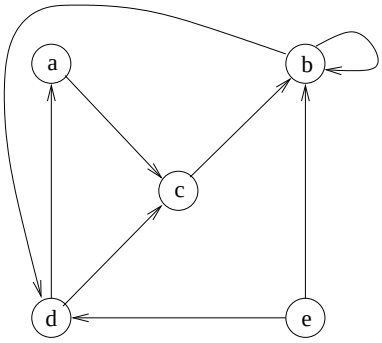
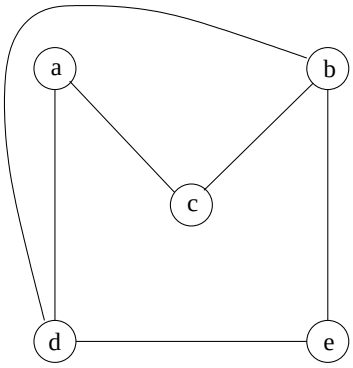
7. Verkot

- *Verkko* (engl. graph) koostuu *solmuista* (engl. node) ja niitä yhdistävistä *kaarista* (engl. edge) joita myös joskus kutsutaan *särmiksi*
- verkkoja on kahta päätyyppiä
  - *suunnatuissa verkoissa* kaarilla on suunta
  - esim:



  - *suuntaamattomien verkkojen* kaarilla ei ole suuntaa
  - esim:



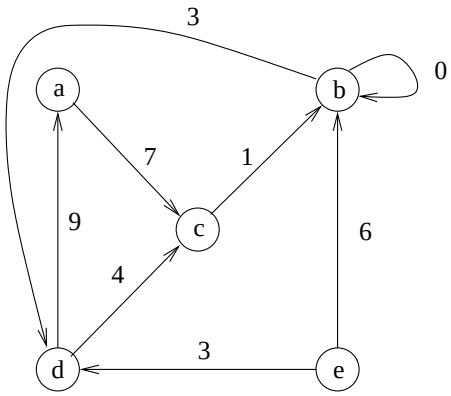
- verkoilla on paljon sovelluksia tietojenkäsittelyssä
- tutustutaan luvussa muutamiin tyypillisimpiin verkkoalgoritmeihin

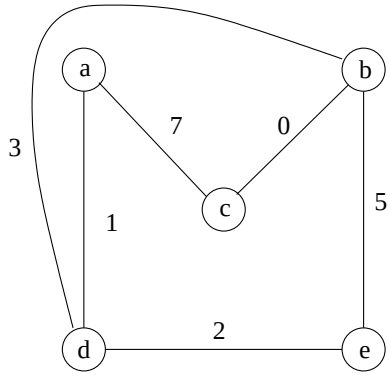
7.1 Käsitteistö

- formaalisti verkko G esitetään parina (V, E), missä
  - V on solmujen joukko
  - E on kaarien joukko
- kaaret ovat siis pareja (u, v) missä u ja v ovat solmuja
- suunnatussa verkossa $(u, v) \in E$ jos solmusta u on kaari solmuun v
  - tällöin u on kaaren *lähtösolmu* ja v kaaren *maalisolmu*
  - solmua v sanotaan solmun u *vierussolmuksi*
  - suunnatun verkon kaarista käytetään usein myös merkintää $u \longrightarrow v$
- edellisen sivun suunnatussa verkossa siis esim. solmun e vierussolmuja ovat b ja d ja solmun a ainoa vierussolmu on c
- esim: edellisen sivun kuvan suunnatun verkon formaali määritelmä:
  - $V = \{a, b, c, d, e\}$
  - $E = \{(a, c), (b, b), (b, d), (c, b), (d, a), (d, c), (e, b), (e, d)\}$
- suuntaamattomassa verkossa kaarten joukko E on *symmetrinen*, eli jos $(u, v) \in E$ niin myös $(v, u) \in E$
  - merkitsemme myös suuntaamattoman verkon kaaria joskus $u \longrightarrow v$
  - jos $(u, v) \in E$ sanotaan että solmut u ja v ovat *vierekkäisiä*, (engl. adjacent) eli v on u:n vierussolmu ja u on v:n vierussolmu

- esim: sivun 192 suuntaamaton verkko formaalisti määriteltynä:
  - $V = \{a, b, c, d, e\}$
  - $E = \{(a, c), (c, a), (b, d), (d, b), (c, b), (b, c), (d, a), (a, d), (e, b), (b, e), (e, d), (d, e)\}$

- usein verkon kaariin liitetään *paino* (engl. weight)





- oletetaan että kaaripainot ovat kokonaislukuja
- kaaripainon käsite määritellä funktiona $w : E \to \{0, 1, 2 \ldots\}$
- eli funktio w liittää jokaiseen kaareen painon, esim. kuvan suunnatussa verkossa $w(a, c) = 7$, $w(e, d) = 3$ jne.
- painotetun verkon kaarista $(u, v) \in E$ käytetään myös merkintää $u \overset{w(u,v)}{\to} v$, eli esimerkissämme on kaari $a \overset{7}{\to} c$

