Grafer og graf-gennemløb

Grafer

En mængde V af *knuder* (vertices).

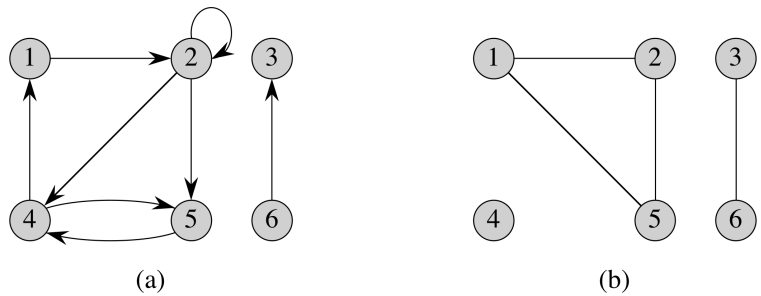
En mængde $E \in V \times V$ af *kanter* (edges). Dvs. ordnede par af knuder.

Grafer

En mængde V af *knuder* (vertices).

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

Figur:



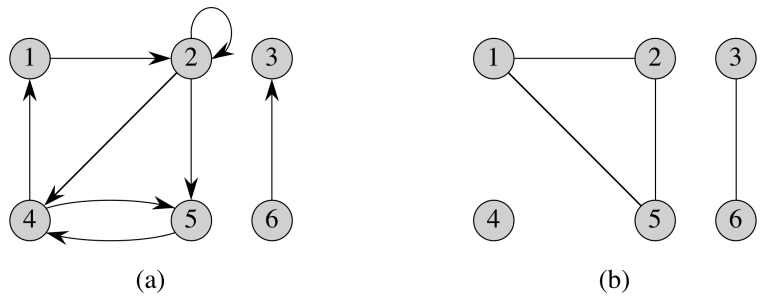
- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.

Grafer

En mængde V af *knuder* (vertices).

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

Figur:



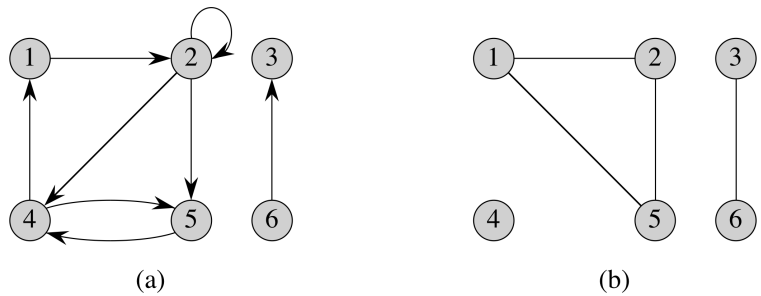
- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- Orienteret vs. uorienteret.

Grafer

En mængde V af *knuder* (vertices).

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

Figur:



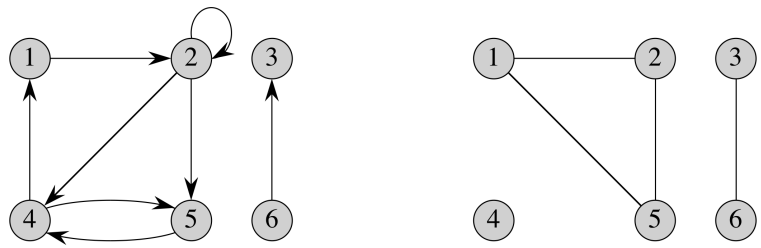
- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- Orienteret vs. uorienteret.
- Evt. loops, multiple kanter.

Grafer

En mængde V af *knuder* (vertices).

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

Figur:



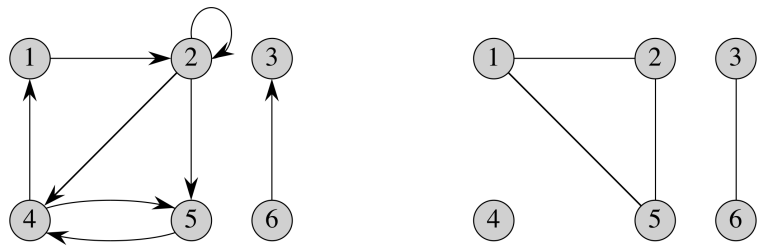
- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- Orienteret vs. uorienteret.
- Evt. loops, multiple kanter.
- $0 \leq m \leq n(n-1)$ og $0 \leq m \leq n(n-1)/2$

Grafer

En mængde V af *knuder* (vertices).

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

Figur:

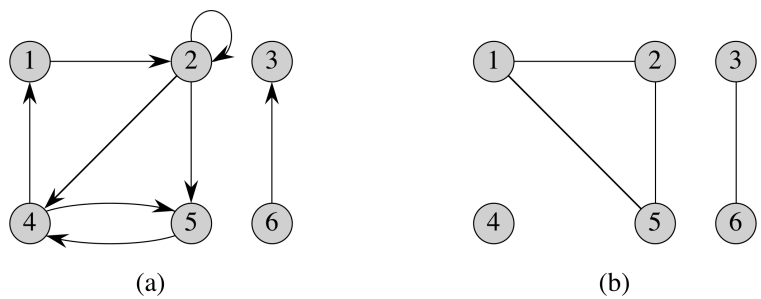


- Terminologi: $n = |V|$, $m = |E|$. Eller V og E genbruges/misbruges til også at betyde $|V|$ og $|E|$.
- Orienteret vs. uorienteret.
- Evt. loops, multiple kanter.
- $0 \leq m \leq n(n-1)$ og $0 \leq m \leq n(n-1)/2$
- Vægtede grafer.

Grafer

Modeller for mange ting:

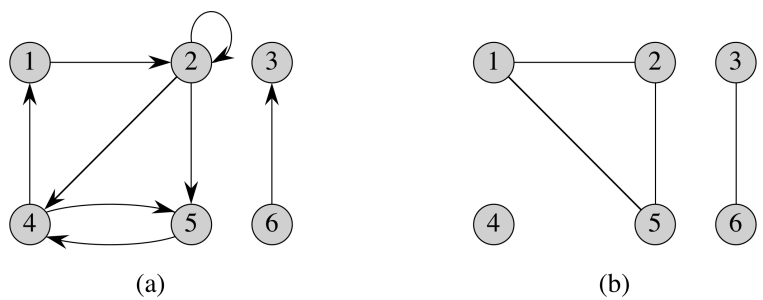
- Ledningsnet (telefon, strøm, olie, vand,...).



Grafer

Modeller for mange ting:

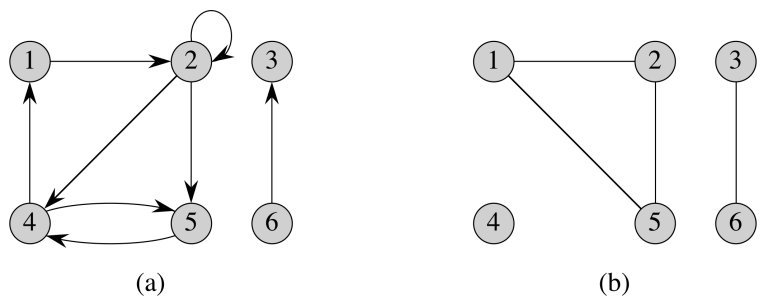
- Ledningsnet (telefon, strøm, olie, vand,...).
- Bekendtskaber.



Grafer

Modeller for mange ting:

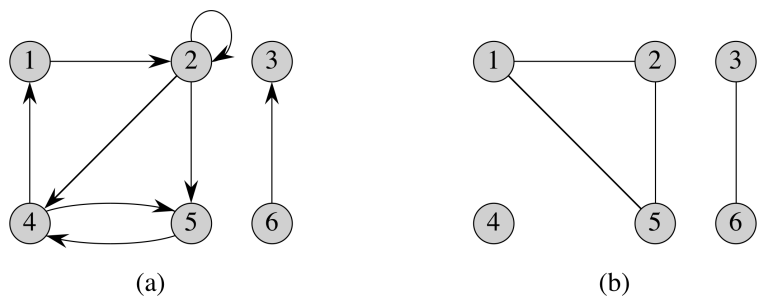
- Ledningsnet (telefon, strøm, olie, vand,...).
- Bekendtskaber.
- Medforfatterskaber.



Grafer

Modeller for mange ting:

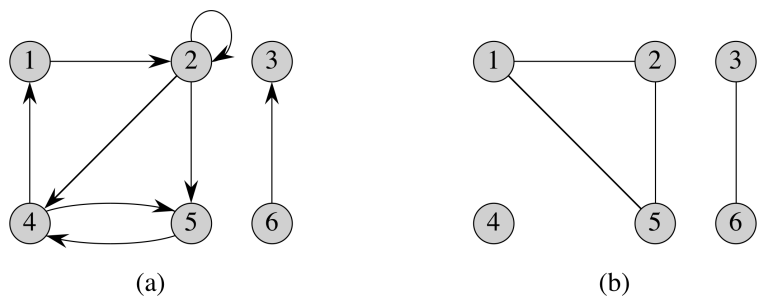
- Ledningsnet (telefon, strøm, olie, vand,...).
- Bekendtskaber.
- Medforfatterskaber.
- WWW-grafen af sider og links.



Grafer

Modeller for mange ting:

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



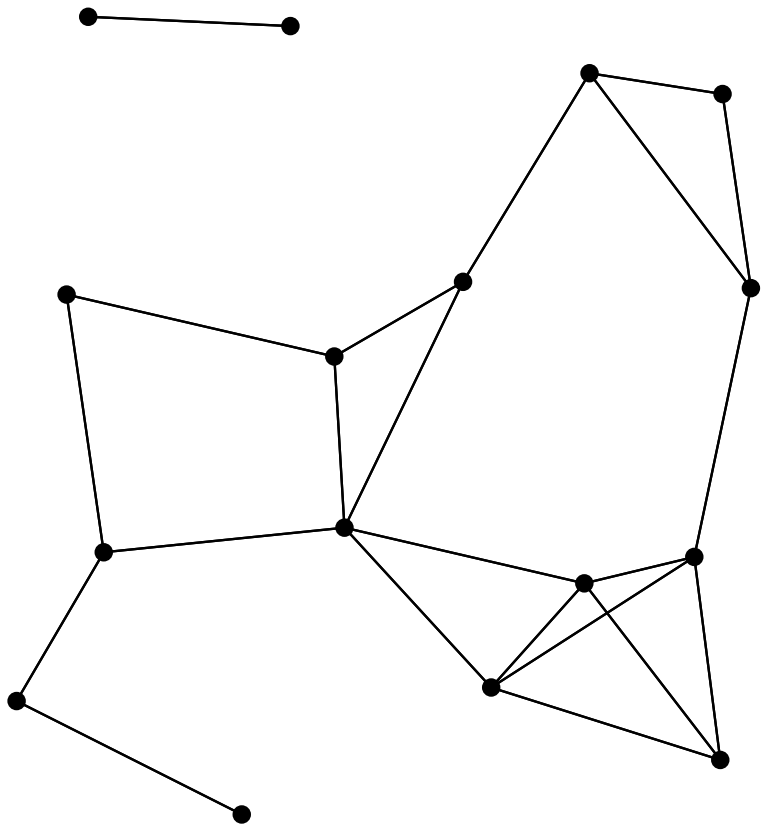
Algoritmiske spørgsmål

- 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).

⋮

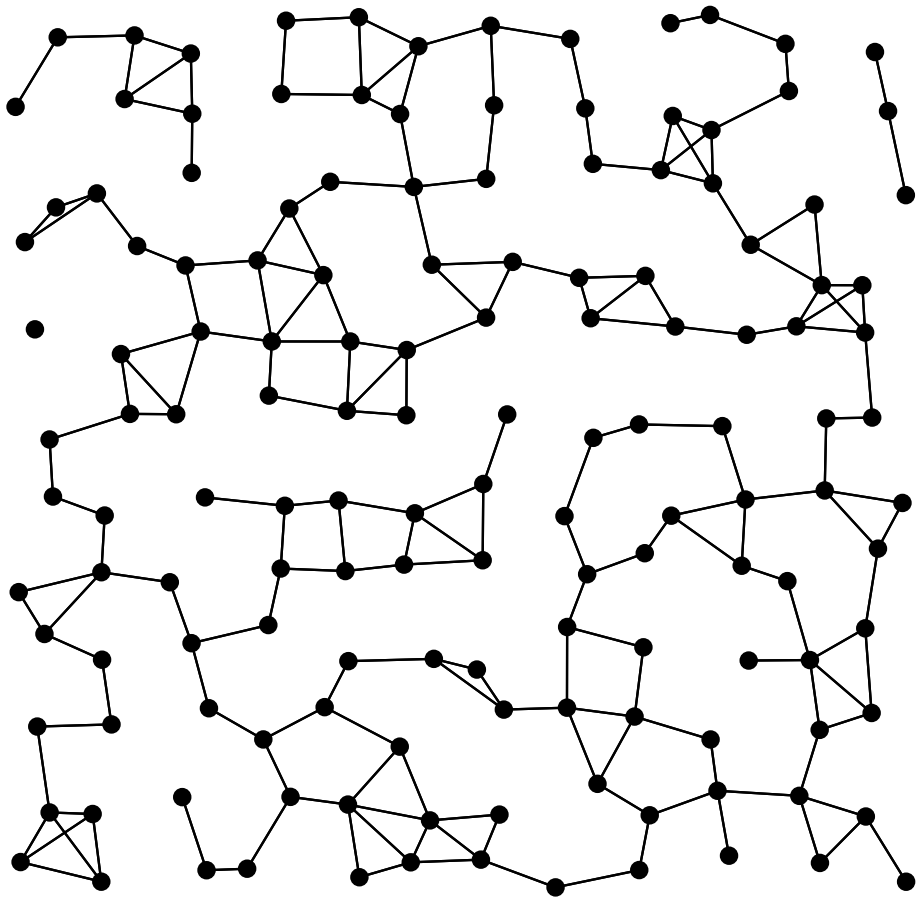
Algoritmiske spørgsmål

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



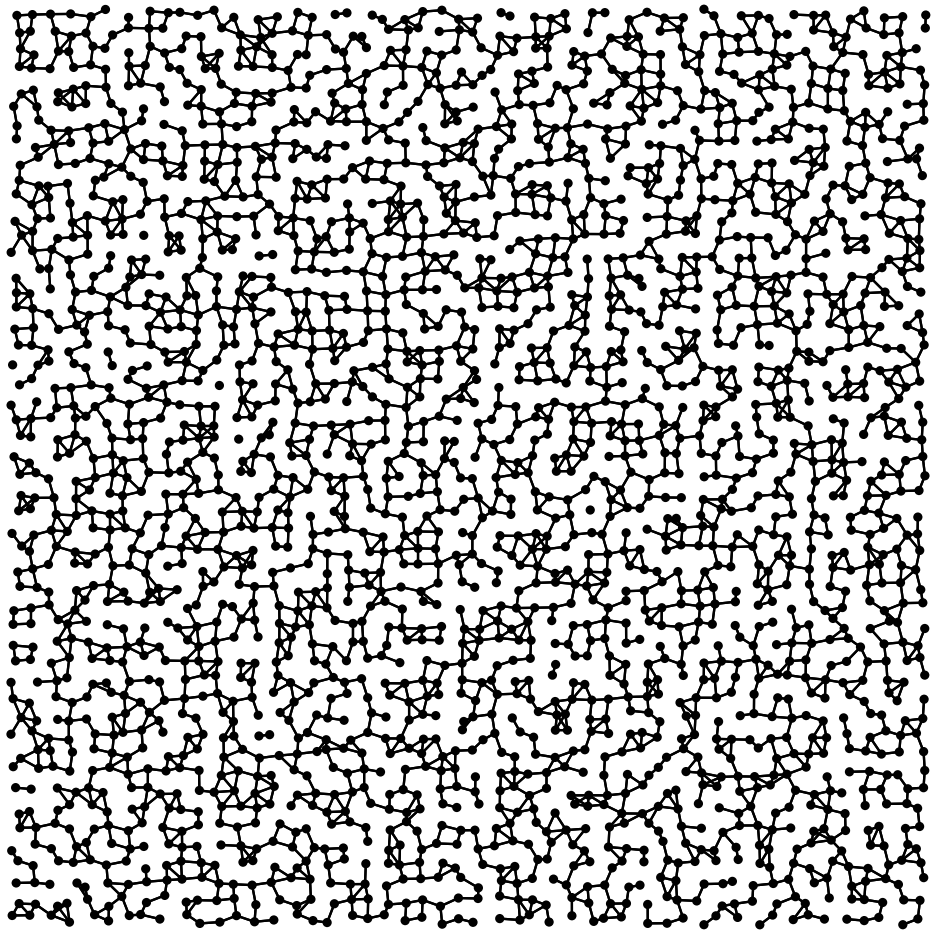
Algoritmiske spørgsmål

- 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.



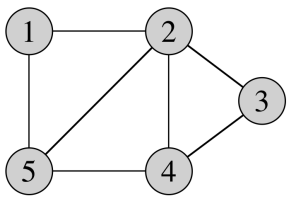
Datastrukturer for grafer

Graf-repræsentationer:

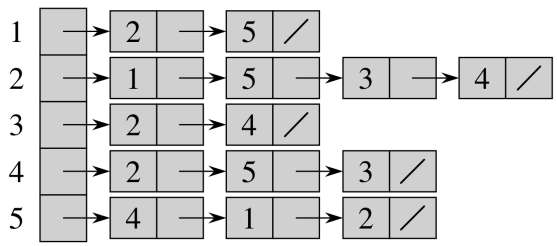
Datastrukturer for grafer

Graf-repræsentationer:

Adjancency list og adjancency matrix



(a)



(b)

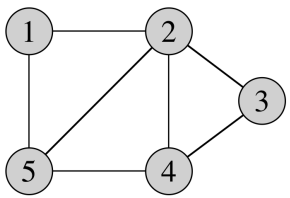
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

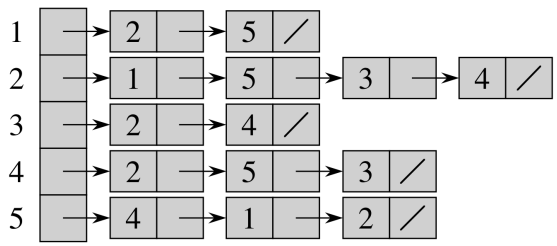
Datastrukturer for grafer

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

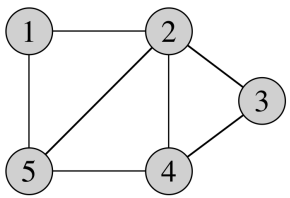
(c)

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

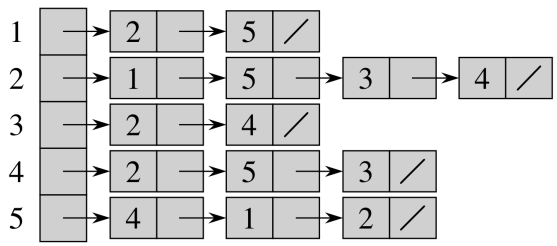
Datastrukturer for grafer

Graf-repræsentationer:

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 $O(n^2)$.

Hvis ikke andet oplyses, bruges adjancency 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

Opgave: givet en graf i adjacency list repræsentation, besøg alle knuder.

Generel idé: Besøg en startknude s. Brug kanter i nabolisterne for besøgte knuder til at besøge flere knuder.

- Hvide knuder: endnu ikke besøgt
- Grå knuder: besøgt, men ikke alle kanter i naboliste brugt
- Sorte knuder: besøgt, alle kanter i naboliste brugt

```
GenericGraphTraversal1(s)
    Gør s grå og resten af knuderne hvide
    while der findes grå knuder:
        vælg en grå knude v
        if v's naboliste er brugt op
            gør v sort
        else
            vælg en ubrugt kant (v, u) fra v's naboliste
            if u hvid:
                gør u grå
```

Grafgennemløb

- Hvide knuder: endnu ikke besøgt
- Grå knuder: besøgt, men ikke alle kanter i naboliste brugt
- Sorte knuder: besøgt, alle kanter i naboliste brugt

```
GenericGraphTraversal1(s)
    Gør s grå og resten af knuderne hvide
    while der findes grå knuder:
        vælg en grå knude v
        if v's naboliste er brugt op
            gør v sort
        else
            vælg en ubrugt kant (v, u) fra v's naboliste
            if u hvid:
                gør u grå
```

En knudes livs-cyklus: hvid $\rightarrow$ grå $\rightarrow$ sort. Når algoritmen stopper, er alle knuder enten hvide eller sorte.

Farven for en knude v opbevares i et felt $v.color$.

Grafgennemløb

Vi skal senere i kurset møde tre varianter, som bruger forskellige strategier for at vælge næste kant (v, u) at bruge, dvs. for valgene (*):

```
GenericGraphTraversal1(s)
    Gør s grå og resten af knuderne hvide
    while der findes grå knuder:
        vælg en grå knude v (*)
        if v's naboliste er brugt op
            gør v sort
        else
            vælg en ubrugt kant (v, u) fra v's naboliste (*)
            if u hvid:
                gør u grå
```

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

Hvor langt når vi rundt i grafen?

Vi når alt, som kan nås fra s:

Sætning: Hvis der er en sti fra s til v, vil v være sort (og dermed besøgt) når GenericGraphTraversal1(s) stopper.

Bevis: Antag v ikke er sort, når algoritmen stopper. Så er v hvid (når algoritmen stopper, er alle knuder enten hvide eller sorte). Lad w være første knude på stien fra s til v som er hvid (da v er hvid, findes w). Lad u være knuden lige før w på stien (da s starter grå, må den nu være sort, så w kan ikke være s, og derfor findes u). Da u ikke er hvid, er den sort. Den kan kun være sort, hvis kanten (u, w) er brugt. Dermed kan v ikke være hvid.

For at nå rundt i *hele* grafen:

```
GenericGraphTraversalGlobal()
    Gør alle knuder hvide
    for alle knuder s:
        if s hvid:
                GenericGraphTraversal2(s)
```

```
GenericGraphTraversal2(s)
    Gør s grå
    while der findes grå knuder:
        vælg en grå knude v (*)
        if v's naboliste er brugt op
            gør v sort
        else
            vælg en ubrugt kant (v, u) fra v's naboliste (*)
            if u hvid:
                gør u grå
```

Hvis (*) tager tid $O(1)$, er samlet køretid $O(n+m)$. [En kant kan kun vælges én gang (pga. farvemærkningen), så alt arbejde udført i **else**-del tager $O(m)$ tid i alt. Resten tager $O(n)$ tid i alt.]

