11. Korteste vei fra alle til alle

Forelesning 11

Vi kan finne de korteste veiene fra hver node etter tur, men mange av delproblemene vil overlappe – om vi har mange nok kanter lønner det seg å bruke dynamisk programmering med dekomponeringen «Skal vi innom k eller ikke?»

Pensum

- [ ] Kap. 23. All-pairs shortest paths

Læringsmål

[K$_1$] Forstå *forgjengerstrukturen* for *alle-til-alle*-varianten av korteste vei-problemet, inkl. PRINT-ALL-PAIRS-SHORTEST-PATH

[K$_2$] Forstå SLOW-APSP og FASTER-APSP

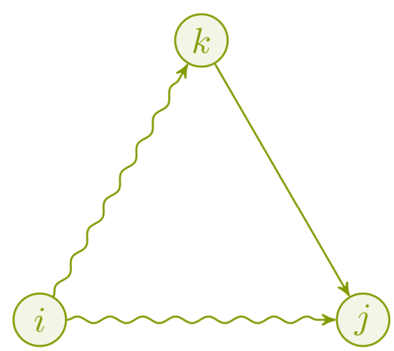
[K$_3$] Forstå FLOYD-WARSHALL

[K$_4$] Forstå TRANSITIVE-CLOSURE

[K$_5$] Forstå JOHNSON

Trekantulikheten kan komme til nytte for å forstå hvordan den korteste veien kan finnes. Hvis det finnes en snarvei fra $i \rightarrow j$ som går gjennom k, så har vi ikke enna funnet den korteste veien fra $i \rightarrow j$.

$$\overline{AC} \leq \overline{AB} + \overline{BC}$$



Varianten vi har brukt: $\delta(i,j) \leqslant \delta(i,k) + w(k,j)$

$Relax$ prosedyren vår sjekker om denne snarveien er kortere, og reparerer/gjeninnfører trekantulikheten. Når vi er ferdige, kan ikke $i \rightarrow k \rightarrow j$ fortsatt være en snarvei. ???

1. $\omega(k,j) + \delta(i,k) - \delta(i,j) \geq 0$

$$\begin{aligned}
&\textbf{if } v.d > u.d + w(u,v) \\
&\qquad v.d = u.d + w(u,v) \\
&\qquad v.\pi = u
\end{aligned}$$

$Relax(u, v, \omega)$

2. Hvis $i \to k$ og $k \to j$ finnes, finnes $i \to j$
3. $\delta(i,k) + \delta(k,j) \geq \delta(i,j)$

💻Johnsons algoritme

Johnsons algoritme er en simpel måte å finne korteste vei fra alle til alle i en graf. Den bruker i all hovedsak Dijkstras algoritme for å finne korteste vei fra én til alle over alle nodene i grafen.

Input: En vektet, rettet graf $G = (V, E)$ uten negative sykler, der $V = \{1, \ldots, n\}$, og vektene er gitt av matrisen $W = (w_{ij})$.

Output: En $n \times n$-matrise $D = (d_{ij})$ med avstander, dvs., $d_{ij} = \delta(i,j)$.

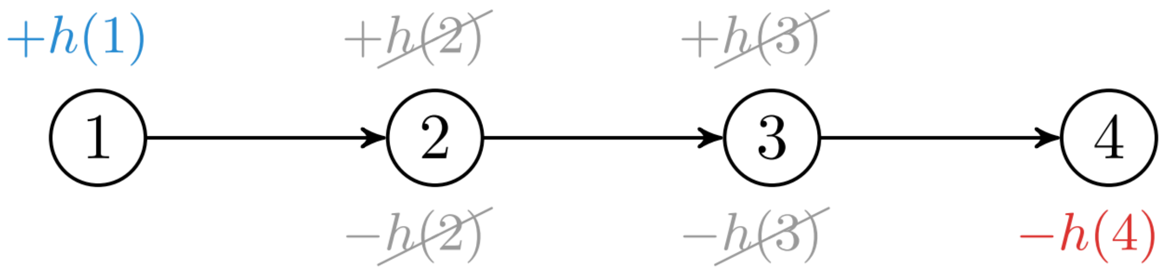
Det er også vanlig å returnere en forgjengermatrise $\Pi = (\pi_{ij})$

Ettersom Dijkstras algoritme ikke fungerer med negative kanter, må vi balansere alle kantene i grafen helt til det ikke finnes negative kanter. Etter utregningen vår, kan vi omjustere igjen for å finne de originale vektene.

