

## 4.3 Erillisten joukkojen yhdisteet

Ongelmana on pitää yllä kokoelmaa  $S_1, \dots, S_k$  perusjoukon  $X$  osajoukkoja, jotka voivat muuttua ajan myötä. Rajoituksena on, että mikään alkio  $x$  ei saa kuulua useampaan kuin yhteen joukkoon.

Tässä **Union-Find-ongelmassa** sallittuja operaatioita ovat siis seuraavat:

**Make-Set( $x$ )** : luo yhden alkion joukon  $\{x\}$ , kun  $x \in X$ . Operaatio saadaan tehdä vain kerran kullekin  $x$ .

**Find( $x$ )** : palauta **edustaja** siitä joukosta  $S$  johon  $x$  kuuluu. Tämä edellyttää, että joskus aiemmin on suoritettu Make-Set( $x$ ). Edustaja on mikä tahansa kiinteä joukon  $S$  alkio. Ainoa vaatimus on, että jos  $x \in S$  ja  $y \in S$ , niin Find( $x$ ) ja Find( $y$ ) palauttavat saman alkion.

**Union( $x, y$ )** : Yhdistä alkion  $x$  sisältävä ja alkion  $y$  sisältävä joukko keskenään. Edellyttää, että kummallekin alkion on joskus tehty Make-Set. Uuden joukon edustaja saa olla mielivaltainen sen alkio, joskin tyypilliset toteutukset valitsevat edustajaksi joko alkion Find( $x$ ) tai alkion Find( $y$ ).

**Esimerkki** perusjoukko  $\{a, \dots, k\}$ ; joukkojen (eräät mahdolliset) edustajat lihavoitu

Make-Set( $a$ )

...

Make-Set( $k$ )

**$\{a\} \{b\} \{c\} \{d\} \{e\} \{f\} \{g\} \{h\} \{i\} \{j\} \{k\}$**

Union( $a, b$ )

Union( $c, d$ )

Union( $g, h$ )

**$\{a, b\} \{c, d\} \{e\} \{f\} \{g, h\} \{i\} \{j\} \{k\}$**

Find( $a$ ) palauttaa  $a$

Find( $b$ ) palauttaa  $a$

Find( $e$ ) palauttaa  $e$

Union( $b, d$ )

Union( $h, i$ )

**$\{a, b, c, d\} \{e\} \{f\} \{g, h, i\} \{j\} \{k\}$**

**Sovellusesimerkki:** Kruskalin algoritmi

Joukko muodostuu solmuista, jotka ovat samassa puussa.

Kaari  $(u, v)$  aiheuttaa syklin jos ja vain jos  $\text{Find}(u) = \text{Find}(v)$ .

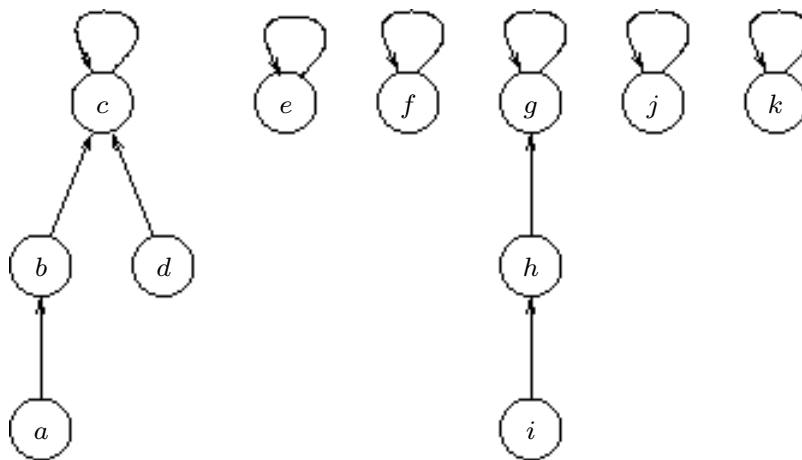
Kaaren  $(u, v)$  lisääminen otetaan huomioon suorittamalla  $\text{Union}(u, v)$ .

