

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

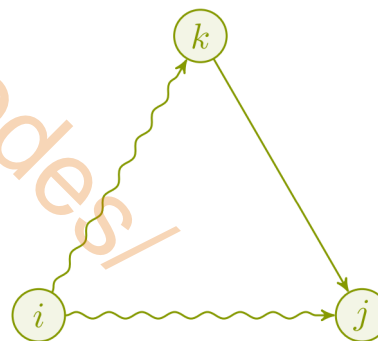
- Kap. 23. All-pairs shortest paths

Læringsmål

- [K₁] Forstå *forgjengerstrukturen* for *alle-til-alle*-varianten av korteste vei-problemet, inkl. PRINT-ALL-PAIRS-SHORTEST-PATH
- [K₂] Forstå SLOW-APSP og FASTER-APSP
- [K₃] Forstå FLOYD-WARSHALL
- [K₄] Forstå TRANSITIVE-CLOSURE
- [K₅] 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: $\delta(i, j) \leq \delta(i, k) + w(k, j)$

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. ???

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

if $v.d > u.d + w(u, v)$
 $v.d = u.d + w(u, v)$
 $v.\pi = u$

Relax(u, v, ω)

2. Hvis $i \rightarrow k$ og $k \rightarrow j$ finnes, finnes $i \rightarrow j$
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 $V = \{1, \dots, n\}$, 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 $\Pi = (\pi_{ij})$

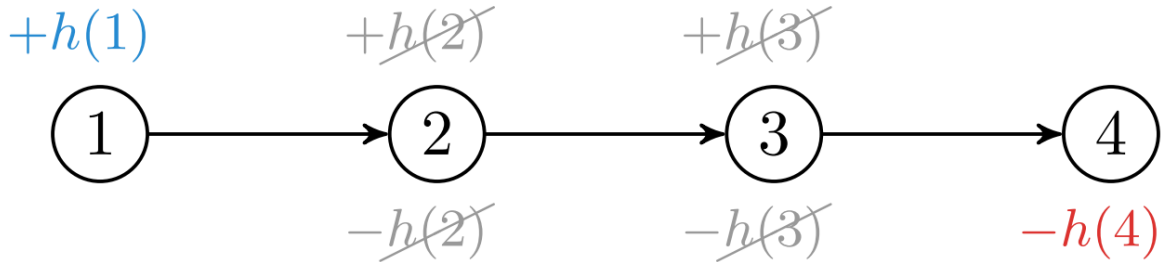
Ettersom Dijkstras algoritme ikke fungerer med negative kanter, må vi balansere alle kantene i grafen helt til det ikke finnes negative kanter. Etter utregningen vår, kan vi omjustere igjen for å finne de originale vektene.

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 \rightarrow ii) $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ Sørg for $h(v) \leq h(u) + w(u, v)$	Rekonstruksjon (ii \rightarrow i) $\delta(u, v) = \hat{\delta}(u, v) + h(v) - h(u)$
Instans (ii) En rettet graf G med vekting $\hat{w} \geq 0$	Løsning (ii) Avstand $\hat{\delta}(u, v)$ for alle noder $u, v \in V$
Reduksjon (ii \rightarrow iii) Bruk hver node $u \in V$ som startnode	Rekonstruksjon (iii \rightarrow ii) Samle delløsninger
Instanser (iii) En rettet graf G med vekting $\hat{w} \geq 0$ 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 $h(\text{node})$. 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 sslett legge til en ny node.

Algoritmen kjører ved å velge en tilfeldig startnode s fra grafen, og deretter kjøre *BellmanFord* 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)

```

1  construct  $G'$  with start node  $s$ 
2  BELLMAN-FORD( $G', w, s$ )
3  for each vertex  $v \in G.V$ 
4     $h(v) = v.d$ 
5  for each edge  $(u, v) \in G.E$ 
6     $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ 
7  let  $D = (d_{uv})$  be a new  $n \times n$  matrix
8  for each vertex  $u \in G.V$ 
9    DIJKSTRA( $G, \hat{w}, u$ )
10   for each vertex  $v \in G.V$ 
11      $d_{uv} = v.d + h(v) - h(u)$ 
12  return  $D$ 

```

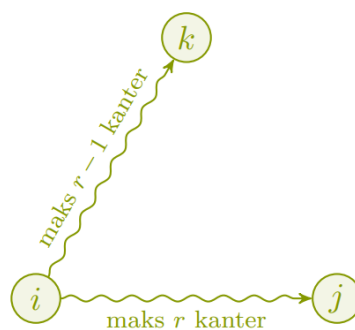
"Matriseprodukt"

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

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

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

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



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

EXTEND-SHORTEST-PATHS(L, W, L', n)

```

1  for  $i = 1$  to  $n$ 
2    for  $j = 1$  to  $n$ 
3      for  $k = 1$  to  $n$ 
4         $l'_{ij} = \min\{l'_{ij}, l_{ik} + w_{kj}\}$ 

```

SLOW-APSP($W, L^{(0)}, n$)

```

1  new  $n \times n$  matrices:  $L, M$ 
2   $L = L^{(0)}$ 
3  for  $r = 1$  to  $n - 1$ 
4     $M = \infty$ 
5    EXTEND-SP( $L, W, M, n$ )
6     $L = M$ 
7  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)

