

Stabilointi

Marja Hassinen

`marja.hassinen@cs.helsinki.fi`

Sisältö

- ✓ Kertausta ja käsitteitä
- ✓ Stabiilointi
- ✓ Resynkroninen stabiilointi
- ✓ Yleinen stabiilointi
 - ★ Tarkkailu
 - ★ Alustus

Kysymyksiä / kommentteja saa esittää esityksen aikana!

Kertausta

- ✓ Kierros (*round*): Jokainen prosessori suorittaa vähintään yhden askelen.
- ✓ Sykli (*cycle*): Jokainen prosessori suorittaa silmukkansa vähintään kerran.
- ✓ Toipumiskykyisyys (*self-stabilization*): Algoritmi päätyy sallittuun tilaan, vaikka suoritus alkaa mielivaltaisesta alkutilasta.
- ✓ Reilu koostaminen (*fair composition*): Toipumiskykyisen algoritmin käyttäminen toisen toipumiskykyisen algoritmin osana.

Stabilointi

- ✓ Stabilointi: Annetun algoritmin muuntaminen toipumiskykyiseksi.
- ✓ Koska stabilointi on mahdollista, toipumiskykyisten ja ei-toipumiskykyisten algoritmien välillä ei ole periaatteellista eroa.
- ✓ Aikaisemmin esitettyjen muunnosten takia voidaan tarkastella synkronisia tai asynkronisia järjestelmiä, viestinvälitykseen tai jaettuun muistiin perustuvia järjestelmiä jne.

Resynkroninen stabilointi (1)

- ✓ Tarkastellaan synkronista järjestelmää ja algoritmia A , jonka suoritus kestää korkeintaan t askelta.
- ✓ Tallennetaan jokaisen prosessorin P_i yhteyteen taulukko sen tiloista algoritmin A suorituksen aikana.
- ✓ Tallennettavat tilat ovat $T_{i,0}, T_{i,1}, \dots, T_{i,t}$.
- ✓ Prosessorin P_i seuraava tila $T_{i,k+1}$ riippuu nykyisestä tilasta $T_{i,k}$ ja naapuriprosessorien P_j nykyisistä tiloista $T_{j,k}$.

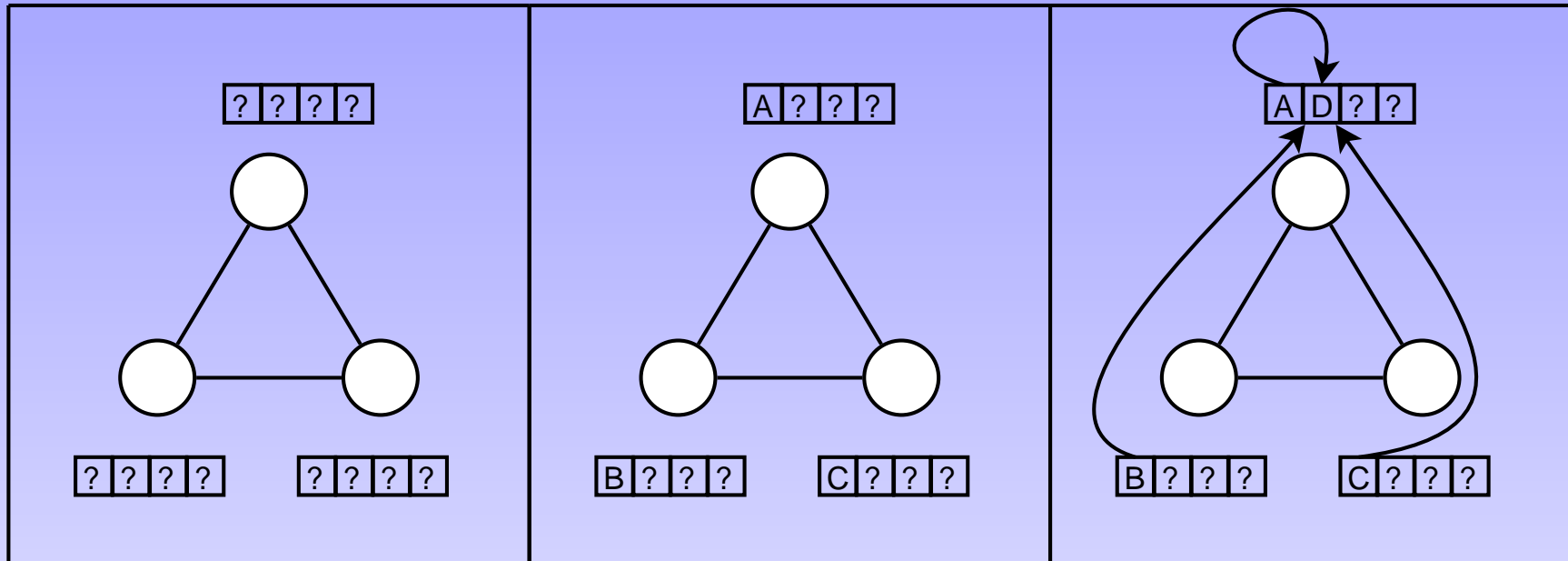
Resynkroninen stabilointi (2)

- ✓ Jokaisen prosessorin P_i algoritmi:
 - ★ Lue naapureiden tilataulukot.
 - ★ Laske **kaikki** omat tilat uudelleen.
 - ★ Siis ensin algoritmin alkutila tallennetaan tilaksi $T_{i,0}$,
 - ★ Sen jälkeen tila $T_{i,1}$ lasketaan tilan $T_{i,0}$ ja naapurien tilojen $T_{j,0}$ avulla jne.

Resynkroninen stabilointi (3)

- ✓ Ensimmäisen virheettömän syklin aikana jokainen prosessori P_i tallentaa ainakin alkutilansa $T_{i,0}$ oikein.
- ✓ Toisen syklin aikana jokainen prosessori P_i laskee tilan $T_{i,1}$ oikein, sillä oma alkutila ja naapurien alkutilat on laskettu oikein.
- ✓ t syklin jälkeen kaikki tilat on laskettu oikein.
- ✓ Vaikka prosessorit laskevat tilansa uudelleen, oikein lasketut tilat eivät muutu.

Esimerkki



Resynkroninen stabilointi (4)

- ✓ Resynkronisen stabiloinnin ongelmia:
 - ★ Vie epäkäytännöllisen paljon muistia.
 - ★ Ei sovellu satunnaisalgoritmeille.
 - ★ Ei sovellu sellaisten algoritmien stabilointiin, joiden tuloste ei ole kiinteä (esim. keskinäinen poissulkeminen: ei erityistä tulostetta, tavoitteena järjestelmän oikeellinen toiminta).
- ✓ Resynkroninen stabilointi osoittaa kuitenkin, että stabilointi on mahdollista eikä toipumiskykyisten ja ei-toipumiskykyisten algoritmien välillä ole periaatteellista eroa.

Yleinen stabilointi

- ✓ Tarkkailu: Havainnoidaan, tapahtuuko virheitä.
- ✓ Alustus: Jos virhe havaitaan, palautetaan järjestelmä sallittuun alkutilaan.
- ✓ Tarkkailun ja alustuksen lomittaminen epätriviaalia: esim. on pidettävä huolta siitä, että alustus suoritetaan loppuun ennen kuin uusia käynnistetään.
- ✓ Seuraavaksi esitetään tarkkailualgoritmi ja alustusalgoritmi.



Tarkkailu: Kuvanottoalgoritmi (1)

- ✓ Kootaan yhdelle prosessorille kuva järjestelmän hetkellisestä tilanteesta.
- ✓ Tilannekuvan perusteella prosessori tarkastaa, onko järjestelmä sallitussa tilassa.
- ✓ Kuvan ottaminen ei keskeytä varsinaista laskentaa vaan suoritetaan sen lomassa.

Suoritus:

- ✓ Valitaan johtaja toipumiskykyisen johtajanvalinta-algoritmin avulla.
- ✓ Tarvittavat muuttujat alustetaan myöhemmin esitettävän alustusalgoritmin avulla.