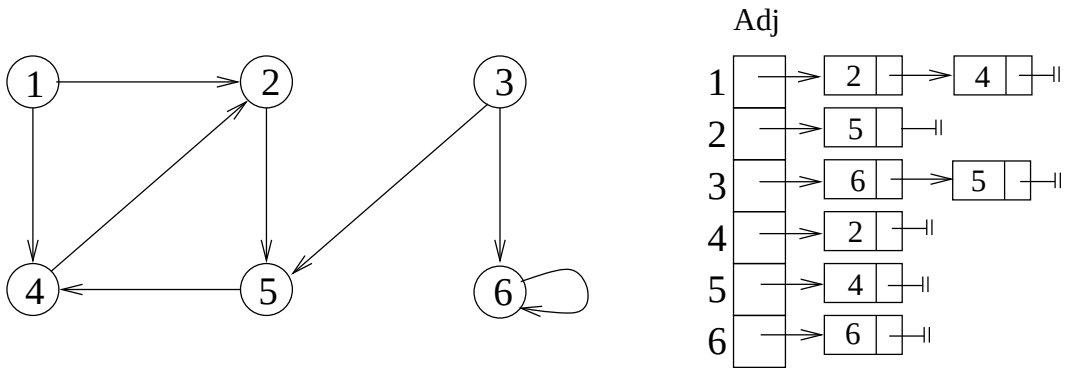
- solmujono $v_1, v_2, \ldots, v_n$ on *polku* solmusta v_1 solmuun v_n jos $v_1 \longrightarrow v_2, v_2 \longrightarrow v_3, \ldots, v_{n-1} \longrightarrow v_n$
- jos solmusta u on polku solmuun v käytetään joskus merkintää $u \rightsquigarrow v$
- jos $u \rightsquigarrow v$ sanotaan että solmu v on *saavutettavissa* solmusta u
- *polun pituus* on polkuun liittyvien kaarien lukumäärä
- painotetussa verkossa *polun paino* on polun kaarien yhteenlaskettu paino
- polku on *yksinkertainen*, jos kukin solmu esiintyy polussa vain kerran, paitsi viimeinen ja ensimmäinen saavat olla sama solmu
- yksinkertainen polku on sykli jos viimeinen ja ensimmäinen solmu ovat samat
- edellisen sivun suunnatun painotetun verkon polkuja:
  - $e \stackrel{3}{\longrightarrow} d \stackrel{4}{\longrightarrow} c \stackrel{1}{\longrightarrow} b$ on yksinkertainen syklitön polku jonka pituus on 3 ja paino 8
  - $d \stackrel{9}{\longrightarrow} a \stackrel{7}{\longrightarrow} c \stackrel{1}{\longrightarrow} b \stackrel{3}{\longrightarrow} d$ on sykli jonka pituus on 4 ja paino 20
  - $c \stackrel{1}{\longrightarrow} b \stackrel{0}{\longrightarrow} b \stackrel{0}{\longrightarrow} b \stackrel{3}{\longrightarrow} d$ on polku jonka pituus 4, paino 4 ja joka *sisältää* kaksi sykliä
- suunnattu verkko on *syklitön* jos se ei sisällä yhtään sykliä
- huom: englannin kielessä *syklittömästä suunnatusta verkosta* käytetään joskus substantiivia *dag* (directed asyclic graph)

7.2 Verkkojen tallettaminen

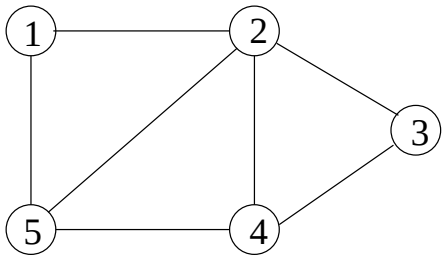
- Tarkastellaan seuraavassa tapoja verkon $G = (V, E)$ tallettamiselle tietokoneen muistiin
- merkitään solmujen lukumäärää symbolilla $|V|$ ja kaarien lukumäärää symbolilla $|E|$
- vaihtoehtoisia talletustapoja on kaksi:
  - *vieruslistat* (engl. adjacency lists)
  - *vierusmatriisit* (engl. adjacency matrices)

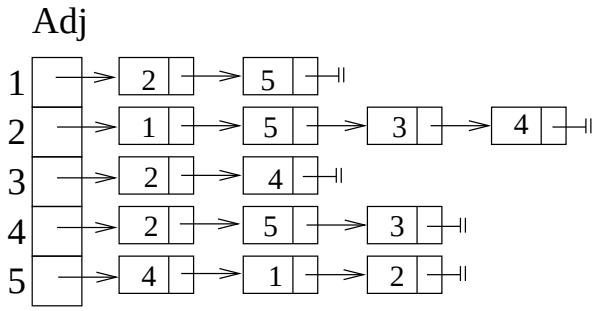
7.2.1 Vieruslistat

- *vieruslistaesityksessä* verkko $G = (V, E)$ esitetään taulukkona Adj joka sisältää $|V|$ kappaletta linkitettyjä listoja, yhden kullekin verkon solmulle
- jokaiselle solmulle $u \in V$ lista $Adj[u]$ sisältää kaikki ne solmut joihin u:sta on kaari
- esim: suunnattu verkko ja sen vieruslistaesitys

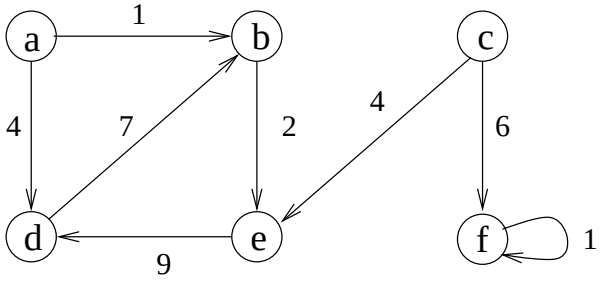


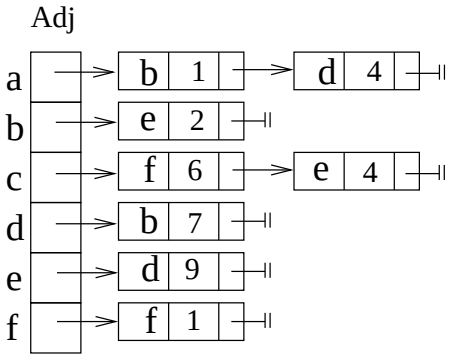
- suunnatun verkon vieruslistojen yhteenlaskettu pituus on $|E|$ sillä jokainen kaari on talletettu kertaalleen yhteen vieruslistoista
- koko vieruslistaesitys vie suunnattujen verkkojen tapauksessa tilaa $\mathcal{O}(|E| + |V|)$, sillä kaarien lisäksi varataan luonnollisesti tila taulukolle Adj
- esim: suuntaamaton verkko ja sen vieruslistaesitys





- suuntaamattoman verkon vieruslistojen yhteenlaskettu pituus on $|E|$ on kaksi kertaa kaarien lukumäärä, sillä jokainen kaari on talletettu kahteen vieruslistaan
- koko vieruslistaesitys vie suuntaamattomien verkkojen tapauksessa tilaa $\mathcal{O}(|V| + 2 * kaarilkm) = \mathcal{O}(|V| + |E|)$
- myös kaarien painot voidaan tallentaa vieruslistarakenteeseen:

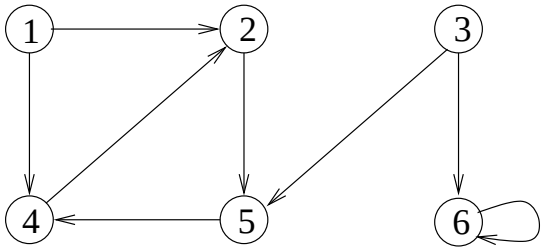




- vieruslistaesityksen hyvä puoli on siis kohtuullinen tilavaativuus joka on $\mathcal{O}(|E| + |V|)$, eli *lineaarinen* suhteessa solmujen ja kaarten määrään
- huonona puolena taas se että tieto onko verkossa kaarta $u \rightarrow v$ ei ole suoraan saatavilla, vaan vaatii vieruslistan $Adj[u]$ läpikäynnin
- pahimmillaan tämä operaatio vie aikaa $\mathcal{O}(|V|)$ sillä solmusta u voi olla pahimmassa tapauksessa kaari kaikkiin verkon solmuihin

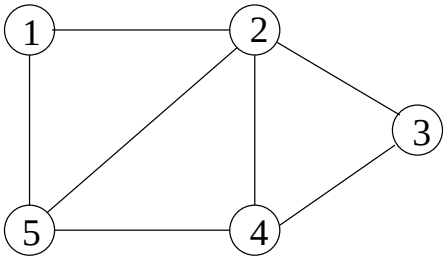
7.2.1 Vierusmatriisit

- Verkon $G = (V, E)$ **vierusmatriisiesityksessä** oletetaan että solmut on numeroitu, eli esim: $V = \{1, 2, \ldots, n\}$
- vierusmatriisi on $n \times n$-matriisi A, missä $A[i, j] = 1$ jos $(i, j) \in E$ ja muuten $A[i, j] = 0$
- esimerkki suunnatusta verkosta:



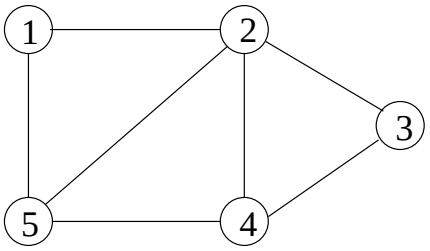
	1	2	3	4	5	6
1	0	**1**	0	**1**	0	0
2	0	0	0	0	**1**	0
3	0	0	0	0	**1**	**1**
4	0	**1**	0	0	0	0
5	0	0	0	**1**	0	0
6	0	0	0	0	0	**1**

- ja suuntaamattomasta verkosta:
- suuntaamattoman verkon tapauksessa jokainen kaari on rekisteröity kahteen kertaan vierusmatriisiin, esim. koska $(2, 5) \in E$, niin $A[2, 5] = 1$ ja $A[5, 2] = 1$



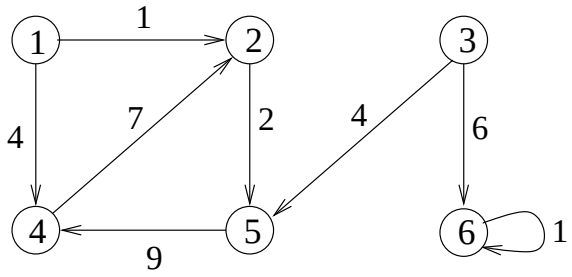
	1	2	3	4	5
1	0	**1**	0	0	**1**
2	**1**	0	**1**	**1**	**1**
3	0	**1**	0	**1**	0
4	0	**1**	**1**	0	**1**
5	**1**	**1**	0	**1**	0

- suuntaamattoman verkon tapauksessa riittäisikin siirtymämatriisista puolikas:



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

- joskus on vieläpä käytössä rajoitus että suuntaamattomassa verkossa kaaret muotoa (i, i) eivät ole sallittuja, jos näin on, ei vierusmatriisin diagonaalia (eli alkioita A[1,1], A[2,2], ...) myöskään tarvita
- kaaripainojen tallettaminen vierusmatriisiin on vaivatonta:



	1	2	3	4	5	6
1	∞	**1**	∞	**4**	∞	∞
2	∞	∞	∞	∞	**2**	∞
3	∞	∞	∞	∞	**4**	**6**
4	∞	**7**	∞	∞	∞	∞
5	∞	∞	∞	**9**	∞	∞
6	∞	∞	∞	∞	∞	**1**

- painotetun verkon vierusmatriisissa periaatteena siis on asettaa $A[i, j] = w(i, j)$ jos $(i, j) \in E$ ja muuten $A[i, j] = \infty/0$

  - jos kaaren $i \xrightarrow{x} j$ paino kuvaa reitin i:stä j:hin pituutta tai kustannusta, on ääretön luonnollinen valinta olemattomien kaarien merkintään
  - jos kaari taas kuvaa reitin kapasiteettia, on luonnollinen valinta olemattomien kaarien merkintään nolla

- hyvänä puolena vierusmatriisissa on se että kaaren olemassaolon näkee matriisista suoraan
- huonona puolena taas tilan tarve, matriisin koko on kaarien lukumäärästä riippumatta aina $|V| \times |V|$
- verkkoa sanotaan *harvaksi* jos kaaria on suhteellisen vähän, esim. vain kaksi kertaa solmujen määrä
- kaarien määrä tällöin $|E| = \mathcal{O}(|V|)$, ja vieruslistana esitetty verkko vie tilaa $\mathcal{O}(|E + V|) = \mathcal{O}(|V|)$ kun taas vierusmatriisiesityksen tilantarve on tähän verrattuna neliöinen $\mathcal{O}(|V| \times |V|)$
- isot harvat verkot siis kannattanee tallentaa vieruslistoja käyttäen
- osa verkkoalgoritmeista tosin olettaa että verkko on talletettu esim. käyttäen vierusmatriiseja, eli verkon talletusmuoto riippuu paljolti verkon käyttötarkoituksesta

7.3 Verkon läpikäynti

- tyypillistä verkkoa käytettäessä on että halutaan kulkea verkossa systemaattisesti vieraillen kaikissa solmuissa tai ainakin kaikissa tietystä solmusta saavutettavissa olevista solmuista
- läpikäyntiin on kaksi perus-strategiaa:
  - *leveyssuuntainen* läpikäynti (engl. breath-first search), ja
  - *syvyyssuuntainen* läpikäynti (engl. depth-first search)

