

---

**Tietorakenteet ja algoritmit**

**Verkot**

**Ari Korhonen, Lauri Malmi**

---

# **11. VERKOT ( graphs )**

## 11.1 Yleistä

## 11.2 Terminologiaa

## 11.3 Verkon esittäminen

## 11.4 Verkon läpikäyntialgoritmit (graph traversal)

## 11.5 Painotetut verkot (weighted graphs)

## 11.6 Pienin virityspuu (minimum spanning tree)

## 11.7 Lyhin reitti (shortest path)

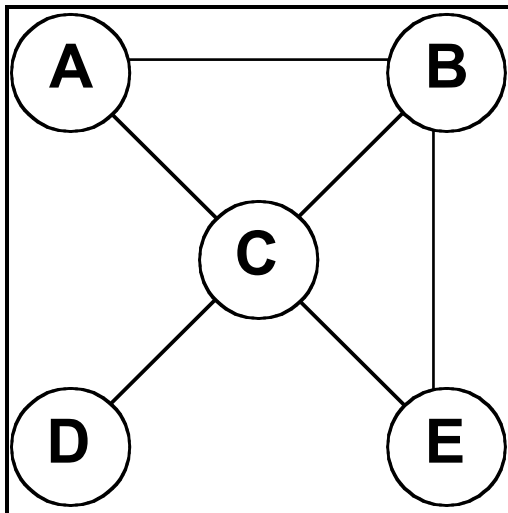
## 11.8 Verkon yhtenäisyys (graph connectivity)

## 11.9 Suunnatut verkot

## 11.10 Joukot ja verkot

## 11.1 Yleistä

- **Verkko on kokoelma *solmuja (vertex)*, joita yhdistää toisiinsa joukko *särmiä (edge)*. Särmiä kutsutaan myös *kaariksi (arc)*.**
- **Solmuilla on nimi ja niihin voi liittyä dataa**



- **Särmät yksilöidään niiden päätesolmujen avulla. Särmiin voi liittyä *paino (weight)***
- **Verkon määrittely ei riipu siitä, miten se on esitetty tai piirretty. Usein maantieteellinen esitys on selkein**
- **Verkkojen sovelluksia:**
  - **Lentoyhtiön aikataulu**
  - **Maantieverkko**
  - **Projektin aikataulu**
  - **Putkistokaavio**
  - **Virtapiiri**
  - **Tietoliikenneverkko**
  - **3D malli kappaleesta**

## **Eräitä verkkoihin liittyviä ongelmia:**

- **Onko reittiä pisteestä A pisteeseen B?**
- **Mikä on lyhin reitti pisteestä A pisteeseen B, kun lasketaan särmät?**
- **Mikä on halvin reitti pisteestä A pisteeseen B, kun lasketaan särmiin liittyvät kustannukset?**
- **Mikä on pienin kustannus, jolla verkon eri solmut voidaan yhdistää toisiinsa?**
- **Voidaanko verkko levittää tasoon?**

## 11.2 Terminologiaa

### Solmujen väliset yhteydet:

- ***Polku (path)*** on solmujen lista, joka johtaa solmusta  $x$  solmuun  $y$ , siten että listan peräkkäisten solmujen välillä on särmä
- ***Yksinkertainen polku (simple path)*** ei kulje saman solmun kautta kahdesti
- ***Silmukka (cycle)*** on yksinkertainen polku, paitsi että sen alku- ja loppusolmut ovat samat

- **Jos verkon särmiä voi edetä vain tiettyyn suuntaan, kyseessä on *suunnattu verkko (directed graph, network)*. Muutoin verkko on *suuntaamaton (undirected)***

**Esimerkki: Projektin aikataulu. Mikä vaihe seuraa mitäkin?**

- **Jos verkon särmiin liittyy jokin kustannus, on kyseessä *painotettu verkko (weighted graph)***

**Esimerkki: Maantieverkko + etäisyydet**

## Verkon yhtenevyys:

- **Suuntaamaton verkko on *yhtenäinen (connected)*, jos jokaisesta solmusta on polku kaikkiin muihin solmuihin**
- **Epäyhtenäinen verkko koostuu yhtenäisistä osaverkoista**

**Esimerkki: Suomen maantieverkko on epäyhtenäinen, koska tietä pitkin ei pääse kaikkiin kaupunkeihin, esim. Maarianhaminaan. Välissä tarvitaan muunlaisia yhteyksiä.**

**Puut:**

- ***Puu (tree)* on yhtenäinen suuntaamaton verkko, jossa ei ole silmukoita**
  - ⇒ **Puussa kahden solmun välillä on vain yksi (yksinkertainen) polku**
  - ⇒ **Jos puuhun lisätään särmä muodostuu silmukka**
- ***Metsä (forest)* on epäyhtenäinen verkko, jonka yhtenäiset osat ovat puita**

- ***Virityspuu (spanning tree)* on verkon yhtenäinen aliverkko, joka sisältää kaikki verkon solmut, mutta vain niin paljon särmiä kuin tarvitaan muodostamaan niistä puu**
- **Virityspuu ei ole yksikäsitteinen**