Jatkossa  $n$  on perusjoukon alkioden lukumäärä.

*Ratkaisuyritys 1:* joukot linkitettyjä listoja.  
Union vakioajassa, mutta Find voi viedä  $\Omega(n)$

*Ratkaisuyritys 2:* taulukoidaan kullekin  $x$  sen sisältävän joukon edustaja.  
Find vakioajassa, mutta Union voi viedä  $\Omega(n)$

**Ratkaisu:** linkitetty metsä. Kukin joukko muodostaa puun, puun juuri joukon edustaja.



$\{a, b, c, d\} \{e\} \{f\} \{g, h, i\} \{j\} \{k\}$

## Perustoteutus linkitettynä metsänä:

Make-Set( $x$ ):

$p[x] := x$

Union( $x, y$ ):

Link(Find( $x$ ), Find( $y$ ))

Link( $x, y$ ):

$p[x] := y$

Find( $x$ ):

**while**  $p[x] \neq x$  **do**  $x := p[x]$   
**return**  $x$

Toteutus linkitettynä metsänä ei vielä takaa tehokkuutta.

Tehostamme operaatioita seuraavasti:

- Pidetään puut matalina käyttämällä [luokkaan](#) (rank) perustuvaa tasapainotusta.
- Vältetään saman työn toistamista suorittamalla Find-operaation yhteydessä [poluntiiivistys](#).

Puun luokan määräytyminen (perusidea):

Yksisolmuisen puun juuren luokka on 0.

Jos solmu  $y$  linkitetään solmun  $x$  lapseksi, niin solmun  $x$  luokka muuttuu seuraavasti:

- Jos  $\text{rank}(x) \neq \text{rank}(y)$  niin  
 $\text{rank}(x) := \max \{ \text{rank}(x), \text{rank}(y) \}$ .
- Jos  $\text{rank}(x) = \text{rank}(y)$  niin  $\text{rank}(x) := \text{rank}(x) + 1$ .

Siis pienin puu, jonka juuren luokka on  $k$ , on binomipuu  $B_k$ .

Jatkossa esitettävä poluntiivistys sotkee hieman tätä perusajatusta. Joka tapauksessa Make-Set ja Link voidaan esittää muodossa

Make-Set( $x$ ):

$p[x] := x$   
 $\text{rank}[x] := 0$

Link( $x, y$ ):

**if**  $\text{rank}[x] > \text{rank}[y]$   
**then**  $p[y] := x$   
**else**  
     $p[x] := y$   
    **if**  $\text{rank}[x] = \text{rank}[y]$   
    **then**  $\text{rank}[y] := \text{rank}[y] + 1$

Poluntiivistyksessä samalla kun operaatio  $\text{Find}(x)$  etsii polun solmusta  $x$  puun juureen, se oikaisee kaikkien polun varrelta löytyvien solmujen vanhempilinkit osoittamaan suoraan puun juureen.

$\text{Find}(x)$ :