> Beskrivelse bruker Turing-reduksjoner, forelesning 13-14 begrenser dette til Karp-reduksjoner.

Instans (i) En rettet graf $G = (V, E)$ med vekting w	Løsning (i) Avstand $\delta(u, v)$ for alle noder $u, v \in V$
Reduksjon $(i \to ii)$ $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$ Sørg for $h(v) \leqslant h(u) + w(u,v)$	Rekonstruksjon $(ii \to i)$ $\delta(u,v) = \hat{\delta}(u,v) + h(v) - h(u)$
Instans (ii) En rettet graf G med vekting $\hat{w} \geqslant 0$	Løsning (ii) Avstand $\hat{\delta}(u,v)$ for alle noder $u, v \in V$
Reduksjon $(ii \to iii)$ Bruk hver node $u \in V$ som startnode	Rekonstruksjon $(iii \to ii)$ Samle delløsninger
Instanser (iii) En rettet graf G med vekting $\hat{w} \geqslant 0$ og startnode u	Løsninger (iii) Avstand $\hat{\delta}(u,v)$ for alle noder $v \in V$

Når vi skal øke kantvektene, kan det virke intuitivt å øke alle kantene med like mye, men stier med mange kanter taper på denne økningen. Vi vil heller gi hver node en tilordnet verdi $h(node)$. Vekten $\omega(v, u)$ økes med differansen $h(u) - h(v)$. På denne måte vil positive og negative ledd bortsett fra det første og det siste oppheve hverandre.



For å sikre oss at vi ikke ender opp med noen negative vekter, kan vi igjen huske på trekantulikheten som forteller oss at $\omega(k, j) + \delta(i, k) - \delta(i, j) \geq 0$. Vi kan rett og slett legge til en ny node.

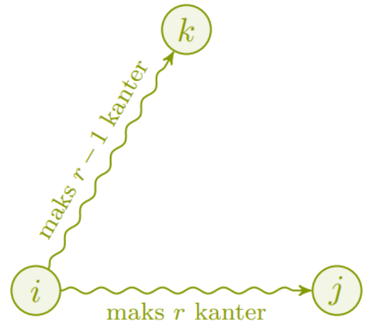
Algoritmen kjører ved å velge en tilfeldig startnode s fra grafen, og deretter kjøre $BellmannFord$ for å finne ut om grafen har noen negative sykler. Deretter definerer vi differansefunksjonen $h(v)$ for enhver node basert på distansene i resultatet fra $BellmanFord$. Vi lager deretter $\hat{w}(u, v)$ med den nye vektingen til hver kant, justert med differansen $h(u) - h(v)$. Deretter kjører vi $Dijkstra$ på hver node i grafen.

```
JOHNSON(G, w)
construct G' with start node s
BELLMAN-FORD(G', w, s)
for each vertex v ∈ G.V
    h(v) = v.d
for each edge (u, v) ∈ G.E
    ŵ(u, v) = w(u, v) + h(u) − h(v)
let D = (d_uv) be a new n × n matrix
for each vertex u ∈ G.V
    DIJKSTRA(G, ŵ, u)
    for each vertex v ∈ G.V
        d_uv = v.d + h(v) − h(u)
return D
```

💻”Matriseprodukt”

Denne algoritmen har egentlig ingen relasjon til matriseprodukt å gjøre, bortsett fra at den ene prosedyren ligner litt.