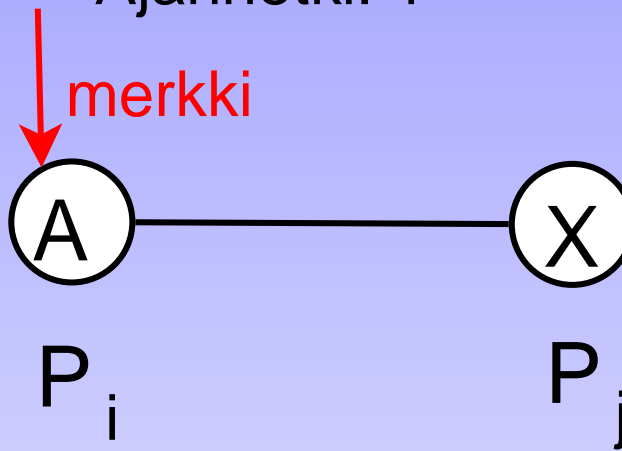
... jatkuu

Tarkkailu: Kuvanottoalgoritmi (2)

- ✓ Johtajan algoritmi:
 - ★ Tallenna oma tila.
 - ★ Lähetä merkkiä kaikille naapureille.
- ✓ Muiden prosessorien algoritmi:
 - ★ Kun saat merkin, tallenna oma tila.
 - ★ Lähetä merkkiä kaikille naapureille P_j .
 - ★ Tallenna naapurilta P_j tulevat viestit kunnes saat siltä merkin.
 - ★ Kun kaikki naapurit ovat lähettäneet merkin, lähetä tiedot johtajalle.

Esimerkki

Ajanhetki: 1

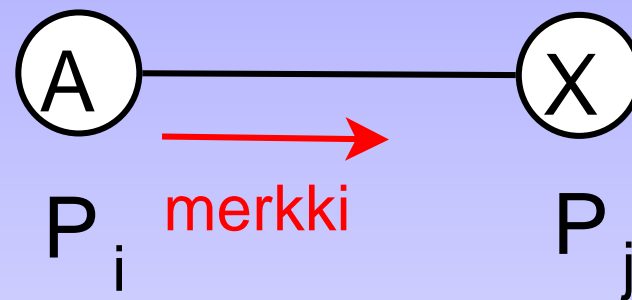


Kuva:

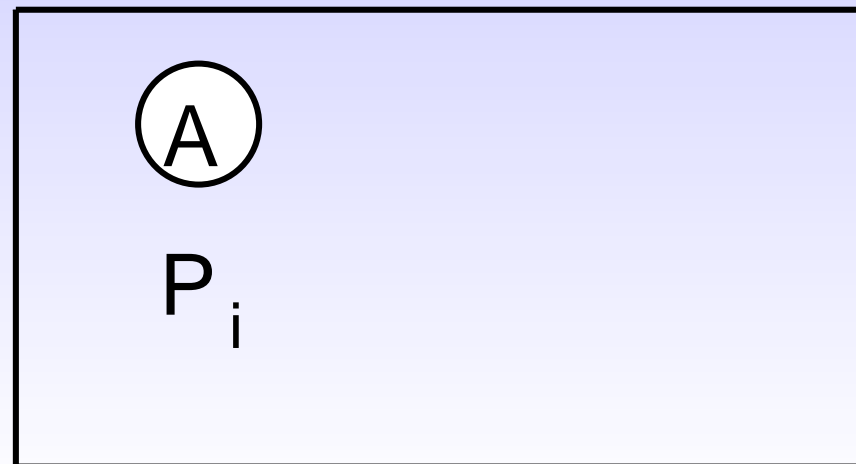


Esimerkki

Ajanhetki: 1

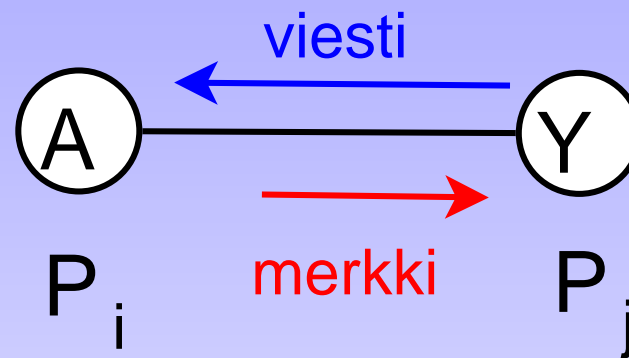


Kuva:

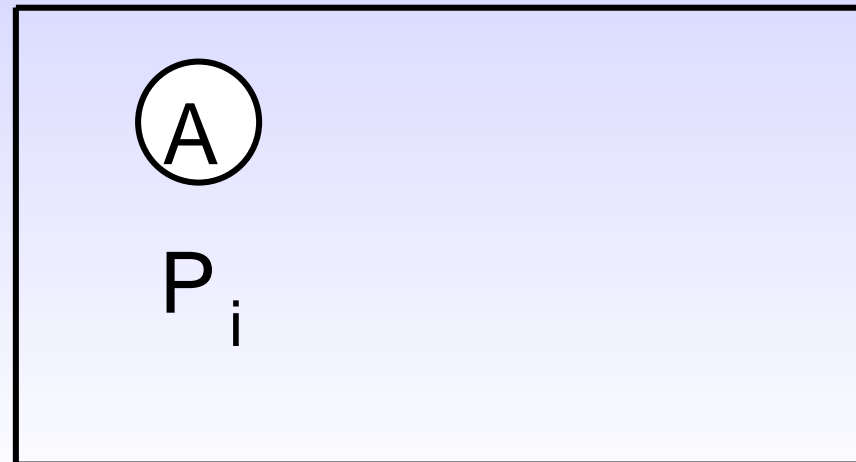


Esimerkki

Ajanhetki: 2

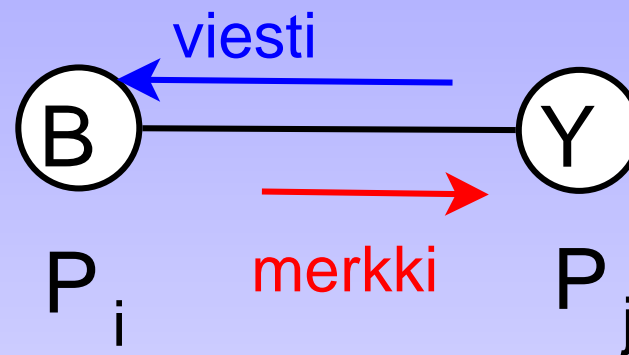


Kuva:

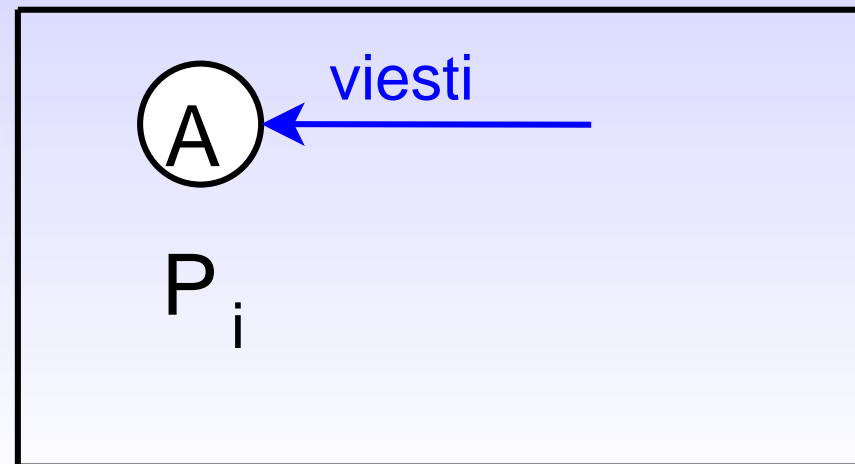


Esimerkki

Ajanhetki: 3



Kuva:

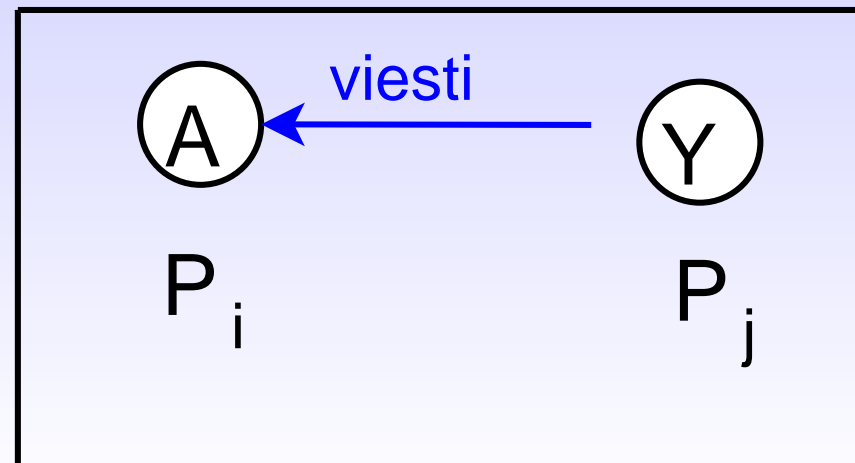


Esimerkki

Ajanhetki: 4



Kuva:



Tarkkailu: Kuvanottoalgoritmi (3)

- ✓ Prosessorien tilat tallennetaan eri ajanhetkinä.
- ✓ Algoritmi johtaa kuitenkin järkevään kokonaiskuvaan, koska myös viestejä tallennetaan.
- ✓ Jos prosessorin P_i tila tallennetaan ennen sen naapurin P_j tilaa, niin P_i tallentaa P_j :n lähettämät viestit.
- ✓ Vaikka prosessorin P_i tila ehtii vaihtua, voidaan kuvitella, ettei prosessori P_i olisi saanut suoritusvuoroa eikä siis vaihtanut tilaansa. Silloin prosessorin P_j lähettämät viestit olisivat viipyneet viestikanaavassa eikä prosessori P_i olisi saanut niitä.

