Korteste veje

Korteste veje i vægtede grafer

Længde af sti $=$ sum af vægte af kanter på sti.

Korteste veje i vægtede grafer

Længde af sti = sum af vægte af kanter på sti.

$\delta(u, v)$ = længden af en korteste sti fra u til v. Sættes til ∞ hvis ingen sti findes.

Korteste veje i vægtede grafer

Længde af sti = sum af vægte af kanter på sti.

$\delta(u, v)$ = længden af en korteste sti fra u til v. Sættes til ∞ hvis ingen sti findes.

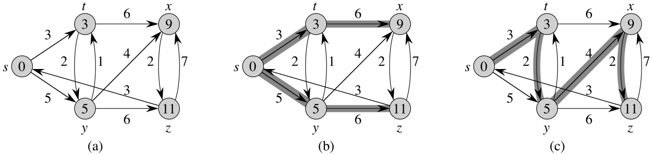
Single-source shortest-path problemet: Givet $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Korteste veje i vægtede grafer

Længde af sti = sum af vægte af kanter på sti.

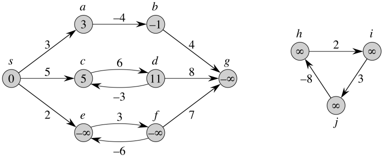
$\delta(u, v)$ = længden af en korteste sti fra u til v. Sættes til ∞ hvis ingen sti findes.

Single-source shortest-path problemet: Givet $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.



Korteste veje i vægtede grafer

Bemærk: problemet er ikke veldefineret hvis der findes kredse (som kan nås fra s) med negativ sum:



Omvendt: hvis der ikke findes sådanne negative kredse, kan vi nøjes med at se på simple stier (ingen gentagelser af knuder på stien). Der er et endeligt antal sådanne (antal $\leq n!$), så "længde af korteste sti" er veldefineret.

Relaxation

```
INIT-SINGLE-SOURCE(G, s)
  for each v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

Relaxation