**Esimerkki: Pienin määrä johtoa, mikä tarvitaan yhdistämään tietty joukko kytkentäpisteitä**

## Verkon särmien määrä:

- **Puu, jossa on  $V$  solmua, sisältää  $V-1$  särmää**
- **Suuntaamattomassa verkossa, jossa on  $V$  solmua, särmien määrä  $E$  voi olla:**

$$0 \leq E \leq V(V-1) / 2$$

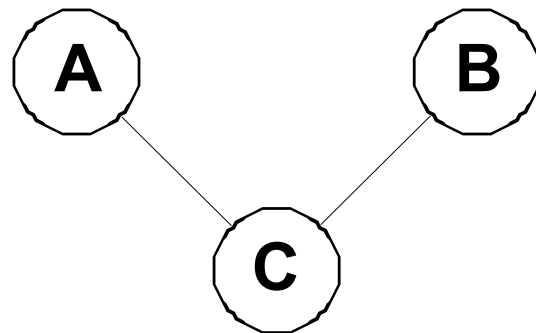
- **Verkko, jossa on kaikki mahdolliset särmät, on *täydellinen (complete)***
- **Verkkoa voidaan sanoa *tiheäksi (dense)* tai *harvaksi (sparse)*. Rajakynnyksenä esimerkiksi  $V \log V$  särmää.**

## 11.3 Verkon esittäminen

Verkko voidaan esittää tietorakenteena usealla eri tavalla:

### 1) *Yhteysmatriisi (adjacency matrix)*

- Oletetaan, että verkon solmut kuvataan kokonaislukuina 1..V
- Esimerkkiverkko voidaan esittää nyt matriisimuodossa, jossa alkio 1 kuvaa särmän olemassaoloa ja alkio 0, että särmää ei ole.

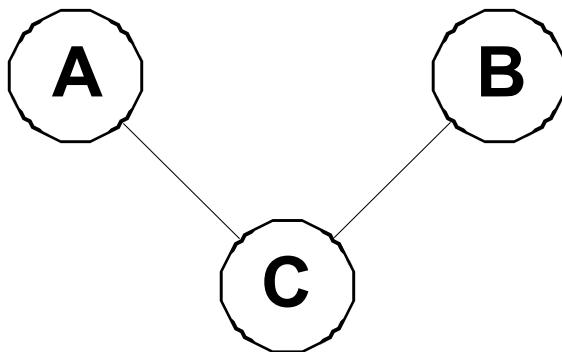


	A	B	C
A	1	0	1
B	0	1	1
C	1	1	1

- **Sovitaan, että alkiosta on yhteys itseensä**
- **Matriisissa puolet alkiosta on tarpeettomia, mutta täydellistä matriisia on helpompi käsitellä algoritmisesti**
- **Yhteysmatriisi soveltuu tiheille verkoille**
- **Suunnatussa verkossa matriisi ei ole symmetrinen**

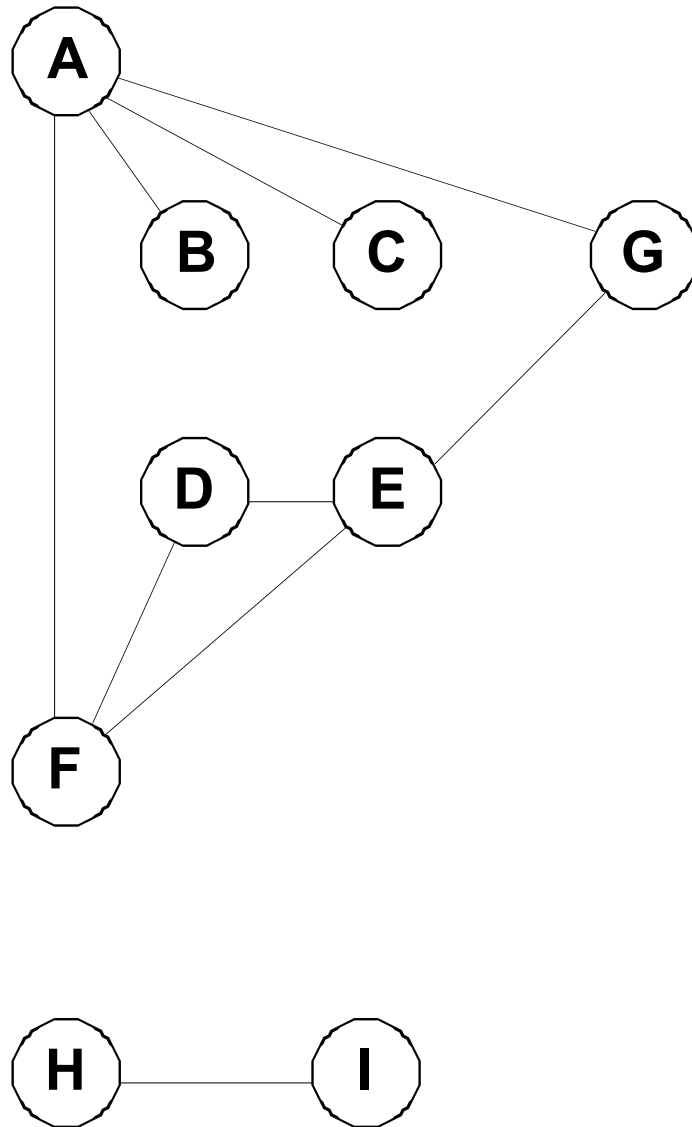
## 2) Seuraajaluettelo (adjacency structure)

- Jokaisen solmun *seuraajalistassa (adjacency list)* luetellaan kaikki ne solmut, joihin on särmä tästä solmusta
- Esimerkkiverkon seuraajaluettelo:



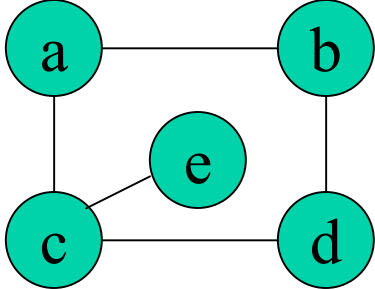
A: C  
B: C  
C: A, B

- Toteutettavissa tehokkaasti linkitettyinä listoina



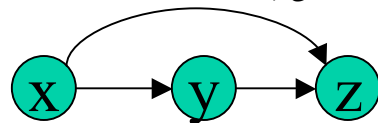
- A: F, C, B, G**
- B: A**
- C: A**
- D: F, E**
- E: G, F, D**
- F: A, E, D**
- G: E, A**
- H: I**
- I: H**