$r := x$

**while**  $p[r] \neq r$  **do**  $r := p[r]$

$q := r$

$r := x$

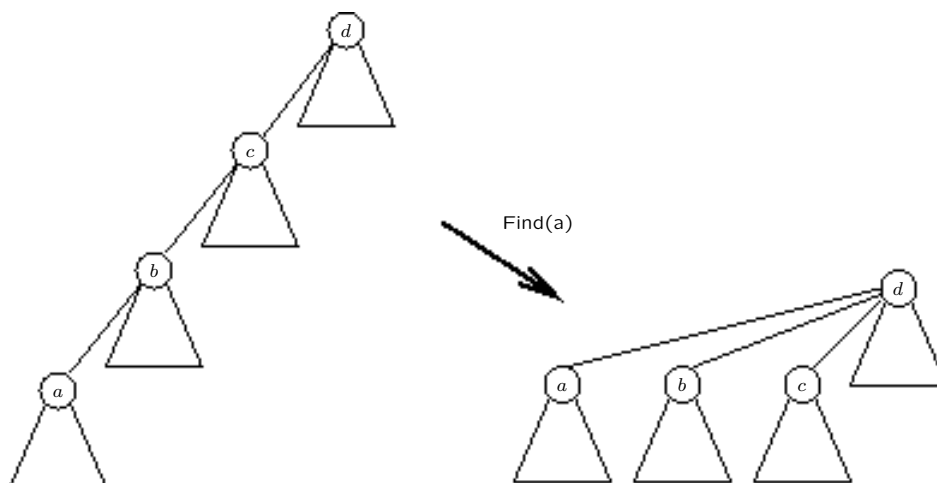
$s := p[x]$

**while**  $s \neq r$  **do**

$p[r] := q$

$r := s$

$s := p[r]$



Huomataan ensin, että yleisyyttä rajoittamatta voidaan olettaa, että Union-operaatioiden sijaan käytetään vain Link-operaatioita, jotka saavat argumenttina osoittimen puun juureen. Tämä seuraa yksinkertaisesti siitä, että operaatio  $\text{Union}(x, y)$  antaa saman tuloksen kuin operaatiot

$$\begin{aligned}x' &:= \text{Find}(x) \\ y' &:= \text{Find}(y) \\ \text{Link}(x', y')\end{aligned}$$

Siis mikä tahansa  $m$  Union-Find-operaation jono voidaan muuntaa korkeintaan  $3m$  operaation Link-Find-jonoksi.

Tavoitteena on osoittaa, että  $m$  operaation jono käyttäen luokkaan perustuvaa tasapainotusta ja poluntiivistystä vie korkeintaan ajan  $O(m\alpha(n))$ , missä  $\alpha$  on *erittäin* hitaasti kasvava ("käytännössä vakio", esim.  $\alpha(n) \leq 4$  kun  $n \leq 10^{500}$ ).

Edellä esitetyn perusteella voidaan olettaa, että Union-operaation kohdistuvat puiden juuriin. Samoin selvästi riittää tarkastella tapausta, että kaikki Make-Set-operaatiot tehdään ennen mitään muita operaatioita.

Sama asymptoottinen aikavaativuus voidaan saavuttaa myös muilla samantyyppisillä tasapainotus- ja polunlyhennystekniikoilla.

Hitaasti kasvava funktio  $\alpha$  saadaan nopeasti kasvavan funktion  $A$  käänteisfunktiona.

Käytetään merkintää  $A^k$  funktion  $A$  iteroinnille  $k$  kertaa:  $A^0(n) = n$  ja  $A^{k+1}(n) = A(A^k(n))$ .

Määritellään rekursiivisesti

$$A_k(j) = \begin{cases} j + 1 & \text{jos } k = 0 \\ A_{k-1}^{j+1}(j) & \text{jos } k \geq 1. \end{cases}$$

Indeksiä  $k$  nimitetään funktion  $A_k$  tasoksi. Selvästi  $A_k(j)$  kasvaa aidosti sekä tason  $k$  että argumentin  $j$  suhteen.

Pari ensimmäistä tasoa saadaan suljettuun muotoon helpolla induktiolla:

$$\begin{aligned} A_1(j) &= 2j + 1 \\ A_2(j) &= 2^{j+1}(j + 1) - 1 \end{aligned}$$

Lasketaan vielä parin tason ensimmäinen arvo:

$$A_3(1) = A_2^2(1) = A_2(7) = 2047$$

ja

$$\begin{aligned} A_4(1) &= A_3^2(1) = A_3(2047) \\ &\gg A_2(2047) = 2^{2048} \cdot 2048 - 1 \approx 10^{620}. \end{aligned}$$

Siis kun määritellään

$$\alpha(n) = \min \{ k \mid A_k(1) \geq n \}$$

saadaan  $\alpha(n) \leq 4$  kun  $n \leq 10^{620}$ . Tätä rajaa voidaan verrata esim. arvioituun universumin atomien lukumäärään  $10^{80}$ .

