10. Korteste vei fra én til alle

Forelesning 10

Bredde-først-søk kan finne stier med færrest mulig kanter, men hva om kantene har ulik lengde? Det generelle problemet er uløst, men vi kan løse problemet med gradvis bedre kjøretid for grafer uten negative sykler, uten negative kanter, og uten sykler. Og vi bruker samme prinsipp for alle tre!

Pensum

- ☐ Kap. 22. Single-source shortest paths: Innl. og 22.1–22.3

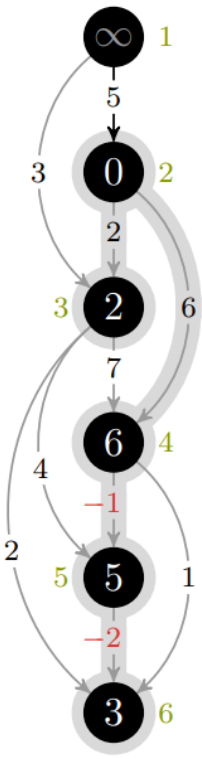
Læringsmål

- [J1] Forstå varianter av *korteste-vei*-problemet
- [J2] Forstå strukturen til problemet
- [J3] Forstå at negative sykler gir mening for korteste *enkle vei*
- [J4] Forstå at *korteste* og *lengste enkle vei* kan løses vha. hverandre
- [J5] Forstå *korteste-vei-trær*
- [J6] Forstå *kant-slakking* og RELAX
- [J7] Forstå ulike egenskaper ved korteste veier og slakking
- [J8] Forstå BELLMAN-FORD
- [J9] Forstå DAG-SHORTEST-PATH
- [J10] Forstå kobling mellom DAG-SHORTEST-PATHS og DP
- [J11] Forstå DIJKSTRA

Når vi ønsker å finne korteste vei fra én til alle, ønsker vi å finne de korteste stiene fra en gitt node til alle andre noder. I tillegg vet vi at:

- En **enkel sti** er en sti uten sykler
- En **kortest sti** er alltid **enkel**
- Hvis det finnes en negativ sykel i veien, finnes det ingen **kortest sti**

- Det finnes fortsatt en **kortest enklest sti** (utenfor syklen)
- Å finne denne er et uløsp NP-hardt problem.



Input: En rettet graf $G = (V, E)$, vektfunksjon $w : E \to \mathbb{R}$ og node $s \in V$.

Output: For hver node $v \in V$, en sti $p = \langle v_0, v_1, \ldots, v_k \rangle$ med $v_0 = s$ og $v_k = v$, som har minimal vektsum

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

💻Dag-Shortest-Paths

Dette er en simpel algoritme for å finne de korteste stiene til alle nodene fra en gitt node gitt at det ikke finnes noen sykler (positive eller negative) i nodene som kan nås fra startnoden. Dette er fordi algoritmen bruker en topologisk sortering når den traverserer nodene i grafen.

```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, in topsort order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

Operasjon	Antall	Kjøretid
Topologisk sortering	1	$\Theta(V + E)$
Initialisering	1	$\Theta(V)$
RELAX	E	$\Theta(1)$

Totalt: $\Theta(V + E)$

💻Bellman-Ford

Hvis vi har sykler grafen som kan nås fra startnoden kan dette løses på flere måter. Vi kan holde styr på hvilke noder som endres ved iterasjon, eller oppdatere alle kantene til ingenting endrer seg.

Teoretisk sett, skal ingenting endre seg etter $|V-1|$ iterasjoner, siden hver node i grafen skal ha blitt besøkt så langt. Hvis ting endrer seg etter så mange iterasjoner må det finnes en negativ sykel.

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
RELAX	$V-1$	$\Theta(E)$
RELAX	1	$O(E)$

Totalt: $\Theta(VE)$

💻Dijkstras algoritme

Dijkstras algoritme tar utgangspunkt i at noden med lavest estimat må være ferdigkalkulert. Den eneste måten vekten til stien kan bli bedre er om vi hadde negative kanter. Dijkstras algoritme fungerer dermed ikke hvis grafen har negative kanter.

```
DIJKSTRA(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S = ∅
 3  Q = ∅
 4  for each vertex u ∈ G.V
 5       INSERT(Q, u)
 6  while Q ≠ ∅
 7       u = EXTRACT-MIN(Q)
 8       S = S ∪ {u}
 9       for each vertex v ∈ G.Adj[u]
10            RELAX(u, v, w)
11            if RELAX decreased v.d
12                 DECR-KEY(Q, v, v.d)
```

Operasjon	Antall	Kjøretid
Initialisering	1	$\Theta(V)$
BUILD-HEAP	1	$\Theta(V)$
EXTRACT-MIN	V	$O(\lg V)$
DECREASE-KEY	E	$O(\lg V)$

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

Kjøretiden kan videre optimaliseres ved å bruke fibonacci-heaps.