Tarkkailu: Kuvanottoalgoritmi (4)

- ✓ Se, miten virheet havaitaan tilannekuvasta, riippuu stabiloitavasta algoritmista.
- ✓ On esimerkiksi pystyttävä erottamaan virhetilanne sellaisesta tilanteesta, jossa algoritmi toimii oikein mutta tuloksen laskenta on vielä kesken.
- ✓ Joissain tapauksissa voidaan ottaa paikallisia kuvia ja havaita virheet niistä. Silloin ei tarvitse koota koko tilannekuvaa yhdelle prosessorille.

Esimerkki (1)

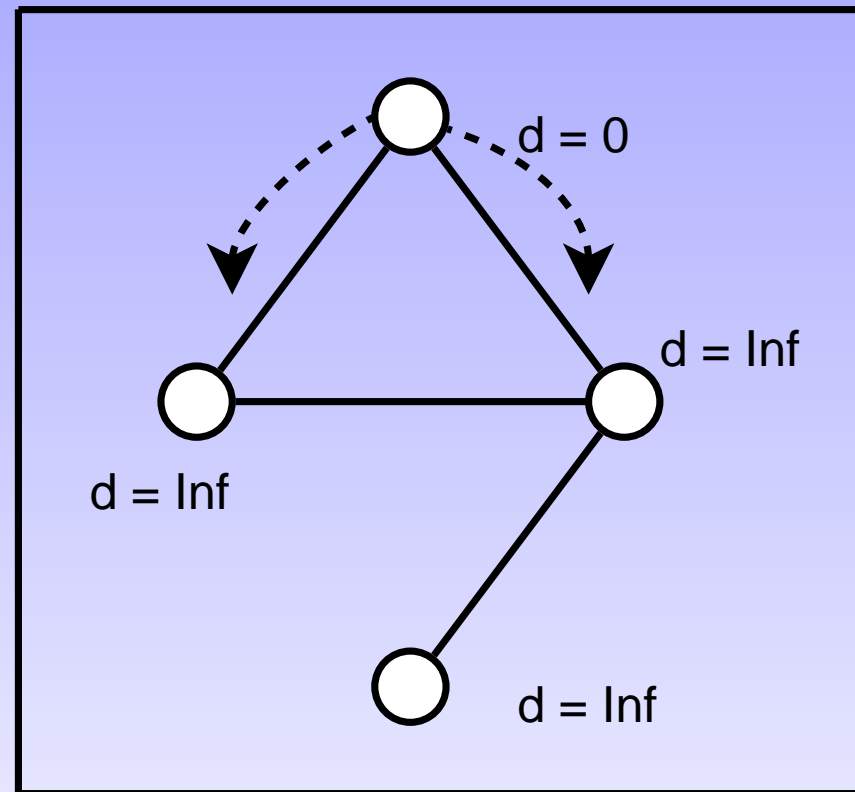
- ✓ Tarkastellaan seuraavaa algoritmia, joka laskee etäisyydet kohdeprosessorista P_0 . Jokaisella prosessorilla P_i on muuttuja d_i , jossa on nykyinen käsitys etäisyydestä.
- ✓ Prosessorin P_0 algoritmi:
 - ★ Aseta $d_0 = 0$ ja lähetä tieto siitä naapureille.
- ✓ Muiden prosessorien P_i algoritmi:
 - ★ Tallenna $d_i = \infty$.
 - ★ Kun saat naapurilta P_j viestinä arvon d_j : Jos $d_j + 1 < d_i$, aseta $d_i = d_j + 1$ ja lähetä d_i viestinä muille naapureille. Muuten älä tee mitään.

Esimerkki (2)

- ✓ Esimerkkialgoritmi ei ole toipumiskykyinen.
- ✓ Esim. jos jonkin prosessorin d -muuttujaan on tallennettu arvo 0 vaikkei prosessori ole kohdeprosessori, niin sitä ei saada korjatuksi.
- ✓ Virhetilanteet voidaan kuitenkin havaita tarkastelemalla tilannekuvaa prosessorien tiloista ja matkalla oleviasta viesteistä.

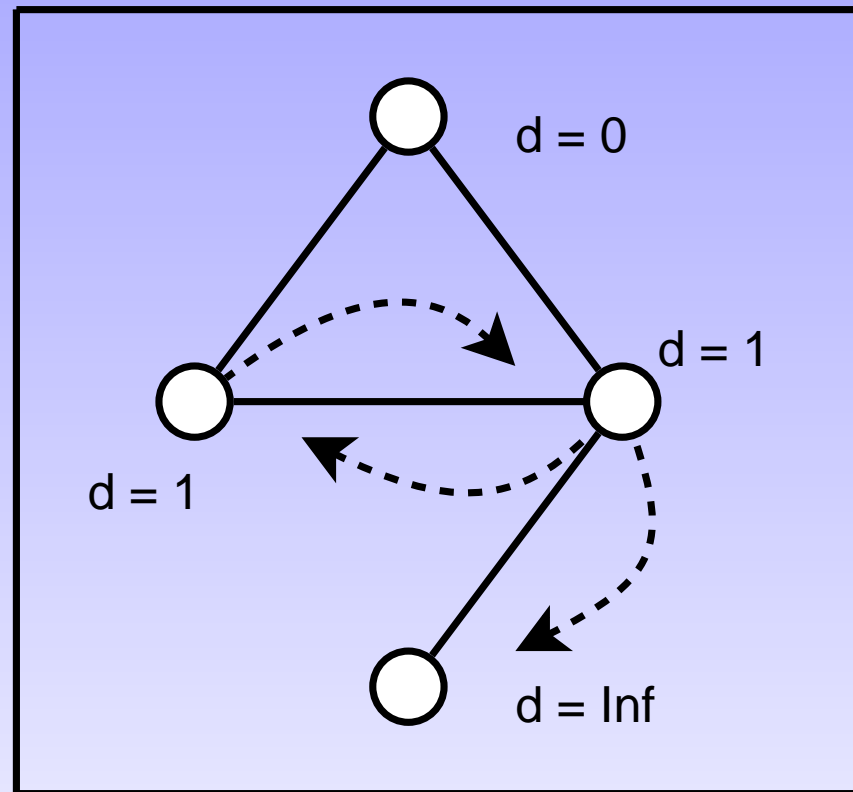
Esimerkki (3)

Esimerkki algoritmin oikeasta suorituksesta:



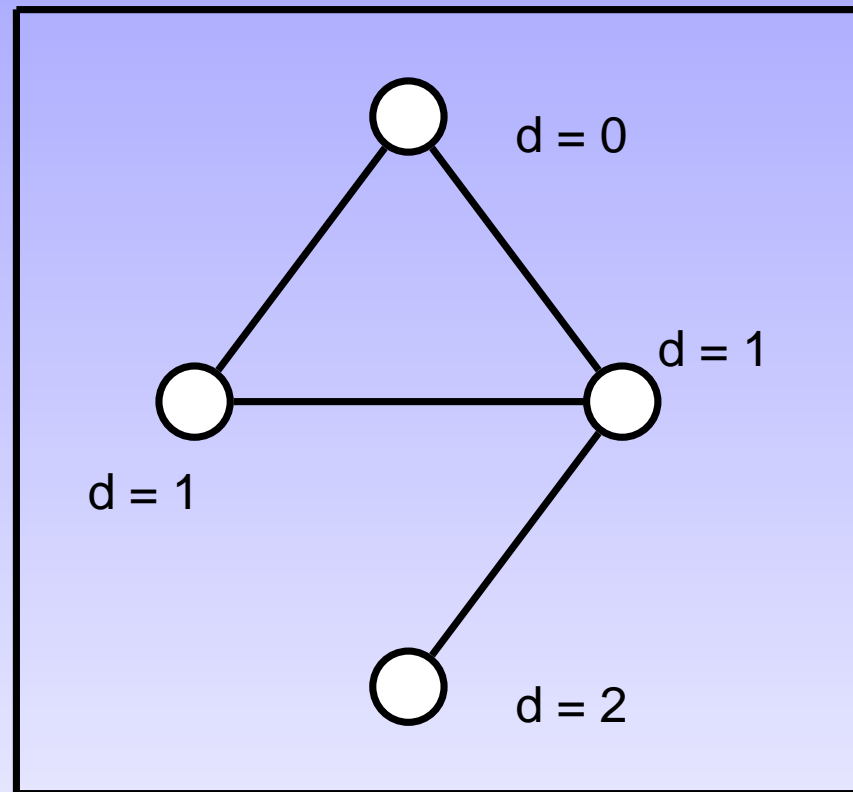
Esimerkki (3)

