

Grafer og graf-gennemløb

Grafer

En mængde V af *knuder* (vertices).

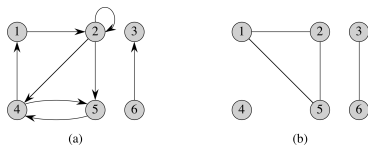
En mængde $E \in V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Grafer

En mængde V af *knuder* (vertices).

En mængde $E \subseteq V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Figur:



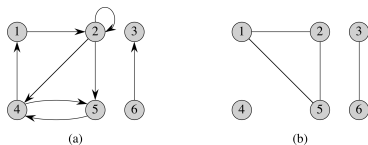
- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.

Grafer

En mængde V af *knuder* (vertices).

En mængde $E \subseteq V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Figur:



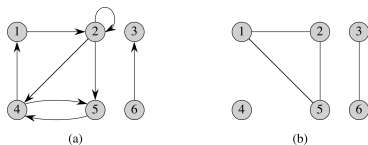
- ▶ Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- ▶ Orienteret vs. uorienteret.

Grafer

En mængde V af *knuder* (vertices).

En mængde $E \subseteq V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Figur:



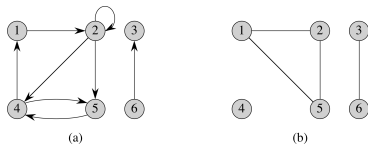
- ▶ Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- ▶ Orienteret vs. uorienteret.
- ▶ Evt. loops, multiple kanter.

Grafer

En mængde V af *knuder* (vertices).

En mængde $E \subseteq V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Figur:



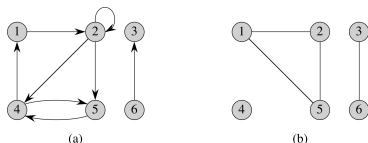
- ▶ Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- ▶ Orienteret vs. uorienteret.
- ▶ Evt. loops, multiple kanter.
- ▶ $0 \leq m \leq n(n-1)$ og $0 \leq m \leq n(n-1)/2$

Grafer

En mængde V af *knuder* (vertices).

En mængde $E \subseteq V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Figur:

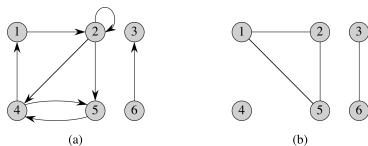


- ▶ Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- ▶ Orienteret vs. uorienteret.
- ▶ Evt. loops, multiple kanter.
- ▶ $0 \leq m \leq n(n-1)$ og $0 \leq m \leq n(n-1)/2$
- ▶ Vægtede grafer.

Grafer

Modeller for mange ting:

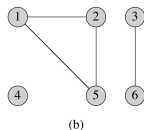
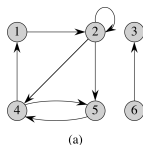
- ▶ Ledningsnet (telefon, strøm, olie, vand, ...).



Grafer

Modeller for mange ting:

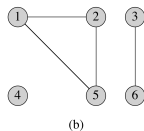
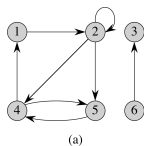
- ▶ Ledningsnet (telefon, strøm, olie, vand, ...).
- ▶ Bekendtskaber.



Grafer

Modeller for mange ting:

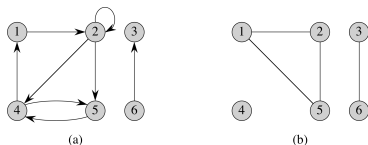
- ▶ Ledningsnet (telefon, strøm, olie, vand, ...).
- ▶ Bekendtskaber.
- ▶ Medforfatterskaber.



Grafer

Modeller for mange ting:

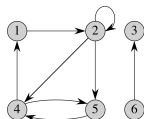
- ▶ Ledningsnet (telefon, strøm, olie, vand, ...).
- ▶ Bekendtskaber.
- ▶ Medforfatterskaber.
- ▶ WWW-grafen af sider og links.



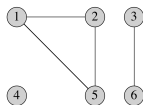
Grafer

Modeller for mange ting:

- ▶ Ledningsnet (telefon, strøm, olie, vand, ...).
- ▶ Bekendtskaber.
- ▶ Medforfatterskaber.
- ▶ WWW-grafen af sider og links.
- ▶ Vejnet.