```
INIT-SINGLE-SOURCE(G, s)
  for each v ∈ G.V
      v.d = ∞
      v.π = NIL
  s.d = 0
```

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```

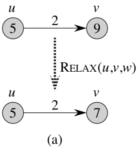
Relaxation

INIT-SINGLE-SOURCE(G, s)

for each $v \in G.V$
$\quad v.d = \infty$
$\quad v.\pi = \text{NIL}$
$s.d = 0$

RELAX(u, v, w)

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



Relaxation

INIT-SINGLE-SOURCE(G, s)

for each $v \in G.V$

$v.d = \infty$

$v.\pi = \text{NIL}$

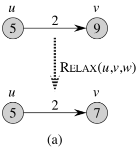
$s.d = 0$

RELAX(u, v, w)

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

$v.d = u.d + w(u, v)$

$v.\pi = u$



Bemærk: Implementation af "∞", skal fungere matematisk korrekt med "$>$" og $+$" (bare at bruge `Integer.MAX_VALUE` som "∞" er ikke nok).

Relaxation

INIT-SINGLE-SOURCE(G, s)
for each $v \in G.V$
  $v.d = \infty$
  $v.\pi = \text{NIL}$
$s.d = 0$

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

Hvis man efter INIT-SINGLE-SOURCE kun ændrer $v.d$ og $v.\pi$ via RELAX, ses nemt ved induktion på antal RELAX at følgende invariant gælder:

Hvis $v.d < \infty$ findes der en sti fra s til v hvor:

- stien har længde $v.d$
- sidste kant på denne sti er $(v.\pi, v)$.

Relaxation

Hvis man efter INIT-SINGLE-SOURCE kun ændrer $v.d$ og $v.\pi$ via RELAX, gælder følgende invariant:

Hvis $v.d < \infty$ findes der en sti fra s til v hvor:

- stien har længde $v.d$
- sidste kant på denne sti er $(v.\pi, v)$.

Relaxation

Hvis man efter INIT-SINGLE-SOURCE kun ændrer $v.d$ og $v.\pi$ via RELAX, gælder følgende invariant:

Hvis $v.d < \infty$ findes der en sti fra s til v hvor:

- stien har længde $v.d$
- sidste kant på denne sti er $(v.\pi, v)$.

Af første punkt følger, at der altid gælder $\delta(s, v) \leq v.d$ (se på cases $v.d < \infty$ og $v.d = \infty$). Deraf ses (da $v.d$ kun kan falde ved brug af RELAX) at hvis $\delta(s, v) = v.d$ på et tidspunkt, vil dette ikke ændres senere. (Specielt gælder det sidste allerede efter INIT-SINGLE-SOURCE for knuder, som ikke kan nås fra s).

Relaxation

Hvis man efter Init-Single-Source kun ændrer $v.d$ og $v.\pi$ via Relax, gælder følgende invariant:

Hvis $v.d < \infty$ findes der en sti fra s til v hvor:

- stien har længde $v.d$
- sidste kant på denne sti er $(v.\pi, v)$.

Af første punkt følger, at der altid gælder $\delta(s, v) \leq v.d$ (se på cases $v.d < \infty$ og $v.d = \infty$). Deraf ses (da $v.d$ kun kan falde ved brug af Relax) at hvis $\delta(s, v) = v.d$ på et tidspunkt, vil dette ikke ændres senere. (Specielt gælder det sidste allerede efter Init-Single-Source for knuder, som ikke kan nås fra s).

Heraf følger, at hvis $\delta(s, v) = v.d$ for alle knuder, vil ingen kant (u, v) kunne relaxeres (dvs. $v.d \leq u.d + w(u, v)$ for alle kanter (u, v)), da dette ville ændre sænke $v.d$.

Relaxation

Hvis man efter Init-Single-Source kun ændrer $v.d$ og $v.\pi$ via Relax, gælder følgende invariant:

Hvis $v.d < \infty$ findes der en sti fra s til v hvor:

- stien har længde $v.d$
- sidste kant på denne sti er $(v.\pi, v)$.

Af første punkt følger, at der altid gælder $\delta(s, v) \leq v.d$ (se på cases $v.d < \infty$ og $v.d = \infty$). Deraf ses (da $v.d$ kun kan falde ved brug af Relax) at hvis $\delta(s, v) = v.d$ på et tidspunkt, vil dette ikke ændres senere. (Specielt gælder det sidste allerede efter Init-Single-Source for knuder, som ikke kan nås fra s).

Heraf følger, at hvis $\delta(s, v) = v.d$ for alle knuder, vil ingen kant (u, v) kunne relaxeres (dvs. $v.d \leq u.d + w(u, v)$ for alle kanter (u, v)), da dette ville ændre sænke $v.d$.

Man kan også vise, at hvis $\delta(s, v) = v.d$ for alle knuder, vil en sti fra s til v af længde $v.d$ kunne gennemløbes baglæns ved at følge π-pointere.

Bellman-Ford-Moore [1956-57-58]

```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

Bellman-Ford-Moore [1956-57-58]

```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| − 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

Køretid:

Bellman-Ford-Moore [1956-57-58]