Tasoitettussa analyysissä keskeiseksi tulee sen tarkastelu, miten arvot  $rank[x]$  ja  $rank[p[x]]$  suhtautuvat toisiinsa.

Luokka  $rank[x]$  voi vaihtua vain kun  $x$  saa uuden lapsen Link-operaatioissa.

Sen sijaan  $p[x]$ , ja näin ollen myön lausekkeen  $rank[p[x]]$  arvo, voi muuttua myös suoritettaessa Find jonka etsintäpolku kulkee solmun  $x$  kautta.

### Propositio A

- Kaikilla  $x$  pätee  $rank[x] \leq rank[p[x]]$ , ja yhtäsuuruus pätee vain jos  $x$  on puun juuri.
- Arvo  $rank[x]$  on aluksi nolla, eikä koskaan pienene. Sen jälkeen, kun  $x$  on linkitetty toisen solmun lapseksi, arvo  $rank[x]$  ei myöskään enää kasva.
- Arvo  $rank[p[x]]$  ei koskaan pienene.

**Todistus** Suoraan operaatioiden toteutuksesta.  $\square$

**Propositio B** Kaikilla  $x$  pätee  $rank[x] \leq \lfloor \log n \rfloor$ .

**Todistus** Helppo induktio.  $\square$

Ryhdyimme nyt määrittelemään potentiaalia  $\Phi$ .

Potentiaali hetkellä  $q$  (kun on suoritettu ensimmäiset  $q$  operaatiota) on

$$\Phi_q = \sum_x \phi_q(x)$$

missä summa on kaikkien alkioden yli ja  $\phi_q(x)$  on alkioille  $x$  pian määriteltävä potentiaali hetkellä  $q$ .

Perusidea on, että  $\phi_q(x)$  on pieni, jos  $\text{rank}[p[x]]$  on hyvin paljon pienempi kuin  $\text{rank}[x]$ .

Tämä on seurausta siitä, että poluntiivistysten takia linkki  $x \rightarrow p[x]$  "oikaisee" monen solmun yli.

Jos jollain  $x$  operaatio  $\text{Find}(x)$  vie hyvin pitkän ajan, niin solmusta  $x$  juureen johtavalla polulla oli paljon solmuja.

Eryteisesti polulla siis on ollut solmuja, joiden vanhempi-linkki ei aiemmin "oikaissut" kovinkaan paljon.

Kun tehdään poluntiivistys, nämä solmut saavat uuden vanhemman ja niiden potentiaali putoaa.

Potentiaaliin  $\phi(x)$  vaikuttaa paitsi suoraan  $\text{rank}[x]$ , myös  $\text{rank}[p[x]]$  epäsuorasti funktioiden *level* ja *iter* välityksellä. (Näissä pitäisi kaikissa olla alaindeksi  $q$ , koska ne vaihtuvat ajan myötä, mutta jätetään se selvyden vuoksi merkitsemättä.)

Määritellään funktio *level* seuraavasti:

$$level(x) = \max \{ k \mid rank[p[x]] \geq A_k(rank[x]) \}.$$

Siis kun merkitään  $rank[x] = r$  pätee  $level(x) = 0$  jos

$$r < rank[p[x]] \leq 2r + 1,$$

ja muuten  $level(x) = k$  jos

$$A_k(r) \leq rank[p[x]] < A_{k+1}(r) = A_k^{r+1}(r).$$

Siis  $level(x)$  kertoo sellaisen  $k$ , että funktiota  $A_k$  voidaan pisteestä  $r$  lähtien iteroida ainakin kerran mutta enintään  $r$  kertaa ennen kuin mennään arvon  $rank[p[x]]$  yli.

