

Puurakenteet

Ari Korhonen

6. PUURAKENTEET

6.1 Käsitteistöä

6.2 Binääripuu (binary tree)

6.3 Puiden esitystapoja

6.4 Puussa kulkeminen (traversing a tree)

6.5 Puiden sovelluksia

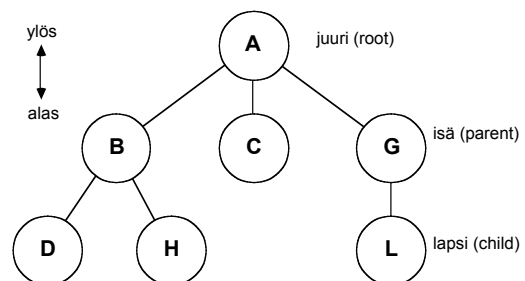
11.1.-11.3 Verkot ja niiden esitystapoja

2/16/05

2

6.1 Käsitteistöä

- Puurakenteen perusmalli on sukupuu



2/16/05

3

Käsitteitä ja ominaisuuksia:

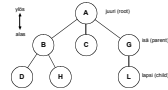
- **Solmu (vertex)** on puuhun kuuluva alkio, johon voidaan tallettaa tietoa
- **Väli, särmä, kaari (edge)** on suora yhteys kahden solmun välillä
- **Juuri (root)** on puun ylin solmu
- **Solmun lähin edeltäjä on sen isä (father, dad, parent).** Kauemmista edeltäjistä käytetään nimeä **isoisä, esi-isä tai edeltäjä (grandfather, ancestor)**
- **Solmun lähinnä alemmat seuraajat ovat sen lapsia (children).** Näiden lapset vastaavasti **lapsenlapsia (grandchildren)**
- **Kullakin solmulla on vain yksi isä**
- **Juurella ei ole isää**



2/16/05

4

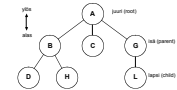
- Solmulla voi olla 0-N kpl lapsia. Yleisessä puussa lasten määrää ei ole rajoitettu
- Saman isän lapset ovat toistensa *sisaruksia (sibling)*
- *Lehti (leaf, external node)* on solmu, jolla ei ole lapsia
- Muut solmut ovat puun *sisäsolmuja (internal node)*
- Lehdet ovat joskus rakenteeltaan erilaisia kuin sisäsolmut
- *Polku (path)* on yhteys kahden solmun välillä
- Juuresta mihin tahansa solmuun on vain yksi polku. Vastaavasti jokaisesta solmusta on vain yksi polku mihin tahansa toiseen solmuun
- Jokainen solmu on oman *alipuunsa (subtree)* juuri
- Puiden joukko on *metsä (forest)*



2/16/05

5

- **Järjestetyssä puussa (ordered tree) lasten järjestys on määrätty**
 - Vasen, Oikea
 - Ensimmäinen, Toinen,...,Viimeinen
 - **EI** tarkoita sitä että solmuihin liittyvä tieto olisi järjestyksessä!
- Solmu on *tasolla N (level N)*, jos polulla siitä juureen on N solmua. Juuren taso on 0.
- Puun *korkeus (height)* on sen tasojen määrä, ts. pisimmän juuresta lehteen ulottuvan polun pituus



2/16/05

6

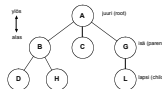
Lause: Jos puussa on N solmua niin siinä on N-1 särmää

Todistus:

- Jokainen solmu kytkeytyy yhdellä särmällä isäsolmuunsa
- Juurisolmulla ei ole isää

Puun rekursiivinen määritelmä:

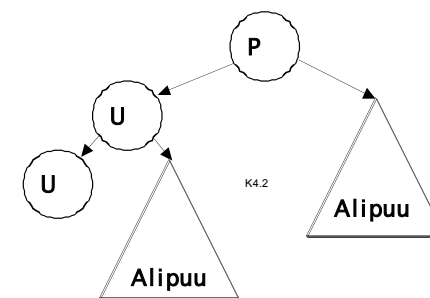
- Puu on joko yksittäinen solmu tai juurisolmu, joka on kytketty joukkoon muita puita (= alipuita)
- Alipuiden tulee olla pareittain pistevieraita



2/16/05

7

- **Usein käytetty merkintä:**
 - Solmut palloja
 - Alipuut kolmioita

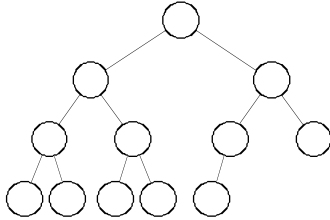


2/16/05

8

6.2 Binääripuu (binary tree)

- **Binääripuu (binary tree)** on järjestetty puu, jossa jokaisella solmulla on täsmälleen kaksi lasta: vasen ja oikea (voivat olla tyhjiä)
- **Binääripuu on täydellinen (complete)**, jos alinta tasoa lukuunottamatta kaikki tasot ovat täynnä ja alimmalla tasolla solmut ovat vasemmassa reunassa



2/16/05

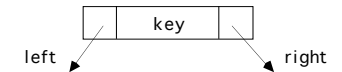
9

6.3 Puiden esitystapoja

6.3.1 Dynaaminen tietorakenne

```
struct node {
    char key;
    struct node *left;
    struct node *right;
};

typedef struct node * link;
```



- **Lehtisolmujen linkit voidaan asettaa arvoon NULL**

2/16/05

10

Sama Javalla

```
class TreeNode
{
    private Object data;
    private TreeNode left, right;

    TreeNode (Object element)
    {
        data = element;
        left = null;
        right = null;
    }
}

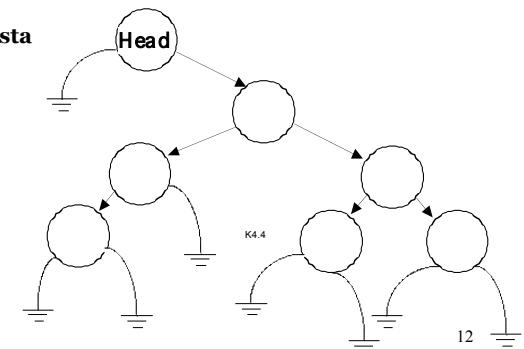
TreeNode root, p, q;
```

- **Lehtisolmujen linkit voidaan asettaa arvoon NULL**

2/16/05

11

- **Joskus juuren käsittely joudutaan suorittamaan erikoistapauksena**
 - Esim. jos kaikkiin solmuihin kohdistettava operaatio viittaa myös solmun isään
- **Kannattaa määritellä ylimääräinen Head-solmu, josta on viittaus juureen**
 - Vrt. linkitetty lista

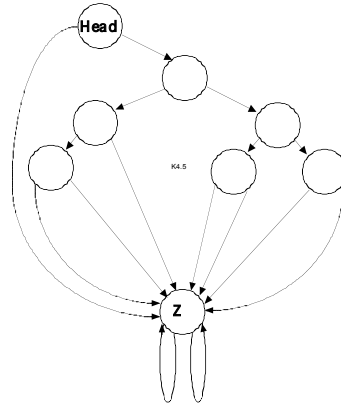


2/16/05

12

- Samoin voidaan määritellä erityinen z-solmu, johon tyhjät alipuut eli lehtisolmujen linkit viittaavat (Sedgewick: Algorithms) :

- z-solmuja on vain yksi kappale
- z-solmun linkit viittaavat z-solmuun itseensä
- Kaikki kerralla käytössä olevat dynaamiset tietorakenteet voivat käyttää samaa z-solmua



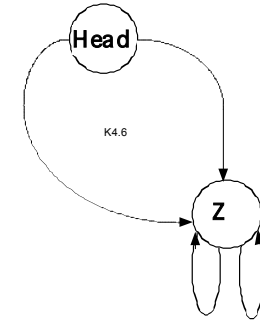
2/16/05

13

- Nyt esim voidaan luoda tyhjä puu ja käsitellä sitä samoilla rutiineilla kuin muitakin puita:

```
link Z = malloc(sizeof(struct node));
Z->left = Z;
Z->right = Z;
```

```
link Head =
    malloc(sizeof(struct node));
Head->left = Z;
Head->right = Z;
```



2/16/05

14

- Tämä esitystapa mahdollistaa liikkumisen puussa vain alaspäin
- Jos halutaan liikkua myös ylös, voidaan lisätä kolmas linkki, dad, joka osoittaa solmun isään
- Yleistä puuta ei kannata esittää samalla tavalla, koska lasten määrä ei ole rajattu

2/16/05

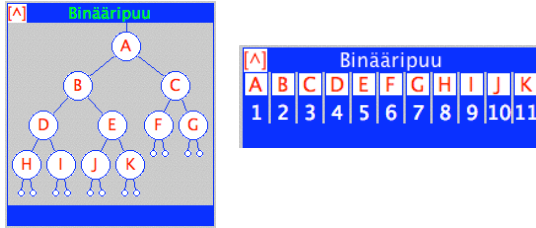
15

Luentotehtävä

- 1. Jos binääripuun korkeus on k, niin kuinka monta alkioita siinä on
 - a) vähintään?
 - b) enintään?
- 2. Jos binääripuussa on N alkioita, niin mikä on sen korkeus
 - a) vähintään?
 - b) enintään?
- 3. Pohdi em. kysymyksiä myös
 - a) täydellisen binääripuun tapauksessa
 - b) yleisen puun tapauksessa.

6.3.2 Taulukkoesitys

- Täydellinen binääripuu voidaan esittää yksinkertaisessa taulukossa
- Juuri on taulukon 1. alkiossa
- Paikassa k olevan solmun isä on paikassa $k \text{ DIV } 2$
- Paikassa k olevan solmun lapset ovat paikoissa $2*k$ ja $2*k+1$
- Viittauksille lapsiin (left, right) ei siis varata lainkaan tilaa
- Esityksessä voi liikkua sekä ylös että alas puussa



2/16/05

17

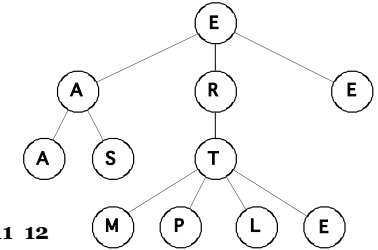
6.3.3 Isä-linkit tai -taulukko yleiselle puulle

- Jos puussa tarvitsee liikkua vain ylöspäin, riittää kussakin solmussa tieto sen isästä. Tämä voidaan toteuttaa joko dynaamisena tietorakenteena tai taulukkona

```
struct node {
    char key;
    struct node *dad;
};

typedef struct node * link;

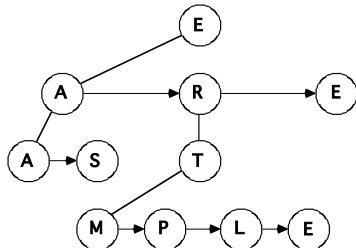
k      1  2  3  4  5  6  7  8  9 10 11 12
a[k]   A  S  A  M  P  L  E  T  R  E  E  X
dad[k]  3  3 10 8  8  8  8  9 10 -1 10 11
```



2/16/05

18

- Jos yleisessä puussa halutaan liikkua alaspäin, kytketään kunkin solmun jälkeläiset linkitetyksi listaksi. Siten voidaan varata tilaa tarpeen mukaan.
- Solmusta on osoitin vasemmanpuolimmaiseen lapseen ja lähinnä oikealla olevaan veljeen



```
struct node {
    char key;
    struct node *First_child;
    struct node *Next_sibling;
};

typedef struct node * link;
```

=> Esitysmuoto palautuu binääripuiksi!

2/16/05

19

6.4 Puussa kulkeminen (traversing a tree)

- Monessa tilanteessa pitää pystyä käymään läpi (*traverse*) kaikki puussa olevat alkiot
- Läpikäyntijärjestyksiä on useita
 - esijärjestys (*preorder*)
 - sisäjärjestys (*inorder*)
 - jälkijärjestys (*postorder*)
 - tasojärjestys (*level order*)

2/16/05

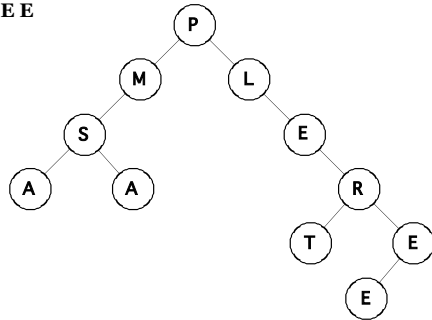
20

- Esijärjestyksessä puu käydään läpi järjestyksessä: juuri - vasen alipuu - oikea alipuu

P M S A A L E R T E E

- Sisäjärjestyksessä puu käydään läpi järjestyksessä: vasen alipuu - juuri - oikea alipuu

A S A M P L E T R E E



2/16/05

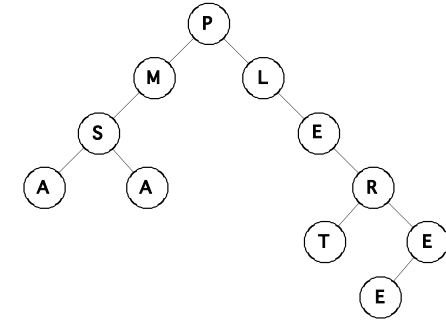
21

- Jälkijärjestyksessä puu käydään läpi järjestyksessä: vasen alipuu - oikea alipuu - juuri

A A S M T E E R E L P

- Tasojärjestyksessä puu käydään läpi kukin taso kerrallaan ylhäältä alkaen

P M L S E A A R T E E



2/16/05

22

“Muistisääntö”:

- Piirretään viiva juuren päältä lähtien vastapäivään puun ympäri läheltä liipaten, tulostetaan solmu kun se
 - ohitetaan vasemmalta => Esijärjestys
 - ohitetaan alta => Sisäjärjestys
 - ohitetaan oikealta => Jälkijärjestys

2/16/05

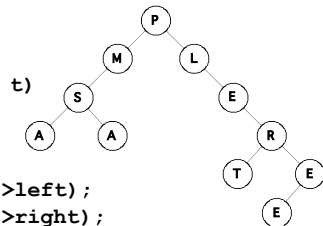
23

- Kolmen ensimmäisen läpikäyntitavan implementaatio on helpointa tehdä rekursion avulla

```

void traverse_preorder(link t)
{
  if (t != NULL) {
    visit(t);
    traverse_preorder(t->left);
    traverse_preorder(t->right);
  }
}

```

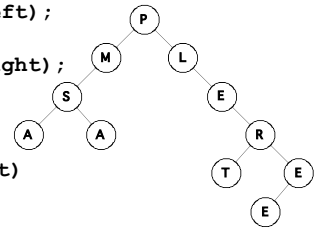


2/16/05

24

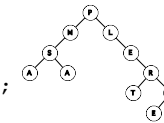
```
void traverse_inorder(link t)
{
  if (t != NULL) {
    traverse_inorder(t->left);
    visit(t);
    traverse_inorder(t->right);
  }
}
```

```
void traverse_postorder(link t)
{
  if (t != NULL) {
    traverse_postorder(t->left);
    traverse_postorder(t->right);
    visit(t);
  }
}
```



Esijärjestys pinon avulla:

```
void
traverse_preorder(link t)
{
  push(t);
  do {
    t = pop();
    if (t != NULL) {
      visit(t);
      push(t->right);
      push(t->left);
    }
  } while (!stack_empty());
}
```



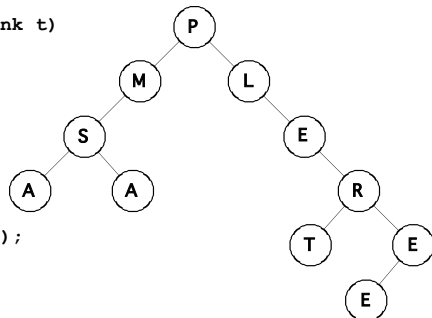
Tasojärjestyksen implementointi käy helposti jonorakenteen avulla:

```
void
traverse_levelorder(link t)
{
  put(t);
  do {
    t = get();
    if (t != NULL) {
      visit(t);
      put(t->left);
      put(t->right);
    }
  } while (!queue_empty());
}
```

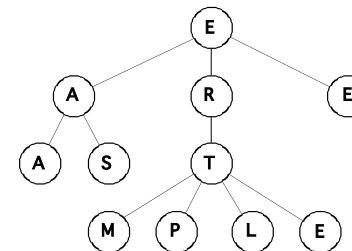
Luentotehtävä

- Esitä jonon sisältö, kun oheinen puu käydään läpi tasojärjestyksessä annetulla algoritmilla

```
void traverse_levelorder(link t)
{
  put(t);
  do {
    t = get();
    if (t != NULL) {
      visit(t);
      put(t->left);
      put(t->right);
    }
  } while (!queue_empty());
}
```



- Yleisen puun läpikäynnissä em. algoritmit yleistetään



Esijärjestys

E A A S R T M P L E E

Jälkijärjestys

A S A M P L E T R E E

- Koska yleinen puu on palautettavissa binääripuiksi, sitä koskevat algoritmit voidaan yleistää binääripuun käsittelyalgoritmeista

6.4.1 Eräitä läpikäynnin sovelluksia:

- **Esijärjestys**
 - sisällysluettelon tulostaminen
 - tiedostoluettelon tulostaminen
- **Sisäjärjestys**
 - hakupuun sisältö aakkostettuna
 - lauseke infix-muodossa
- **Jälkijärjestys**
 - tulosityhteenvedo
 - lauseke postfix-muodossa

6.4.2 Läpikäyntijärjestyksen merkityksiä

- **Esijärjestys vastaa *depth first-hakua***

“etsitään tietty haara mahdollisimman pitkälle ennen kuin peräännyttään”

- **Tasojärjestys vastaa *breadth first -hakua***

“etsitään tietyllä etäisyydellä juuresta olevia solmuja.”

6.5 Puiden sovelluksia**6.5.1 Jäsennyspuu.**

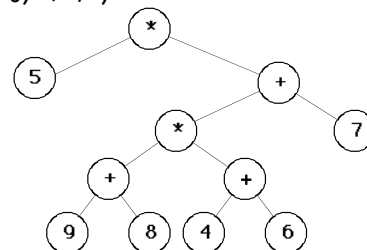
Esimerkiksi lauseke voidaan esittää puurakenteen avulla

- **Alkuperäinen infix-lauseke:**

$$5 * ((9 + 8) * (4 + 6) + 7)$$

- **Kun tämä puu käydään läpi jälkijärjestyksessä, saadaan lauseke postfix-muodossa :**

$$5 9 8 + 4 6 + * 7 + *$$



- **Sisäjärjestys antaa lausekkeen alkuperäisessä infix-muodossa, kunhan sulut lisätään yhteenlaskusolmujen oikeille puolille**

$$5 * ((9 + 8) * (4 + 6) + 7)$$

- **Esijärjestys antaa ns. prefix-muodon**

$$* 5 + * + 9 8 + 4 6 7$$

Jäsennyspuun luominen:

1. Luetaan postfix-lauseketta merkki kerrallaan

5 9 8 + 4 6 + * 7 + *

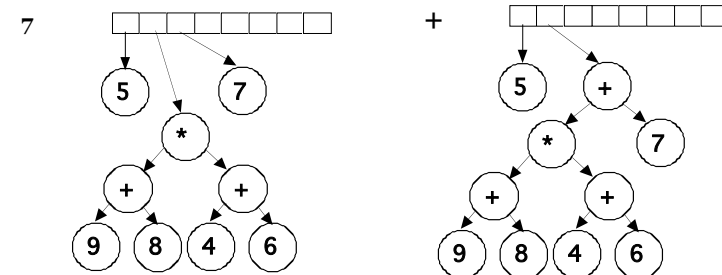
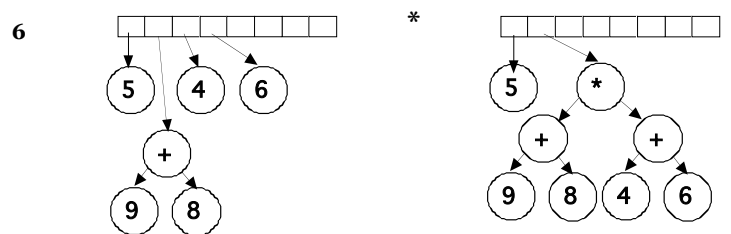
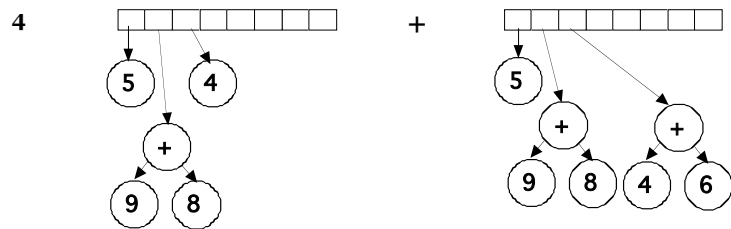
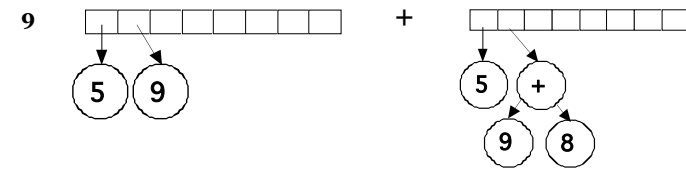
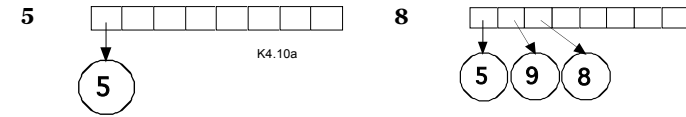
2. Operandi =>

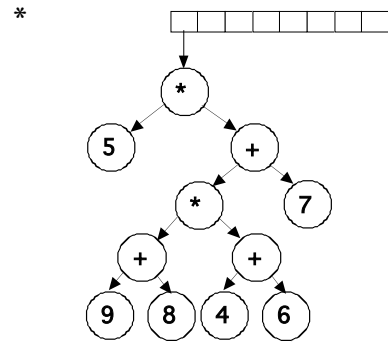
- Tehdään tästä uuden puun juuri
- Osoitin uuteen puuhun asetetaan pinoon

3. Operaattori =>

- Tehdään tästä uuden puun juuri
- Oikeaksi lapseksi popataan pinon päällimmäinen
- Vasemmaksi lapseksi seuraava
- Osoitin uuteen puuhun pinoon

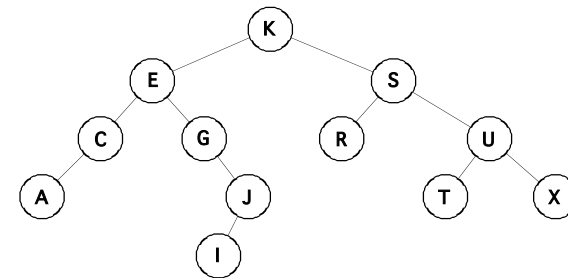
Esim:





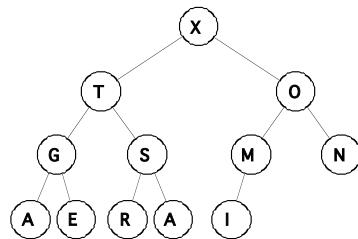
6.5.2 Hakupuu

- Tieto voidaan järjestää puuhun siten, että haku on nopeaa.
Esimerkki: binäärinen hakupuu (binary search tree)



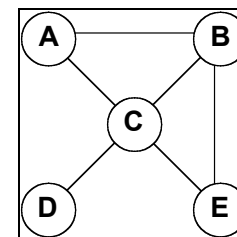
6.5.3 Keko

- Tieto voidaan järjestää puuhun siten, että suurimman (pienimmän) alkion etsiminen on nopeaa.
- Esimerkki: keko (heap)



11.1 Verkot

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



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

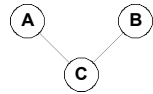
Esimerkki: Projektin aikataulu. Mikä vaihe seuraa mitäkin?

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

Esimerkki: Maantieverkko + etäisyydet

- Särmet E yksilöidään niiden päätepisteiden V mukaan

Esimerkki: $V = \{ A, B, C \}$ $E = \{ (A,C), (B,C) \}$



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 särkeä kuin tarvitaan muodostamaan niistä puu
- Virityspuu ei ole yksikäsitteinen

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

Verkon särkeiden määrä:

- Puu, jossa on V solmua, sisältää $V-1$ särkeä
- Suuntaamattomassa verkossa, jossa on V solmua, särkeiden määrä E voi olla:

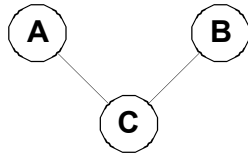
$$0 \leq E \leq V(V-1)/2$$
- Verkko, jossa on kaikki mahdolliset särkeä, on **täydellinen (complete)**
- Verkkoa voidaan sanoa **tiheäksi (dense)** tai **harvaksi (sparse)**. Rajakynnyksenä esimerkiksi $V \log V$ särkeä.

11.3 Verkon esittäminen

Verkko voidaan esittää tietorakenteena usealla eri tavalla:

1) Yhteysmatriisi (*adjacency matrix*)

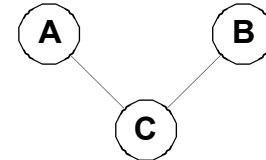
- Oletetaan, että verkon solmut kuvataan kokonaislukuina 1..V
- Esimerkkiverkko voidaan esittää nyt matriisimuodossa, jossa alkio 1 kuvaa särmän olemassaoloa ja alkio 0, että särmää ei ole.



| | A | B | C |
|---|---|---|---|
| A | 1 | 0 | 1 |
| B | 0 | 1 | 1 |
| C | 1 | 1 | 1 |

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

Luentotehtävä: mikä seuraavista ei kuulu joukkoon? Nimeä esitysmuodot.

| <p>1 Verkko $G = (V,E)$, jossa</p> <p>$V = \{ a, b, c, d, e \}$</p> <p>$E = \{ (a,b), (a,c), (b,d), (c,d), (c,e) \}$</p> | <p>2</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>3</p> <pre> graph TD a --- e b --- e c --- e d --- e </pre> | <p>4</p> <p>a: b, c</p> <p>b: d</p> <p>c: d, e</p> <p>d:</p> <p>e:</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

▪ Verkkojen sovelluksia:

- Lentoyhtiön aikataulu
- Maantieverkko
- Projektin aikataulu
- Putkistokaavio
- Virtapiiri
- Tietoliikenneverkko
- 3D malli kappaleesta