Hvor langt når vi rundt i grafen per kald?

Sætning: Hvis der ved starten af et kald til GenericGraphTraversal2(s) er en sti fra s til v bestående af hvide knuder (inkl. v), vil v være sort (og dermed besøgt) når GenericGraphTraversal2(s) stopper.

Bevis: Det samme som før. [Antag v ikke er sort, når algoritmen stopper. Så er v hvid (når algoritmen stopper, er alle knuder enten hvide eller sorte). Lad w være første knude på stien fra s til v som er hvid (da v er hvid, findes w). Lad u være knuden lige før w på stien (da s starter grå, må den nu være sort, så w kan ikke være s, og derfor findes u). Da u ikke er hvid, er den sort. Den kan kun være sort, hvis kanten (u, w) er brugt. Dermed kan v ikke være hvid.]

Husk hvem der opdagede hvem:

Når en knude u ($\neq s$) besøges første gang husker den, fra hvilken knude den blev opdaget (dens predecessor) i variablen $u.\pi$. Bemærk at $u.\pi$ højst bliver sat én gang (efter initialisering til NIL).

```
GenericGraphTraversalGlobal()
    Gør alle knuder hvide og sæt deres π til NIL
    for alle knuder s:
        if s hvid:
                GenericGraphTraversal3(s)
```

```
GenericGraphTraversal3(s)
    Gør s grå
    while der findes grå knuder:
        vælg en grå knude v (*)
        if v's naboliste er brugt op
            gør v sort
        else
            vælg en ubrugt kant (v, u) fra v's naboliste (*)
            if u hvid:
                gør u grå
                sæt u.π lig v
```

Husk hvem der opdagede hvem:

Sætning: De knuder, som er opdaget (gjort ikke-hvide) i et kald GenericGraphTraversal3(s), udgør et træ med s som rod og π i opdagede knuder som parent pointers. For hver sti fra en knude v til roden i træet findes den samme sti i grafen, men i modsat retning (fra s til v).

Bevis: Det er nemt at se, at dette udsagn er en invariant som vedligeholdes under kørslen af GenericGraphTraversal3(s).

Bemærk at i GenericGraphTraversalGlobal() kaldes GenericGraphTraversal3(s) gentagne gange. Hvert kald giver ét træ. Træerne fra forskellige kald deler ikke knuder, og tilsammen indeholder de alle knuder i grafen.

Bredde-Først-Søgning (BFS)

Strategi: Hold de grå knuder i en KØ, brug nabolister op med det samme.

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

Bredde-Først-Søgning (BFS)

Strategi: Hold de grå knuder i en KØ, brug nabolister op med det samme.

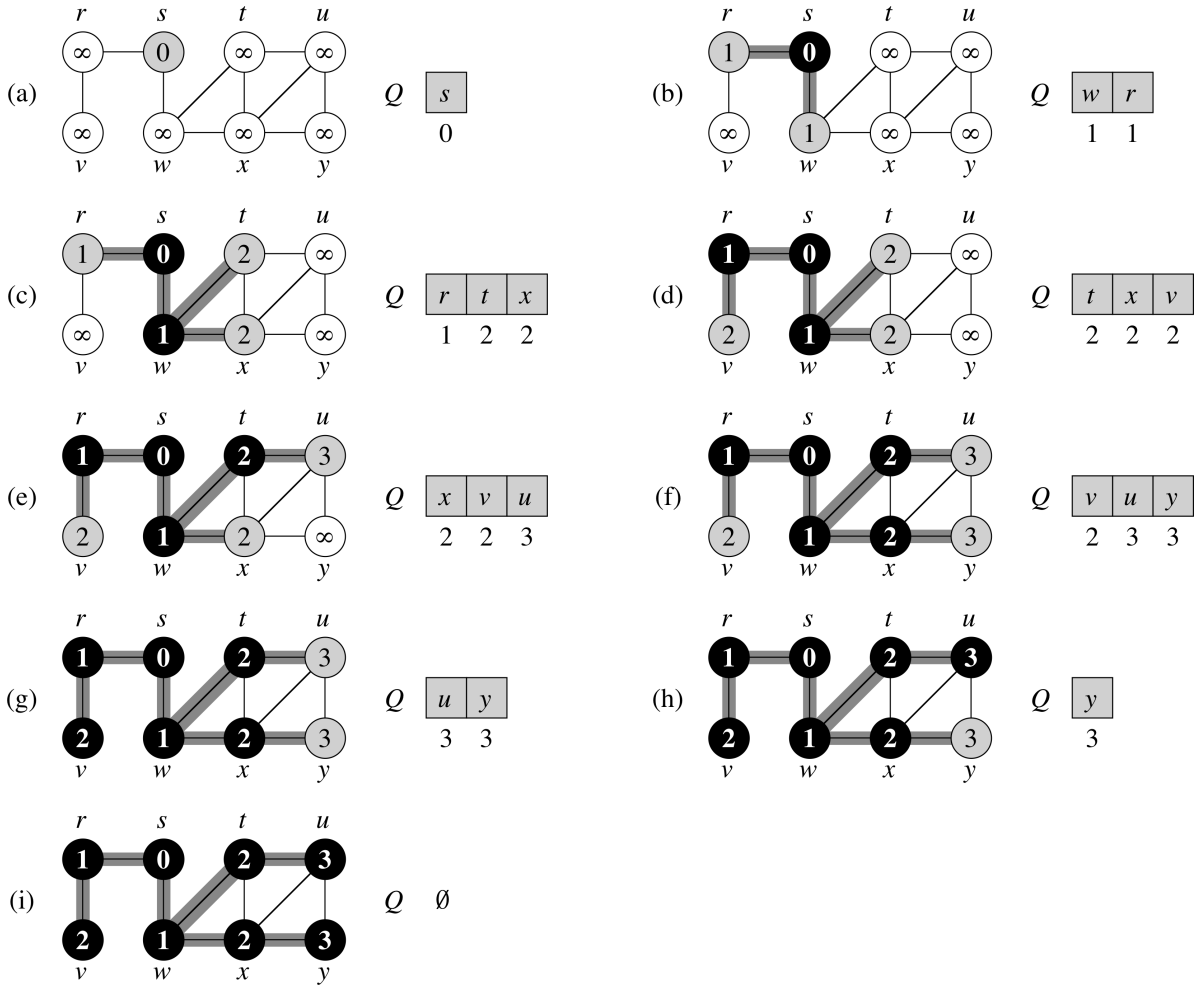
Tilføj også en variabel $v.d$ til alle knuder v (d for distance.)

Mest brugt er versionen uden GLOBAL-del (for BFS er vi ofte mere interesserede i ét bestemt s fremfor at komme hele grafen rundt):

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

Bredde-Først-Søgning (BFS)

Eksempel:



Bredde-Først-Søgning (BFS)

For BFS kan sætningen om $\textsc{GenericGraphTraversal3}(s)$ udvides til også at sige noget om værdierne af $v.d$:

Sætning: De knuder, som er opdaget (gjort ikke-hvide) i et kald $\textsc{GenericGraphTraversal3}(s)$, udgør et træ med s som rod og π i opdagede knuder som parent pointers. For hver sti fra en knude v til roden i træet findes den samme sti i grafen, men i modsat retning (fra s til v) og $v.d$ indeholder længden af denne sti.

Bevis: Det er nemt at se, at dette udsagn er en invariant som vedligeholdes under kørslen af $\mathrm{BFS}(G, s)$.

Bemærk at $v.d$ højst bliver sat én gang (efter initialisering til $-\infty$).

Egenskaber for BFS

Køretid: $O(n + m)$.

Beviset er det samme som under GenericGraphTraversalGlobal, da valgene (*) i BFS tager $O(1)$ tid. I BFS bruger man som sagt ofte kun at kalde på én startknude s, dvs. uden at bruge Global-delen. Men køretiden kan kun falde ved dette.

Egenskaber for BFS

Køretid: $O(n + m)$.

Beviset er det samme som under GenericGraphTraversalGlobal, da valgene (*) i BFS tager $O(1)$ tid. I BFS bruger man som sagt ofte kun at kalde på én startknude s, dvs. uden at bruge Global-delen. Men køretiden kan kun falde ved dette.

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: 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 for sætning

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 for sætning

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 på næste side at for alle i gælder

$$\Delta_i = D_i.$$

Bevis for sætning

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 på næste side at for alle i gælder

$$\Delta_i = D_i.$$

Dette medfører sætningen for alle knuder v for hvilke der findes en sti fra s til v. Resten af knuderne kan ikke være opdaget (pga. tidligere sætning) og $v.d = \infty$ fra initialisering er derfor deres endelige værdi. Så sætningen gælder også for dem.

Bevis for sætning

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 på næste side at for alle i gælder

$$\Delta_i = D_i.$$

Dette medfører sætningen for alle knuder v for hvilke der findes en sti fra s til v. Resten af knuderne kan ikke være opdaget (pga. tidligere sætning) og $v.d = \infty$ fra initialisering er derfor deres endelige værdi. Så sætningen gælder også for dem.

Observér først at BFS-algoritment starter med $D_0 = \{s\}$ i køen, og derefter for $i = 0, 1, 2, 3, \ldots$ udtager alle knuder i D_i mens den indsætter alle knuder i D_{i+1}. Så d-værdierne for de udtagne knuder stiger monotont.

Bevis for sætning

Induktionsbevis for $\Delta_i = D_i$:

Bevis for sætning

Induktionsbevis for $\Delta_i = D_i$:

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

Bevis for sætning

Induktionsbevis for $\Delta_i = D_i$:

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

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

Bevis for sætning

Induktionsbevis for $\Delta_i = D_i$:

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

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

For en knude $v \in D_i$ er $v.d = i$ og der eksisterer iflg. tidligere sætning 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 for sætning

Induktionsbevis for $\Delta_i = D_i$:

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

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

For en knude $v \in D_i$ er $v.d = i$ og der eksisterer iflg. tidligere sætning 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 $\delta(s,w) = i-1$ (hvis w havde en kortere vej, ville u også have det). 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 derfor allerede var taget ud og derfor (via observationen på sidste side) har $t.d \leq w.d$. I begge tilfælde bliver $u.d$ blev sat til højst $w.d + 1 = \delta(s,w) + 1 = i = \delta(s,u)$. Vi ved (jvf. tidligere sætning) at $u.d \geq \delta(s,u)$. I alt gælder $u.d = \delta(s,u)$. Dette viser $\Delta_i \subseteq D_i$.

Bevis for sætning

Induktionsbevis for $\Delta_i = D_i$:

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

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

For en knude $v \in D_i$ er $v.d = i$ og der eksisterer iflg. tidligere sætning 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 $\delta(s,w) = i-1$ (hvis w havde en kortere vej, ville u også have det). 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 derfor allerede var taget ud og derfor (via observationen på sidste side) har $t.d \leq w.d$. I begge tilfælde bliver $u.d$ blev sat til højst $w.d + 1 = \delta(s,w) + 1 = i = \delta(s,u)$. Vi ved (jvf. tidligere sætning) at $u.d \geq \delta(s,u)$. 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)

Strategi: Hold de grå knuder i en STAK, brug kanter nabolister én for én.

Dybde-Først-Søgning (DFS)

Strategi: Hold de grå knuder i en STAK, brug kanter nabolister én for én.

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, nemlig gennemløbet i for-løkken i DFS-VISIT. [Bemærk: koden til venstre svarer til GLOBAL-delen i terminologien fra tidligere.]

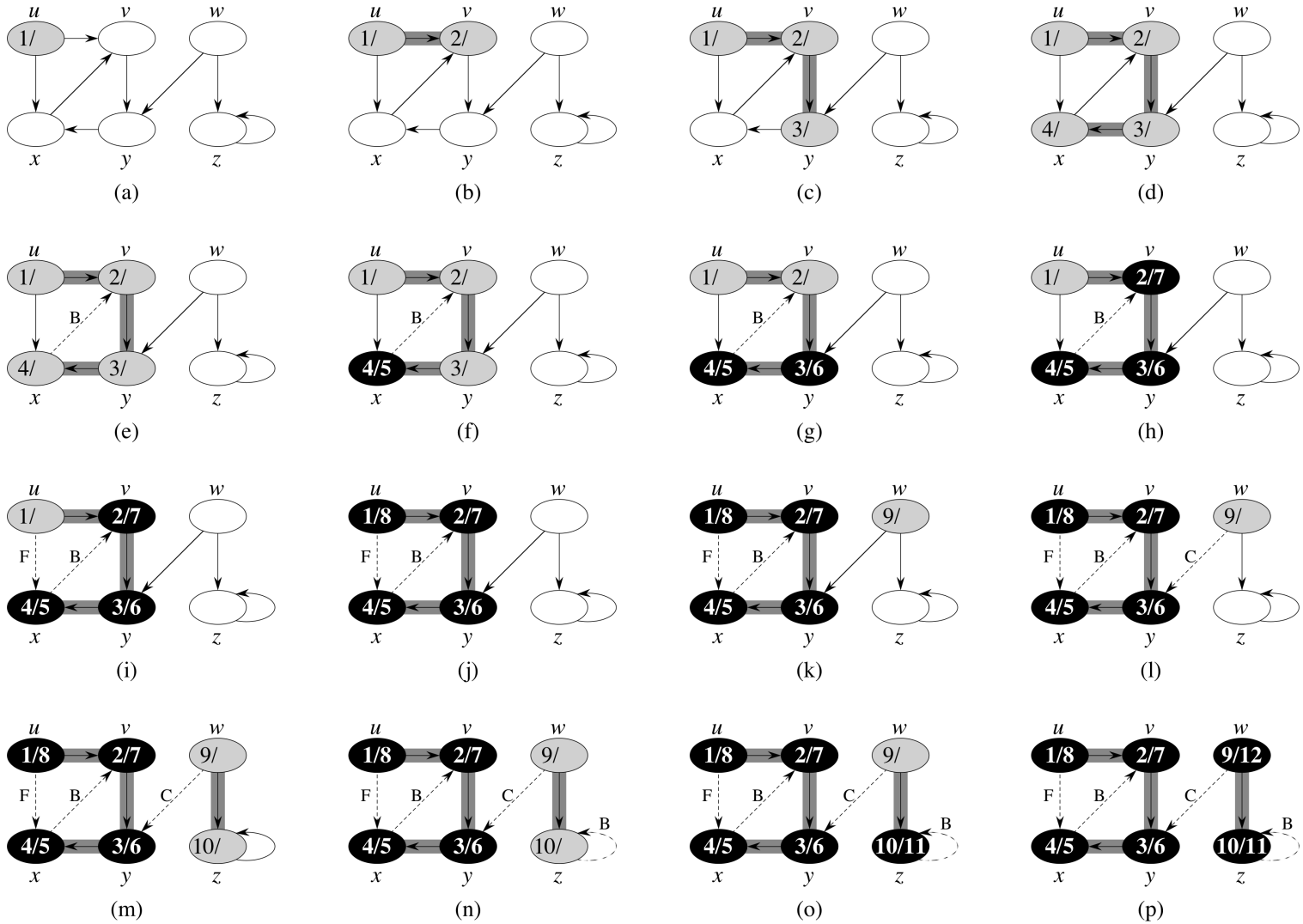
DFS tilføjer også timestamps $u.d$ for "discovery" (hvid $\rightarrow$ grå) og $u.f$ for "finish" (grå $\rightarrow$ sort) til alle knuder u. [$u.d$ er *ikke* "distance" i DFS.]

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ 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 ∈ G.Adj[u]    // explore edge (u, v)
5      if v.color == WHITE
6          v.π = 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: $O(n + m)$.

Beviset er det samme som under GenericGraphTraversalGlobal, da valgene (*) i DFS tager $O(1)$ tid.

Egenskaber

Køretid: $O(n + m)$.

Beviset er det samme som under GenericGraphTraversalGlobal, da valgene (*) i DFS tager $O(1)$ tid.

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: $O(n+m)$.

Beviset er det samme som under GenericGraphTraversalGlobal, da valgene (*) i DFS tager $O(1)$ tid.

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.

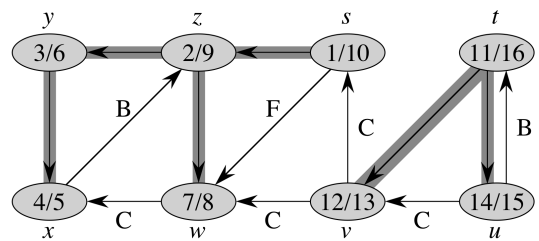
Kanten $(v, v.\pi)$ sættes ved kald af DFS-Visit på v. Af dette, samt ovenstående:

- 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.

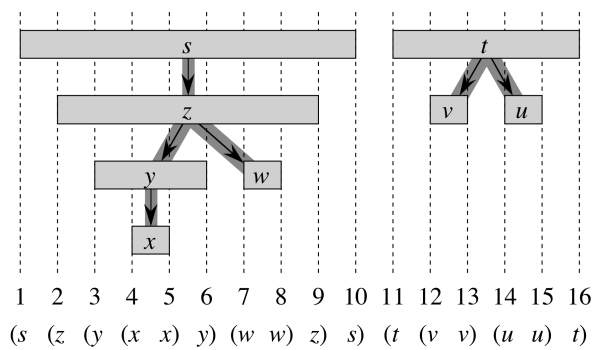
Egenskaber

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).



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

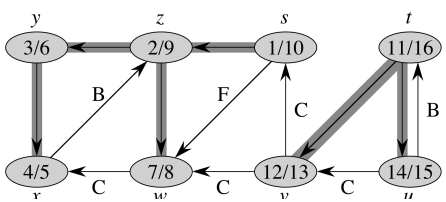


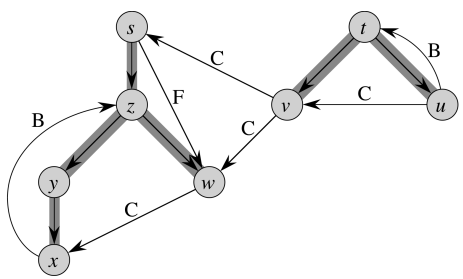
Egenskaber

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

1. *tree-kanter:* v hvid. Her er $u.d < v.d = \text{nu} < v.f < u.f$.
2. *back-kanter:* v er grå (er på stak – det må være under u, som er toppen af stakken (evt. $u = v$ hvis self-loop)). Her er $v.d \leq u.d < \text{nu} < u.f \leq v.f$.
3. *forward-kanter:* v er sort (den er ikke længere på stak, men har været det sammen med u). Her er $u.d < v.d < v.f < \text{nu} < u.f$.
4. *cross-kanter:* v er sort (den er ikke længere på stak, og har ikke været det sammen med u). Her er $v.d < v.f < u.d < \text{nu} < u.f$.

Bemærk at disse cases kan genkendes under DFS via hvid/grå/sort-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

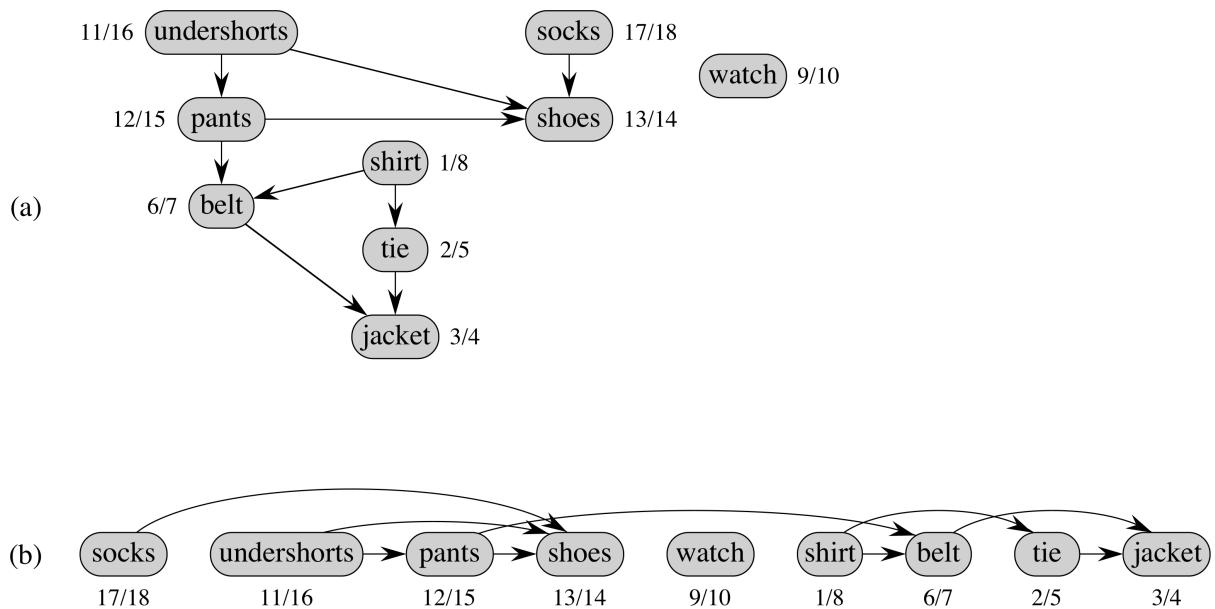
DAG = Directed Acyclic Graph. En orienteret graf uden kredse (cycles).

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 GLOBAL 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 GLOBAL 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 $u.f \leq v.f \Leftrightarrow$ kanten er en back-edge.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f \leq v.f \Leftrightarrow$ 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 slide.

DAGs og topologisk sortering

Lemma: For en kant (u, v) gælder $u.f \leq v.f \Leftrightarrow$ 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 slide.

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 $u.f \leq v.f \Leftrightarrow$ 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 slide.

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.

Så følgende algoritme finder en topologisk sortering i en DAG:

```
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 $u.f \leq v.f \Leftrightarrow$ 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 slide.

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.

Så følgende algoritme finder en topologisk sortering i en DAG:

```
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: $O(n + m)$.