## Luentotehtävä: mikä seuraavista ei kuulu joukkoon? Nimeä esitysmuodot.

<p><b>1</b> Verkko <math>G = (V,E)</math>, jossa</p> <p><math>V = \{ a, b, c, d, e \}</math></p> <p><math>E = \{ (a,b), (a,c), (b,d), (c,d), (c,e) \}</math></p>	<p><b>2</b></p> <table border="1"> <thead> <tr> <th></th> <th>a</th> <th>b</th> <th>c</th> <th>d</th> <th>e</th> </tr> </thead> <tbody> <tr> <th>a</th> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <th>b</th> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <th>c</th> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>d</th> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <th>e</th> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>		a	b	c	d	e	a	0	1	1	0	0	b	0	0	0	1	0	c	0	0	0	1	1	d	0	0	0	0	0	e	0	0	0	0	0
	a	b	c	d	e																																
a	0	1	1	0	0																																
b	0	0	0	1	0																																
c	0	0	0	1	1																																
d	0	0	0	0	0																																
e	0	0	0	0	0																																
<p><b>3</b></p> 	<p><b>4</b></p> <p>a: b, c</p> <p>b: d</p> <p>c: d, e</p> <p>d:</p> <p>e:</p>																																				

## Relaatioista

- **Olkoon verkko  $G = (V, E)$ , jossa**
  - **$V$  on verkon solmujen joukko ja**
  - **$E$  on verkon kaarien joukko  $E \subseteq V \times V$**
- **Binäärirelaatio  $E$  on**
  - **refleksiivinen, jos kaikille  $x \in V$  pätee  $(x, x) \in E$**
  - **symmetrinen, jos  $(x, y) \in E \rightarrow (y, x) \in E$**
  - **transitiivinen, jos  $(x, y) \in E$  ja  $(y, z) \in E \rightarrow (x, z) \in E$**



- **Relaatio, joka on refleksiivinen, symmetrinen sekä transitiivinen on ekvivalenssirelaatio**
  - **vrt ”=” tai ”<” luonnollisille luvuille**

## 11.4 Verkon läpikäyntialgoritmit (graph traversal)

- Algoritmien tavoitteena on käydä läpi verkon kaikki solmut
- Tätä varten verkon ajatellaan koostuvan kolmesta osasta:

### 1) *Puu (tree)*

- Solmut, joissa on jo käyty

### 2) *Reunus (fringe)*

- Solmut, joiden etäisyys puusta on yksi särmä

### 3) *Tuntematon (unseen)*

- Muut solmut, jotka ovat siis kauempana puusta

- **Verkko (sen yhtenäinen osa) käydään läpi valitsemalla reunukselta jokin solmu, liittämällä se puuhun ja siirtämällä sen naapurit reunukseen**
- **Eri algoritmit poikkeavat siinä, mikä solmu reunukselta valitaan**
- **Lopputuloksena saadaan *verkon hakupuu***

### 11.4.1 Depth-first haku (depth-first search, DFS)

- Menetelmässä valitaan aina reunuksen uusin solmu ja liitetään se puuhun
- Reunus toteutetaan pinona:
- Pinon päällimmäinen solmu liitetään puuhun
- Tämän vierekkäiset alkiot työnnetään pinoon, elleivät ne jo ole siellä
- Pinon lisäksi ylläpidetään aputaulukoita *visited[]* ja *parent[]*
  - *visited[i]*:  $i = 0$  (tuntematon),  $i = -1$  (reunuksessa),  
 $i > 0$  (puussa)
  - *parent[i]*: solmun  $i$  isä verkon hakupuussa

### 11.4.2 Rekursiivinen DFS

- **Rekursiivinen toteutus käy verkon kaikki solmut läpi seuraavalla tavalla:**
  1. *Vieraile (visit)* solmussa
  2. Tutki, onko seuraajalistan mukaisissa solmuissa vierailtu
  3. Jos jossakin listan solmussa ei ole käyty, kutsu DFS:ää tälle solmulle
- **Rekursiivisen algoritmin ja pinoalgoritmin DFS-puut ovat erilaisia. Miksi?**

## Depth-first Search

### Algorithm DFS(G)

1. for each  $u \in V[G]$  do
2.      $visited[u] \leftarrow false$
3.      $finished[u] \leftarrow false$
4. for each  $u \in V[G]$  do
5.     if  $visited[u] = false$
6.         DFS-VISIT(G, u)

### Algorithm DFS-VISIT(G, u)

1.  $visited[u] \leftarrow true$
2. for each  $v \in Adj[u]$  do
3.     if  $visited[v] = false$  then
4.         DFS-VISIT(G, v)
5.  $finished[u] \leftarrow true$

## **DFS algoritmin tehokkuus on:**

- **Seuraajalistalle  $O(V + E)$**
- **Yhteismatriisille  $O(V^2)$**

**DFS-algoritmilla voidaan ratkaista eräitä tärkeitä verkkoihin liittyviä ongelmia, mm.**

- **Onko verkko yhtenäinen?**
- **Mitkä ovat verkon yhtenäiset osaverkot?**
- **Onko verkossa silmukoita?**