Instans En rettet graf $G = (V, E)$ med vekting w Korteste veier har maks $r = \lvert V \rvert - 1$ kanter	Løsning Stilengde $l_{ij}^{(r)} = \delta(i, j)$ for alle noder $i, j \in V$
Dekomponering Reduser maks antall kanter med 1	Kombinasjon IS $l_{ij}^{(r)} = \min\{l_{ik}^{(r-1)} + w(k, j) : k \in V\}$
Delinstanser En rettet graf $G = (V, E)$ med vekting w Korteste veier får maks bruke $r - 1$ kanter	Delløsninger IH Stilengde $l_{ij}^{(r-1)}$ for alle noder $i, j \in V$
Grunntilfelle $r = 0$	Grunnløsning 0 fra en node til seg selv; ∞ ellers



Induktiv hypotese: Vi har alle avstander for $r - 1$

I denne algoritmen innfører vi et parameter r som er antall kanter vi har lov til å besøke på stien fra $i \to j$. Vi antar induktivt at løst det for alle avstander $r - 1$, og kan dermed finne $l_{ij}^{(r)} = min_k l_{ij}^{(r-1)} + \omega(k, j)$.

Induktivt vil vi dermed finne den korteste stien ved dette "punktet" i grafen.

```
EXTEND-SHORTEST-PATHS(L, W, L', n)
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            l'_ij = min{l'_ij, l_ik + w_kj}
```

```
SLOW-APSP(W, L^(0), n)
new n × n matrices: L, M
L = L^(0)
for r = 1 to n − 1
    M = ∞
    EXTEND-SP(L, W, M, n)
    L = M
return L
```

Desverre er kjøretiden på denne algoritmen $O(n^4)$. (Kanskje derfor den heter "Slow"). Bedre versjon bruker repeated squaring.

Dette resulterer i en bedre kjøretid på $\Theta(n^3 \lg n)$

```
FASTER-APSP(W, n)
new n × n matrices: L, M
L = W
r = 1
while r < n − 1
    M = ∞
    EXTEND-SP(L, L, M, n)
    r = 2r
    L = M
return L
```

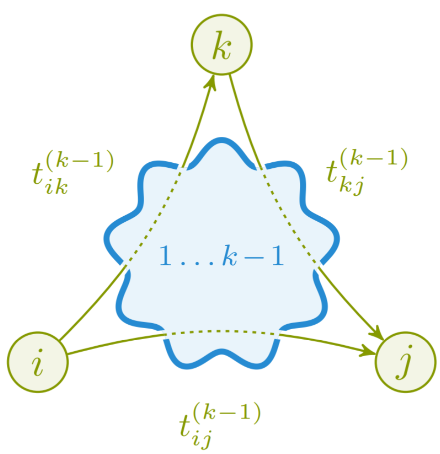
Transitiv lukning

Gitt en graf, så ønsker vi å finne ut om enhver node i kan nås fra alle de andre nodene. Her kan vi igjen løse induktivt ved å bruke trekantulikheten.

Finnes det en sti $i \to j$ som får gå innom nodene $1, \ldots, n$, dvs. alle?

Input: En rettet graf $G = (V, E)$.

Output: En rettet graf $G^* = (V, E^*)$ der $(i, j) \in E^*$ hvis og bare hvis det finnes en sti fra i til j i G.



$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Instans En rettet graf $G = (V, E)$ Stier kan gå innom noder $V = \{1, \ldots, k\}$	Løsning $t_{ij}^{(k)} \iff i \rightsquigarrow j$, for alle noder $i, j \in V$
Dekomponering Reduser maks-nodenummer med 1	Kombinasjon IS $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$
Delinstanser En rettet graf $G = (V, E)$ Stier får bare gå innom noder $\{1, \ldots, k-1\}$	Delløsninger IH $t_{ij}^{(k-1)} \iff i \rightsquigarrow j$ via $1 \ldots k-1$
Grunntilfelle $k = 0$	Grunnløsning 1 fra en node til seg selv; 0 ellers

For enhver vei fra $i \to j$, kan vi enten velge den direkte strekningen $i \to j$, eller så kan vi velge $i \to j \to k$.