Esimerkki algoritmin oikeasta suorituksesta:



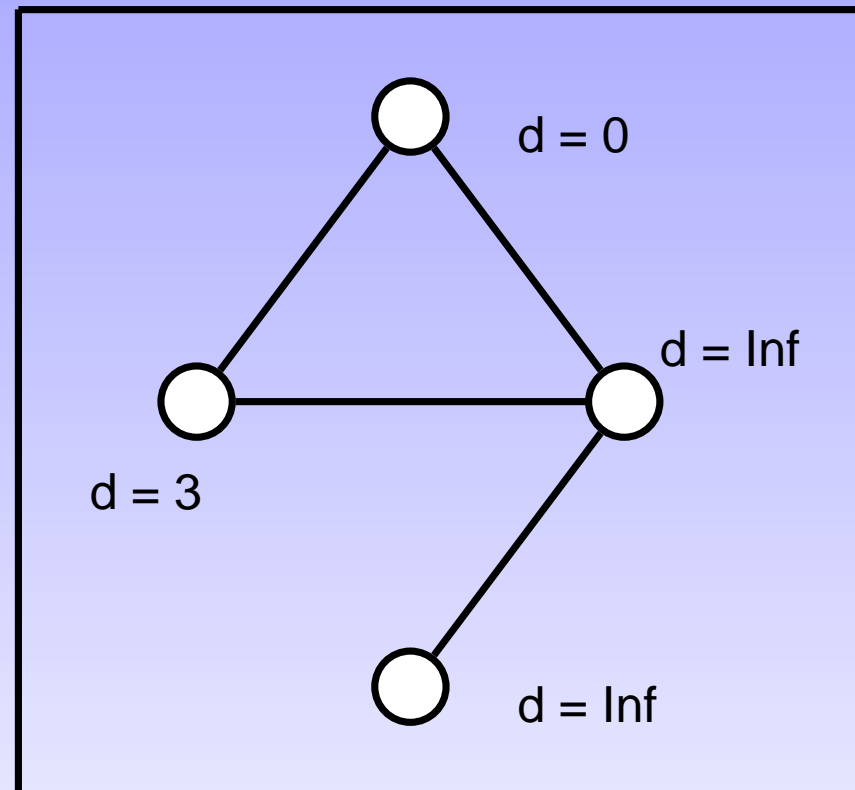
Esimerkki (3)

Esimerkki algoritmin oikeasta suorituksesta:



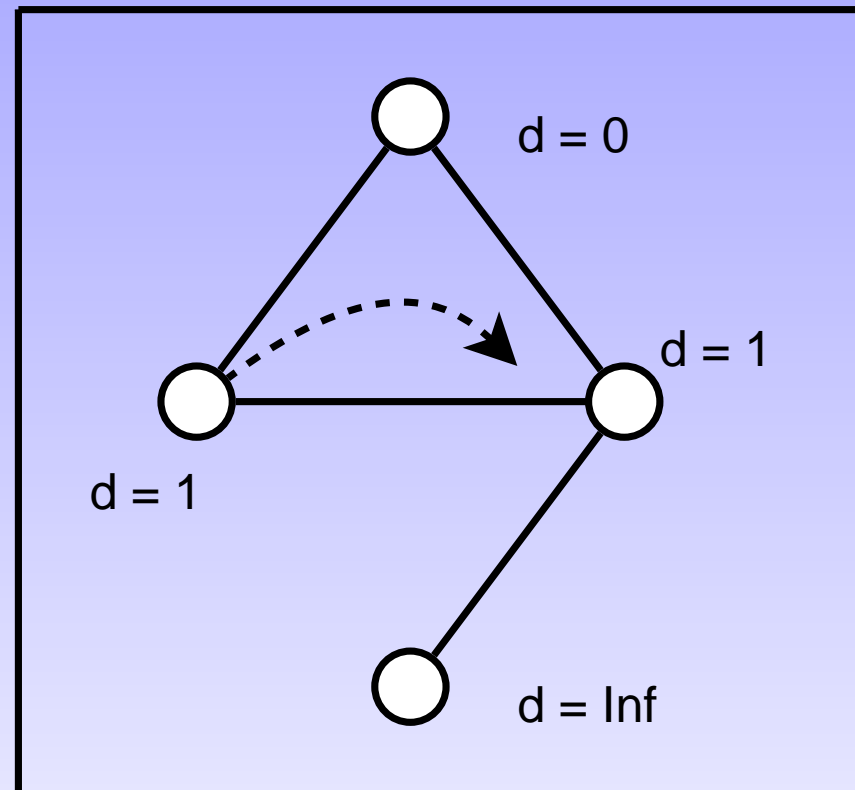
Esimerkki (4)

Esimerkkejä virhetilanteista:



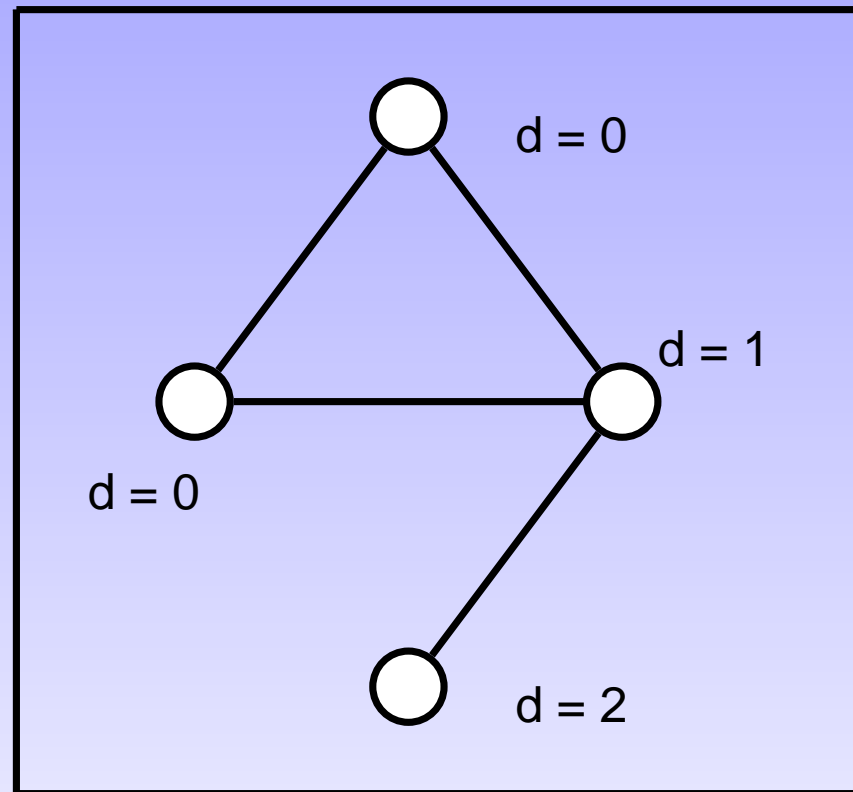
Esimerkki (5)

Esimerkkejä virhetilanteista:



Esimerkki (6)

Esimerkkejä virhetilanteista:



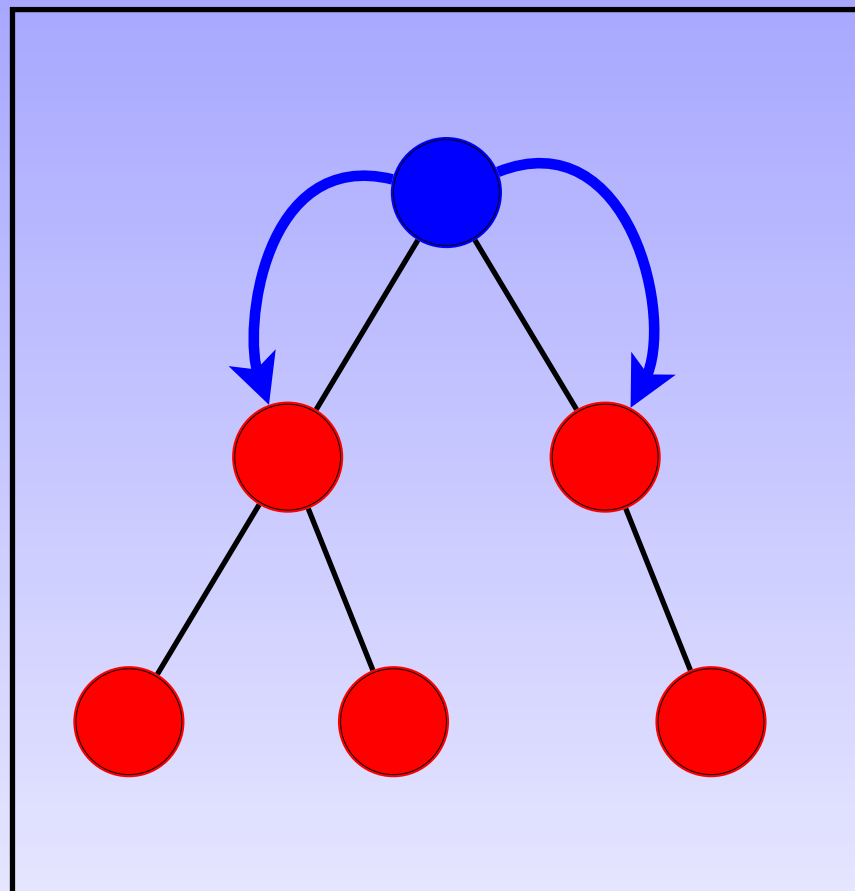
Alustusalgoritmi (1)

- ✓ Pohjana puun toistuva väritys-algoritmi (käsitelty β -synkronoinnin yhteydessä).
- ✓ Muodostetaan virittävä puu toipumiskykyisen algoritmin avulla.
- ✓ Jokaisella prosessorilla on tieto omasta väristään. Väri on kokonaisluku väliltä $[0, 5n - 4]$, $n =$ prosessorien määrä.
- ✓ Jokaiseen kommunikointikanavaan (lapsen P_i ja vanhemman P_j välillä) liittyvät muuttujat:
 - ★ $\text{color}_{j,i}$: Vanhempi kertoo lapselleen värinsä
 - ★ $\text{color}_{i,j}$: Lapsi kertoo vanhemmalleen alipuunsa värin

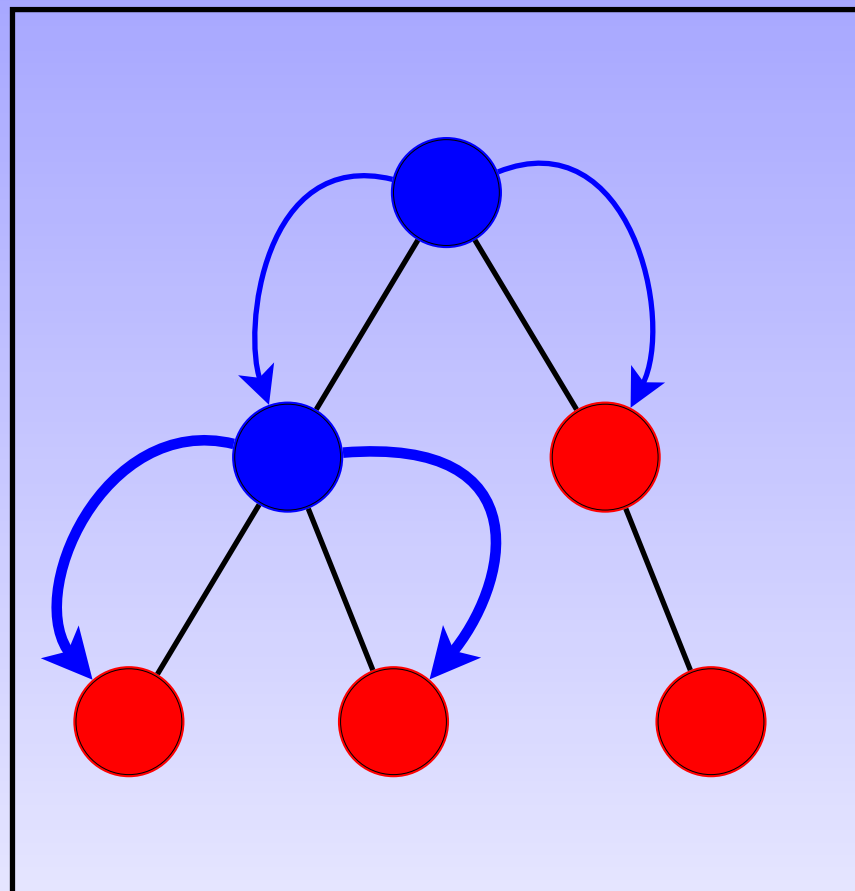
Alustusalgoritmi (2)

- ✓ Juuren P_j algoritmi:
 - ★ Jos kaikki lapset kertovat, että niiden alipuut on väritetty juuren värillä, vaihda väri.
 - Uusi väri = vanha väri + 1 mod $(5n - 3)$
 - ★ Kerro oma väri lapsille P_i (kirjoita $\text{color}_{j,i}$).
- ✓ Muiden prosessorien P_i algoritmi:
 - ★ Jos vanhempi on eri värinen kuin itse, vaihda oma väri vanhemman väriksi.
 - ★ Jos kaikki lapset kertovat, että niiden alipuut on väritetty nykyisellä värillä, kerro se eteenpäin vanhemmalle P_j (kirjoita $\text{color}_{i,j}$)
 - ★ Kerro oma väri lapsille.

