldrig nogensinde blive uendelig pga. no path property).
    Når vi tilføjede den, så ville den altså allerede være lig den korteste vej. Så lad os sige at den korteste vej går gennem kanten (x, y). Vi definerer y til at være den første knude i stien fra s til u som ikke allerede er tilføjet til løsningsmængden S:

    Dvs. at x (som evt. godt kan være s) er i vores løsningssæt S, som pr. den stærke induktionsantagelse må have, at $x.d = \delta(s, x)$. Vi ved derudover, at da vi tilføjede x, så har vi relaxet alle udadgående kanter.
    Pr. convergence property betyder det, at $y.d = \delta(s, y)$. Derudover ved vi, at y kommer før u i den korteste vej til u. Da vi ikke har nogle negative kanter, så må den korteste vej til y må være mindre end eller lig den korteste vej fra s til u, hvilket er mindre end den distance vi har fundet til u pr. vores modstridsantagelse:

    $$y.d = \delta(s, y) \leq \delta(s, u) < u.d$$

    Vi har nu både y og u, og hvis de var ens, så havde vi allerede fundet den korteste vej til u når vi tilføjer den. Derfor må de være forskellige. Men vi har tænkt os at tilføje u til S, dvs.

    $$u.d \leq y.d$$

    (ellers ville vi være igang med at tilføje y pr. Dijkstras algoritme).
    Vi ser nu, at vi får følgende ulighed hvilket er en modstrid, og derfor er Dijkstra korrekt:

    $$y.d < u.d \leq y.d$$

- **Egenskaber**
  - **Trekantsulighed:** For enhver kant $(u, v) \in E$ har vi at $\delta(s, v) \leq \delta(s, u) + w(u, v)$.
  - **Upper-bound property:** Vi har altid at $v.d \geq \delta(s, v)$ for alle knuder $v \in V$, og når $v.d$ får værdien $\delta(s, v)$ ændres den aldrig.
  - **No-path property:** Hvis der ikke er nogen vej fra s til v, så har vi altid at $v.d = \delta(s, v) = \infty$.
  - **Konvergensegenskab:** Hvis $s \rightsquigarrow u \rightarrow v$ er en kortest vej for $u, v \in V$ og $u.d = \delta(s, u)$ på ethvert tidspunkt før vi relaxer kanten (u, v), så vil $v.d = \delta(s, v)$ efter vi relaxer og for altid fremover.
  - **Path-relaxation property:** Hvis $p = \langle v_0, v_1, ..., v_k \rangle$ er en kortest vej fra $s = v_0$ til v_k, og vi relaxer kanterne af p i rækkefølgen $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$, så vil $v_k.d = \delta(s, v_k)$. Dette holder uanset hvad vi relaxer i mellemtiden.
  - **Predecessor subgrapgh-property:** Når $v.d = \delta(s, v)$ for alle $v \in V$, så er forgænger-delgrafen et kortest vej træ med rod i s.