7.3.1 leveyssuuntainen läpikäynti

- Verkon $G = (V, E)$ *leveyssuuntaisessa läpikäynnissä* tutkitaan mitkä verkon solmuista ovat saavutettavissa annetusta aloitussolmusta $s \in V$
- etsintä etenee uusiin solmuihin "taso kerrallaan", eli ensin etsitään mitkä solmut saavutetaan s:stä yhden pituista polkua käyttäen, tämän jälkeen edetään solmuihin mitkä ovat saavutettavissa s:stä kahden mittaista polkua käyttäen, jne
- algoritmin sivutuotteena saadaan tieto mikä on polun pituus aloitussolmusta s kuhunkin läpikäynnin aikana löydettyyn solmuun v, tieto talletetaan solmun attribuuttiin $d[v]$
- toisena sivutuotteena algoritmi muodostaa verkkoa läpikäydessään *leveyssuuntaispuuta* (engl. breath-first tree)
  - puu kertoo mitä reittiä etsintä on edennyt kuhunkin solmuun v
  - algoritmin suorituksen jälkeen puun polku solmusta s solmuun v vastaa verkon lyhintä polkua $s \rightsquigarrow v$

  - puun kaaret talletetaan verkon solmuihin käyttäen attribuuttia $p[v]$

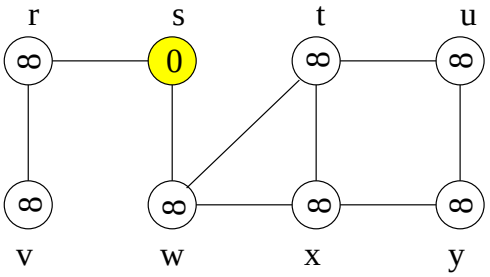
- algoritmin kirjanpitoa varten verkon solmuihin tarvitaan vielä kolmaskin attribuutti, $color[v]$
- aluksi kaikki solmut paitsi aloitussolmu s merkitään *valkoisiksi*
- solmu on valkoinen jos haku ei ole vielä edennyt solmuun asti
- aloitussolmu merkataan *harmaaksi*
- harmaa väri merkkaa että solmu on löydetty mutta etsintä ei ole vielä edennyt kaikkiin sen vierussolmuihin
- kolmas solmujen väri on *musta*
- mustan solmun kaikki vierussolmut ovat joko harmaita tai mustia
- algoritmi käyttää aputietorakenteenaan *jonoa* Q joka on aluksi tyhjä
- syötteenä algoritmille siis verkko $G = (V, E)$ ja jokin verkon solmu $s \in V$, algoritmi olettaa että verkko on esitetty vieruslistamuodossa
- toimintaperiaate
  - aluksi harmaan aloitussolmun s kaikki vierus-solmut laitetaan jonoon
  - vierussolmujen etäisyydeksi päivitetään 1, leveyssuuntaispuussa vanhemmaksi asetetaan s ja värjätään solmut harmaiksi

  - solmu s on nyt käsitelty ja värjätään mustaksi
  - tämän jälkeen niin kauan kun jonossa on solmuja, otetaan käsittelyyn jonon alussa oleva solmu u
  - laitetaan kaikki u:n valkeat vierussolmut (eli ne joita etsintä ei ole vielä kohdannut) jonoon
  - päivitetään jonoon laitettujen etäisyys- ja vanhempikenttää sekä värjätään jonoon laitetut harmaiksi
  - solmu u on nyt käsitelty ja värjätään mustaksi

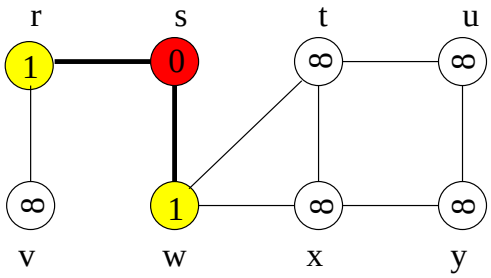
- algoritmi

```
BFS(G,s)
1   for jokaiselle solmulle u ∈ V do
2       color[u] ←white
3       d[u] ←∞
4       p[u] ←NIL
5   color[s] ←gray
6   d[s] ←0
7   enqueue(Q,s)
8   while ( not empty(Q) ) do
9       u ←dequeue(Q)
10      for jokaiselle solmulle v ∈ Adj[u] do
11          if color[v]=white then
12              color[v] ←gray
13              d[v] ←d[u]+1
14              p[v] ←u
15              enqueue(Q,v)
16      color[u] ←black
```

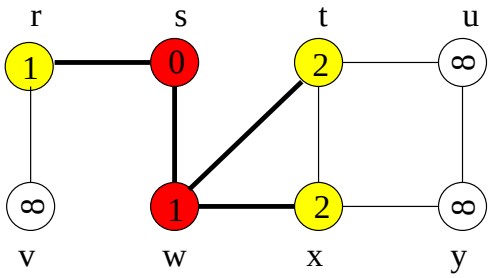
- esimerkki algoritmin toiminnasta seuraavalla sivulla:



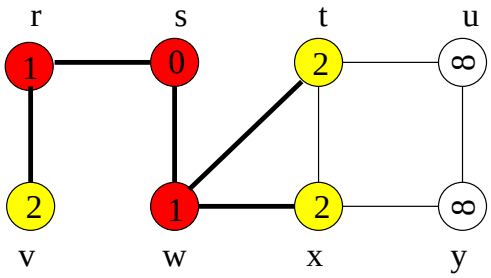
Q | s |



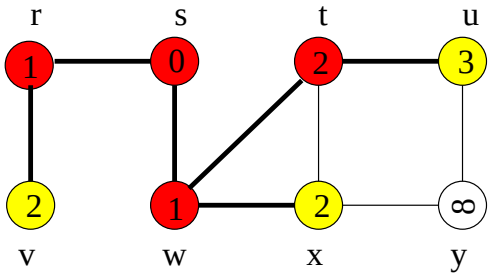
Q | w | r |



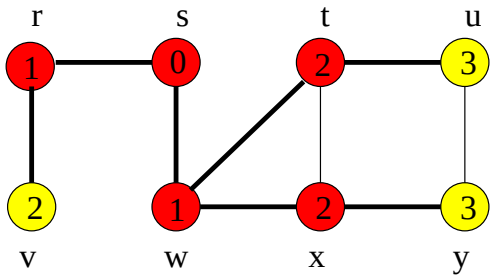
Q | r | t | x |



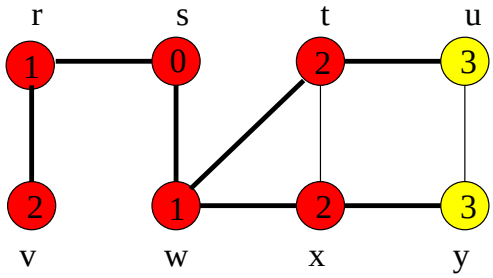
Q | t | x | v |



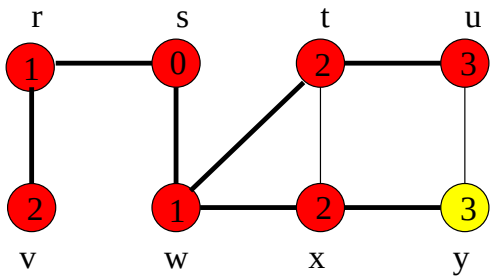
Q | x | v | u |



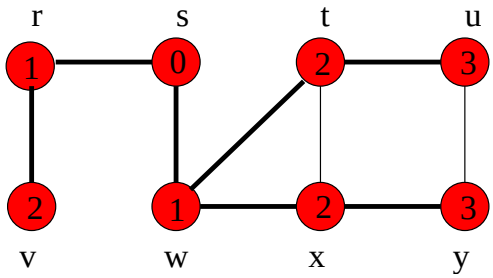
Q | v | u | y |



Q | u | y |



Q | y |



Q

- huomionarvoista algoritmissa on että se toimii sekä suunnatuilla että suuntaamattomilla verkoilla!
- leveyssuuntaisen haun aikavaativuus:
  - alustukseen (rivit 1-6) kuluu aikaa $\mathcal{O}(|V|)$
  - alustuksen jälkeen mitään solmua ei värjätä valkoiseksi

  - koska jonoon laitettava solmu värjätään heti harmaaksi, takaa rivin 11 testi että jokainen solmu laitetaan jonoon vain kerran
  - jokainen solmu siis myös poistetaan jonosta korkeintaan kerran
  - enqueue ja dequeue -operaatiot voidaan toteuttaa ajassa $\mathcal{O}(1)$, eli kokonaisuudessaan jono-operaatioihin kuluu aikaa $\mathcal{O}(|V|)$
  - kunkin solmun vieruslista käydään läpi ainoastaan silloin kuin solmu poistetaan jonosta, eli korkeintaan kerran
  - vieruslistojen yhteispituus on $\mathcal{O}(|E|)$, eli yhteensä vieruslistojen läpikäyntiin käytetään aikaa korkeintaan $\mathcal{O}(|E|)$
- kokonaisuudessaan aikaa siis kuluu $\mathcal{O}(|V| + |V| + |E|)$ eli $\mathcal{O}(|V| + |E|)$
- tilavaativuus algoritmilla on $\mathcal{O}(|V|)$ sillä pahimmassa tapauksessa aloitussolmusta on kaari kaikkiin verkon solmuihin, ja tässä tapauksessa jono Q tulisi sisältämään kaikki verkon solmut

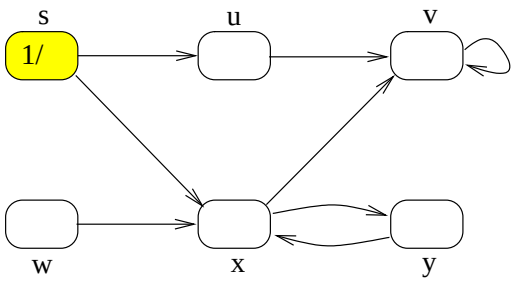
7.3.2 Syvyyssuuntainen läpikäynti

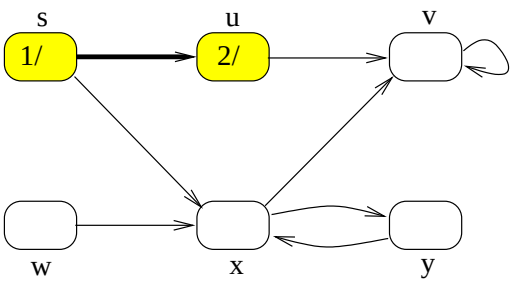
- toinen tyypillisistä läpikäyntitavoista siis on *syvyyssuuntainen* läpikäynti
- strategiana on nyt edetä aloitussolmusta s yhtä polkua niin pitkälle kuin mahdollista
- kun tullaan solmuun josta ei enää päästä uusiin, vielä tutkimattomiin solmuihin, peruutetaan tutkitulla polulla lähimpään sellaiseen solmuun josta lähtee vielä tutkimaton haara
- näin löydetään kaikki solmusta s saavutettavissa olevat solmut
- jos halutaan käydä läpi kaikki verkon solmut ja verkossa on solmuja jotka eivät ole saavutettavissa solmusta s, valitaan yksi saavuttamattomissa olevista solmuista ja käynnistetään uusi etsintä
- läpikäynnin edetessä solmuja väritetään samaan tapaan kuin leveyssuuntaisessa läpikäynnissä
  - jokainen solmu on aluksi valkea
  - solmu väritetään harmaaksi kun se löydetään
  - solmu väritetään mustaksi kun sen vieruslista on käyty kokonaan läpi
- algoritmi *aikaleimaa* jokaisen solmun
  - leima $d[v]$ kirjaa solmun v löytymishetken (harmaaksi värittäminen), ja
  - leima $f[v]$ kirjaa hetken jolloin solmun vieruslista on tutkittu (mustaksi värittäminen)

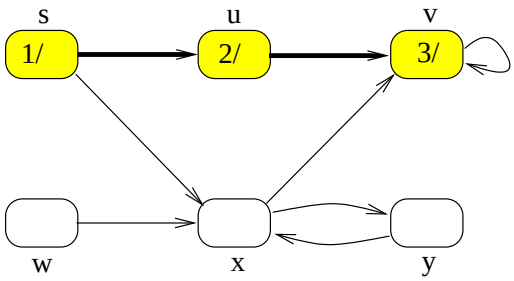
- seuraavassa algoritmi joka joka selvittää aloitussolmusta s saavutettavissa olevat solmut, algoritmi käyttää "aikaleimaamiseen" globaalia muuttujaa *time*

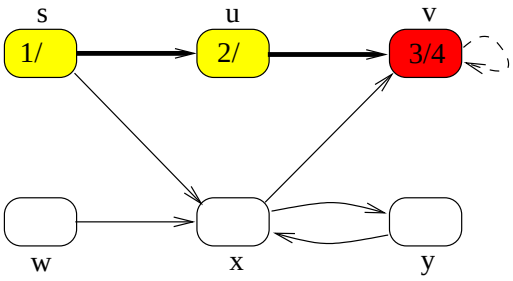
```
DFS(G,s)
1   for jokaiselle solmulle u ∈ V do
2       color[u] ←white
3   time ←0
4   DFS-visit(s)

DFS-visit(G,u)
1   color[u] ←gray
2   time ←time +1
3   d[u] ←time
4   for jokaiselle solmulle v ∈ Adj[u] do
5       if color[v]=white then DFS-visit(v)
6   color[u] ←black
7   time ←time+1
8   f[u] ←time
```