```
BELLMAN-FORD(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  for i = 1 to |G.V| - 1
      for each edge (u, v) ∈ G.E
          RELAX(u, v, w)
  for each edge (u, v) ∈ G.E
      if v.d > u.d + w(u, v)
          return FALSE
  return TRUE
```

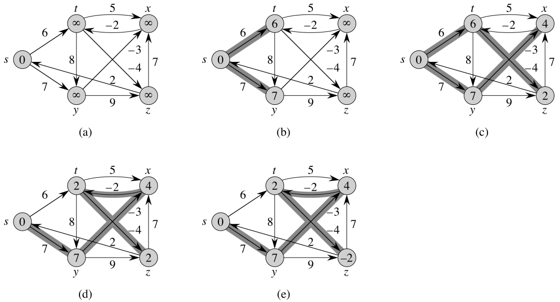
Køretid: $O(nm)$

Bellman-Ford-Moore [1956-57-58]

$\textsc{Bellman-Ford}(G, w, s)$

$\textsc{Init-Single-Source}(G, s)$
for $i = 1$ to $|G.V| - 1$
  for each edge $(u, v) \in G.E$
    $\textsc{Relax}(u, v, w)$
for each edge $(u, v) \in G.E$
  if $v.d > u.d + w(u, v)$
    return FALSE
return TRUE

Køretid: $O(nm)$



Bellman-Ford, korrekthed

Sætning: Hvis der findes en negativ kreds, som kan nås fra s, svarer algoritmen FALSE. Ellers svarer den TRUE, og $v.d = \delta(s, v)$ for alle $v \in V$ når den stopper.

Bellman-Ford, korrekthed

Sætning: Hvis der findes en negativ kreds, som kan nås fra s, svarer algoritmen FALSE. Ellers svarer den TRUE, og $v.d = \delta(s, v)$ for alle $v \in V$ når den stopper.

Bevis: Hovedpointen i algoritmen er, at den vedligeholder følgende invariant: efter i iterationer af første **for**-løkke gælder $v.d = \delta(s, v)$ for alle knuder v, der har en korteste vej med højst i kanter. Dette ses ved induktion på i. Vi viser nu sætningen.

Bellman-Ford, korrekthed

Sætning: Hvis der findes en negativ kreds, som kan nås fra s, svarer algoritmen FALSE. Ellers svarer den TRUE, og $v.d = \delta(s, v)$ for alle $v \in V$ når den stopper.

Bevis: Hovedpointen i algoritmen er, at den vedligeholder følgende invariant: efter i iterationer af første **for**-løkke gælder $v.d = \delta(s, v)$ for alle knuder v, der har en korteste vej med højst i kanter. Dette ses ved induktion på i. Vi viser nu sætningen.

Case 1: der er ingen negative cykler som kan nås fra s. Så har alle knuder, som kan nås fra s, en simpel korteste vej, dvs. en vej uden gentagelser af knuder. En sådan vej har højst n knuder, og derfor højst $n - 1$ kanter. Af invarianten ovenfor gælder $v.d = \delta(s, v)$ for disse knuder når algoritment stopper. For knuder, der ikke kan nås fra s gælder dette altid. Så $v.d = \delta(s, v)$ for alle knuder når algoritmen stopper.

Bellman-Ford, korrekthed

Sætning: Hvis der findes en negativ kreds, som kan nås fra s, svarer algoritmen FALSE. Ellers svarer den TRUE, og $v.d = \delta(s, v)$ for alle $v \in V$ når den stopper.

Bevis: Hovedpointen i algoritmen er, at den vedligeholder følgende invariant: efter i iterationer af første **for**-løkke gælder $v.d = \delta(s, v)$ for alle knuder v, der har en korteste vej med højst i kanter. Dette ses ved induktion på i. Vi viser nu sætningen.

Case 1: der er ingen negative cykler som kan nås fra s. Så har alle knuder, som kan nås fra s, en simpel korteste vej, dvs. en vej uden gentagelser af knuder. En sådan vej har højst n knuder, og derfor højst $n - 1$ kanter. Af invarianten ovenfor gælder $v.d = \delta(s, v)$ for disse knuder når algoritment stopper. For knuder, der ikke kan nås fra s gælder dette altid. Så $v.d = \delta(s, v)$ for alle knuder når algoritmen stopper.

Når $v.d = \delta(s, v)$ for alle knuder, kan RELAX ikke ændre nogen $v.d$ længere (jvf. tidligere observation). Derfor svarer bliver sidste **if**-case aldrig sand, og algoritmen svarer TRUE.