### 11.4.3 Breadth-first-haku (breath-first search, BFS)

- Menetelmässä valitaan aina reunuksen vanhin solmu ja liitetään se puuhun
- Reunus toteutetaan jonona:
- Jonon ensimmäinen solmu liitetään puuhun
- Tämän vierekkäiset alkiot laitetaan jonon loppuun, elleivät ne jo ole jonossa

Tuloksena saadaan *BFS-puu*

## Breadth-first search

1. Initialize empty queue Q
2. for each  $u \in V[G]$  do
3.      $visited[u] \leftarrow false$
4.      $finished[u] \leftarrow false$
5.  $visited[s] \leftarrow true$
6. ENQUEUE(Q, s)
7. while Q not empty do
8.      $u \leftarrow DEQUEUE(Q)$
9.     for each  $v \in Adj[u]$  do
10.         if  $visited[v] = false$  then
11.              $visited[v] \leftarrow true$
12.             ENQUEUE(Q, v)
13.      $finished[u] \leftarrow true$

## **BFS ratkaisee eräitä verkko-ongelmia**

- **Mikä on lyhin reitti pisteestä A pisteeseen B, jos lasketaan vain kuljettujen särmien määrää?**
- **Mitkä ovat lyhimmat reitit kaikkiin solmuihin lähtien solmusta A, jos lasketaan vain kuljettujen särmien määrää?**

## **DFS:n ja BFS:n perusero:**

- **DFS etenee verkossa niin pitkälle kuin mahdollista ja peruuttaa sitten etsiessään uutta etenemishaaraa**
- **BFS etenee verkossa leveällä rintamalla säilyttäen saman etäisyyden lähtöpisteeseen**

- **Reunuksen esittämiseen käytetty tietorakenne voidaan yleistää prioriteettijonoksi, jollain saadaan erilaisia *PFS-hakuja* (*priority-first search*)**
  - **Mikä on lyhin reitti pisteestä A pisteeseen B painotetussa verkossa?**
  - **Mitkä ovat lyhimmat reitit kaikkiin solmuihin lähtien solmusta A, jos verkko on painotettu?**
  - **Mikä on painotetun verkon pienin virityspuu?**

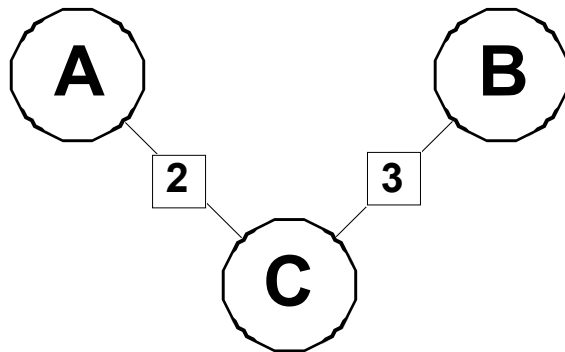
## 11.5 Painotetut verkot (weighted graphs)

- **Painotetussa verkossa jokaiseen särmään liittyy kustannus, esim.**
  - **Aikataulu: hinta tai matkan pituus**
  - **Projektikaavio: osatyön kesto aika**
  - **Sähkökytkentä: johdon pituus**
- **Algoritmeissa käytetään usein termiä etäisyys, vaikka kysymys on painosta**
- **Yleensä painot ovat positiivisia. Negatiivisten painojen esiintyminen voi mutkistaa algoritmeja merkittävästi**

## **Yleisimmät ongelmat:**

- 1. Mikä on lyhin reitti pisteestä A pisteeseen B?**
- 2. Mikä on (painotetun) verkon pienin virityspuu?**

- **Painot voidaan esittää yhteismatriisissa hyvin helposti**



	A	B	C
A	0	0	2
B	0	0	3
C	2	3	0

- **Lävistäjäalkiot on selkeintä pitää nollina**
- **Seuraajaluettelossa esitykseen täytyy liittää painotiedot:**

**A: (C, 2)**

**B: (C, 3)**

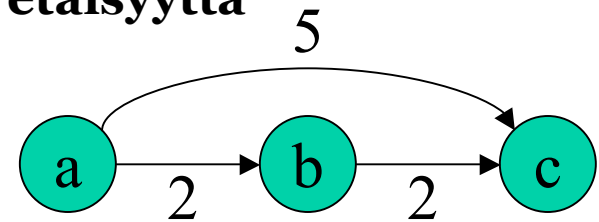
**C: (A, 2), (B, 3)**

## 11.6 Pienin virityspuu (minimum spanning tree)

- **Yhtenäisen verkon pienin virityspuu on niiden särmien joukko, jotka yhdistävät verkon kaikki solmut toisiinsa siten, että kaarien painojen summa on pienin mahdollinen.**
  - ⇒ **Pienin virityspuu ei ole välttämättä yksikäsitteinen**
  - ⇒ **Tämän joukon täytyy olla välttämättä puu, koska muuten siinä olisi silmukka ja sen purkaminen vähentäisi kokonaispainoa.**
- **Puu voidaan löytää useilla erilaisilla algoritmeilla**

### 11.6.1 PFS-haku

- **Verkko kuvataan seuraajalistan avulla**
- **Menetelmässä valitaan aina reunuksen lähin solmu ja liitetään se puuhun**
- **Reunus toteutetaan prioriteettijonona, jossa prioriteettina on solmun etäisyys puusta**
- **Pienimmällä etäisyydellä oleva solmu liitetään puuhun**
- **Tämän vierekkäiset alkioit laitetaan jonoon ja tarvittaessa päivitetään jo jonossa olevien solmujen etäisyyttä**



- **Prioriteettijonon toteutuksena on keko**
- **Tuloksena saadaan pienin virityspuu**
- **Tehokkuus:  $O((E + V) \log V)$**

**Lause:**

**Kun verkon solmut jaetaan kahteen joukkoon, niin pienin virityspuu sisältää lyhimmän särmän, joka yhdistää näiden joukkojen jotkin kaksi solmua.**

**Todistus:**

**Vastaväite: Kyseinen särmä ei kuulu pienimpään vityspuuhun.**

⇒ **Lyhin särmän voidaan lisätä pienimpään vityspuuhun**

⇒ **Vastaava pidempi yhteys voidaan poistaa**

⇒ **Saatiin uusi vityspuu, joka on pienempi**

⇒ **Alkuperäinen ei ollutkaan pienin vityspuu**

⇒ **Päädytään ristiriitaan, joten alkuperäinen lause on tosi**

- **PFS-haussa kyseiset joukot ovat {visited} ja {unvisited}**

### 11.6.2 Primin algoritmi

- **Verkko esitetään yhteysmatriisin avulla, jonka koko on  $V * V$**
- **Prioriteettijonoa ylläpidetään erillisessä taulukossa, jonka koko on  $V$**
- **Jokaisella kierroksella käydään läpi yksi yhteysmatriisin rivi (solmusta  $k$  lähtevät särmät) ja tehdään seuraavat asiat:**
  - **Poistetaan solmu  $k$  prioriteettitaulukosta ja liitetään puuhun**
  - **Lisätään  $k$ :n viereiset solmut taulukkoon ja merkitään niiden prioriteetiksi etäisyys  $k$ :hon**

- **Jos viereinen solmu on jo taulukossa, päivitetään sen prioriteetti-arvo, jos tämä on suurempi kuin solmun etäisyys k:hon**
- **Etsitään prioriteettitaulukon pienin alkio, josta tulee uusi k seuraavalle kierrokselle**

**Tehokkuus:  $O(E) = O(V^2)$  tiheille verkoille**

## 11.7 Lyhin reitti (shortest path)

- Halutaan löytää reitti solmusta A solmuun B siten, että polkuun kuuluvien särmien painojen summa on pienin mahdollinen
- Tuloksena saadaan samalla *lyhimpien reittien virityspuu (shortest path spanning tree)*
- Useita eri algoritmeja

### 11.7.1 BFS

- Jos painot ovat saman suuruisia, voidaan käyttää breadth-first hakua

### **11.7.2 PFS-haku**

- **Yleisessä tapauksessa voidaan käyttää PFS-hakua, jossa verkko kuvataan seuraajalistan avulla**
- **Menetelmässä valitaan aina reunuksen alkupistettä lähin solmu ja liitetään se puuhun**
- **Reunus toteutetaan prioriteettijonona, jossa prioriteettina on solmun etäisyys alkupisteeseen (puun juureen)**
- **Pienimmällä etäisyydellä oleva solmu liitetään puuhun**
- **Tämän vierekkäiset alkiot laitetaan jonoon ja tarvittaessa päivitetään jo jonossa olevien solmujen etäisyyttä**

- **Tämän vierekkäiset alkioit laitetaan jonoon ja tarvittaessa päivitetään jo jonossa olevien solmujen etäisyyttä**
- **Prioriteettijonon toteutuksena on keko**
- **Tuloksena saadaan lyhimpien reittien virityspuu**

**Tehokkuus:  $O((E + V) \log V)$**

### 11.7.3 Dijkstran algoritmi

- **Verkko esitetään yhteysmatriisin avulla, jonka koko on  $V * V$**
- **Prioriteettijonoa ylläpidetään erillisessä taulukossa, jonka koko on  $V$**
- **Jokaisella kierroksella käydään läpi yksi yhteysmatriisin rivi (solmusta  $k$  lähtevät särmät) ja tehdään seuraavat asiat:**
  - **Poistetaan solmu  $k$  prioriteettitaulukosta ja liitetään puuhun**
  - **Lisätään  $k:n$  viereiset solmut taulukkoon ja merkitään niiden prioriteetiksi etäisyys lähtösolmuun ( $k:n$  prioriteetti +  $k:n$  ja vierussolmun välinen paino)**

- **Jos viereinen solmu on jo taulukossa, päivitetään sen prioriteettiarvo, jos tämä on suurempi kuin solmun etäisyys lähtösolmuun k:n kautta**
- **Etsitään prioriteettitaulukon pienin alkio, josta tulee uusi k seuraavalle kierrokselle**

**Tehokkuus:  $O(E) \approx O(V^2)$  tiheille verkoille**

### 11.7.4 Geometriset tehtävät

- **"Hae lyhin reitti tasolla olevassa verkossa pisteestä A pisteeseen B."**
- **Käytetään PFS-hakua, jossa solmun  $x$  prioriteettina käytetään seuraavaa summaa**

**Etäisyys A- $x$  puuta pitkin laskien**

**+**

**geometrinen etäisyys  $x$ -B**

**Lopetetaan, kun B on lisätty puuhun**

**=> Lyhin reitti löytyy nopeasti, koska algoritmi hakeutuu heti oikeaan suuntaan verkossa**

## 11.8 Verkon yhtenäisyys (graph connectivity)

- **Verkko on yhtenäinen, jos jokaisesta solmusta on polku kaikkiin muihin solmuihin**
- **Epäyhtenäinen verkko koostuu yhtenäisistä osaverkoista**
- **DFS-algoritmi löytää verkon yhtenäiset osat**

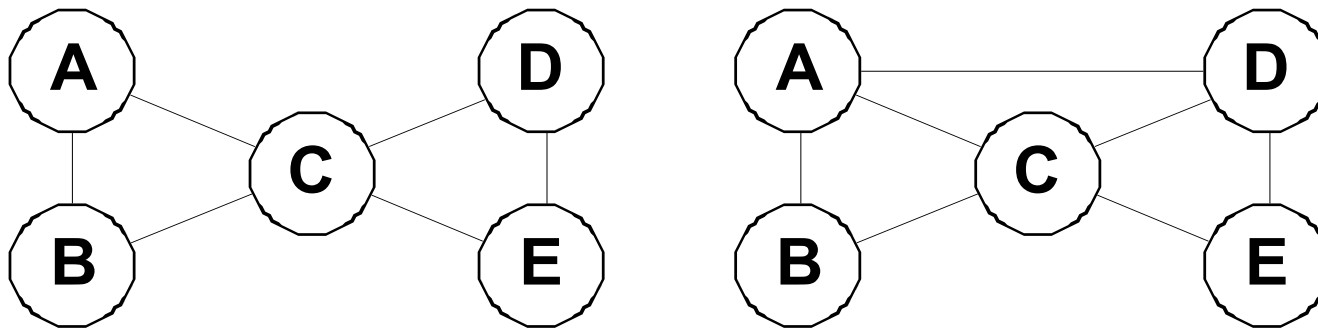
**Jos näet rekursiivisessa DFS-haussa joudutaan tekemään ei-rekursiivinen DFS:n kutsu, ollaan siirrytty toiseen osaverkkoon**

### 11.8.1 Kahdesti yhtenevyys

- **Verkko on *kahdesti yhtenäinen (biconnected)*, jos ja vain jos jokaisen solmuparin välillä on vähintään kaksi erilaista polkua**
- **Kahdella erilaisella polulla ei saa olla muita yhteisiä solmuja kuin alku- ja loppusolmut**
- ***Leikkaussolmu (articulation point)* on solmu, jonka poistaminen aiheuttaa verkon jakautumisen kahteen tai useampaan eri osaan**
  - ⇒ **Kahdesti yhtenäisessä verkossa ei ole leikkaussolmuja**

**Leikkaussolmut jakavat yhtenäisen verkon kahdesti yhteneviin osaverkkoihin**

- **Leikkaussolmu on kriittinen komponentti, jonka vioittuminen voi lamauttaa systeemin toiminnan**
- **Kun tällaisia solmuja ei ole, saadaan vikasietoinen (fault tolerant) systeemi**



- **Leikkauspisteet voidaan etsiä DFS-haun avulla tutkimalla syntyvää DFS-puuta**
- **Solmu  $x$  ei ole leikkauspiste, jos sen *jokaisen* lapsen muodostamasta alipuusta on yhteys hakupuuhun  $k:n$  yläpuolelle johonkin  $k:n$  edeltäjään**
- **Juuren yläpuolelle ei voi olla yhteyksiä. Siksi juuri on leikkauspiste, jos sillä on useampia kuin yksi lapsi**

## **Asian tutkiminen voidaan järjestää seuraavasti:**

- **Jokainen rekursiivinen DFS-kutsu sijoittaa käsittelemänsä (visited) solmun taulukkoon käsittelyjärjestyksessä**
- **Solmun edeltäjät ovat taulukossa ennen solmua itseään**
- **Jokainen rekursiivinen DFS-kutsu palauttaa tiedon ylimmästä (lähinnä taulukon alkua) kutsun aikana viitatusta solmusta**

⇒ **Jos johonkin solmun  $k$  lapseen liittyvä kutsu palauttaa arvon, joka on  $\geq k$ ,  $k$  on leikkauspiste**

- **Algoritmin tehokkuus:**
  - **Seuraajaluettelossa:  $O(V + E)$**
  - **Yhteysmatriisissa:  $O(V^2)$**

## 11.9 Suunnatut verkot

- **Suunnatussa verkossa särkei voi kulkea vain yhteen suuntaan**
- **Kahden solmun välli voi olla kaksi eri suuntiin menevää särkei**
- **Suunnattu verkko voi olla myös painotettu**

### **Esimerkkejä:**

- **Helsingin katuverkko**
- **Projektin ajoituskaavio**

## **Ongelmia:**

### **1) Pääseekö pisteestä A pisteeseen B?**

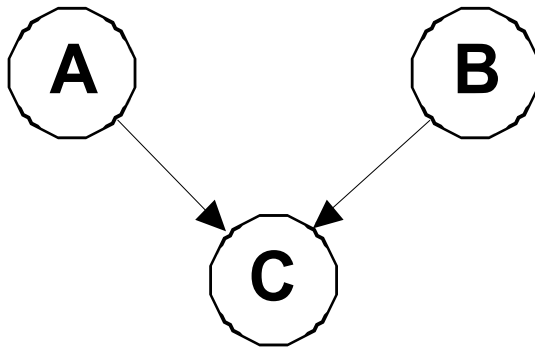
- **Suunnatussa verkossa ei välttämässä pääse kaikista pisteistä toisiin pisteisiin, vaikka verkko olisi yhtenäinen**

### **2) Muodosta suora yhteys kaikkiin niihin solmuihin, joihin pisteestä A voi päästä**

### **3) Etsi verkon topologinen lajittelu**

## Suunnatun verkon esitysmuoto:

- **Yhteismatriisi ei ole symmetrinen**



	A	B	C
A	1	0	1
B	0	1	1
C	0	0	1

- **Seuraajaluettelossa on särmät vain yhteen kertaan**

A: C

B: C

C: -

### 11.9.1 DFS-haku suunnatussa verkossa

- **Perusmenetelmä toimii samalla tavalla kuin suuntaamattomassa graafissa**
- **Algoritmin avulla löydetään kaikki ne solmut, joihin on mahdollista päästä lähtösolmusta käsin**
- **DFS-hakupuun sijasta syntyy yleensä *hakumetsä (DFS search forest)***
- **Alunperin yhtenäisestä verkosta syntyy useita puita**
- **Katkoviivat voivat kulkea paitsi puussa ylöspäin, myös alaspäin ja ristiin eri haarojen välillä**
  - => Algoritmit, joissa tutkitaan DFS-hakupuun rakennetta, mutkistuvat.**

### 11.9.2 Transitiivinen sulkeuma (transitive closure)

- **Jos suunnattuun verkkoon lisätään välit XY, mikäli X:stä on olemassa reitti Y:hyn kaikille solmuille X ja Y, saadaan verkon *transitiivinen sulkeuma***
- **Se ilmaisee suunnatun verkon yhtenäisyyden mahdollisimman kompaktisti. Sen avulla on helppo vastata kysymykseen:**

**"Onko reittiä solmusta A solmuun B?"**

- **Transitiivinen sulkeuma voidaan tuottaa DFS-haun avulla, kun haku suoritetaan lähtien kaikista solmuista erikseen**
- **Tehokkuus:**
  - **Seuraajaluettelolla:  $O(V(E + V))$**
  - **Yhteysmatriisilla:  $O(V^3)$**

## Warshallin algoritmi:

- **Sulkeuma voidaan laskea yksinkertaisella tavalla seuraavan periaatteen pohjalta:**
    - **Solmut esitetään yhteysmatriisissa, jolloin ne numeroidaan välillä 1..V**
    - **Oletetaan, että on olemassa reitti  $X \rightarrow Y$ , joka käyttää vain solmuja välillä 1..Y-1**
    - **Oletetaan lisäksi, että on olemassa yhteys  $Y \rightarrow J$**
- $\Rightarrow$  On olemassa yhteys  $X \rightarrow J$ , joka käyttää vain solmuja välillä 1..Y**

```
for (y=1; y<=V; y++)
  for (x=1; x<=V; x++)
    if (adjmatrix[x,y])
      for (j=1; j<=V; j++)
        if (adjmatrix[y, j])
          adjmatrix[x, j]= TRUE;
```

- **Tehokkuus:  $O(VE + V^2)$**

### 11.9.3 Topologinen lajittelu (topological sorting)

- Joissakin verkkosovelluksissa silmukat eivät ole mielekkäitä.

**Esimerkki:**

- Työnkulkukaaviossa ei ehkä ole mielekästä, että vaihetta A seuraa vaihe B, jota seuraa vaihe C, jota seuraa vaihe A.
- Suunnattu verkko, jossa ei ole silmukoita, on nimeltään *DAG (Directed Acyclic Graph)*
- Sen DFS-puussa ei ole ylöspäin osoittavia (katkoviivalla esitettyjä) nuolia
- Kun DAGin solmut esitetään siinä järjestyksessä, että kaikki solmun edeltäjät ovat listassa ennen ko. solmua, saadaan verkon *topologinen lajittelu*

- **Jos mukaan piirrettäisiin nuolet, ne menisivät kaikki vasemmalta oikealle**
  
- **Hyödyllinen esitysmuoto esim. projektin ajoituskaaviole**
  - **Kullekin osavaiheelle välttämättömät esivaiheet selvästi näkyvillä**

## **Topologinen lajittelu voidaan muodostaa eri tavoin**

- 1. Poistetaan verkosta solmu, josta ei lähde yhtään kaarta**
  - **Samalla poistetaan siihen saapuvat kaaret (matriisin sarake)**
  - **Päivitetään tieto myös poistettujen kaarten lähtösolmujen osalta (laskuri / solmu)**
  - **Valitetaan seuraavaa poistoa varten uusi solmu.**
  - **Lopuksi tulostetaan solmut käänteisessä järjestyksessä**
  
- 2. DFS-haun avulla seuraavasti:**
  - **Käännetään kaikkien särmien järjestys**
  - **Tulostetaan solmun nimi juuri ennen Visit-proseduurin päättymistä**

### 3. Vaihtoehtoisesti voidaan toimia näin:

- Tulostetaan solmun nimi juuri ennen Visit-proseduurin päättymistä, jolloin kaikki seuraajat on jo käsitelty
  - Tuloksena saadaan *käänteinen topologinen lajittelu (reverse topological sorting)*. Se kuvaa tilannetta, jossa solmun nimeä seuraavat ne solmut, joista ko. solmu on riippuvainen
  - Tulostetaan tämä päinvastaisessa järjestyksessä
- 
- Topologinen lajittelu saadaan laskettua ajassa  $O(E + V)$

#### **11.9.4 Vahvasti yhtenäiset komponentit (strongly connected components)**

- **Jos suunnatussa verkossa on joukko solmuja, joista jokaisesta pääsee saman joukon muihin solmuihin, on kyseessä *vahvasti yhtenäinen komponentti***
- **Tällaisten komponenttien ulkopuolelle ei ole edestakaista yhteyttä**
- **Komponentit pystytään löytämään DFS-haulla, jota muokataan sopivasti:**
  - **Tunnistetaan tilanteet, joissa syntyy suunnattu silmukka, eli viitataan hakupuussa ylöspäin jo läpikäytyyn solmuun.**
- **Vahvasti yhtenäisten komponenttien laskemisesta on hyötyä transitiivisen sulkeuman laskemisessa.**

## 11.10 Joukot ja verkot

- **Joukko (set) on kokoelma alkioita**
- **Verkko- ja joukkorakenteita voidaan joissakin ongelmissa käsitellä samalla tavalla**