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)})$$

Forenklet implementasjon av algoritmen til høyre kan implementeres med en enkel tabell. Man risikerer å gå innom samme node flere ganger, men det betyr bare at vi løste et senere delproblem tidligere.

Dessuten om det finnes stier med sykler, så finnes det også en uten.

$\textsc{Transitive-Closure}(G, n)$

let $T^{(0)} = (t_{ij}^{(0)})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
  for $j = 1$ **to** n
    if $i == j$ or $(i, j) \in G.E$
      $t_{ij}^{(0)} = 1$
    else $t_{ii}^{(0)} = 0$

for $k = 1$ **to** n
  let $T^{(k)} = (t_{ij}^{(k)})$ be a new $n \times n$ matrix
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $t_{ij}^{(k)} = t_{ij}^{(k-1)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)})$
return $T^{(n)}$

$\textsc{Transitive-Closure}'(G, n)$

initialize T
for $k = 1$ **to** n
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $t_{ij} = t_{ij} \lor (t_{ik} \land t_{kj})$
return T

Totalt blir kjøretiden på $TransitiveClosure$ $\Theta(n^3)$. Analyse fra forelesning:

$$
\begin{array}{l}
\text{init} \triangleright \Theta(n^2) \\
\textbf{for } i = 1 \textbf{ to } n \\
\quad \textbf{for } j = 1 \textbf{ to } n \\
\quad\quad \text{sett } t_{ij}^{(0)} \triangleright \mathrm{O}(1) \\
\textbf{for } k = 1 \textbf{ to } n \\
\quad \text{evt. ny matrise} \triangleright \Theta(n^2) \\
\quad \textbf{for } i = 1 \textbf{ to } n \\
\quad\quad \textbf{for } j = 1 \textbf{ to } n \\
\quad\quad\quad \text{sett } t_{ij}^{(k)} \triangleright \mathrm{O}(1) \\
\textbf{return} \triangleright \mathrm{O}(1)
\end{array}
$$

$$
\text{Totalt: } \Theta(n^3)
$$

💻Floyd-Warshall

Vi har allerede sett på måter vi kan løse alle til alle problemet med Dijkstras algoritme over hver node, men vi vil gjøre det mulig å kjøre algoritmen flere typer grafer.