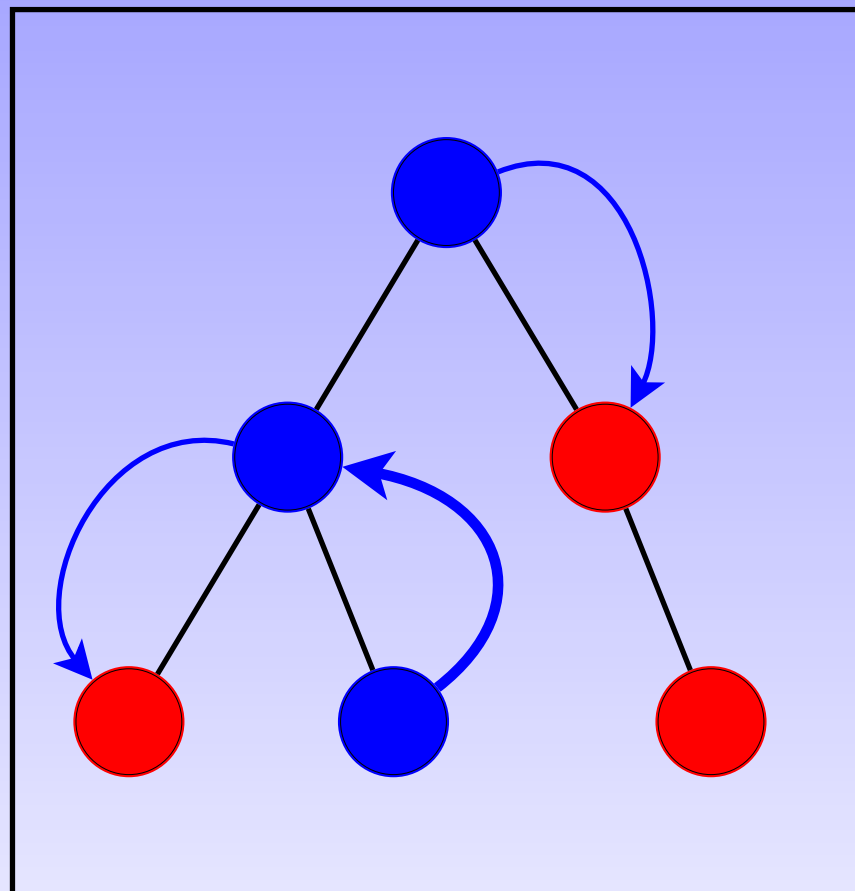
Esimerkki



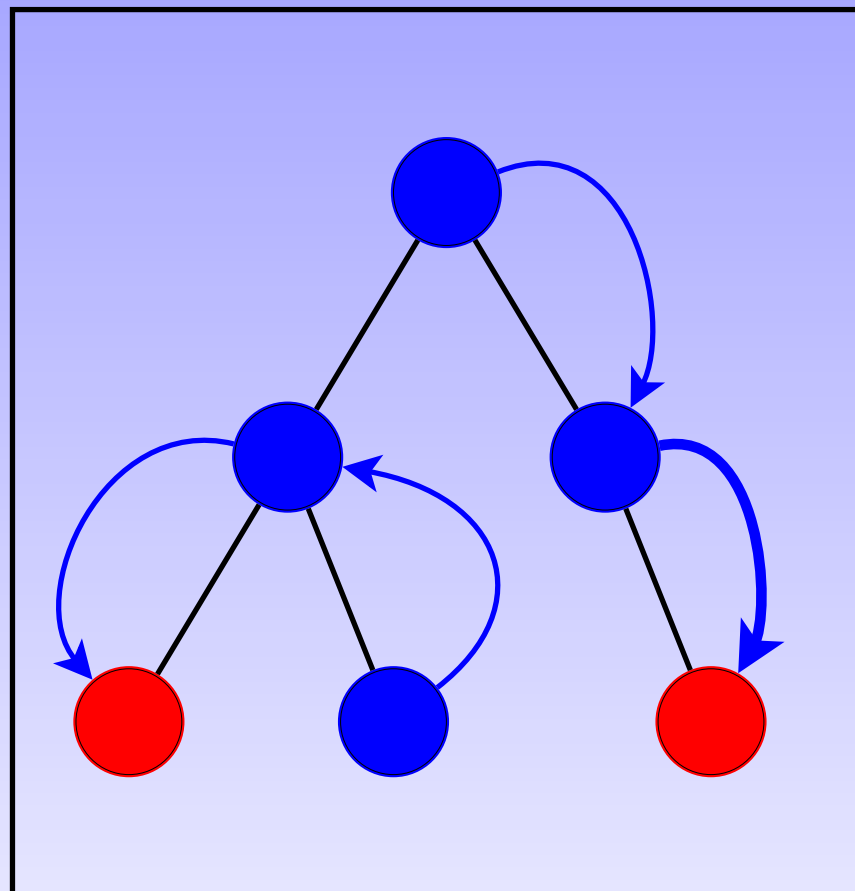
Esimerkki



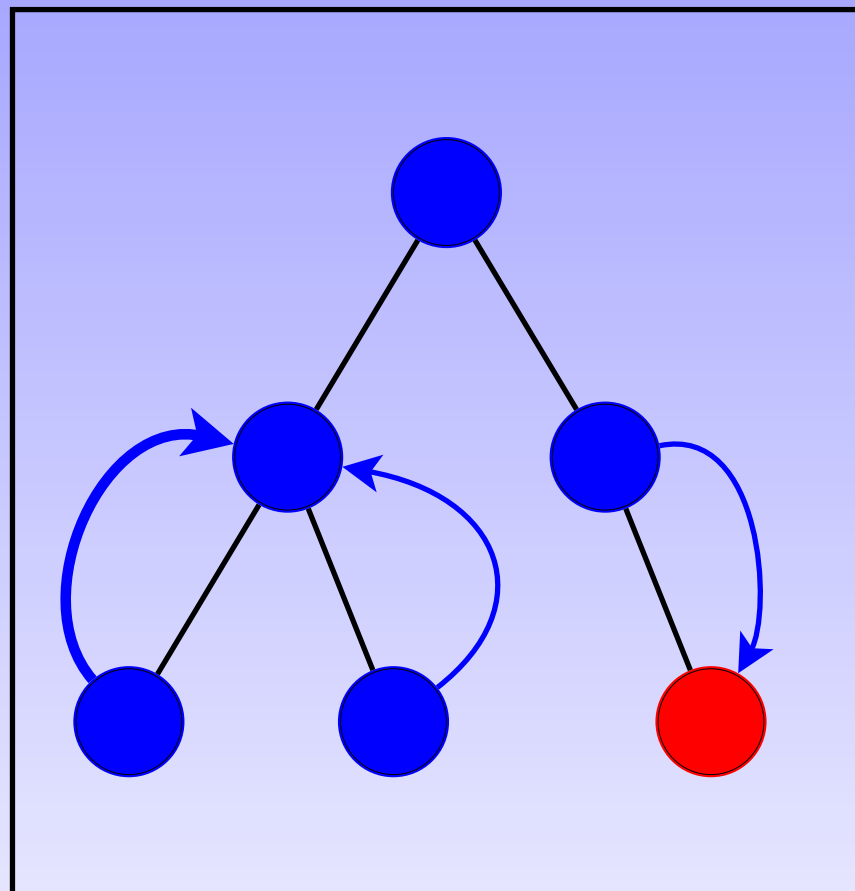
Esimerkki



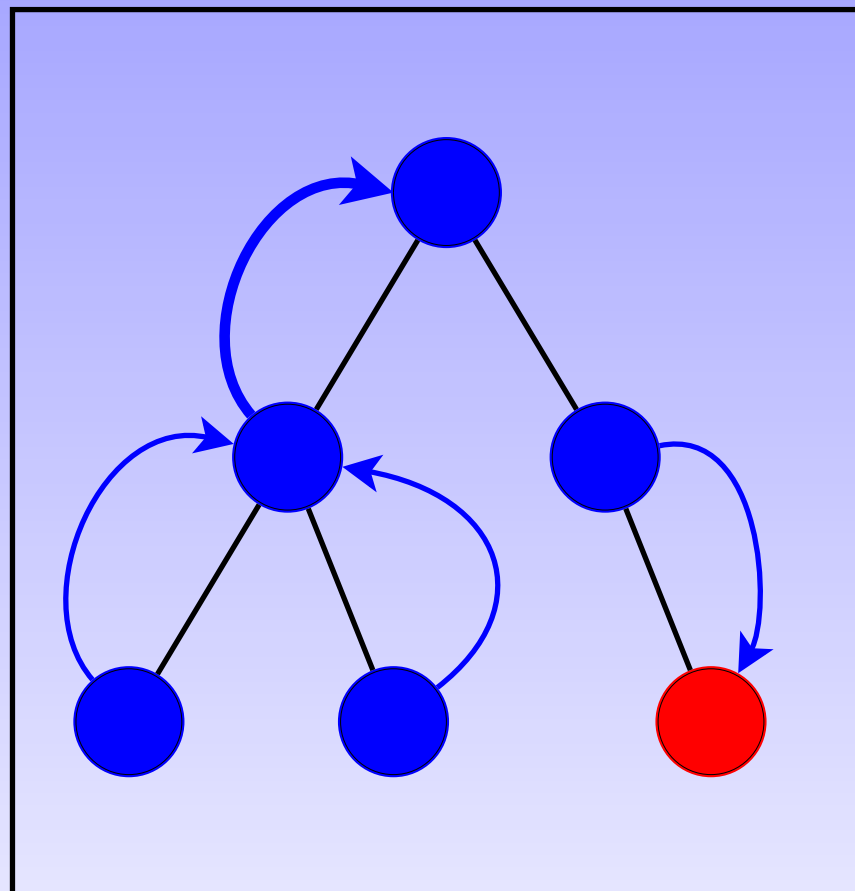
Esimerkki



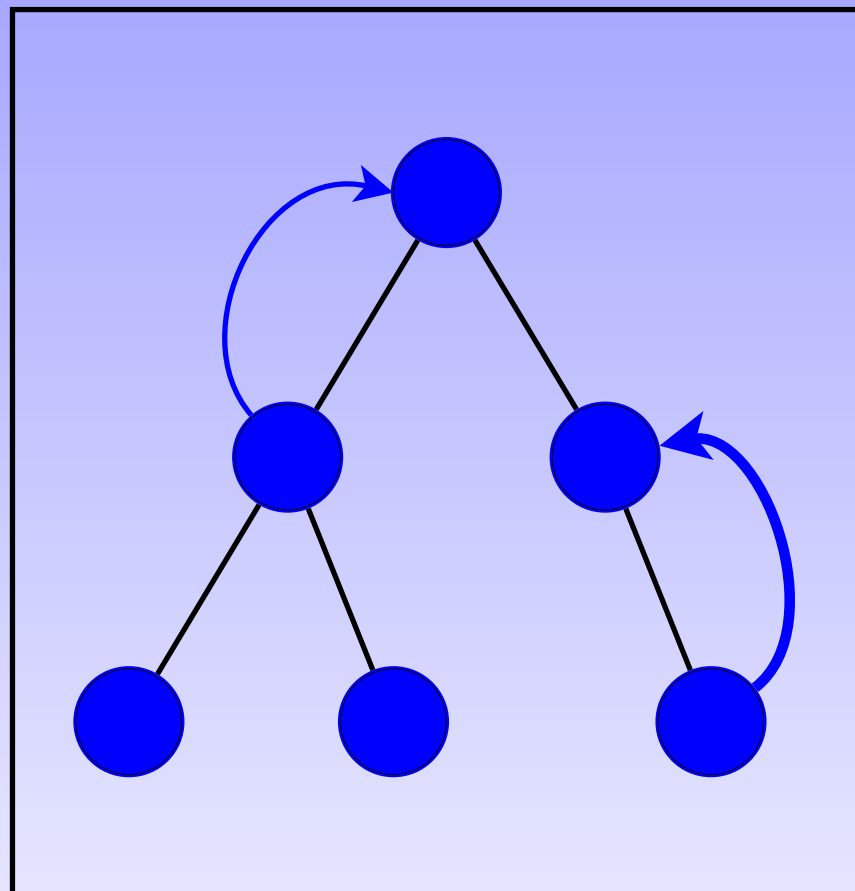
Esimerkki



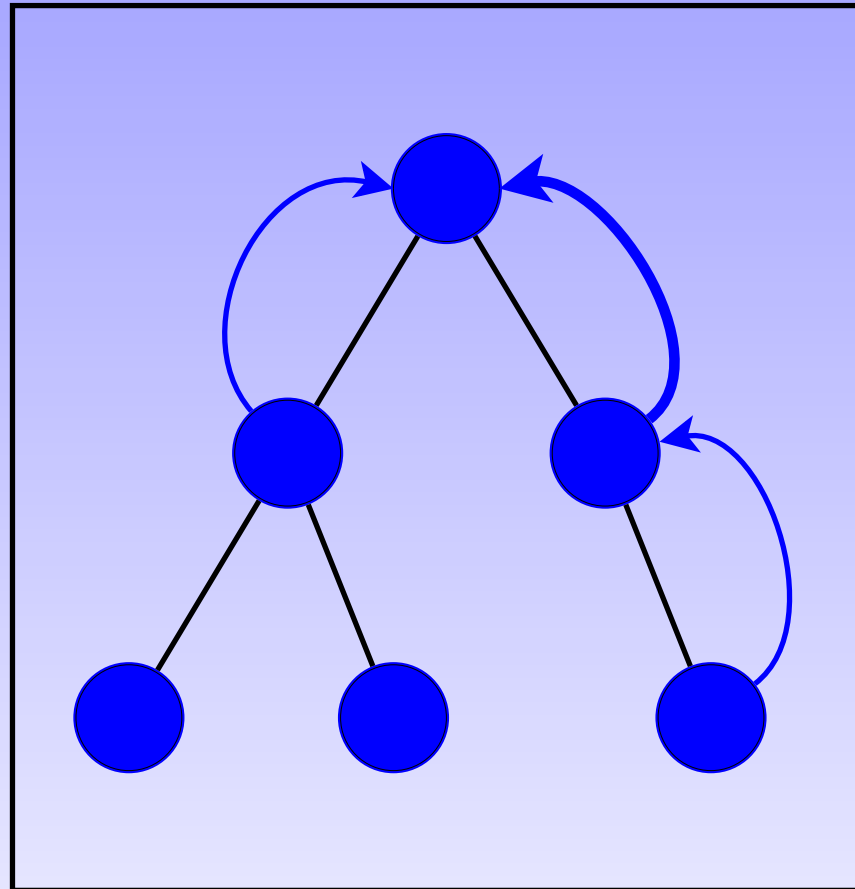
Esimerkki



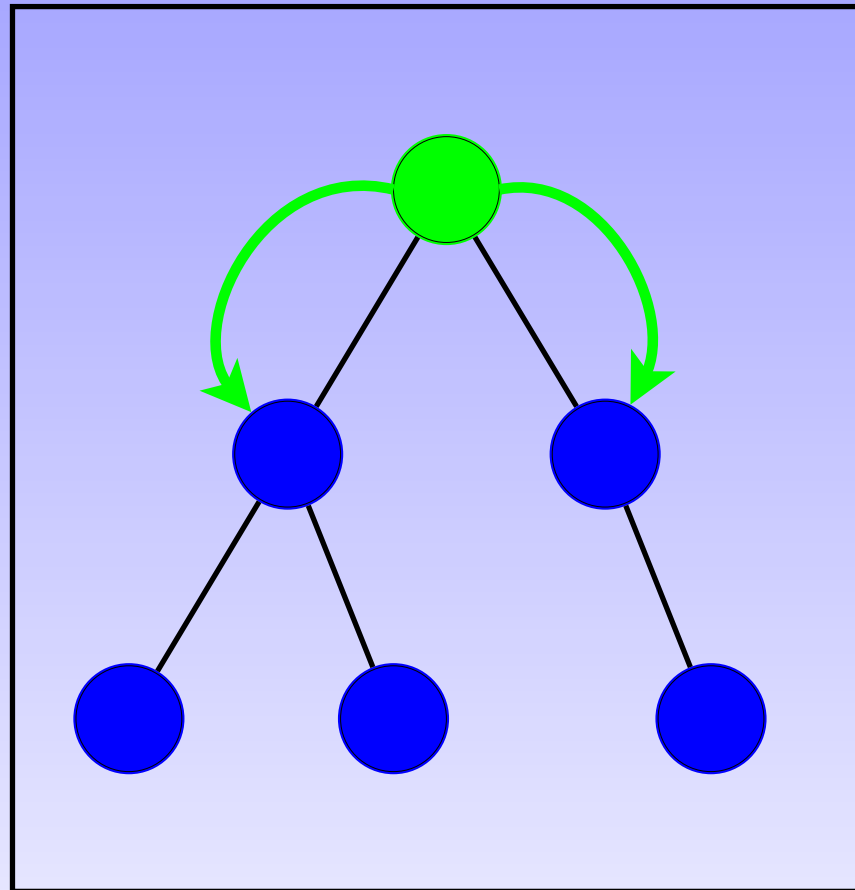
Esimerkki



Esimerkki



Esimerkki



Alustusalgoritmi (3)

- ✓ Tarkastellaan yleisempää tapausta: Mikä tahansa prosessori voi pyytää alustusta, jos se huomaa virheen. (Siis ei vain edellä kuvatun virheentarkastusalgoritmin johtaja.)