Bellman-Ford, korrekthed

Case 2: der er en negativ cykel som kan nås fra s. Vi kan antage at denne cykel er simpel (hvis den har gentagelser blandt knuderne, består den af flere, mindre cykler, hvoraf mindst én må være negativ).

Bellman-Ford, korrekthed

Case 2: der er en negativ cykel som kan nås fra s. Vi kan antage at denne cykel er simpel (hvis den har gentagelser blandt knuderne, består den af flere, mindre cykler, hvoraf mindst én må være negativ).

Lad knuderne på denne cykel være $v_1, v_2, \ldots, v_k$, hvor v_1 er en knude i cyklen som kan nås fra s med en sti $(s =) u_1, u_2, \ldots, u_j (= v_1)$ med færrest muligt kanter.

På grund af minimaliteten af stien (og simpelheden af cyklen) kan der ikke være gentagelser blandt knuderne på stien og cyklen (udover v_1). Så $s = u_1, u_2, \ldots, u_j = v_1, v_2, \ldots, v_k$ er en sti med højst n knuder. Det er let at se via induktion over i at efter i iterationer af første **for**-løkke er $v.d < \infty$ for de første $i + 1$ knuder på denne sti. Derfor gælder $v.d < \infty$ for alle knuder på cyklen når første **for**-løkke er færdig.

Bellman-Ford, korrekthed

Antag at algoritmen ikke svarer FALSE. Så gælder ved algoritmens afslutning

$$v_{i+1}.d \leq v_i.d + w(v_i, v_{i+1})$$

for $1 \leq i \leq k$ (med $v_{k+1} = v_1$). Og dermed gælder

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

Da $v_i.d < \infty$ for alle i, er de to første summer ikke bare ens, men også $< \infty$, så de kan trækkes fra og give

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

hvilket er i modstrid med at cyklen er negativ. Så algoritmen må svare FALSE. $\square$

Dijkstras algoritme [1959]

Grådig algoritme som trinvis opbygger mængde S af knuder med korrekte $v.d$ og $v.\pi$. Bruger en prioritetskø Q. Kræver alle kantvægte ≥ 0.

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Dijkstras algoritme [1959]

Grådig algoritme som trinvis opbygger mængde S af knuder med korrekte $v.d$ og $v.\pi$. Bruger en prioritetskø Q. Kræver alle kantvægte ≥ 0.

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid:

Dijkstras algoritme [1959]

Grådig algoritme som trinvis opbygger mængde S af knuder med korrekte $v.d$ og $v.\pi$. Bruger en prioritetskø Q. Kræver alle kantvægte ≥ 0.

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: n INSERT (eller én BUILD-HEAP), n EXTRACT-MIN og m DECREASE-KEY (i RELAX).

Dijkstras algoritme [1959]

Grådig algoritme som trinvis opbygger mængde S af knuder med korrekte $v.d$ og $v.\pi$. Bruger en prioritetskø Q. Kræver alle kantvægte ≥ 0.

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: n INSERT (eller én BUILD-HEAP), n EXTRACT-MIN og m DECREASE-KEY (i RELAX). I alt $O(m \log n)$ hvis prioritetskøen implementeres med en heap.

Dijkstras algoritme [1959]

Grådig algoritme som trinvis opbygger mængde S af knuder med korrekte $v.d$ og $v.\pi$. Bruger en prioritetskø Q. Kræver alle kantvægte ≥ 0.

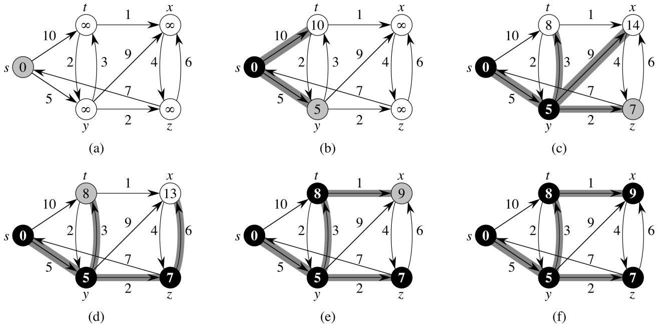
```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: n INSERT (eller én BUILD-HEAP), n EXTRACT-MIN og m DECREASE-KEY (i RELAX). I alt $O(m \log n)$ hvis prioritetskøen implementeres med en heap.

Invariant: Når u indlemmes i S (dvs. udtages med en EXTRACT-MIN) er $u.d = \delta(s, u)$ (hvis alle kantvægte er ≥ 0).

Bevis for invariant: Et induktionsbevis (gennemgået på tavle). Af invarianten følger at algoritmen er korrekt (da alle knuder er i S til sidst).

Dijkstra, eksempel



Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n + m)$.

Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n+m)$.

DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
    for each vertex $v \in G.Adj[u]$
      RELAX(u, v, w)

Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n + m)$.

DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
    for each vertex $v \in G.Adj[u]$
      RELAX(u, v, w)

Køretid:

Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n+m)$.

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: $O(n+m)$.

Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n + m)$.

```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: $O(n + m)$.

Sætning: Når algoritmen stopper er $v.d = \delta(s, v)$ for alle $v \in V$.

Algoritme for DAGs [unknown]

Recall: DAG = Directed Acyclic Graph.

Recall: En topologisk sortering kan findes via DFS i tid $O(n+m)$.

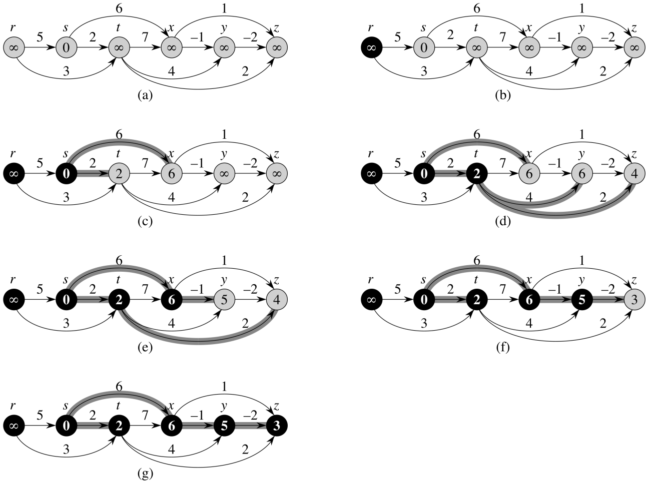
```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Køretid: $O(n+m)$.

Sætning: Når algoritmen stopper er $v.d = \delta(s,v)$ for alle $v \in V$.

Bevis: For en knude v med en sti fra s til v: alle knuder på en korteste sti er blevet relaxeret i rækkefølge (hvorved korrekte δ-værdier sættes på denne sti). For alle andre knuder gælder $\infty = \delta(s,v)$ så korrekthed her følger af $\delta(s,v) \leq v.d$.

Algoritmen for DAG, eksempel



Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte):

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte): $O(nm \log n)$ tid.

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte): $O(nm \log n)$ tid.

Eller: køre Bellman-Ford-Moore fra hver source $s \in V$ (hvis der er negative vægte):

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte): $O(nm \log n)$ tid.

Eller: køre Bellman-Ford-Moore fra hver source $s \in V$ (hvis der er negative vægte): $O(n^2 m)$ tid.

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte): $O(nm \log n)$ tid.

Eller: køre Bellman-Ford-Moore fra hver source $s \in V$ (hvis der er negative vægte): $O(n^2 m)$ tid.

En anden mulighed: Floyd-Warshalls algoritme. $O(n^3)$ tid. Klarer negative vægte.

Korteste veje mellem alle par af knuder

All-pairs shortest-path problemet: For alle $s \in V$, find $\delta(s, v)$ (og en konkret sti) for alle $v \in V$.

Én mulighed: køre Dijkstra fra hver source $s \in V$ (kræver ikke-negative vægte): $O(nm \log n)$ tid.

Eller: køre Bellman-Ford-Moore fra hver source $s \in V$ (hvis der er negative vægte): $O(n^2 m)$ tid.

En anden mulighed: Floyd-Warshalls algoritme. $O(n^3)$ tid. Klarer negative vægte.

Endnu en mulighed: Johnsons algoritme. Kører i $O(nm \log n)$ tid. Klarer negative vægte.

Floyd-Warshalls algoritme [1962]

Dynamisk programmeringalgoritme.

Bruger adjacency-matrix repræsentationen.

Output også på matrice-form:

$D = (d_{ij})$, $d_{ij} = \delta(v_i, v_j) =$ længden af en korteste sti fra v_i til v_j. Sættes til ∞ hvis ingen sti findes.

Floyd-Warshalls algoritme [1962]

Dynamisk programmeringalgoritme.

Bruger adjacency-matrix repræsentationen.

Output også på matrice-form:

$D = (d_{ij})$, $d_{ij} = \delta(v_i, v_j) =$ længden af en korteste sti fra v_i til v_j. Sættes til ∞ hvis ingen sti findes.

$\Pi = (\pi_{ij})$, $\pi_{ij} =$ sidste knude før v_j på en korteste sti fra knude v_i til knude v_j. Sættes til NIL hvis ingen sti findes.

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

FLOYD-WARSHALL(W, n)

```
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

$D^{(0)} = W$
for $k = 1$ **to** n
  let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
  for $i = 1$ **to** n
    for $j = 1$ **to** n
      $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

FLOYD-WARSHALL(W, n)

$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

Køretid:

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

```
FLOYD-WARSHALL(W, n)
  D^(0) = W
  for k = 1 to n
      let D^(k) = (d_ij^(k)) be a new n × n matrix
      for i = 1 to n
          for j = 1 to n
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
  return D^(n)
```

Køretid: $O(n^3)$.

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

```
FLOYD-WARSHALL(W, n)
  D^(0) = W
  for k = 1 to n
      let D^(k) = (d_ij^(k)) be a new n × n matrix
      for i = 1 to n
          for j = 1 to n
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
  return D^(n)
```

Køretid: $O(n^3)$. Plads: $O(n^2)$ (kun forrige $D^{(k)}$ matrice behøves gemmes).

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

```
FLOYD-WARSHALL(W, n)
  D^(0) = W
  for k = 1 to n
      let D^(k) = (d_ij^(k)) be a new n × n matrix
      for i = 1 to n
          for j = 1 to n
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
  return D^(n)
```

Køretid: $O(n^3)$. Plads: $O(n^2)$ (kun forrige $D^{(k)}$ matrice behøves gemmes).

Sætning: Når algoritmen stopper er d_{ij} og π_{ij} sat korrekt for alle $v_i, v_j \in V$ (hvis ingen negativ kreds er i grafen).

Floyd-Warshalls algoritme

(Kun konstruktion af D-matricen vises.)

```
FLOYD-WARSHALL(W, n)
  D^(0) = W
  for k = 1 to n
      let D^(k) = (d_ij^(k)) be a new n × n matrix
      for i = 1 to n
          for j = 1 to n
              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
  return D^(n)
```

Køretid: $O(n^3)$. Plads: $O(n^2)$ (kun forrige $D^{(k)}$ matrice behøves gemmes).

Sætning: Når algoritmen stopper er d_{ij} og π_{ij} sat korrekt for alle $v_i, v_j \in V$ (hvis ingen negativ kreds er i grafen).

Bevis: Invarianten er, at $D^{(k)}$ indeholder længden af korteste veje mellem v_i og v_j som passerer knuderne $v_1, v_2, \ldots, v_k$ (udover endepunkterne v_i og v_j).

Johnsons algoritme [1977]

Bruger:

- Kører Bellman-Ford-Moore én gang på let udvidet graf.
- Herudfra justering af kantvægte så alle bliver positive uden essentielt at ændre korteste veje.
- Kører Dijkstra fra alle knuder.

Kører i $O(nm \log n + nm) = O(nm \log n)$ tid, klarer negative vægte.