### **Analogiat:**

**solmu = joukon alkio**

**särmä = alkiot kuuluvat samaan joukkoon**

**verkon yhtenäinen osa = joukko**

## **Eräitä ratkaistavia ongelmia:**

- **Kuuluvatko solmut A ja B verkon samaan yhtenäiseen osaan?**
- **Kuuluvatko alkiot X ja Y samaan joukkoon?**
- **Ratkaisuna on kaksi eri algoritmia**

### 11.10.1 DFS-haku

- **Jos verkko on rakenteeltaan staattinen, käydään verkko läpi DFS-haulla ja merkitään solmutaulukkoon jokainen erillinen osaverkko omalla tunnuksella**
  - ⇒ **Ongelma ratkeaa vertaamalla sitä, ovatko A:n ja B:n osaverkkojen tunnukset samat**
  
- **Jos verkkoon lisätään särmä, kaikki on tehtävä uudestaan**
  - ⇒ **Dynaamiseen tilanteeseen tarvitaan erilainen algoritmi**

### **11.10.2 Etsi-Liitä-algoritmi (Union-find)**

- **Algoritmia käytetään tilanteissa, joissa verkkoon (joukkoon) liitetään peräkkäin uusia särmiiä (yhteenkuuluvuuksia)**
- **Jokaisen lisäyksen jälkeen voidaan heti kysyä, kuuluvatko jotkin solmut (alkiot) samaan komponenttiin (joukkoon)**

**Huom! Algoritmissa muodostuva tietorakenne ei vastaa alkuperäistä verkkoa, vaan on ainoastaan apuväline tässä ongelmassa**

## Perusalgoritmin toiminta:

- **Luodaan aluksi metsä, jossa kaikki solmut ovat erillisinä puina**
- **Kun rakenteeseen lisätään verkkoon kuuluvia särmiä, yhdistetään metsään kuuluvia puita yhteen**
- **Tarvittavat toiminnot *Etsi (find)* ja *Liitä (union)* toteutetaan seuraavasti:**

### Etsi:

- **Kulje molemmista särmän päätesolmuista puun juurta kohden ja katso, löytyykö yhteinen isä**

**Liitä:**

- **Jos särmän päätepisteet kuuluvat samaan puuhun, ei tehdä mitään**
  - **Jos päätepisteet kuuluvat eri puihin, puut yhdistetään siten, että loppupisteen sisältävä puu tulee alkupisteen sisältävän puun juuren alipuuksi**
- 
- **Metsä esitetään dad-linkkien avulla, jolloin juuren etsiminen ja puiden yhdistäminen on helppoa**

**Esimerkki: Verkkoon kuuluu solmut A-M ja siihen liitetään särmiä järjestyksessä AG, AB, AC, LM, JM, JL, JK, ED, FD, HI, FE, AF, GE, GC, GH, JG ja LG.**

- **Perusalgoritmi toimii huonosti, jos lisättävät särmät ovat sopivassa järjestyksessä. Esimerkiksi:**

**BA, CB, DC, ED, ...**

**tai**

**BA, CA, DA, EA, ...**

**⇒ Rakenne degeneroituu listaksi, jolloin tehokkuus on kvadraattinen**

- **Ongelma voidaan kiertää muuttamalla algoritmia hieman:**

### **11.10.3 Tasapainotus painon mukaan (weight balancing, balancing by size)**

- **Liitä-operaatiossa yhteiseksi juureksi tulee se puu, joka on suurempi**
- **Kunkin puun juureen liitetään tieto puun koosta**

### **11.10.4 Tasapainotus korkeuden mukaan (height balancing)**

- **Liitä-operaatiossa yhteiseksi juureksi tulee se puu, joka on korkeampi**
- **Kunkin puun juureen liitetään tieto puun korkeudesta**

### 11.10.5 Polun tiivistäminen (path compressing)

- **Kun juuri on löytynyt, liitetään kaikki lähtösolmusta juureen kulkevan polun solmut suoraan juuren alle. Tämä tehdään sekä särmän alku- että loppusolmulle.**

⇒ **Puu lyhenee, mikä nopeuttaa algoritmia**

- **Menetelmiä voidaan myös yhdistää, jolloin algoritmin tehokkuus on  $O(E^\alpha(E))$ , missä**

$$\alpha(E) < 4, \text{ kun } \log(\log(\log\dots(E)\dots)) \leq 1$$

16 kpl

- **Algoritmin vaatima tila  $\approx V$ , vaikka verkossa voi olla  $E \approx V^2$** 
  - ⇒ **Hyvin suurten verkkojen yhtenevyyttä voidaan helposti tutkia**
- **DFS vaatisi tilaa kaikille särmille**

### 11.10.6 Sovellus: Kruskalin algoritmi

**Verkon pienin virityspuu voidaan laskea seuraavasti:**

- **Ideana on rakentaa asteittain verkon komponentteja suuremmiksi. Uudet lisättävät särmät voivat joko liittää kaksi komponenttia yhteen tai muodostaa silmukan yhdessä komponentissa (jolloin lisäystä ei tehdä).**
  
- **Verkko esitetään seuraajalistan avulla**
- **Verkon särmät lisätään yksi kerrallaan aloittaen lyhimmästä, kunnes kaikki solmut on yhdistetty**
- **Jos särmä muodostaisi silmukan, sitä ei liitetä mukaan**
- **Silmukat voidaan todeta union-find-algoritmillä**
- **Algoritmin oikeellisuus perustuu kohdan 11.6.1 lauseeseen**
  
- **Tehokkuus:  $O(E \log E)$**