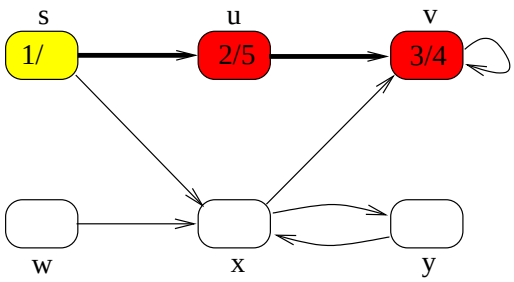
- esimerkki algoritmin toiminnasta seuraavalla sivulla

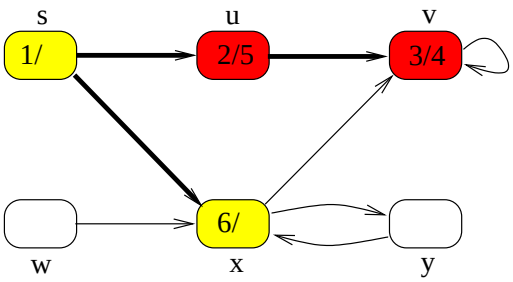

s
u
v
1/
w
x
y

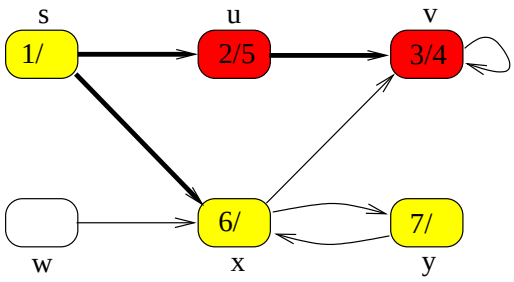

s
u
v
1/
2/
w
x
y

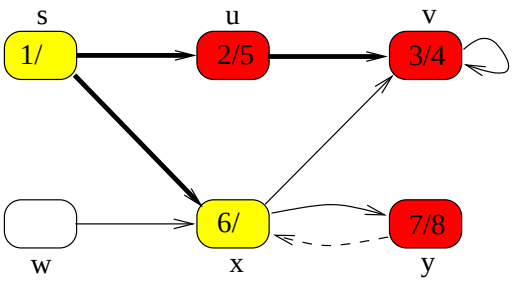

s
u
v
1/
2/
3/
w
x
y

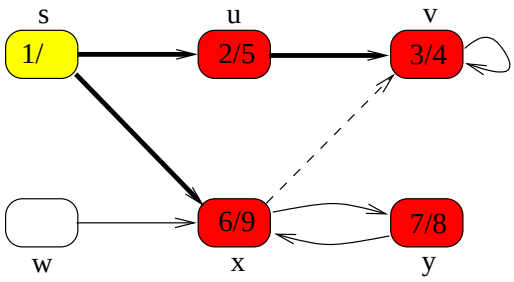

s
u
v
1/
2/
3/4
w
x
y

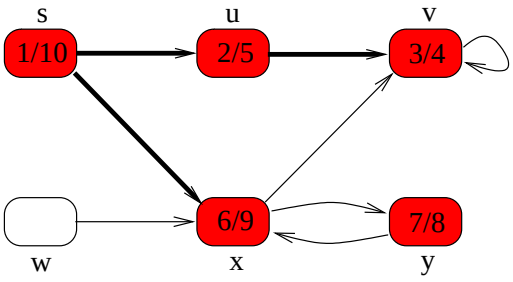

s
u
v
1/
2/5
3/4
w
x
y


s
u
v
1/
2/5
3/4
6/
w
x
y


s
u
v
1/
2/5
3/4
6/
7/
w
x
y


s
u
v
1/
2/5
3/4
6/
7/8
w
x
y


s
u
v
1/
2/5
3/4
6/9
7/8
w
x
y

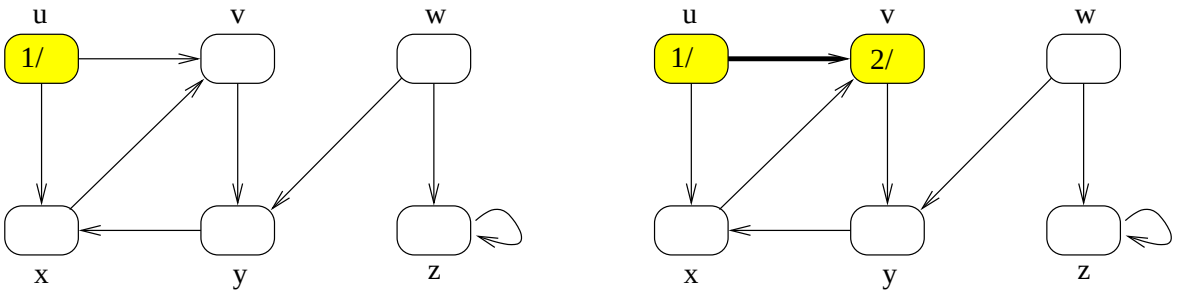

s
u
v
1/10
2/5
3/4
6/9
7/8
w
x
y

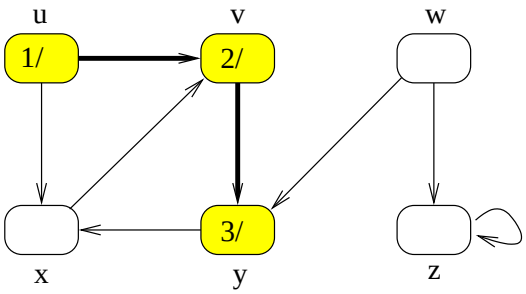
- algoritmi siis värjää mustaksi kaikki aloitussolmusta s saavutettavissa olevat solmut
- kaaret joita pitkin läpikäynti on edennyt on kuvassa paksunnettu
- suorituksen jälkeen solmut sisältävät kaksi lukua, eli löytöajan ja ajan milloin solmun käsittely on valmis
- syvyyssuuntainen läpikäynti siis löysi solmut seuraavassa järjestyksessä: s, u, v, x, y
- solmua w ei saavuteta aloitussolmusta, w jää valkeaksi