- ✓ Jos alustus ei ole käynnissä, jokainen prosessori suorittaa virheentarkastusalgoritmin ennen kuin se raportoi alipuunsa värin vanhemmalleen.
- ✓ Puussa ylöspäin kulkevaan tietoon liitetään tieto siitä, onko alustuspyyntö voimassa vai ei.
- ✓ Puussa alaspäin kulkevaan tietoon liitetään tieto siitä, onko alustus käynnissä vai ei.

Alustusalgoritmi (4)

Alustus tapahtuu siis näin:

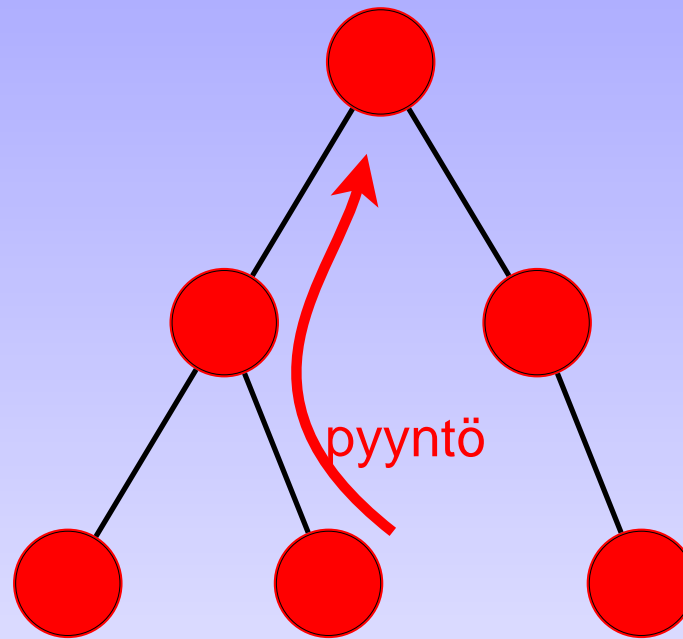
- ✓ Jokin solmu huomaa virheen ja tekee alustuspyynnön.
- ✓ Pyyntö kulkee puussa ylöspäin samalla kun solmu kertoo vanhemmalleen että sen alipuu on väritetty.
- ✓ Juuri saa alustuspyynnön ja käynnistää alustuksen.
- ✓ Tieto alustuksesta kulkee alaspäin puussa samalla kun solmu kertoo oman värinsä lapsilleen.