(a)



(b)

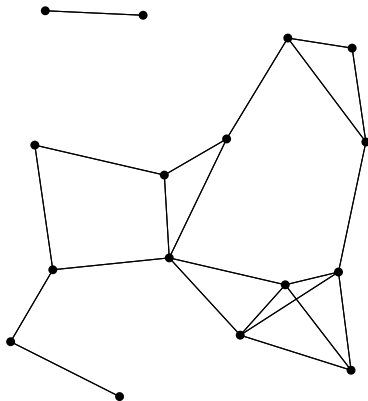
Algoritmiske spørgsmål

- ▶ Datastruktur for grafer.
- ▶ Afgør hvilke knuder som kan nås fra hvilke knuder.
- ▶ Find korteste sti mellem to knuder.
- ▶ Find mindst mulige delmængde af kanter som stadig holder alle knuder forbundet.
- ▶ Find største samling kanter som ikke deler knuder (en matching).

⋮

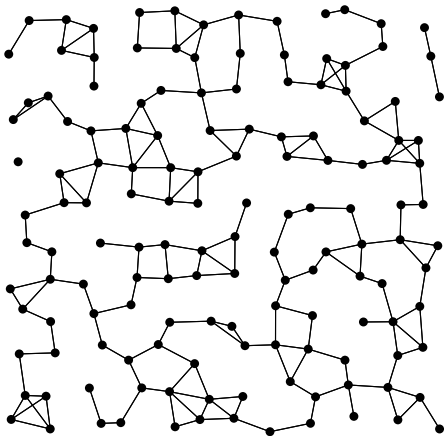
Algoritmiske spørgsmål

- ▶ Afgør hvilke knuder som kan nås fra hvilke knuder.
- ▶ Find korteste sti mellem to knuder.



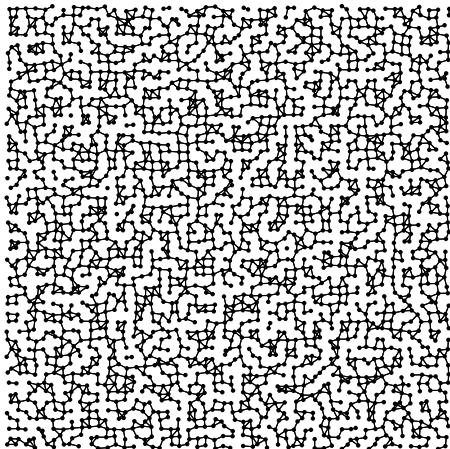
Algoritmiske spørgsmål

- ▶ Afgør hvilke knuder som kan nås fra hvilke knuder.
- ▶ Find korteste sti mellem to knuder.



Algoritmiske spørgsmål

- ▶ Afgør hvilke knuder som kan nås fra hvilke knuder.
- ▶ Find korteste sti mellem to knuder.



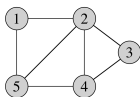
Datastrukturer for grafer

Graf-repræsentationer:

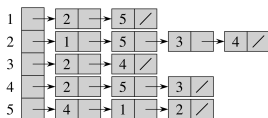
Datastrukturer for grafer

Graf-repræsentationer:

Adjacency list og adjacency matrix



(a)



(b)

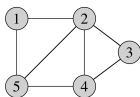
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

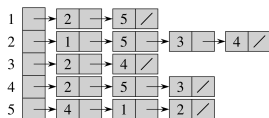
Datastrukturer for grafer

Graf-repræsentationer:

Adjancency list og adjancency matrix



(a)



(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

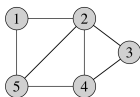
(c)

Plads: Henholdsvis $O(n + m)$ og $O(n^2)$.

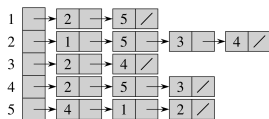
Datastrukturer for grafer

Graf-repræsentationer:

Adjacency list og adjacency matrix



(a)



(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Plads: Henholdsvis $O(n + m)$ og $O(n^2)$.

Hvis ikke andet oplyses, bruges adjacency list repræsentationen i algoritmer i resten af kurset.

Fra nu af: en kant i en uorienterede graf repræsenteres som to orienterede kanter (så mht. implementation er uorienterede grafer et specialtilfælde af orienterede grafer).

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide
- ▶ Knuder i fronten (opdagede, men naboliste ikke brugt op) er grå

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide
- ▶ Knuder i fronten (opdagede, men naboliste ikke brugt op) er grå
- ▶ Opdagede knuder med opbrugt naboliste er sorte

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide
- ▶ Knuder i fronten (opdagede, men naboliste ikke brugt op) er grå
- ▶ Opdagede knuder med opbrugt naboliste er sorte
- ▶ Basalt skridt: vælg en grå knude v , finde næste kant (v, u) i dens naboliste, opdag u hvis den ikke er det allerede.

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide
- ▶ Knuder i fronten (opdagede, men naboliste ikke brugt op) er grå
- ▶ Opdagede knuder med opbrugt naboliste er sorte
- ▶ Basalt skridt: vælg en grå knude v , finde næste kant (v, u) i dens naboliste, opdag u hvis den ikke er det allerede.
- ▶ Når en knude v ($\neq s$) besøges første gang husker den, fra hvilken knude den blev opdaget (dens predecessor) i variabelen $v.\pi$.

Grafgennemløb

- ▶ Breath-First-Search (BFS)
- ▶ Depth-First-Search (DFS)
- ▶ Priority-Search (Dijkstras algoritme, A*)

Tre varianter med fælles grundstruktur:

- ▶ En startknode (source) s
- ▶ Ikke-opdagede knuder er hvide
- ▶ Knuder i fronten (opdagede, men naboliste ikke brugt op) er grå
- ▶ Opdagede knuder med opbrugt naboliste er sorte
- ▶ Basalt skridt: vælg en grå knude v , finde næste kant (v, u) i dens naboliste, opdag u hvis den ikke er det allerede.
- ▶ Når en knude v ($\neq s$) besøges første gang husker den, fra hvilken knude den blev opdaget (dens predecessor) i variabelen $v.\pi$.
- ▶ Vi bruger adjacency list repræsentationen.

Kanterne $(v, v.\pi)$ udgør tilsammen altid et træ med s som rod og indeholdende alle opdagede knuder (ses nemt via induktion over antal skridt).

Bredde-Først-Søgning (BFS)

Holder de grå knuder (fronten) i en **KØ**.

Bredde-Først-Søgning (BFS)

Holder de grå knuder (fronten) i en **KØ**.

Tilføjer også en variabel **$v.d$** til alle knuder v (d for distance.)

Bredde-Først-Søgning (BFS)

Holder de grå knuder (fronten) i en **KØ**.

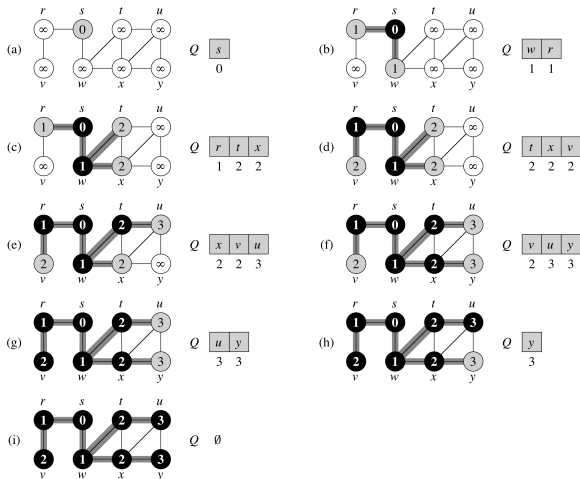
Tilføjer også en variabel **$v.d$** til alle knuder v (d for distance.)

BFS(G, s)

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == \text{WHITE}$ 
14              $v.color = \text{GRAY}$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = \text{BLACK}$ 
```


Bredde-Først-Søgning (BFS)

Eksempel:



Egenskaber

Køretid: $O(n + m)$

Egenskaber

Køretid: $O(n + m)$

Bevis: Initialisering tager $O(n)$ tid. I resten af algoritmen kan en knude ikke indsættes i køen mere end én gang (pga. farvemærkningen), så maksimalt arbejde er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Egenskaber

Køretid: $O(n + m)$

Bevis: Initialisering tager $O(n)$ tid. I resten af algoritmen kan en knude ikke indsættes i køen mere end én gang (pga. farvemærkningen), så maksimalt arbejde er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Definition: $\delta(s, v)$ er længden af en korteste sti, *målt i antal kanter*, fra startknuden s til knuden v . Findes ingen sti, defineres $\delta(s, v) = \infty$.

Egenskaber

Køretid: $O(n + m)$

Bevis: Initialisering tager $O(n)$ tid. I resten af algoritmen kan en knude ikke indsættes i køen mere end én gang (pga. farvemærkningen), så maksimalt arbejde er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Definition: $\delta(s, v)$ er længden af en korteste sti, *målt i antal kanter*, fra startknuden s til knuden v . Findes ingen sti, defineres $\delta(s, v) = \infty$.

Sætning:

1. Når BFS stopper, har den opdaget alle knuder v for hvilke der findes en sti fra startknuden s til v .
2. En sti af længde $v.d$ kan for alle opdagede knuder findes (i baglæns rækkefølge) ved at følge predecessor-referencer $v.d$ gange, startende fra v (dvs. $v.\pi$, $(v.\pi).\pi$, $((v.\pi).\pi).\pi, \dots$), og $.d$ -værdierne for knuderne på denne sti falder med én for hvert skridt.
3. Når BFS stopper, gælder $v.d = \delta(s, v)$ for alle knuder.

Egenskaber

Køretid: $O(n + m)$

Bevis: Initialisering tager $O(n)$ tid. I resten af algoritmen kan en knude ikke indsættes i køen mere end én gang (pga. farvemærkningen), så maksimalt arbejde er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Definition: $\delta(s, v)$ er længden af en korteste sti, *målt i antal kanter*, fra startknuden s til knuden v . Findes ingen sti, defineres $\delta(s, v) = \infty$.

Sætning:

1. Når BFS stopper, har den opdaget alle knuder v for hvilke der findes en sti fra startknuden s til v .
2. En sti af længde $v.d$ kan for alle opdagede knuder findes (i baglæns rækkefølge) ved at følge predecessor-referencer $v.d$ gange, startende fra v (dvs. $v.\pi$, $(v.\pi).\pi$, $((v.\pi).\pi).\pi, \dots$), og $.d$ -værdierne for knuderne på denne sti falder med én for hvert skridt.
3. Når BFS stopper, gælder $v.d = \delta(s, v)$ for alle knuder.

Dvs. BFS kan finde korteste veje (målt i antal kanter) fra alle v til s .

Bevis

Bemærk først at en knude v indsættes i køen én gang, så $v.d$ og $v.\pi$ gives en værdi præcis én gang, nemlig når v opdages.

Bevis

Bemærk først at en knude v indsættes i køen én gang, så $v.d$ og $v.\pi$ gives en værdi præcis én gang, nemlig når v opdages.

Punkt 1):

Bevis

Bemærk først at en knude v indsættes i køen én gang, så $v.d$ og $v.\pi$ gives en værdi præcis én gang, nemlig når v opdages.

Punkt 1):

Antag der findes en knude, som kan nås med en sti fra s , men som BFS ikke opdager. Lad v være første knude (set fra s) på denne sti som BFS ikke når, og lad u være knuden før på stien. Da u blev taget ud af køen, burde v (som er i u 's naboliste) være opdaget. Modstrid.

Bevis

Bemærk først at en knude v indsættes i køen én gang, så $v.d$ og $v.\pi$ gives en værdi præcis én gang, nemlig når v opdages.

Punkt 1):

Antag der findes en knude, som kan nås med en sti fra s , men som BFS ikke opdager. Lad v være første knude (set fra s) på denne sti som BFS ikke når, og lad u være knuden før på stien. Da u blev taget ud af køen, burde v (som er i u 's naboliste) være opdaget. Modstrid.

Punkt 2):

Bevis

Bemærk først at en knude v indsættes i køen én gang, så $v.d$ og $v.\pi$ gives en værdi præcis én gang, nemlig når v opdages.

Punkt 1):

Antag der findes en knude, som kan nås med en sti fra s , men som BFS ikke opdager. Lad v være første knude (set fra s) på denne sti som BFS ikke når, og lad u være knuden før på stien. Da u blev taget ud af køen, burde v (som er i u 's naboliste) være opdaget. Modstrid.

Punkt 2):

Induktion over antal indsættelser i køen: Når en knude u udtages af køen, og en knude v fra u 's naboliste indsættes i køen, sættes $v.d = u.d + 1$ og $v.\pi = u$. Hvis udsagnet gælder for u , kommer det klart til at gælde for v . Basis er første indsættelse (af s , for hvilken udsagnet gælder pga. initialiseringen i BFS).

Bevis

Punkt 3):

Bevis

Punkt 3):

Definér for heltal $i \geq 0$:

- ▶ $\Delta_i = \{v \in V \mid \delta(s, v) = i\}$, dvs. de knuder, som faktisk har afstand i til s .
- ▶ $D_i = \{v \in V \mid v.d = i\}$, dvs. de knuder, som algoritmen påstår har afstand i til s .

Bevis

Punkt 3):

Definér for heltal $i \geq 0$:

- ▶ $\Delta_i = \{v \in V \mid \delta(s, v) = i\}$, dvs. de knuder, som faktisk har afstand i til s .
- ▶ $D_i = \{v \in V \mid v.d = i\}$, dvs. de knuder, som algoritmen påstår har afstand i til s .

Vi viser via induktion på i at for alle i gælder

$$\Delta_i = D_i.$$

Bevis

Punkt 3):

Definér for heltal $i \geq 0$:

- ▶ $\Delta_i = \{v \in V \mid \delta(s, v) = i\}$, dvs. de knuder, som faktisk har afstand i til s .
- ▶ $D_i = \{v \in V \mid v.d = i\}$, dvs. de knuder, som algoritmen påstår har afstand i til s .

Vi viser via induktion på i at for alle i gælder

$$\Delta_i = D_i.$$

Dette vil medføre punkt 3) for alle knuder v for hvilke der findes en sti fra s til v . For resten af knuderne forbliver $v.d = \infty$ efter initialisering [pga. punkt 2)], så punkt 3) gælder også for dem.

Bevis

Punkt 3):

Definér for heltal $i \geq 0$:

- ▶ $\Delta_i = \{v \in V \mid \delta(s, v) = i\}$, dvs. de knuder, som faktisk har afstand i til s .
- ▶ $D_i = \{v \in V \mid v.d = i\}$, dvs. de knuder, som algoritmen påstår har afstand i til s .

Vi viser via induktion på i at for alle i gælder

$$\Delta_i = D_i.$$

Dette vil medføre punkt 3) for alle knuder v for hvilke der findes en sti fra s til v . For resten af knuderne forbliver $v.d = \infty$ efter initialisering [pga. punkt 2)], så punkt 3) gælder også for dem.

Observér først at BFS-algoritmen starter med $D_0 = \{s\}$ i køen, og derefter for $i = 0, 1, 2, 3, \dots$ udtager alle knuder i D_i mens den indsætter alle knuder i D_{i+1} .

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Basis ($i=0$): klar, da $D_0 = \{s\} = \Delta_0$.

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Basis ($i=0$): klar, da $D_0 = \{s\} = \Delta_0$.

Induktionsskridt: antag udsagn er sandt for $i - 1$ og lavere, vis det er sandt for i .

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Basis ($i=0$): klar, da $D_0 = \{s\} = \Delta_0$.

Induktionsskridt: antag udsagn er sandt for $i - 1$ og lavere, vis det er sandt for i .

For en knude $v \in D_i$ eksisterer via punkt 2) en sti fra s til v af længde i . Derfor gælder $\delta(s, v) \leq i$. Vi kan ikke have $\delta(s, v) \leq i - 1$, da induktionsantagelse så ville give $v.d = \delta(s, v) \leq i - 1$, i modstrid med $v \in D_i$. Derfor er $\delta(s, v) = i$. Dette viser $D_i \subseteq \Delta_i$.

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Basis ($i=0$): klar, da $D_0 = \{s\} = \Delta_0$.

Induktionsskridt: antag udsagn er sandt for $i - 1$ og lavere, vis det er sandt for i .

For en knude $v \in D_i$ eksisterer via punkt 2) en sti fra s til v af længde i . Derfor gælder $\delta(s, v) \leq i$. Vi kan ikke have $\delta(s, v) \leq i - 1$, da induktionsantagelse så ville give $v.d = \delta(s, v) \leq i - 1$, i modstrid med $v \in D_i$. Derfor er $\delta(s, v) = i$. Dette viser $D_i \subseteq \Delta_i$.

For enhver knude $u \in \Delta_i$ eksisterer pr. definition af δ en sti fra s til u af længde i . For næstsidste knude w på denne sti gælder $\delta(s, w) = i - 1$. Fra induktionsantagelsen får vi at $w.d = \delta(s, w)$. Da w blev taget ud af køen, var u (en nabo til w) enten hvid, eller u var allerede nået fra en knude t , som allerede var taget ud og derfor (via observationen på sidste side) har $t.d \leq w.d$. I begge tilfælde bliver $u.d$ sat til højst $w.d + 1 = \delta(s, w) + 1 = i = \delta(s, u)$. Fra punkt 2) haves $u.d \geq \delta(s, u)$. I alt gælder $u.d = \delta(s, u)$. Dette viser $\Delta_i \subseteq D_i$.

Bevis

Induktionsbevis for $\Delta_i = D_i$:

Basis ($i=0$): klar, da $D_0 = \{s\} = \Delta_0$.

Induktionsskridt: antag udsagn er sandt for $i - 1$ og lavere, vis det er sandt for i .

For en knude $v \in D_i$ eksisterer via punkt 2) en sti fra s til v af længde i . Derfor gælder $\delta(s, v) \leq i$. Vi kan ikke have $\delta(s, v) \leq i - 1$, da induktionsantagelse så ville give $v.d = \delta(s, v) \leq i - 1$, i modstrid med $v \in D_i$. Derfor er $\delta(s, v) = i$. Dette viser $D_i \subseteq \Delta_i$.

For enhver knude $u \in \Delta_i$ eksisterer pr. definition af δ en sti fra s til u af længde i . For næstsidste knude w på denne sti gælder $\delta(s, w) = i - 1$. Fra induktionsantagelsen får vi at $w.d = \delta(s, w)$. Da w blev taget ud af køen, var u (en nabo til w) enten hvid, eller u var allerede nået fra en knude t , som allerede var taget ud og derfor (via observationen på sidste side) har $t.d \leq w.d$. I begge tilfælde bliver $u.d$ sat til højst $w.d + 1 = \delta(s, w) + 1 = i = \delta(s, u)$. Fra punkt 2) haves $u.d \geq \delta(s, u)$. I alt gælder $u.d = \delta(s, u)$. Dette viser $\Delta_i \subseteq D_i$.

Alt i alt: $\Delta_i = D_i$.

Dybde-Først-Søgning (DFS)

Holder de grå knuder (fronten) i en **STAK**.

Dybde-Først-Søgning (DFS)

Holder de grå knuder (fronten) i en **STAK**.

Stakken er implicit i den rekursive formulering nedenfor (dvs. er lig rekursionsstakken), men kan også kodes eksplicit.

Mere præcist: elementerne på stakken er de grå knuder, hver med en delvist gennemløbet naboliste [gennemløbet i for-løkken i DFS-VISIT].

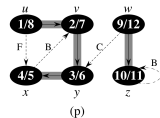
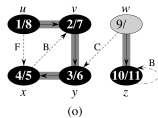
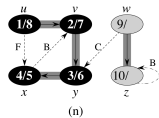
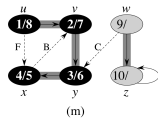
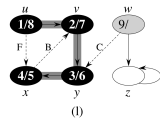
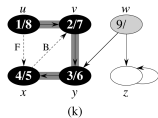
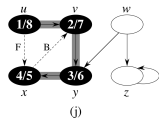
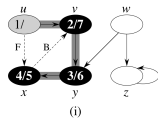
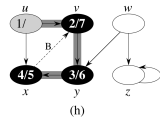
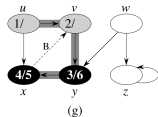
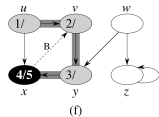
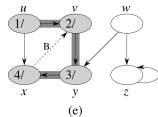
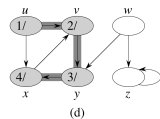
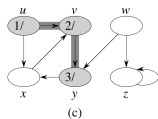
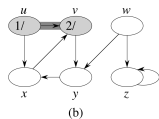
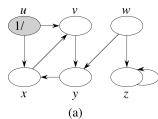
DFS tilføjer også timestamps $u.d$ for “discovery” (hvid \rightarrow grå) og $u.f$ for “finish” (grå \rightarrow sort) til alle knuder u .

```
DFS(G)
1  for each vertex  $u \in G.V$ 
2     $u.color = WHITE$ 
3     $u.\pi = NIL$ 
4   $time = 0$ 
5  for each vertex  $u \in G.V$ 
6    if  $u.color == WHITE$ 
7      DFS-VISIT( $G, u$ )

DFS-VISIT( $G, u$ )
1   $time = time + 1$  // white vertex  $u$  has just been discovered
2   $u.d = time$ 
3   $u.color = GRAY$ 
4  for each  $v \in G.Adj[u]$  // explore edge ( $u, v$ )
5    if  $v.color == WHITE$ 
6       $v.\pi = u$ 
7      DFS-VISIT( $G, v$ )
8   $u.color = BLACK$  // blacken  $u$ ; it is finished
9   $time = time + 1$ 
10  $u.f = time$ 
```


Dybde-Først-Søgning (DFS)

Eksempel:



Egenskaber

Køretid: $O(n + m)$

Egenskaber

Køretid: $O(n + m)$

Bevis: Den ydre algoritme DFS tager $O(n)$ tid. Hovedalgoritmen DFS-VISIT kan ikke kaldes på en knude mere end én gang (den kaldes kun på hvide knuder, som straks gøres grå), så arbejdet dér er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Egenskaber

Køretid: $O(n + m)$

Bevis: Den ydre algoritme DFS tager $O(n)$ tid. Hovedalgoritmen DFS-VISIT kan ikke kaldes på en knude mere end én gang (den kaldes kun på hvide knuder, som straks gøres grå), så arbejdet dér er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Observér:

- ▶ Discovery (hvid \rightarrow grå) af $v =$ sæt $v.d =$ kald af DFS-VISIT på $v =$ PUSH af v på stakken.
- ▶ Finish (grå \rightarrow sort) af $v =$ sæt $v.f =$ retur fra kald af DFS-VISIT på $v =$ POP af v fra stakken.

Egenskaber

Køretid: $O(n + m)$

Bevis: Den ydre algoritme DFS tager $O(n)$ tid. Hovedalgoritmen DFS-VISIT kan ikke kaldes på en knude mere end én gang (den kaldes kun på hvide knuder, som straks gøres grå), så arbejdet dér er proportionalt med ét gennemløb af alle nabolister, dvs. $O(m)$ arbejde.

Observér:

- ▶ Discovery (hvid \rightarrow grå) af $v =$ sæt $v.d =$ kald af DFS-VISIT på $v =$ PUSH af v på stakken.
- ▶ Finish (grå \rightarrow sort) af $v =$ sæt $v.f =$ retur fra kald af DFS-VISIT på $v =$ POP af v fra stakken.

Kanten $(v, v.\pi)$ sættes ved kald af DFS-VISIT på v . Af dette, samt ovenstående:

- ▶ Kanterne $(v, v.\pi)$ udgør præcis rekursionstræerne for DFS-VISIT (ét træ for hvert kald fra DFS).
- ▶ Intervallet $[v.d, v.f]$ er den periode v er på stakken.
- ▶ Knuden v er grå hvis og kun hvis den er på stakken.

Egenskaber

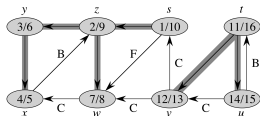
Hvis u og v er på en stak samtidig, og v er pushed sidst, må v poppes før u kan poppes.

Af dette, samt at push/pop sætter d/f , følger at for alle knuder u og v må intervallerne $[u.d, u.f]$ og $[v.d, v.f]$ enten være disjunkte (u og v var ikke på stakken samtidig) eller det ene må være helt indeholdt i den andet (knuden med det største interval kom på stakken først).

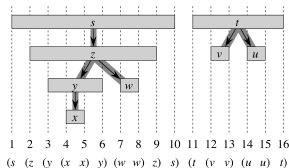
Egenskaber

Hvis u og v er på en stak samtidig, og v er pushed sidst, må v poppes før u kan poppes.

Af dette, samt at push/pop sætter d/f , følger at for alle knuder u og v må intervallerne $[u.d, u.f]$ og $[v.d, v.f]$ enten være disjunkte (u og v var ikke på stakken samtidig) eller det ene må være helt indeholdt i den andet (knuden med det største interval kom på stakken først).



Discovery- og finish-tider er derfor nestede som parenteser er det.

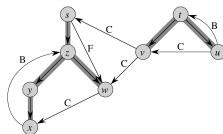
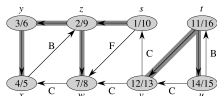


Egenskaber

Når en kant (u, v) undersøges fra u haves flg. tilfælde:

1. *tree-kanter*: v hvid. Her er $u.d < v.d = nu < v.f < u.f$.
2. *back-kanter*: v er grå (er på stak). Her er $v.d < u.d < nu < u.f < v.f$.
3. *forward-kanter*: v er sort (den er ikke længere på stak, men har været det sammen med u). Her er $u.d < v.d < v.f < nu < u.f$.
4. *cross-kanter*: v er sort (den er ikke længere på stak, og har ikke været det sammen med u). Her er $v.d < v.f < u.d < nu < u.f$.

Bemærk at disse cases kan genkendes under DFS via farvningen og d -værdierne i knuder.



Egenskaber

1. *tree-kanter*: v hvid.
2. *back-kanter*: v er grå (er på stak).
3. *forward-kanter*: v er sort (den er ikke længere på stak, men har været det sammen med u).
4. *cross-kanter*: v er sort (den er ikke længere på stak, og har ikke været det sammen med u).

For *uorienterede grafer* er der kun *tree-kanter* og *back-kanter* (såfremt en kant kategoriseres første gang den undersøges fra én af dens ender).

Dette følger af at u allerede må være blevet undersøgt fra v hvis v er sort (hele nabolisten er gennemløbet) og kanten (v, u) må derfor allerede være kategoriseret. Derved kan 3 og 4 ikke opstå.

DAGs og topologisk sortering

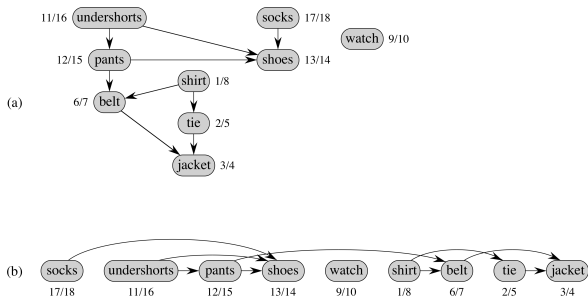
DAG = Directed Acyclic Graph. En orienteret graf uden kredse (cycles).

Topologisk sortering af en DAG: en lineær ordning af knuderne så alle kanter går fra venstre til højre.

DAGs og topologisk sortering

DAG = Directed Acyclic Graph. En orienteret graf uden kredse (cycles).

Topologisk sortering af en DAG: en lineær ordning af knuderne så alle kanter går fra venstre til højre.



DAGs og topologisk sortering

Lemma: En orienteret graf har en kreds (cycle) \Leftrightarrow der findes back-edges under et DFS-gennemløb.

DAGs og topologisk sortering

Lemma: En orienteret graf har en kreds (cycle) \Leftrightarrow der findes back-edges under et DFS-gennemløb.

Bevis:

\Rightarrow : DFS (med ydre loop) opdager alle knuder. Se på første knude v i kredsen som bliver grå - dvs. alle andre knuder u i kredsen har $v.d < u.d$. Da intervaller enten er helt indeholdt i hinanden eller er disjunkte, gælder enten $v.d < u.d < u.f < v.f$ eller $v.f < u.d$.

Antag det sidste tilfælde forekommer, og se på den første (set fra v) sådanne knude u i kredsen, og lad w være dens forgænger (evt. v selv). Da haves $w.f \leq v.f < u.d$, men pga. kanten (w, u) kan dette ikke forekomme (kanten må blive undersøgt, og u opdaget inden w er færdig).

Så kun første tilfælde forekommer. Specielt gælder dette sidste knude u' i kredsen (som peger på v), hvorved kanten (u', v) erklæres en backedge.

DAGs og topologisk sortering

Lemma: En orienteret graf har en kreds (cycle) \Leftrightarrow der findes back-edges under et DFS-gennemløb.

Bevis:

\Rightarrow : DFS (med ydre loop) opdager alle knuder. Se på første knude v i kredsen som bliver grå - dvs. alle andre knuder u i kredsen har $v.d < u.d$. Da intervaller enten er helt indeholdt i hinanden eller er disjunkte, gælder enten $v.d < u.d < u.f < v.f$ eller $v.f < u.d$.

Antag det sidste tilfælde forekommer, og se på den første (set fra v) sådanne knude u i kredsen, og lad w være dens forgænger (evt. v selv). Da haves $w.f \leq v.f < u.d$, men pga. kanten (w, u) kan dette ikke forekomme (kanten må blive undersøgt, og u opdaget inden w er færdig).

Så kun første tilfælde forekommer. Specielt gælder dette sidste knude u' i kredsen (som peger på v), hvorved kanten (u', v) erklæres en backedge.

\Leftarrow : Når en back-edge findes: Der er en kreds af trækanter (mellem knuderne som lige nu er på stakken) og én back-kant.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f < v.f \Leftrightarrow$ kanten er en back-edge.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f < v.f \Leftrightarrow$ kanten er en back-edge.

Bevis: Check de fire cases for kanter (tree, back, forward, cross) og deres ordning af $u.f$ og $v.f$, se tidligere.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f < v.f \Leftrightarrow$ kanten er en back-edge.

Bevis: Check de fire cases for kanter (tree, back, forward, cross) og deres ordning af $u.f$ og $v.f$, se tidligere.

Korollar til to foregående lemmaer: Graf er en DAG \Leftrightarrow DFS finder ingen back-edges \Leftrightarrow ordning af knuder efter faldende finish-tider giver en topologisk sortering.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f < v.f \Leftrightarrow$ kanten er en back-edge.

Bevis: Check de fire cases for kanter (tree, back, forward, cross) og deres ordning af $u.f$ og $v.f$, se tidligere.

Korollar til to foregående lemmaer: Graf er en DAG \Leftrightarrow DFS finder ingen back-edges \Leftrightarrow ordning af knuder efter faldende finish-tider giver en topologisk sortering.

TOPOLOGICAL-SORT(G)

- 1 call DFS(G) to compute finishing times $v.f$ for each vertex v
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 **return** the linked list of vertices

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f < v.f \Leftrightarrow$ kanten er en back-edge.

Bevis: Check de fire cases for kanter (tree, back, forward, cross) og deres ordning af $u.f$ og $v.f$, se tidligere.

Korollar til to foregående lemmaer: Graf er en DAG \Leftrightarrow DFS finder ingen back-edges \Leftrightarrow ordning af knuder efter faldende finish-tider giver en topologisk sortering.

TOPOLOGICAL-SORT(G)

- 1 call DFS(G) to compute finishing times $v.f$ for each vertex v
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 **return** the linked list of vertices

Tid: $O(n + m)$.