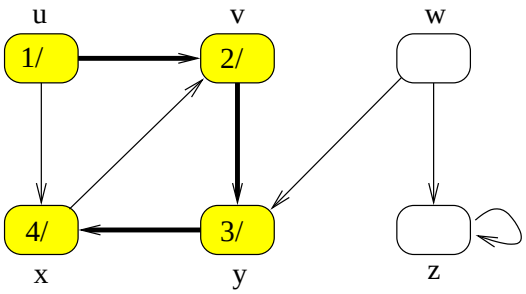
- koko verkolle tehtävä syvyyssuuntainen läpikäynti tapahtuu seuraavasti:

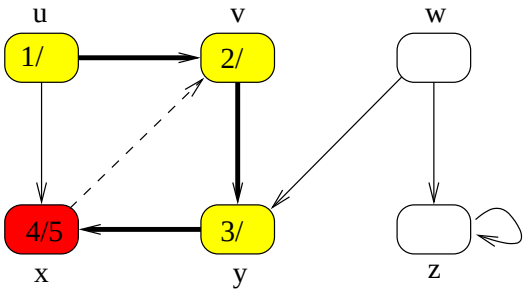
```
DFS-all(G)
1   for jokaiselle solmulle u ∈ V do
2       color[u] ←white
3   time ←0
4   for jokaiselle solmulle u ∈ V do
5       if color[v]=white then DFS-visit(u)
```

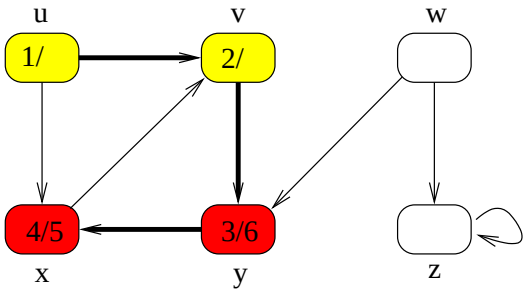
- seuraavassa esimerkki algoritmin toiminnasta:

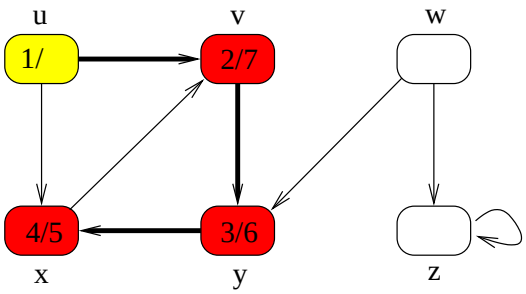


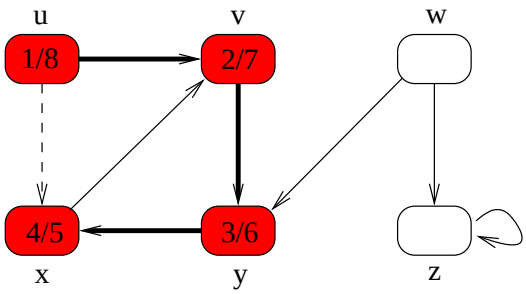

u
v
w
1/
2/
3/
x
y
z

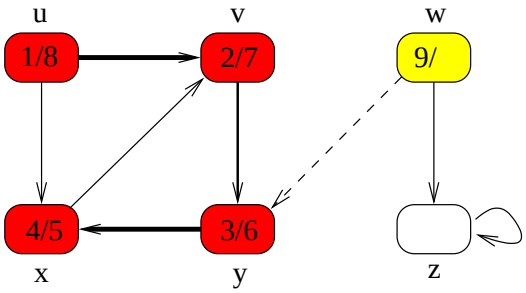

u
v
w
1/
2/
4/
3/
x
y
z

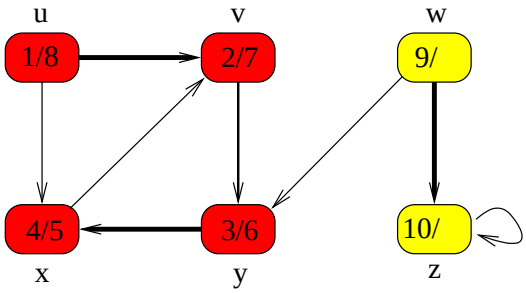

u
v
w
1/
2/
4/5
3/
x
y
z

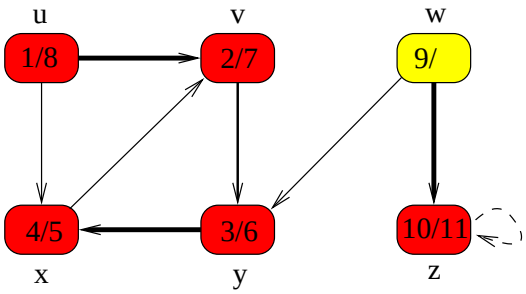

u
v
w
1/
2/
4/5
3/6
x
y
z

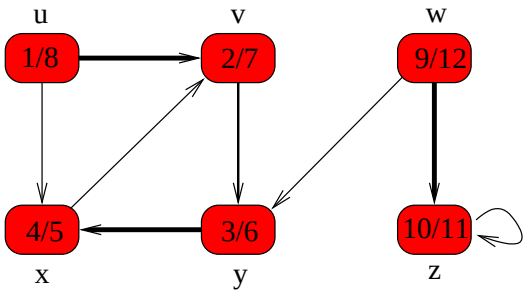

u
v
w
1/
2/7
4/5
3/6
x
y
z


u
v
w
1/8
2/7
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/11
x
y
z


u
v
w
1/8
2/7
9/12
4/5
3/6
10/11
x
y
z

- leveyssuuntaisen läpikäynnin vaativuus
  - alustustoimet, eli solmujen värjääminen valkoiseksi vie aikaa $\mathcal{O}(|V|)$
  - operaatio DFS-visit kutsutaan (korkeintaan) kerran jokaiselle solmulle, sillä operaatiota kutsutaan ainoastaan valkoisille solmuille, jotka heti kutsun jälkeen värjätään harmaaksi
  - yhteensä DFS-visit-operaation kutsuja siis $|V|$ kappaletta
  - DFS-visitin for-lause käy jokaisen solmun vieruslistan läpi, eli for-osa toistetaan yhteensä $|E|$ kertaa
  - kokonaisuudessaan aikaa siis kuluu $\mathcal{O}(|V| + |V| + |E|)$ eli $\mathcal{O}(|V| + |E|)$
  - tilavaativuus algoritmilla on $\mathcal{O}(|V|)$ sillä pahimmassa tapauksessa aloitussolmusta pääsee yhtä polkua kaikkiin muihin solmuihin ja tällöin sisäkkäisiä rekursiivisia DFS-visit -kutsuja tehdään $|V|$ kappaletta
- aivan kuten leveyssuuntaisen läpikäynnin tapauksessa myös syvyyssuuntaisen läpikäynnin algoritmi toimii sellaisenaan niin suunnatuilla kuin suuntaamattomillakin verkoilla