Alustusalgoritmi (5)

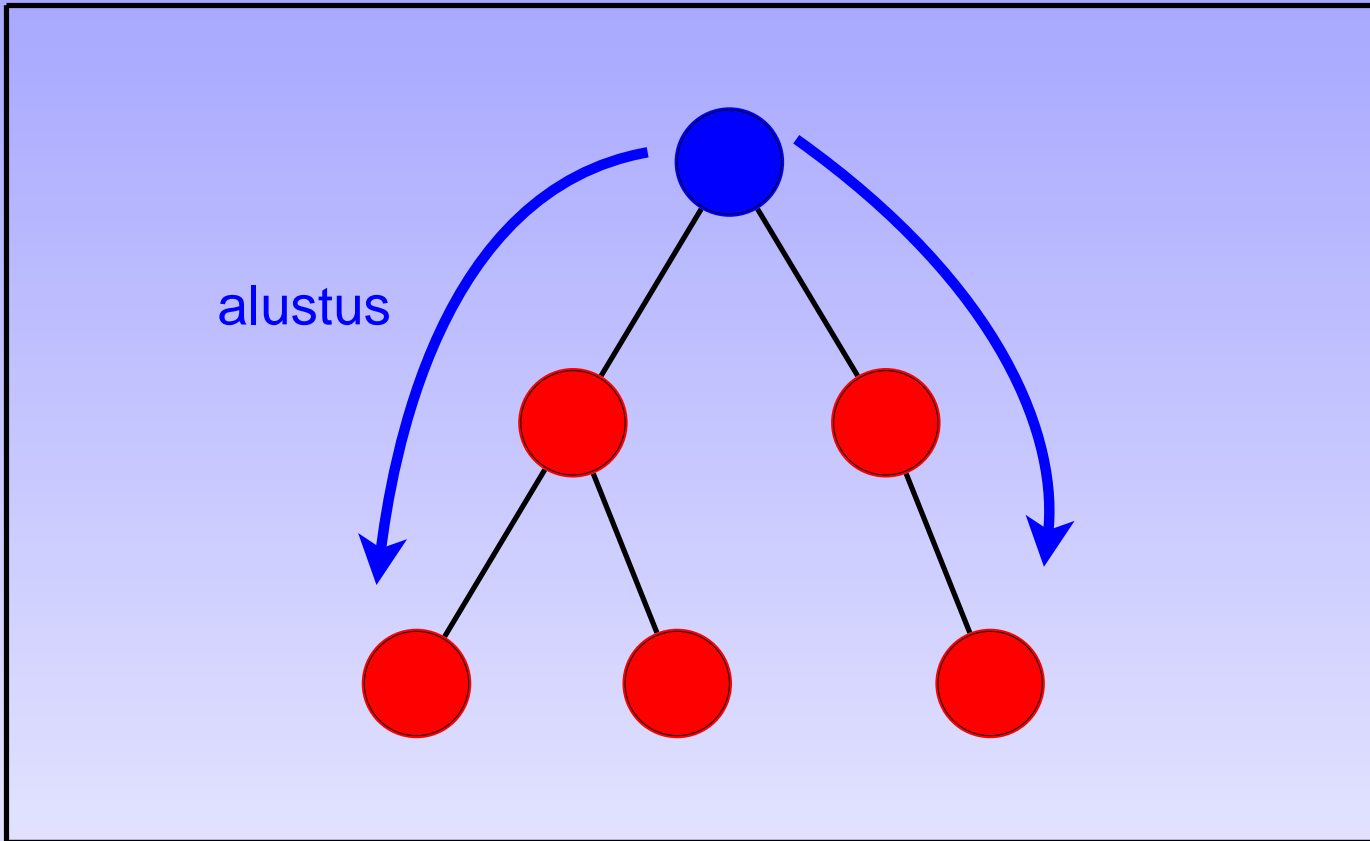
Alustus tapahtuu siis näin (jatkuu):

- ✓ Kun alustus on käynnissä, mikään prosessori ei suorita virheentarkastusalgoritmia eikä pyydä uutta alustusta.
- ✓ Kun tietoa seuraavan kerran välitetään puussa ylöspäin, mukana ei kulje alustuspyyntöjä.
- ✓ Kun juuri vaihtaa väriään, se todetaa alustuksen loppuneeksi.
- ✓ Tieto alustuksen loppumisesta kulkee puussa alaspäin.

Esimerkki

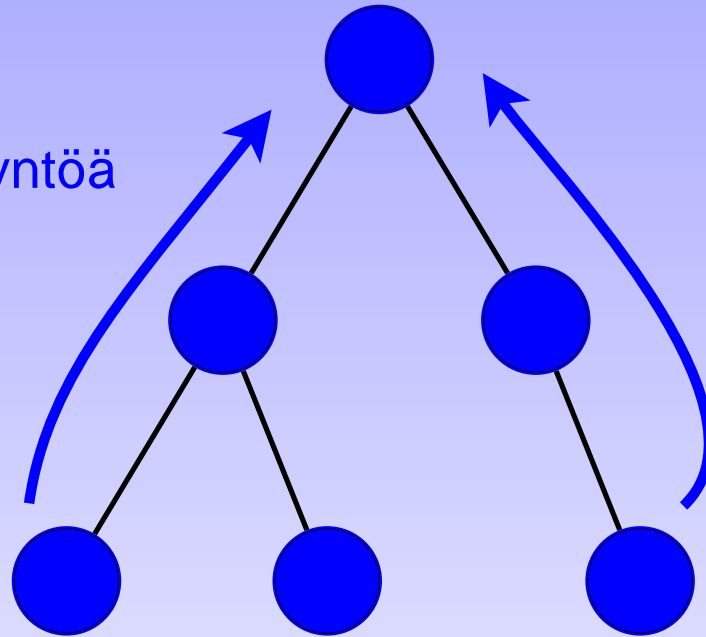


Esimerkki



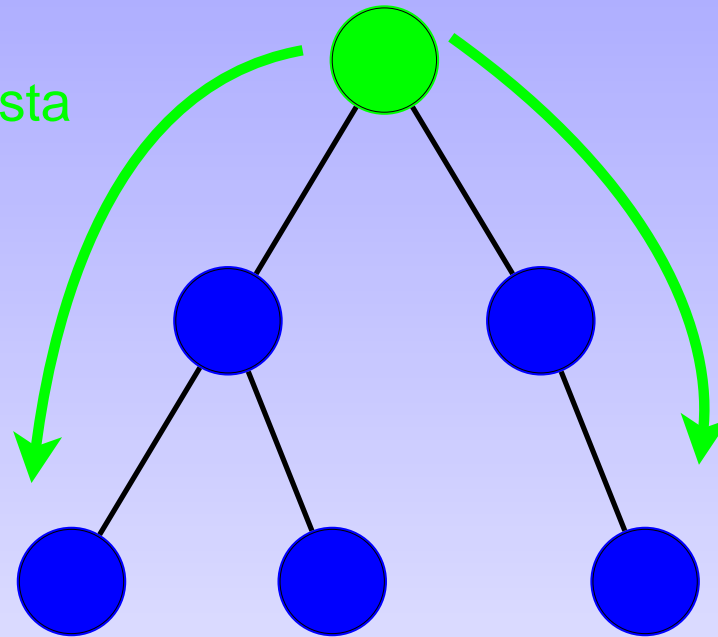
Esimerkki

ei pyyntöä



Esimerkki

ei alustusta



Toipumiskykyisyys (1)

- ✓ Reilu koostaminen: kun osatehtäviä suorittavat algoritmit (esim. virittävän puun muodostus, toistuva väritys) ovat toipuneet virheistä, voi itse stabilointialgoritmin toipuminen alkaa.
- ✓ Aluksi virheentarkkailualgoritmi voi olla mielivaltaisessa tilassa (esim. ilmoittaa olemattomista virheistä).
- ✓ Kun virheidentarkkailualgoritmi on toipunut, voi alustusalgoritmi alkaa toipua.

Toipumiskykyisyys (2)

- ✓ Kun alustusalgoritmi on toipunut, varsinainen stabiloitava algoritmi voi olla mielivaltaisessa tilassa.
- ✓ Virheet kuitenkin havaitaan ja algoritmi palautetaan tarvittaessa sallittuun alkutilaan.

Yhteenveto (1)

- ✓ Algoritmi voidaan muuntaa toipumiskykyiseksi stabiloinnin avulla.
- ✓ Resynkronisessa stabiloinnissa prosessorit tallentavat kaikki algoritmin suorituksen aikana esiintyvät tilansa ja laskevat niitä uudelleen.
- ✓ Yleinen stabilointi suoritetaan tarkkailu- ja alustusalgoritmien avulla.
- ✓ Tarkkailualgoritmi havaitsee, jos algoritmi on virheellisessä tilassa.
 - ★ Se, miten virheet havaitaan, riippuu stabiloitavasta algoritmista.
- ✓ Alustusalgoritmi palauttaa järjestelmän sallittuun alkutilaan.

Yhteenveto (2)

- ✓ On varmistettava, että alustus suoritetaan loppuun ennen uusien virheiden havaitsemista ja uusien alustusten pyytämistä.

Kysyttävää, kommentteja?