```

1  new  $n \times n$  matrices: L, M
2  L = W
3  r = 1
4  while  $r < n - 1$ 
5      M =  $\infty$ 
6      EXTEND-SP(L, L, M, n)
7      r = 2r
8      L = M
9  return L

```

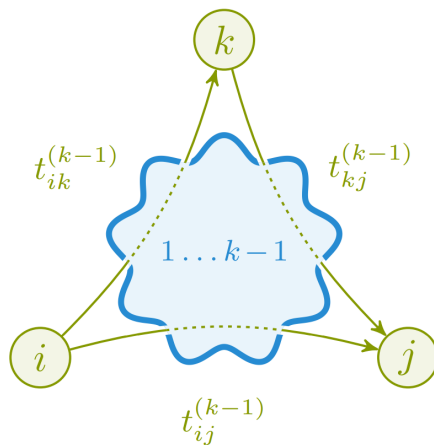
Transitiv lukning

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

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

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

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.

TRANSITIVE-CLOSURE(G, n)

```

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

7  for  $k = 1$  to  $n$ 
8      let  $T^{(k)} = (t_{ij}^{(k)})$  be a new  $n \times n$  matrix
9      for  $i = 1$  to  $n$ 
10         for  $j = 1$  to  $n$ 
11              $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$ 
12  return  $T^{(n)}$ 
```

TRANSITIVE-CLOSURE'(G, n)

```

1  initialize T
2  for  $k = 1$  to  $n$ 
3      for  $i = 1$  to  $n$ 
4          for  $j = 1$  to  $n$ 
5               $t_{ij} = t_{ij} \vee (t_{ik} \wedge t_{kj})$ 
6  return T
```

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

```

init  $\succ \Theta(n^2)$ 
for  $i = 1$  to  $n$ 
    for  $j = 1$  to  $n$ 
        sett  $t_{ij}^{(0)} \succ O(1)$ 
for  $k = 1$  to  $n$ 
    evt. ny matrise  $\succ \Theta(n^2)$ 
    for  $i = 1$  to  $n$ 
        for  $j = 1$  to  $n$ 
            sett  $t_{ij}^{(k)} \succ O(1)$ 
return  $\succ O(1)$ 

```

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.

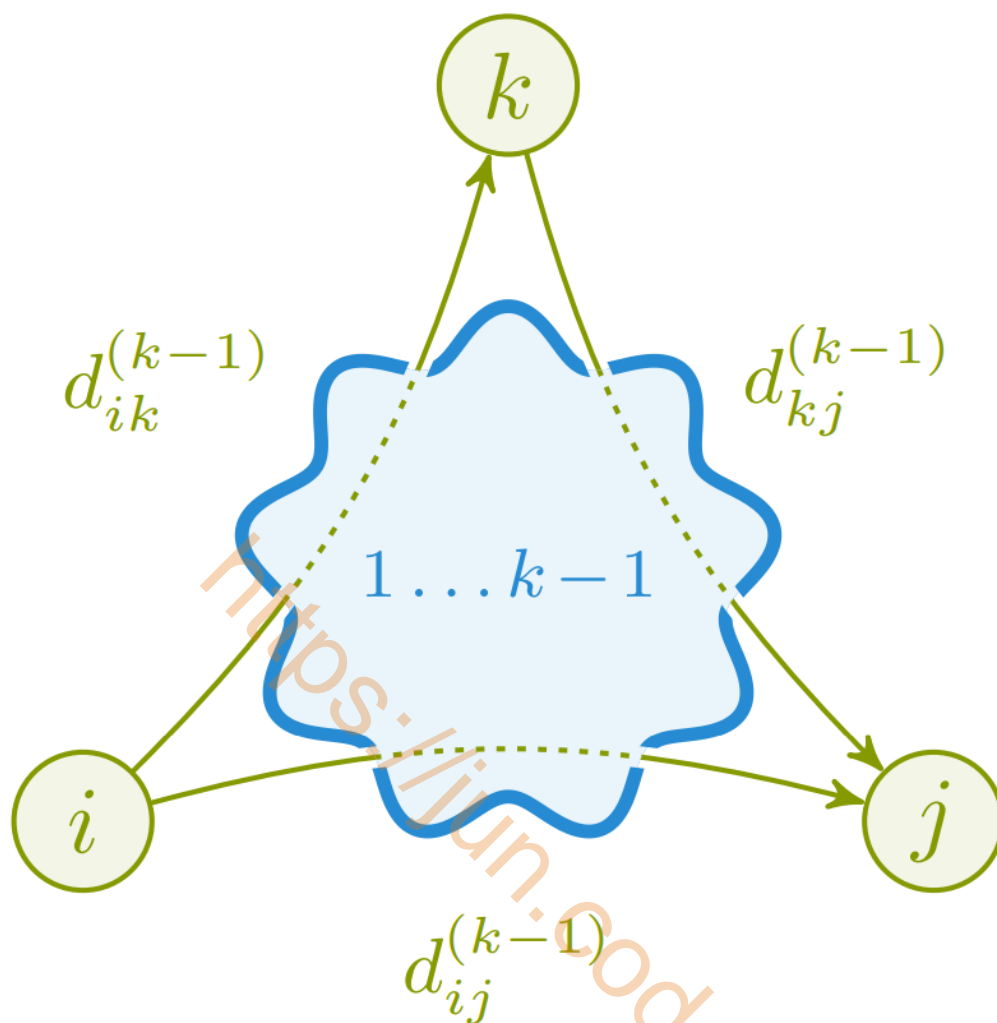
Fra hver node til alle andre?

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

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.



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

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

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

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

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)

1 initialize D and Π

2 **for** $k = 1$ **to** n

3 **for** $i = 1$ **to** n

4 **for** $j = 1$ **to** n

5 **if** $d_{ij} > d_{ik} + d_{kj}$

6 $d_{ij} = d_{ik} + d_{kj}$

7 $\pi_{ij} = \pi_{kj}$

8 **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)

1 **if** $i == j$

2 print i

3 **elseif** $\pi_{ij} == \text{NIL}$

4 print “no path from” i “to” j “exists”

5 **else** PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_{ij})

6 print j