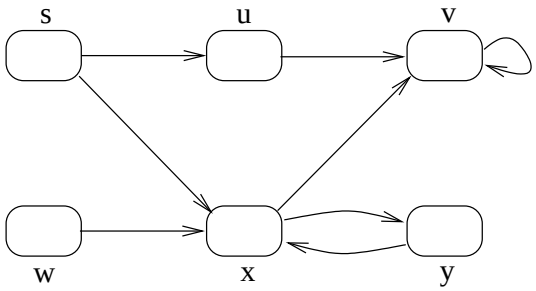
7.3.3 Verkon syklittömyyden tarkastus

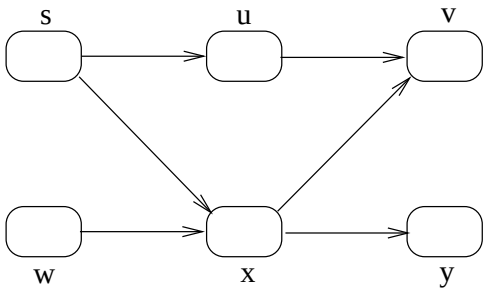
- joskus oi olla tarvetta testata onko annetussa suunnatussa verkossa sykliä
- pienellä muutoksella voimme käyttää syvyyssuuntaisen läpikäynnin algoritmia syklittömyyden testaamiseen:
  - löydetyt solmut joiden vierussolmujen käsittely on vielä kesken ovat väriltään harmaita
  - oletetaan että ollaan tutkimassa solmun u vieruslistaa $Adj[u]$
  - jos vieruslistalta löytyy solmu v joka on harmaa, tiedämme että v:n käsittely on kesken, ja
  - solmusta v johtaa (harmaista solmuista koostuva) polku solmuun u
  - nyt siis on olemassa polku $v \rightsquigarrow u \rightsquigarrow v$, eli verkossa on sykli
- tarvittava muutos on siis testi onko vieruslistalta löytyvä solmu harmaa
- solmun löytymisajan ja käsittelyn päättymisajan tallentavat attribuutit d ja f ovat nyt tarpeettomia

```
DFS-cycles(G)
1  for jokaiselle solmulle u ∈ V do
2      color[u] ←white
3  cycle ←false
4  for jokaiselle solmulle u ∈ V do
5      if color[v]=white then cycle ←cycle or DFS-c-visit(u)
7  return cycle
```

```
DFS-c-visit(u)
1  color[u] ←gray
2  cycle ←false
3  for jokaiselle solmulle v ∈ Adj[u] do
4      if color[v]=gray then return true
5      if color[v]=white then cycle ←cycle or DFS-c-visit(v)
6  color[u] ←black
7  return cycle
```

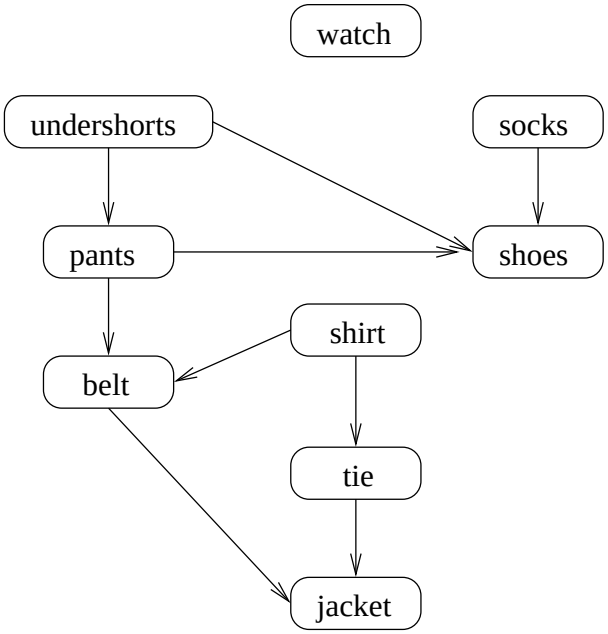
- huom: mustan solmun v löytyminen solmun u vieruslistaa tutkittaessa ei tarkoita syklin olemassaoloa, polkua $v \rightsquigarrow u$ ei voi olla olemassa sillä siinä tapauksessa v ei olisi musta!
- esim: miten syklien etsintä toimisi seuraavissa tapauksissa?





7.3.4 Topologinen järjestäminen

- syklittömien suunnattujen verkkojen avulla voimme kuvata tapahtumien välisiä riippuvuuksia
- esim. oheinen kaavio sisältää onnistuneen pukeutumiseen kannalta oleelliset riippuvuudet:



- eli sukat on laitettava ennen kenkiä, kravatti ja vyö ennen takkia, jne.
- herää kysymys voisimmeko järjestää asiat sellaiseen lineaariseen järjestykseen että voisimme pukea vaatekappaleen kerrallaan siten että mikään riippuvuuksista ei rikkoudu, eli
- jos riippuvuusverkossa on kaari $u \longrightarrow v$, tulee u järjestyksessä ennen v:tä
- tällaista järjestystä kutsutaan *topologiseksi järjestykseksi*
- asia hoituu helposti käyttäen apuna syvyyssuuntaista läpikäyntiä

Topological-Sort(G)
1 kutsutaan DFS(G)
2 kun solmu värjätään mustaksi lisätään se listan L alkuun
3 **return** L

- operaation jälkeen solmut ovat listalla L järjestettynä siten että jos solmu u on ennen solmua v, on $f[u] > f[v]$
  - näin listan viimeiseksi tulee solmu v_n mistä ei ole kaarta mihinkään muuhun solmuun, ja
  - listan toiseksiviimeiseksi solmu v_{n-1} josta on kaari korkeintaan solmuun v_n, jne
- seuraavassa esimerkkimme topologisesti järjestettynä: