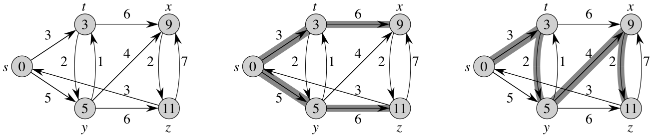
Korteste veje

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 af denne længde) for alle $v \in V$.



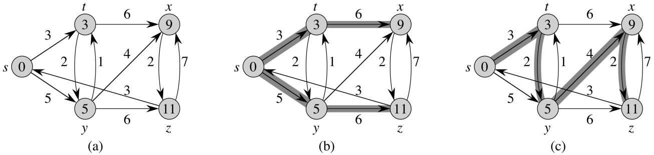
(a) (b) (c)

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.

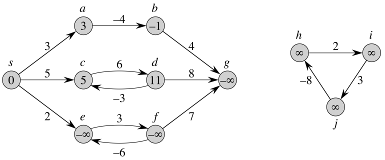
Single-source shortest-path problemet: Givet $s \in V$, find $\delta(s, v)$ (og en konkret sti af denne længde) for alle $v \in V$.



Bemærk at prefixer af korteste veje selv må være korteste veje: hvis $v_1, v_2, \ldots, v_k$ er en korteste vej fra v_1 til v_k, så er $v_1, v_2, \ldots, v_i$ en korteste vej fra v_1 til v_i for alle $i \leq k$ (ellers kan vejen fra v_1 til v_k gøres kortere).

Korteste veje i vægtede grafer

Problemet er ikke veldefineret, hvis der findes kredse (som kan nås fra s) med negativ sum, idet der så findes veje med vilkårlig lav længde:



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 stier (højst $n!$), så "længde af korteste sti" er veldefineret.

Relaxation – en generel teknik til at finde korteste veje

Idé: Brug kanter til at udbrede information fra knude til knude om længder af stier. Kaldes RELAX af kanten.

Hvis u har information om, at der er en sti fra s til u af længde $u.d$, og (u, v) er en kant af med vægt w, så findes der en sti af længde $u.d + w$ til v. Er det bedre information for v, end hvad den har lige nu?

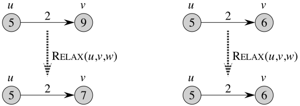
Relaxation – en generel teknik til at finde korteste veje

Idé: Brug kanter til at udbrede information fra knude til knude om længder af stier. Kaldes RELAX af kanten.

Hvis u har information om, at der er en sti fra s til u af længde $u.d$, og (u, v) er en kant af med vægt w, så findes der en sti af længde $u.d + w$ til v. Er det bedre information for v, end hvad den har lige nu?

```
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
```



(a) (b)

Relaxation – en generel teknik til at finde korteste veje

Idé: Brug kanter til at udbrede information fra knude til knude om længder af stier. Kaldes RELAX af kanten.

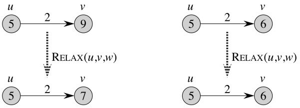
Hvis u har information om, at der er en sti fra s til u af længde $u.d$, og (u, v) er en kant af med vægt w, så findes der en sti af længde $u.d + w$ til v. Er det bedre information for v, end hvad den har lige nu?

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



(a) (b)

[Detalje: 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 ∈ 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
```

Vi vil møde en række algoritmer, som starter med INIT-SINGLE-SOURCE og derefter kun ændrer $v.d$ og $v.\pi$ via RELAX.

For sådanne algoritmer gælder følgende invariant (ses nemt ved induktion på antal RELAX):

Hvis $v.d < \infty$ findes en sti fra s til v af længde $v.d$.

Relaxation

$\boxed{\text{Hvis } v.d < \infty \text{ findes en sti fra } s \text{ til } v \text{ af længde } v.d.}$

Relaxation

$\boxed{\text{Hvis } v.d < \infty \text{ findes en sti fra } s \text{ til } v \text{ af længde } v.d.}$

Derfor gælder altid $\delta(s, v) \leq v.d$ (for $v.d < \infty$ følger det af ovenstående, for $v.d = \infty$ er det oplagt).

Relaxation

$\boxed{\text{Hvis } v.d < \infty \text{ findes en sti fra } s \text{ til } v \text{ af længde } v.d.}$

Derfor gælder altid $\delta(s, v) \leq v.d$ (for $v.d < \infty$ følger det af ovenstående, for $v.d = \infty$ er det oplagt).

Da $v.d$ kun kan falde ved brug af RELAX, følger det, at hvis på et tidspunkt $\delta(s, v) = v.d$, vil hverken $v.d$ eller $v.\pi$ ændres senere.

Specielt gælder, 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)).

Relaxation

De korteste stier kan findes via $v.\pi$-pointers:

- Mængden S af knuder v med $\delta(s, v) = v.d < \infty$ udgør et træ med $v.\pi$ som parent pointers og s som rod.
- For en knude v i træet vil stien mod roden svare til et baglæns gennemløb af en sti i grafen fra s til v af længde $\delta(s, v)$.

Relaxation

De korteste stier kan findes via $v.\pi$-pointers:

- Mængden S af knuder v med $\delta(s, v) = v.d < \infty$ udgør et træ med $v.\pi$ som parent pointers og s som rod.
- For en knude v i træet vil stien mod roden svare til et baglæns gennemløb af en sti i grafen fra s til v af længde $\delta(s, v)$.

Dette vises ved induktion på antal RELAX.

Basis: Lige efter initialisering er s den eneste knude v, som har $v.d < \infty$. Da $s.d = 0$ og $s.\pi =$ NIL efter initialisering, er $S = \{s\}$ og invarianten opfyldt med et træ af størrelse én, hvis blot $\delta(s, s) = 0$.

[Bemærk at $\delta(s, s) \neq 0$ kun er muligt hvis s ligger på en negativ kreds. Men så gælder for alle knuder v enten $\delta(s, v) = -\infty$ (hvis v kan nås fra s) eller $\delta(s, v) = \infty$ (hvis v ikke kan nås fra s). Hvis $v.d < \infty$, svarer $v.d$ til længden af en konkret sti (se tidligere), og er derfor forskellig fra $-\infty$. Så S er altid tom, og der er intet at vise.]

Relaxation

- Mængden S af knuder v med $\delta(s, v) = v.d < \infty$ udgør et træ med $v.\pi$ som parent pointers og s som rod.
- For en knude v i træet vil stien mod roden svare til et baglæns gennemløb af en sti i grafen fra s til v af længde $\delta(s, v)$.

Induktionsskridt: For en RELAX som ikke ændrer noget, er der intet at vise. For en RELAX, som ændrer $v.d$ (og dermed $v.\pi$): her kan v ikke være i S før RELAX (da $\delta(s, v) < v.d$ på det tidspunkt), så alle eksisterende knuder i træet er uændrede (inkl. deres parent pointers). Hvis v indlemmes i S pga. denne RELAX, så lad (u, v) var kanten, som blev relaxeret, og lad w være dens vægt. Vi har så $\delta(s, v) = v.d = u.d + w$. Derfor må $\delta(s, u) = u.d$ gælde (hvis der var en vej kortere end $u.d$ til u, var der en vej kortere end $u.d + w$ til v) og $u.d < \infty$ gælder også (ellers ville RELAX ikke ske). Så u er med i S, får v som barn, og sætningen gælder klart igen.

Relaxation

Vi skal senere bruge:

Lemma: Hvis $s = v_1, v_2, \ldots, v_k = v$ er en korteste vej fra s til v, og en algoritme laver $\textsc{Relax}$ på kanterne $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ efter tur (med en vilkårlig mængde $\textsc{Relax}$ af andre kanter mellem disse $\textsc{Relax}$), da er $\delta(s, v) = v.d$ efter den sidste af disse $\textsc{Relax}$.

Bevis: Det ses ved induktion på i, at når der laves $\textsc{Relax}$ på kanten (v_{i-1}, v_i) i sekvensen ovenfor, er $v_i.d$ højest lig summen af vægtene af de første $i-1$ kanter i stien.

Da stien er en korteste sti til v, og da $\delta(s, v) \le v.d$ altid gælder, er $\delta(s, v) = v.d$ efter den sidste af disse $\textsc{Relax}$.

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
```

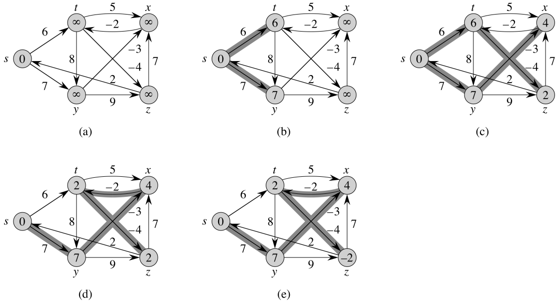
Køretid:

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]



[Relaxeringsrækkefølge:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.]

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 følger af lemmaet ovenfor.

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 følger af lemmaet ovenfor.

Vi viser nu sætningen:

Case 1: der er ingen negative kredse 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 følger af lemmaet ovenfor.

Vi viser nu sætningen:

Case 1: der er ingen negative kredse 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 $\textsc{Relax}$ 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 kreds som kan nås fra s. Vi kan antage at denne kreds er simpel (hvis den har gentagelser blandt knuderne, består den af flere, mindre kredse, hvoraf mindst én må være negativ).

Bellman-Ford, korrekthed

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

Lad knuderne på denne kreds være $v_1, v_2, \ldots, v_k$, hvor v_1 er en knude i kredsen 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 kredsen) kan der ikke være gentagelser blandt knuderne på stien og kredsen (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 kredsen 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 kredsen er negativ. Så algoritmen må svare FALSE. □

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.

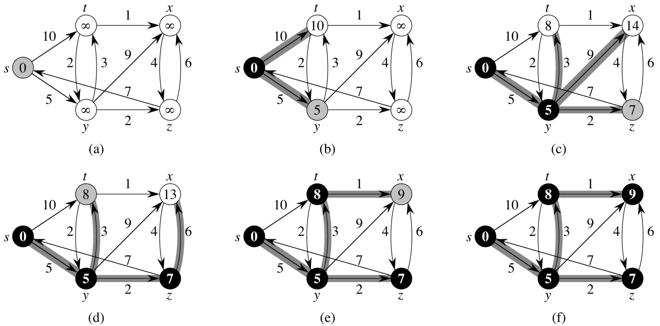
```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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 ∈ 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 ∈ 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 $O(n+m)$.

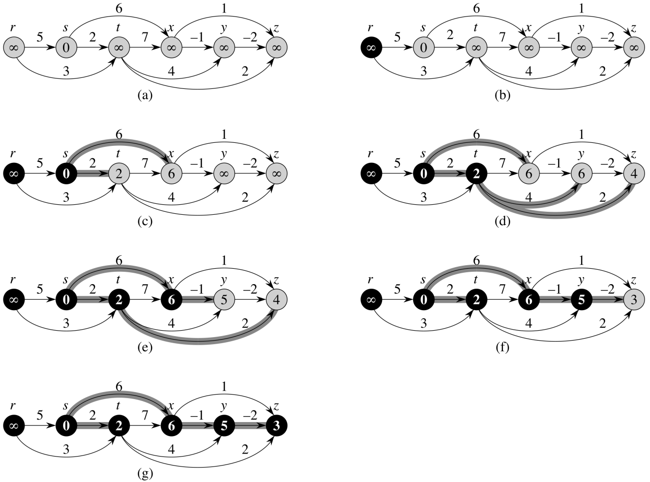
```
DAG-SHORTEST-PATHS(G, w, s)
  topologically sort the vertices
  INIT-SINGLE-SOURCE(G, s)
  for each vertex u, taken in topologically sorted order
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

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, se tidligere lemma). 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]

Baseret på dynamisk programmering.

Bruger adjacency-matrix repræsentationen.

Output er 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]

Baseret på dynamisk programmering.

Bruger adjacency-matrix repræsentationen.

Output er 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 $D^{(k)} = \left(d_{ij}^{(k)}\right)$ 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 $D^{(k)} = \left(d_{ij}^{(k)}\right)$ 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: $O(n^3)$. 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.