Ylläoleva kaava osoittaa, että funktioiden  $A_k$  määritelmän nojalla tämä  $k$  on yksikäsitteinen.

Lisäksi on helppo nähdä

$$0 \leq level(x) \leq \alpha(n) - 1.$$

Olkoon edelleen  $rank[x] = r$  ja  $k = level(x)$ .

Määritellään

$$iter(x) = \max \{ i \mid rank[p[x]] \geq A_k^i(r) \}.$$

Siis  $iter$  antaa hienosäätöä sille, missä välillä  $[A_k(r), A_k^{r+1}(r)]$  arvo  $rank[p[x]]$  sijaitsee. Ylläesitetyn perusteella

$$1 \leq iter(x) \leq r.$$

Olemme nyt valmiit esittämään potentiaalin määritelmän. Kun  $rank$ ,  $level$  ja  $iter$  kaikki viittaavat tilanteeseen hetkellä  $q$ , asetetaan

$$\begin{aligned}\phi_q(x) &= \alpha(n)rank[x] && \text{jos } x \text{ juuri tai } rank[x] = 0 \\ \phi_q(x) &= (\alpha(n) - level(x))rank[x] - iter(x) && \text{muuten.}\end{aligned}$$

Edellä esitetyistä rajoista

$$0 \leq level(x) \leq \alpha(n) - 1$$

ja

$$1 \leq iter(x) \leq rank[r]$$

seuraa suoraan ylä- ja alarajat

$$0 \leq \phi_q(x) \leq \alpha(n)rank(x).$$

Tarkastellaan nyt potentiaalin *muutoksia*.

**Lemma 1** Make-Set-operaation tasoitettu aikavaativuus on  $O(1)$ .

**Todistus** Selvästi todellinen aikavaativuus on vakio, ja potentiaali ei muutu.  $\square$

Tarkastellaan nyt potentiaalin muutoksia Union- ja Find-operaatioissa.

**Lemma 2** Oletetaan, että  $x$  ei ole juurisolmu.

Missään Union- tai Find-operaatiossa solmun  $x$  potentiaali ei kasva.

Jos  $rank[x] > 0$  ja *level* tai *iter* muuttuvat, solmun  $x$  potentiaali pienenee ainakin yhdellä.

**Todistus** Muulla kuin juurisolmulla *rank* ei muutu. Jos  $rank[x] = 0$ , mikään potentiaalin komponentti ei muutu. Tarkastellaan siis tapausta  $rank[x] \geq 1$ .

Koska  $rank[x]$  ei muutu, potentiaali muuttuu vain arvon  $rank[p[x]]$  muutoksen takia.

Olkoon  $r = rank[x]$ ,  $k = level(x)$  ja  $i = iter(x)$  jollain hetkellä.

Siis aluksi

$$A_k^i(r) \leq rank[p[x]] < A_k^{i+1}(r).$$

Kun arvoa  $rank[p[x]]$  ruvetaan kasvattamaan, se voi ylittää rajat  $A_k^{i+1}(r)$ ,  $A_k^{i+2}(r)$ ,  $A_k^{i+3}(r)$ , ... ja vastaavasti  $iter(x)$  saada arvot  $i + 1$ ,  $i + 2$ ,  $i + 3$ , ...

Niin kauan kuin  $\text{rank}[p[x]] < A_k^{r+1}(r) = A_{k+1}(r)$ , taso  $\text{level}(x)$  ei muutu, joten jokainen luvun  $\text{iter}(x)$  kasvu pienentää suoraan potentiaalia  $\phi(x)$ .

Jos  $\text{rank}[p[x]]$  lopulta saavuttaa rajan  $A_k^{r+1}(r) = A_{k+1}(r)$ , niin  $\text{iter}$  ei enää kasva rajan  $r$  yli vaan  $x$  siirtyy tasolle  $k + 1$ .

Tason  $\text{level}(x)$  kasvaminen yhdellä pienentää potentiaalia  $\text{rank}[x]$  verran.

Samalla tosin  $\text{iter}(x)$  voi pienentyä, mutta koska muuttujan  $\text{iter}(x)$  arvoalue on  $\{1, \dots, \text{rank}(x)\}$ , tästä aiheutuva potentiaalın kasvu on enimmillään  $\text{rank}[x] - 1$  yksikköä.

Siis tässäkin muutoksessa potentiaali kokonaisuutena pienenee ainakin yhdellä.

Yksi operaatio voi tietysti muuttaa arvoa  $\text{rank}[p[x]]$  mielivaltaisen paljon, mutta muutos voidaan aina palautaa sarjaksi yhden mittaisia askelia ja soveltaa jokaiseen askeleeseen erikseen tätä päättelyä.  $\square$

Olemme nyt valmiit varsinaiseen tasoitettuun analyysiin.

**Lemma 3** Kunkin Link-operaation tasoitettu aikavaativuus on  $O(\alpha(n))$ .

**Todistus** Symmetrian perusteella riittää tarkastella operaatiota  $\text{Link}(x, y)$  kun  $y$  tulee solmun  $x$  vanhemmaksi.

Todellinen aikavaativuus on selvästi vakio. Pitää osoittaa, että potentiaalin mahdollinen kasvu on  $O(\alpha(n))$ .

Lemman 2 nojalla potentiaali voi kasvaa ainoastaan solmuissa  $x$  ja  $y$ .

Solmu  $x$  muuttuu juuresta ei-juureksi ja sen *rank* ei muutu. Määritelmänsä mukaan  $\phi(x)$  siis joko pysyy arvossa 0 (tapaus  $\text{rank}[x] = 0$ ) tai pienenee vähintään määrällä  $\text{iter}(x) \geq 1$ . Joka tapauksessa  $\phi(x)$  ei ainakaan kasva.

Solmu  $y$  on juuri ennen ja jälkeen operaation, ja  $\text{rank}[y]$  joko pysyy ennallaan tai kasvaa yhdellä. Siis  $\phi(y)$  joko pysyy ennallaan tai kasvaa määrällä  $\alpha(n)$ .  $\square$

Jäljellä on enää Find-operaatio, joka onkin hankalin ja selittää potentiaalifunktion valinnan.

**Lemma 4** Kunkin Find-operaation tasoitettu aikavaativuus on  $O(\alpha(n))$ .

**Todistus** Tarkastellaan operaatiota  $\text{Find}(z)$ . Olkoon  $s$  solmun  $z$  etäisyys puunsa juuresta. Siis todellinen aikavaativuus on  $O(s)$ .

Osoitetaan, että potentiaali ei ainakaan kasva, ja jos  $s \geq \alpha(n) + 2$  niin potentiaali pienenee ainakin määrällä  $s - \alpha(n) - 2$ . Väite seuraa, kun oletetaan potentiaalinvakiokerroin sopivasti valituksi.

Lemman 2 nojalla ainakaan muiden solmujen kuin juuren potentiaali ei kasva. Juuren *rank* ei muutu, joten sen potentiaalikaan ei muutu. Kokonaispotentiaali ei siis ainakaan kasva.

Olkoon nyt  $s \geq \alpha(n)$ . Väitämme, että ainakin  $s - \alpha(n) - 2$  solmun potentiaali aidosti pienenee.

Tämä seuraa, kun osoitamme, että solmun  $x$  potentiaali pienenee ainakin, jos seuraavat ehdot ovat voimassa:

1.  $x$  on hakupolulla solmun  $z$  ja juuren välissä (nämä solmut *poislukien*) ja
2. solmun  $x$  ja juuren välissä (taas nämä solmut *poislukien*) on ainakin yksi solmu  $y$  jolla  $\text{level}(y) = \text{level}(x)$ .

Olkoot siis  $x$  ja  $y$  sellaiset hakupolun solmut, että kumpikaan ei ole polun päätepiste ja  $level[x] = level[y] = k$  ennen poluntiivistystä. Merkitään vielä  $i = iter(x)$ .

Tällöin seuraavat arviot pätevät:

$$\begin{aligned} rank[p[x]] &\geq A_k^i(rank[x]) \\ rank[p[y]] &\geq A_k(rank[y]) \\ rank[y] &\geq rank[p[x]] \end{aligned}$$

Koska  $A_k$  on kasvava, saadaan ennen poluntiivistystä

$$\begin{aligned} rank[p[y]] &\geq A_k(rank[y]) \\ &\geq A_k(rank[p[x]]) \\ &\geq A_k(A_k^i(rank[x])) \end{aligned}$$

Poluntiivistyksen jälkeen  $p[x] = p[y]$  ja siis  $rank[p[x]] = rank[p[y]]$ .

Koska poluntiivistyksessä  $rank[x]$  ei muutu ja  $rank[p[y]]$  ei ainakaan pienene, poluntiivistyksen jälkeen saadaan

$$rank[p[x]] \geq A_k^{i+1}(rank[x]).$$

Tiivistyksessä siis solmulla  $x$  joko  $iter$  tai  $level$  kasvaa, ja siis lemmän 2 mukaan potentiaali pienenee.  $\square$

Lemmoista 1, 3 ja 4 seuraa suoraan

**Korollaari** Poluntiivistyksellä ja tasapainotuksella  $m$  *Union-Find*-operaatiota  $n$ -alkioisessa perusjoukossa voidaan suorittaa ajassa  $O(m\alpha(n))$ .  $\square$

## Pienin yhteinen esivanhempi (Least Common Ancestor, LCA)

Puun solmujen  $u$  ja  $v$  pienin yhteinen esivanhempi,  $LCA(u, v)$ , on solmujen  $u$  ja  $v$  esivanhemmista se, joka on kauimpana puun juuresta.

Tarkastellaan *Union-Find*-sovellusesimerkinä LCA-ongelman *offline-versiota*: on annettu joukko  $P$  solmupareja, ja halutaan yhdellä puun läpikäynnillä määrätä  $LCA(u, v)$  kaikille  $\{u, v\} \in P$ .

Ongelma voidaan ratkaista värittämällä aluksi kaikki puun solmut valkoisiksi ja kutsumalla sitten  $LCA(r)$ , missä  $r$  on puun juuri ja LCA seuraava rekursiivinen proseduuri:

### LCA( $u$ ):

1.  $color[u] := \text{Gray}$
2.  $\text{Make-Set}(u)$
3.  $ancestor[\text{Find}(u)] := u$
4. **for** kaikille solmun  $u$  lapsille  $v$  **do**
5.      $LCA(v)$
6.      $\text{Union}(u, v)$
7.      $ancestor[\text{Find}(u)] := u$
8.  $color[u] := \text{Black}$
9. **for** kaikilla  $v$  joilla  $\{u, v\} \in P$  **do**
10.     **print**  $u, v, ancestor[\text{Find}(v)]$

(Harmaa väri on vain analyysin selventämiseksi, harmaat solmut voitaisiin jättää myös valkeiksi.)

Harmaat solmut muodostavat polun juuresta käsiteltävänä olevaan solmuun.

Kussakin joukossa on yksi harmaa solmu ja tämän solmun kokomustat alipuut (joita voi tietysti olla useampiakin, toisin kuin alla olevassa kaaviomaisessa kuvassa).

*ancestor-Find*-kombinaatiolla kukin musta solmu löytää "oman" harmaan solmunsä. Tämä harmaa solmu on selvästi oikea vastaus kyseistä mustaa solmua koskeviin LCA-kyselyihin. (Todistuksen yksityiskohdat sivuutetaan.)