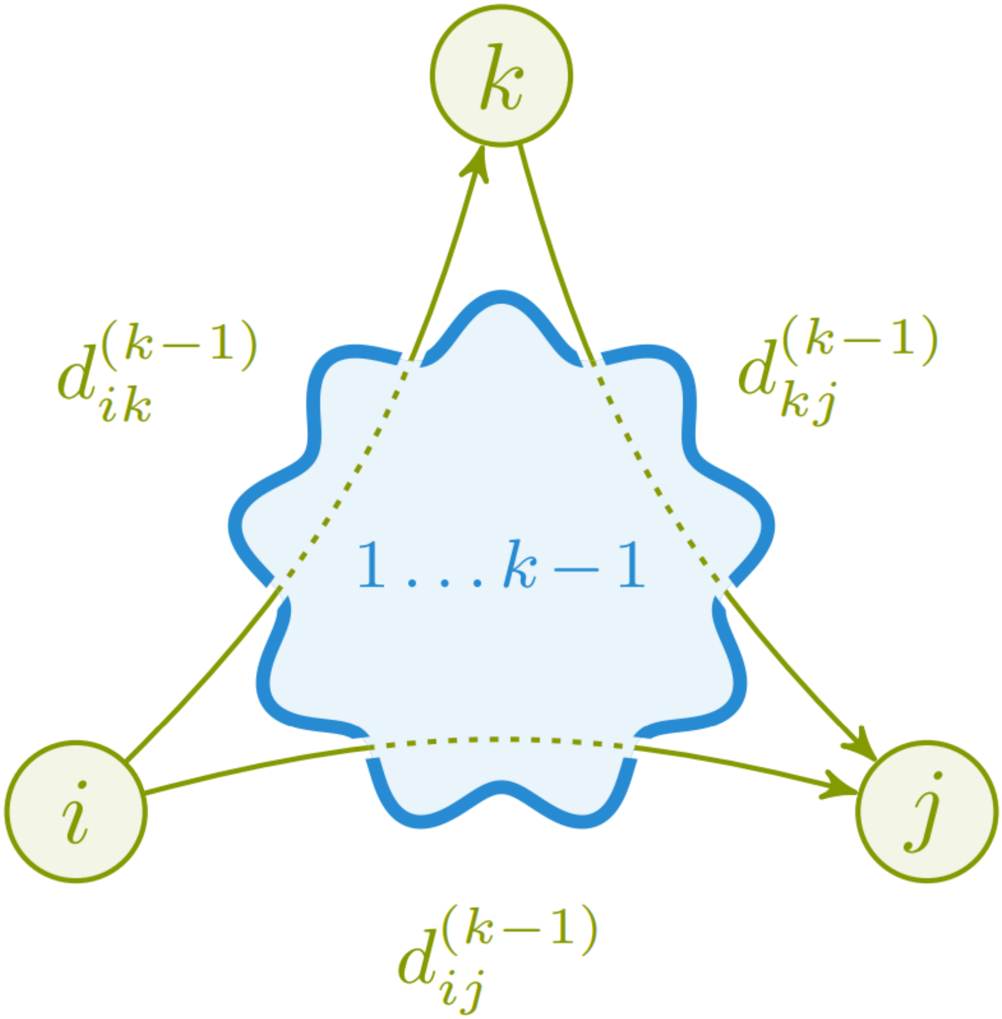
Fra hver node til alle andre?

- DIJKSTRA med tabell: $O(V^3 + VE)$
- ...med binærhaug: $O(VE \lg V)$
- ...med Fib.-haug: $O(V^2 \lg V + VE)$
- BELLMAN-FORD: $\Theta(V^2 E)$

Målsetting:

- Vi vil tillate negative kanter
- Vi vil ha lavere asymptotisk kjøretid...
- ...og vil ha lavere konstantledd

Ved å bygge videre på prinsippet fra transitiv lukning kan vi bruke samme måte på å velge den korteste av de to veiene som forgjenger til en hver node. Det er akkurat dette $FloydWarshall$ gjør.



$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty\,, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty\,. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,. \end{cases}$$

Instans En rettet graf $G = (V, E)$ med vekting w Stier kan gå innom noder $V = \{1, \dots, k\}$	Løsning Stilengde $d_{ij}^{(k)} = \delta(i, j)$ for alle $i, j \in V$
Dekomponering Reduser maks-nodenummer med 1	Kombinasjon IS $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
Delinstanser En rettet graf $G = (V, E)$ med vekting w Stier får bare gå innom noder $\{1, \dots, k-1\}$	Delløsninger IH Stilengde $d_{ij}^{(k-1)}$ for alle $i, j \in V$
Grunntilfelle $k = 0$	Grunnløsning 0 fra en node til seg selv; $w(i, j)$ ellers

```
FLOYD-WARSHALL'(W, n)
initialize D and Π
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            if d_ij > d_ik + d_kj
                d_ij = d_ik + d_kj
                π_ij = π_kj
return D, Π
```

Dette resulterer i en total kjøretid på $\Theta(n^3)$. I tillegg kan det være nyttig å kunne printene veiene fra en node til en annen. Dette kan enkelt gjøres ved å besøke forgjengermatrisen.

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
if i == j
    print i
elseif π_ij == NIL
    print "no path from" i "to" j "exists"
else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
    print j
```